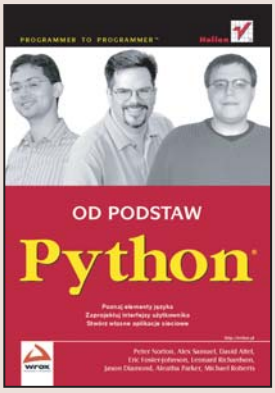
Python. Od podstaw



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

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.

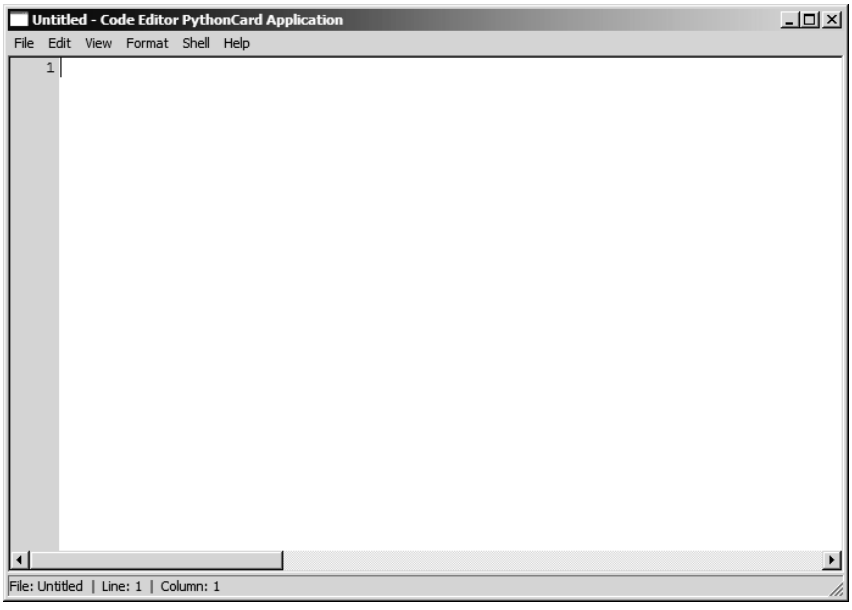
spróbuj sam Uruchomienie powłoki Pythona

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?