

Inhaltsverzeichnis

Einleitung	13
Über den Autor	17
Danksagungen	17
1 Pythons Sicht der Dinge	19
Punkt 1 Kenntnis der Python-Version	19
Punkt 2 Stilregeln gemäß PEP 8	21
Punkt 3 Unterschiede zwischen bytes und str	23
Punkt 4 Interpolierte F-Strings statt C-Formatierungsstrings und str.format	29
Punkt 5 Hilfsfunktionen statt komplizierter Ausdrücke	40
Punkt 6 Mehrfachzuweisung beim Entpacken statt Indizierung	43
Punkt 7 enumerate statt range	47
Punkt 8 Gleichzeitige Verarbeitung von Iteratoren mit zip	49
Punkt 9 Verzicht auf else-Blöcke nach for- und while-Schleifen	52
Punkt 10 Wiederholungen verhindern durch Zuweisungsausdrücke	55
2 Listen und Dictionaries	63
Punkt 11 Zugriff auf Listen durch Slicing	63
Punkt 12 Verzicht auf Abstand und Start-/Ende-Index in einem einzigen Ausdruck	67
Punkt 13 Vollständiges Entpacken statt Slicing	69
Punkt 14 Sortieren nach komplexen Kriterien mit dem key-Parameter	73
Punkt 15 Vorsicht bei der Reihenfolge von dict-Einträgen	80
Punkt 16 get statt in und KeyError zur Handhabung fehlender Dictionary-Schlüssel verwenden	88
Punkt 17 defaultdict statt setdefault zur Handhabung fehlender Objekte verwenden	94
Punkt 18 Schlüsselabhängige Standardwerte mit __missing__ erzeugen	97

3	Funktionen	101
Punkt 19	Bei Funktionen mit mehreren Rückgabewerten nicht mehr als drei Variablen entpacken	101
Punkt 20	Exceptions statt Rückgabe von None	104
Punkt 21	Closures und der Gültigkeitsbereich von Variablen	107
Punkt 22	Klare Struktur dank variabler Anzahl von Positionsargumenten	112
Punkt 23	Optionale Funktionalität durch Schlüsselwort-Argumente	115
Punkt 24	Dynamische Standardwerte von Argumenten mittels None und Docstrings	120
Punkt 25	Eindeutigkeit durch reine Schlüsselwort- und reine Positionsargumente	123
Punkt 26	Funktions-Decorators mit <code>functools.wraps</code> definieren	129
4	Listen-Abstraktionen und Generatoren	133
Punkt 27	Listen-Abstraktionen statt <code>map</code> und <code>filter</code>	133
Punkt 28	Nicht mehr als zwei Ausdrücke in Listen-Abstraktionen	135
Punkt 29	Doppelte Arbeit in Abstraktionen durch Zuweisungsausdrücke vermeiden	137
Punkt 30	Generatoren statt Rückgabe von Listen	141
Punkt 31	Vorsicht beim Iterieren über Argumente	144
Punkt 32	Generatorausdrücke für umfangreiche Listen-Abstraktionen verwenden	150
Punkt 33	Mehrere Generatoren mit <code>yield from</code> verknüpfen	152
Punkt 34	Keine Datenübermittlung an Generatoren mit <code>send</code>	155
Punkt 35	Zustandsübergänge in Generatoren mit <code>throw</code> vermeiden	162
Punkt 36	Iteratoren und Generatoren mit <code>itertools</code> verwenden	166
5	Klassen und Schnittstellen	173
Punkt 37	Klassen statt integrierte Datentypen verschachteln	173
Punkt 38	Funktionen statt Klassen bei einfachen Schnittstellen	180
Punkt 39	Polymorphismus und generische Erzeugung von Objekten	184
Punkt 40	Initialisierung von Basisklassen durch <code>super</code>	190
Punkt 41	Verknüpfte Funktionalität mit Mix-in-Klassen	195
Punkt 42	Öffentliche statt private Attribute	201
Punkt 43	Benutzerdefinierte Container-Klassen durch Erben von <code>collections.abc</code>	207

6	Metaklassen und Attribute	213
Punkt 44	Einfache Attribute statt Getter- und Setter-Methoden	213
Punkt 45	@property statt Refactoring von Attributen	218
Punkt 46	Deskriptoren für wiederverwendbare @property-Methoden verwenden	223
Punkt 47	Verzögerte Zuweisung zu Attributen mittels <code>__getattribute__</code> , <code>__getattr__</code> und <code>__setattr__</code>	229
Punkt 48	Unterklassen mit <code>__init_subclass__</code> überprüfen	235
Punkt 49	Klassen mittels <code>__init_subclass__</code> registrieren	244
Punkt 50	Zugriff auf Klassenattribute mit <code>__set_name__</code>	250
Punkt 51	Klassen-Decorators statt Metaklassen für Klassen-erweiterungen nutzen	255
7	Nebenläufigkeit und parallele Ausführung	263
Punkt 52	Verwaltung von Kindprozessen mittels <code>subprocess</code>	264
Punkt 53	Threads, blockierende Ein-/Ausgabevorgänge und parallele Ausführung	269
Punkt 54	Wettlaufsituationen in Threads mit Lock verhindern	274
Punkt 55	Threads koordinieren mit Queue	278
Punkt 56	Erkennen, wann parallele Ausführung erforderlich ist	288
Punkt 57	Keine neuen Thread-Instanzen beim Fan-Out	294
Punkt 58	Die Verwendung von Queue zur parallelen Ausführung erfordert Refactoring	298
Punkt 59	<code>ThreadPoolExecutor</code> zur parallelen Ausführung verwenden	305
Punkt 60	Parallele Ausführung mehrerer Funktionen mit Coroutinen	309
Punkt 61	<code>asyncio</code> statt <code>threaded</code> I/O	314
Punkt 62	Threads und Coroutinen verwenden, um die Umstellung auf <code>asyncio</code> zu erleichtern	326
Punkt 63	<code>asyncio</code> -Event-Loop nicht blockieren	333
Punkt 64	Echte parallele Ausführung mit <code>concurrent.futures</code>	337
8	Robustheit und Performance	343
Punkt 65	Alle Blöcke einer <code>try/except/else/finally</code> -Anweisung nutzen	343
Punkt 66	<code>contextlib</code> und <code>with</code> -Anweisung für wiederverwendbares <code>try/finally</code> -Verhalten verwenden	349
Punkt 67	Für örtliche Zeitangaben <code>datetime</code> statt <code>time</code> verwenden	353
Punkt 68	Verlässliches <code>pickle</code> durch <code>copyreg</code>	358

Punkt 69	Falls Genauigkeit an erster Stelle steht, decimal verwenden	366
Punkt 70	Vor der Optimierung Messungen vornehmen	369
Punkt 71	deque für Erzeuger-Verbraucher-Warteschlangen verwenden	374
Punkt 72	Durchsuchen von Sequenzen geordneter Elemente mit bisect	382
Punkt 73	heapq für Prioritätswarteschlangen verwenden	385
Punkt 74	memoryview und bytearray für Zero-Copy-Interaktionen mit bytes verwenden	396
9	Testen und Debuggen	403
Punkt 75	Debuggen mit repr-Strings	404
Punkt 76	Ähnliches Verhalten mit TestCase-Unterklassen überprüfen	408
Punkt 77	Tests mit setUp, tearDown, setUpModule und tearDownModule voneinander abschotten	415
Punkt 78	Mocks für Tests von Code mit komplexen Abhängigkeiten verwenden	418
Punkt 79	Abhängigkeiten kapseln, um das Erstellen von Mocks und Tests zu erleichtern	427
Punkt 80	Interaktives Debuggen mit pdb	432
Punkt 81	Nutzung des Arbeitsspeichers und Speicherlecks mit tracemalloc untersuchen	437
10	Zusammenarbeit	443
Punkt 82	Module der Python-Community	443
Punkt 83	Virtuelle Umgebungen zum Testen von Abhängigkeiten	444
Punkt 84	Docstrings für sämtliche Funktionen, Klassen und Module	451
Punkt 85	Pakete zur Organisation von Modulen und zur Bereitstellung stabiler APIs verwenden	457
Punkt 86	Modulweiter Code zur Konfiguration der Deployment-Umgebung	462
Punkt 87	Einrichten einer Root-Exception zur Abschottung von Aufrufen und APIs	465
Punkt 88	Zirkuläre Abhängigkeiten auflösen	470
Punkt 89	warnings für Refactoring und Migration verwenden	476
Punkt 90	Statische Analyse mit typing zum Vermeiden von Bugs	484
	Stichwortverzeichnis	495