

Spis treści

WPROWADZENIE	13
---------------------------	-----------

Część I. Pierwsze kroki	19
--------------------------------	-----------

1	
OBŚLUGA BŁĘDÓW I POSZUKIWANIE POMOCY	21

Komunikaty o błędach w Pythonie	22
Analiza śladu	22
Wyszukiwanie informacji na temat komunikatów o błędach	25
Zapobieganie błędom dzięki wykorzystaniu linterów	26
Jak prosić o pomoc w programowaniu?	28
Ogranicz konieczność dopowiadania — od razu podaj jak najwięcej informacji	29
Sformułuj swój problem w postaci rzeczywistego pytania	29
Zadawaj pytania na właściwej stronie internetowej	29
Podsumuj swoje pytanie w nagłówku	30
Wyjaśnij, co ma robić Twój kod	30
Dołącz kompletny komunikat o błędzie	30
Udostępnij swój pełny kod	30
Zastosuj odpowiednie formatowanie, aby poprawić czytelność kodu	32
Powiedz osobom pomagającym, czego już próbowałeś	33
Opisz swoją konfigurację	33
Przykłady pytań	34
Podsumowanie	34

2	
KONFIGURACJA ŚRODOWISKA I WIERSZ POLECENIA	36

System plików	37
Ścieżki w Pythonie	38
Katalog domowy	38
Bieżący katalog roboczy	39
Ścieżki względne i bezwzględne	39
Programy i procesy	40
Wiersz poleceń	41
Otwieranie okna terminala	42
Uruchamianie programów z wiersza poleceń	43
Korzystanie z argumentów wiersza poleceń	44

Uruchamianie kodu Pythona z wiersza poleceń z wykorzystaniem argumentu -c	46
Uruchamianie programów w Pythonie z wiersza poleceń	46
Uruchamianie programu py.exe	46
Uruchamianie poleceń systemu operacyjnego z programu Pythona	47
Minimalizacja wpisywania dzięki mechanizmowi uzupełniania z wykorzystaniem klawisza Tab	48
Wyświetlanie historii poleceń	48
Korzystanie z popularnych poleceń	49
Zmienne środowiskowe i PATH	57
Wyświetlanie wartości zmiennych środowiskowych	57
Korzystanie ze zmiennej środowiskowej PATH	58
Zmiana zawartości zmiennej środowiskowej PATH dla wiersza polecenia	59
Trwałe dodawanie folderów do zmiennej PATH w systemie Windows	60
Trwałe dodawanie folderów do zmiennej PATH w systemach macOS i Linux	61
Uruchamianie programów Pythona bez wiersza poleceń	61
Uruchamianie programów Pythona w systemie Windows	62
Uruchamianie programów Pythona w systemie macOS	63
Uruchamianie programów Pythona w systemie Ubuntu Linux	63
Podsumowanie	64

Część II. Najlepsze praktyki, narzędzia i techniki 65

3 FORMATOWANIE KODU ZA POMOCĄ NARZĘDZIA BLACK 67

Jak stracić przyjaciół i zrobić sobie wrogów wśród współpracowników?	68
Przewodniki stylu i PEP 8	68
Odstępy w poziomie	69
Używaj jako wcięć znaków spacji	69
Odstępy w obrębie wiersza	71
Odstępy w pionie	74
Przykład zastosowania odstępów w pionie	75
Najlepsze praktyki dotyczące odstępów w pionie	76
Black: bezkompromisowy formater kodu	77
Instalacja narzędzia Black	77
Uruchamianie narzędzia Black z wiersza polecenia	78
Wyłączenie programu Black dla części Twojego kodu	81
Podsumowanie	82

4 WYBIERANIE ZROZUMIAŁYCH NAZW 83

Style wielkości liter	84
Konwencje nazewnictwa PEP 8	85
Odpowiednia długość nazw	85
Zbyt krótkie nazwy	86
Zbyt długie nazwy	87
Korzystaj z nazw ułatwiających wyszukiwanie	89
Unikaj dowcipów, kalamburów i określeń żargonowych	89
Nie nadpisuj wbudowanych nazw	90
Najgorsze możliwe nazwy zmiennych	91
Podsumowanie	92

WYSZUKIWANIE CUCHNĄCEGO KODU	93
Powielony kod	94
Magiczne liczby	95
Kod wykomentowany i martwy	98
Debugowanie za pomocą komunikatów	100
Zmienne z przyrostkami numerycznymi	101
Klasy, które powinny być funkcjami lub modułami	101
Listy składane wewnątrz list składanych	102
Puste bloki except i niejasne komunikaty o błędach	104
Mity związane z cuchnącym kodem	106
Mit: funkcje powinny mieć tylko jedną instrukcję return na końcu	106
Mit: funkcje powinny zawierać co najwyżej jedną instrukcję try	106
Mit: argumenty-flagi są złe	107
Mit: zmienne globalne są złe	108
Mit: komentarze są niepotrzebne	109
Podsumowanie	110
 6	
PISANIE „PYTHONICZNEGO” KODU	111
Zen Pythona	112
Naucz się cenić znaczące wcięcia	115
Częste przypadki niewłaściwego korzystania ze składni	116
Używaj funkcji enumerate() zamiast range()	117
Używaj instrukcji with zamiast open() i close()	118
Do porównywania z None używaj is zamiast ==	119
Formatowanie ciągów znaków	119
Jeśli ciąg zawiera wiele lewych ukośników, używaj surowych ciągów znaków	120
Formatowanie ciągów za pomocą f-stringów	120
Tworzenie płytkich kopii list	122
Pythoniczne sposoby korzystania ze słowników	123
Używaj ze słownikami wywołań get() i.setdefault()	123
Użyj dla wartości domyślnych klasy collections.defaultdict	124
Używaj słowników zamiast instrukcji switch	125
Wyrażenia warunkowe: „brzydki” operator trójargumentowy Pythona	126
Korzystanie z wartości zmiennych	128
Operatory przypisania i operatory porównania	128
Sprawdzanie, czy zmienna jest jedną z wielu wartości	129
Podsumowanie	130
 7	
PROGRAMISTYCZNY ŻARGON	131
Definicje	132
Python język i Python interpreter	132
Odśmiecianie	133
Literały	133
Słowa kluczowe	134
Obiekty, wartości, egzemplarze i tożsamości	135
Elementy	138
Obiekty mutowalne i niemutowalne	138
Indeksy, klucze i skróty	141

Kontenery, sekwencje, mapowanie i typy zbiorów	144
Metody dunder i metody magiczne	145
Moduły i pakiety	145
Obiekty wywoływalne i obiekty pierwszej klasy	146
Często mylone terminy	147
Instrukcje a wyrażenia	147
Blok, klauzula i ciało	148
Zmienna a atrybut	149
Funkcja a metoda	150
Obiekt iterowalny a iterator	150
Błędy składniowe, błędy wykonania a błędy semantyczne	152
Parametry a argumenty	153
Koercja typu a rzutowanie typu	154
Właściwości a atrybuty	154
Kod bajtowy a kod maszynowy	155
Skrypt a program, język skryptowy a język programowania	155
Biblioteka, framework, SDK, silnik i API	156
Podsumowanie	157
Dalsza lektura	158
8	
ZNANE PUŁAPKI PYTHONA	159
Nie dodawaj ani nie usuwaj elementów z listy, kiedy po niej iterujesz	160
Nie kopiuj mutowalnych wartości inaczej niż poprzez wywołania <code>copy.copy()</code> lub <code>copy.deepcopy()</code>	166
Nie używaj wartości mutowalnych w roli argumentów domyślnych	169
Nie buduj ciągów za pomocą konkatenacji	171
Nie oczekuj, że funkcja <code>sort()</code> posortuje listę alfabetycznie	173
Nie zakładaj, że liczby zmiennoprzecinkowe są idealnie dokładne	174
Nie twórz łańcucha operatorów nierówności <code>!=</code>	177
Nie zapominaj o przecinku w krotce złożonej z jednego elementu	178
Podsumowanie	178
9	
EZOTERYCZNE OSOBLIWOŚCI PYTHONA	180
Dlaczego 256 to jest 256, ale 257 to nie jest 257	180
Internowanie ciągów	182
Sztuczne operatory inkrementacji i dekrementacji w Pythonie	183
Wszystko z nic	185
Wartości logiczne są liczbami całkowitymi	186
Tworzenie łańcucha operatorów różnego rodzaju	187
Antygravitacja w Pythonie	188
Podsumowanie	189
10	
PISANIE SKUTECZNYCH FUNKCJI	190
Nazwy funkcji	190
Kompromisy dotyczące rozmiaru funkcji	191
Parametry i argumenty funkcji	194
Argumenty domyślne	194
Korzystanie z <code>*</code> i <code>**</code> w celu przekazywania argumentów do funkcji	195

Użycie * do tworzenia funkcji wariadycznych	197
Tworzenie funkcji wariadycznych za pomocą składni **	199
Tworzenie funkcji-wrapperów za pomocą składni * i **	201
Programowanie funkcyjne	202
Skutki uboczne	202
Funkcje wyższego rzędu	204
Funkcje lambda	205
Mapowanie i filtrowanie z wykorzystaniem list składanych	206
Zwracane wartości zawsze powinny mieć ten sam typ danych	207
Zgłaszanie wyjątków a zwracanie kodów błędów	209
Podsumowanie	210
11	
KOMENTARZE, DOCSTRINGI I WSKAZÓWKI TYPU	211
Komentarze	212
Styl komentarzy	213
Komentarze inline	214
Komentarze wyjaśniające	214
Komentarze podsumowujące	215
Komentarze typu „wyciągnięte wnioski”	216
Komentarze prawne	216
Profesjonalny ton	217
Znaczniki kodu i komentarze TODO	217
Magiczne komentarze i kodowanie plików źródłowych	218
Docstringi	218
Wskazówki typu	221
Korzystanie z narzędzi do statycznej analizy kodu	223
Ustawianie wskazówek dla wielu typów	225
Ustawianie wskazówek typu dla list, słowników i nie tylko	226
Backport wskazówek typu z wykorzystaniem komentarzy	227
Podsumowanie	229
12	
ORGANIZOWANIE PROJEKTÓW KODU Z WYKORZYSTANIEM SYSTEMU GIT	230
Commity i repozytoria systemu Git	231
Korzystanie z narzędzia Cookiecutter do tworzenia nowych projektów w Pythonie	231
Instalacja Gita	234
Konfigurowanie nazwy użytkownika Gita i adresu e-mail	234
Instalacja narzędzi GUI dla Gita	235
Przepływ pracy w systemie Git	236
W jaki sposób Git śledzi stan pliku?	236
Po co jest stan staged?	238
Tworzenie repozytorium Gita na komputerze lokalnym	238
Dodawanie plików do śledzenia przez Gita	240
Ignorowanie plików w repozytorium	241
Zatwierdzanie zmian	242
Usuwanie plików z repozytorium	247
Zmiana nazwy i przenoszenie plików w repozytorium	248
Przeglądanie loga commitów	249
Przywracanie wcześniejszych zmian	250
Cofanie niezatwierdzonych lokalnych zmian	251
Anulowanie stanu staged pliku	251

Cofanie najnowszych commitów	252
Cofanie zmian do określonego commita dla pojedynczego pliku	252
Przepisywanie historii commitów	254
GitHub i polecenie git push	254
Przesyłanie istniejącego repozytorium do usługi GitHub	255
Klonowanie repozytorium z istniejącego repozytorium w serwisie GitHub	256
Podsumowanie	257
13	
MIERZENIE WYDAJNOŚCI ALGORYTMÓW I ANALIZA BIG O	258
Moduł timeit	259
Profiler cProfile	261
Analiza algorytmów Big O	263
Rzędy w notacji Big O	264
Metafora regału dla rzędów notacji Big O	265
Notacja Big O mierzy najgorszy scenariusz	269
Określanie rzędu Big O kodu	271
Dlaczego niższe rzędy i współczynniki nie mają znaczenia?	272
Przykłady analizy Big O	273
Rzędy Big O popularnych wywołań	276
Analiza Big O w mgnieniu oka	277
Big O nie ma znaczenia dla małych n, czyli dla większości przypadków	279
Podsumowanie	279
14	
PRAKTYCZNE PROJEKTY	281
Wieża Hanoi	282
Wyjście	283
Kod źródłowy	284
Pisanie kodu	286
Cztery w rzędzie	294
Wyjście	295
Kod źródłowy	296
Pisanie kodu	299
Podsumowanie	308
Część III. Python obiekowy	309
15	
KLASY I PROGRAMOWANIE OBIEKTOWE	311
Analogia do rzeczywistego świata: wypełnianie formularza	312
Tworzenie obiektów na podstawie klas	314
Tworzenie prostej klasy: WizCoin	315
Metody <code>__init__()</code> i <code>self</code>	317
Atrybuty	318
Atrybuty prywatne i metody prywatne	319
Funkcja <code>type()</code> i atrybut <code>__qualname__</code>	321
Przykłady kodu obiektowego i nieobiekowego: kółko i krzyżyk	322
Projektowanie klas dla rzeczywistych aplikacji jest trudne	327
Podsumowanie	328

16

PROGRAMOWANIE OBIEKTOWE I DZIEDZICZENIE329

Jak działa dziedziczenie	330
Przestanianie metod	332
Funkcja super()	334
Staraj się stosować kompozycję zamiast dziedziczenia	336
Minusy dziedziczenia	337
Funkcje isinstance() i isinstance()	340
Metody klasy	341
Atrybuty klasy	343
Metody statyczne	344
Kiedy używać metod i atrybutów klasy oraz metod statycznych w programach obiektowych?	344
Terminologia obiektowa	345
Hermetyzacja	345
Polimorfizm	345
Kiedy nie używać dziedziczenia?	346
Dziedziczenie wielokrotne	346
Kolejność rozpoznawania metod	348
Podsumowanie	350

17

PYTHONICZNY PARADYGMAT OOP: WŁAŚCIWOŚCI I METODY DUNDER352

Właściwości	353
Przekształcanie atrybutu we właściwość	353
Używanie setterów do sprawdzania poprawności danych	356
Właściwości tylko do odczytu	358
Kiedy używać właściwości?	359
Metody dunder w Pythonie	360
Metody dunder reprezentacji tekstowych	360
Liczbowe metody dunder	363
Odbite numeryczne metody dunder	366
Metody dunder rozszerzonego przypisania w miejscu	369
Metody dunder porównań	370
Podsumowanie	376