Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Guidance to avoiding vulnerabilities in programming languages — Guidance to avoiding vulnerabilities in programming languages — Sulnerability descriptions for the programming language Python Clement introductif — Element principal — Partie m: Titre de la partie Clement introductif — Element principal — Partie m: Titre de la partie Redents of this dafi are invited to salamit, with their comments, notification of any relevant patent rights of which they are avare and to provide supportin documentation.	Baseline Edition, TR 24772–4		Deleted: - 3
Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Guidance to avoiding vulnerabilities in programming languages — Vulnerability descriptions for the programming language [vython] Element introductif — Élément principal — Partie n: Titre de la partie The document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this dark are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.			
Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Guidance to avoiding vulnerabilities in programming languages — Vulnerability descriptions for the programming language Python Elément introductif — Elément principal — Portie n: Titre de la partie Elément introductif — Elément principal — Portie n: Titre de la partie Recipients of this draft ate invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.			
Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Guidance to avoiding vulnerabilities in programming languages — Guidance to avoiding vulnerabilities in programming languages — Vulnerability descriptions for the programming language Python Élément introductif — Élément principal — Partie n: Titre de la partie Élément introductif — Élément principal — Partie n: Titre de la partie Recipients of this draft are invited to submit, with hier comments, notification of any relevant patent rights of which they are wave and to provide supporting documentation.	ISO/IEC JTC 1/SC 22 <u>/WG23 N07</u> 4		Deleted: 41N 0000
Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Guidance to avoiding vulnerabilities in programming languages — Vulnerability descriptions for the programming language Python Elément introductif — Élément principal — Partie n: Titre de la partie Elément introductif — Élément principal — Partie n: Titre de la partie Recipients of this draft are invident to das an International Standard. Recipients of this draft are invident to comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.	Date: <u>2017</u> ;10-1		Deleted: 5
Builde In 447-2 Beleted: 2013-08-07 Edition Deleted: 3 IsO/EC IT 1/S 22/WG 23 Secretariat: ANSI Secretariat: ANSI Secretariat: ANSI Understanding languages — Guidance to avoiding Vulnerabilities in programming languages — Vulnerability descriptions for the programming language selection and use Deleted: through language selection and use programming language Python Deleted: through language selection and use Deleted: Ada Elément introductif — Élément principal — Portie n: Titre de la partie Deleted: Ada Deleted: Ada Marning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an international Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation. Deleted: Market Ada			Deleted: 2015-095-1826
Edition	150/1EC 1R 24/72-	-	Deleted: 2013-08-07
Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages <u>— Vulnerability descriptions for the programming language Python Élément introductif — Élément principal — Partie n: Titre de la partie Waring This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an international Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.</u>	Edition	¥	Deleted: 3
Information Technology – Programming languages – Guidance to avoiding vulnerabilities in programming languages – Vulnerability descriptions for the programming language Python Élément introductif – Élément principal – Partie n: Titre de la partie Elément introductif – Élément principal – Partie n: Titre de la partie Reipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.	ISO/IEC JTC 1/SC 22/WG 23		
Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Vulnerability descriptions for the programming language Python Élément introductif — Élément principal — Partie n: Titre de la partie Warning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an international Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.	Secretariat: ANS		
Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages <u>— Vulnerability descriptions for the programming language Python Elément introductif — Élément principal — Partie n: Titre de la partie Marning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.</u>			
Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages, — Vulnerability descriptions for the Deleted: through language selection and use programming language Python Deleted: Ada Élément introductif — Élément principal — Partie n: Titre de la partie Deleted: Ada This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.			
Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages <u>— Vulnerability descriptions for the programming language Python Élément introductif — Élément principal — Partie n: Titre de la partie Warning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an international Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.</u>			
Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Vulnerability descriptions for the programming language Python Élément introductif — Élément principal — Partie n: Titre de la partie Warning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.			
Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages <u>— Vulnerability descriptions for the programming language Python Élément introductif — Élément principal — Partie n: Titre de la partie Warning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.</u>			
witherabilities in programming languages _ Underability descriptions for the programming language Python Deleted: through language selection and use Élément introductif - Élément principal - Partie n: Titre de la partie Deleted: through language selection and use Warning Marning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.	Information Technology — Programming languages — Guidance to avoiding		
wither ability descriptions for the Detect: drogsinggeseccoul added. programming language Python Detect: drogsinggeseccoul added. Élément introductif – Élément principal – Partie n: Titre de la partie Detect: drogsinggeseccoul added. Warning Marning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.	will persphilities in programming languages. Vulnershility descriptions for the		Deleted: through language selection and use
Élément introductif – Élément principal – Partie n: Titre de la partie Warning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.	vullerabilities in programming languages <u>- vullerability descriptions for the</u>		Deleted: till big i fallig dage selection and use
Élément introductif — Élément principal — Partie n: Titre de la partie Warning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.	programming language Python		
Élément introductif — Élément principal — Partie n: Titre de la partie Warning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.			
Warning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.	Élément introductif — Élément principal — Partie n: Titre de la partie		
Warning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.			
Warning This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.			
This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.	Warning		
without notice and may not be referred to as an International Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.	This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change		
Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.	without notice and may not be referred to as an International Standard.		
are aware and to provide supporting documentation.	Recipients of this draft are invited to submit with their comments potification of any relevant patent rights of which they		
	are aware and to provide supporting documentation.		
		_	
Document type: international standard Document subtype: if applicable	Document type: international standard Document subtype: if applicable		
Document stage: (10) development stage	Document stage: (10) development stage		
Document language: E	Document language: E		Deleted: 3
© ISO/IEC 2015/- All rights reserved i	© ISO/IEC 2015- All rights reserved i		

Copyright notice

This ISO document is a working draft or committee draft and is copyright-protected by ISO. While the reproduction of working drafts or committee drafts in any form for use by participants in the ISO standards development process is permitted without prior permission from ISO, neither this document nor any extract from it may be reproduced, stored or transmitted in any form for any other purpose without prior written permission from ISO.

Requests for permission to reproduce this document for the purpose of selling it should be addressed as shown below or to ISO's member body in the country of the requester:

ISO copyright office Case postale 56, CH-1211 Geneva 20 Tel. + 41 22 749 01 11 Fax + 41 22 749 09 47 E-mail copyright@iso.org Web www.iso.org

Reproduction for sales purposes may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.

© ISO/IEC 2015 - All rights reserved

Contents Page	
Forewordv	
Introductionvi	
1. Scope	
2. Normative references	
3. Terms and definitions, symbols and conventions1	
3.1 Terms and definitions1	
3.2 Key Concepts	
5. General guidance for Python	
6. Specific Guidance for Python	
6.1 General	
6.2 Type System [IHN]	
6.3 Bit Representations [STR]	
6.45 Floating-point Arithmetic [PLF]	
6.5 Enumerator Issues [CCB]	
6.6 Numeric Conversion Errors [FLC]	
6.7 String Termination [CJM]	
6.8 Buffer Boundary Violation [HCB]	
6.9 Unchecked Array Indexing [XYZ]	
6.10 Unchecked Array Copying [XYW]	
6.11 Pointer Casting and Pointer Type Changes [HFC]13	
6.12 Pointer Arithmetic [RVG]13	
6.13 Null Pointer Dereference [XYH]13	
6.14 Dangling Reference to Heap [XYK]13	
6.15 Arithmetic Wrap-around Error [FIF]13	
6.16 Using Shift Operations for Multiplication and Division [PIK]14	
6.17 Sign Extension Error [XZI]14	
6.18 Choice of Clear Names [NAI]14	
6.19 Dead Store [WXQ]16	
6.20 Unused Variable [YZS]17	
6.21 Identifier Name Reuse [YOW]17	
6.22 Namespace Issues [BJL]19	
6.23 Initialization of Variables [LAV]	
6.24 Operator Precedence/Order of Evaluation [JCW]22	
6.25 Side-effects and Order of Evaluation [SAM]23	
6.26 Likely Incorrect Expression [KOA]24	
6.27 Dead and Deactivated Code [XYQ]25	
6.28 Switch Statements and Static Analysis [CLL]26	
© ISO/IEC 2015- All rights reserved iii	

Deleted: x Deleted: 461

Deleted: 3

[... [1]]

6.29 Demarcation of Control Flow [E0] 26 6.30 Loop Control Variables [TEX] 27 6.31 Off-by-one Error [XZH] 28 6.32 Structured Programming [EWD] 28 6.33 Passing Parameters and Return Values [CSJ] 29 6.34 Dangling References to Stack Frames [DCM] 30 6.35 Stubprogram Signature Mismatch [OTR] 31 6.36 Recursion [GDL] 31 6.37 Ignored Error Status and Unhandled Exceptions [OYB] 31 6.38 Termination Strategy [REU] 32 6.39 Type-breaking Reinterpretation of Data [AMV] 32 6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [LRM] 33 6.44 rupument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DIS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.51 Drovision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure L	
6.30 Loop Control Variables [TEX]. 27 6.31 Off-by-one Error [XZH] 28 6.32 Structured Programming [EWD]. 28 6.33 Passing Parameters and Return Values [CSJ] 29 6.34 Dangling References to Stack Frames [DCM] 30 6.35 Subprogram Signature Mismatch [OTR]. 31 6.36 Recursion [GDL] 31 6.37 Ignored Error Status and Unhandled Exceptions [OYB]. 31 6.38 Termination Strategy [REU] 32 6.39 Type-breaking Reinterpretation of Data [AMV]. 32 6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [LRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Unter-language Calling [DIS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 36 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherguage Teatures [RES] 37	6.29 Demarcation of Control Flow [EOJ]26
6.31 Off-by-one Error [XZH] 28 6.32 Structured Programming [EWD] 28 6.33 Passing Parameters and Return Values [CSJ] 29 6.34 Dangling References to Stack Frames [DCM] 30 6.35 Subprogram Signature Mismatch [OTR] 31 6.36 Recursion [GDL] 31 6.37 Ignored Error Status and Unhandled Exceptions [OYB] 31 6.38 Termination Strategy [REU] 32 6.39 Type-breaking Reinterpretation of Data [AMV] 32 6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [LRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DIS] 35 6.46 Unamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Ianguage-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37	6.30 Loop Control Variables [TEX]27
6.32 Structured Programming [EWD]. 28 6.33 Passing Parameters and Return Values [CSJ] 29 6.34 Dangling References to Stack Frames [DCM] 30 6.35 Subprogram Signature Mismatch [OTR]. 31 6.36 Recursion [GDL] 31 6.37 Ignored Error Status and Unhandled Exceptions [OYB]. 31 6.38 Termination Strategy [REU] 32 6.39 Type-breaking Reinterpretation of Data [AMV]. 32 6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP]. 33 6.43 Luneritance [RIP]. 33 6.44 Argument Passing to Library Functions [TRJ]. 35 6.45 Inter-language Calling [DJS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives (NMP] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [EMF] 40 6.54 Undefined Behaviour [EME] 40 6.55 Unspeci	6.31 Off-by-one Error [XZH]28
6.33 Passing Parameters and Return Values [CS] 29 6.34 Dangling References to Stack Frames [DCM] 30 6.35 Subprogram Signature Mismatch [OTR] 31 6.36 Recursion [GDL] 31 6.37 Ignored Error Status and Unhandled Exceptions [OYB] 31 6.38 Termination Strategy [REU] 32 6.39 Type-breaking Reinterpretation of Data [AMV] 32 6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [IRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DIS] 35 6.46 Unamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [BUF] 40 6.55 Implementation-defined Behaviour [FAB] 41	6.32 Structured Programming [EWD]28
6.34 Dangling References to Stack Frames [DCM] 30 6.35 Subprogram Signature Mismatch [OTR] 31 6.36 Recursion [GDL] 31 6.37 Ignored Error Status and Unhandled Exceptions [OYB] 31 6.38 Termination Strategy [REU] 32 6.40 Memory Leak [XYL] 32 6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [LRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DJS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.49 Pre-processor Directives [MMP] 36 6.49 Pre-processor Directives [MMP] 36 6.49 Inducting Easting Easti	6.33 Passing Parameters and Return Values [CSJ]29
6.35 Subprogram Signature Mismatch [OTR] 31 6.36 Recursion [GDL] 31 6.37 Ignored Error Status and Unhandled Exceptions [OYB] 31 6.38 Termination Strategy [REU] 32 6.39 Type-breaking Reinterpretation of Data [AMV] 32 6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [LRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.44 Templates and Gode and Self-modifying Code [NYY] 35 6.45 Inter-language Calling [DIS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 40 6.54 Undefined Behaviour [FAB] 41 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 </td <td>6.34 Dangling References to Stack Frames [DCM]</td>	6.34 Dangling References to Stack Frames [DCM]
6.36 Recursion [GDL] 31 6.37 Ignored Error Status and Unhandled Exceptions [OYB] 31 6.38 Termination Strategy [REU] 32 6.39 Type-breaking Reinterpretation of Data [AMV] 32 6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [LRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DJS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [EVF] 40 6.54 Undefined Behaviour [EVF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibl	6.35 Subprogram Signature Mismatch [OTR]
6.37 Ignored Error Status and Unhandled Exceptions [OYB] 31 6.38 Termination Strategy [REU] 32 6.39 Type-breaking Reinterpretation of Data [AMV] 32 6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [LRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DJS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [EVF] 40 6.54 Indefined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.36 Recursion [GDL]
6.38 Termination Strategy [REU] 32 6.39 Type-breaking Reinterpretation of Data [AMV] 32 6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [LRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DIS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [EWF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.37 Ignored Error Status and Unhandled Exceptions [OYB]
6.39 Type-breaking Reinterpretation of Data [AMV] 32 6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [IRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DJS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [BQF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.38 Termination Strategy [REU]32
6.40 Memory Leak [XYL] 32 6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [LRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DIS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [FAB] 41 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.39 Type-breaking Reinterpretation of Data [AMV]
6.41 Templates and Generics [SYM] 33 6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [LRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DJS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [BQF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.40 Memory Leak [XYL]
6.42 Inheritance [RIP] 33 6.43 Extra Intrinsics [LRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DJS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.41 Templates and Generics [SYM]
6.43 Extra Intrinsics [LRM] 33 6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DJS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.42 Inheritance [RIP]33
6.44 Argument Passing to Library Functions [TRJ] 35 6.45 Inter-language Calling [DJS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.43 Extra Intrinsics [LRM]33
6.45 Inter-language Calling [DJS] 35 6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.44 Argument Passing to Library Functions [TRJ]
6.46 Dynamically-linked Code and Self-modifying Code [NYY] 35 6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.45 Inter-language Calling [DJS]35
6.47 Library Signature [NSQ] 36 6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.46 Dynamically-linked Code and Self-modifying Code [NYY]
6.48 Unanticipated Exceptions from Library Routines [HJW] 36 6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.47 Library Signature [NSQ]
6.49 Pre-processor Directives [NMP] 37 6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.48 Unanticipated Exceptions from Library Routines [HJW]
6.50 Suppression of Language-defined Run-time Checking [MXB] 37 6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.49 Pre-processor Directives [NMP]
6.51 Provision of Inherently Unsafe Operations [SKL] 37 6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.50 Suppression of Language-defined Run-time Checking [MXB]
6.52 Obscure Language Features [BRS] 37 6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.51 Provision of Inherently Unsafe Operations [SKL]
6.53 Unspecified Behaviour [BQF] 40 6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.52 Obscure Language Features [BRS]
6.54 Undefined Behaviour [EWF] 40 6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48	6.53 Unspecified Behaviour [BQF]
6.55 Implementation-defined Behaviour [FAB] 41 6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48 Index 48 48	6.54 Undefined Behaviour [EWF]40
6.56 Deprecated Language Features [MEM] 42 8 Implications for standardization 43 Bibliography 48 Index 48 48	6.55 Implementation–defined Behaviour [FAB]
8 Implications for standardization	6.56 Deprecated Language Features [MEM]42
Bibliography	8 Implications for standardization43
Index 48	Bibliography48
	Index 48
3	۲

iv

© ISO/IEC 2015 – All rights reserved

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example), it may decide to publish a Technical Report. A Technical Report is entirely informative in nature and shall be subject to review every five years in the same manner as an International Standard.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 24772, was prepared by Joint Technical Committee ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming languages, their environments and system software interfaces.

Deleted: x

Introduction

vi

This Technical Report provides guidance for the programming language Python, so that application developers considering Python or using Python will be better able to avoid the programming constructs that lead to vulnerabilities in software written in the Python language and their attendant consequences. This guidance can also be used by developers to select source code evaluation tools that can discover and eliminate some constructs that could lead to vulnerabilities in their software. This report can also be used in comparison with companion Technical Reports and with the language-independent report, TR 24772–1, to select a programming language that provides the appropriate level of confidence that anticipated problems can be avoided.

This technical report part is intended to be used with TR 24772–1, which discusses programming language vulnerabilities in a language independent fashion.

It should be noted that this Technical Report is inherently incomplete. It is not possible to provide a complete list of programming language vulnerabilities because new weaknesses are discovered continually. Any such report can only describe those that have been found, characterized, and determined to have sufficient probability and consequence.

Deleted: All programming languages contain constructs that are incompletely specified, exhibit undefined behaviour, are implementation-dependent, or are difficult to use correctly. The use of those constructs may therefore give rise to *vulnerabilities*, as a result of which, software programs can execute differently than intended by the writer. In some cases, these vulnerabilities can compromise the safety of a system or be exploited by attackers to compromise the security or privacy of a system. -

Deleted:

Deleted: 3

Deleted: Furthermore, to focus its limited resources, the working group developing this report decided to defer comprehensive treatment of several subject areas until future editions of the report. These subject areas include: - (...[2])
Formatted: Space After: 10 pt, Line spacing: multiple 1.15
Ii, No bullets or numbering

© ISO/IEC 2015 - All rights reserved

Information Technology — Programming Languages — Guidance to avoiding vulnerabilities in programming languages — Vulnerability descriptions for the programming language Python

1. Scope

This Technical Report specifies software programming language vulnerabilities to be avoided in the development of systems where assured behaviour is required for security, safety, mission-critical and business-critical software. In general, this guidance is applicable to the software developed, reviewed, or maintained for any application.

Vulnerabilities are described in this Technical Report document the way that the vulnerability described in the language-independent TR 24772–1 are manifested in Python.

2. Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC TR 24772–1:2017, Information Technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages.

ISO 80000–2:2009, Quantities and units — Part 2: Mathematical signs and symbols to be use in the natural sciences and technology

ISO/IEC 2382–1:1993, Information technology — Vocabulary — Part 1: Fundamental terms IEC 60559:2011, Information technology — Microprocessor Systems — Floating-Point arithmetic

3. Terms and definitions, symbols and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 2382–1, in TR 24772–1, and the following apply. Other terms are defined where they appear in *italic* type.

<u>assignment statement</u>: Used to create (or rebind) a variable to an object. The simple syntax is a=b, the augmented syntax applies an operator at assignment time (for example, a += 1) and therefore cannot create a variable since it operates using the current value referenced by a variable. Other syntaxes support multiple targets (that is, x = y = z = 1).

© ISO/IEC 2015 – All rights reserved

Deleted: Ada

Formatted: Font:16 pt, Bold

Formatted: English (UK)
Formatted: English (UK), Highlight
Formatted: English (UK)
Formatted: Font:Italic, English (UK)
Deleted: TR 24772-1 Programming Languages - etc.
Formatted: English (UK)
Formatted: Space After: 0 pt
Formatted: Font:Italic
Formatted: Font:Italic
Formatted: Font:Not Italic
 Invoted town [1]: Active of a state of the set of the
Retrieved 3 5, 2012, from
http://www.cs.utah.edu/~regehr/papers/overflow12.pdf [3]
Formatted: Font:
Deleted:

Deleted: 3

<u>body</u>: The portion of a compound statement that follows the header. It may contain other compound (nested) statements.

<u>boolean</u>: A truth value where True equivalences to any non-zero value and False equivalences to zero. Commonly expressed numerically as 1 (true), or 0 (false) but referenced as True and False.

 $\underline{built-in}$: A function provided by the Python language intrinsically without the need to import it (called the, str, slice, type).

<u>class</u>: A program defined type which is used to instantiate objects and provide attributes that are common to all the objects that it instantiates.

comment: Comments are preceded by a hash symbol "#".

<u>complex number</u>: A number made up of two parts each expressed as floating-point numbers: a real and an imaginary part. The imaginary part is expressed with a trailing upper or lower case "J or j".

compound statement: A structure that contains and controls one or more statements.

<u>CPython</u>: The standard implementation of Python coded in ANSI portable C.

dictionary: A built-in mapping consisting of zero or more key/value "pairs". Values are stored and retrieved using keys which can be of mixed types (with some caveats beyond the scope of this annex).

docstring: One or more lines in a unit of code that serve to document the code. Docstrings are retrievable at runtime.

<u>exception</u>: An object that encapsulates the attributes of an exception (an error or abnormal event). Raising an exception is a process that creates the exception object and propagates it through a process that is optionally defined in a program. Lacking an exception 'handler'', Python terminates the program with an error message.

floating-point number: A real number expressed with a decimal point, an exponent expressed as an upper or lower case "e or E" or both (for example, 1.0, 27e0, .456).

 $\underline{function}$: A grouping of statements, either built-in or defined in a program using the def statement, which can be called as a unit.

<u>garbage collection</u>: The process by which the memory used by unreferenced object and their namespaces is reclaimed. Python provides a gc module to allow a program to direct when and how garbage collection is done.

<u>global</u>: A variable that is scoped to a module and can be referenced from anywhere within the module including within functions and classes defined in that module.

<u>guerrilla patching</u>: Also known as Monkey Patching, the practice of changing the attributes and/or methods of a module's class at run-time from outside of the module.

immutability: The characteristic of being unchangeable. Strings, tuples, and numbers are immutable objects in Python.

import: A mechanism that is used to make the contents of a module accessible to the importing program.

2

© ISO/IEC 2015 - All rights reserved

<u>inheritance</u>: The ability to define a class that is a subclass of other classes (called the superclass). Inheritance uses a method resolution order (MRO) to resolve references to the correct inheritance level (that is, it resolves attributes (methods and variables)).

<u>instance</u>: A single occurrence of a class that is created by calling the class as if it was a function (for example, a = Animal()).

<u>integer</u>: An integer can be of any length but is more efficiently processed if it can be internally represented by a 32 or 64 bit integer. Integer literals can be expressed in binary, decimal, octal, or hexadecimal formats.

<u>keyword</u>: An identifier that is reserved for special meaning to the Python interpreter (for example, if, else, for, class).

<u>*lambda expression*</u>: A convenient way to express a single return function statement within another statement instead of defining a separate function and referencing it.

list: An ordered sequence of zero or more items which can be modified (that is, is mutable) and indexed.

<u>literals</u>: A string or number (for example, 'abc', 123, 5.4). Note that a string literal can use either double quote (") or single apostrophe pairs (') to delimit a string.

 $\underline{membership}: If an item occurs within a sequence it is said to be a member. Python has built-ins to test for membership (for example, if a in b). Classes can provide methods to override built-in membership tests.$

<u>module</u>: A file containing source language (that is, statements) in Python (or another) language. A module has its own namespace and scope and may contain definitions for functions and classes. A module is only executed when first imported and upon reloading.

<u>mutability</u>: The characteristic of being changeable. Lists and dictionaries are two examples of Python objects that are mutable.

name: A variable that references a Python object such as a number, string, list, dictionary, tuple, set, builtin, module, function, or class.

<u>namespace</u>: A place where names reside with their references to the objects that they represent. Examples of objects that have their own namespaces include: blocks, modules, classes, and functions. Namespaces provide a way to enforce scope and thus prevent name collisions since each unique name exists in only one namespace.

none: A null object.

number: An integer, floating point, decimal, or complex number.

<u>operator</u>: Non-alphabetic characters, characters, and character strings that have special meanings within expressions (for example, +, -, not, is).

overriding: Coding an attribute in a subclass to replace a superclass attribute.

package: A collection of one or more other modules in the form of a directory.

© ISO/IEC 2015 – All rights reserved

Deleted: 3

pickling: The process of serializing objects using the pickle module.

<u>polymorphism</u>: The meaning of an operation – generally a function/method call – depends on the objects being operated upon, not the *type* of object. One of Python's key principles is that object interfaces support operations regardless of the type of object being passed. For example, string methods support addition and multiplication just as methods on integers and other numeric objects do.

<u>recursion</u>: The ability of a function to call itself. Python supports recursion to a level of 1,000 unless that limit is modified using the setrecursionlimit function.

<u>scope</u>: The visibility of a name is its scope. All names within Python exist within a specific namespace which is tied to a single block, function, class, or module in which the name was last assigned a value.

<u>script</u>: A unit of code generally synonymous with a *program* but usually connotes code run at the highest level as in "scripts run modules".

self: By convention, the name given to a class' instance variable.

<u>sequence</u>: An ordered container of items that can be indexed or sliced using positive numbers. Python provides three built-in sequences: strings, tuples, and lists. New sequences can also be defined in libraries, extension modules, or within classes.

<u>set</u>: An unordered sequence of zero or more items which do not need to be of the same type. Sets can be frozen (immutable) or unfrozen (mutable).

<u>short-circuiting operators</u>: Operators and or can short-circuit the evaluation of their operand if the left side evaluates to true (in the case of the or) or false (in the case of and). For example, in the expression a or b, there is no need to evaluate b if a is True, likewise in the expression a and b, there is no need to evaluate b if a is False.

<u>statement</u>: An expression that generally occupies one line. Multiple statements can occupy the same line if separated by a semicolon (;) but this is very unconventional in Python where each line typically contains one statement.

<u>string</u>: A built-in sequence object consisting of one or more characters. Unlike many other languages, Python strings cannot be modified (that is, they are "immutable") and they do not have a termination character.

<u>tuple</u>: A sequence of zero or more items (for example, (1, 2, 3) or ("A", "B", "C")). Tuples are immutable and may contain different object types (for example, (1, "a", 5.678)).

<u>variable</u>: Python variables (that is, names) are not like variables in most other languages - they are never declared they are dynamically referenced to objects, they have no type, and they may be bound to objects of different types at different times. Variables are bound explicitly (for example, a = 1 binds a to the integer 1) and unbound implicitly (for example, a=1; a=2). In the last example, a is bound to the object (value) 1 then implicitly unbound to that object when bound to 2 - a process known as rebinding. Variables can also be unbound explicitly using the del statement (for example, del a, b, c).

.

© ISO/IEC 2015 – All rights reserved

4. Language concepts

The key concepts discussed in this section are not entirely unique to Python but they are implemented in Python in ways that are not intuitive to new and experienced programmers alike.

Dynamic Typing – A frequent source of confusion is Python's dynamic typing and its effect on variable assignments (*name* is synonymous with *variable* in this annex). In Python there are no static declarations of variables - they are created, rebound, and deleted dynamically. Further, variables are not the objects that they point to - they are just references to objects which can be, and frequently are, bound to other objects at any time:

a = 1 # a is bound to an integer object whose value is 1
a = 'abc' # a is now bound to a string object

Variables have no type – they reference objects which have types thus the statement a = 1 creates a new variable called a that references a new object whose value is 1 and type is integer. That variable can be deleted with a del statement or bound to another object any time as shown above. Refer to <u>6.2</u> Type System [IHN] for more on this subject. For the purpose of brevity this annex often treats the term variable (or name) as being the object which is technically incorrect but simpler. For example, in the statement a = 1, the numeric object a is assigned the value 1. In reality the name a is assigned to a newly created *object* of type integer which is assigned the value 1.

covers dynamic typing in more detail.

Mutable and Immutable Objects - Note that in the statement: a = a + 1, Python creates a *new* object whose value is calculated by adding 1 to the value of the current object referenced by a. If, prior to the execution of this statement a's object had contained a value of 1, then a new integer object with a value of 2 would be created. The integer object whose value was 1 is now marked for deletion using garbage collection (provided no other variables reference it). Note that the value of a is not updated in place, that is, the object references by a does not simply have 1 added to it as would be typical in other languages. The reason this does not happen in Python is because integer objects, as well as string, number and tuples, are immutable – they cannot be changed in place. Only lists and dictionaries can be changed in place – they are mutable. In practice this restriction of not being able to change a mutable object in place is mostly transparent but a notable exception is when immutable objects are passed as a parameter to a function or class. See <u>6.22 Initialization of Variables [LAV]</u> for a description of this.

The underling actions that are performed to enable the *apparent* in-place change do not update the immutable object – they create a new object and "point" the variable to new object. This can be proven as below (the id function returns an object's address):

```
a = 'abc'
print(id(a))#=> 30753768
a = 'abc' + 'def'
print(id(a))#=> 52499320
print(a)#=> abcdef
```

© ISO/IEC 2015 – All rights reserved

Deleted:

Deleted: E.3 Type System [IHN]

Deleted: Section 6.42 Violations of the Liskov Substitution
Principle or the Contract Model [BLP][4]
Deleted: E.43 Extra Intrinsics [LRM]

Deleted: 3

The updating of objects referenced in the parameters passed to a function or class is governed by whether the object is mutable, in which case it is updated in place, or immutable in which case a local copy of the object is created and updated which has no effect on the passed object. This is described in more detail in <u>6.32 Passing</u> Parameters and Return Values [CSJ].

5. General guidance for Python

5.1 Top avoidance mechanisms

Each vulnerability listed in clause 6 provides a set of ways that the vulnerability can be avoided or mitigated. Many of the mitigations and avoidance mechanisms are common. This subclause provides the most most effective and the most common mitigations, together with references to which vulnerabilities they apply. The references are hyperlinked to provide the reader with easy access to those vulnerabilities for rationale and further exploration. The mitigations provided here are in addition to the ones provided in TR 24772-1, clause 5.4

The expectation is that users of this document will develop and use a coding standard based on this document that is tailored to their risk environment.

Number	Recommended avoidance mechanism	References	
<u>1</u>	Do not use floating-point arithmetic when integers or booleans		
	would suffice		
<u>2</u>	Use of enumeration requires careful attention to readability,	•	 Formatted: Normal, Indent: Left: 0.63 cm, Space After: 0
	performance, and safety. There are many complex, but useful		pt, Line spacing: single, No bullets of humbering
	ways to simulate enums in Python [(Enums for Python (Python		
	recipe))]and many simple ways including the use of sets:		
	<pre>colors = {'red', 'green', 'blue'}</pre>		
	if red in colors: print('valid color')		
	Be aware that the technique shown above, as with almost all		
	other ways to simulate enums, is not safe since the variable		
	can be bound to another object at any time.		
	en functions return error values, check the error return values before		
	processing any other returned data.		
3	Ensure that when examining code that you take into account	6 4	 Formatted: Indent: Left: 0.63 cm, Space After: 0 pt, Line
_	that a variable can be bound (or rebound) to another object (of	_	spacing: single, Adjust space between Latin and Asian text,
	same or different type) at any time.		Adjust space between Asian text and numbers
4	Avoid implicit references to global values from within functions to		Formatted: Font:11 pt, Not Bold
	make code clearer. In order to update globals within a function or		
	class, place the global statement at the beginning of the function		
	definition and list the variables so it is clearer to the reader which		
	variables are local and which are global (for example, global a, b, c)		
	windores are rocar and which are global (for example, global a, b, c).		
5	Use only spaces or tabs, not both, to indent to demark control flow.	4	 Formatted: Font:Not Bold, English
	Never use form feed characters for indentation		 Formatted: Space Before: 0 pt, After: 0 pt, Line spacing:
			single

© ISO/IEC 2015 - All rights reserved

Deleted: 3

Deleted: [See Template] [Thoughts welcomed as to what could be provided here. Possibly an opportunity for the language community to address issues that do not correlate to

the guidance of section 6. For languages that provide nonmandatory tools, how those tools can be used to provide

effective mitigation of vulnerabilities described in the following sections]

6	Use Python's huilt-in documentation (such as docstrings) to obtain			
<u> </u>	information about a class' method before inheriting from it			
	mornation about a class method before innerting from it			
<u>7</u>	Either avoid logic that depends on byte order or use the			Formatted: Space After: 0 pt, Line spacing: single, Adjust
	sys.byteorder variable and write the logic to account for byte order			space between Latin and Asian text, Adjust space between
	dependent on its value ('little' or 'big').			Exemptited: Exert:11 at Not Bold
<u>8</u>	When launching parallel tasks don't raise a BaseException subclass in			Tormatted. Font.11 pt, Not bold
	a callable in the Future class			
0				
<u>9</u>	Do not depend on the way Python may or may not optimize object			Deleted: 10
	references for small integer and string objects because it may vary			
	for environments or even for releases in the same environment.			
10	Be aware of short-circuiting behaviour when expressions with side effects			Deleted: 8
	are used on the right side of a Boolean expression such as if the first			Perettano
	expression evaluates to false in an and expression, then the remaining			
	expressions, including functions calls, will not be evaluated.			
.				
<u>11</u>	Do not use floating-point arithmetic when integers or booleans would			Deleted: 19
	suffice, especially for counters associated with program flow, such as loop			Deleted: 20
	<u>control variables.</u>			
12	Sanitize, erase or encrypt data that will be visible to others (for example,			Deleted: 21
	freed memory, transmitted data).			Comment [Office1]: This is a section 7 rule?
				Yes, but this section will cover sections 6 and 7. One we pull up rules
				from clause 7, we will need to triage.
				Formatted: Normal
6. Spe	cific Guidance for Python			
6.1 Ge	neral			
This clau	se contains specific advice for Python about the possible presence of vuln	erabilities as described in	٠٠٠٠٠	Formatted: Normal
TR 24772	2-1, and provides specific guidance on how to avoid them in Python code.	This section mirrors TR 24772-	1	
clause 6	in that the vulnerability "Type System [IHN]" is found in 6.2 of TR 24772-1	L, and Python specific guidance		
is found	in clause 6.2 and subclauses in this TR.			
Sav som	ething about the changes from Python 2 to Python 3, not backwards comp	atible		
04) 0011				
How do	we treat libraries? Python has many libraries that essentially change the p	rogramming paradigm.		
			1	Deleted: 3
© ISO/I	EC 2015 – All rights reserved	7		
	7			

6.2, Type System [IHN]

6.2,1 Applicability to language

Python abstracts all data as objects and every object has a type (in addition to an identity and a value). Extensions to Python, written in other languages, can define new types.

Python is also a strongly typed language – you cannot perform operations on an object that are not valid for that type. Python's dynamic typing is a key feature designed to promote polymorphism to provide flexibility. Another aspect of dynamic typing is a variable does not maintain any type information – that information is held by the object that the variable references at a specific time. A Python program is free to assign (bind), and reassign (rebind), any variable to any type of object at any time.

Variables are created when they are first assigned a value (see <u>6.17 Choice of Clear Names [NAI]</u> for more on this subject). Variables are generic in that they do not have a type, they simply reference objects which hold the object's type information. Variables in an expression are replaced with the object they reference when that expression is evaluated therefore a variable must be explicitly assigned before being referenced otherwise a runtime exception is raised:

a = 1
if a == 1 : print(b) # error - b is not defined

When line 1 above is interpreted an object of type integer is created to hold the value 1 and the variable a is created and linked to that object. The second line illustrates how an error is raised if a variable (b in this case) is referenced before being assigned to an object.

a = 1
b = a
a = 'x'
print(a,b) #=> x 1

Variables can share references as above – b is assigned to the same object as a. This is known as a shared reference. If a is later reassigned to another object (as in line 3 above), b will still be assigned to the initial object that a was assigned to when b shared the reference, in this case b would equal to 1.

The subject of shared references requires particular care since its effect varies according to the rules for in-place object changes. In-places object changes are allowed only for mutable (that is, alterable) objects. Numeric objects and strings are immutable (unalterable). Lists and dictionaries are mutable which affects how shared references operate as below:

```
a = [1,2,3]
b = a
a[0] = 7
print(a) # [7, 2, 3]
print(b) # [7, 2, 3]
```

8

Deleted: E.3

Comment [SGM2]: Recommendation from Nick Coghlan: - the section on typing should discuss the official introduction of gradual typing, and the availability of static type checkers such as mypy and pytype (see PEP 484 and 526)

Deleted: E.3

© ISO/IEC 2015 – All rights reserved

In the example above, a and b have a shared reference to the same list object so a change to that list object affects both references. If the shared reference effects are not well understood the change to b can cause unexpected results.

Automatic conversion occurs only for numeric types of objects. Python converts (coerces) from the simplest type up to the most complex type whenever different numeric types are mixed in an expression. For example:

```
a = 1

b = 2.0

c = a + b; print(c) #=> 3.0
```

In the example above, the integer a is converted up to floating point (that is, 1.0) before the operation is performed. The object referred to by a is not affected – only the intermediate values used to resolve the expression are converted. If the programmer does not realize this conversion takes place he may expect that c is an integer and use it accordingly which could lead to unexpected results.

Automatic conversion also occurs when an integer becomes too large to fit within the constraints of the large integer specified in the language (typically C) used to create the Python interpreter. When an integer becomes too large to fit into that range it is converted to an unlimited precision integer of arbitrary length.

Explicit conversion methods can also be used to explicitly convert between types though this is seldom required since Python will automatically convert as required. Examples include:

```
a = int(1.6666) # a converted to 1
b = float(1) # b converted to 1.0
c = int('10') # c integer 10 created from a string
d = str(10) # d string '10' created from an integer
e = ord('x') # e integer assigned integer value 120
f = chr(121) # f assigned the string 'y'
```

Dynamic typing is a key feature of Python which promotes polymorphism for flexibility. Strict typing can, however, be imposed:

```
a = 'abc' # a refers to a string object
if isinstance(a, str): print('a type is string')
```

Using code to explicitly check the type of an object is strongly discouraged in Python since it defeats the benefit that dynamic typing provides - flexibility which allows functions to potentially operate correctly with objects of more than one type.

<u>6.2</u> .2 Guidance to language users	D	eleted: E.3
 Use static type checkers such as mypy and pytype to detect typing errors 		
Pay special attention to issues of magnitude and precision when using mixed type expressions;		
Be aware of the consequences of shared references;		
Be aware of the conversion from simple to complex; and		
• Do not check for specific types of objects unless there is good justification, for example, when calling an		
	D	eleted: 3
© ISO/IEC 2015- All rights reserved 9		

extension that requires a specific type.

extension that requires a specific type.	
6.3 Bit Representations [STR]	Deleted: E.4
<u>6.3</u> ,1 Applicability to language	Deleted: E.4
Python provides hexadecimal, octal and binary built-in functions. oct converts to octal, hex to hexadecimal and bin to binary:	
print(oct(256)) # 0o400 print(hex(256)) # 0x100 print(bin(256)) # 0b10000000	
The notations shown as comments above are also valid ways to specify octal, hex and binary values respectively:	
print(00400)# => 256 a=0x100+1; print(a)# => 257	
The built-in int function can be used to convert strings to numbers and optionally specify any number base:	
<pre>int('256') # the integer 256 in the default base 10 int('400', 8) # => 256 int('100', 16) # => 256 int('24', 5) # => 14</pre>	
Python stores integers that are beyond the implementation's largest integer size as an internal arbitrary length so that programmers are only limited by performance concerns when very large integers are used (and by memory when extremely large numbers are used). For example:	
a=2**100 # => 1267650600228229401496703205376	
Python treats positive integers as being infinitely padded on the left with zeroes and negative numbers (in two's complement notation) with 1's on the left when used in bitwise operations:	
a< b # a shifted left b bits a>>b # a shifted right b bits	
There is no overflow check for shifting left or right so a program expecting an exception to halt it will instead unexpectedly continue leading to unexpected results.	
<u>6.3</u> ,2 Guidance to language users	Deleted: E.4
 Keep in mind that using a very large integer will have a negative effect on performance; and Don't use bit operations to simulate multiplication and division. 	
	Deleted: 3
10 © ISO/IEC 2015-All rights reserved	

6.4 Floating-point Arithmetic [PLF]	Deleted: E.
<u> </u>	Deleted: 5
6.4,1 Applicability to language	Deleted: E.5
Python supports floating-point arithmetic. Literals are expressed with a decimal point and or an optional $ m e$ or $\mathbb E$:	
1., 1.0, .1, 1.e0	
There is no way to determine the precision of the implementation from within a Python program. For example, in the CPython implementation, it's implemented as a C double which is approximately 53 bits of precision.	
6.4.2 Guidance to language users	Deleted: E.5
 Use floating-point arithmetic only when absolutely needed; Do not use floating-point arithmetic when integers or booleans would suffice; Be aware that precision is lost for some real numbers (that is, floating-point is an approximation with limited precision for some numbers); Be aware that results will frequently vary slightly by implementation (see <u>6.53 Provision of Inherently Unsafe Operations [SKL]</u> for more on this subject); and Testing floating-point numbers for equality (especially for loops) can lead to unexpected results. Instead, if floating-point numbers are needed for loop control use >= or <= comparisons, <u>unless it can be shown</u> that the logic implemented by the equality test cannot be affected by prior rounding errors. 	
6.5. Enumerator Issues [CCB]	Deleted: E.6
6.5,1 Applicability to language	Comment [SGM3]: From Nick Coghlan (2017-09-21) - the section on enumerations should discuss the standa
Python has an enumerate built-in type but it is not at all related to the implementation of enumeration as defined in other languages where constants are assigned to symbols. Given that enumeration is a useful programming device and that there is no enumeration construct in Python, many programmers choose to implement their own "enum" objects or types using a wide variety of methods including the creation of "enum" classes, lists, and even dictionaries. One simple method is to simply assign a list of names to integers:	library's enum module (added in Python 3.4, available for 2.7 on l as enum34) Deleted: E.6
Red, Green, Blue = range (3) print(Red, Green, Blue) # => 0 1 2	
Code can then reference these "enum" values as they would in other languages which have native support for enumeration:	
<pre>a = 1 if a == Green: print("a=Green")# => a=Green</pre>	
There are disadvantages to the approach above though since any of the "enum" variables could be assigned new values at any time thereby undoing their intended role as "pseudo" constants. There are many forum discussions and articles that illustrate other, safer ways to simulate enumeration which are beyond the scope of this annex.	
	Deleted: 3

<u>6.5</u> ,2 Guidance to language users	Deleted: E.6
Use of enumeration requires careful attention to readability, performance, and safety. There are many complex, but useful ways to simulate enums in Python [[1]]and many simple ways including the use of sets:	
<pre>colors = {'red', 'green', 'blue'} if "red" in colors: print('valid color')</pre>	
Be aware that the technique shown above, as with almost all other ways to simulate enums, is not safe since the variable can be bound to another object at any time.	
6.6 Conversion Errors [FLC]	Comment [SM4]: We removed "Numeric" from "Numeric Conversion Error" and are generalizing the issues. Please try to
6.6.1 Applicability to language	ensure that Python 6.6 is in sync.
	Deleted: Numeric
Python converts numbers to a common type before performing any arithmetic operations. The common type is	Deleted: F7
coerced using the following rules as defined in the standard (http://docs.python.org/release/1.4/ref/ref5.html):	Comment [SM5]: But in bibliography and reference
<pre>If either argument is a complex number, the other is converted to the complex type; otherwise, if either argument is a long integer, the other is converted to floating point; otherwise, if either argument is a long integer, the other is converted to long integer; otherwise, both must be plain integers and no conversion is necessary. Integers in the Python language are of a length bounded only by the amount of memory in the machine. Integers are stored in an internal format that has faster performance when the number is smaller than the largest integer supported by the implementation language and platform. Implicit or explicit conversion floating point to integer, implicitly (or explicitly using the int function), will typically cause a loss of precision:</pre>	
<u>6.6</u> ,2 Guidance to language users	Deleted: E.7
 Though there is generally no need to be concerned with an integer getting too large (rollover) or small, be aware that iterating or performing arithmetic with very large positive or small (negative) integers will hurt performance; and Be aware of the potential consequences of precision loss when converting from floating point to integer. 	
6.7, String Termination [CJM]	Deleted: E.8
This vulnerability is not applicable, Python strings are immutable objects whose length can be queried with built- in functions therefore Python does not permit accesses past the end, or beginning, of a string.	Deleted: 3
ISO/IEC 2015- All rights reserved	

<pre>a = '12345' b = a[5] #=> IndexError: string index out of range</pre>	
6.8, Buffer Boundary Violation [HCB]	Deleted: E.9
This vulnerability is not applicable to Python because Python's run-time checks the boundaries of arrays and raises an exception when an attempt is made to access beyond a boundary.	
6.9 Unchecked Array Indexing [XYZ]	Deleted: E.10
This vulnerability is not applicable to Python because Python's run-time checks the boundaries of arrays and raises an exception when an attempt is made to access beyond a boundary.	
6.10, Unchecked Array Copying [XYW]	Deleted: E.11
This vulnerability is not applicable to Python because Python's run-time checks the boundaries of arrays and raises an exception when an attempt is made to access beyond a boundary.	
6.11 Pointer Type Conversions [HFC]	Deleted: E.12
This vulnerability is not applicable to Python because Python does not use pointers.	
6.12, Pointer Arithmetic [RVG]	Deleted: E.13
This vulnerability is not applicable to Python because Python does not use pointers.	
6.13, Null Pointer Dereference [XYH]	Deleted: E.14
This vulnerability is not applicable to Python because Python does not use pointers.	
6.14 Dangling Reference to Heap [XYK]	Deleted: E.15
This vulnerability is not applicable to Python because Python does not use pointers. Specifically, Python only uses namespaces to access objects therefore when an object is deallocated, any reference to it causes an exception to be raised.	
6.15 Arithmetic Wrap-around Error [FIF]	Deleted: E.16
<u>6.15</u> ,1 Applicability to language	Deleted: E.16
Operations on integers in Python cannot cause wrap-around errors because integers have no maximum size other than what the memory resources of the system can accommodate.	
Normally the OverflowError exception is raised for floating point wrap-around errors but, for implementations of Python written in C, exception handling for floating point operations cannot be assumed to catch this type of error because they are not standardized in the underlying C language. Because of this, most floating point operations cannot be depended on to raise this exception.	
	Deleted: 3

6.15,2 Guidance to language users

- Be cognizant that most arithmetic and bit manipulation operations on non-integers have the potential for undetected wrap-around errors.
- Avoid using floating point or decimal variables for loop control but if you must use these types then bound the loop structures so as to not exceed the maximum or minimum possible values for the loop control variables.
- Test the implementation that you are using to see if exceptions are raised for floating point operations
 and if they are then use exception handling to catch and handle wrap-around errors.

6.16, Using Shift Operations for Multiplication and Division [PIK]

This vulnerability is not applicable to Python because it does not check for overflow. In addition there is no practical way to overflow an integer since integers have unlimited precision.

>>> print(-1<<100)#=> -1267650600228229401496703205376
>>> print(1<<100) #=> 1267650600228229401496703205376

6.17 Choice of Clear Names [NAI]

6.17,1 Applicability to language

Python provides very liberal naming rules:

- Names may be of any length and consist of letters, numerals, and underscores only. All characters in a name are significant. Note that unlike some other languages where only the first *n* number of characters in a name are significant, *all* characters in a Python name are significant. This eliminates a common source of name ambiguity when names are identical up to the significant length and vary afterwards which effectively makes all such names a reference to one common variable.
- All names must start with an underscore or a letter; and
- Names are case sensitive, for example, Alpha, ALPHA, and alpha are each unique names. While this is
 a feature of the language that provides for more flexibility in naming, it is also can be a source of
 programmer errors when similar names are used which differ only in case, for example, aLpha versus
 alpha.

The following naming conventions are not part of the standard but are in common use:

- Class names start with an upper case letter, all other variables, functions, and modules are in all lower case;
- Names starting with a single underscore (_) are not imported by the from module import * statement – this not part of the standard but most implementations enforce it; and
- Names starting and ending with two underscores (___) are system-defined names.
- Names starting with, but not ending with, two underscores are local to their class definition
- Python provides a variety of ways to package names into namespaces so that name clashes can be avoided:
- Names are scoped to functions, classes, and modules meaning there is normally no collision with names utilized in outer scopes and vice versa; and

14

© ISO/IEC 2015 – All rights reserved

Deleted: E.16

Deleted: E.17

Deleted: 6.16E.17.1 Applicability to language

Formatted: French

Deleted: 6.17E.18 Sign Extension Error [XZI]

Deleted: 8 Deleted: E.19

Comment [SGM6]: Email from Nick Coghlan (2017-09-21) - the section on ambiguous naming needs to be updated to account for

full Unicode identifier support in Python 3:

Confused = True Confused = False Confused == Confused False

"Confused" 'Confused'

ascii("Confused") "<u>\\u0421onfused"</u>

ascii("Confused") "'Confused""

Deleted: 8 Deleted: E.19

Names in modules (a file containing one or more Python statements) are local to the module and are
referenced using qualification (for example, a function x in module y is referenced as y.x). Though local
to the module, a module's names can be, and routinely are, copied into another namespace with a from
module import statement.

Python's naming rules are flexible by design but are also susceptible to a variety of unintentional coding errors:

- Names are never declared but they must be assigned values before they are referenced. This means that
 some errors will never be exposed until runtime when the use of an unassigned variable will raise an
 exception (see <u>6.22 Initialization of Variables [LAV]</u>).
- Names can be unique but may look similar to other names, for example, alpha and alpha, __x and __x, _beta__ and __beta_ which could lead to the use of the wrong variable. Python will not detect this problem at compile-time.

Python utilizes dynamic typing with types determined at runtime. There are no type or variable declarations for an object ,which can lead to subtle and potentially catastrophic errors:

```
x = 1
# lots of code...
if some rare but important case:
        X = 10
```

In the code above the programmer intended to set (lower case) \times to 10 and instead created a new *upper case* \times to 10 so the *lower case* \times remains unchanged. Python will not detect a problem because there is no problem – it sees the upper case \times assignment as a legitimate way to bring a *new* object into existence. It could be argued that Python could statically detect that \times is never referenced and therefore indicate the assignment is dubious but there are also cases where a dynamically defined function defined downstream could legitimately reference \times as a global.

6.17,2 Guidance to language users

- For more guidance on Python's naming conventions, refer to Python Style Guides contained in PEP 8 at http://www.python.org/dev/peps/pep-0008/.
- Avoid names that differ only by case unless necessary to the logic of the usage;
- Adhere to Python's naming conventions;
- Do not use overly long names;
- Use names that are not similar (especially in the use of upper and lower case) to other names;
- Use meaningful names; and
- Use names that are clear and visually unambiguous because the compiler cannot assist in detecting
 names that appear similar but are different.

ISO/IEC	2015-	All rights	reserved

Deleted: 3

Deleted: 8 Deleted: E.19

6.18 Dead Store [WXQ]

6.18,1 Applicability to language

It is possible to assign a value to a variable and never reference that variable which causes a "dead store". This in itself is not harmful, other than the memory that it wastes, but if there is a substantial amount of dead stores then performance could suffer or, in an extreme case, the program could halt due to lack of memory.

Python provides the ability to dynamically create variables when they are first assigned a value. In fact, assignment is the *only* way to bring a variable into existence. All values in a Python program are accessed through a reference which refers to a memory location which is always an object (for example, number, string, list, and so on). A variable is said to be bound to an object when it is assigned to that object. A variable can be rebound to another object which can be of any type. For example:

The first three statements show dynamic binding in action. The variable a is bound to a string, then to a float, then to another variable which in turn is assigned a tuple of value (1, 2, 3). The del statement then unbinds the variable a from the tuple object which effectively deletes the a variable (if there were no other references to the tuple object it too would have been deleted because an object with zero references is *marked* for garbage collection (but is not necessarily actually deleted immediately)). But in this case we see that b is still referencing the tuple object so the tuple is not deleted. The final statement above shows that an exception is raised when an unbound variable is referenced.

The way in which Python dynamically binds and rebinds variables is a source of some confusion to new programmers and even experienced programmers who are used to static binding where a variable is permanently bound to a single memory location.

The Python language, by design, allows for dynamic binding and rebinding. Because Python performs a syntactic analysis and not a semantic analysis (with one exception which is covered in <u>6.21</u> Namespace Issues [BJL]. Applicability to language) and because of the dynamic way in which variables are brought into a program at runtime, Python cannot warn that a variable is referenced but never assigned a value. The following code illustrates this:

```
if a > b:
    import x
else:
    import y
```

Depending on the current value of a and b, either module x or y is imported into the program. If x assigns a value to a variable z and module y references z then, dependent on which import statement is executed first

16

© ISO/IEC 2015 – All rights reserved

Deleted: 9

Deleted: E.20 Comment [SGM7]: Email from Nick Coghlan (2017-09-21) - the discussion of dead stores may want to mention ResourceWarning (which emits a warning when external resources are cleaned up implicitly rather than explicitly) and the tracemalloc module (which allows resource warnings to report where the resource managing object was allocated) Deleted: 9

Deleted: E.20

Deleted: E.22.1 Namespace Issues [BJL]

(an import always executes all code in the module when it is first imported), an unassigned variable reference exception will or will not be raised.

6.18,2 Guidance to language users

- Avoid rebinding except where it adds value;
- Ensure that when examining code that you take into account that a variable can be bound (or rebound) to another object (of same or different type) at any time; and
- Variables local to a function are deleted automatically when the encompassing function is exited but, though not a common practice, you can also explicitly delete variables using the del statement when they are no longer needed.

6.19, Unused Variable [YZS]

The applicability to language and guidance to language users sections of <u>TR 24772-1 clause 6.18</u> Dead Store [WXQ] write-up are applicable to Python.

6.20, Identifier Name Reuse [YOW]

6.20,1 Applicability to language

Python has the concept of namespaces which are simply the places where names exist in memory. Namespaces are associated with functions, classes, and modules. When a name is created (that is, when it is first assigned a value), it is associated (that is, bound) to the namespace associated with the location where the assignment statement is made (for example, in a function definition). The association of a variable to a specific namespace is elemental to how scoping is defined in Python.

Scoping allows for the definition of more than one variable with the same name to reference different objects. For example:

a =	1	
def	x():	
	a = 2	
	<pre>print(a) #=></pre>	2
prir	nt(a) #=> 1	

The a variable within the function x above is local to the function only – it is created when x is called and disappears when control is returned to the calling program. If the function needed to update the outer variable named a then it would need to specify that a was a global before referencing it as in:

a =	1	
def	x():	
	global a	
	a = 2	
	<pre>print(a) #=></pre>	2
prin	nt(a) #=> 2	

© ISO/IEC 2015 – All rights reserved

-{	Deleted: 20
{	Deleted: E.21
{	Deleted: the
{	Deleted: E.19
{	Deleted: here
{	Deleted: 21
·{	Deleted: E.22
{	Deleted: 1
{	Deleted: E.22

Deleted: 9 Deleted: E.20

In the case above, the function is updating the variable a that is defined in the calling module. There is a subtle but important distinction on the locality versus global nature of variables: *assignment* is always local unless global is specified for the variable as in the example above where a is *assigned* a value of 2. If the function had instead simply *referenced* a without assigning it a value, then it would reference the topmost variable a which, by definition, is always a global:

```
a = 1
def x():
    print(a)
x() #=> 1
```

The rule illustrated above is that attributes of modules (that is, variable, function, and class names) are global to the module meaning any function or class can reference them.

Scoping rules cover other cases where an identically named variable name references different objects:

- A nested function's variables are in the scope of the nested function only; and
- Variables defined in a module are in *global* scope which means they are scoped to the module only and are therefore not visible within functions defined in that module (or any other function) unless explicitly identified as global at the start of the function.

Python has ways to bypass implicit scope rules:

- The global statement which allows an inner reference to an outer scoped variable(s); and
- The nonlocal statement which allows an enclosing function definition to reference a nested function's variable(s).

The concept of scoping makes it safer to code functions because the programmer is free to select any name in a function without worrying about accidentally selecting a name assigned to an outer scope which in turn could cause unwanted results. In Python, one must be explicit when intending to circumvent the intrinsic scoping of variable names. The downside is that identical variable names, which are totally unrelated, can appear in the same module which could lead to confusion and misuse unless scoping rules are well understood.

Names can also be qualified to prevent confusion as to which variable is being referenced:

```
a = 1
class xyz():
    a = 2
    print(a) #=> 2
print(xyz.a, a) #=> 2 1
```

The final print function call above references the a variable within the xyz class and the global a.

6.20,2 Guidance to language users

- Do not use identical names unless necessary to reference the correct object;
- Avoid the use of the global and nonlocal specifications because they are generally a bad programming practice for reasons beyond the scope of this annex and because their bypassing of

Deleted: 1 Deleted: E.22

Deleted: 3

18

© ISO/IEC 2015 – All rights reserved

standard scoping rules make the code harder to understand; and

٠ Use qualification when necessary to ensure that the correct variable is referenced.

6.21, Namespace Issues [BJL]

6.21,1 Applicability to language

Python has a hierarchy of namespaces which provides isolation to protect from name collisions, ways to explicitly reference down into a nested namespace, and a way to reference up to an encompassing namespace. Generally speaking, namespaces are very well isolated. For example, a program's variables are maintained in a separate namespace from any of the functions or classes it defines or uses. The variables of modules, classes, or functions are also maintained in their own protected namespaces.

Accessing a namespace's attribute (that is, a variable, function, or class name), is generally done in an explicit manner to make it clear to the reader (and Python) which attribute is being accessed:

n = Animal.num # fetches a class' variable called num x = mymodule.y # fetches a module's variable called y

The examples above exhibit qualification - there is no doubt where a variable is being fetched from. Qualification can also occur from an encompassed namespace up to the encompassing namespace using the global statement:

def x():		Formatted: Spanish
global y		
y = 1		
The example above uses an explicit global statement which makes it clear that the variable y is not local to the		
unction x; it assigns the value of 1 to the variable y in the encompassing module 1.		
Python also has some subtle namespace issues that can cause unexpected results especially when using imports		
of modules. For example, assuming module a . by contains:		
a = 1		
and module b . py contains:		
b = 1		
Executing the following code is not a problem since there is no variable name collision in the two modules (the		
from modulename import * statement brings all of the attributes of the named module into the local		
namespace):		
from a import *		
$\operatorname{print}(a) = 1$		
Values are assigned to objects which in turn are referenced by variables but it's simpler to say the value is assigned to the variable. Also, he encompassing code could be at a promot level instead of a module. For brevity this annex uses this simpler, though not as exact.		
vording.	ļ	Deleted: 3
	1	
© ISO/IEC 201 <u>5</u> – All rights reserved 19	1	

Deleted: 2 Deleted: E.23

class body

Deleted: 2

Deleted: E.23

types.prepare_class and

(see

Comment [SGM8]: Email from Nick Coghlan (2017-09-21) metaclass __prepare__ methods can inject extra names into a

execution namespace that the compiler knows nothing about

https://docs.python.org/3/reference/datamodel.html#preparing-the-class-namespace)

from b import *
print(b) #=> 1

Later on the author of the b module adds a variable named a and assigns it a value of 2. b. py now contains:

```
b = 1
a = 2 \# new assignment
```

The programmer of module $b \cdot py$ may have no knowledge of the a module and may not consider that a program would import both a and b. The importing program, with no changes, is run again:

```
from a import *
print(a) #=> 1
from b import *
print(a) #=> 2
```

The results are now different because the importing program is susceptible to unintended consequences due to changes in variable assignments made in two unrelated modules as well as the sequence in which they were imported. Also note that the from *modulename* import * statement brings all of the modules attributes into the importing code which can silently overlay like-named variables, functions, and classes.

A common misunderstanding of the Python language is that Python detects local names (a local name is a name that lives within a class or function's namespace) *statically* by looking for one or more assignments to a name within the class/function. If one or more assignments are found then the name is noted as being local to that class/function. This can be confusing because if only *references* to a name are found then the name is referencing a global object so the only way to know if a reference is local or global, barring an explicit global statement, is to examine the entire function definition looking for an assignment. This runs counter to Python's goal of Explicit is Better Than Implicit (EIBTI):

```
a = 1
def f():
print(a)
a = 2
f() #=> UnboundLocalError: local variable 'a' referenced before
       assignment
# now with the assignment commented out
a = 1
def f():
print(a) #=> 1
#a = 2
# Assuming a new session:
a = 1
def f():
   global a
   a = 2
f()
```

```
print(a) #=> 2
```

20

© ISO/IEC 2015 - All rights reserved

Note that the rules for determining the locality of a name applies to the assignment operator = as above, but also to all other kinds of assignments which includes module names in an import statement, function and class names, and the arguments declared for them. See <u>6.19 Unused Variable [YZS]</u> for more detail on this.

Name resolution follows a simple Local, Enclosing, Global, Built-ins (LEGB) sequence:

- First the local namespace is searched;
- Then the enclosing namespace (that is, a def or lambda (A lambda is a single expression function definition));
- Then the global namespace; and
- Lastly the built-in's namespace.

6.21,2 Guidance to language users

- When practicable, consider using the import statement without the from clause. This forces the importing program to use qualification to access the imported module's attributes. While it is true that using the from statement is more convenient due to less typing required (for example, no need to qualify names), the from statement can cause namespace corruption;
- When using the import statement, rather than use the from X import * form (which imports all of module X's attributes into the importing program's namespace), instead explicitly name the attributes that you want to import (for example, from X import a, b, c) so that variables, functions and classes are not inadvertently overlaid; and
- Avoid implicit references to global values from within functions to make code clearer. In order to update
 globals within a function or class, place the global statement at the beginning of the function definition
 and list the variables so it is clearer to the reader which variables are local and which are global (for
 example, global a, b, c).

6.22, Initialization of Variables [LAV]

6.22,1 Applicability of language

Python does not check to see if a statement references an uninitialized variable until runtime. This is by design in order to support dynamic typing which in turn means there is no ability to declare a variable. Python therefore has no way to know if a variable is referenced before or after an assignment. For example:

if y > 0:
 print(x)

The above statement is legal at compile time even if x is not defined (that is, assigned a value). An exception is raised at runtime only if the statement is executed and y>0. This scenario does not lend itself to static analysis because, as in the case above, it may be perfectly logical to not ever print x unless y>0.

There is no ability to use a variable with an uninitialized value because *assigned* variables always reference objects which always have a value and *unassigned* variables do not exist. Therefore Python raises an exception when an unassigned (that is, non-existent) variable is referenced.

© ISO/IEC 2015 – All rights reserved

Deleted: 2

Deleted: E.23

Deleted: 3 Deleted: E.24

Deleted: 3 Deleted: E.24

Deleted: 3

Initialization of class arguments can cause unexpected results when an argument is set to a default object which is mutable:

```
def x(y=[]):
    y.append(1)
    print(y)
x([2])#=> [2, 1], as expected (default was not needed)
x() # [1]
x() # [1, 1] continues to expand with each subsequent call
```

The behaviour above is not a bug - it is a defined behaviour for mutable objects but it's a very bad idea in almost all cases to assign default values to mutable objects.

6.22,2 Guidance to language users

• Ensure that it is not logically possible to reach a reference to a variable before it is assigned. The example above illustrates just such a case where the programmer wants to print the value of x but has not assigned a value to x - this proves that there is missing, or bypassed, code needed to provide x with a meaningful value at runtime.

6.23, Operator Precedence and Associativity [JCW]

6.23,1 Applicability to language

Python provides many operators and levels of precedence so it is not unexpected that operator precedence and order of operation are not well understood and hence misused. For example:

```
1 + 2 * 3 #=> 7, evaluates as 1 + (2 * 3) (1 + 2) * 3 #=> 9, parenthesis are allowed to coerce precedence
```

Expressions that use and or or are evaluated left-to-right which can cause a short circuit:

a or b or c

In the expression above c is never evaluated if either a or b evaluate to True because the entire expression evaluates to True immediately when any sub expression evaluates to True. The short circuit effect is non-consequential above but in the case below the effect is subtle and potentially destructive:

```
def x(i):
    if i:
        return True
    else:
        1/0 # Hard stop
a = 1
b = 0
while True:
    if x(a) or x(b):
        print('a or b is True')
```

Deleted: 3 Deleted: E.24

	Deleted: 4
	Deleted: E.25
	Deleted: /Order of Evaluation
	Deleted: 4
1	Deleted: E.25

Deleted: 3

© ISO/IEC 2015 - All rights reserved

The code above will go into an endless loop because x (b) is never evaluated. If it was the program would terminate due to an attempted division by zero.

6.23,2 Guidance to language users

- Use parenthesis liberally to force intended precedence and increase readability;
- Be aware that short-circuited expressions can cause subtle errors because not all sub-expressions may be evaluated; and
- Break large/complex statements into smaller ones using temporary variables for interim results

6.24 Side-effects and Order of Evaluation of Operands [SAM]

6.24,1 Applicability to language

Python supports sequence unpacking (parallel assignment) in which each element of the right hand side (expressed as a tuple) is evaluated and then assigned to each element of the left-hand side (LHS) in left-to-right sequence. For example, the following is a safe way to exchange values in Python:

```
a = 1

b = 2

a, b = b, a \# swap values between a and b

print (a,b) \# > 2, 1
```

Assignment of the targets (LHS) proceeds left-to-right so overlaps on the left side are not safe:

```
a = [0,0]
i = 0
i, a[i] = 1, 2 #=> Index is set to 1; list is updated at [1]
print(a) #=> 0,2
```

Python Boolean operators are often used to assign values as in:

a = b or c or d or None

a is assigned the first value of the first object that has a non-zero (that is, True) value or, in the example above, the value None if b, c, and d are all False. This is a common and well understood practice. However, trouble can be introduced when functions or other constructs with side effects are used on the right side of a Boolean operator:

if a() or b()

If function a returns a True result then function b will not be called which may cause unexpected results.

6.24,2 Guidance to language users

• Be aware of Python's short-circuiting behaviour when expressions with side effects are used on the right side of a Boolean expression; if necessary perform each expression first and then evaluate the results:

x = a()

© ISO/IEC 2015 - All rights reserved

he program would	
	Deleted: 4
	Deleted: E.25
; Il sub-expressions may be	
r interim results.	
	Deleted: 5
	Deleted: E.26
e right hand side ide (LHS) in left-to-right	Comment [SGM9]: Email from Nick Coghlan (2017-09-21) - for order of evaluation: it was noticed a couple of years ago that dictionary displays didn't actually evaluate in the expected left to right order (they went value/key rather than key/value). This has been fixed (in 3.6 if I recall correctly), but may be useful as an example of the value of ensuring that operations with side effects don't depend on subtle order of evaluation details
	Deleted: 6
	Deleted: E.26
at [1]	
or, in the example above, actice. However, trouble right side of a Boolean	
nexpected results.	Deleted: 5
ects are used on the right n evaluate the results:	Deleted: E.26
	Deleted: 3

y = b() if x or y ...

Be aware that, even though overlaps between the left hand side and the right hand side are safe, it is possible to have unintended results when the variables on the left side overlap with one another so always ensure that the assignments and left-to-right sequence of assignments to the variables on the left hand side never overlap. If necessary, and/or if it makes the code easier to understand, consider breaking the statement into two or more statements;

overlapping a = [0,0] i = 0 i, a[i] = 1, 2 #=> Index is set to 1; list is updated at [1] print(a) #=> 0,2 # Non-overlapping a = [0,0] i, a[0] = 1, 2 print(a) #=> 2,0

6.25, Likely Incorrect Expression [KOA]

6.25,1 Applicability to language

Python goes to some lengths to help prevent likely incorrect expressions:

```
    Testing for equivalence cannot be confused with assignment:
    a = b = 1
    if (a=b): print(a,b) #==> syntax error
    if (a==b): print(a,b) #==> 1 1
```

- Boolean operators use English words not, and, or; bitwise operators use symbols ~, &, | respectively. However Python does have some subtleties that can cause unexpected results:
 - o Skipping the parentheses after a function does not invoke a call to the function and will fail
 silently because it's a legitimate reference to the function object:
 class a:
 def demo():
 print("in demo")
 a.demo() #=> in demo
 a.demo #=> <function demo at 0x0000000342A9C8>
 x = a.demo
 x() #=> in demo

The two lines that reference the function without trailing parentheses above demonstrate how that syntax is a reference to the function *object* and not a call to the function.

• Built-in functions that perform in-place operations on mutable objects (that is, lists, dictionaries, and some class instances) do not return the changed object – they return None:

© ISO/IEC 2015 - All rights reserved

Deleted: 6 Deleted: E.27

Comment [SGM10]: Email from Nick Coghlan (2017-09-21) - async/await syntax introduces another opportunity for a "likely incorrect expression", which is to forget to await a coroutine – see https://github.com/python-trio/trio/issues/79 for discussion (it does cause a "Coroutine was never awaited" runtime warning)

Deleted: 6 Deleted: E.27

Deleted: 3

```
a = []
a.append("x")
print(a) #=> ['x']
a = a.append("y")
print(a) #=> None
```

6.25,2 Guidance to language users

- Be sure to add parentheses after a function call in order to invoke the function; and
- Keep in mind that any function that changes a mutable object in place returns a None object not the changed object since there is no need to return an object because the object has been changed by the function.

6.26 Dead and Deactivated Code [XYQ]

6.26,1 Applicability to language

There are many ways to have dead or deactivated code occur in a program and Python is no different in that regard. Further, Python does not provide static analysis to detect such code nor does the very dynamic design of Python's language lend itself to such analysis.

The module and related import statement provides convenient ways to group attributes (for example, functions, names, and classes) into a file which can then be copied, in whole, or in part (using the from statement), into another Python module. All of the attributes of a module are copied when either of the following forms of the import statement is used. This is roughly equivalent to simply copying in all of code directly into the importing program which can result in code that is never invoked (for example, functions which are never called and hence "dead"):

import modulename
from modulename import *

The import statement in Python loads a module into memory, compiles it into byte code, and then executes it. Subsequent executions of an import for that same module are ignored by Python and have no effect on the program whatsoever. The reload statement is required to force a module, and its attributes, to be loaded, compiled, and executed.

6.26,2 Guidance to language users

- $\bullet \quad \text{Import just the attributes that are required by using the from statement to avoid adding dead code; and \\$
- Be aware that subsequent imports have no effect; use the reload statement instead if a fresh copy of the module is desired.

	Deleted:	6	
••••	Dolotody	E	

ſ	Deleted: 7
ſ	Deleted: E.28
ſ	Deleted: 7
ſ	Deleted: E.28

Deleted: 7

Deleted: 3

25

© ISO/IEC 2015 – All rights reserved

6.27, Switch Statements and Static Analysis [CLL]

6.27,1 Applicability to language

By design Python does not have a switch statement nor does it have the concept of labels or branching to a demarcated "place". Python enforces structure by not providing these constructs but it also provides several statements to select actions to perform based on the value of a variable or expression. The first of these are the if/elif/else statements which operate as they do in other languages so this warrants no further coverage here.

Python provides a break statement which allows a loop to be broken with an immediate branch to the first statement after the loop body:

```
a = 1
while True:
    if a > 3:
        break
    else:
        print(a)
        a += 1
```

The loop above prints 1, 2 and 3, each on separate lines, then terminates upon execution of the break statement.

6.27,2 Guidance to language users

Use if/else statements to provide the equivalent of switch statements.

6.28 Demarcation of Control Flow [EOJ]

6.28,1 Applicability to language

Python makes demarcation of control flow very clear because it uses indentation (using spaces or tabs – but not both) and undentation as the *only* demarcation construct:

a, b = 1, 1
if a:
 print("a is True")
else:
 print("False")
 if b:
 print("b is true")
print("back to main level")

The code above prints "a is True" followed by "back to main level". Note how control is passed from the first if statement's True path to the main level based entirely on indentation while in most other languages the final line would execute only when the second if evaluated to True.

Deleted: 8 Deleted: E.29 Deleted: 8 Deleted: E.29

Deleted:	3

Deleted: 8 Deleted: E.29

Deleted: 9

Deleted: 9

Deleted: E.30

'undentatior

Deleted: de

Deleted: se

compile-time error

Comment [SGM11]: Email from Nick Coghlan (20170921) - Python 3 makes mixing tabs and spaces for indentation a

Comment [SM12]: Check - is it "dendentation" or

26

© ISO/IEC 2015 - All rights reserved

6.28,2 Guidance to language users Deleted: 9 Deleted: E.30 Use only spaces or tabs, not both, to indent to demark control flow. 6.29 Loop Control Variables [TEX] Deleted: 30 Deleted: E.31 6.29,1 Applicability to language Comment [SGM13]: Email from Nick Coghlan (2017-09-21) - in Python 2, a particularly problematic case of loop control variables leaking is in list comprehensions. In Python 3, Python provides two loop control statements: while and for. They each support very flexible control comprehensions use their own scope, so the loop variable doesn't leak anymore constructs beyond a simple loop control variable. Assignments in the loop control statement (that is, while or Deleted: 30 for) which can be a frequent source of problems, are not allowed in Python - Python's loop control statements Deleted: E.31 use expressions which cannot contain assignment statements.

The while statement leaves the loop control entirely up to the programmer as in the example below:

```
a = 1
while a:
    print('in loop')
    a = False # force loop to end after one iteration
else:
    print('exiting loop')
```

The for statement is unusual in that it does not provide a loop control variable therefore it is not possible to vary the sequence or number of iterations that are performed other than by the use of the break statement (covered in 6.28 Demarcation of Control Flow [EOJ]) which can be used to immediately branch to the statement after the loop block.

When using the for statement to iterate though an iterable object such as a list, there is no way to influence the loop "count" because it's not exposed. The variable a in the example below takes on the value of the first, then the second, then the third member of the list:

```
x = ['a', 'b', 'c']
for a in x:
    print(a)
#=>a
#=>b
#=>c
```

©

It is possible, though not recommended, to change a mutable object as it is being traversed which in turn changes the number of iteratons performed. In the case below the loop is performed only two times instead of the three times had the list been left intact:

x = ['a', 'b', 'c']		
for a in x:		
print(a)		 Formatted: Spanish
del x[0]		
print(x)		
#=> a	•	 Deleted: 3
SO/IEC 2015 - All rights reserved	27	

Deleted: F 29

#=> c #=> ['c']			
6.29.2 Guidance to language users			Deleted: 30
 Be careful to only modify loop control variables in ways that are easily understood and in ways cannot lead to a premature exit or an endless loop. When using the for statement to iterate through a mutable object, do not add or delete men because it could have unexpected results. 	that nbers	{	Deleted: E.31
6.30 Off-by-one Error [XZH]			Deleted: 1
			Deleted: E.32
<u>6.30</u> ,1 Applicability to language			Deleted: 1
The Python language itself is vulnerable to off by one errors as is any language when used carelessly of person not familiar with Python's index from zero versus from one. Python does not prevent off by one its runtime bounds checking for strings and lists does lessen the chances that doing so will cause harm not possible to index past the end or beginning of a string or list by being off by one because Python does a sentinel character and it always checks indexes before attempting to index into strings and lists and it exception when their bounds are exceeded.	r by a e errors but . It is also pes not use raises an	-1	Deleted: E.32
6.30.2 Guidance to language users			Deleted: 1
		·····(Deleted: E.32
Be aware of Python's indexing from zero and code accordingly.			
6.31,Structured Programming [EWD]		_	Deleted: 2
			Deleted: E.33
6.31,1 Applicability to language Python is designed to make it simpler to write structured program by requiring indentation and deden	tation to	$\left\langle \right\rangle$	Comment [SGM14]: Email from Nick Coghlan (2017-09-21) - for structured programming, the use of with statements and context managers may be preferable to ad hoc try/except and try/finally statements
show scope of control in blocks of code:			Deleted: 2
<pre>a = 1 b = 1 if a == b: print("a == b") #=> a == b if a > b: print("a > b") else: print("a != b")</pre>		l	Dereted. E.33
In many languages the last print statement would be executed because they associate the else wi immediately prior if while Python uses indentation to link the else with its associated if statemen the one <i>above</i> it).	th the t (that is <i>,</i>		
Python also encourages structured programming by <i>not</i> introducing any language constructs which co unstructured code (for example, GO TO statements).	uld lead to	/	Deleted: 3
28 © ISO/IEC 2015- All rit	shts reserved		

Python does have two statements that could be viewed as unstructured. The first is the break statement. It's used in a loop to exit the loop and continue with the first statement that follows the last statement within the loop block. This is a type of branch but it is such a useful construct that few would consider it "unstructured" or a bad coding practice.

The second is the try/except block which is used to trap and process exceptions. When an exception is thrown a branch is made to the except block:

```
def divider(a,b):
    return a/b
try:
    print(divider(1,0))
except ZeroDivisionError:
    print('division by zero attempted')
```

6.31,2 Guidance to language users

 Python offers few constructs that could lead to unstructured code. However, judicious use of break statements is encouraged to avoid confusion.

6.32, Passing Parameters and Return Values [CSJ]

Python's only subprogram type is the function. Even though the import statement does execute the imported module's top level code (the first time it is imported), the import statement cannot effectively be used as a way to repeatedly execute a series of statements

Python passes arguments by assignment which is similar to passing by pointer or reference. Python assigns the passed arguments to the function's local variables but unlike some other languages, simply having the address of the caller's argument does not automatically allow the called function to change any of the objects referenced by those arguments – only *mutable* objects referenced by passed arguments can be changed. Python has no concept of aliasing where a function's variables are mapped to the caller's variables such that any changes made to the function's variables are mapped over to the memory location of the caller's arguments.

```
a = 1
def f(x):
    x += 1
    print(x) #=> 2
f(a)
print(a) #=> 1
```

In the example above, an immutable integer is passed as an argument and the function's local variable is updated and then discarded when the function goes out of scope therefore the object the caller's argument references is not affected. In the example below, the argument is mutable and is therefore updated in place:

a = [1] def f(x):

© ISO/IEC 2015 - All rights reserved

Deleted: 2 Deleted: E.33

Deleted: 3

-{	Deleted: 3
{	Deleted: E.34
-{	Deleted: 3
{	Deleted: E.34

```
x[0] = 2
f(a)
print(a) #=> [2]
```

Note that the list object a is not changed – it's the same object but its content at index 0 has changed.

The return statement can be used to return a value for a function:

```
def doubler(x):
    return x * 2
x = 1
x = doubler(x)
print(x) #=> 2
```

The example above also demonstrates a way to emulate a call by reference by assigning the returned object to the passed argument. This is not a true call by reference and Python does not replace the value of the object x, rather it creates a new object x and assigns it the value returned from the doubler function as proven by the code below which displays the address of the initial and the new object x:

```
def doubler(x):
    return x * 2
x = 1
print(id(x)) #=> 506081728
x = doubler(x)
print(id(x)) #=> 506081760
```

The object replacement process demonstrated above follows Python's normal processing of *any* statement which changes the value of an immutable object and is not a special exception for function returns.

Note that Python functions return a value of none when no return statement is executed or when a return with no arguments is executed.

6.32,2 Guidance to language users

- Create copies of mutable objects before calling a function if changes are not wanted to mutable arguments; and
- If a function wants to ensure that it does not change mutable arguments it can make copies of those
 arguments and operate on them instead.

6.33 Dangling References to Stack Frames [DCM]

This vulnerability is not applicable to Python because, while Python does provide a way to inspect the address of an object, for example, the *id* function, it does not provide a way to use that address to access an object.

Formatted: French

Comment [SGM15]: This section needs a rewrite to acknowledge the vulnerability. Email from Nick Coghlan (2017-09-21)

 Initial work togothal (2016) 21/1 preferences to stack frames reminded me that if you want to write robust, secure, and reliable code, don't use the ctypes module (since that "does" let you access arbitrary memory addresses). (off is a safer third party alternative, since it will read C header files and generate

safe(r) Python wrappers than direct C ABI access with ctypes.

Deleted: 4 Deleted: E.35

Deleted: 3

Deleted: E.34

Deleted: 3

30

© ISO/IEC 2015 - All rights reserved
6.24 Suburguam Signatura Miamatah [OTD]	1	đ	Deleted: 5
<u>0.34</u> Subprogram Signature Mismatch [01K]			Deleted: F 36
6.24 1 Applicability to language			Deleted: 5
0.34,1 Applicability to language			Deleted: E.36
Python supports positional, "keyword=value", or both kinds of arguments. It also supports variable numbers of		, i	
arguments and, other than the case of variable arguments, will check at runtime for the correct number of			
arguments making it impossible to corrupt the call stack in Python when using standard modules.			
, , , , , , , , , , , , , , , , , , ,			
Python has extensive extension and embedding APIs that includes functions and classes to use when extending of	or		
embedding Python. These provide for subprogram signature checking at runtime for modules coded in non-			
Python languages. Discussion of this API is beyond the scope of this annex but the reader should be aware that			
improper coding of any non-Python modules or their interface could cause a call stack problem			
<u>6.34</u> ,2 Guidance to language users			Deleted: 5
			Deleted: E.36
Apply the guidance described in <u>TR 24772-1 clause</u> 6.34.5.			Deleted: 6
		-1	Dalatadı (
<u>0.35</u> Recursion [GDL]	· · · · ·		Deleted: F 37
6 25 1 Applicability to language			Deleted: 6
0.53,1 Applicability to language			Deleted: E 37
Recursion is supported in Python and is by default, limited to a denth of 1,000 which can be overridden using th		l	
set recursion limit function. If the limit is set high enough, a runaway recursion could exhaust all memory	,		
resources leading to a denial of service			
6.35,2 Guidance to language users			Deleted: 6
			Deleted: E.37
Apply the guidance described in <u>TR 24772-1 clause</u> 6.35,5			Deleted: 7
		1	Polytoda z
6.36 Ignored Error Status and Unnandled Exceptions [UYB]			Deleted: 7
() () Annelise billion to long one of			Deleted: E.38
<u>0.30</u> , 1 Applicability to language			Deleted: 7
Python provides statements to handle exceptions which considerably simplify the detection and handling of	1	l	Deletedi E30
excentions. Bother than being a vulnerability. Bython's excention bandling statements provide a way to foil denis			
of service attacks.			
<pre>def mainpgm(x, y):</pre>			
return x/y			
for x in range(3):			
try:			
y = mainpgm(1,x)			
except:			
print('Problem in mainpgm')			
# crean up code			
print (v)			
		Å	Deleted: 3
		1	
	1	1	
© ISO/IEC 2015- All rights reserved 31	1./		

The example code above prints:

Problem in mainpgm 1.0 0.5

The idea above is to ensure that the main program, which could be a web server, is allowed to continue to run after an exception by virtue of the try/except statement pair.

6.36,2 Guidance to language users

- Use Python's exception handling with care in order to not catch errors that are intended for other exception handlers; and
- Use exception handling, but directed to specific tolerable exceptions, to ensure that crucial processes can continue to run even after certain exceptions are raised.

6.37, Type-breaking Reinterpretation of Data [AMV]

This vulnerability is not applicable to Python because assignments are made to objects and the object always holds the type – not the variable, therefore all referenced objects has the same type and there is no way to have more than one type for any given object.

6.38 Deep vs. Shallow Copying [YAN]

6.38.1 Ar	plicability	to	language	

TBD

6.38,2 Guidance to language users

TBD

6.39 Memory Leaks and Heap Fragmentation [XYL]

6.39,1 Applicability to language

Python supports automatic garbage collection so in theory it should not have memory leaks. However, there are at least three general cases in which memory can be retained after it is no longer needed. The first is when implementation-dependent memory allocation/de-allocation algorithms (or even bugs) cause a leak – this is beyond the scope of this annex. The second general case is when objects remain referenced after they are no longer needed. This is a logic error which requires the programmer to modify the code to delete references to objects when they are no longer required.

There is a third very subtle memory leak case wherein objects mutually reference one another without any outside references remaining – a kind of deadly embrace where one object references a second object (or group of objects) so the second object(s) can't be collected but the second object(s) also reference the first one(s) so it/they too can't be collected. This group is known as cyclic garbage. Python provides a garbage collection

© ISO/IEC 2015 – All rights reserved

Deleted: 7	
Deleted: E.38	

Deleted: 6.38E.39 Termination Strategy [REU]	l	[7]	
Deleted: 9			
Bulutuda Bula			

Comment [SGM16]: Comment from Nick Coghlan: For shallow copying: we don't detect or prevent it, but reference counting at least ensures the references copied that way remain alive. (Hmm, that does prompt a thought though: memoryview and the PEP 3118 buffer protocol do create some interesting new

the PEP 3118 buffer protocol do create some interesting new issues, since the obligation is on the buffer publisher to ensure that the memory remains valid at least as long as the object lives, while buffer consumers need to make sure they keep an active reference to the publisher)

Formatted: Normal, Level 1	
Deleted: 9	
Formatted: Normal, Level 1	
Deleted: 40	
Deleted: E.41	
Deleted: 40	
Deleted: E.41	

Deleted: 3

module called gc which has functions which enable the programmer to enable and disable cyclic garbage collection as well as inspect the state of objects tracked by the cyclic garbage collector so that these, often very subtle leaks, can be traced and eliminated.

Release all objects when they are no longer required.		 Deleted: 6.40E.41.2 Guidance to language users .
6.40 Templates and Generics [SYM]		 Deleted: 1
		 Deleted: E.42
This vulnerability is not applicable to Python because Python does not implement these mechanisms.		
6.41 Inheritance [RIP]	Τ.	Deleted: 2
		 Deleted: E.43
6.41,1 Applicability to language		Deleted: 2
Python supports inheritance through a hierarchical search of namespaces starting at the subclass and proceedir upward through the superclasses. Multiple inheritance is also supported. Any inherited methods are subject to the same vulnerabilities that occur whenever using code that is not well understood.	ng	Deleted: E.43
6.41,2 Guidance to language users		Deleted: 2
 Inherit only from trusted classes; and Use Python's built-in documentation (such as docstrings) to obtain information about a class' method before inheriting from it. 	1	Deleted: E.43
6.42 Violations of the Liskov Substitution Principle or the Contract Model [BLP]		Comment [SGM17]: Note from Nick Coghlan: For Liskov/redispatch/oolymorphism. I'm not really the right person
6.42.1 Applicability to language		to ask - the folks working on mypy and other typechecking tools are. Probably the best way to contact them would be to file an issue on https://dithub.com/ovthon/typing/issues asking for their
TBD	*	 feedback.
6.42.2 Guidance to language users		Formatted: Normal, Level 1
722		
	4	Formatted: Normal, Level 1
6.43 Redispatching [PPH]		Comment [SGM18]: Comment from Nick Coghlan: For Liskov/redispatch/polymorphism. I'm not really the right
6.43.1 Applicability to language		person to ask - the folks working on mypy and other typechecking tools are. Probably the best way to contact them would be to file an issue
TBD		on <u>https://github.com/python/typing/issues</u> asking for their feedback.
6.43.2 Guidance to language users		
<u>TBD</u>		
© ISO/IEC 2015- All rights reserved 33	3	Deleted: 3

6.44 Polymorphic variables		Comment [SGM19]: Note from Nick Coghlan:
6.44.1 Applicability to language		For Liskov/redispatch/polymorphism, I'm not really the right person to ask - the folks working on mypy and other typechecking tools are. Prohably the bast way to contact them would be to file an issue
TBD		on <u>https://github.com/python/typing/issues</u> asking for their feedback.
6.44.2 Guidance to language users		
TBD		
6.45, Extra Intrinsics [LRM]		Deleted: 3
(AF 1 Applicability to lap guage		Deleted: E.44
0.45,1 Applicability to language		Deleted: 3
Python provides a set of built-in intrinsics which are implicitly imported into all Python scripts. Any ovariables and functions can therefore easily be overridden:	of the built-in	
x = 'abc'		
<pre>print(len(x))#=> 3</pre>		
<pre>def len(x):</pre>		
return 10 print (len(x)) $\#=>10$		
princ(ren(x))#=> 10		
If the example above the built-in len function is overridden with logic that always returns 10. Note statement is executed dynamically so the new overriding len function has not yet been defined we call to len is made therefore the built-in version of len is called in line 2 and it returns the expecte this case). After the new len function is defined it overrides all references to the builtin-in len fun- script. This can later be "undone" by explicitly importing the built-in len function with the followin	that the def en the first d result (3 in ction in the g code:	
<pre>from builtins import len print(len(x))#=> 3</pre>		
It's very important to be aware of name resolution rules when overriding built-ins (or anything else matter). In the example below, the overriding len function is defined within another function and not found using the LEGB rule for name resolution (see <u>6.21 Namespace Issues [BJL]</u>):	for that herefore is	
x = 'abc'		
print(len(x)) #=> 3		
<pre>def len(x):</pre>		
return 10		
print(len(x)) $\# = > 3$		
6.45,2 Guidance to language users		Deleted: 3
L		Deleted: E.44
Do not override built-in "intrinsics" unless absolutely necessary		Deleted: 3
34 © ISO/IEC 2015	Il rights reserved	

6.46 Argument Passing to Library Functions [TR]]		Deleted: 4
		Deleted: E.45
6.46,1 Applicability to language		Deleted: 4
		Deleted: E.45
Refer to 6.34 Subprogram Signature Mismatch [OTR],		Deleted: E.35 Subprogram Signature Mismatch [OTR]
6.46.2 Guidance to language users		Deleted: 4
		Deleted: E.45
Refer to 6.34 Subprogram Signature Mismatch [OTR],		Deleted: E.36 Subprogram Signature Mismatch [OTR]
6.47, Inter-language Calling [DJS]		Deleted: 5
		Deleted: E.46
6.47,1 Applicability to language		Deleted: 5
Python has a documented API for extending Python using libraries coded in C or C++. The library(s) are then imported into a Python module and used in the same manner as a module written in Python. Python's standard for interfacing to the "C" language is documented in http://docs.python.org/py3k/c-api/ . Conversely, code written in C or C++ can embed Python. The standard for embedding Python is documented in:		Comment [SM20]: Put reference in the bibliography and reference the bibliography (here and 2 lines down).
The Jython system is a Java-based implementation that interfaces with Java and IronPython provides interfaces to Microsoft .NET languages.	,	
6.47,2 Guidance to language users		Deleted: 5
 Use the language interface APIs documented on the Python web site for interfacing to C/C++, the Jython web site for Java, the IronPython web site for .NET languages, and for all other languages consider creating intermediary C or C++ modules to call functions in the other languages since many languages have documented API's to C and C++. 		
6.48 Dynamically-linked Code and Self-modifying Code [NYY]		Deleted: 6
		Deleted: E.47
6.48,1 Applicability to language		Deleted: 6
Python supports dynamic linking by design. The import statement fetches a file (known as a module in Python), compiles it and executes the resultant byte code at run time. This is the normal way in which external logic is made accessible to a Python program therefore Python is inherently exposed to any vulnerabilities that cause a different file to be imported:		Deleted: E.47
 Alteration of a file directory path variable to cause the file search locate a different file first; and Overlaying of a file with an alternate. 		
Python also provides an $\verb+eval$ and an $\verb+exec$ statement each of which can be used to create self-modifying code:		
<pre>x = "print('Hello " + "World')" eval(x) #=> Hello World</pre>		Deletedia
	1	
© ISO/IEC 2015 – All rights received 35		

Guerrilla patching, also known as monkey patching, is a way to dynamically modify a module or class at run-time to extend, or subvert their processing logic and/or attributes. It can be a dangerous practice because once "patched" any other modules or classes that use the modified class or module may unwittingly be using code that does not do what they expect which could cause unexpected results.

6.48.2 Guidance to language users		Deleted: 6
		Deleted: E.47
 Avoid using exec or eval and never use these with untrusted code; Be careful when using Guerrilla patching to ensure that all users of the patched classes and/or modules. 		Comment [SM21]: This may not be dynamically linked code, but the recommendation is good (just maybe elsewhere).
continue to function as expected; conversely, be aware of any code that natches classes and/or modules		
that your code is using to avoid unexpected results; and		
Encure that the file path and files being imported are from tructed sources		
• Lisure that the me path and mes being imported are nonn trusted sources.		
6.49 Library Signature [NSQ]		Deleted: 7
6.49,1 Applicability to language	and the second s	Deleted: 7
Python has an extensive API for extending or embedding Python using modules written in C, Java, and Fortran. Extensions themselves have the potential for vulnerabilities exposed by the language used to code the extension which is beyond the scope of this annex.		Deleted: 1.48
Python does not have a library signature checking mechanism but its API provides functions and classes to help		
ensure that the signature of the extension matches the expected call arguments and types. See 6.34 Subprogram		
Signature Mismatch [OTR]		
<u>signature mismaten [orn]</u> .		
6.49,2 Guidance to language users		Deleted: 7
— ₁		Deleted: E.48
Use only trusted modules as extensions; and		
• If coding an extension utilize Python's extension API to ensure a correct signature match.		
6.50 Unanticipated Exceptions from Library Routines [HJW]		Deleted: 48
		Deleted: E.49
<u>6.50</u> ,1 Applicability to language		Deleted: 48
		Deleted: E.49
Python is often extended by importing modules coded in Python and other languages. For modules coded in Python the risks include:		
 Interception of an exception that was intended for a module's imported exception handling code (and vice versa): and 		
 Unintended results due to namespace collisions (covered in 6.21 Namespace Issues [BJL] and elsewhere in 		Deleted: E.22
this annex).		
For modules coded in other languages the risks include:		
 Unexpected termination of the program: and 		
Inexpected termination of the program, and		
• Onexpected side energy on the operating environment.		
	1	Deleted: 3
36 © ISO/IEC 2015-All rights reserved		

6 50 2 Guidance to language users	1		Deleted: 48
0.00,2 Outdance to language users			Deleted: E.49
 Wrap calls to library routines and use exception handling logic to intercept and handle exceptions when practicable. 	I	l	
6.51 Pre-processor Directives [NMP]		{	Deleted: 49
—— t		{	Deleted: E.50
This vulnerability is not applicable to Python because Python has no pre-processor directives.			Comment [SGM22]: Email from Nick Coghlan (2017-09-21)
6.52 Suppression of Language-defined Run-time Checking [MXB]			- the "pre-processor directives" section isn't strictly true: "fromfuture import feature" is a compile-time directive, and the
This vulnerability is not applicable to Python because Python does not have a mechanism for suppressing run- time error checking. The only suppression available is the suppression of run-time warnings using the command line –W option which suppresses the printing of warnings but does not affect the execution of the program.			encoding cookie declarations in source headers allow for arbitrary source-source translations when loading source modules. The import hook mechanisms also provide a lot of flexibility for runtime code to change how imports in other parts of the program are actually handled.
(5) Provision of Inhonorthy Uncole Operations [CVI]	1	N	Deleted: 50
6.53 Provision of innerently Unsate Operations [SKL]		1	Deleted: E.51
			Deleted: 51
<u>6.53</u> ,1 Applicability to language		j	Deleted: E.52
Buthen has yong few energians that are inherently uncefe. For example, there is no way to suppress error			Deleted: 1
Python has very lew operations that are innerently unsafe. For example, there is no way to suppress error		ો	Deleted: E.52
any language:			
 Interfaces to modules coded in other languages since they could easily violate the security of the calling c embedded Python code; and 	of		
 Use of the exec and eval dynamic execution functions (see <u>6.48 Dynamically-linked Code and Self-modifying Code [NYY]</u>). 			
6.53,2 Guidance to language users		{	Deleted: 1
\		{	Deleted: E.52
Use only trusted modules; and			
• Avoid the use of the exec and eval functions.			
6 54 Obscure Language Features [BRS]			Deleted: 2
			Deleted: E.53
6 54 1 Applicability of language			Comment [SGM23]: Email from Nick Coghlan (2017-09-21)
Python has some obscure language features as described below:		N	 the asyncio infrastructure has introduced a number of new "obscure language features" for use by event loop implementors (e.g. there's a hook that gets called any time a native coroutine is created)
Functions are defined when executed:			Deleted: 2
a = 1		Ì	Deleted: E.53
while $a < 3$:			
if a == 1:			
def f():			
print("a must equal 1")			
else:			
def f():		Å	Deleted: 3
		\mathbb{Z}	
	1		
© ISO/IEC 2015- All rights reserved 37	/		

```
\label{eq:print("a must not equal 1")} f() \\ a \ += \ 1
```

The function ${\tt f}$ is defined and redefined to result in the output below:

```
a must equal 1
a must not equal 1
```

A function's variables are determined to be local or global using static analysis: if a function only references a variable and never assigns a value to it then it is assumed to be global otherwise it is assumed to be local and is added to the function's namespace. This is covered in some detail in 6.22 Initialization of Variables [LAV],

A function's default arguments are assigned when a function is *defined*, not when it is *executed*:

```
def f(a=1, b=[]):
    print(a, b)
    a += 1
    b.append("x")
f()
f()
f()
```

The output from above is typically expected to be:

1 [] 1 [] 1 []

But instead it prints:

1 [] 1 ['x'] 1 ['x', 'x']

This is because neither a nor b are reassigned when f is *called* with *no* arguments because they were assigned values when the function was *defined*. The local variable a references an immutable object (an integer) so a new object is created when the a += 1 statement is created and the default value for the a argument remains unchanged. The mutable list object b is updated in place and thus "grows" with each new call.

The += Operator does not work as might be expected for mutable objects:

x = 1 $x \neq 1$ print(x) #=> 2 (Works as expected)

But when we perform this with a mutable object:

x = [1, 2, 3]

38

© ISO/IEC 2015 - All rights reserved

Deleted: E.23

Formatted: Spanish

Deleted: 3

```
y = x
print(id(x), id(y))#=> 38879880 38879880
x += [4]
print(id(x), id(y))#=> 38879880 38879880
x = x + [5]
print(id(x), id(y))#=> 48683400 38879880
print(x,y)#=> [1, 2, 3, 4, 5] [1, 2, 3, 4]
```

The += operator changes x in place while the x = x + [5] creates a new list object which, as the example above shows, is not the same list object that y still references. This is Python's normal handling for all assignments (immutable or mutable) – create a new object and assign to it the value created by evaluating the expression on the right hand side (RHS):

```
x = 1
print(id(x)) #=> 506081728
x = x + 1
print(id(x)) #=> 506081760
```

Equality (or equivalence) refers to two or more objects having the same value. It is tested using the == operator which can thought of as the 'is equal to test'. On the other hand, two or more *names* in Python are considered identical only if they reference the same object (in which case they would, of course, be equivalent too). For example:

a = [0,1] b = a c = [0,1] a is b, b is c, a == c #=> (True, False, True)

a~ and b~ are both names that reference the same objects while c~ references a different object which has the same value as both a~ and b.

Python provides built-in classes for persisting objects to external storage for retrieval later. The complete object, *including its methods*, is serialized to a file (or DBMS) and re-instantiated at a later time by any program which has access to that file/DBMS. This has the potential for introducing rogue logic in the form of object methods within a substituted file or DBMS.

Python supports passing parameters by keyword as in:

a = myfunc(x = 1, y = "abc")

This can make the code more readable and allows one to skip parameters. It can also reduce errors caused by confusing the order of parameters.

6.54,2 Guidance to language users

Ensure that a function is defined before attempting to call it; Be aware that a function is defined dynamically so its composition and operation may vary due to variations in the flow of control within the defining program;

© ISO/IEC 2015 - All rights reserved

Formatted:	French
ronnaccean	1 I CHICH

Formatted: Spanish

Deleted: 2

Deleted: 3

39

Deleted: E.53

- Be aware of when a variable is local versus global;
- Do not use mutable objects as default values for arguments in a function definition unless you absolutely need to and you understand the effect;
- Be aware that when using the += operator on mutable objects the operation is done in place;
- Be cognizant that assignments to objects, mutable and immutable, always create a new object;
- Understand the difference between equivalence and equality and code accordingly; and
- Ensure that the file path used to locate a persisted file or DBMS is correct and never ingest objects from an untrusted source.

6.55 Unspecified Behaviour [BQF]

6.55,1 Applicability of language

Understanding how Python manages identities becomes less clear when a script is run using integers (or short strings):

```
a=1
b=a
c=1
a is b, b is c, a == c #=> (True, True, True)
```

In the example above c references the same object as a and b even though c was never assigned to either a or b. This is a nuance of how Python is optimized to cache short strings and small integers. Other than in a test for identity as above, this nuance has no effect on the logic of the program (for example, changing the value of c to 2 will not affect a or b). Refer also to 4. Language concepts,

When persisting objects using pickling, if an exception is raised then an unspecified number of bytes may have already been written to the file.

6.55,2 Guidance to language users

- Do not rely on the content of error messages use exception objects instead;
- When persisting object using pickling use exception handling to cleanup partially written files; and
- Do not depend on the way Python may or may not optimize object references for small integer and string objects because it may vary for environments or even for releases in the same environment.

6.56, Undefined Behaviour [EWF]

6.56,1 Applicability to language

Python has undefined behaviour in the following instances:

 Caching of immutable objects can result in (or not result in) a single object being referenced by two or more variables. Comparing the variables for equivalence (that is, if a == b) will always yield a True but checking for equality (using the is built-in) may, or may not, dependent on the implementation: a = 1

Deleted: 3

Deleted: 3 Deleted: E.54

Deleted: 3 Deleted: E.54

Deleted: E.2.2 Key Concepts

Deleted: 3

Deleted: 4 Deleted: E.55

Deleted: 4 Deleted: E.55

Deleted: E.54

40

© ISO/IEC 2015 – All rights reserved

b = 2-1

print(a == b, a is b) #=> (True, ?)

- The sequence of keys in a dictionary is undefined because the hashing function used to index the keys is
 unspecified therefore different implementations are likely to yield different sequences.
- The Future class encapsulates the asynchronous execution of a callable. The behaviour is undefined if the add_done_callback(fn) method (which attaches the callable fn to the future) raises a BaseException subclass.
- Modifying the dictionary returned by the vars built-in has undefined effects when used to retrieve the dictionary (that is, the namespace) for an object.
- Form feed characters used for indentation have an undefined effect on the character count used to
 determine the scope of a block.
- The catch_warnings function in the context manager can be used to temporarily suppress warning
 messages but it can only be guaranteed in a single-threaded application otherwise, when two or more
 threads are active, the behaviour is undefined.
- When sorting a list using the sort() method, attempting to inspect or mutate the content of the list will
 result in undefined behaviour.
- The order of sort of a list of sets, using list.sort(), is undefined as is the use of the function used on a list of sets that depend on total ordering such as min(), max(), and sorted().
- Undefined behaviour will occur if a thread exits before the main procedure from which it was called itself exits.

6.56,2 Guidance to language users

- Understand the difference between testing for equivalence (for example, ==) and equality (for example, is) and never depend on object identity tests to pass or fail when the variables reference immutable objects;
- Do not depend on the sequence of keys in a dictionary to be consistent across implementations.
- When launching parallel tasks don't raise a BaseException subclass in a callable in the Future class;
- Never modify the dictionary object returned by a vars call;
- Never use form feed characters for indentation;
- Consider using the id function to test for object equality;
- Do not try to use the catch_warnings function to suppress warning messages when using more than
 one thread; and
- Never inspect or change the content of a list when sorting a list using the sort () method.

6.57 Implementation-defined Behaviour [FAB]

6.57,1 Applicability to language

Python has implementation-defined behaviour in the following instances:

- Mixing tabs and spaces to indent is defined differently for UNIX and non-UNIX platforms;
- Byte order (little endian or big endian) varies by platform;
- Exit return codes are handled differently by different operating systems;

© ISO/IEC 2015- All rights reserved

Deleted: 3

41

Deleted: 5 Deleted: E.56 Deleted: 5

Deleted: E.56

Deleted: 4

Deleted: E.55

- The characteristics, such as the maximum number of decimal digits that can be represented, vary by platform;
- The filename encoding used to translate Unicode names into the platform's filenames varies by platform; and
- Python supports integers whose size is limited only by the memory available. Extensive arithmetic using integers larger than the largest integer supported in the language used to implement Python will degrade performance so it may be useful to know the integer size of the implementation.

6.57,2 Guidance to language users

- Always use either spaces or tabs (but not both) for indentations;
- Consider using the -tt command line option to raise an IndentationError;
- Consider using a text editor to find and make consistent, the use of tabs and spaces for indentation;
- Either avoid logic that depends on byte order or use the sys.byteorder variable and write the logic to account for byte order dependent on its value ('little' or 'big').
- Use zero (the default exit code for Python) for successful execution and consider adding logic to vary the exit code according to the platform as obtained from sys.platform (such as, 'win32', 'darwin', or other).
- Interrogate the sys.float.info system variable to obtain platform specific attributes and code according to those constraints.
- Call the sys.getfilesystemcoding() function to return the name of the encoding system used.
- When high performance is dependent on knowing the range of integer numbers that can be used without degrading performance use the sys.int_info struct sequence to obtain the number of bits per digit (bits_per_digit) and the number of bytes used to represent a digit (sizeof_digit).

6.58 Deprecated Language Features [MEM]

6.58,1 Applicability to language

The following features were deprecated in the latest (as of this writing) version of E 3.1. These are documented at http://docs.python.org/release/3.1.3/whatsnew/3.1.html

- The string.maketrans() function is deprecated and is replaced by new static methods, bytes.maketrans() and bytearray.maketrans(). This change solves the confusion around which types were supported by the string module. Now, str, bytes, and bytearray each have their own maketrans and translate methods with intermediate translation tables of the appropriate type.
- The syntax of the with statement now allows multiple context managers in a single statement:

```
with open('mylog.txt') as infile, open('a.out', 'w') as outfile:
    for line in infile:
        if '<critical>' in line:
            outfile.write(line)
```

• With the new syntax, the <code>contextlib.nested()</code> function is no longer needed and is now deprecated.

Deleted: 3

© ISO/IEC 2015 – All rights reserved

Deleted: 5

Deleted: 6 Deleted: E.57

Deleted: 6 Deleted: E.57

bibliography

Comment [SM24]: Put in bibliography and reference

Deleted: E.56

42

- Deprecated PyNumber_Int(). Use PyNumber_Long() instead.
- Added a new PyOS_string_to_double() function to replace the deprecated functions PyOS_ascii_strtod() and PyOS_ascii_atof().
- Added PyCapsule as a replacement for the PyCObject API. The principal difference is that the new type has a well defined interface for passing typing safety information and a less complicated signature for calling a destructor. The old type had a problematic API and is now deprecated.

6.58,2 Guidance to language users

• When practicable, migrate Python programs to the current standard.

6.59 Concurrency – Activation [CGA]

6.59.1 Applicability to language

TBW: Analyze the standard Python libraries:

<u>threading</u> Reference implementation seems to always raise an exception if <u>start()</u> method is not able to create the thread, but is not documented in the specification and thus the user cannot rely on this. Furthermore, even if the standard library / OS can create the new thread, it can die during the initialization phase when executing the user's code, Method join() does not return if the thread died through an unhandled exception? (Method is <u>alive()</u> to check whether is still running, and timeouts for lock objects. Timer object TBA
 <u>multiprocessing</u>: Exception raised if not activated? TBA

concurrency.futures:TBA

6.59.2 Guidance to language users

TBW

<u>6.60 Concurrency – Directed termination [CGT]</u>

6.60.1 Applicability to language

TBW: Analyze the standard Python libraries:

 threading: No mechanism to abort another thread, the thread has to terminate itself. Alien threads cannot be terminated nor joined.

multiprocessing: TBA

concurrency.futures:TBA

6.60.2 Guidance to language users

TBW:

© ISO/IEC 2015 - All rights reserved

Deleted: 7 Deleted: E.57

43

Deleted: Formatted: Heading 3 Formatted: Highlight Formatted: Normal, Level 1 Formatted: Font:(Default) Courier New, English (UK), Kern at 14 pt, Highlight Formatted: List Paragraph, Space After: 6 pt, Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm, No widow/orphan control, Suppress line numbers, Don't allow hanging punctuation Formatted: Highlight Formatted: Font:(Default) Courier New, English (UK), Kern at 14 pt, Highlight Formatted: Highlight Formatted: Highlight Formatted: Highlight Formatted: Font:(Default) Courier New, English (UK), Kern at 14 pt, Highlight Formatted: Highlight Formatted: Font: (Default) Calibri, Highlight Formatted: Font:(Default) Courier New, English (UK), Kern at 14 pt, Highlight Formatted: Font:(Default) Calibri, Highlight Formatted: Font:(Default) Courier New, English (UK), Kern at 14 pt, Highlight Formatted: Highlight Formatted: Normal, Level 1 Deleted: Deleted: Formatted: Heading 3 Formatted: Normal Formatted: Normal, Level 1 Deleted: 3

6.61 Concurrent Data Access [CGX]

((1.1 Ampliashility to language			Delated
6.61.1 Applicability to language	*****	\leq	Formatted: Heading 3
TBW: Analyze the standard Python libraries:		(
 threading: Different mechanisms TBA:: Lock, RLock (recursive lock), Sem 	aphore, Condition, Event,		
Barrier. Ose with statement with locks			
 concurrency futures: TRA 			
	••••••••••••••••••••••••••••••••••••	l	Formatted: Normal
6.61.2 Guidance to language users			
		(
1BW	4	l	Formatted: Normal, Level 1
threading: Use 'with statement' with locks			
 multiprocessing: TBA 			
concurrency.futures:TBA			
	4		Formatted: Normal
		ſ	
<u>6.62 Concurrency – Premature Termination [CGS]</u>			Deleted:
6.62.1 Applicability to language			Deleted:
			Formatted: Heading 3
TBW: Analyze the standard Python libraries:			
• threading TBA			
• multiprocessing:TBA			
 concurrency.futures:TBA 			
	4		Formatted: Normal
		U	
6.62.2 Guidance to language users			
TRM		(Formatted: Normal Level 1
		l	
6.63 Protocol Lock Errors [CGM]		\leq	Deleted: 0
(C21 Applicability to longuage			Deleted: 3
6.03.1 Applicability to language	>	\leq	Formatted: Heading 3
TBW: Analyze the standard Python libraries:		(
• threading: Use with statement with locks			
Concurrency futures TRA			
		S	Deleted: 3
44	© ISO/IEC 2015 - All rights reserved	1	
	A BOTTO LOLA AN INTERIO ICALIVED		

	.	{	Formatted: Normal
		1	Deleted: 0
<u>6.63,2 Guidance to language users</u>		7	Formatted: Highlight
	1		Formatted: Normal, Level 1
TEW	/		Formatted: Normal, Level 1
			Deleted:
threading: IBA			Formatted: Normal
 multiprocessing:TBA 			Deleted:
concurrency.futures:TBA			Comment [SGM25]: Note from Nick Coghlan: Speaking of clocks & timing, there are some use cases that
6.64 Reliance on External Format String [SHL]	- / //		should be updated to use time.monotonic() rather than
6.64.1 Applicability to language	11		https://www.python.org/dev/peps/pep-0418/#time-monotonic
TBD	-		Windows applications should also be aware of the fact that Python 3.6 always uses utf-8 for binary filesystem and console interfaces:
6.64.2 Guidance to language users	\mathbb{N}		https://docs.python.org/dev/whatsnew/3.6.html#pep-529- change-windows-filesystem-encoding-to-utf-8
TBD	-		Non-Windows applications should be aware of the fact that Python 3.7+
			will attempt to coerce the C locale to C.UTF-8 (or an equivalent
τ	1		option
			for redistributor's Python 3.6 implementations (e.g. the system Python
7. Language specific vulnerabilities for Python	•		in Fedora implements the option). https://www.python.org/dev/peps/pep-0538/ has the details of
	and the second		that.
¥	1 🔊	\setminus	
	and the second	Y	Comment [SGM26R25]:
	N	\setminus	
8. Implications for standardization or future revision		Y	Formattadi Normal
		}	Formatted: Highlight
Future standardization efforts should consider the following items to address vulnerability issues identified earlie		A	Formatted: Not Highlight
in this Technical Report.		4	Formatted: Not Highlight
			Formatted: Not Highlight
Inis is a dummy clauon with the Word bibliography feature [2], and the following one using bookmars [1].			Formatted: Not Highlight
	1	\searrow	Field Cade Channed
Υ	and the second second		Polotodi Dogo Brook
Bibliography		\}	Exempting Coase Afters 6 pt Ne widew/emban control
			Suppress line numbers, Don't allow hanging punctuation
[1] ISU/IEC Directives, Part 2, Rules for the structure and drafting of International Standards, 2004		1	Deleted: 2
[2] ISO/IEC TR 10000-1. Information technology — Framework and taxonomy of International Standardized			Deleted: 3
Profiles — Part 1: General principles and documentation framework	· · · ·	$\langle \rangle$	Deleted: [4] ISO/IEC 9899:2011. Information technology —
rights - ruit 1. General principles and accumentation framework	1	/	Programming languages — C
[3] ISO 10241 (all parts). International terminology standards	11		[26] <i>ARIANE 5: Flight 501 Failure</i> , Report by the Inquiry Board,
• Terrent Stand S	1		<u>1819eng.pdf</u> - [9]
[4] Steve Christy, Vulnerability Type Distributions in CVE, V1.0, 2006/10/04	1		Deleted: 25
			Deleted: [26] . ARIANE 5: Flight 501 Failure, Report by t
[5] Carlo Ghezzi and Mehdi Jazayeri, Programming Language Concepts, 3 rd edition, ISBN-0-471-10426-4, John			Deleted: 28
Wiley & Sons, 1998		A	Deleted: 3
	1	/)
	1		
© ISO/IEC 2015- All rights reserved 45			

 John David N. Dionisio. Type Checking. http://myweb.lmu.edu/dondi/share/pl/type-checking-v02.pdf John David N. Dionisio. Type Checking. http://myweb.lmu.edu/dondi/share/pl/type-checking-v02.pdf John David N. Dionisio. Type Checking. http://cwe.mitre.org/). The Common Weakness Enumeration (CWE) Initiative, MITRE Corporation, (http://cwe.mitre.org/). Goldberg, David, What Every Computer Scientist Should Know About Floating-Point Arithmetic, ACM Goldberg, David, What Every Computer Scientist Should Know About Floating-Point Arithmetic, ACM IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard Robert W. Sebesta, Concepts of Programming Languages, 8th edition, ISBN-13: 978-0-321-49362-0, ISBN-11:8). Bo Einarsson, ed. Accuracy and Reliability in Scientific Computing, SIAM, July 2005 http://www.nsc.liu.se/wg25/book 	eport. :e Study Use of IIRA 906400
 [7] The Common Weakness Enumeration (CWE) Initiative, MITRE Corporation, (http://cwe.mitre.org/), [8] Goldberg, David, What Every Computer Scientist Should Know About Floating-Point Arithmetic, ACM [8] Goldberg, David, What Every Computer Scientist Should Know About Floating-Point Arithmetic, ACM [9] IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard [9] IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard [9] Robert W. Sebesta, Concepts of Programming Languages, 8th edition, ISBN-13: 978-0-321-49362-0, ISBN-10: 0-321-49362-1, Pearson Education, Boston, MA, 2008 [11] Bo Einarsson, ed. Accuracy and Reliability in Scientific Computing, SIAM, July 2005 http://www.nsc.liu.se/wg25/book 	use of IIRA 906400
 [7] The Common Weakness Enumeration (CWE) Initiative, MITRE Corporation, (http://cwe.mitre.org/). [8] Goldberg, David, What Every Computer Scientist Should Know About Floating-Point Arithmetic, ACM [8] Goldberg, David, What Every Computer Scientist Should Know About Floating-Point Arithmetic, ACM [9] IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard [9] IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard [9] Robert W. Sebesta, Concepts of Programming Languages, 8th edition, ISBN-13: 978-0-321-49362-0, ISBN-10: 0-321-49362-1, Pearson Education, Boston, MA, 2008 [11] Bo Einarsson, ed. Accuracy and Reliability in Scientific Computing, SIAM, July 2005 http://www.nsc.liu.se/wg25/book 	Use of IIRA 906400
 [8] Goldberg, David, What Every Computer Scientist Should Know About Floating-Point Arithmetic, ACM [8] Goldberg, David, What Every Computer Scientist Should Know About Floating-Point Arithmetic, ACM [9] IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard [9] IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard [9] Robert W. Sebesta, Concepts of Programming Languages, 8th edition, ISBN-13: 978-0-321-49362-0, ISBN-10: 0-321-49362-1, Pearson Education, Boston, MA, 2008 [11] Bo Einarsson, ed. Accuracy and Reliability in Scientific Computing, SIAM, July 2005 http://www.nsc.liu.se/wg25/book 	IIRA L-906400 nes for IRA 906400
11-8) Computing Surveys, vol 23, issue 1 (March 1991), ISSN 0360-0300, pp 5-48. [9] IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008. Institute of Electrical and Electronics Engineers, New York, 2008. [10] Robert W. Sebesta, Concepts of Programming Languages, 8 th edition, ISBN-13: 978-0-321-49362-0, ISBN-10: 0-321-49362-1, Pearson Education, Boston, MA, 2008 [11] Bo Einarsson, ed. Accuracy and Reliability in Scientific Computing, SIAM, July 2005 http://www.nsc.liu.se/wg25/book Deleted: 35	nes for 1RA 906400
[9] IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-2008. Institute of Electrical and Electronics Engineers, New York, 2008. [10] Robert W. Sebesta, Concepts of Programming Languages, 8 th edition, ISBN-13: 978-0-321-49362-0, ISBN- 10: 0-321-49362-1, Pearson Education, Boston, MA, 2008 [11] Bo Einarsson, ed. Accuracy and Reliability in Scientific Computing, SIAM, July 2005 http://www.nsc.liu.se/wg25/book Deleted: 35	nes for IIRA 906400
[9] IEEE Standards Committee 754. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard Formatted: English (US) 754-2008. Institute of Electrical and Electronics Engineers, New York, 2008. Deleted: [32] . MISRA Limited. "MISRA C: 2012 Guidelithe C Language in Critical Systems." Warwickshire, UK: N Limited, March 2013 (ISBN 978-1-906400-10-1 and 978-1-9064	nes for IIRA I-906400
754-2008. Institute of Electrical and Electronics Engineers, New York, 2008. 101 Robert W. Sebesta, Concepts of Programming Languages, 8 th edition, ISBN-13: 978-0-321-49362-0, ISBN- 10: 0-321-49362-1, Pearson Education, Boston, MA, 2008 111 Bo Einarsson, ed. Accuracy and Reliability in Scientific Computing, SIAM, July 2005 http://www.nsc.liu.se/wg25/book	nes for IIRA 906400
[10] Robert W. Sebesta, Concepts of Programming Languages, 8 edition, ISBN-13: 978-0-321-49362-0, ISBN- 10: 0-321-49362-1, Pearson Education, Boston, MA, 2008 Deleted: 33 [11] Bo Einarsson, ed. Accuracy and Reliability in Scientific Computing, SIAM, July 2005 Deleted: 34 http://www.nsc.liu.se/wg25/book Deleted: 35	
10: 0-321-49362-1, Pearson Education, Boston, MA, 2008 Deleted: 33 [11] Bo Einarsson, ed. Accuracy and Reliability in Scientific Computing, SIAM, July 2005 Deleted: 34 http://www.nsc.liu.se/wg25/book Deleted: 35	
[11] Bo Einarsson, ed. Accuracy and Reliability in Scientific Computing, SIAM, July 2005 Deleted: 34 http://www.nsc.liu.se/wg25/book Deleted: 35	
http://www.nsc.liu.se/wg25/book	
Deleted: 20	
Detect: 30	
[1] "Enums for Python (Python recipe)," [Online]. Available: http://code.activestate.com/recipes/67107/.	
rormatted: rreach	
[2] M. Pilgrim, Dive Into Python, 2004.	
[3] M. Lutz, Learning Python, Sebastopol, CA: O'Reilly Media, Inc, 2009.	
[4] "The Python Language Reference." [Online]. Available:	
http://docs.python.org/reference/index.html#reference-index.	
[5] A. Martelli, Python in a Nutshell, Sebastopol, CA: O'Reilly Media, Inc., 2006.	
[6] M. Lutz, Programming Python, Sebastopol, CA: O'Reilly Media, Inc., 2011.	
[7] A. G. Isaac, "Python Introduction," 23 06 2010. [Online]. Available:	
https://subversion.american.edu/aisaac/notes/python4class.xhtml#introduction-to-the-interpreter.	
[Accessed 12 05 2011].	
[8] H. Norwak, "10 Python Pitfalls," [Online]. Available:	
http://zephyrfalcon.org/labs/python_pitfalls.html. [Accessed 13 05 2011].	
[9] "Python Gotchas," [Online]. Available: http://www.ferg.org/projects/python_gotchas.html.	
Formatted: French	
[10] G. source, "Big List of Portability in Python," [Online]. Available:	
http://stackoverflow.com/questions/1883118/big-list-of-portability-in-python. [Accessed 12 6 2011]. Polented: [38] - GAO Report, Partinot Missile Defense: St. Problem Led to System Failure at Dhahran, Saudi Arabia, 247094, Feb. 4, 1992, http://archive.gao.gov/t2phat6/1.	ftware B-
Formatted: English (US)	
//// Formatted: English (US)	
Formatted: English (US)	
Formatted: English (US)	
Formatted: English (US) Deleted: 3	

46

© ISO/IEC 2015 – All rights reserved

		Deleted: 3
© ISO/IEC 2015- All rights reserved	47	



Page iv: [1] Deleted	Santiago Urueña	5/26/15 12:38:00 PM
Foreword		xvi
Introduction		xvii
<u>1. Scope</u>		1
2. Normative references		1
3. Terms and definitions, symbols and c	onventions	1
3.1 Terms and definitions		1
3.2 Symbols and conventions		5
4. Basic concepts		6
4.1 Purpose of this Technical Report		6
4.2 Intended audience		6
4.3 How to use this document		7
5 Vulnerability issues		8
5.1 Predictable execution		8
5.2 Sources of unpredictability in languation	age specification	9
5.2.1 Incomplete or evolving specificati	<u>on</u>	9
5.2.2 Undefined behaviour		
5.2.3 Unspecified behaviour		
5.2.4 Implementation-defined behaviou	<u>ur</u>	10
5.2.5 Difficult features	_	
5.2.6 Inadequate language support		10
5.3 Sources of unpredictability in langua	age usage	
5.3.1 Porting and interoperation		
5.3.2 Compiler selection and usage		11
6. Programming Language Vulnerabilition	<u>es</u>	11
<u>6.1 General</u>		11
6.2 Terminology		11
6.3 Type System [IHN]		12
6.4 Bit Representations [STR]		14
6.5 Floating-point Arithmetic [PLF]		16
6.6 Enumerator Issues [CCB]		18
6.7 Numeric Conversion Errors [FLC]		20
6.8 String Termination [CJM]		22
6.9 Buffer Boundary Violation (Buffer O	verflow) [HCB]	23
6.10 Unchecked Array Indexing [XYZ]		25
6.11 Unchecked Array Copying [XYW]		27
6.12 Pointer Casting and Pointer Type C	<u>Changes [HFC]</u>	28
6.13 Pointer Arithmetic [RVG]		29
6.14 Null Pointer Dereference [XYH]		
6.15 Dangling Reference to Heap [XYK].		31
6.16 Arithmetic Wrap-around Error [FIF]	34
6.17 Using Shift Operations for Multipli	cation and Division [PIK]	35
6.18 Sign Extension Error [XZI]		
6.19 Choice of Clear Names [NAI]		
6.20 Dead Store [WXQ]		
6.21 Unused Variable [YZS]		40

6.22 Identifier Name Reuse [YOW]	41
6.23 Namespace Issues [BJL]	43
6.24 Initialization of Variables [LAV]	45
6.25 Operator Precedence/Order of Evaluation [JCW]	47
6.26 Side-effects and Order of Evaluation [SAM]	49
6.27 Likely Incorrect Expression [KOA]	50
6.28 Dead and Deactivated Code [XYQ]	52
6.29 Switch Statements and Static Analysis [CLL]	54
6.30 Demarcation of Control Flow [EOJ]	56
6.31 Loop Control Variables [TEX]	57
6.32 Off-by-one Error [XZH]	58
6.33 Structured Programming [EWD]	60
6.34 Passing Parameters and Return Values [CSJ]	61
6.35 Dangling References to Stack Frames [DCM]	63
6.36 Subprogram Signature Mismatch [OTR]	65
6.37 Recursion [GDL]	67
6.38 Ignored Error Status and Unhandled Exceptions [OYB]	68
6.39 Termination Strategy [REU]	70
6.40 Type-breaking Reinterpretation of Data [AMV]	72
6.41 Memory Leak [XYL]	74
6.42 Templates and Generics [SYM]	76
6.43 Inheritance [RIP]	78
6.44 Extra Intrinsics [LRM]	79
6.45 Argument Passing to Library Functions [TRJ]	80
6.46 Inter-language Calling [DJS]	81
6.47 Dynamically-linked Code and Self-modifying Code [NYY]	83
6.48 Library Signature [NSQ]	84
6.49 Unanticipated Exceptions from Library Routines [HJW]	86
6.50 Pre-processor Directives [NMP]	87
6.51 Suppression of Language-defined Run-time Checking [MXB]	89
6.52 Provision of Inherently Unsafe Operations [SKL]	90
6.53 Obscure Language Features [BRS]	91
6.54 Unspecified Behaviour [BQF]	92
6.55 Undefined Behaviour [EWF]	94
6.56 Implementation-defined Behaviour [FAB]	95
6.57 Deprecated Language Features [MEM]	97
6.58 Concurrency – Activation [CGA]	98
6.59 Concurrency – Directed termination [CGT]	.100
6.60 Concurrent Data Access [CGX]	.101
6.61 Concurrency – Premature Termination [CGS]	.103
6.62 Protocol Lock Errors [CGM]	.105
6.63 Inadequately Secure Communication of Shared Resources [CGY]	.107
6.64 Use of unchecked data from an uncontrolled or tainted source [EFS]	.109
6.65 Uncontrolled Format String [SHL]	.110
7. Application Vulnerabilities	.111
7.1 General	.111
7.2 Terminology	.111
7.3 Unspecified Functionality [BVQ]	.111
7.4 Distinguished Values in Data Types [KLK]	.112

7.5 Adherence to Least Privilege [XYN]	113
7.6 Privilege Sandbox Issues [XYO]	114
7.7 Executing or Loading Untrusted Code [XYS]	116
7.8 Memory Locking [XZX]	117
7.9 Resource Exhaustion [XZP]	118
7.10 Unrestricted File Upload [CBF]	119
7.11 Resource Names [HTS]	120
7.12 Injection [RST]	122
7.13 Cross-site Scripting [XYT]	125
7.14 Unquoted Search Path or Element [XZQ]	127
7.15 Improperly Verified Signature [XZR]	128
7.16 Discrepancy Information Leak [XZL]	129
7.17 Sensitive Information Uncleared Before Use [XZK]	130
7.18 Path Traversal [EWR]	130
7.19 Missing Required Cryptographic Step [XZS]	133
7.20 Insufficiently Protected Credentials [XYM]	133
7.21 Missing or Inconsistent Access Control [XZN]	134
7.22 Authentication Logic Error [XZO]	135
7.23 Hard-coded Password [XYP]	136
7.24 Download of Code Without Integrity Check [DLB]	137
7.25 Incorrect Authorization [BJE]	138
7.26 Inclusion of Functionality from Untrusted Control Sphere [DHU]	139
7.27 Improper Restriction of Excessive Authentication Attempts [WPL]	140
7.28 URL Redirection to Untrusted Site ('Open Redirect') [PYQ]	140
7.29 Use of a One-Way Hash without a Salt [MVX]	141
8. New Vulnerabilities	
8.1 General	
8.2 Terminology	
Annex A (informative) Vulnerability Taxonomy and List	
<u>A.1 General</u>	
A.2 Outline of Programming Language Vulnerabilities	
A.3 Outline of Application Vulnerabilities	
<u>A.4 Vulnerability List</u>	145
Annex B (informative) Language Specific Vulnerability Template	148
Annex C (<i>informative</i>) Vulnerability descriptions for the language Ada	150
C.1 Identification of standards and associated documentation	150
C.2 General terminology and concepts	150
C.3 Type System [IHN]	156
C.4 Bit Representation [STR]	156
C.5 Floating-point Arithmetic [PLF]	
C.6 Enumerator Issues [CCB]	
C.7 Numeric Conversion Errors [FLC]	
C.8 String Termination [CJM]	
C.9 Buffer Boundary Violation (Buffer Overflow) [HCB]	
C.10 Unchecked Array Indexing [XYZ]	
C.11 Unchecked Array Copying [XYW]	
C.12 Pointer Casting and Pointer Type Changes [HFC]	

C.13 Pointer Arithmetic [RVG]	160
C.14 Null Pointer Dereference [XYH]	160
C.15 Dangling Reference to Heap [XYK]	160
C.16 Arithmetic Wrap-around Error [FIF]	160
C.17 Using Shift Operations for Multiplication and Division [PIK]	161
C.18 Sign Extension Error [XZI]	161
C.19 Choice of Clear Names [NAI]	161
C.20 Dead store [WXQ]	162
C.21 Unused Variable [YZS]	162
C.22 Identifier Name Reuse [YOW]	163
C.23 Namespace Issues [BJL]	163
C.24 Initialization of Variables [LAV]	163
C.25 Operator Precedence/Order of Evaluation [JCW]	164
C.26 Side-effects and Order of Evaluation [SAM]	164
C.27 Likely Incorrect Expression [KOA]	165
C.28 Dead and Deactivated Code [XYQ]	166
C.29 Switch Statements and Static Analysis [CLL]	166
C.30 Demarcation of Control Flow [EOJ]	167
C.31 Loop Control Variables [TEX]	167
C.32 Off-by-one Error [XZH]	167
C.33 Structured Programming [EWD]	168
C.34 Passing Parameters and Return Values [CSJ]	
C.35 Dangling References to Stack Frames [DCM]	
C.36 Subprogram Signature Mismatch [OTR]	
C.37 Recursion [GDL]	
C.38 Ignored Error Status and Unhandled Exceptions [OYB]	
C.39 Termination Strategy [REU]	
C.40 Type-breaking Reinterpretation of Data [AMV]	
C.41 Memory Leak [XYL]	
C.42 Templates and Generics [SYM]	
C.44 Extra Intrinsics [LRM]	
C.45 Argument Passing to Library Functions [TRJ]	1/3
C.46 Inter-language Calling [DJS]	1/4
C.47 Dynamically-linked Code and Self-modifying Code [NYY]	1/4
C.48 Library Signature [NSQ]	1/4
C.49 Onanticipated Exceptions from Library Routines [HJW]	1/4
C.50 Pre-Processor Directives [NWP]	1/5
C.52 Provision of Inhorently Unsafe Operations [SKL]	175
C.52 Provision of Interently Ofisate Operations [SKL]	175
C.53 Unspecified Rehaviour [ROE]	176
C 55 Undefined Behaviour [EWE]	170
C 56 Implementation-Defined Behaviour [EAB]	172
C 57 Deprecated Language Features [MFM]	
C.58 Implications for standardization	
Annex D (informative) Vulnerability descriptions for the language C	
D.1 Identification of standards and associated documents	
D.2 General terminology and concepts	

D.3 Type System [IHN]	184
D.4 Bit Representations [STR]	185
D.5 Floating-point Arithmetic [PLF]	186
D.6 Enumerator Issues [CCB]	187
D.7 Numeric Conversion Errors [FLC]	188
D.8 String Termination [CJM]	190
D.9 Buffer Boundary Violation (Buffer Overflow) [HCB]	190
D.10 Unchecked Array Indexing [XYZ]	192
D.11 Unchecked Array Copying [XYW]	192
D.12 Pointer Casting and Pointer Type Changes [HFC]	193
D.13 Pointer Arithmetic [RVG]	193
D.14 Null Pointer Dereference [XYH]	194
D.15 Dangling Reference to Heap [XYK]	194
D.16 Arithmetic Wrap-around Error [FIF]	196
D.17 Using Shift Operations for Multiplication and Division [PIK]	197
D.18 Sign Extension Error [XZI]	197
D.19 Choice of Clear Names [NAI]	197
D.20 Dead Store [WXQ]	198
D.21 Unused Variable [YZS]	198
D.22 Identifier Name Reuse [YOW]	198
D.23 Namespace Issues [BJL]	199
D.24 Initialization of Variables [LAV]	199
D.25 Operator Precedence/Order of Evaluation [JCW]	200
D.26 Side-effects and Order of Evaluation [SAM]	200
D.27 Likely Incorrect Expression [KOA]	201
D.28 Dead and Deactivated Code [XYQ]	202
D.29 Switch Statements and Static Analysis [CLL]	203
D.30 Demarcation of Control Flow [EOJ]	204
D.31 Loop Control Variables [TEX]	205
D.32 Off-by-one Error [XZH]	206
D.33 Structured Programming [EWD]	206
D.34 Passing Parameters and Return Values [CSJ]	207
D.35 Dangling References to Stack Frames [DCM]	208
D.36 Subprogram Signature Mismatch [OTR]	208
D.37 Recursion [GDL]	209
D.38 Ignored Error Status and Unhandled Exceptions [OYB]	209
D.39 Termination Strategy [REU]	210
D.40 Type-breaking Reinterpretation of Data [AMV]	210
D.41 Memory Leak [XYL]	211
D.42 Templates and Generics [SYM]	211
D.43 Inheritance [RIP]	211
D.44 Extra Intrinsics [LRM]	211
D.45 Argument Passing to Library Functions [TRJ]	212
D.46 Inter-language Calling [DJS]	212
D.47 Dynamically-linked Code and Self-modifying Code [NYY]	212
D.48 Library Signature [NSQ]	213
D.49 Unanticipated Exceptions from Library Routines [HJW]	213
D.50 Pre-processor Directives [NMP]	214
D.51 Suppression of Language-defined Run-time Checking [MXB]	215

D.52 Provision of Inherently Unsafe Operations [SKL]	215
D.53 Obscure Language Features [BRS]	215
D.54 Unspecified Behaviour [BQF]	216
D.55 Undefined Behaviour [EWF]	216
D.56 Implementation-defined Behaviour [FAB]	217
D.57 Deprecated Language Features [MEM]	217
D.58 Implications for standardization	218
Annex E (informative) Vulnerability descriptions for the language Python	221
E.1 Identification of standards and associated documents	221
E.2 General Terminology and Concepts	222
E.3 Type System [IHN]	226
E.4 Bit Representations [STR]	228
E.5 Floating-point Arithmetic [PLF]	229
E.6 Enumerator Issues [CCB]	229
E.7 Numeric Conversion Errors [FLC]	230
E.8 String Termination [CJM]	231
E.9 Buffer Boundary Violation [HCB]	231
E.10 Unchecked Array Indexing [XYZ]	231
E.11 Unchecked Array Copying [XYW]	231
E.12 Pointer Casting and Pointer Type Changes [HFC]	231
E.13 Pointer Arithmetic [RVG]	231
E.14 Null Pointer Dereference [XYH]	231
E.15 Dangling Reference to Heap [XYK]	231
E.16 Arithmetic Wrap-around Error [FIF]	232
E.17 Using Shift Operations for Multiplication and Division [PIK]	232
E.18 Sign Extension Error [XZI]	232
E.19 Choice of Clear Names [NAI]	232
E.20 Dead Store [WXQ]	234
E.21 Unused Variable [YZS]	235
E.22 Identifier Name Reuse [YOW]	235
E.23 Namespace Issues [BJL]	237
E.24 Initialization of Variables [LAV]	240
E.25 Operator Precedence/Order of Evaluation [JCW]	240
E.26 Side-effects and Order of Evaluation [SAM]	241
E.27 Likely Incorrect Expression [KOA]	242
E.28 Dead and Deactivated Code [XYQ]	243
E.29 Switch Statements and Static Analysis [CLL]	244
E.30 Demarcation of Control Flow [EOJ]	244
E.31 Loop Control Variables [TEX]	245
E.32 Off-by-one Error [XZH]	246
E.33 Structured Programming [EWD]	246
E.34 Passing Parameters and Return Values [CSJ]	247
E.35 Dangling References to Stack Frames [DCM]	249
E.36 Subprogram Signature Mismatch [OTR]	249
E.37 Recursion [GDL]	249
E.38 Ignored Error Status and Unhandled Exceptions [OYB]	249
E.39 Termination Strategy [REU]	250
E.40 Type-breaking Reinterpretation of Data [AMV]	250
E.41 Memory Leak [XYL]	250

E.42 Templates and Generics [SYM]	251
E.43 Inheritance [RIP]	251
E.44 Extra Intrinsics [LRM]	251
E.45 Argument Passing to Library Functions [TRJ]	252
E.46 Inter-language Calling [DJS]	252
E.47 Dynamically-linked Code and Self-modifying Code [NYY]	253
E.48 Library Signature [NSQ]	253
E.49 Unanticipated Exceptions from Library Routines [HJW]	254
E.50 Pre-processor Directives [NMP]	254
E.51 Suppression of Language-defined Run-time Checking [MXB]	254
E.52 Provision of Inherently Unsafe Operations [SKL]	254
E.53 Obscure Language Features [BRS]	255
E.54 Unspecified Behaviour [BQF]	257
E.55 Undefined Behaviour [EWF]	258
E.56 Implementation-defined Behaviour [FAB]	259
E.57 Deprecated Language Features [MEM]	
Annex F (<i>informative</i>) Vulnerability descriptions for the language Ruby	
F.1 Identification of standards and associated documents	
F.2 General Terminology and Concepts	
F.3 Type System [IHN]	
F.4 Bit Representations [STR]	
F.5 Floating-point Arithmetic [PLF]	264
F.6 Enumerator Issues [CCB]	264
F.7 Numeric Conversion Errors [FLC]	265
F.8 String Termination [CJM]	265
F.9 Buffer Boundary Violation (Buffer Overflow) [HCB]	265
F.10 Unchecked Array Indexing [XYZ]	265
F.11 Unchecked Array Copying [XYW]	265
F.12 Pointer Casting and Pointer Type Changes [HFC]	
F.13 Pointer Arithmetic [RVG]	
F.14 Null Pointer Dereference [XYH]	
F.15 Dangling Reference to Heap [XYK]	
F.16 Arithmetic Wrap-around Error [FIF]	
F.17 Using Shift Operations for Multiplication and Division [PIK]	
F.18 Sign Extension Error [XZI]	
F.19 Choice of Clear Names [NAI]	
F.20 Dead Store [WXQ]	
F.21 Unused Variable [YZS]	
F.22 Identifier Name Reuse [YOW]	
F.23 Namespace Issues [BJL]	
F.24 Initialization of Variables [LAV]	
F.25 Operator Precedence/Order of Evaluation [JCW]	
F.26 Side-effects and Order of Evaluation [SAM]	
F.27 Likely Incorrect Expression [KOA]	270
F.28 Dead and Deactivated Code [XYQ]	270
F.29 Switch Statements and Static Analysis [CLL]	271
F.30 Demarcation of Control Flow [EOJ]	271
F.31 Loop Control Variables [TEX]	271
F.32 Off-by-one Error [XZH]	271

F.33 Structured Programming [EWD]	272
F.34 Passing Parameters and Return Values [CSJ]	272
F.35 Dangling References to Stack Frames [DCM]	273
F.36 Subprogram Signature Mismatch [OTR]	273
F.37 Recursion [GDL]	274
F.38 Ignored Error Status and Unhandled Exceptions [OYB]	274
F.39 Termination Strategy [REU]	274
F.40 Type-breaking Reinterpretation of Data [AMV]	274
F.41 Memory Leak [XYL]	274
F.42 Templates and Generics [SYM]	275
F.43 Inheritance [RIP]	275
F.44 Extra Intrinsics [LRM]	275
F.45 Argument Passing to Library Functions [TRJ]	275
F.46 Inter-language Calling [DJS]	275
F.47 Dynamically-linked Code and Self-modifying Code [NYY]	276
F.48 Library Signature [NSQ]	276
F.49 Unanticipated Exceptions from Library Routines [HJW]	276
F.50 Pre-processor Directives [NMP]	276
F.51 Suppression of Language-defined Run-time Checking [MXB]	277
F.52 Provision of Inherently Unsafe Operations [SKL]	277
F.53 Obscure Language Features [BRS]	277
F.54 Unspecified Behaviour [BQF]	277
F.55 Undefined Behaviour [EWF]	277
F.56 Implementation-defined Behaviour [FAB]	278
F.57 Deprecated Language Features [MEM]	278
Annex G (<i>informative</i>) Vulnerability descriptions for the language SPARK	279
Annex G (<i>informative</i>) Vulnerability descriptions for the language SPARK	279
Annex G (<i>informative</i>) Vulnerability descriptions for the language SPARK G.1 Identification of standards and associated documentation G.2 General terminology and concepts	279 279 279
Annex G (<i>informative</i>) Vulnerability descriptions for the language SPARK G.1 Identification of standards and associated documentation G.2 General terminology and concepts G.3 Type System [IHN]	279 279 279 279
Annex G (<i>informative</i>) Vulnerability descriptions for the language SPARK G.1 Identification of standards and associated documentation G.2 General terminology and concepts G.3 Type System [IHN] G.4 Bit Representation [STR]	279 279 279 280 281
Annex G (<i>informative</i>) Vulnerability descriptions for the language SPARK	279 279 279 280 281 281
Annex G (<i>informative</i>) Vulnerability descriptions for the language SPARK G.1 Identification of standards and associated documentation G.2 General terminology and concepts G.3 Type System [IHN] G.4 Bit Representation [STR] G.5 Floating-point Arithmetic [PLF] G.6 Enumerator Issues [CCB]	279 279 280 281 281 281
Annex G (<i>informative</i>) Vulnerability descriptions for the language SPARK	279 279 280 281 281 281 281 281
Annex G (informative) Vulnerability descriptions for the language SPARK	279 279 280 281 281 281 281 281
Annex G (informative) Vulnerability descriptions for the language SPARK	279 279 280 281 281 281 281 281 281
Annex G (informative) Vulnerability descriptions for the language SPARK G.1 Identification of standards and associated documentation G.2 General terminology and concepts G.3 Type System [IHN] G.4 Bit Representation [STR] G.5 Floating-point Arithmetic [PLF] G.6 Enumerator Issues [CCB] G.7 Numeric Conversion Errors [FLC] G.8 String Termination [CJM] G.9 Buffer Boundary Violation (Buffer Overflow) [HCB] G.10 Unchecked Array Indexing [XYZ]	279 279 280 281 281 281 281 281 281 281
Annex G (informative) Vulnerability descriptions for the language SPARK G.1 Identification of standards and associated documentation G.2 General terminology and concepts G.3 Type System [IHN] G.4 Bit Representation [STR] G.5 Floating-point Arithmetic [PLF] G.6 Enumerator Issues [CCB] G.7 Numeric Conversion Errors [FLC] G.8 String Termination [CJM] G.9 Buffer Boundary Violation (Buffer Overflow) [HCB] G.10 Unchecked Array Indexing [XYZ] G.11 Unchecked Array Copying [XYW]	279 279 280 281 281 281 281 281 281 281 281
Annex G (informative) Vulnerability descriptions for the language SPARK	279 279 280 281 281 281 281 281 281 281 281 281 281 281
Annex G (informative) Vulnerability descriptions for the language SPARK	279 279 280 281 281 281 281 281 281 281 281 281 281 282
Annex G (<i>informative</i>) Vulnerability descriptions for the language SPARK G.1 Identification of standards and associated documentation G.2 General terminology and concepts G.3 Type System [IHN] G.4 Bit Representation [STR] G.5 Floating-point Arithmetic [PLF] G.6 Enumerator Issues [CCB] G.7 Numeric Conversion Errors [FLC] G.8 String Termination [CJM]. G.9 Buffer Boundary Violation (Buffer Overflow) [HCB] G.10 Unchecked Array Indexing [XYZ] G.11 Unchecked Array Copying [XYW] G.12 Pointer Casting and Pointer Type Changes [HFC] G.14 Null Pointer Dereference [XYH]	279 279 279 280 281 281 281 281 281 281 281 281 281 281 282 282
Annex G (informative) Vulnerability descriptions for the language SPARK G.1 Identification of standards and associated documentation G.2 General terminology and concepts G.3 Type System [IHN] G.4 Bit Representation [STR] G.5 Floating-point Arithmetic [PLF] G.5 Floating-point Arithmetic [PLF] G.6 Enumerator Issues [CCB] G.7 Numeric Conversion Errors [FLC] G.8 String Termination [CJM] G.9 Buffer Boundary Violation (Buffer Overflow) [HCB] G.10 Unchecked Array Indexing [XYZ] G.11 Unchecked Array Copying [XYW] G.12 Pointer Casting and Pointer Type Changes [HFC] G.13 Pointer Arithmetic [RVG] G.15 Dangling Reference to Heap [XYK]	279 279 280 281 281 281 281 281 281 281 281 281 281 282 282 282 282
Annex G (informative) Vulnerability descriptions for the language SPARK	279 279 279 280 .281 .281 .281 .281 .281 .281 .281 .281
Annex G (informative) Vulnerability descriptions for the language SPARK. G.1 Identification of standards and associated documentation. G.2 General terminology and concepts. G.3 Type System [IHN] G.4 Bit Representation [STR] G.5 Floating-point Arithmetic [PLF]. G.6 Enumerator Issues [CCB] G.7 Numeric Conversion Errors [FLC] G.8 String Termination [CJM]. G.9 Buffer Boundary Violation (Buffer Overflow) [HCB]. G.10 Unchecked Array Indexing [XYZ] G.11 Unchecked Array Copying [XYW] G.12 Pointer Casting and Pointer Type Changes [HFC]. G.13 Pointer Arithmetic [RVG]. G.14 Null Pointer Dereference [XYH] G.15 Dangling Reference to Heap [XYK] G.16 Arithmetic Wrap-around Error [FIF]. G.17 Using Shift Operations for Multiplication and Division [PIK].	279 279 279 280 281 281 281 281 281 281 281 281 281 282 282 282 282 282 282 282
Annex G (informative) Vulnerability descriptions for the language SPARK G.1 Identification of standards and associated documentation G.2 General terminology and concepts G.3 Type System [IHN] G.4 Bit Representation [STR] G.5 Floating-point Arithmetic [PLF] G.6 Enumerator Issues [CCB] G.7 Numeric Conversion Errors [FLC] G.8 String Termination [CJM] G.9 Buffer Boundary Violation (Buffer Overflow) [HCB] G.11 Unchecked Array Indexing [XYZ] G.11 Unchecked Array Copying [XYW] G.12 Pointer Casting and Pointer Type Changes [HFC] G.13 Pointer Arithmetic [RVG] G.14 Null Pointer Dereference [XYH] G.15 Dangling Reference to Heap [XYK] G.16 Arithmetic Wrap-around Error [FIF] G.17 Using Shift Operations for Multiplication and Division [PIK]	279 279 279 280 281 281 281 281 281 281 281 281 281 281 282 282 282 282 282 282 282 282
Annex G (informative) Vulnerability descriptions for the language SPARK. G.1 Identification of standards and associated documentation. G.2 General terminology and concepts. G.3 Type System [IHN] G.4 Bit Representation [STR] G.5 Floating-point Arithmetic [PLF] G.6 Enumerator Issues [CCB] G.7 Numeric Conversion Errors [FLC] G.8 String Termination [CJM] G.9 Buffer Boundary Violation (Buffer Overflow) [HCB] G.10 Unchecked Array Indexing [XYZ] G.11 Unchecked Array Copying [XYW] G.12 Pointer Casting and Pointer Type Changes [HFC] G.13 Pointer Arithmetic [RVG] G.15 Dangling Reference to Heap [XYK] G.16 Arithmetic Wrap-around Error [FIF] G.17 Using Shift Operations for Multiplication and Division [PIK] G.18 Sign Extension Error [XZI]	279 279 280 281 281 281 281 281 281 281 281 281 282 282 282 282 282 282 282 282 282
Annex G (informative) Vulnerability descriptions for the language SPARK. G.1 Identification of standards and associated documentation G.2 General terminology and concepts G.3 Type System [IHN] G.4 Bit Representation [STR] G.5 Floating-point Arithmetic [PLF] G.6 Enumerator Issues [CCB] G.7 Numeric Conversion Errors [FLC] G.8 String Termination [CJM] G.9 Buffer Boundary Violation (Buffer Overflow) [HCB] G.10 Unchecked Array Indexing [XYZ] G.11 Unchecked Array Copying [XYW] G.12 Pointer Casting and Pointer Type Changes [HFC] G.13 Pointer Arithmetic [RVG] G.14 Null Pointer Dereference [XYH] G.15 Dangling Reference to Heap [XYK] G.16 Arithmetic Wrap-around Error [FIF] G.17 Using Shift Operations for Multiplication and Division [PIK] G.18 Sign Extension Error [XZI] G.19 Choice of Clear Names [NAI] G.20 Dead store [WXQ]	279 279 279 280 281 281 281 281 281 281 281 281 281 281 282 282 282 282 282 282 282 282 282 282
Annex G (informative) Vulnerability descriptions for the language SPARK G.1 Identification of standards and associated documentation G.2 General terminology and concepts G.3 Type System [IHN] G.4 Bit Representation [STR] G.5 Floating-point Arithmetic [PLF] G.6 Enumerator Issues [CCB] G.7 Numeric Conversion Errors [FLC] G.8 String Termination [CJM] G.9 Buffer Boundary Violation (Buffer Overflow) [HCB] G.10 Unchecked Array Indexing [XYZ] G.11 Unchecked Array Copying [XYW] G.12 Pointer Casting and Pointer Type Changes [HFC] G.13 Pointer Arithmetic [RVG] G.14 Null Pointer Dereference [XYH] G.15 Dangling Reference to Heap [XYK] G.16 Arithmetic Wrap-around Error [FIF] G.17 Using Shift Operations for Multiplication and Division [PIK] G.19 Choice of Clear Names [NAI] G.20 Dead store [WXQ] G.21 Unused Variable [YZS]	279 279 280 281 281 281 281 281 281 281 281 281 282 282 282 282 282 282 282 282 282 282 282 282 282 282
Annex G (informative) Vulnerability descriptions for the language SPARK G.1 Identification of standards and associated documentation G.2 General terminology and concepts G.3 Type System [IHN] G.4 Bit Representation [STR] G.5 Floating-point Arithmetic [PLF]. G.6 Enumerator Issues [CCB] G.7 Numeric Conversion Errors [FLC] G.8 String Termination [CJM] G.9 Buffer Boundary Violation (Buffer Overflow) [HCB] G.10 Unchecked Array Indexing [XYZ] G.11 Unchecked Array Copying [XYW] G.12 Pointer Casting and Pointer Type Changes [HFC] G.13 Pointer Arithmetic [RVG] G.14 Null Pointer Dereference [XYH] G.15 Dangling Reference to Heap [XYK] G.16 Arithmetic Wrap-around Error [FIF] G.17 Using Shift Operations for Multiplication and Division [PIK] G.18 Sign Extension Error [XZI] G.12 Unused Variable [YZS] G.21 Unused Variable [YZS] G.22 Identifier Name Reuse [YOW]	279 279 279 280 281 281 281 281 281 281 281 281 281 281 282 282 282 282 282 282 282 282 282 283 283

G.24 Initialization of Variables [LAV]	
G.25 Operator Precedence/Order of Evaluation [JCW]	
G.26 Side-effects and Order of Evaluation [SAM]	
G.27 Likely Incorrect Expression [KOA]	
G.28 Dead and Deactivated Code [XYQ]	
G.29 Switch Statements and Static Analysis [CLL]	
G.30 Demarcation of Control Flow [EOJ]	
G.31 Loop Control Variables [TEX]	
G.32 Off-by-one Error [XZH]	
G.33 Structured Programming [EWD]	
G.34 Passing Parameters and Return Values [CSJ]	
G.35 Dangling References to Stack Frames [DCM]	
G.36 Subprogram Signature Mismatch [OTR]	
G.37 Recursion [GDL]	
G.38 Ignored Error Status and Unhandled Exceptions [OYB]	
G.39 Termination Strategy [REU]	
G.40 Type-breaking Reinterpretation of Data [AMV]	
G.41 Memory Leak [XYL]	
G.42 Templates and Generics [SYM]	
G.43 Inheritance [RIP]	
G.44 Extra Intrinsics [LRM]	
G.45 Argument Passing to Library Functions [TRJ]	
G.46 Inter-language Calling [DJS]	
G.47 Dynamically-linked Code and Self-modifying Code [NYY]	
G.48 Library Signature [NSQ]	
G.49 Unanticipated Exceptions from Library Routines [HJW]	
G.50 Pre-Processor Directives [NMP]	287
G.51 Suppression of Language-defined Run-time Checking [MXB]	287
G.52 Provision of Inherently Unsafe Operations [SKL]	
G.53 Obscure Language Features [BRS]	
G.54 Unspecified Behaviour [BQF]	
G.55 Undefined Behaviour [EWF]	
G.56 Implementation-Defined Behaviour [FAB]	
G.57 Deprecated Language Features [MEM]	
G.58 Implications for standardization	
Annex H (<i>informative</i>) Vulnerability descriptions for the language PHP	
H.1 Identification of standards and associated documentation	
H.2 General Terminology and Concepts	290
H.3 Type System [IHN]	291
H.4 Bit Representations [STR]	292
H.5 Floating-point Arithmetic [PLF]	293
H.6 Enumerator Issues [CCB]	293
H.7 Numeric Conversion Errors [FLC]	294
H.8 String Termination [CJM]	295
H.9 Buffer Boundary Violation (Buffer Overflow) [HCB]	296
H.10 Unchecked Array Indexing [XYZ]	296
H.11 Unchecked Array Copying [XYW]	296
H.12 Pointer Casting and Pointer Type Changes [HFC]	296
H.13 Pointer Arithmetic [RVG]	

H.14 Null Pointer Dereference [XYH]	297
H.15 Dangling Reference to Heap [XYK]	297
H.16 Arithmetic Wrap-around Error [FIF]	297
H.17 Using Shift Operations for Multiplication and Division [PIK]	298
H.18 Sign Extension Error [XZI]	299
H.19 Choice of Clear Names [NAI]	299
H.20 Dead Store [WXQ]	301
H.21 Unused Variable [YZS]	301
H.22 Identifier Name Reuse [YOW]	301
H.23 Namespace Issues [BJL]	302
H.24 Initialization of Variables [LAV]	303
H.25 Operator Precedence/Order of Evaluation [JCW]	304
H.26 Side-effects and Order of Evaluation [SAM]	304
H.27 Likely Incorrect Expression [KOA]	305
H.28 Dead and Deactivated Code [XYQ]	306
H.29 Switch Statements and Static Analysis [CLL]	307
H.30 Demarcation of Control Flow [EOJ]	307
H.31 Loop Control Variables [TEX]	308
H.32 Off-by-one Error [XZH]	309
H.33 Structured Programming [EWD]	309
H.34 Passing Parameters and Return Values [CSJ]	310
H.35 Dangling References to Stack Frames [DCM]	310
H.36 Subprogram Signature Mismatch [OTR]	310
H.37 Recursion [GDL]	311
H.38 Ignored Error Status and Unhandled Exceptions [OYB]	311
H.39 Termination Strategy [REU]	313
H.40 Type-breaking Reinterpretation of Data [AMV]	313
H.41 Memory Leak [XYL]	313
H.42 Templates and Generics [SYM]	314
H.43 Inheritance [RIP]	314
H.44 Extra Intrinsics [LRM]	314
H.45 Argument Passing to Library Functions [TRJ]	314
H.46 Inter-language Calling [DJS]	314
H.47 Dynamically-linked Code and Self-modifying Code [NYY]	315
H.48 Library Signature [NSQ]	315
H.49 Unanticipated Exceptions from Library Routines [HJW]	315
H.50 Pre-processor Directives [NMP]	316
H.51 Suppression of Run-time Checking [MXB]	316
H.52 Provision of Inherently Unsafe Operations [SKL]	316
H.53 Obscure Language Features [BRS]	316
H.54 Unspecified Behaviour [BQF]	317
H.55 Undefined Behaviour [EWF]	318
H.56 Implementation-defined Behaviour [FAB]	319
H.57 Deprecated Language Features [MEM]	319
Annex I (informative) Vulnerability descriptions for the language Fortran	320
I.1 Identification of Standards	320
I.2 General Terminology and Concepts	320
I.3 Type System [IHN]	323
I.4 Bit Representations [STR]	324

L6 Enumerator Issues [CCB] 326 17 Numeric Conversion Errors [FLC] 326 18 String Termination [CM] 327 19 Buffer Boundary Violation [HCB] 327 110 Unchecked Array (ndexing [XYZ] 328 111 Unchecked Array Copying [XYV] 329 112 Pointer Casting and Pointer Type Changes [HFC] 330 113 Pointer Arithmetic [RVG] 330 114 A Null Pointer Dereference [XYH] 330 115 L1 Applicability to language 331 111 Using Shift Operations for Multiplication and Division [PIK] 332 113 Object Of Clear Names [NA] 332 121 Ounder Essues [UVM] 333 121 Unused Variable [YZS] 333 121 Unused Variable [YZS] 333 121 Unused Variable [YZS] 333 122 Udentifier Name Reuse [YOW] 333 123 Namespace Issues [BIL] 334 124 Unitialization of Variables [LAV] 334 125 Obserator Precedence/Order of Evaluation [LCW] 334 126 Side-effice Evaluation [SAM] 335 127 Likely Incorrect Expression [KOA] 336 128 Decad and Deactivated Code [XVQ] 336 <td< th=""><th>I.5 Floating-point Arithmetic [PLF]</th><th>325</th></td<>	I.5 Floating-point Arithmetic [PLF]	325
1.7 Numeric Conversion Errors [EC]3261.8 String Termination [CIM]3271.9 Buffer Boundary Violation [HCB]3271.10 Unchecked Array Indexing [XY2]3281.11 Dinchecked Array Copying [XVV]3291.12 Pointer Casting and Pointer Type Changes [HFC]3301.13 Pointer Arithmetic [RVG]3301.14 Null Pointer Dereference [XYH]3301.15 1 Applicability to language.3311.16 Arithmetic Wrap-around Error [FIF]3311.17 Using Shift Operations for Multiplication and Division [PIK]3321.20 Choice of Clear Names [NAI]3321.20 Dead Store [WXQ]3331.21 Unused Variable [YZS]3331.21 Unused Variable [YZS]3331.22 Identifier Name Reuse [YOW]3331.23 Namespace Issues [BUL]3441.24 Initialization of Variables [LAV]3341.25 Operator Precedence/Order of Evaluation [SAM]3351.22 Udentifier Name Reuse [YOW]3361.23 Logentifier Name Reuse [YOW]3361.24 Initialization of Cortrol Flow [EOJ]3361.25 Operator Precedence/Order of Evaluation [SAM]3351.27 Likely Incorrect Expression [KOA]3361.28 Dead and Deatityted Code [XYQ]3361.31 Logo Control Variables [TEV]3371.33 Structured Programming [EWD]3361.34 Passing Parameters and Return Values [CS]3361.35 Dangling References US Stack Frames [DCM]3391.36 Logo Control Isov [GD]3361.37 Recursing IGDL]337 <tr< td=""><td>I.6 Enumerator Issues [CCB]</td><td>326</td></tr<>	I.6 Enumerator Issues [CCB]	326
L8 String Termination [CM] 327 19 Buffer Boundary Violation [HCB] 327 130 Unchecked Array Locaing [XYV] 328 131 Unchecked Array Locaing [XYV] 330 132 Pointer Casting and Pointer Type Changes [HFC] 330 131 Unchecked Array Locaing [XYV] 330 132 Pointer Arithmetic [RVG] 330 134 Null Pointer Dereference [XYH] 330 115.1 Applicability to language 331 115.1 Applicability to language 331 113 Chicker Wrap-around Error [FIF] 331 131 Display Extension Error [XZ] 332 132 Choice OClear Names [NA] 332 132 Obead Store [WXQ] 333 132 Udentifier Name Reuse [YOW] 333 132 Namespace Issues [OU] 334 125 Operator Precedence/Order of Evaluation [ICW] 334 126 Side-effects and Order of Evaluation [SAM] 335 131 Loop Control Variables [LAV] 336 132 Operator Precedence/Order of Evaluation [SAM] 335 132 Dapatign References to SACk Frames [DCM] 336 132 Dapatign References to SAck Frames [DCM] 336 132 Opt-Ly-one Error [XZH] 337 <td>I.7 Numeric Conversion Errors [FLC]</td> <td>326</td>	I.7 Numeric Conversion Errors [FLC]	326
19 Buffer Boundary Violation [HCB] 327 1.10 Unchecked Array Indexing [XYZ] 328 1.11 Unchecked Array Copying [XYV] 329 1.12 Pointer Casting and Pointer Type Changes [HFC] 330 1.13 Pointer Arithmetic [RVG] 330 1.14 Null Pointer Dereference [XYH] 330 1.15 Applicability to language 331 1.16 Arithmetic Wrap-around Error [FF] 331 1.17 Using Shift Operations for Multiplication and Division [PIX] 332 1.20 Dead Store [WXQ] 333 1.21 Oncide Ariable [YZS] 333 1.22 Identifier Name Reuse [YOW] 333 1.23 Identifier Name Reuse [YOW] 333 1.24 Initialization of Variables [LAV] 334 1.25 Operator Precedence/Order of Evaluation [JCW] 334 1.28 Dead and Deactivated Code [XYQ] 336 1.31 Loop Control Flow [EOJ] 336 1.32 Off-by-one Error [XZH] 337 1.33 Extructured Programming [EWD] 336 1.34 Passing Parameters and Return Values [CS] 336 1.33 Extructured Programming [EWD] 336 1.34 Passing Parameters and Return Values [CS] 340 1.34 Passing Parameters and	I.8 String Termination [CJM]	327
1.10 Unchecked Array (ndexing [XY2]	I.9 Buffer Boundary Violation [HCB]	327
1.11 Unchecked Array Copying [XYW]3291.12 Pointer Casting and Pointer Type Changes [HFC]3301.13 Pointer Arithmetic [RVG]3301.14 Null Pointer Dereference [XYH]3301.15 A Indhretic Wrap-around Error [FE]3311.16 Arithmetic Kwrap-around Error [FE]3311.17 Using Shift Operations for Multiplication and Division [PIK]3321.18 Sign Extension Error [X2]3321.20 Dead Store (WXQ)3331.21 Unused Variable [YZS]3331.21 Unused Variable [YZS]3331.22 Identifier Name Reuse [YOW]3341.23 Namespace Issues [BIL]3341.24 Initialization of Variables [LAV]3341.25 Side-effects and Order of Evaluation [JCW]3341.26 Side-effects and Order of Evaluation [JCW]3341.27 Likely Incorrect Expression [KOA]3351.28 Dead and Deactivated Code [XYQ]3361.29 Switch Statements and Static (SI)3371.31 Loop Control Flow [EO]3361.31 Loop Control Variables [TEX]3371.32 Off-by-one Error [X2H]3381.35 Suctured Programming [EWD]3381.36 Subprogram Signature Mismatch [OTR]3391.37 Recursion [CDL]3391.38 Ignored Error Status and Unhandled Exceptions [OYB]3401.34 Laterianguage Caling [DIS]3411.44 Extra Intrinsics [LRM]3411.41 Interiance [RIP]3431.42 Templates and Generics [SYM]3411.43 Inheritance [RIP]3431.44 Extra Intrinsics [LRM] <t< td=""><td>I.10 Unchecked Array Indexing [XYZ]</td><td>328</td></t<>	I.10 Unchecked Array Indexing [XYZ]	328
112 Pointer Casting and Pointer Type Changes [HFC] 330 113 Pointer Arithmetic [RVG] 330 114 Null Pointer Dereferace [XYH] 330 115.1 Applicability to language 331 115.1 Applicability to language 331 115.1 Applicability to language 331 115.1 Applicability to language 332 115.2 National Statematics 332 12.1 Unused Variable [YZS] 333 12.2 Identifier Name Reuse [YOW] 333 12.3 Namespace Issues [BIL] 334 12.4 Initialization of Variables [LAV] 334 12.5 Operator Precedence/Order of Evaluation [JCW] 334 12.5 Generator Precedence/Order of Evaluation [JCW] 334 12.5 Operator Precedence/Order of Evaluation [JCW] 335 12.7 Likely Incorrect Expression [KOA] 335 12.8 Dead and Deactivated Code [XYQ] 336 13.9 Demarcation of Control Flow [EO] 336 13.9 Demarcation of Control Flow [EO] 336<	I.11 Unchecked Array Copying [XYW]	329
1.13 Pointer Arithmetic [RVG] 330 1.14 Null Pointer Dereference [XYH] 330 1.15.1 Applicability to language 331 1.16 Arithmetic Wrap-around Error [FIF] 331 1.16 Arithmetic Wrap-around Error [FIF] 331 1.17 Using Shift Operations for Multiplication and Division [PIK] 332 1.18 Sign Extension Error [XZ] 332 1.20 Dead Store [WXQ] 333 1.21 Unsed Variable [YS] 333 1.22 Identifier Name Reuse [YOW] 333 1.23 Namespace Issues [BIL] 334 1.24 Initialization of Variables [LAV] 334 1.25 State-effects and Order of Evaluation [ICW] 334 1.26 Side-effects and Order of Evaluation [ICW] 336 1.27 Likely incorrect Expression [KOA] 335 1.28 Dead and Deactivated Code [XYQ] 336 1.30 Demarcation of Control Flow [EO] 337 1.31 Loop Control Variables [LEV] 337 1.32 Off-by-one Error [XZH] 337 1.33 Structured Programming [EWD] 338 1.34 Passing Parameters and Return Values [CSJ] 338 1.33 Egnored Error Status and Unhandled Exceptions [OYB] 340 1.34 Passing P	I.12 Pointer Casting and Pointer Type Changes [HFC]	330
114 Null Pointer Dereference [XYH]330115 1 Applicability to language.331115 1 Applicability to language.331115 1 Life Using Shift Operations for Multiplication and Division [PIK]332118 Sign Extension Error [XZI]332120 Dead Store [WXQ]333121 Unused Variable [VZS]333122 Unused Variable [VZS]333123 Namespace Issues [BiL]334124 Initialization of Variables [LAV]334125 Operator Precedence/Order of Evaluation [JCW]334126 Scheeffects and Order of Evaluation [JCW]335127 Likely Incorrect Expression [KOA]335128 Dead and Deactivated Code [XYQ]336129 Switch Statements and Static Analysis [CLL]336131 Loop Control Variables [LEX]337132 Off-by-one Error [XZH]337133 Loop Control Variables [LEX]337134 Passing Parameters and Return Values [CSJ]338135 Dangling References to Stack Frames [DCM]339136 Subprogram Signature Mismatch [OTR]339137 Recursion [GDL]340144 Extra Intrinsics [LRM]341142 Templates and Generics [SYM]341144 Extra Intrinsics [LM]341144 Extra Intrinsics [LM]341145 Subprogram Signature Mismatch [OTR]342146 Inter-language Calling [DIS]342145 Lapper-processor Directives [NMP]341144 Extra Intrinsics [LM]344145 Lapper-processor Directives [NMP]343146 Inter-language Calling [DIS]<	I.13 Pointer Arithmetic [RVG]	330
L15.1 Applicability to language331L16 Arithmetic Wrap-around Error [FF]331L17 Using Shift Operations for Multiplication and Division [PIK]332L18 Sign Extension Error [XZ]332L19 Choice of Clear Names [NAI]332L20 Dead Store [WXQ]333L21 Unused Variable [YZS]333L22 Identifier Name Reuse [YOW]333L23 Identifier Name Reuse [YOW]333L24 Initialization of Variables [LAV]344L25 Operator Precedence/Order of Evaluation [ICW]344L26 Side-effects and Order of Evaluation [ICW]344L26 Side-effects and Order of Evaluation [SAM]335L27 Likely Incorrect Expression [KOA]336L30 Demarcation of Control Flow [EOJ]336L30 Demarcation of Control Flow [EOJ]336L31 Loop Control Variables [TEX]337L32 Off-by-one Error (XZH]337L33 Structured Programming [EWD]338L34 Passing Parameters and Return Values [CSJ]338L35 Dangling References to Stack Frames [DCM]339L36 Ubprogram Signature Mismatch [OTR]340L30 Termination Strategy [REU]340L42 Templates and Generics [SYM]341L41 Intensica [L4X]341L42 Templates and Generics [SYM]342L43 Inheritance [RIP]342L44 Intrinsics [L4X]341L45 Argument Passing to Library Functions [TRI]342L45 Inter-Inaguage Calling [DIS]342L45 Inter-Inaguage Calling [DIS]342L46 Inter-Inaguage Calling [I.14 Null Pointer Dereference [XYH]	330
1.16 Arithmetic Wrap-around Error [FIF]	I.15.1 Applicability to language	331
1.17 Using Shift Operations for Multiplication and Division [PIK]3321.18 Sign Extension Error [X21]3321.19 Choice of Clear Names [NAI]3321.20 Dead Store [WXQ]3331.21 Unused Variable [YZS]3331.22 Identifier Name Reuse [YOW]3331.23 Namespace Issues [BIL]3341.24 Initialization of Variables [LAV]3341.25 Operator Precedence/Order of Evaluation [JCW]3341.26 Side-effects and Order of Evaluation [SAM]3351.27 Likely Incorrect Expression [KOA]3351.28 Dead and Deactivated Code [XYQ]3361.29 Switch Statements and Static Analysis [CLL]3361.20 Demarcation of Control Flow [E0J]3361.31 Loop Control Variables [TEX]3371.33 Structured Programming [EWD]3381.34 Passing Parameters and Return Values [CSJ]3381.35 Supprogram Signature Mismatch [OTR]3391.36 Subprogram Signature Mismatch [OTR]3391.37 Recursion [SOL]3401.40 Type-breaking to Library Functions [TWJ]3411.41 Memory Leek [XYL]3411.42 Templates and Generics [SYM]3411.43 Inheritance [RIP]3421.44 Extra Intrinsics to Library Functions [TRJ]3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-Ianguage Calling [DIS]3421.47 Dynamically-linked Code and Self-modifying Code [NYY]3431.48 Ubrary Signature [NSQ]3431.49 Unanticipated Exceptions from Library Routines [HJW]343 <tr< td=""><td>I.16 Arithmetic Wrap-around Error [FIF]</td><td>331</td></tr<>	I.16 Arithmetic Wrap-around Error [FIF]	331
1.18 Sign Extension Error [X2]	I.17 Using Shift Operations for Multiplication and Division [PIK]	332
L19 Choice of Clear Names [NAI]332L20 Dead Store [WXQ]333L21 Unused Variable [YZS]333L22 Identifier Name Reuse [YOW]333L23 Namespace Issues [BIL]334L24 Initialization of Variables [LAV]334L25 Operator Precedence/Order of Evaluation [JCW]334L26 Side-effects and Order of Evaluation [SAM]335L27 Likely Incorrect Expression [KOA]335L29 Switch Statements and Static Analysis [CLL]336L30 Demarcation of Control Flow [EOJ]336L31 Loop Control Variables [TEX]337L32 Off-by-one Error [XZH]337L32 Off-by-one Error [XZH]337L35 Dangling References to Stack Frames [DCM]338L34 Passing Parameters and Return Values [CSJ]338L35 Dangling References to Stack Frames [DCM]339L36 Suborgram Signature Mismatch [DTR]339L37 Recursion [GDL]340L40 Type-breaking References to Stack Frames [DCM]341L41 Interinsics [LRM]341L42 Templates and Generics [SYM]341L44 Extra Intrinsics [LRM]341L44 Extra Intrinsics [LCM]342L45 Argument Passing to Ubrary Functions [TRJ]342L45 Argument Passing to Ubrary Functions [TRJ]342L45 Argument Passing to Ubrary Functions [TRJ]342L46 Inter-language Calling [DJ5]342L45 Argument Passing to Ubrary Functions [TRJ]343L45 Dynamically-linked Code and Self-modifying Code [NYY]343L45 Argument Passing to Ubrary Function	I.18 Sign Extension Error [XZI]	332
120 Dead Store [WXQ]333121 Unused Variable [YZS]333122 Identifier Name Reuse [YOW]333123 Namespace Issues [BLI]334124 Initialization of Variables [LAV]334125 Operator Precedence/Order of Evaluation [JCW]334126 Side-effects and Order of Evaluation [SAM]335127 Likely Incorrect Expression [KOA]335127 Likely Incorrect Expression [KOA]336129 Switch Statements and Static Analysis [CLL]336120 Demarcation of Control Flow [EO]336131 Loop Control Variables [TEX]337132 Off-by-one Error [XZH]337133 Structured Programming [EWD]338134 Passing Parameters and Return Values [CSJ]338135 Dangling References to Stack Frames [DCM]339136 Subprogram Signature Mismatch [OTR]339137 Recursion [GDL]340140 Type-breaking Reinterpretation of Data [AMV]341141 Memory Leak [XYL]341142 Templates and Generics [SYM]342144 Extra Intrinsics [LRM]342145 Argument Passing to Library Functions [TRJ]342146 Library Signature [NSQ]343150 Pre-processor Directives [NMP]343148 Library Signature [NSQ]343151 Suppression of Language-defined Run-time Checking [MXB]344152 Provision of Inherently Usafe Operations [SKL]344153 Oper-processor Directives [NMP]343154 Dispression of Language-defined Run-time Checking [MXB]344155 Operations of Inherently Usafe	I.19 Choice of Clear Names [NAI]	332
121 Unused Variable [YZ]333122 Identifier Name Reuse [YOW]333122 Identifier Name Reuse [YOW]334123 Namespace Issues [BU]334125 Operator Precedence/Order of Evaluation [JCW]334125 Operator Precedence/Order of Evaluation [JCW]334126 Side-effects and Order of Evaluation [SAM]335127 Likely Incorrect Expression [KOA]335128 Dead and Deactivated Code [XYQ]336129 Switch Statements and Static Analysis [CLL]336130 Demarcation of Control Flow [EOJ]336131 Loop Control Variables [TEX]337132 Off-by-one Error [XZH]337133 Structured Programming [EWD]338134 Passing Parameters and Return Values [CSJ]338135 Dangling References to Stack Frames [DCM]339138 Ignored Error Status and Unhandled Exceptions [OYB]340140 Type-breaking Reinterpretation of Data [AMV]341141 Memory Leak [XYL]341142 Templates and Generics [SYM]342143 Inheritance [RIP]341144 Extra Intrinsics [LRM]342145 Argument Passing to Library Functions [TRI]342146 Inter-language Calling [DJS]342148 Library Signature [NSQ]343149 Unanticipated Exceptions [OKB]343148 Library Signature [NSQ]343145 Dynamically-linked Code and Self-modifying Code [NYY]343145 Library Signature [NSQ]343145 Library Signature [NSQ]343145 Library Signature [NSQ]343	I.20 Dead Store [WXQ]	333
122 Identifier Name Reuse [YOW]333123 Namespace Issues [BJL]334124 Initialization of Variables [LAV]334125 Operator Precedence/Order of Evaluation [JCW]334126 Side-effects and Order of Evaluation [SAM]335127 Likely Incorrect Expression [KOA]335127 Likely Incorrect Expression [KOA]335128 Dead and Deactivated Code [XYQ]336129 Switch Statements and Static Analysis [CLI]336130 Demarcation of Control Flow [EO]336131 Loop Control Variables [TEX]337132 Off-by-one Error [XZH]337133 Structured Programming [EWD]338134 Passing Parameters and Return Values [CSJ]338135 Gsubprogram Signature Mismatch [OTR]339136 Subprogram Signature Mismatch [OTR]339137 Recursion [GOL]340139 Termination Strategy [REU]340140 Type-breaking Reinterpretation of Data [AMV]341141 L42 Templates and Generics [SYM]341142 Templates and Generics [SYM]342143 Inheritance [RIP]342144 Extra Intrinsics [LRM]342145 Argument Passing to Library Functions [TRJ]342146 Inter-language Calling [DIS]342147 Dynamically-linked Code and Self-modifying Code [NYY]343150 Pre-processor Directives [NMP]343151 Suppression of Language-defined Run-time Checking [MXB]344152 Provision of Inherenty Unsafe Operations [SKL]344153 Control States [Steperation]344154 Argumen	I.21 Unused Variable [YZS]	333
123 Namespace Issues [BIL]334124 Initialization of Variables [LAV]334125 Operator Precedence/Order of Evaluation [JCW]334126 Side-effects and Order of Evaluation [SAM]335127 Likely Incorrect Expression [KOA]336129 Switch Statements and Static Analysis [CLL]336120 Demarcation of Control Flow [EOJ]336131 Loop Control Variables [TEX]337132 Off-by-one Error [XZH]337133 Structured Programming [EWD]338134 Passing Parameters and Return Values [CSJ]338135 Dangling References to Stack Frames [DCM]339136 Subprogram Signature Mismatch [OTR]339137 Recursion [GDL]340140 Type-breaking Reinterpretation of Data [AMV]341141 Memory Leak [XYL]341143 Inheritance [RIP]341144 Extra Intrinsics [LRM]342144 Extra Intrinsics [LRM]342144 Extra Intrinsics [LRM]342145 Argument Passing to Library Functions [TRJ]342144 Extra Intrinsics [LRM]343145 Dre-processor Directives [NMP]343150 Dre-processor Directives [NMP]343151 Suppression of Language-defined Run-time Checking [MXB]344152 Provision of Inherenty Unsafe Operations [SKL]344153 Provision of Inherenty Unsafe Operations [SKL]344154 Operation States [NMP]343155 Operation of Language-defined Run-time Checking [MXB]344155 Operation of Language-defined Run-time Checking [MXB]344 <td>I.22 Identifier Name Reuse [YOW]</td> <td>333</td>	I.22 Identifier Name Reuse [YOW]	333
124 Initialization of Variables [LAV]334125 Operator Precedence/Order of Evaluation [JCW]334126 Side-effects and Order of Evaluation [SAM]335127 Likely Incorrect Expression [KOA]335128 Dead and Deactivated Code [XYQ]336129 Switch Statements and Static Analysis [CLL]336130 Demarcation of Control Flow [EOJ]336131 Loop Control Variables [TEX]337132 Off-by-one Error [XZH]337133 Structured Programming [EWD]338134 Passing Parameters and Return Values [CSJ]338135 Dangling References to Stack Frames [DCM]339136 Subprogram Signature Mismatch [OTR]339137 Recursion [GDL]340139 Iremination Strategy [REU]340140 Type-breaking Reinterpretation of Data [AMV]341141 Memory Leak [XYL]341143 Inheritance [RIP]341144 Extra Intrinsics [LRM]342145 Argument Passing to Library Functions [TRJ]342146 Inter-language Calling [DIS]342147 Dynamically-linked Code and Self-modifying Code [NYY]343148 Library Signature [NSQ]343151 Suppression of Language-defined Run-time Checking [MXB]344152 Prevision of Language-defined Run-time Checking [MXB]344153 Chernery Signature Nose [SKL]344	I.23 Namespace Issues [BJL]	334
L25 Operator Precedence/Order of Evaluation [JCW]334L26 Side-effects and Order of Evaluation [SAM]335L27 Likely Incorrect Expression [KOA]335L28 Dead and Deactivated Code [XYQ]336L29 Switch Statements and Static Analysis [CLL]336L30 Demarcation of Control Flow [EO]336L31 Loop Control Variables [TEX]337L32 Off-by-one Error [XZH]337L33 Structured Programming [EWD]338L34 Passing Parameters and Return Values [CS]338L35 Dangling References to Stack Frames [DCM]339L36 Subprogram Signature Mismatch [OTR]339L37 Recursion [GDL]340L39 Termination Strategy [REU]340L40 Type-breaking Reinterpretation of Data [AMV]341L41 Memory Leak [XYL]341L42 Templates and Generics [SYM]341L44 Extra Intrinsics [LRM]342L45 Argument Passing to Library Functions [TRJ]342L45 Dynamically-linked Code and Self-modifying Code [NYY]343L45 Dreprocessor Directives [NMP]343L50 Pre-processor Directives [NMP]343L51 Suppression of Language-defined Run-time Checking [MXB]344L52 Provision of Inherently Unsafe Operations [SKL]344	I.24 Initialization of Variables [LAV]	334
1_26 Side-effects and Order of Evaluation [SAM]3351_27 Likely Incorrect Expression [KOA]3351_28 Dead and Deactivated Code [XYQ]3361_29 Switch Statements and Static Analysis [CLL]3361_30 Demarcation of Control Flow [E0]3361_31 Loop Control Variables [TEX]3371_32 Off-by-one Error [XZH]3371_33 Structured Programming [EWD]3381_34 Passing Parameters and Return Values [CS]3381_35 Structured Program Signature Mismatch [OTR]3391_36 Subprogram Signature Mismatch [OTR]3391_37 Recursion [GDL]3401_38 Ignored Error Status and Unhandled Exceptions [OYB]3401_40 Type-breaking Reinterpretation of Data [AMV]3411_41 Memory Leak [XYL]3411_42 Templates and Generics [SYM]3421_44 Extra Intrinsics [LRM]3421_45 Argument Passing to Library Functions [TRJ]3421_46 Inter-language Calling [DJS]3431_47 Dynamically-linked Code and Self-modifying Code [NYY]3431_50 Pre-processor Directives [NMP]3431_50 Pre-processor Directives [NMP]3431_51 Suppression of Language-defined Run-time Checking [MXB]3441_52 Provision of Innerently Unsafe Operations [SKL]344	I.25 Operator Precedence/Order of Evaluation [JCW]	334
1.27 Likely Incorrect Expression [KOA]3351.28 Dead and Deactivated Code [XYQ]3361.29 Switch Statements and Static Analysis [CLL]3361.30 Demarcation of Control Flow [EOJ]3361.31 Loop Control Variables [TEX]3371.32 Off-by-one Error [XZH]3381.33 Structured Programming [EWD]3381.34 Passing Parameters and Return Values [CSJ]3381.35 Dangling References to Stack Frames [DCM]3391.36 Subprogram Signature Mismatch [OTR]3391.37 Recursion [GDL]3391.38 Ignored Error Status and Unhandled Exceptions [OYB]3401.39 Termination Strategy [REU]3401.40 Type-breaking Reinterpretation of Data [AMV]3411.41 Memory Leak [XYL]3411.42 Templates and Generics [SYM]3421.43 Inheritance [RIP]3421.44 Extra Intrinsics [LRM]3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-language Calling [DJS]3431.48 Library Signature [NSQ]3431.49 Unanticipated Exceptions from Library Routines [HJW]3431.51 Suppression of Language-defined Run-time Checking [MXB]3441.52 Prevision of Language-defined Run-time Checking [MXB]344	I.26 Side-effects and Order of Evaluation [SAM]	335
128 Dead and Deactivated Code [XYQ]336129 Switch Statements and Static Analysis [CLL]336130 Demarcation of Control Flow [EOJ]336131 Loop Control Variables [TEX]337132 Off-by-one Error [XZH]337133 Structured Programming [EWD]338134 Passing Parameters and Return Values [CSJ]338135 Dangling References to Stack Frames [DCM]339136 Subprogram Signature Mismatch [OTR]339137 Recursion [GDL]339138 Ignored Error Status and Unhandled Exceptions [OYB]340140 Type-breaking Reinterpretation of Data [AMV]341141 L42 Templates and Generics [SYM]341143 Inheritance [RIP]342144 Extra Intrinsics [LRM]342145 Argument Passing to Library Functions [TRJ]342146 Inter-language Calling [DIS]343148 Library Signature [NSQ]343150 Pre-processor Directives [NMP]343151 Suppression of Language-defined Run-time Checking [MXB]344152 Ore-processor Directives [NMP]344152 Ore-processor Directives [NMP]344	I.27 Likely Incorrect Expression [KOA]	335
1.29 Switch Statements and Static Analysis [CLL]3361.30 Demarcation of Control Flow [EOJ]3361.31 Loop Control Variables [TEX]3371.32 Off-by-one Error [XZH]3371.33 Structured Programming [EWD]3381.34 Passing Parameters and Return Values [CSJ]3381.35 Dangling References to Stack Frames [DCM]3391.36 Subprogram Signature Mismatch [OTR]3391.37 Recursion [GDL]3391.38 Ignored Error Status and Unhandled Exceptions [OYB]3401.41 Memory Leak [XYL]3411.41 Memory Leak [XYL]3411.42 Templates and Generics [SYM]3411.43 Inheritance [RIP]3421.44 Extra Intrinsics [LRM]3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-language Calling [DJS]3431.48 Ubrary Signature [NSQ]3431.49 Unanticipated Exceptions from Library Routines [HJW]3431.51 Suppression of Language-defined Run-time Checking [MXB]3441.52 Provision of Inherently Unsafe Operations [SKL]344	1.28 Dead and Deactivated Code [XYQ]	
130 Demarcation of Control How [E0J]336131 Loop Control Variables [TEX]337132 Off-by-one Error [XZH]337133 Structured Programming [EWD]338134 Passing Parameters and Return Values [CSJ]338135 Dangling References to Stack Frames [DCM]339136 Subprogram Signature Mismatch [OTR]339137 Recursion [GDL]339138 Ignored Error Status and Unhandled Exceptions [OYB]340140 Type-breaking Reinterpretation of Data [AMV]341141 Memory Leak [XYL]341142 Templates and Generics [SYM]341143 Inheritance [RIP]342144 Extra Intrinsics [LRM]342145 Argument Passing to Library Functions [TRJ]342146 Inter-language Calling [DJS]343148 Ubrary Signature [NSQ]343149 Unanticipated Exceptions from Library Routines [HJW]343151 Suppression of Language-defined Run-time Checking [MXB]344152 Orver Jone (RDC)344	1.29 Switch Statements and Static Analysis [CLL]	
1.31 Loop Control Variables [TEX]	1.30 Demarcation of Control Flow [EOJ]	
1.32 Off-by-one Error [X2H]	1.31 Loop Control Variables [TEX]	337
1.33 Structured Programming [EWU]	1.32 Off-by-one Error [X2H]	
1.34 Passing Parameters and Return Values [CSJ]3381.35 Dangling References to Stack Frames [DCM]3391.36 Subprogram Signature Mismatch [OTR]3391.37 Recursion [GDL]3391.38 Ignored Error Status and Unhandled Exceptions [OYB]3401.39 Termination Strategy [REU]3401.40 Type-breaking Reinterpretation of Data [AMV]3411.41 Memory Leak [XYL]3411.42 Templates and Generics [SYM]3411.43 Inheritance [RIP]3421.44 Extra Intrinsics [LRM]3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-language Calling [DJS]3431.48 Library Signature [NSQ]3431.50 Pre-processor Directives [NMP]3431.51 Suppression of Language-defined Run-time Checking [MXB]3441.52 Provision of Inherently Unsafe Operations [SKL]344	1.33 Structured Programming [EWD]	
1.35 Danging References to stack Frames (DCM)3391.36 Subprogram Signature Mismatch [OTR]3391.37 Recursion [GDL]3391.38 Ignored Error Status and Unhandled Exceptions [OYB]3401.39 Termination Strategy [REU]3401.40 Type-breaking Reinterpretation of Data [AMV]3411.41 Memory Leak [XYL]3411.42 Templates and Generics [SYM]3411.43 Inheritance [RIP]3411.44 Extra Intrinsics [LRM]3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-language Calling [DJS]3431.48 Library Signature [NSQ]3431.49 Unanticipated Exceptions from Library Routines [HJW]3431.50 Pre-processor Directives [NMP]3431.51 Suppression of Language-defined Run-time Checking [MXB]3441.52 Provision of Inherently Unsafe Operations [SKL]344	1.34 Passing Parameters and Return Values [CSJ]	
1.36 Subprogram Signature Mismatch [UTK]3391.37 Recursion [GDL]3391.38 Ignored Error Status and Unhandled Exceptions [OYB]3401.39 Termination Strategy [REU]3401.40 Type-breaking Reinterpretation of Data [AMV]3411.41 Memory Leak [XYL]3411.42 Templates and Generics [SYM]3411.43 Inheritance [RIP]3411.44 Extra Intrinsics [LRM]3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-language Calling [DJS]3421.47 Dynamically-linked Code and Self-modifying Code [NYY]3431.49 Unanticipated Exceptions from Library Routines [HJW]3431.50 Pre-processor Directives [NMP]3431.51 Suppression of Language-defined Run-time Checking [MXB]3441.52 Provision of Inherently Unsafe Operations [SKL]344	1.35 Dangling References to Stack Frames [DCIVI]	
1.37 Recursion [GDL]3391.38 Ignored Error Status and Unhandled Exceptions [OYB]3401.39 Termination Strategy [REU]3401.40 Type-breaking Reinterpretation of Data [AMV]3411.41 Memory Leak [XYL]3411.42 Templates and Generics [SYM]3411.43 Inheritance [RIP]3411.44 Extra Intrinsics [LRM]3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-language Calling [DJS]3421.47 Dynamically-linked Code and Self-modifying Code [NYY]3431.49 Unanticipated Exceptions from Library Routines [HJW]3431.50 Pre-processor Directives [NMP]3431.51 Suppression of Language-defined Run-time Checking [MXB]3441.52 Provision of Inherently Unsafe Operations [SKL]344	1.36 Subprogram Signature Mismatch [OTK]	
1.39 Termination Strategy [REU]3401.39 Termination Strategy [REU]3401.40 Type-breaking Reinterpretation of Data [AMV]3411.41 Memory Leak [XYL]3411.42 Templates and Generics [SYM]3411.43 Inheritance [RIP]3411.44 Extra Intrinsics [LRM]3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-language Calling [DJS]3421.47 Dynamically-linked Code and Self-modifying Code [NYY]3431.48 Library Signature [NSQ]3431.50 Pre-processor Directives [NMP]3431.51 Suppression of Language-defined Run-time Checking [MXB]3441.52 Provision of Inherently Unsafe Operations [SKL]344	1.37 Recursion [GDL]	
1.39 Termination Strategy [REO]	1.38 Ignored Error Status and Onnandied Exceptions [OYB]	
1.40 Type-Dreaking keinterpretation of Data [AWV]3411.41 Memory Leak [XYL]3411.42 Templates and Generics [SYM]3411.43 Inheritance [RIP]3411.44 Extra Intrinsics [LRM]3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-language Calling [DJS]3421.47 Dynamically-linked Code and Self-modifying Code [NYY]3431.48 Library Signature [NSQ]3431.49 Unanticipated Exceptions from Library Routines [HJW]3431.50 Pre-processor Directives [NMP]3431.51 Suppression of Language-defined Run-time Checking [MXB]3441.52 Provision of Inherently Unsafe Operations [SKL]344	1.39 Termination Strategy [REO]	
1.41 Memory Leak [XI12]3411.42 Templates and Generics [SYM].3411.43 Inheritance [RIP].3421.44 Extra Intrinsics [LRM].3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-language Calling [DJS]3421.47 Dynamically-linked Code and Self-modifying Code [NYY]3431.48 Library Signature [NSQ]3431.49 Unanticipated Exceptions from Library Routines [HJW]3431.50 Pre-processor Directives [NMP]3431.51 Suppression of Language-defined Run-time Checking [MXB]3441.52 Provision of Inherently Unsafe Operations [SKL]3441.52 Observed Reserved Exceptions [NDC]345		2/1
1.42 Templates and Generics [51W]3411.43 Inheritance [RIP]3411.44 Extra Intrinsics [LRM]3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-language Calling [DJS]3421.47 Dynamically-linked Code and Self-modifying Code [NYY]3431.48 Library Signature [NSQ]3431.49 Unanticipated Exceptions from Library Routines [HJW]3431.50 Pre-processor Directives [NMP]3431.51 Suppression of Language-defined Run-time Checking [MXB]3441.52 Provision of Inherently Unsafe Operations [SKL]344	1.41 Memory Leak [XTL]	2/1
1.49 Intertaitee [ttit]3421.44 Extra Intrinsics [LRM]3421.45 Argument Passing to Library Functions [TRJ]3421.46 Inter-language Calling [DJS]3421.47 Dynamically-linked Code and Self-modifying Code [NYY]3431.48 Library Signature [NSQ]3431.49 Unanticipated Exceptions from Library Routines [HJW]3431.50 Pre-processor Directives [NMP]3431.51 Suppression of Language-defined Run-time Checking [MXB]3441.52 Provision of Inherently Unsafe Operations [SKL]344	1.42 Inheritance [RID]	2/1
1.45 Argument Passing to Library Functions [TRJ] 342 1.46 Inter-language Calling [DJS] 342 1.47 Dynamically-linked Code and Self-modifying Code [NYY] 343 1.48 Library Signature [NSQ] 343 1.49 Unanticipated Exceptions from Library Routines [HJW] 343 1.50 Pre-processor Directives [NMP] 343 1.51 Suppression of Language-defined Run-time Checking [MXB] 344 1.52 Provision of Inherently Unsafe Operations [SKL] 344	1.44 Extra Intrinsics [LBM]	342
1.45 Argument Passing to Ebolary Functions [FIG] 342 1.46 Inter-language Calling [DJS] 343 1.47 Dynamically-linked Code and Self-modifying Code [NYY] 343 1.48 Library Signature [NSQ] 343 1.49 Unanticipated Exceptions from Library Routines [HJW] 343 1.50 Pre-processor Directives [NMP] 343 1.51 Suppression of Language-defined Run-time Checking [MXB] 344 1.52 Provision of Inherently Unsafe Operations [SKL] 344	1.45 Argument Passing to Library Functions [TR]	342
1.40 Inter tangdage cannig [DJS] 342 1.47 Dynamically-linked Code and Self-modifying Code [NYY] 343 1.48 Library Signature [NSQ] 343 1.49 Unanticipated Exceptions from Library Routines [HJW] 343 1.50 Pre-processor Directives [NMP] 343 1.51 Suppression of Language-defined Run-time Checking [MXB] 344 1.52 Provision of Inherently Unsafe Operations [SKL] 344	1.46 Inter-Janguage Calling [DIS]	342
1.48 Library Signature [NSQ] 343 1.49 Unanticipated Exceptions from Library Routines [HJW] 343 1.50 Pre-processor Directives [NMP] 343 1.51 Suppression of Language-defined Run-time Checking [MXB] 344 1.52 Provision of Inherently Unsafe Operations [SKL] 344	1.47 Dynamically-linked Code and Self-modifying Code [NYY]	
1.49 Unanticipated Exceptions from Library Routines [HJW] 343 1.50 Pre-processor Directives [NMP] 343 1.51 Suppression of Language-defined Run-time Checking [MXB] 344 1.52 Provision of Inherently Unsafe Operations [SKL] 344	1.48 Library Signature [NSO]	
1.50 Pre-processor Directives [NMP] 343 1.51 Suppression of Language-defined Run-time Checking [MXB] 344 1.52 Provision of Inherently Unsafe Operations [SKL] 344	1.49 Unanticipated Exceptions from Library Routines [HJW]	
I.51 Suppression of Language-defined Run-time Checking [MXB] 344 I.52 Provision of Inherently Unsafe Operations [SKL] 344	I.50 Pre-processor Directives [NMP]	
I.52 Provision of Inherently Unsafe Operations [SKL]	I.51 Suppression of Language-defined Run-time Checking [MXB]	
	I.52 Provision of Inherently Unsafe Operations [SKL]	344
1.53 Obscure Language Features [BRS]	I.53 Obscure Language Features [BRS]	345

I.54 Unspecified Behaviour [BQF]	345
I.55 Undefined Behaviour [EWF]	345
I.56 Implementation-defined Behaviour [FAB]	
I.57 Deprecated Language Features [MEM]	
I.58 Implications for Standardization	
Bibliography	348
Index 251	

<u>Index</u> 351

Page vi: [2] Deleted Santiago Urueña 5/26/15 1:35:00 PM

Furthermore, to focus its limited resources, the working group developing this report decided to defer comprehensive treatment of several subject areas until future editions of the report. These subject areas include:

Object-oriented language features (although some simple issues related to inheritance are described in <u>6.43 Inheritance [RIP]</u>)

Numerical analysis (although some simple items regarding the use of floating point are described in <u>6.5</u> <u>Floating-point Arithmetic [PLF]</u>)

Inter-language operability

Page 1: [3] Moved to page 46 (Move #1)		(Move #1)	Stephe	en Michell		9/18/15 3:14:00	РМ
		1.0.1			. /	1/ /	

Achour, M. (n.d.). PHP Manual. Retrieved 3 5, 2012, from PHP: http://www.php.net/manual/en/

Brueggeman, E. (n.d.). Retrieved 3 5, 2012, from The Website of Elliott Brueggeman : http://www.ebrueggeman.com/blog/integers-and-floating-numbers

Enums for Python (Python recipe). (n.d.). Retrieved from ActiveState: http://code.activestate.com/recipes/67107/

- Goleman, S. (n.d.). *Extension Writing Part I: Introduction to PHP and Zend*. Retrieved 5 5, 12, from Zend Developer Zone: http://devzone.zend.com/303/extension-writing-part-i-introduction-to-php-and-zend/
- Isaac, A. G. (2010, 06 23). *Python Introduction*. Retrieved 05 12, 2011, from https://subversion.american.edu/aisaac/notes/python4class.xhtml#introduction-to-the-interpreter
- Lutz, M. (2009). Learning Python. Sebastopol, CA: O'Reilly Media, Inc.
- Lutz, M. (2011). Programming Python. Sebastopol, CA: O'Reilly Media, Inc.
- Martelli, A. (2006). Python in a Nutshell. Sebastopol, CA: O'Reilly Media, Inc.
- Norwak, H. (n.d.). *10 Python Pitfalls*. Retrieved 05 13, 2011, from 10 Python Pitfalls: http://zephyrfalcon.org/labs/python_pitfalls.html
- Pilgrim, M. (2004). Dive Into Python.
- Python Gotchas. (n.d.). Retrieved from http://www.ferg.org/projects/python_gotchas.html

source, G. (n.d.). *Big List of Portabilty in Python*. Retrieved 6 12, 2011, from stackoverflow: http://stackoverflow.com/questions/1883118/big-list-of-portability-in-python

The Python Language Reference. (n.d.). Retrieved from python.org: http://docs.python.org/reference/index.html#reference-index

Page 5: [4] Deleted	Stephen Michell	9/22/17 9:29:00 AM
· · · · · · · · · · · · · · · · · · ·		-,,

Section 6.42 Violations of the Liskov Substitution Principle or the Contract Model [BLP]

6.42.1 Applicability to language

TBD

6.42.2 Guidance to language users

TBD

6.43 Redispatching [PPH]

6.43.1 Applicability to language

TBD

6.43.2 Guidance to language users

TBD

6.44 Polymorphic variables [BKK]

6.44.1 Applicability to language

TBD

6.44.2 Guidance to language users

TBD

6.45 Extra Intrinsics [LRM]6.43 Extra Intrinsics [LRM]

Page 7: [5]	Deleted Stephen Michell	9/22/17 9:38:00 AM
19	Avoid fall-through from one case (or switch) statement into the following case statement: if a fall-through is necessary then provide a comment to inform the reader that it is intentional.	

Page 14: [6] Deleted	Stephen Michell	6/25/15 4:35:00 AM

6.17E.18 Sign Extension Error [XZI]

This vulnerability is not applicable to Python because Python converts between types without ever extending the sign.

Page 32: [7] Deleted Stephen Michell 3/7/17 11:09:00 Al	je 32: [7] Deleted
---	--------------------

6.38E.39 Termination Strategy [REU]

6.38E.39.1 Applicability to language

Python has a rich set of exception handling statements which can be utilized to implement a termination strategy that assures the best possible outcome ranging from a hard stop to a clean-up and fail soft strategy. Refer to 6.37 Ignored Error Status and Unhandled Exceptions [OYB]E.38 for an example of an implementation that cleans up and continues.

6.38E.39.2 Guidance to language users

Use Python's exception handling statements to implement an appropriate termination strategy.

Page 45: [8] Deleted	Stephen Michell	3/7/17 11:27:00 AM

Page 4	45: [9] Deleted Santiago Urueña	5/26/15 12:47:00 PM
[4]	ISO/IEC 9899:2011, Information technology — Programming languages — C	
[5]	ISO/IEC 9899:2011/Cor.1:2012, Technical Corrigendum 1	
[6]	ISO/IEC 30170:2012, Information technology — Programming languages — Ruby	
[7]	ISO/IEC/IEEE 60559:2011, Information technology – Microprocessor Systems – Floatin	ng-Point arithmetic
[8]	ISO/IEC 1539-1:2010, Information technology — Programming languages — Fortran language	— Part 1: Base
[9]	ISO/IEC 8652:1995, Information technology — Programming languages — Ada	
[10]	ISO/IEC 14882:2011, Information technology — Programming languages — C++	
[11]	R. Seacord, The CERT C Secure Coding Standard. Boston, MA: Addison-Westley, 2008.	
[12]	Motor Industry Software Reliability Association. <i>Guidelines for the Use of the C Langu Software</i> , 2012 (third edition). ¹ .	ıage in Vehicle Based
[13]	ISO/IEC TR24731–1, Information technology — Programming languages, their enviro software interfaces — Extensions to the C library — Part 1: Bounds-checking interface	nments and system es

¹ The first edition should not be used or quoted in this work.

- [14] ISO/IEC TR 15942:2000, Information technology Programming languages Guide for the use of the Ada programming language in high integrity systems
- [15] Joint Strike Fighter Air Vehicle: C++ Coding Standards for the System Development and Demonstration Program. Lockheed Martin Corporation. December 2005.
- [16] Motor Industry Software Reliability Association. *Guidelines for the Use of the C++ Language in critical systems*, June 2008
- [17] ISO/IEC TR 24718: 2005, Information technology Programming languages Guide for the use of the Ada Ravenscar Profile in high integrity systems
- [18] L. Hatton, Safer C: developing software for high-integrity and safety-critical systems. McGraw-Hill 1995
- [19] ISO/IEC 15291:1999, Information technology Programming languages Ada Semantic Interface Specification (ASIS)
- [20] Software Considerations in Airborne Systems and Equipment Certification. Issued in the USA by the Requirements and Technical Concepts for Aviation (document RTCA SC167/DO-178B) and in Europe by the European Organization for Civil Aviation Electronics (EUROCAE document ED-12B).December 1992.
- [21] IEC 61508: Parts 1-7, Functional safety: safety-related systems. 1998. (Part 3 is concerned with software).
- [22] ISO/IEC 15408: 1999 Information technology. Security techniques. Evaluation criteria for IT security.
- [23] J Barnes, High Integrity Software the SPARK Approach to Safety and Security. Addison-Wesley. 2002.

Page 4	I5: [10] Deleted	Santiago Urueña	5/26/15 12:48:00 PM
[26]	ARIANE 5: Flight 501 Failure,	Report by the Inquiry Board, July 19, 1996	
	http://esamultimedia.esa.int	/docs/esa-x-1819eng.pdf	
[77]	Llagahaam Diahard A Canar	is ADI Bit Maninulation in C. Embodded System	as Dragramming Val 12 No 7

 [27] Hogaboom, Richard, A Generic API Bit Manipulation in C, Embedded Systems Programming, Vol 12, No 7, July 1999 <u>http://www.embedded.com/1999/9907/9907feat2.htm</u>

Page 46	6: [11] Deleted	Santiago Urueña	5/26/15 1:31:00 PM
[38]	GAO Report, 247094, Feb.	Patriot <i>Missile Defense: Software Problem Led to System Failure at Dhahi</i> 4, 1992, <u>http://archive.gao.gov/t2pbat6/145960.pdf</u>	ran, Saudi Arabia, B-
[39]	Robert Skeel, 11, <u>http://www</u>	Roundoff Error Cripples Patriot Missile, SIAM News, Volume 25, Number isiam.org/siamnews/general/patriot.htm	4, July 1992, page
[40]	CERT. CERT C- Standard. <u>htt</u>	++ Secure Coding ps://www.securecoding.cert.org/confluence/pages/viewpage.action?pa	geld=637 (2009).
[41]	Holzmann, Ga Safety-Critica	rrard J., Computer, vol. 39, no. 6, pp 95-97, Jun., 2006, <i>The Power of 10: I</i> I Code	Rules for Developing
[42]	P. V. Bhansali Software Eng	, A systematic approach to identifying a safe subset for safety-critical sof neering Notes, v.28 n.4, July 2003	tware, ACM SIGSOFT
[40]			e (i

 [43] Ada 95 Quality and Style Guide, SPC-91061-CMC, version 02.01.01. Herndon, Virginia: Software Productivity Consortium, 1992. Available from: <u>http://www.adaic.org/docs/95style/95style.pdf</u>

- [44] Ghassan, A., & Alkadi, I. (2003). Application of a Revised DIT Metric to Redesign an OO Design. *Journal of Object Technology*, 127-134.
- [45] Subramanian, S., Tsai, W.-T., & Rayadurgam, S. (1998). Design Constraint Violation Detection in Safety-Critical Systems. The 3rd IEEE International Symposium on High-Assurance Systems Engineering , 109 -116.

[46] Lundqvist, K and Asplund, L., "A Formal Model of a Run-Time Kernel for Ravenscar", The 6th International Conference on Real-Time Computing Systems and Applications – RTCSA 1999

Page 48: [12] Deleted	Santiago Urueña	5/26/15 12:38:00 PM
Section Break (Continuous)		
Ada, 13, 59, 63, 73, 76	bitwise operators, 48	
AMV – Type-breaking Reinterpretation of Data	, 72 BJE – Incorrect Authorizati	ion, 138
API	BJL – Namespace Issues, 4	3
Application Programming Interface, 16	black-list, 120, 124	
APL, 48	BQF – Unspecified Behavio	our, 92, 94, 95
Apple	break , 60	
OS X, 120	BRS – Obscure Language F	eatures, 91
application vulnerabilities, 9	buffer boundary violation,	23
Application Vulnerabilities	buffer overflow, 23, 26	
Adherence to Least Privilege [XYN], 113	buffer underwrite, 23	
Authentication Logic Error [XZO], 135	BVQ – Unspecified Functio	onality, 111
Cross-site Scripting [XYT], 125		
Discrepancy Information Leak [XZL], 129	C, 22, 48, 50, 51, 58, 60, 63	3, 73
Distinguished Values in Data Types [KLK], 112	C++, 48, 51, 58, 63, 73, 76,	, 86
Download of Code Without Integrity Check [DLB]], 137 C11, 192	
Executing or Loading Untrusted Code [XYS], 116	call by copy, 61	
Hard-coded Password [XYP], 136	call by name, 61	
Improper Restriction of Excessive Authentication	call by reference, 61	
Attempts [WPL], 140	call by result, 61	
Improperly Verified Signature [XZR], 128	call by value, 61	
Inclusion of Functionality from Untrusted Contro	call by value-result, 61	
Sphere [DHU]. 139	CBF – Unrestricted File Up	load, 119
Incorrect Authorization [BJE], 138	CCB – Enumerator Issues,	18
Injection [RST], 122	CGA – Concurrency – Activ	vation, 98
Insufficiently Protected Credentials [XYM], 133	CGM – Protocol Lock Error	s, 105
Memory Locking [XZX], 117	CGS – Concurrency – Prem	nature Termination, 103
Missing or Inconsistent Access Control [X7N] 13	CGT - Concurrency – Direc	ted termination, 100
Missing Required Cryptographic Step [XZS] 133	CGX – Concurrent Data Ac	cess, 101
Path Traversal [FWR] 130	CGY – Inadequately Secure	e communication of
Privilege Sandhox Issues [XVO] 114	CIM String Tormination	22
Resource Exhaustion [X7D] 118	CIVI – String Termination,	22 nd Static Analysis E4
Resource Names [HTS] 120		nu static Analysis, 54
Sonsitive Information Lineleared Refere Lise [V7k	d 120 continue 60	
Linguisted Soarch Path or Element [V70] 127	cryptologic 71 128	
Unrestricted File Unlead [CDF] 110	CSI - Dassing Darameters	and Return Values 61 82
Unexpecticed File Opload [CBF], 119		
Unspecified Functionality [BVQ], 111	dangling reference 31	
	DCM – Dangling Reference, SI	os to Stack Frames 63
	1 Deactivated code 53	15 to Stack Frames, 05
Use of a Une-way Hash without a Salt [MVX], 14	Dead code 53	
application vulnerability, 5	deadlock 106	
Ariane 5, 21		

DHU - Inclusion of Functionality from Untrusted Control Sphere, 139 Diffie-Hellman-style, 136 digital signature, 84 DJS – Inter-language Calling, 81 DLB – Download of Code Without Integrity Check, 137 DoS Denial of Service, 118 dynamically linked, 83 EFS – Use of unchecked data from an uncontrolled or tainted source, 109 encryption, 128, 133 endian big, 15 little, 15 endianness, 14 Enumerations, 18 EOJ – Demarcation of Control Flow, 56 EWD – Structured Programming, 60 EWF – Undefined Behaviour, 92, 94, 95 EWR - Path Traversal, 124, 130 exception handler, 86 FAB – Implementation-defined Behaviour, 92, 94, 95

FIF – Arithmetic Wrap-around Error, 34, 35 FLC – Numeric Conversion Errors, 20

GDL – Recursion, 67 generics, 76 GIF, 120 goto, 60

Fortran, 73

HCB – Buffer Boundary Violation (Buffer Overflow), 23, 82
HFC – Pointer Casting and Pointer Type Changes, 28
HJW – Unanticipated Exceptions from Library Routines, 86 *HTML*Hyper Text Markup Language, 124
HTS – Resource Names, 120 *HTTP*Hypertext Transfer Protocol, 127

IEC 60559, 16 IEEE 754, 16 IHN –Type System, 12 inheritance, 78 IP address, 119

Java, 18, 50, 52, 76 JavaScript, 125, 126, 127 JCW – Operator Precedence/Order of Evaluation, 47 KLK – Distinguished Values in Data Types, 112 KOA - Likely Incorrect Expression, 50 language vulnerabilities, 9 Language Vulnerabilities Argument Passing to Library Functions [TRJ], 80 Arithmetic Wrap-around Error [FIF], 34 Bit Representations [STR], 14 Buffer Boundary Violation (Buffer Overflow) [HCB], 23 Choice of Clear Names [NAI], 37 Concurrency - Activation [CGA], 98 Concurrency - Directed termination [CGT], 100 Concurrency - Premature Termination [CGS], 103 Concurrent Data Access [CGX], 101 Dangling Reference to Heap [XYK], 31 Dangling References to Stack Frames [DCM], 63 Dead and Deactivated Code [XYQ], 52 Dead Store [WXQ], 39 Demarcation of Control Flow [EOJ], 56 Deprecated Language Features [MEM], 97 Dynamically-linked Code and Self-modifying Code [NYY], 83 Enumerator Issues [CCB], 18 Extra Intrinsics [LRM], 79 Floating-point Arithmetic [PLF], xvii, 16 Identifier Name Reuse [YOW], 41 Ignored Error Status and Unhandled Exceptions [OYB], 68 Implementation-defined Behaviour [FAB], 95 Inadequately Secure Communication of Shared Resources [CGY], 107 Inheritance [RIP], 78 Initialization of Variables [LAV], 45 Inter-language Calling [DJS], 81 Library Signature [NSQ], 84 Likely Incorrect Expression [KOA], 50 Loop Control Variables [TEX], 57 Memory Leak [XYL], 74 Namespace Issues [BJL], 43 Null Pointer Dereference [XYH], 30 Numeric Conversion Errors [FLC], 20 Obscure Language Features [BRS], 91 Off-by-one Error [XZH], 58 Operator Precedence/Order of Evaluation [JCW], 47 Passing Parameters and Return Values [CSJ], 61, 82 Pointer Arithmetic [RVG], 29 Pointer Casting and Pointer Type Changes [HFC], 28 Pre-processor Directives [NMP], 87 Protocol Lock Errors [CGM], 105 Provision of Inherently Unsafe Operations [SKL], 90 Recursion [GDL], 67 Side-effects and Order of Evaluation [SAM], 49 Sign Extension Error [XZI], 36

String Termination [CJM], 22 Structured Programming [EWD], 60 Subprogram Signature Mismatch [OTR], 65 Suppression of Language-defined Run-time Checking [MXB], 89 Switch Statements and Static Analysis [CLL], 54 Templates and Generics [SYM], 76 Termination Strategy [REU], 70 Type System [IHN], 12 Type-breaking Reinterpretation of Data [AMV], 72 Unanticipated Exceptions from Library Routines [HJW], 86 Unchecked Array Copying [XYW], 27 Unchecked Array Indexing [XYZ], 25 Uncontrolled Fromat String [SHL], 110 Undefined Behaviour [EWF], 94 Unspecified Behaviour [BFQ], 92 Unused Variable [YZS], 40 Use of unchecked data from an uncontrolled or tainted source [EFS], 109 Using Shift Operations for Multiplication and Division [PIK], 35 language vulnerability, 5 LAV – Initialization of Variables, 45 LHS (left-hand side), 241 Linux, 120 livelock, 106 longjmp,60 LRM - Extra Intrinsics, 79 MAC address, 119 macof. 118 MEM – Deprecated Language Features, 97 memory disclosure, 130 Microsoft Win16, 121 Windows, 117 Windows XP, 120 MIME Multipurpose Internet Mail Extensions, 124 MISRA C, 29 MISRA C++, 87 mlock(),117 MVX – Use of a One-Way Hash without a Salt, 141 MXB - Suppression of Language-defined Run-time Checking, 89 NAI – Choice of Clear Names, 37 name type equivalence, 12 NMP – Pre-Processor Directives, 87 NSQ – Library Signature, 84 NTFS New Technology File System, 120

NULL, 31, 58

NULL pointer, 31

symlink, 131 *tail-recursion*, 68 templates, 76, 77 TEX – Loop Control Variables, 57 **thread**, 2 TRJ – Argument Passing to Library Functions, 80 *type casts*, 20

null-pointer, 30

Code, 83

Pascal, 82

PHP, 124

POSIX, 99

pragmas, 75, 96

Redirect'), 140

Real-Time Java, 105

resource exhaustion, 118

RIP – Inheritance, xvii, 78

RST - Injection, 109, 122

safety-critical software, 5

security vulnerability, 5

SelmpersonatePrivilege, 115

SHL – Uncontrolled Format String, 110

REU – Termination Strategy, 70

runtime-constraint handler, 191

RVG - Pointer Arithmetic, 29

real numbers, 16

rsize t,22

safety hazard, 4

setjmp,60

size t,22

strcpy,23

strncpy, 23

switch, 54

SQL

software quality, 4

software vulnerabilities, 9

Structured Query Language, 112

STR – Bit Representations, 14

structure type equivalence, 12

SYM – Templates and Generics, 76

Exceptions, 68, 163

Division, 34, 35, 197

predictable execution, 4, 8

NYY - Dynamically-linked Code and Self-modifying

OTR – Subprogram Signature Mismatch, 65, 82

PIK – Using Shift Operations for Multiplication and

PYQ – URL Redirection to Untrusted Site ('Open

SAM - Side-effects and Order of Evaluation, 49

SKL – Provision of Inherently Unsafe Operations, 90

OYB - Ignored Error Status and Unhandled

PLF – Floating-point Arithmetic, xvii, 16
type coercion, 20 type safe, 12 type secure, 12 type system, 12

UNC

Uniform Naming Convention, 131 Universal Naming Convention, 131 Unchecked_Conversion, 73 UNIX, 83, 114, 120, 131 unspecified functionality, 111 Unspecified functionality, 111 URI Uniform Resource Identifier, 127 URL Uniform Resource Locator, 127

VirtualLock(),117

white-list, 120, 124, 127 Windows, 99 WPL – Improper Restriction of Excessive Authentication Attempts, 140 WXQ – Dead Store, 39, 40, 41

XSS

Cross-site scripting, 125 XYH - Null Pointer Deference, 30 XYK – Dangling Reference to Heap, 31 XYL – Memory Leak, 74 XYM – Insufficiently Protected Credentials, 9, 133 XYN –Adherence to Least Privilege, 113 XYO – Privilege Sandbox Issues, 114 XYP – Hard-coded Password, 136 XYQ - Dead and Deactivated Code, 52 XYS – Executing or Loading Untrusted Code, 116 XYT – Cross-site Scripting, 125 XYW – Unchecked Array Copying, 27 XYZ – Unchecked Array Indexing, 25, 28 XZH – Off-by-one Error, 58 XZI - Sign Extension Error, 36 XZK – Senitive Information Uncleared Before Use, 130 XZL – Discrepancy Information Leak, 129 XZN - Missing or Inconsistent Access Control, 134 XZO – Authentication Logic Error, 135 XZP – Resource Exhaustion, 118 XZQ – Unquoted Search Path or Element, 127 XZR – Improperly Verified Signature, 128 XZS – Missing Required Cryptographic Step, 133 XZX – Memory Locking, 117 YOW – Identifier Name Reuse, 41, 44 YZS – Unused Variable, 39, 40

Section Break (Next Page)