helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

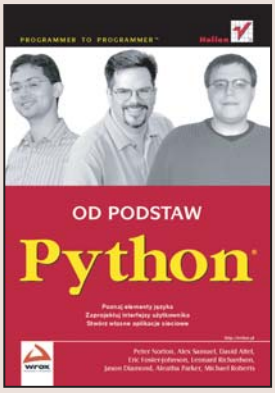
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Python. Od podstaw



Autor: Zespół autorów
Tłumaczenie: Rafał Jońca
ISBN: 83-246-0528-2
Tytuł oryginału: Beginning Python
Format: B5, stron: 704

Dołącz do społeczności programistów Pythona!

- Poznaj elementy języka
- Zaprojektuj interfejsy użytkownika
- Stwórz własne aplikacje sieciowe

Python to jeden z najszybciej rozwijających się języków programowania. Jest dostępny na licencji open source i posiada elastyczną, czytelną składnię. Jego możliwości pozwalają programistom na tworzenie aplikacji sieciowych, komunikację z bazami danych i zarządzanie systemami operacyjnymi. Python jest językiem wieloplatformowym, dzięki czemu napisane w nim programy można uruchamiać w różnych środowiskach i pod kontrolą różnych systemów operacyjnych. Ogromne możliwości tego języka zainspirowały duże grono entuzjastów aktywnie dzielących się wiedzą na jego temat na różnego rodzaju forach i listach dyskusyjnych. Gwarantuje to, że żadne zadane im pytanie dotyczące Pythona nie pozostanie bez odpowiedzi.

Książka „Python. Od podstaw" to podręcznik dla tych, którzy chcą opanować ten język i tworzyć w nim własne aplikacje. Dzięki niej poznasz wszystkie elementy Pythona i dowiesz się, na czym polega programowanie obiektowe. Nauczysz się przetwarzać dane tekstowe i liczbowe, tworzyć graficzne interfejsy użytkownika za pomocą GTK oraz łączyć aplikacje z bazami danych. Poznasz zasady korzystania z plików XML, pisania aplikacji internetowych i integrowania Pythona z usługami sieciowymi oraz innymi językami programowania.

Oto niektóre z zagadnień poruszanych w tej książce:

- operacje na liczbach i ciągach znaków,
- konstrukcje sterujące,
- funkcje i moduły,
- programowanie obiektowe,
- operacje na plikach i folderach,
- połączenia z bazami danych,
- przetwarzanie plików XML,
- obsługa serwerów pocztowych,
- tworzenie własnych rozszerzeń w języku C,
- aplikacje biznesowe,
- usługi sieciowe,
- integracja Pythona i Javy.

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

O autorach 15

Wprowadzenie 17

Rozdział 1. Podstawy programowania i ciągi znaków 25

Czym różni się programowanie od używania komputera? 25
Programowanie to spójność 26
Programowanie to sterowanie 26
Programowanie podąża za zmianami 27
Co to wszystko oznacza? 27
Pierwsze kroki 27
Uruchamianie edytora codeEditor 28
Wykorzystywanie powłoki Pythona w edytorze codeEditor 28
Zaczynamy korzystać z Pythona — ciągi znaków 29
Czym jest ciąg znaków? 30
Dlaczego cudzysłowy? 30
Stosowanie apostrofów i cudzysłowów 31
Łączenie dwóch ciągów znaków 32
Złączanie ciągów znaków na różne sposoby 33
Wyświetlanie tekstów za pomocą instrukcji print 34
Podsumowanie 35
Ćwiczenia 36

Rozdział 2. Liczby i operatory 37

Różne rodzaje liczb 37
Liczby w Pythonie 38
Pliki programów 40
Korzystanie z różnych typów 41
Podstawowe działania matematyczne 43
Kilka niespodzianek 45
Wykorzystywanie obliczeń matematycznych 45
Kolejność wykonywania działań 45
Formaty liczb 46

Błędy się zdarzają ... 47
Pewne nietypowe rozwiązania ... 48
Podsumowanie ... 49
Ćwiczenia ... 50

Rozdział 3. Zmienne — nazwy dla wartości ... 51

Przechowywanie danych — wykorzystywanie nazw ... 51
Zmiana danych za pomocą nazwy zmiennej ... 52
Kopiowanie danych ... 53
Nazwy, których nie można używać, i kilka zasad ... 53
Kolejne wbudowane typy danych ... 54
Krotki — niezmienne sekwencje danych ... 54
Listy — modyfikowalne sekwencje danych ... 57
Słowniki — grupowanie danych z indeksacją na podstawie nazw ... 59
Traktowanie ciągu znaków jak listy ... 61
Typy specjalne ... 62
Inne typowe właściwości sekwencji ... 63
Dostęp do ostatniego elementu ... 63
Zakresy sekwencji ... 63
Rozszerzanie list przez dodawanie kolejnych elementów ... 64
Wykorzystywanie list do tymczasowego przechowywania danych ... 65
Podsumowanie ... 66
Ćwiczenia ... 66

Rozdział 4. Podejmowanie decyzji ... 69

Porównywanie wartości — czy są takie same? ... 69
Operacja przeciwna — nierówność ... 71
Porównywanie wartości — która jest większa? ... 71
Większy lub równy, mniejszy lub równy ... 73
Negacja prawdy lub fałszu ... 73
Poszukiwanie wyniku więcej niż jednego porównania ... 74
Podejmowanie decyzji ... 75
Powtarzanie ... 77
Jak wykonywać coś raz za razem? ... 77
Zatrzymywanie pętli ... 79
Obsługa błędów ... 81
Wypróbowywanie kodu ... 82
Podsumowanie ... 84
Ćwiczenia ... 85

Rozdział 5. Funkcje ... 87

Umieszczanie programu w osobnym pliku ... 87
Funkcje — grupowanie kodu pod konkretną nazwą ... 89
Dobór nazwy ... 90
Opisywanie funkcji w jej wnętrzu ... 91
Ta sama nazwa w dwóch różnych miejscach ... 92
Pozostawianie notatek samemu sobie ... 93
Przekazywanie wartości do funkcji ... 94
Sprawdzanie parametrów ... 96
Ustawianie wartości domyślnej parametru ... 98

Wywoływanie funkcji wewnątrz innych funkcji ... 99
Funkcje wewnątrz funkcji ... 101
Zgłaszanie własnych błędów ... 102
Warstwy funkcji ... 103
Sposób analizy błędów w zagnieżdżonych funkcjach ... 103
Podsumowanie ... 104
Ćwiczenia ... 105

Rozdział 6. Klasy i obiekty ... 107

Podejścia do programowania ... 107
Pojęcie obiektu jest powszechnie znane ... 107
W jaki sposób korzystać z obiektów? ... 109
Definiowanie klasy ... 109
W jaki sposób wykonać obiekt? ... 110
Obiekty i ich zasięg ... 117
Podsumowanie ... 120
Ćwiczenia ... 122

Rozdział 7. Organizacja programów ... 123

Moduły ... 124
Importowanie modułów, z których chce się skorzystać ... 124
Tworzenie modułu na podstawie istniejącego kodu ... 125
Korzystanie z modułów — zaczynamy od wiersza poleceń ... 127
Zmiana sposobu działania importu ... 129
Pakiety ... 129
Moduły i pakiety ... 131
Przeniesienie wszystkiego do aktualnego zasięgu ... 131
Ponowny import modułów i pakietów ... 132
Podstawy testowania modułów i pakietów ... 134
Podsumowanie ... 135
Ćwiczenia ... 136

Rozdział 8. Pliki i foldery ... 137

Obiekty file ... 137
Zapis plików tekstowych ... 138
Odczyt plików tekstowych ... 139
Wyjątki dotyczące plików ... 141
Ścieżki i foldery ... 142
Ścieżki ... 142
Zawartość folderu ... 145
Uzyskiwanie informacji o plikach ... 146
Rekurencyjne wyświetlanie folderów ... 146
Zmiana nazwy, przenoszenie, kopiowanie i usuwanie plików ... 148
Przykład — rotacja plików ... 148
Tworzenie i usuwanie folderów ... 150
Globbing ... 150
Serializacja ... 152
Wskazówki dotyczące serializacji ... 153
Wydajna serializacja ... 154
Podsumowanie ... 154
Ćwiczenia ... 155

Rozdział 9. Inne elementy języka Python 157
Lambda i filtry — krótkie funkcje anonimowe 157
Funkcja reduce 158
Funkcja map — krótsza wersja pętli 159
Decyzje wewnątrz list — listy składane 160
Generowanie list dla pętli 161
Zastępowanie ciągów znaków wartościami ze słowników 163
Przydatne modułu 165
Getopt — pobieranie opcji z wiersza poleceń 165
Wykorzystywanie więcej niż jednego procesu 167
Wątki — wiele zadań wykonywanych przez jeden proces 169
Przechowywanie haseł 171
Podsumowanie 172
Ćwiczenia 173

Rozdział 10. Tworzenie modułu 175
Szczegóły działania modułów 175
Importowanie modułów 177
Znajdowanie modułu 177
Analiza istniejącego modułu 178
Tworzenie modułów i pakietów 181
Stosowanie klas 182
Elementy programowania obiektowego 183
Tworzenie klas 183
Rozszerzanie istniejących klas 185
Wykonywanie pozostałych zadań związanych z modułami 186
Definiowanie błędów specyficznych dla modułu 186
Określanie eksportowanych informacji 187
Dokumentowanie modułu 188
Testowanie modułu 194
Uruchamianie modułu jako programu 195
Tworzenie pełnego modułu 196
Jak to działa? 199
Instalacja własnych modułów 202
Podsumowanie 205
Ćwiczenia 206

Rozdział 11. Przetwarzanie tekstu 207
Dlaczego przetwarzanie tekstów jest tak istotne? 207
Wyszukiwanie plików 208
Analiza dzienników 209
Przeszukiwanie poczty 210
Poruszanie się po systemie plików za pomocą modułu os 210
Wyrażenia regularne i moduł re 216
Podsumowanie 219
Ćwiczenia 220

Rozdział 12. Testy 221
Asercje 222
Przypadki testowe i zestawy testowe 223
Osprzęt testowy 227

Łączymy wszystko, wykorzystując metodologię XP ... 230
Implementacja w Pythonie narzędzia wyszukiwania ... 231
Bardziej zaawansowany skrypt wyszukujący ... 236
Testy formalne w cyklu życia oprogramowania ... 238
Podsumowanie ... 239

Rozdział 13. Tworzenie graficznych interfejsów użytkownika ... 241

Środowiska do tworzenia graficznych interfejsów dostępne w Pythonie ... 241
Wprowadzenie do pyGTK ... 243
Zasoby dotyczące pyGTK ... 243
Tworzenie interfejsów graficznych za pomocą pyGTK ... 245
Sygnały GUI ... 247
Wątki pomocnicze GUI i kolejka zdarzeń GUI ... 248
Pakowanie widgetów ... 254
Glade — tworzenie interfejsów graficznych dla pyGTK ... 255
Systemy budowania GUI dla innych szkieletów interfejsów graficznych ... 256
Wykorzystywanie libglade w Pythonie ... 256
Krótki przewodnik po Glade ... 257
Uruchamianie Glade ... 257
Tworzenie projektu ... 259
Wykorzystywanie palety do utworzenia okna ... 259
Umieszczanie widgetów w oknie ... 260
Glade tworzy plik XML opisujący interfejs graficzny ... 261
Tworzenie rozbudowanej aplikacji z wykorzystaniem Glade ... 263
Zaawansowane widgety ... 269
Dalsza rozbudowa PyRAP ... 272
Podsumowanie ... 278
Ćwiczenia ... 279

Rozdział 14. Dostęp do baz danych ... 281

Korzystanie z trwałych słowników DBM ... 282
Wybór modułu DBM ... 282
Tworzenie trwałego słownika ... 283
Dostęp do danych trwałego słownika ... 285
Kiedy stosować trwały słownik, a kiedy relacyjną bazę danych? ... 287
Korzystanie z relacyjnych baz danych ... 288
Instrukcje SQL ... 289
Definicje tabel ... 291
Tworzenie bazy danych ... 292
Stosowanie interfejsu programistycznego baz danych ... 294
Pobieranie modułów ... 295
Tworzenie połączeń ... 296
Korzystanie z kursorów ... 296
Transakcje — zatwierdzanie i wycofywanie zmian ... 304
Sprawdzanie możliwości modułu oraz metadane ... 304
Obsługa błędów ... 305
Podsumowanie ... 306
Ćwiczenia ... 306

Rozdział 15. Python i XML 309
Czym jest XML? 309
Hierarchiczny język znaczników 309
Rodzina standardów 311
Czym jest Schema i DTD? 312
Do czego używa się modelu dokumentów? 312
Czy model dokumentu jest potrzebny? 312
Dokument DTD 312
Przykład DTD 313
DTD to nie XML 314
Ograniczenia DTD 314
Dokument Schema 314
Przykład dokumentu Schema 315
Schema to standardowy dokument XML 315
Schema jest hierarchiczny 315
Inne zalety Schema 316
Schema jest znacznie rzadziej obsługiwany 316
XPath 316
HTML jako podzbiór języka XML 317
Modele DTD dla języka HTML 317
Moduł i klasa HTMLParser 317
Moduł htmllib 318
Biblioteki XML dostępne w Pythonie 319
Walidacja dokumentu XML w Pythonie 320
Czym jest walidacja? 320
Poprawność formatu a walidacja 320
Dostępne narzędzia 321
Czym jest SAX? 322
Bazujący na strumieniu 323
Sterowany zdarzeniami 323
Czym jest DOM? 323
Dostęp bezpośrednio z pamięci operacyjnej 323
SAX czy DOM? 324
Możliwości 324
Wymagania pamięciowe 324
Szybkość działania 324
Analizatory SAX i DOM dostępne w Pythonie 325
Pakiet PyXML 325
Pakiet xml.sax 325
Pakiet xml.dom.minidom 325
Wprowadzenie do XSTL 328
XSTL to XML 328
Język formatowania i przekształceń 329
Funkcyjny, sterowany szablonami 329
Wykorzystywanie Pythona do przekształceń XML za pomocą XSTL 329
Łączymy wszystko — kanały RSS 331
Wstęp do kanałów RSS 332
Model DTD dla RSS 333
Problem z rzeczywistego świata 333
Kolejny problem z rzeczywistego świata 336
Podsumowanie 338
Ćwiczenia 338

Rozdział 16. Programowanie sieciowe 341

Protokoły 343
Porównanie protokołów i języków programowania 343
Stos protokołów internetowych 344
Dodatkowe informacje na temat protokołu IP 346
Wysyłanie wiadomości e-mail 347
Format wiadomości e-mail 348
Wiadomości MIME 349
Wysyłanie poczty za pomocą SMTP i smtplib 358
Odczytywanie wiadomości e-mail 360
Przetwarzanie lokalnej skrzynki pocztowej modułem mailbox 360
Pobieranie poczty z serwera POP3 za pomocą modułu poplib 361
Pobieranie poczty z serwera IMAP modułem imaplib 364
Bezpieczne połączenia POP3 i IMAP 368
Aplikacje obsługi poczty przez WWW nie są typowymi klientami poczty 369
Programowanie z wykorzystaniem gniazd 369
Gniazda 369
Dołączanie do zewnętrznych interfejsów 372
Serwer odwracający tekst 372
Klient dla serwera odwracającego tekst 374
Moduł SocketServer 375
Serwery wielowątkowe 377
Serwer pogawędek internetowych 378
Projekt serwera pogawędek 378
Protokół serwera pokoju pogawędek 379
Klient pogawędek 384
Jednowątkowa wielozadaniowość uzyskiwana za pomocą select 386
Inne tematy 388
Rozważania na temat projektowania protokołów 388
Szkielet Twisted 389
Architektura komunikacji równorzędnej 392
Podsumowanie 392
Ćwiczenia 393

Rozdział 17. Tworzenie rozszerzeń w języku C 395

Ogólna postać modułu rozszerzeń 396
Kompilacja i instalacja modułu rozszerzenia 398
Przekazywanie parametrów z Pythona do C 400
Zwracanie wartości z C do Pythona 403
Projekt LAME 404
Moduł rozszerzenia LAME 408
Używanie obiektów Pythona z poziomu kodu języka C 420
Podsumowanie 423
Ćwiczenia 423

Rozdział 18. Pisanie programów komercyjnych i shareware 425

Analiza przypadku — tło 426
W jakim stopniu wykorzystywać Pythona? 426
Licencjonowanie kodu napisanego w Pythonie 428
Warto skorzystać z usług internetowych 429

Strategia cen ... 430
Znaki wodne ... 430
Inne modele ... 435
Sprzedaż platformy, a nie produktu ... 436
Twoje środowisko programistyczne ... 436
Poszukiwanie programistów języka Python ... 438
Szkolenie programistów znających inne języki niż Python ... 438
Doświadczeni programiści Pythona ... 438
Problemy z Pythonem ... 438
Tworzenie kodu dla innych wersji Pythona ... 439
Przejście na inny system operacyjny ... 440
Debugowanie wątków ... 441
Typowe pomyłki ... 441
Przenośna dystrybucja ... 442
Istotne biblioteki ... 443
Timeoutsocket ... 443
PyGTK ... 444
GEOip ... 444
Podsumowanie ... 444

Rozdział 19. Programowanie numeryczne ... 445

Liczby w Pythonie ... 446
Liczby całkowite ... 446
Długie liczby całkowite ... 447
Wartości zmiennoprzecinkowe ... 447
Formatowanie liczb ... 448
Znaki jako liczby ... 451
Matematyka ... 452
Arytmetyka ... 452
Wbudowane funkcje matematyczne ... 454
Moduł math ... 455
Liczby zespolone ... 456
Tablice ... 459
Moduł array ... 460
Pakiet numarray ... 462
Podsumowanie ... 465
Ćwiczenia ... 465

Rozdział 20. Python w firmie ... 467

Aplikacje biznesowe ... 468
Zarządzanie dokumentem ... 468
Ludzie skatalogowani ... 471
Podejmowanie akcji za pomocą diagramów przepływu ... 472
Audyty, ustawy i inna cenna wiedza ... 473
Audyty i zarządzanie dokumentami ... 474
Korzystanie z rzeczywistych systemów biznesowych ... 475
Wprowadzenie do zestawu narzędzi wftk ... 476
Moduł python-ldap ... 490
Dodatkowe informacje na temat LDAP ... 494
Wracamy do wftk ... 495
Podsumowanie ... 500
Ćwiczenia ... 500

Rozdział 21. Aplikacje i usługi internetowe 501

REST — architektura sieci WWW 502
Charakterystyka REST 502
Operacje REST 504
HTTP — REST w realnym świecie 505
Widoczny serwer WWW 506
Żądanie HTTP 509
Odpowiedź HTTP 510
CGI — zamiana skryptu w aplikację internetową 511
Umowa między serwerem WWW a skryptem CGI 513
Specjalne zmienne środowiskowe CGI 514
Pobieranie danych od użytkownika przy użyciu formularzy HTML 516
Moduł cgi — przetwarzanie formularzy HTML 517
Tworzenie wiki 522
Główna biblioteka BittyWiki 524
Interfejs WWW dla BittyWiki 527
Usługi internetowe 536
Sposób działania usług internetowych 537
Usługi internetowe REST 538
Szybkie wprowadzenie do REST — znajdowanie okazji w Amazon.com 538
Znajdowanie okazji 540
Dodawanie interfejsu REST do BittyWiki 544
Wyszukiwanie i zastępowanie w wiki przy użyciu usługi internetowej w systemie REST 547
Protokół XML-RPC 551
Krótkie wprowadzenie do XML-RPC — pobieranie wiadomości z Meerkat 553
Żądanie XML-RPC 555
Odpowiedź XML-RPC 556
Gdy coś nie działa 557
Udostępnianie BittyWiki dzięki XML-RPC 558
Wyszukiwanie i zastępowanie korzystające z usługi XML-RPC 561
SOAP 563
Krótkie wprowadzenie do SOAP — interfejs Google API 563
Żądanie SOAP 566
Odpowiedź SOAP 567
Jeśli zdarzy się błąd 568
Udostępnianie BittyWiki za pomocą interfejsu SOAP 569
Wyszukiwanie i zastępowanie korzystające z usługi SOAP 571
Dokumentowanie interfejsu usługi internetowej 572
Dokumentacja czytelna dla ludzi 573
Interfejs introspekcji XML-RPC 574
WSDL 575
Wybór standardu usług internetowych 578
Etykieta usług internetowych 579
Użytkownicy usług 579
Twórcy usług 580
Wykorzystywanie aplikacji internetowych jako usługi internetowe 580
Przykłady publicznie dostępnych usług internetowych 581
Podsumowanie 582
Ćwiczenia 582

Rozdział 22. Integracja Pythona i Javy 583
Powody tworzenia skryptów w aplikacjach Javy 584
Porównanie implementacji Pythona 585
Instalacja Jythona 586
Uruchamianie Jythona 586
Uruchamianie trybu interaktywnego 586
Wykonywanie skryptów w Jythonie 588
Opcje sterowania skryptami 589
Tworzenie poleceń wykonywalnych 589
Uruchamianie Jythona bez użycia skryptu 591
Pakiet aplikacji bazującej na Jythonie 591
Integracja Javy i Jythona 592
Stosowanie klas Javy w Jythonie 592
Dostęp do baz danych z poziomu Jythona 597
Tworzenie serwletów J2EE w Jythonie 602
Rozszerzanie HttpServlet 605
Wybór narzędzi dla Jythona 608
Testowanie z wykorzystaniem Jythona 609
Osadzanie interpretera Jythona 610
Wywoływanie skryptu Jythona w Javie 610
Kompilacja skryptu Pythona do kodu Javy 612
Obsługa różnic między rozwiązaniami CPython i Jython 613
Podsumowanie 614
Ćwiczenia 615

Dodatek A Odpowiedzi do ćwiczeń 617

Dodatek B Zasoby internetowe 651

Dodatek C Co nowego w Pythonie 2.4? 655

Słowniczek 659

Skorowidz 669

1

Podstawy programowania i ciągi znaków

Niniejszy rozdział stanowi wprowadzenie do programowania w Pythonie. Python to bardzo bogaty język, z wieloma funkcjami, więc warto poświęcić chwilę na naukę chodzenia, zanim rozpocznie się bieganie. Rozdziały od 1. do 3. zawierają opis podstawowych zasad programowania przedstawiony w sposób jak najbardziej przyjazny, wraz z prostymi przykładami.

Jeśli jesteś doświadczonym programistą i chcesz jedynie poznać Pythona, możesz tylko przejrzeć ten rozdział, zwracając baczniejszą uwagę na prezentowane przykłady. Od rozdziału 3. zaczyna się jednak materiał, który warto przeczytać w całości, gdyż zawiera wiele cennych informacji o nowym języku.

Jeśli jesteś programistą początkującym, pod koniec rozdziału będziesz posiadał już pewną wiedzę na temat programowania, a także miał za sobą pierwsze interakcje z językiem programowania — Pythonem. Ćwiczenia znajdujące się na końcu rozdziału mają za zadanie sprawdzić i ugruntować zdobytą wiedzę.

Czym różni się programowanie od używania komputera?

Warto zrozumieć, że gdy się programuje, ma się kontrolę nad poczynaniami komputera. Czasem nie wykonuje on od razu tego, czego się oczekiwało, niemniej po kilku próbach i modyfikacjach najczęściej zaczyna poprawnie realizować powierzone mu zadania — oczywiście do momentu, gdy nie zmieni się programu.

Niestety, w komputerach osobistych pojawił się trend, który oddalił nas od niezawodności, ponieważ nowe oprogramowanie tworzone jest na podstawie innego oprogramowania, które nie zawsze bywa stabilne. Z tego powodu typowy użytkownik komputera odnosi czasem

wrażenie, że komputer jest złośliwą i cwaną bestią, która wymaga ogromnego nakładu pracy i środków, by coś poprawnie wykonać. Jeśli też tak czujesz, wiedz, że nie jesteś sam. Gdy ktoś nauczy się programować, zaczyna rozumieć powody takiego stanu rzeczy. Co więcej, często zdaje sobie sprawę, że pewne zadania mógłby wykonać lepiej niż programiści piszący oprogramowanie, z którego aktualnie korzysta.

Pamiętaj, że programowanie w języku takim jak Python, czyli w języku interpretowanym, oznacza, że nie trzeba zawracać sobie głowy całym sprzętem, pamięcią i długimi sekwencjami zer i jedynek. Wszystko zapisuje się w formie tekstowej podobnej do języka naturalnego, choć prostszej i bardziej zhierarchizowanej. Python jest językiem, więc podobnie jak język polski jest zrozumiały dla wszystkich osób, które go poznały. Nauka języka programowania jest łatwiejsza od nauki języka naturalnego, ponieważ nie jest związana z dyskusjami, debatami, rozmowami telefonicznymi, filmami, sztukami i innymi interakcjami. Język programowania ma za zadanie przekazać komputerowi konkretne instrukcje i zapewnić ich poprawne wykonanie. Z biegiem czasu okazało się, że komputery można zastosować niemal wszędzie tam, gdzie wcześniej podobne zadania wykonywali ludzie. Mimo to cały czas składają się z podobnych części i wykonują swe zadania w zbliżony sposób.

Programowanie to spójność

Pomimo całej złożoności różnorakich zastosowań komputerów jego podstawowa struktura i sposób działania nie ulegały znaczącym zmianom. W zasadzie wewnętrzne mechanizmy definiujące sposób działania komputera nie zmieniły się w znaczący sposób od lat pięćdziesiątych ubiegłego wieku, kiedy to do budowy komputerów zaczęto wykorzystywać tranzystory.

Dzięki tej wewnętrznej prostocie mamy pewność, że komputery mogą i powinny być wysoce spójne. Dla programistów oznacza to mniej więcej tyle, że gdy każe się komputerowi wykonać wirtualny skok, należy podać jego wysokość i miejsce lądowania, a komputer będzie wykonywał to zadanie dowolnie długo z identycznym rezultatem. Program nie powinien zmienić sposobu swego działania, o ile nie wymusimy tych zmian.

Programowanie to sterowanie

Programowanie komputera znacząco różni się od tworzenia programu dla ludzi, w potocznym znaczeniu tego słowa. W świecie rzeczywistym, gdy poprosi się osobę o wykonanie pewnego zadania, czasem trzeba się nieźle natrudzić, by prośba ta została spełniona. Jeśli na przykład planuje się przyjęcie na 30 osób i poprosi dwie z nich, by przyniosły chipsy, może się okazać, że jedna z nich przyniesie ciasteczka lub drinki.

W świecie komputerów ten problem nie istnieje. Komputer będzie wykonywał dokładnie to zadanie, które zostało mu powierzone. Nietrudno sobie wyobrazić, że trzeba zwrócić uwagę na szczegóły — komputer może wykonać nasze zadanie niezależnie od jego konsekwencji.

Jednym z celów Pythona jest zapewnienie możliwości tworzenia **bloków** kodu, które to pozwalają kreować coraz większe aplikacje w sposób modułowy i jak najbardziej zrozumiały. Stanowi to kluczowy element stylu programowania nazywanego **programowaniem obiektowym**. Główne zasady tego stylu dotyczą tworzenia godnych zaufania fragmentów, które

działają poprawnie po złączeniu ich w jedną całość, a jednocześnie są zrozumiałe i użyteczne jako niezależne komponenty. Programista ma pełną kontrolę nad sposobem wykonania poszczególnych części programu, ale jednocześnie ma możliwość rozszerzania jego funkcjonalności, gdy zajdzie taka potrzeba.

Programowanie podąża za zmianami

Programy działają na komputerach borykających się z problemami świata rzeczywistego. W tym rzeczywistym świecie plany i warunki często się zmieniają. Z racji tych zmian programista bardzo rzadko ma okazję napisać perfekcyjnie zgrany, użyteczny i elastyczny program. Najczęściej można osiągnąć tylko dwa z podanych celów. Zmiany, z którymi trzeba się zmierzyć, powodują ostrożne podchodzenie do pewnych zagadnień programistycznych. Przy odrobinie uwagi można napisać programy, które będą potrafiły stwierdzić, że zostały poproszone o wykonanie zadania przekraczającego ich możliwości i informowały o tym użytkownika. Czasem można nawet utworzyć programy, które dokładnie wskażą miejsce i przyczynę niepowodzenia. Python oferuje specjalne funkcje pozwalające opisać warunki, które musiały zajść, by uniemożliwić poprawne działanie.

Co to wszystko oznacza?

Po połączeniu tych wszystkich wymienionych elementów można wysnuć następujący wniosek: programowanie umożliwia poinformowanie komputera, jakie zadania ma wykonywać przy jednoczesnej pełnej kontroli nad tym procesem. Czasem zdarzają się wypadki. Można je często przewidzieć i zaoferować mechanizm, który pozwoli je obsłużyć w odpowiedni sposób, a nawet powrócić do pracy.

Pierwsze kroki

Na początek warto odwiedzić witrynę wydawnictwa Helion (*http://helion.pl*), pobrać znajdujące się tam przykłady oraz zainstalować program PythonCard. PythonCard to zbiór narzędzi ułatwiających programowanie w Pythonie. Są one dostępne bezpłatnie i zostały napisane specjalnie z myślą o tym języku. Zawierają edytor, codeEditor, z którego będziemy korzystać w pierwszej części książki. Ma on wiele wspólnego z domyślnym edytorem dostarczanym wraz z Pythonem (idle), ale w opinii autorów codeEditor jest lepszym narzędziem do nauki programowania, gdyż został napisany z myślą o prostszych projektach. Co więcej, sam edytor został napisany w Pythonie.

> Programy pisze się w postaci tak zwanego **kodu źródłowego**. Kod źródłowy to zestaw instrukcji w formie zgodnej z danym językiem programowania. Komputer analizuje kod i wykonuje zawarte w nim zadania.

Tak jak autorzy mają wyspecjalizowane narzędzia do pisania tekstów dla magazynów, książek i publikacji online, tak programiści mają własne wyspecjalizowane narzędzia. Dla początkującego programisty języka Python dobrym narzędziem będzie codeEditor.

Uruchamianie edytora codeEditor

Sposób uruchomienia edytora zależy od wykorzystywanego systemu operacyjnego.

Po zainstalowaniu pakietu PythonCard w systemach Linux i Unix wystarczy w oknie terminala wpisać `codeEditor`, by uruchomić edytor.

W systemie Windows edytor znajduje się w menu *Start* na zakładce *Programy/PythonCard*. Kliknięcie elementu *codeEditor* spowoduje uruchomienie edytora.

Uruchomiony po raz pierwszy edytor nie otwiera żadnego pliku — zapewnia najmniejszy możliwy punkt startowy, czyli jedno proste okienko. Po lewej stronie znajduje się numeracja wierszy. Programiści bardzo często stosują numery wierszy, by przekazać sobie informacje na temat miejsca wystąpienia błędu. Jest to jedna z podstawowych cech dobrego edytora dla programistów.

Wykorzystywanie powłoki Pythona w edytorze codeEditor

Zanim zaczniemy pisać programy, poeksperymentujemy trochę z powłoką Pythona. Na razie powłokę warto traktować jako sposób na uzyskanie dostępu do Pythona. Umożliwia ona wejście do wnętrza działającego egzemplarza interpretera Pythona w celu przekazywania kodu do wykonania. Co więcej, w tym samym czasie Python wykona wszystkie powierzone mu zadania i wyświetli odpowiedzi. Ponieważ uruchomione programy często mają **kontekst** — określone środowisko dostosowane do potrzeb programu przez programistę — powłoka jest doskonałym narzędziem, gdyż daje dostęp do utworzonego kontekstu. Czasem zamiast słowa kontekst mówi się o **środowisku** aplikacji.

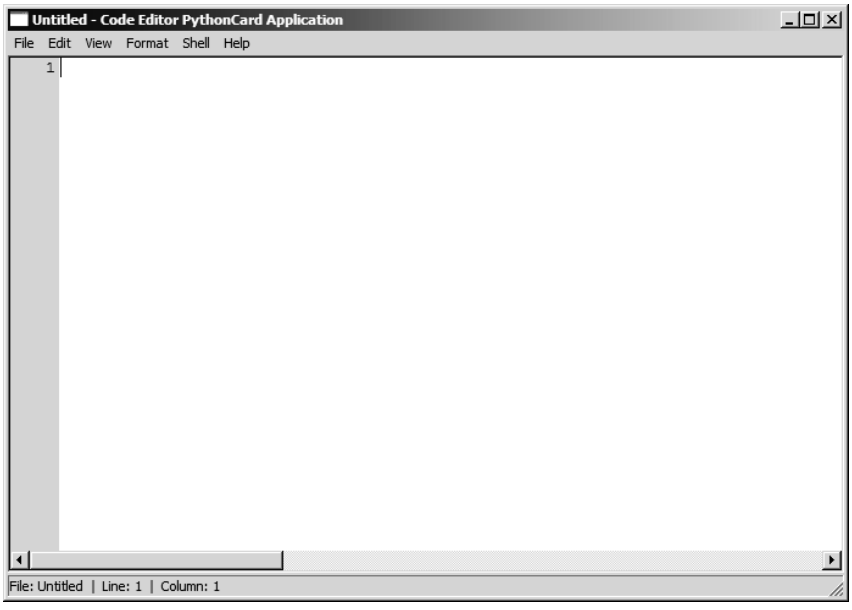
spróbuj sam Uruchomienie powłoki Pythona

Aby uruchomić powłokę Pythona z poziomu edytora codeEditor, rozwiń menu *Shell* i kliknij polecenie *Shell Window*. Pojawi się nowe okno z powłoką Pythona (to chyba nikogo nie zaskoczyło?) obok głównego okna edytora z numerami wierszy (rysunek 1.1). Podobny interfejs łatwo uzyskać bez stosowania PythonCard — wystarczy wywołać interpreter języka Python, korzystając z polecenia `python` w systemie Unix lub wywołując polecenie *Python* z menu *Start* systemu Windows.

W uruchomionej powłoce pojawiły się dodatkowe informacje, którymi nie należy się na razie przejmować (`from`, `import`, `pcapp` itp.). Na końcu znajduje się znak zachęty interpretera (`>>>`), który informuje o gotowości do przyjmowania kolejnych poleceń.

```
>>> import wx
>>> from PythonCard import dialog, util
>>> bg = pcapp.getCurrentBackground()
>>> self = bg
>>> comp = bg.components
>>>
```

Rysunek 1.1.



Jak to działa?

Edytor codeEditor jest programem napisanym w Pythonie, a zawarta w nim powłoka jest specjalnym środowiskiem, z którego skorzystamy w dalszej części książki, aby poznawanie języka Python było możliwie wygodne. Instrukcje `from`, `import` i inne zostały szczegółowo omówione w rozdziale 7. Na razie nie warto zaprzątać sobie nimi głowy.

Zaczynamy korzystać z Pythona — ciągi znaków

W tym momencie warto poeksperymentować, by poznać podstawowe zachowanie powłoki. Wpisz dowolny tekst umieszczony w cudzysłowach. Oto przykład:

```
>>> "Ten tekst tak naprawdę nic nie robi"
'Ten tekst tak naprawd\xea nic nie robi'
>>>
```

Zauważ, że zaraz po wpisaniu pierwszego cudzysłowu (") powłoka edytora zmieniła kolor tekstu dla całego zdania. Oczywiście powyższy tekst jest całkowicie prawdziwy — nic nie robi. Nie zmienia środowiska Pythona; został po prostu wyliczony przez egzemplarz interpretera jako wyrażenie, ponieważ takie polecenie wydaliśmy. Polecenie brzmiało: odczytaj podany tekst. Nie została zawarta tu żadna informacja o zmianie środowiska.

Python informuje, że odczytał wpisany tekst. Przedstawił go poniżej w odpowiedniej postaci, czyli w apostrofach. Co więcej, polskie znaki diakrytyczne zostały zastąpione odpowiednimi kodami. Gdy opiszemy **typy danych**, przedstawimy sposoby wyświetlania informacji przez Pythona w zależności od jej typu.

Czym jest ciąg znaków?

Ciąg znaków (ang. *string*) to pierwszy typ danych, którym zajmiemy się w języku Python. Komputery, a w szczególności języki programowania, dzielą wszystko na typy. Typy to kategorie elementów występujących w trakcie działania programu. Program (i programista), gdy zna typ, wie, co może zrobić z danym elementem. Jest to bardzo istotny aspekt działania komputerów, ponieważ nie rozumieją one pojęć abstrakcyjnych i muszą dokładnie wiedzieć, w jaki sposób połączyć dwie różne wartości. Gdy jasno zdefiniuje się typ tych dwóch wartości, nic nie stoi na przeszkodzie, by określić zasady rządzące ich łączeniem. Gdy znany jest typ elementu, Python wie, jakie zadania może wykonać. Podobnie dzieje się z programistą.

Dlaczego cudzysłowy?

Powróćmy do tematu ciągu znaków. W Pythonie ciąg znaków to podstawowa jednostka tekstu. W odróżnieniu od innych języków programowania pojedyncza litera jest reprezentowana przez ciąg znaków o długości 1. Zamiast wyjaśniać działanie ciągów znaków w próżni, wykonajmy kilka przykładów w powłoce Pythona i prześledźmy wyniki.

spróbuj sam Wpisywanie ciągów znaków z cudzysłowami i apostrofami w różnych postaciach

Wpisz następujące ciągi znaków, umieszczając je w cudzysłowach lub apostrofach. Na końcu każdego wiersza naciśnij klawisz *Enter*.

```
>>> "To jest kolejny tekst"
'To jest kolejny tekst'
>>> 'To także jest tekst'
'To tak\xbfe jest tekst'
>>> """To kolejny tekst, który różni
...    się od poprzednich"""
'To kolejny tekst, kt\xf3ry r\xf3\xbfni\n    si\xea od poprzednich'
```

Jak to działa?

W trakcie pisania kodu można stosować apostrofy lub cudzysłowy — Python traktuje zawarty w nich tekst dokładnie w ten sam sposób. Więcej informacji na ten temat znajdzie się w dalszej części rozdziału.

Po wykonaniu ćwiczenia Czytelnik zapewne zwrócił uwagę na kilka kwestii. Po pierwsze, że tekst początkowo zawarty w cudzysłowach pojawił się później w apostrofach. Po drugie, w trzecim przykładzie pojawiają się obok siebie trzy cudzysłowy. Po słowie „różni" stosujemy klawisz *Enter*, by przejść do kolejnego wiersza. Kolejny podrozdział dokładnie wyjaśnia wszystkie te wątpliwości.

Stosowanie apostrofów i cudzysłowów

Python stosuje trzy różne tryby oznaczania ciągów znaków. Istnieją apostrofy i cudzysłowy, na które można patrzeć w dwojaki sposób. W pierwszym przypadku są one równoważne — działają tak samo i dają te same wyniki. Dlaczego zatem stosuje się jedne i drugie? Powodów jest kilka. Ciągi znaków pełnią ogromną rolę w niemal każdym programie, a ciągi znaków definiuje się za pomocą cudzysłowów. Trzeba jednak pamiętać, że znaki te nie są znakami specjalnymi i wymysłem programistów. Stanową część zwykłego tekstu w języku polskim i najczęściej wskazują na zastosowanie cytatu. Co więcej, używa się ich również do podkreślenia pewnych słów lub zaakcentowania, że nie należy ich traktować dosłownie.

Języki programowania mają pewien problem, ponieważ stosować mogą tylko te znaki, które występują na klawiaturze. Z drugiej strony, typowy użytkownik klawiatury również ma dostęp do wszystkich znaków, więc może ich użyć do zadań innych niż programowanie! W jaki sposób wskazać, że jakiś znak jest znakiem specjalnym? W jaki sposób wskazać językowi programowania, że jako programiści chcemy uzyskać inne znaczenie cudzysłowu, a jako użytkownik programu całkowicie odmienne?

Jednym z rozwiązań tego problemu są tak zwane sekwencje specjalne. W większości języków programowania przyjmują one najczęściej formę pojedynczego znaku nazywanego **znakiem specjalnym**. Znak ten ma za zadanie znosić specjalne znaczenie innych znaków, na przykład cudzysłowów. W Pythonie znakiem tym jest lewy ukośnik (\). Jeśli więc chce się umieścić w apostrofach tekst, który również zawiera apostrofy, należy wewnętrzne apostrofy poprzedzić znakiem specjalnym, by znieść ich specjalne znaczenie i zapewnić poprawną analizę tekstu przez Pythona. Ponieważ czasem przykład jest lepszy od słów, oto on:

```
>>> 'I powiedział \'ten tekst zawiera apostrofy\''
"I powiedzia\xb3 'ten tekst zawiera apostrofy'"
```

Powróćmy do analizy wcześniejszego przykładu. Standardowo powłoka Pythona wyświetla ciąg znaków wewnątrz apostrofów. Jeśli jednak w tekście zastosuje się apostrof, a sam cytowany tekst umieści w cudzysłowach, Python zastosuje odpowiednią kombinację tych dwóch znaków, by ułatwić odczyt danych. Oto przykład:

```
>>> 'Bartek powiedział "To Jerry\'ego wina"'
'Bartek powiedzia\xb3 "To Jerry\'ego wina"'
```

Tak naprawdę miedzy apostrofami i cudzysłowami nie ma żadnej różnicy. Oczywiście należy pamiętać o tym, że ciąg znaków zaczynający się znakiem cudzysłowu nie może zakończyć się znakiem apostrofu i na odwrót. Jeśli ciąg znaków zawiera tylko cudzysłowy, można otoczyć go apostrofami i w ten sposób uniknąć pisania znaków specjalnych. Podobnie sprawa wygląda z tekstami z apostrofami — warto umieszczać je w cudzysłowach. Przydaje się to szczególnie w trakcie tworzenia zapytań SQL (język zapewniający dostęp do baz danych), gdyż w nich najczęściej trzeba stosować apostrofy. Więcej informacji na temat SQL-a znajduje się w rozdziale 14. Warto pamiętać o jednej sprawie — same cudzysłowy lub apostrofy nie dopuszczają przenoszenia tekstu do kolejnego wiersza (czyli utworzenia znaku **nowego wiersza**). Wynika to z faktu, że Python stosuje ten znak do oznaczania końca instrukcji. Ogólnie jednak znak ten służy do wskazania, że należy przenieść kursor do nowego wiersza.

> Wewnątrz ciągów znaków Python dopuszcza kilka sposobów określania znaków sterujących, które standardowo nie są wyświetlane — najczęściej znaki sterujące dotyczą wykonania pewnej akcji, na przykład przejścia do nowego wiersza. Do wskazania takich sytuacji przeważnie korzysta się ze znaku sterującego \ (pamiętaj, że ten znak jest używany także do wcześniej wymienionych celów, więc w zasadzie można mówić o znaku superspecjalnym). Kombinacja \n powoduje wstawienie znaku przejścia do nowego wiersza. Stosuje się ją najczęściej.
>
> Zanim nie nauczymy się przekazywać tekstów na ekran w inny sposób, powłoka będzie wyświetlała znaki specjalne w postaci \n zamiast dokonać rzeczywistego przejścia do nowego wiersza.

Python wykorzystuje jeszcze jeden sposób tworzenia ciągów znaków, który w zasadzie eliminuje potrzebę stosowania znaków specjalnych, a także dopuszcza tworzenie tekstów wielowierszowych. Sposób ten wymaga użycia trzech cudzysłowów lub apostrofów. Gdy tylko umieści się trzy takie znaki obok siebie, nie trzeba przejmować się stawianiem znaków specjalnych przed apostrofami i cudzysłowami stanowiącymi część tekstu. Python traktuje wszystko, co zostało wpisane, jako element ciągu znaków, dopóki ponownie nie napotka trzech znaków cudzysłowu lub apostrofu.

```
>>> """To jest specjalny ciąg znaków pozwalający złamać
...    kilka zasad, których jeszcze nie wspominaliśmy."""
'To jest specjalny ci\xb9g znak\xf3w pozwalaj\xb9cy z\xb3ama\xe6\n    kilka zasad,
kt\xf3rych jeszcze nie wspominali\x9cmy.'
```

Nietrudno zauważyć, że pomiędzy potrójnymi cudzysłowami Python jest bardzo wyrozumiały co do zawartości tekstu. Pojawia się jednak pewne pytanie — dlaczego w wyniku pojawił się znak \n? W tekście dokonaliśmy przejścia do nowego wiersza, więc dlaczego Python nie wykonał tego samego zadania w wyniku? Otóż Python stara się zachować jednolity sposób wyświetlania i zapewnić jego dokładność. Oto dlaczego \n może być bardziej preferowane od przejścia do nowego wiersza: po pierwsze, istnieje sposób poinformowania Pythona, że chce się dokonać rzeczywistego przejścia do nowego wiersza. Po drugie, gdyby domyślnie w takich sytuacjach było stosowane przejście do nowego wiersza, tak naprawdę nie byłoby wiadomo, czy znak \n naprawdę występuje, gdyż przejście mogło zostać wywołane na przykład przez zawinięcie tekstu z racji osiągnięcia końca wiersza terminala. Python, stosując znak \n, informuje o rzeczywistej zawartości tekstu.

Łączenie dwóch ciągów znaków

Bardzo często w trakcie programowania zachodzi potrzeba połączenia kilku ciągów znaków w jeden i wyświetlenia jako całość. Przykładem może być tu posiadanie osobnych rekordów dla imienia, drugiego imienia i nazwiska osoby lub też adresu rozbitego na kilka części. Python każdy z elementów może traktować jako niezależną część.

```
>>> "Jan"
'Jan'
>>> "T."
'T.'
>>> "Kowalski"
'Kowalski'
>>>
```

spróbuj sam Zastosowanie znaku + do łączenia tekstów

Istnieje kilka rozwiązań pozwalających połączyć poszczególne ciągi znaków. Pierwsze z rozwiązań polega na zastosowaniu domyślnego łączenia zakodowanego w języku Python.

```
>>> "Jan" + "T." + "Kowalski"
'JanT.Kowalski'
```

Jak to działa?

Poszczególne ciągi zostały złączone w jedną całość, ale między nimi nie pojawiły się tak pożądane odstępy. Teraz nazwisko i imię są mało czytelne, ponieważ operator + nie bierze pod uwagę żadnego specjalnego sposobu łączenia tekstów.

Aby uzyskać właściwy efekt, między poszczególnymi elementami można wstawić spacje. Podobnie jak znaki nowego wiersza, spacje są traktowane jak każdy inny znak, na przykład A, s lub 5. Python nie usuwa spacji z tekstu, choć nie są one widoczne.

```
>>> "Jan" + " " + "T." + " " + "Kowalski"
'Jan T. Kowalski'
```

Takie podejście zapewnia dużą elastyczność, ale istnieją inne rozwiązania ułatwiające tworzenie złożonych tekstów.

Złączanie ciągów znaków na różne sposoby

Kolejne rozwiązanie polega na użyciu tak zwanego **określnika formatu**. W ciągu znaków umieszcza się specjalną sekwencję, która Python zamienia później na wskazaną przez nas wartość. Choć początkowo może się wydawać, że takie podejście komplikuje całą sprawę, tak naprawdę ułatwia sterowanie wyświetlaniem, na przykład pozwala stosować pewne dodatkowe sztuczki.

spróbuj sam Wykorzystanie określnika formatu do wypełnienia ciągu znaków

Najpierw przetestujmy działanie określnika na nazwisko Jana T.:

```
>>> Jan T. %s" % ("Kowalski")
'Jan T. Kowalski'
```

Jak to działa?

Określnik formatu `%s` oznacza ciąg znaków. Kilka kolejnych określników przedstawimy w dalszych rozdziałach wraz z nowymi typami danych. Każdy określnik tymczasowo zastępuje rzeczywisty tekst. Znak `%` po ciągu znaków informuje, iż wszystkie określniki formatu z wcześniejszego tekstu należy zastąpić wartościami podanymi zaraz po nim.

Dlaczego docelowy tekst znalazł się w nawiasach? Ponieważ informują one Pythona, że powinien spodziewać się **sekwencji** zawierającej wartości mające zastąpić określniki formatu.

Sekwencje są bardzo istotną częścią programowania w Pythonie. Wkrótce zostaną opisane bardziej szczegółowo. Na razie po prostu będziemy je stosować. Trzeba pamiętać, że każdy określnik formatu występujący w oryginalnym tekście musi posiadać odpowiadającą mu wartość zastępującą. Poszczególne wartości umieszcza się właśnie w sekwencji i oddziela przecinkami (jeśli jest więcej niż jedna). W przedstawionym przykładzie sekwencji nie trzeba stosować, gdyż zastępujemy tylko jeden określnik. Nic jednak nie stoi na przeszkodzie, by jej użyć.

Nazwa **określnik formatu** nie jest przypadkowa, ponieważ określnik nie tylko umożliwia wstawienie wartości, ale również odpowiednie jej sformatowanie. Zagadnienie to przedstawimy w kolejnym przykładzie.

spróbuj sam Formatowanie ciągów znaków

Istnieje kilka bardzo prostych opcji związanych z formatowaniem ciągów znaków.

```
>>> "%s %s %10s" % ("Jan", "T.", "Kowalski")
'Jan T.   Kowalski'
>>> "%-10s %s %10s" % ("Jan", "T.", "Kowalski")
'Jan        T.   Kowalski'
```

Jak to działa?

W pierwszym wyniku nazwisko znalazło się bardziej na prawo, ponieważ w określniku formatu pojawiła się informacja o tym, by zrobić miejsce na tekst o długości 10 znaków. To właśnie oznacza `%10s`. Ponieważ słowo `Kowalski` ma tylko 8 znaków, po jego lewej stronie pojawiły się dwa dodatkowe znaki spacji.

W drugim przykładzie T. pozostało samotne na środku, a pierwsze imię i nazwisko znalazły się na bokach. Opisaliśmy już działanie prawego określnika. Określnik po lewej stronie działa bardzo podobnie, choć na odwrót — także rezerwuje miejsce na 10 znaków tekstu, ale zastosowano kod `%-10s`. Znak minusa oznacza, że przekazany tekst należy umieścić po lewej zamiast po prawej stronie dziesięcioznakowego miejsca.

Wyświetlanie tekstów za pomocą instrukcji print

Do tej pory Python wyświetlał wprowadzane przez nas teksty w taki sposób, w jaki przechowuje je wewnętrznie. Nie zastosowaliśmy żadnego polecenia, które powodowało wyświetlenie tekstu użytkownikowi. Zdecydowania większość programów przedstawia użytkownikom różnego rodzaju informacje: statystyki rozgrywek sportowych, tabelę odjazdów pociągów, witryny internetowe, a nawet informacje rozliczeniowe za telefon. Najważniejsze jest, by zostały one poprawnie zrozumiane przez osobę, która je zobaczy.

spróbuj sam Wyświetlanie tekstu

Większość języków posiada specjalne polecenia zapewniające wyświetlanie tekstu użytkownikom. W Pythonie najprostszym poleceniem tego typu jest funkcja `print`.

```
>>> print "%s %s %10s" % ("Jan", "T.", "Kowalski")
Jan T.   Kowalski
```

Zauważ, że tym razem całego tekstu nie otaczają żadne cudzysłowy ani apostrofy. Jest to niezmiernie istotne — po raz pierwszy wykonaliśmy instrukcję, która spowodowałaby wyświetlenie tekstu użytkownikowi!

Jak to działa?

Instrukcja `print` jest **funkcją** — specjalną nazwą, którą nadaje się fragmentowi programu wykonującemu jedno lub więcej ściśle określonych zadań. Najczęściej nie trzeba się w ogóle przejmować sposobem wykonania tych zadań. (Dopiero w rozdziale 5., w którym zajmiemy się pisaniem własnych funkcji, zastanowimy się nad szczegółami ich działania.)

Funkcja `print` jest przykładem funkcji wbudowanej, czyli funkcji stanowiącej część języka Python, w przeciwieństwie do pozostałych funkcji pisanych przez siebie lub też innych programistów. Funkcja `print` wyświetla dane na wyjściu, czyli przedstawia coś użytkownikowi w miejscu, które może zobaczyć: terminalu, oknie, drukarce, a nawet wyświetlaczu ciekłokrystalicznym. Inne funkcje zajmują się pobieraniem danych wejściowych, czyli danych pochodzących od użytkownika, z pliku, z sieci itp. Python wszystkie te funkcje łączy w jeden pakiet funkcji wejścia-wyjścia. Funkcje tego typu dotyczą wszystkich operacji zajmujących się pobieraniem lub też wysyłaniem danych na zewnątrz programu. Więcej informacji na ten temat znajduje się w rozdziale 8.

Podsumowanie

W niniejszym rozdziale przedstawiliśmy, w jaki sposób korzystać z edytora programistycznego codeEditor napisanego w Pythonie i służącego do edycji programów w tym języku. Poza edycją plików codeEditor obsługuje również powłokę Python, która ułatwia testowanie instrukcji Pythona.

Korzystając z powłoki, poznaliśmy podstawy obsługi ciągów znaków, wliczając w to łączenie kilku mniejszych ciągów znaków w jeden duży, a także zastosowanie określników formatu. Określnik `%s` informuje o wstawianiu ciągu znaków. Zastosowanie w nim wartości liczbowej, na przykład `%8s`, pozwala określić ilość miejsca przewidzianego dla tekstu (w przykładzie dokładnie 8 znaków). W kolejnych rozdziałach przedstawimy określniki formatu dla innych typów danych.

Zaznajomiliśmy się również z wyświetlaniem utworzonych ciągów znaków. Wyświetlanie tekstu to operacja wejścia-wyjścia (szczegółowe informacje na ten temat znajdują się w rozdziale 8.). Funkcja `print` wyświetla na ekranie przekazany do niej ciąg znaków.

W kolejnym rozdziale zajmiemy się obsługą liczb, a także operacjami, które można na nich przeprowadzać, łączeniem liczb i tekstów, by wyświetlić wyniki za pomocą funkcji `print`. Wykorzystamy w tym celu poznaną w tym rozdziale technikę określników formatu.

Ćwiczenia

1. W powłoce Pythona wpisz następujący tekst `"Szalej dziecinko, szalej,\n\tna wierzchołkach drzew,\t\tgdy wiatr zawieje\n\t\t\t spadniesz mi na ziemię."`. Poeksperymentuj z liczbą i rozmieszczeniem sekwencji specjalnych `\t` i `\n`. Sprawdź efekt. Co się stało?
2. W powłoce Pythona zastosuj ten sam tekst co w poprzednim ćwiczeniu, ale użyj funkcji `print` do jego wyświetlenia. Ponownie poeksperymentuj z rozmieszczeniem sekwencji specjalnych `\t` i `\n`. Jak sądzisz, co się stało?