

Inhaltsverzeichnis

	Vorwort	11
	Über den Autor	15
	Danksagungen	15
I	Pythons Sicht der Dinge	17
I.1	Punkt 1: Kenntnis der Python-Version	17
I.2	Punkt 2: Stilregeln gemäß PEP 8	19
I.3	Punkt 3: Unterschiede zwischen bytes, str und unicode	21
I.4	Punkt 4: Hilfsfunktionen statt komplizierter Ausdrücke	24
I.5	Punkt 5: Zugriff auf Listen	27
I.6	Punkt 6: Verzicht auf Abstand und Start-/End-Index	30
I.7	Punkt 7: Listen-Abstraktionen statt map und filter	32
I.8	Punkt 8: Nicht mehr als zwei Ausdrücke in Listen-Abstraktionen	34
I.9	Punkt 9: Generator-Ausdrücke in Listen-Abstraktionen	36
I.10	Punkt 10: enumerate statt range	38
I.11	Punkt 11: Gleichzeitige Verarbeitung von Iteratoren mit zip	39
I.12	Punkt 12: Verzicht auf else-Blöcke nach for- und while-Schleifen	42
I.13	Punkt 13: Alle Blöcke einer try/except/else/finally-Anweisung nutzen	45
2	Funktionen	49
2.1	Punkt 14: Exceptions statt Rückgabe von None	49
2.2	Punkt 15: Closures und der Gültigkeitsbereich von Variablen	51
2.3	Punkt 16: Generatoren statt Rückgabe von Listen	57
2.4	Punkt 17: Vorsicht beim Iterieren über Argumente	59
2.5	Punkt 18: Klare Struktur dank variabler Argumente	64
2.6	Punkt 19: Optionale Funktionalität durch Schlüsselwort- Argumente	67

2.7	Punkt 20: Dynamische Standardwerte von Argumenten mittels None und Docstrings	70
2.8	Punkt 21: Eindeutigkeit durch Schlüsselwort-Argumente	73
3	Klassen und Vererbung	79
3.1	Punkt 22: Hilfsklassen statt Dictionaries und Tupel	79
3.2	Punkt 23: Funktionen statt Klassen bei einfachen Schnittstellen ...	85
3.3	Punkt 24: Polymorphismus und generische Erzeugung von Objekten	89
3.4	Punkt 25: Initialisierung von Basisklassen durch super.	95
3.5	Punkt 26: Mehrfache Vererbung nur für Mix-in-Hilfsklassen	100
3.6	Punkt 27: Öffentliche statt private Attribute	105
3.7	Punkt 28: Benutzerdefinierte Container-Klassen durch Erben von collections.abc	110
4	Metaklassen und Attribute	115
4.1	Punkt 29: Einfache Attribute statt Getter- und Setter-Methoden ...	115
4.2	Punkt 30: @property statt Refactoring von Attributen	120
4.3	Punkt 31: Deskriptoren für wiederverwendbare @property-Methoden verwenden.	124
4.4	Punkt 32: Verzögerte Zuweisung zu Attributen mittels __getattr__, __getattribute__ und __setattr__	130
4.5	Punkt 33: Unterklassen mit Metaklassen überprüfen.	136
4.6	Punkt 34: Klassen mittels Metaklassen registrieren	138
4.7	Punkt 35: Zugriff auf Klassenattribute mit Metaklassen	143
5	Nebenläufigkeit und parallele Ausführung	147
5.1	Punkt 36: Verwaltung von Kindprozessen mittels subprocess	148
5.2	Punkt 37: Threads, blockierende Ein-/Ausgabevorgänge und parallele Ausführung.	152
5.3	Punkt 38: Wettlaufsituationen in Threads mit Lock verhindern ...	157
5.4	Punkt 39: Threads koordinieren mit Queue	160
5.5	Punkt 40: Parallele Ausführung mehrerer Funktionen mit Coroutinen	169
5.6	Punkt 41: Echte parallele Ausführung mit concurrent.futures	179
6	Standardmodule	185
6.1	Punkt 42: Funktions-Decorators mit functools.wraps definieren ...	185
6.2	Punkt 43: contextlib und with-Anweisung statt try/finally-Konstrukt	188
6.3	Punkt 44: Verlässliches pickle durch copyreg	192

6.4	Punkt 45: Für örtliche Zeitangaben <code>datetime</code> statt <code>time</code> verwenden	198
6.5	Punkt 46: Verwendung von Algorithmen und Datenstrukturen der Standardbibliothek	203
6.6	Punkt 47: Falls Genauigkeit an erster Stelle steht, <code>decimal</code> verwenden	208
6.7	Punkt 48: Module der Python-Community	210
7	Zusammenarbeit	213
7.1	Punkt 49: Docstrings für sämtliche Funktionen, Klassen und Module	213
7.2	Punkt 50: Pakete zur Organisation von Modulen und zur Bereitstellung stabiler APIs verwenden	218
7.3	Punkt 51: Einrichten einer Root-Exception zur Abschottung von Aufrufern und APIs	223
7.4	Punkt 52: Zirkuläre Abhängigkeiten auflösen	227
7.5	Punkt 53: Virtuelle Umgebungen zum Testen von Abhängigkeiten	233
8	Veröffentlichung	241
8.1	Punkt 54: Modulbezogener (Module-scoped) Code zur Konfiguration der Deployment-Umgebung	241
8.2	Punkt 55: Debuggen mit repr-Strings	244
8.3	Punkt 56: Überprüfung mit <code>unittest</code>	247
8.4	Punkt 57: Interaktives Debuggen mit <code>pdb</code>	251
8.5	Punkt 58: Vor der Optimierung Messungen vornehmen	253
8.6	Punkt 59: Nutzung des Arbeitsspeichers und Speicherlecks mit <code>tracemalloc</code> untersuchen	258
	Stichwortverzeichnis	263