Helion

Microsoft

Python

Zacznij programować!

Rob Miles

Tytuł oryginału: Begin to Code with Python

Tłumaczenie: Radosław Meryk

ISBN: 978-83-283-4654-3

Authorized translation from the English language edition, entitled: BEGIN TO CODE WITH PYTHON, First Edition, ISBN 9781509304523; by Rob Miles; published by Pearson Education, Inc., publishing as Microsoft Press.

HELION SA
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/pytzap.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/pytzap
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Część II Zaawansowane programowanie

Część III Przydatny Python

2.

Python i programowanie

Czego się nauczysz?

W tym rozdziale zaczniesz pracować z Pythonem. Jednak zanim to zrobisz, pobawimy się trochę w detektywa i spróbujemy ustalić, co sprawia, że ktoś jest programistą, i co naprawdę robi program komputerowy. Przyjrzymy się także językowi programowania Python i odkryjemy, w jaki sposób pasuje do tematyki tworzenia oprogramowania.

Co to znaczy być programistą?

Jeśli dotychczas jeszcze nie programowałeś, nie martw się. Programowanie to nie fizyka jądrowa. Najtrudniejszą częścią nauki programowania są początki — gdy musisz zapoznać się z mnóstwem pojęć, które mogą być mylące. Jeśli jednak uważasz, że nauka programowania wydaje się wyzwaniem, któremu nie będziesz w stanie sprostać, to zdecydowanie sugeruję, abyś odłożył te myśli na bok. Programowanie jest tak samo proste jak zorganizowanie przyjęcia urodzinowego dla grupy dzieci.

Programowanie a planowanie przyjęcia

Gdybyście organizowali przyjęcie urodzinowe dla dzieci, musielibyście zdecydować, kogo zaprosić. Trzeba by zapamiętać, kto lubi pizzę wegetariańską oraz które dzieci nie mogą siedzieć obok siebie, bo się pokłócą. Musielibyście ustalić, jakie prezenty zabierze do domu każde dziecko oraz co będzie robić na przyjęciu. Musielibyście tak zorganizować czas, aby magik nie przybył akurat wtedy, kiedy będzie podawane jedzenie. Dla ułatwienia organizacji przyjęcia można by skorzystać z list podobnych do tych, które pokazano na rysunku 2.1. Programowanie to dokładnie to samo — wszystko sprowadza się do organizacji.

LISTA GOŚCI	MENU	PROGRAM	PREZENTY DLA GOŚCI
Robert	Pizza	15:00 Przyjazd	Kapelusz
Marysia	Chipsy	15:30 Gry Xbox	Gwizdek (być może)
Dawid	Woda sodowa	16:30 Jedzenie	Słodycze
Janina	Cola	17:15 Magik	Puzzle
Krzyś	Sok pomarańczowy		Książka
Iza			
Maja			
Zosia			

Rysunek 2.1. Planowanie przyjęcia jest bardzo podobne do programowania. Trzeba zadbać o dobrą organizację

Jeśli potrafisz zorganizować przyjęcie, to potrafisz również napisać program. To, co się dzieje w programie, jest trochę inne, ale podstawowe zasady są takie same. Ponieważ program zawiera elementy, które tworzysz i którymi zarządzasz (w przeciwieństwie do niesfornych dzieci), masz pełną kontrolę nad tym, co się dzieje. Co więcej, kiedy już nabierzesz trochę doświadczenia w programowaniu, będziesz potrafił podejść do zadań wykonywanych w życiu w sposób bardziej usystematyzowany. Zatem odrobina doświadczenia w programowaniu może sprawić, że staniesz się lepszym organizatorem.

Większość ludzi definiuje programowanie jako „zarabianie dużych sum pieniędzy na robieniu czegoś, czego nikt nie potrafi zrozumieć". Ja definiuję programowanie jako „ustalenie rozwiązania określonego problemu i wyrażenie go w formie, którą potrafi zrozumieć i wykonać system komputerowy". Z powyższej definicji wynikają jedna lub dwie rzeczy:

- Zanim napiszesz program, który wykona określone działanie, musisz potrafić rozwiązać problem samodzielnie.
- Komputer musi rozumieć to, co próbujesz mu powiedzieć.

Program można porównać do przepisu kulinarnego. Jeśli nie wiesz, jak upiec ciasto, nie będziesz w stanie powiedzieć komuś, jak ma to zrobić. A jeśli osoba, z którą rozmawiasz, nie rozumie instrukcji typu „przygotuj mąkę i cukier do zmieszania", to nadal nie będziesz w stanie wytłumaczyć jej, jak upiec ciasto.

Aby utworzyć program, musisz skorzystać z wypracowanego rozwiązania, a następnie zapisać je w prostych krokach, które komputer jest w stanie wykonać.

Programowanie a problemy

Czasami porównuję programistów do hydraulików. Hydraulik przychodzi do pracy z dużą torbą narzędzi i części zamiennych. Przygląda się problemowi przez dłuższą chwilę, otwiera torbę, wyjmuje różne narzędzia i części, dopasowuje je do siebie i rozwiązuje problem. Programowanie jest bardzo podobne. Masz problem do rozwiązania i dysponujesz dużą torbą narzędzi — w tym przypadku językiem programowania. Przyglądasz się problemowi przez chwilę i zastanawiasz się, jak go rozwiązać, a następnie dopasowujesz elementy języka, aby wyeliminować problem. Sztuka programowania polega na ustaleniu tych elementów, które trzeba wyjąć z torby narzędzi po to, by rozwiązać każdą część problemu.

Interesujący składnik programowania stanowi sztuka podzielenia problemu na zbiór instrukcji, które można przekazać komputerowi. Jednak umiejętność programowania nie sprowadza się tylko do nauczenia się języka. Programowanie nie jest również wyłącznie kwestią wymyślenia programu, który rozwiązuje dany problem. Podczas pisania programu trzeba wziąć pod uwagę wiele czynników i nie wszystkie one są bezpośrednio związane z konkretnym problemem. Na początek załóżmy, że piszesz swoje programy dla klienta. Klient ma problem i chciałby, żebyś napisał program, który go rozwiąże. Zakładamy także, że klient wie jeszcze mniej o komputerach niż my!

Początkowo nie będziemy nawet wspominać o języku programowania, typie komputera lub podobnych rzeczach. Najpierw powinniśmy się upewnić, że wiemy, czego chce klient. Ponieważ programiści chwalą się, że potrafią wymyślać rozwiązania, gdy tylko dostaną problem, natychmiast zaczynają myśleć o sposobach rozwiązania — to niemal odruchowe działanie. Niestety, wiele projektów oprogramowania nie powiodło się ze względu na to, że próbowano rozwiązywać niewłaściwy problem. Znalezienie idealnego rozwiązania problemu, który nie

jest problemem klienta, to zjawisko występujące zaskakująco często w rzeczywistym świecie. Twórcy oprogramowania po prostu nie ustalili tego, co było potrzebne lub pożądane. Zamiast tego zbudowali to, co uważali za potrzebne. Klienci założyli, że skoro programiści przestali zadawać pytania, to zaczęli tworzyć właściwe rozwiązanie. Dopiero przy ostatecznym przekazaniu programu odkryli niewygodną prawdę. Jest bardzo ważne, aby programista nie zaczynał pracy, zanim nie będzie dokładnie wiedział, co jest potrzebne.

Najgorszą rzeczą, jaką możesz od razu powiedzieć klientowi, jest zdanie „potrafię to zrobić". Zamiast tego powinieneś najpierw zapytać: „Czy właśnie tego chce klient?", „Czy naprawdę rozumiem, na czym polega problem?". Zadawanie tych pytań jest czymś w rodzaju samodyscypliny. Zanim rozwiążesz problem, powinieneś się upewnić, że posiadasz dokładną definicję tego, na czym problem polega. Ta definicja powinna być zrozumiała i akceptowalna zarówno dla Ciebie, jak i klienta.

W realnym świecie taką definicję czasem nazywa się *specyfikacją funkcjonalną projektu* (ang. *functional design specification* — **FDS**). Specyfikacja FDS mówi dokładnie, czego chce klient. Podpisujesz ją zarówno Ty, jak i klient. Jeśli dostarczysz system, który zachowuje się zgodnie ze specyfikacją projektu, klient musi za niego zapłacić. Gdy już masz specyfikację projektu, możesz zacząć rozważać sposoby rozwiązania problemu.

Mógłbyś pomyśleć, że posiadanie specyfikacji nie będzie konieczne, jeśli piszesz program dla siebie, ale to nie jest prawda. Napisanie specyfikacji w jakiejś formie zmusza do myślenia o problemie na bardzo szczegółowym poziomie. Zmusza także do myślenia o tym, czego Twój system nie będzie robić. Potrzebujesz takiej przejrzystości w równym stopniu, kiedy budujesz program dla siebie, jak i wtedy, kiedy pracujesz z klientem. Specyfikacja na samym początku określa oczekiwania.

KĄCIK PROGRAMISTY

Zawsze powinna istnieć jakaś forma specyfikacji

Napisałem wiele programów za pieniądze. Nigdy nie napisałbym programu bez otrzymania wcześniej solidnej specyfikacji. Zdefiniowanie specyfikacji jest niezbędne nawet wtedy (a może zwłaszcza wtedy), gdy wykonuję pracę dla przyjaciela.

W nowoczesnych technikach wytwarzania oprogramowania klient znajduje się w centrum uwagi i na bieżąco angażuje się w proces projektowania. Takie podejście jest bardzo wygodne, ponieważ na początku prac nad projektem bardzo trudno jest uzyskać specyfikację w ostatecznej postaci. Programista nie wie zbyt wiele na temat działalności klienta, a klient nie zna ograniczeń i możliwości technologii, których można użyć do rozwiązania problemu. Dobrym pomysłem jest wykonanie serii wersji rozwiązania i omówienie każdej wersji z klientem przed przystąpieniem do opracowywania następnej. Technikę tę nazywamy *prototypowaniem*.

Takie podejście do rozwiązywania problemów sprawdza się niezależnie od wykorzystywanych języków programowania. Kwestie zapewnienia odpowiednich specyfikacji i unikania założeń są tak samo ważne, gdy próbujesz cokolwiek zorganizować — w tym również przyjęcie urodzinowe.

Programiści a inne osoby

Ustalenie, czego chce klient, to jeden z najważniejszych aspektów każdego zadania programistycznego. Jednak komunikacja z innymi ludźmi jest ważna także w wielu innych sytuacjach. Być może chcesz przekonać bogatego inwestora, że masz pomysł na następne wielkie oprogramowanie. Być może chcesz przekonać potencjalnego klienta, że dysponujesz najlepszym rozwiązaniem jego problemów.

Nie wszyscy programiści od początku są świetni w komunikacji. Jednak trzeba zapamiętać, że umiejętności komunikacyjnych można się nauczyć, tak samo jak nowego języka programowania. Stawanie się lepszym w komunikacji może oznaczać wyjście poza strefę komfortu — nikt nie lubi stać przed publicznością po raz pierwszy. Ale wystarczy trochę wprawy, aby opanować umiejętności komunikacyjne i znacznie zwiększyć swoje szanse na odegranie znaczącej roli w tym biznesie.

Skuteczna komunikacja dotyczy także dokumentów pisemnych. Umiejętność tworzenia tekstu, który inni mogą przeczytać, jest bardzo przydatna, a — tak jak wcześniej wspominałem — najlepszym sposobem na doskonalenie tych umiejętności jest praktyka. Radzę zacząć pisać blog lub dziennik. To nic, że początkowo Twój blog czyta tylko Twoja mama. Ważne, że piszesz regularnie. Jeśli piszesz o czymś, czym się interesujesz (ja — niespodzianka — piszę o programowaniu pod adresem *www.robmiles.com*), szybko osiągniesz postępy.

KĄCIK PROGRAMISTY

Programiści, którzy potrafią się dobrze komunikować z otoczeniem, zarabiają najwięcej pieniędzy i otrzymują najciekawszą pracę

Można nieźle zarabiać na programowaniu nawet wtedy, gdy potrafisz komunikować się tylko pojedynczymi słowami i mruczeniem — pod warunkiem, że potrafisz szybko napisać kod, który spełnia podane wymagania. Ale najciekawsze zadania trafiają do programistów, którzy potrafią się dobrze komunikować. To oni potrafią sprzedawać swoje pomysły i najlepiej rozmawiać z klientami, dzięki czemu mogą się dowiedzieć, czego chce klient.

Komputery jako procesory danych

Teraz, gdy wiemy, co robią programiści, możemy zacząć się zastanawiać, czym są komputery i co sprawia, że są tak wyjątkowe.

Maszyny i komputery a ludzie

Ludzie są rasą producentów narzędzi. Wymyślamy rzeczy, które ułatwiają nam życie, i robimy to od tysięcy lat. Zaczęliśmy od urządzeń mechanicznych, takich jak pług, dzięki któremu rolnictwo stało się bardziej wydajne, w poprzednim stuleciu przełączyliśmy się na urządzenia elektroniczne, a ostatnio na komputery.

W miarę jak komputery stały się mniejsze i tańsze, znalazły drogę do otaczających nas rzeczy. Wiele urządzeń (np. telefon komórkowy) może działać tylko dlatego, że da się zamontować w nich komputer, który jest ich centralną częścią. Musimy jednak pamiętać, co robią komputery — automatyzują operacje, które wcześniej wymagały pracy mózgu. W komputerach nie ma nic szczególnie inteligentnego. Komputer po prostu postępuje zgodnie z instrukcjami, które zostały mu przekazane.

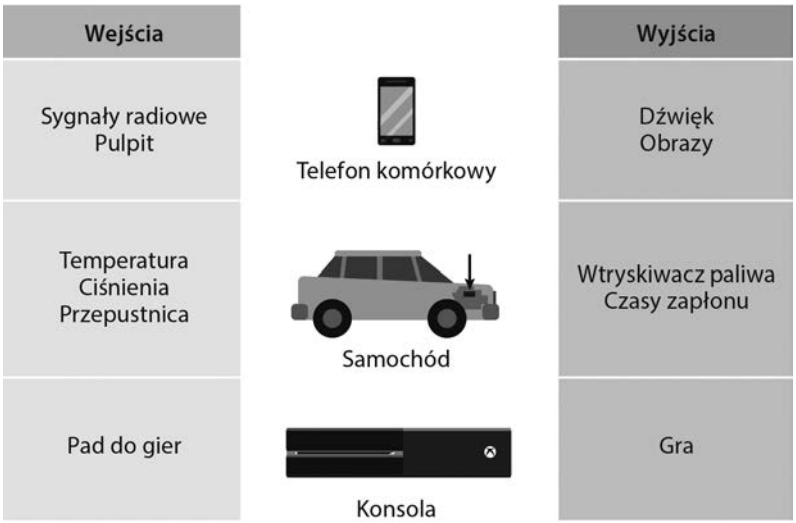
Komputer przetwarza dane w podobny sposób, w jaki maszyna do produkcji kiełbasek przetwarza mięso: coś wkłada się na jednym końcu, wykonywane jest pewne przetwarzanie i coś wychodzi z drugiej strony. Program można porównać do instrukcji, jakie trener przekazuje drużynie futbolu amerykańskiego lub piłki nożnej przed rozpoczęciem meczu. Trener może powiedzieć coś w stylu: „Jeśli będą atakować z lewej strony, to Jerzy i Krzysztof pilnują skrzydła, ale jeśli piłka będzie zagrana na środek pola, to Jerzy idzie za piłką". Następnie, podczas gry, zespół będzie reagował na wydarzenia w sposób, który powinien pozwolić pokonać przeciwników.

Istnieje jedna ważna różnica pomiędzy programem komputerowym a sposobem, w jaki zespół może się zachowywać podczas meczu piłki nożnej. Piłkarz rozpozna sytuację, gdy otrzyma jakieś bezsensowne instrukcje. Na przykład gdy trener powie: „Jeśli będą atakować z lewej strony, to Jerzy śpiewa pierwszą zwrotkę hymnu narodowego, a następnie biegnie jak najszybciej w kierunku wyjścia", to gracz zaprotestuje.

Niestety, program nie jest świadomy wrażliwości danych, które przetwarza, tak samo jak maszyna do kiełbasek nie jest świadoma tego, jakie przerabia mięso. Jeśli wrzucimy do niej rower, to maszyna spróbuje zrobić z niego kiełbaski. Jeśli przekażemy do komputera bezsensowne dane, to przetworzy je na bezsensowne wyniki. W przypadku komputerów dane są po prostu wzorcem sygnałów, które muszą zostać przetworzone w jakiś sposób, aby utworzyć inny wzorzec sygnałów. Program komputerowy to sekwencja instrukcji informujących komputer, co należy zrobić z danymi wejściowymi i jaką postać powinny przyjąć dane wyjściowe.

Oto przykłady typowych aplikacji przetwarzania danych (rysunek 2.2):

- Telefon komórkowy — mikrokomputer w Twoim telefonie odbiera sygnały radiowe i przekształca je na dźwięk. Jednocześnie pobiera sygnały z mikrofonu i przekształca je na wzorce bitów, które zostaną przesłane przez radio.
- Samochód — mikrokomputer w silniku pobiera informacje z czujników informujących o aktualnych obrotach silnika, prędkości jazdy, zawartości tlenu w powietrzu, ustawieniach akceleratora itd. Wytwarza też sygnały napięciowe, które kontrolują ustawienia wtrysku paliwa, czasy dla świec zapłonowych i inne elementy optymalizujące wydajność silnika.
- Konsola do gier — komputer pobiera instrukcje z kontrolerów i wykorzystuje je do zarządzania sztucznym światem, który tworzy dla gracza.



Rysunek 2.2. Komputery w urządzeniach

Większość współczesnych umiarkowanie złożonych urządzeń zawiera komponenty do przetwarzania danych, które są wykorzystywane w celu optymalizacji ich wydajności. Niektóre z tych komponentów istnieją tylko dlatego, że możemy je wbudować w urządzenia. Rozwój internetu rzeczy wprowadza komputery do ogromnego zakresu obszarów. Ważne jest, aby myśleć o przetwarzaniu danych jako o czymś więcej niż narzędziu do obliczeń płac — obliczania i drukowania wyników (tradycyjne zastosowania komputerów). Inżynierowie oprogramowania z pewnością poświęcą mnóstwo czasu na montowanie do urządzeń komponentów przetwarzania danych, aby nimi sterować. Dzięki tym wbudowanym systemom wiele osób będzie korzystało z komputerów, nawet o tym nie wiedząc!

KĄCIK PROGRAMISTY

Oprogramowanie może być kwestią życia i śmierci

Pamiętaj, że pozornie nieszkodliwe programy mogą zagrażać życiu. Na przykład lekarz może używać arkusza kalkulacyjnego, który napisałeś, do ustalania dawki leków dla pacjentów. W takim przypadku defekt programu może spowodować fizyczne uszkodzenia (nie sądzę, żeby lekarze posługiwali się oprogramowaniem z błędami — ale nigdy nie wiadomo). Przerażający opis tego, co może pójść źle, gdy programiści nie zwracają uwagi na podstawy, można znaleźć w internecie. W tym celu poszukaj frazy *Therac-25*.

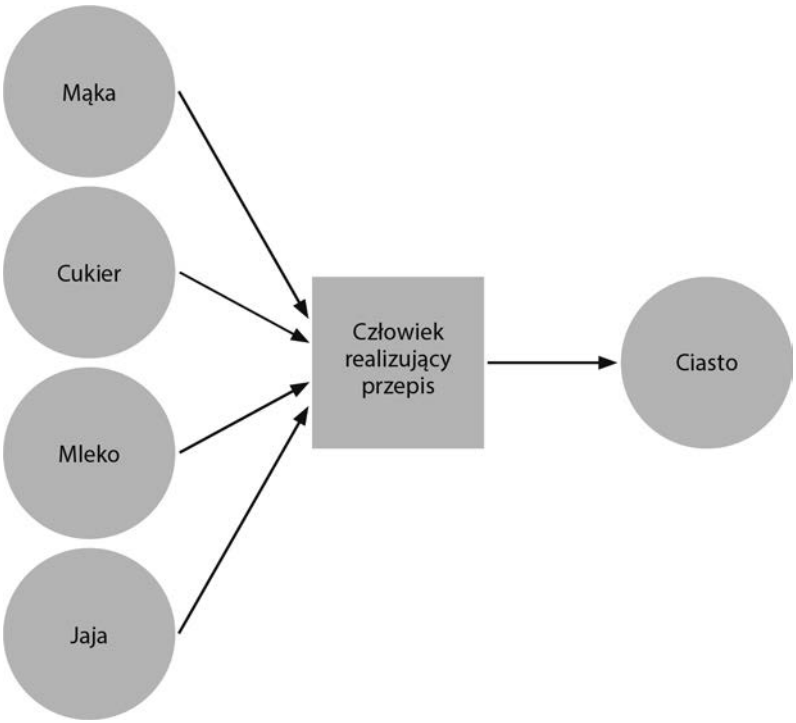
Programy jako procesory danych

Zadania realizowane przez każdy komputer pokazano na rysunku 2.3. Do komputera trafiają dane, komputer coś z nimi robi, a następnie zwraca przetworzone dane. Format, jaki przyjmują dane, oraz to, co oznaczają dane wyjściowe, a także co robi program, zależy wyłącznie od nas.

Rysunek 2.3. Komputer jako procesor danych

Jak wspomniano wcześniej, program można porównać do przepisu kulinarnego. Zilustrowano to na rysunku 2.4.

W tym przykładzie rolę komputera odgrywa kucharz, a przepis to program, który kontroluje to, co kucharz robi ze składnikami. W przepisie może występować wiele różnych składników, a program może pracować z wieloma różnymi danymi wejściowymi. Na przykład może pobierać Twój wiek i tytuł filmu, który chcesz zobaczyć, i na tej podstawie zwracać wynik określający, czy możesz obejrzeć ten film.



Rysunek 2.4. Przepisy a programy

Python jako procesor danych

Język programowania Python również można traktować jako procesor danych (rysunek 2.5). Kod napisany w Pythonie trafia do silnika Pythona, który następnie generuje jakieś wyniki.

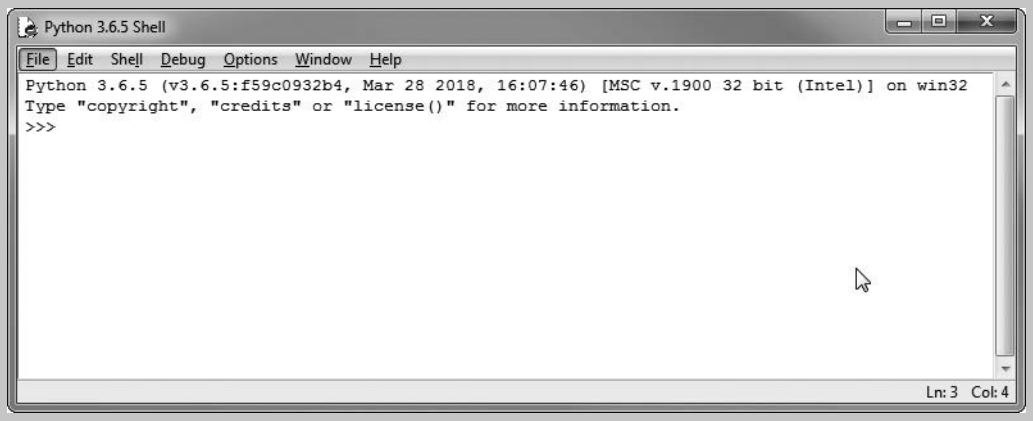
Rysunek 2.5. Python jako procesor danych

Czasami, jak już widzieliśmy, wyjście reprezentuje komunikat o błędzie (np. gdy wpiszemy `hello`). Innym razem mogą to być filozoficzne zdania opisujące naturę języka Python (jak wtedy, gdy wpisaliśmy `import this`). Spróbujmy użyć powłoki poleceń Pythona, aby dowiedzieć się więcej o tym, jak ten język działa.

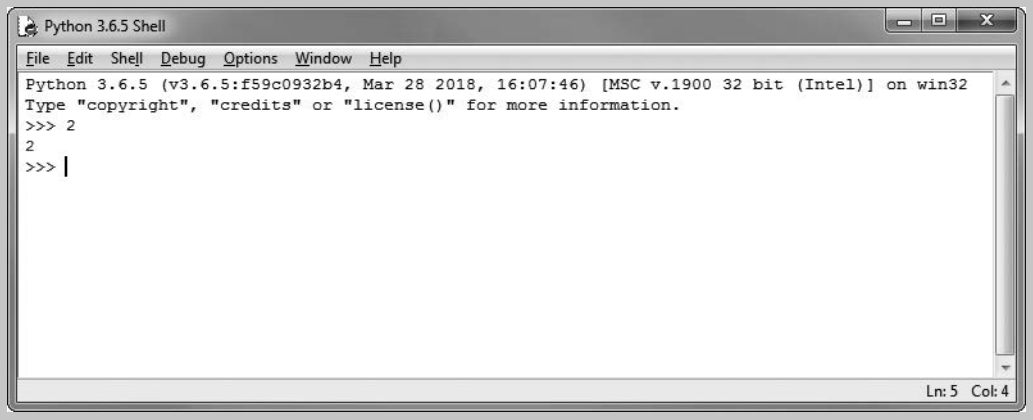
ZRÓB TO SAM

Porozmawiaj z Pythonem

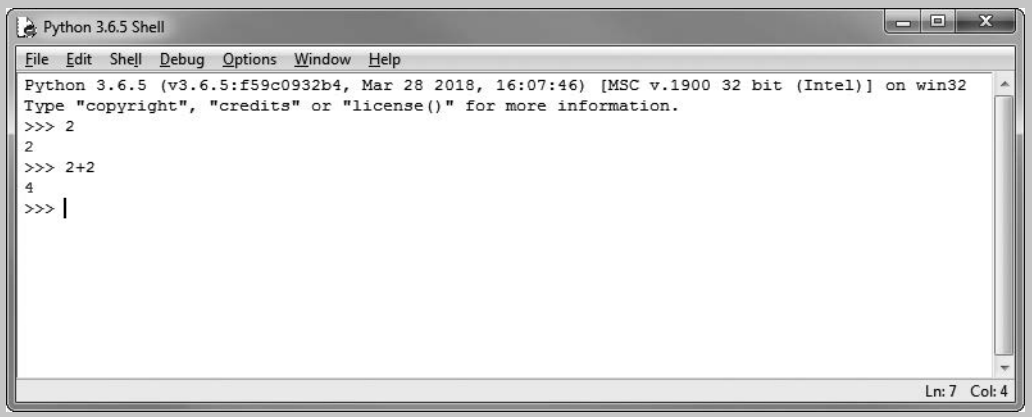
Kiedy ostatnio rozmawialiśmy z Pythonem, niewiele powiedzieliśmy. Teraz przeprowadzimy bardziej szczegółową rozmowę i zobaczymy, czego możemy się dowiedzieć o tym, jak działa ten język. Najpierw musimy użyć polecenia IDLE, aby uruchomić powłokę Pythona — tak jak to zrobiliśmy w poprzednim rozdziale:



W rozdziale 1. próbowaliśmy przywitać się z Pythonem, ale to nie skończyło się pomyślnie. Spróbujmy zatem wprowadzić do powłoki poleceń coś, o czym wiemy, że komputery rozumieją — np. liczbę. Wpisz wartość 2 i naciśnij *Enter*:

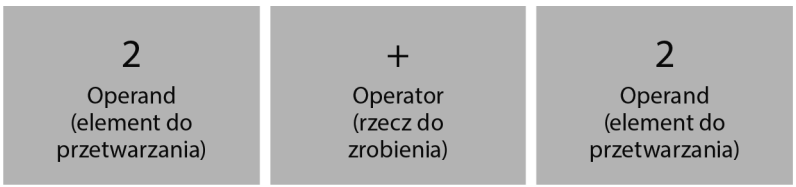


Tym razem nie wyświetli się komunikat o błędzie; wyświetli się wartość 2. Wygląda na to, że powłoka Pythona potrafi znaleźć odpowiedź i nam ją przesłać. Możemy to udowodnić, wprowadzając sumę — np. 2+2:



Tym razem, zamiast wyświetlić `echo 2+2`, powłoka Pythona obliczyła wynik sumy i nam go zwróciła.

Wygląda na to, że powłoka Pythona odbiera od nas instrukcje i na tej podstawie wykonuje jakieś działania. W rzeczywistości tak właśnie się dzieje. Wewnętrznie Python jest narzędziem do obliczania wartości wyrażeń, co jest wyszukanym sposobem na stwierdzenie, że Python coś dla nas robi. Jeśli wprowadzisz wyrażenie w powłoce Pythona, powłoka wyświetli jego wartość jako odpowiedź. Proste wyrażenie zaprezentowano na rysunku 2.6.

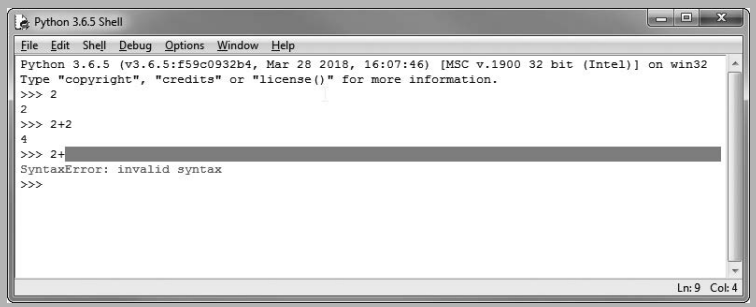


Rysunek 2.6. Anatomia prostego wyrażenia

Elementy, na których działa wyrażenie, nazywane są *operandami*. Elementy, które wykonują rzeczywistą pracę, nazywają się *operatorami*. Wyrażenie 2+2 zawiera dwa operandy (dwie liczby 2) i jeden operator (plus). Kiedy wprowadzisz wyrażenie do powłoki Pythona, zidentyfikuje ona operatory i operandy, a następnie obliczy działanie i da odpowiedź.

Błędne wyrażenia

Wcześniej zobaczyliśmy, co może się stać, jeśli wpiszemy w powłoce coś, czego Python nie rozumie. Wyświetli się komunikat o błędzie. To samo się stanie, jeśli przekażesz Pythonowi nieprawidłowe wyrażenie.



Programista wprowadził wyrażenie 2+. To wyrażenie nie jest poprawne, więc powłoka Pythona wyświetliła czerwony pasek i czerwony komunikat o błędzie.

Python bardzo dobrze się sprawdza w obliczaniu wartości wyrażeń. Jeśli chcesz, możesz używać Pythona zamiast kalkulatora. Wyrażenia są obliczane w taki sam sposób, w jaki wykonałby je matematyk. Python uwzględnia kolejność działań — np. wykonuje mnożenie przed dodawaniem i przestrzega nawiasów.

Aby zbadać wyrażenia, możemy skorzystać z powłoki Pythona w celu przeprowadzenia kilku eksperymentów. Od tej chwili, zamiast pokazywania zrzutów ekranu powłoki Pythona, będę zamieszczał same wyniki, które wyświetlają się w środowisku IDLE. Zatem trzy poprzednie polecenia Pythona, które wydaliśmy, wyglądałyby tak:

```
>>> 2
2
>>> 2+2
4
>>> 2+
SyntaxError: invalid syntax
```

Wprowadzany tekst jest wyświetlany na czarno, dane wyjściowe z Pythona są wyświetlane na niebiesko, a monity poleceń wyświetlają się w kolorze brązowym. Błędy wyświetlają się na czerwono.

ANALIZA KODU

Wyrażenia w Pythonie

Od czasu do czasu w książce pojawiają się sekcje „ANALIZA KODU", zawierające pytania na temat kodu, który właśnie zaprezentowaliśmy. Zanim przeczytasz odpowiedź, możesz spróbować odpowiedzieć na te pytania samodzielnie.

Pytanie: Jak sądzisz, co by się stało, gdybyś spróbował obliczyć wartość wyrażenia `2+3*4`?

Odpowiedź: Operator `*` (gwiazdka) oznacza mnożenie. Python używa gwiazdki zamiast znaku × (symbolu mnożenia) stosowanego w matematyce. W matematyce zawsze wykonujemy operacje o wyższym priorytecie, takie jak mnożenie i dzielenie, przed dodawaniem, więc spodziewam się, że powyższe wyrażenie wyświetli wartość 14. Wyraz `3*4` zostanie obliczony najpierw, co daje wynik 12, a do niego zostanie dodana wartość 2. Jeśli spróbujesz tego w środowisku IDLE, powinieneś zobaczyć to, czego oczekiwałeś:

```
>>> 2+3*4
14
```

Pytanie: Jak sądzisz, co by się stało, gdybyś spróbował obliczyć wartość wyrażenia `(2+3)*4`?

Odpowiedź: Nawiasy obejmują obliczenia, które należy wykonać najpierw, więc w powyższym wyrażeniu spodziewam się, że obliczona zostanie wartość 5 (`2 + 3`), a następnie ta wartość zostanie pomnożona przez 4, co da wynik `20`.

```
>>> (2+3)*4
20
```

Pytanie: Jak sądzisz, co by się stało, gdybyś spróbował obliczyć wartość wyrażenia `(2+3*4`?

Odpowiedź: To dość ciekawy przypadek. Powinieneś go wypróbować w powłoce Pythona. W odpowiedzi na wprowadzenie tego wyrażenia Python powie: „Wyrażenie, które próbuję obliczyć, jest niekompletne. Potrzebuję nawiasu zamykającego". Tak więc powłoka Pythona czeka na więcej informacji od Ciebie. Jeśli wpiszesz nawias zamykający, który uzupełni wyrażenie, powłoka Pythona obliczy wartość i wyświetli wynik. Możesz nawet wprowadzić w drugim wierszu następne komponenty wyrażenia.

```
>>> (2+3*4
  )
14
```

Pytanie: Jak sądzisz, co by się stało, gdybyś spróbował obliczyć wartość wyrażenia)2+3*4?

Odpowiedź: Jeśli powłoka Pythona zauważy nawias zamykający przed otwierającym, natychmiast zorientuje się, że coś jest nie tak, i wyświetli komunikat o błędzie.

```
>>> )2+3*4
SyntaxError: invalid syntax
```

Zwróćmy uwagę, że powłoka poleceń próbuje pomóc Ci ustalić, gdzie znajduje się błąd, podświetlając niepoprawny znak.

Python jako język skryptowy

Możemy używać powłoki Pythona do prowadzenia konwersacji takich jak pokazana powyżej, ponieważ Python jest „skryptowym" językiem programowania. Powłokę Pythona można porównać do rodzaju robota wykonującego polecenia Pythona, które do niej wprowadzimy. Innymi słowy: mówimy powłoce poleceń, co chcemy, żeby zrobił program za pomocą języka Python. Jeśli instrukcje nie mają sensu dla „robota", powłoka mówi nam, że nie potrafi ich zrozumieć (zwykle wyświetla je na czerwono).

Proces czytania programu, a następnie działania zgodnie z zawartymi w nim instrukcjami nazywamy *interpretacją* programu. Aktorzy zarabiają na życie, interpretując tekst sztuki; komputery rozwiązują dla nas problemy, interpretując instrukcje programów.

KĄCIK PROGRAMISTY

Nie wszystkie języki programowania działają tak jak Python

Nie wszystkie języki programowania są językami skryptowymi, które interpretują programy w taki sam sposób jak Python. Czasami instrukcje programu są konwertowane na instrukcje bardzo niskiego poziomu, które rozumie wyłącznie Twój komputer. Ten proces nazywa się kompilacją, a program wykonujący tę konwersję nazywa się kompilatorem. Skompilowane instrukcje można następnie załadować do komputera w celu ich uruchomienia. W wyniku zastosowania tej techniki powstają programy, które działają bardzo szybko, ponieważ w czasie wykonywania skompilowanych instrukcji niskiego poziomu komputer nie musi ustalać, co oznaczają instrukcje. Musi je tylko wykonywać.

Można by pomyśleć, że w związku z tym Python jest „powolnym" językiem programowania, ponieważ za każdym razem gdy działa program w Pythonie, „aktor-robot" zanim wykona jakieś polecenie, musi ustalić jego znaczenie. To jednak nie stanowi problemu, ponieważ współczesne komputery pracują bardzo szybko, a Python używa pewnych sprytnych sztuczek w celu kompilacji programu podczas jego uruchamiania.

Dane i informacje

Teraz, gdy wiemy, że komputery są maszynami przetwarzającymi dane, i rozumiemy, że programy mówią komputerom, co zrobić z danymi, zagłębimy się nieco w naturę danych i informacji. Ludzie używają słów „dane" i „informacje" zamiennie. Ważne jest jednak rozróżnienie pomiędzy nimi, ponieważ sposób, w jaki komputery i ludzie interpretują dane, jest zupełnie inny. Spójrzmy na rysunek 2.7, na którym pokazano różnicę.

Rysunek 2.7. Dane i informacje

Dwie części rysunku 2.7 zawierają te same dane, z tym że obraz po lewej stronie bardziej przypomina to, w jaki sposób są przechowywane dokumenty w komputerze. Do reprezentowania każdej litery i spacji w tekście komputer używa wartości liczbowej. Jeśli przeanalizujemy te liczby, możemy odczytać znaczenie każdej z nich — zaczynając od liczby 87, która reprezentuje wielką literę W (początek słowa *When*, od którego zaczyna się pierwszy akapit w dokumencie po prawej).

Ze względu na sposób przechowywania danych przez komputery w mapowaniu liczb na litery bierze udział jeszcze jedna warstwa. Każda liczba jest pamiętana przez komputer jako unikatowy wzorzec sygnałów włączenia i wyłączenia (jedynek i zer). W informatyce każda **1** lub **0** to tzw. **bit** (doskonały opis działania komputerów na tym poziomie oraz wyjaśnienie, jak działanie to tworzy podstawę dla kodowania, można znaleźć w książce Charlesa Petzolda *Code: The Hidden Language of Computer Hardware and Software*). Wartość 87, o której wiemy, że oznacza wielką literę W, ma następującą postać:

```
1010111
```

Jest to *binarna* reprezentacja wartości. Dokładny opis konwersji wartości binarnej na liczbę dziesiętną wykracza poza ramy tej książki (poza tym zrobił to Charles Petzold!), ale powyższy wzorzec można rozwikłać następująco: „87 to 1 plus 2 plus 4 plus 16 plus 64".

Każdy z bitów wzorca informuje sprzęt komputerowy o tym, czy w wartości występuje określona potęga liczby 2. Nie przejmuj się zbytnio, jeśli nie do końca to rozumiesz. Zapamiętaj jednak, że dane dla komputerów są zbiorem przechowywanych i przetwarzanych jedynek i zer. Tym właśnie są dane.

Z kolei *informacje* to interpretacja danych przez ludzi, w wyniku czego nabierają one znaczenia. Ściśle rzecz biorąc, komputery przetwarzają dane, a ludzie wykorzystują informacje.

Na przykład komputer może przechowywać w pamięci następujący wzorzec bitowy:

```
11111111 11111111 11111111 00000000
```

Można to zinterpretować jako: „Masz 256 zł debetu w banku" lub „Jesteś 256 m pod powierzchnią ziemi", lub „Osiem z trzydziestu dwóch wyłączników światła jest wyłączonych". Konwersja danych na informacje zwykle odbywa się w chwili, gdy człowiek czyta wyjście programu.

Podkreślam to po to, aby czytelnik zapamiętał, że komputer nie „wie", co znaczą dane, które przetwarza. Dla komputerów dane są tylko wzorcami bitów, to użytkownik nadaje im sens. Pamiętaj o tym, gdy otrzymasz wyciąg z konta, który mówi, że masz 8 388 608 PLN na swoim koncie, kiedy naprawdę masz tylko 83 PLN.

Przetwarzanie danych w Pythonie

Teraz wiemy, że Python jest procesorem danych. Skrypt napisany w Pythonie jest interpretowany przez system Pythona, który następnie generuje pewne dane wyjściowe. Wiemy również, że wewnątrz komputera, na którym działa program w Pythonie, wartości danych są reprezentowane przez wzorce bitów (jedynki i zera).

ZRÓB TO SAM

Praca z tekstem w Pythonie

Spróbujmy powiedzieć „cześć" Pythonowi w taki sposób, żeby to zrozumiał. Wróć do powłoki Pythona w środowisku IDLE i wpisz słowo `'witaj'`. Tym razem jednak ujmij to słowo w apostrofy:

```
>>> 'witaj'
'witaj'
```

Teraz nie wyświetli się żaden komunikat o błędzie. Python powtórzy tekst, który wprowadziłeś. Jeśli porównasz to z zachowaniem, które obserwowaliśmy po wprowadzeniu liczby, zauważysz, że w rzeczywistości Python zrobił to samo. Poprzednio wprowadziliśmy wartość 2, a Python powtórzył 2. Gdy wprowadziliśmy wartość tekstową, Python powtórzył ten tekst. Następną rzeczą, którą próbowaliśmy zrobić z liczbami, było ich dodawanie. Spróbujmy to zrobić dla ciągów tekstowych:

```
>>> 'witaj' + ' świecie'
'witaj świecie'
```

Fajnie. Python zachowuje się dokładnie tak, jak tego oczekujemy. Wiemy, że gdy przekażemy Pythonowi sumę do obliczenia, to obliczy jej wynik, a następnie go nam zwróci. Skorzystaliśmy z tego, aby dodać 2 do 2. Teraz odkryliśmy, że możemy użyć tej samej procedury, aby dodać słowo „witaj" do słowa „ świecie". Zwróćmy jednak uwagę, że sprytnie umieściliśmy spację przed słowem „świecie". W przeciwnym razie program wyświetliłby „witajświecie".

Wewnątrz Pythona także dzieje się coś sprytnego, ponieważ sposób działania operacji + (dodawanie) jest poprawny zarówno dla liczb, jak i ciągów znaków. Jeśli poprosimy Pythona o dodanie dwóch liczb, zwróci nam sumę. Jeśli poprosimy go o dodanie dwóch ciągów znaków, zwróci jeden ciąg dołączony na końcu drugiego.

ANALIZA KODU

Łamanie reguł Pythona

Pytanie: Jak sądzisz, co by się stało, gdybyś pominął końcowy apostrof w napisanym ciągu znaków?

Odpowiedź: Z naszych eksperymentów z nawiasami można by się spodziewać, że Python będzie cierpliwie czekał na następny wiersz, gdzie będziesz mógł wpisać resztę ciągu znaków. Niestety, tak się nie stanie.

```
>>> 'witaj
SyntaxError: EOL while scanning string literal
```

Literał tekstowy (ang. *string literal*) to ciąg tekstu, który „literalnie" znajduje się w tekście. Litery EOL są skrótem od *End Of Line*, co oznacza „koniec wiersza". Powłoka Pythona mówi, że nie podoba jej się, jeśli ciągi znaków nie mają zakończenia. Python traktuje poszczególne operandy (liczby i ciągi znaków) jako konstrukcje, które nie mogą obejmować wielu wierszy. Nie ma nic złego w tworzeniu wyrażenia tekstowego, które obejmuje kilka wierszy (możesz tego spróbować), ale Python nie pozwala, aby operandy tego wyrażenia obejmowały wiele wierszy.

Pytanie: Jak sądzisz, co by się stało, gdybyś spróbował odjąć jeden ciąg od drugiego?

Odpowiedź: Python jest wystarczająco sprytny, żeby wiedzieć, że chociaż rozsądnie jest używać operacji – (odejmowanie), która oznacza „odejmowanie jednej liczby całkowitej od innej" (możesz tego spróbować, jeśli chcesz), to nie jest rozsądne, aby program próbował odejmować jeden ciąg znaków od innego.

```
>>> 'witaj' - ' świecie'
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    'witaj' - ' świecie'
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

Python podaje nam szczegóły tego, co poszło nie tak, ale zamiast powiedzieć: „Odejmowanie od siebie ciągów znaków jest głupie", wyświetla opis, który jest znacznie trudniejszy do zrozumienia. Aby to zrozumieć, musisz wiedzieć, że słowo „operand" oznacza „coś, co jest przetwarzane przez operator". W tym przypadku operatorem jest – (minus), a operandy są dwoma łańcuchami (*witaj* i *świecie*). Python mówi nam, że nie można umieścić operatora minus pomiędzy dwoma ciągami znaków.

Pytanie: Jak sądzisz, co by się stało, gdybyś spróbował dodać liczbę do ciągu znaków?

Odpowiedź: Dodanie liczby do ciągu znaków jest tak samo głupie jak odjęcie jednego ciągu znaków od innego, więc możesz oczekiwać, że Python wyświetli komunikat o błędzie. Ale możesz również oczekiwać, że ten komunikat będzie trudny do zrozumienia:

```
>>> 'witaj' + 2
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    'witaj' + 2
TypeError: must be str, not int
```

Mam nadzieję, że tym razem komunikat o błędzie jest nieco bardziej jasny. Python mówi, że coś „musi być ciągiem znaków, a nie liczbą całkowitą" (chociaż nie jest zbyt pomocny, ponieważ nie mówi Ci, który z operandów jest nieprawidłowy).

Gdybyśmy naprawdę chcieli umieścić cyfrę 2 na końcu słowa „witaj", moglibyśmy to zrobić, ujmując cyfrę w apostrofy:

```
>>> 'witaj' + '2'
'witaj2'
>>>
```

Pytanie: Jak sądzisz, co by się stało, gdybyś spróbował pomnożyć ciąg znaków przez liczbę?

Odpowiedź: To działa. Ciąg znaków zostanie powtórzony podaną liczbę razy:

```
>>> 'witaj' * 3
'witajwitajwitaj'
```

Python, kiedy tylko będzie mógł, spróbuje zrobić coś sensownego. To wyrażenie nadal będzie działać, jeśli kolejność operandów zostanie odwrócona. Python spróbuje również zrobić coś sensownego, jeśli spróbujesz pomnożyć ciąg znaków przez zero lub liczbę ujemną.

Tekst i liczby jako typy danych

Jeśli przyjrzysz się uważnie wynikom powyższych instrukcji, zauważysz, że kiedy Python ocenia wartość wyrażenia numerycznego (które w wyniku daje liczbę), zwraca tylko cyfry, ale jeśli ocenia wartość ciągu znaków, zwraca tekst zamknięty w apostrofach. Dzieje się tak dlatego, ponieważ Python przestrzega zasad dotyczących sposobu prezentowania różnych rodzajów danych.

Python wymusza ścisłą separację danych liczbowych (wartość 2) i tekstowych (ciąg `witaj`). Sposób przechowywania wartości i efekt wykonywania na nich operacji jest inny dla każdego typu danych nawet wtedy, kiedy operacja zapewnia ten sam operator dla każdego typu. Możemy użyć operatora + do dodawania zarówno liczb, jak i ciągów tekstowych, a Python zadba o realizację właściwej operacji dzięki ustaleniu kontekstu działania. Jeśli zauważy operator + pomiędzy dwiema liczbami, użyje liczbowej wersji operatora +. Jeśli zauważy operator + pomiędzy dwoma ciągami znaków, użyje tekstowej wersji operatora +.

Ludzie postępują tak samo. Mówimy o myciu twarzy, myciu naczyń lub myciu samochodu i chociaż działanie jest zasadniczo takie samo (coś myjemy), to faktycznie wykonywana czynność jest w każdym przypadku inna. Można sobie wyobrazić, że w języku mogłyby istnieć inne słowa oznaczające mycie, z których jedno oznaczałoby „mycie samochodu", ale w języku mówionym nie zawsze tak jest. Jeśli po prostu używamy słowa „myć", to czytelnik musi użyć możliwości mózgu i własnego doświadczenia, aby ustalić, co się dzieje. Oczywiście praca mózgu i doświadczenie nie są czymś, co komputer posiada w nadmiarze, więc kiedy piszemy program, musimy być konsekwentni co do sposobu wyrażania tego, czego chcemy. Takie postępowanie wymusza projekt Pythona (a także innych języków programowania).

Praca z funkcjami Pythona

Teraz, gdy wiemy coś o tym, jak Python przetwarza tekst i liczby, możemy zacząć się zastanawiać, w jaki sposób elementy tekstowe są reprezentowane za pomocą liczb, a mówiąc dokładniej: wzorców bitów. Aby to zrobić, skorzystamy z samego Pythona w celu zbadania, w jaki sposób przechowuje on wartości. Użyjemy do tego celu pewnych funkcji wbudowanych w Pythona.

Funkcja to zachowanie identyfikowane przez odrębną nazwę. Gdybyś pisał scenariusz do zagrania przez aktora, mógłbyś uwzględnić wskazówki dotyczące jego zachowania na scenie, np.: „przejdź w lewo", „wyjrzyj przez okno" lub — mój faworyt — „biegnij tak, jakby ścigał cię niedźwiedź". Te wskazówki inicjują działania, które aktor wykona podczas gry na scenie. Można je potraktować jako „funkcje", które aktor potrafi zrealizować. Python przypomina aktora. Wie, jak wykonać zbiór wbudowanych funkcji. Niektóre z tych funkcji wykorzystamy do zbadania sposobu, w jaki w komputerze jest reprezentowany tekst.

KĄCIK PROGRAMISTY

Funkcje są ważną częścią każdego języka programowania

Dużą część nauki języka programowania stanowi poznawanie funkcji, które ten język dostarcza. W następnych kilku rozdziałach zapoznamy się z kilkoma funkcjami, a następnie zaczniemy pisać także własne funkcje.

Każda funkcja w Pythonie ma odrębną nazwę i można do niej przekazać dane do przetwarzania. Pod tym względem funkcję można porównać do niewielkiego procesora danych. Funkcja pobiera dane wejściowe i generuje wynik.

Funkcja ord

Jedno z działań, które Python potrafi wykonać, nosi nazwę `ord`. Nazwa jest skrótem od *ordinal value*, czyli „wartość porządkowa". Jeśli poszukasz znaczenia słowa „porządkowy", znajdziesz bardzo mylący opis (przynajmniej ten, który ja znalazłem, był mylący). W tym kontekście oznacza to „podaj mi wartość, która reprezentuje ten znak w sekwencji możliwych kodów znaków". Lub, w krótszej formie, „podaj mi liczbę reprezentującą ten znak".

Funkcję wywołujemy przez podanie jej nazwy, a następnie, w nawiasach, danych, na których funkcja działa. Anatomię wywołania funkcji `ord` pokazano na rysunku 2.8. Nazwa programistyczna dla „danych, na których funkcja działa" to argumenty.

Rysunek 2.8. Anatomia wywołania funkcji

ZRÓB TO SAM

Badanie reprezentacji tekstowej za pomocą funkcji `ord`

Spróbujmy wykorzystać funkcję `ord`, aby zbadać, w jaki sposób wewnątrz komputera jest przechowywany tekst. Możemy zacząć od znalezienia liczby używanej do reprezentowania określonego znaku. Możemy przekazać do funkcji `ord` ciąg znaków zawierający pojedynczą literę `W` i zobaczyć, jaką zwróci nam liczbę. Wprowadź w powłoce Pythona w środowisku IDLE poniższe polecenie i sprawdź zwrócony wynik.

```
>>> ord('W')
87
```

Dokładnie taką wartość widzieliśmy powyżej na rysunku 2.8. Litera W w pierwszym słowie Deklaracji Niepodległości była reprezentowana przez liczbę 87.

Podczas wpisywania wyrażeń w Pythonie trzeba jednak zachować ostrożność. Litera W, która nas interesuje, musi być częścią ciągu znaków, dlatego trzeba ją ująć w apostrofy. Jeśli pominiemy apostrofy, system Python pomyśli, że pytamy o coś, co ma nazwę `W`, i odpowie nam, że nic o tym nie wie.

```
>>> ord(W)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    ord(W)
NameError: name 'W' is not defined
```

W programach w języku Python nie będziemy zbyt często korzystać z funkcji `ord`. Funkcja ta dostarcza jednak bardzo przydatnego okna, przez które można spojrzeć, by zobaczyć, w jaki sposób są przetwarzane wartości w kodzie Pythona.

Funkcja chr

Funkcję `ord` uzupełnia funkcja Pythona o nazwie `chr`. Ta funkcja pobiera liczbowy kod znaku i zwraca znak reprezentowany przez ten kod.

ZRÓB TO SAM

Konwersja liczb na tekst za pomocą funkcji chr

Nie ma absolutnie żadnych nagród za odgadnięcie, co zwróci funkcja `chr`, jeśli przekażemy do niej kod 87, ale trzeba tego spróbować.

```
>>> chr(87)
'W'
```

Liczby reprezentujące znaki są ułożone w rozsądny sposób, ponieważ jeśli spróbujemy wyświetlić znak odpowiadający kodowi 88, to otrzymamy dokładnie to, czego można oczekiwać:

```
>>> chr(88)
'X'
```

W świecie komputerów konkretne kody konkretnych znaków są mapowane zgodnie z międzynarodowymi standardami. Piszę programy komputerowe od bardzo dawna i dzięki temu doświadczeniu wiem, że wielką literę A reprezentuje kod 65, a spację (która jest bardzo ważna, ponieważ pozwala na wstawianie odstępów pomiędzy słowami) reprezentuje kod 32. Zwykle jednak nie ma potrzeby uczenia się tych liczb na pamięć, ponieważ o wyświetlanie tekstu dba Python i system operacyjny, w którym Python działa (odkryłem również, że znajomość tych liczb nie pomaga mi imponować innym na przyjęciach).

Analiza sposobu przechowywania danych za pomocą funkcji `bin`

Pamięć komputera można traktować jako ogromną przestrzeń małych pudełek, z których każde ma unikatowy liczbowy adres. Każda lokalizacja w pamięci zawiera 8 pojedynczych bitów, które mogą być włączone lub wyłączone (tak jak widzieliśmy wcześniej). Taką lokalizację w pamięci nazywa się bajtem. Kiedy chwalicie się komputerem, który ma „16 gigabajtów pamięci", naprawdę mówicie, że komputer zawiera 16 mld pojedynczych lokalizacji, z których każda ma rozmiar bajta.

W pojedynczym bajcie nie da się przechowywać dużego zakresu wartości, zatem można ze sobą pogrupować kilka bajtów, by przechowywać w nich większe liczby. W dalszej części tej książki zobaczymy, jak to działa, i opowiemy o rodzajach wartości, które może przechowywać Python. Na razie przyjrzymy się jedynie wzorcom bitów używanych do przechowywania tych wartości.

Wbudowana w Pythona funkcja `bin` pobiera liczbę i zwraca ciąg bitów reprezentujących wartość tej liczby.

ZRÓB TO SAM

Odkryj reprezentację binarną

Funkcję `bin` możemy wykorzystać do sprawdzenia sposobu przechowywania danych w komputerze.

```
>>> bin(87)
'0b1010111'
```

Funkcja `bin` zwraca ciąg znaków odpowiadający binarnej reprezentacji tej liczby. Zwróćmy uwagę, że jest to ciąg złożony z zer i jedynek. Ciąg jest poprzedzony prefiksem `0b`, który informuje użytkownika, że to jest binarna reprezentacja liczby.

ANALIZA KODU

Budowanie liczb binarnych

Funkcja `bin` nie jest zbyt często stosowana w programach w Pythonie (chyba że klient poprosi Cię o napisanie programu wyświetlającego wartości binarne), ale można jej użyć do zbadania sposobu zapisywania liczb przechowywanych w komputerze. Warto zapamiętać, że wewnątrz sprzętu komputerowego wszystkie dane są przetwarzane jako wysokie lub niskie napięcia reprezentujące określony sygnał. Możemy zinterpretować je jako 0 (brak napięcia) i 1 (jakieś napięcie).

Pytanie: Jak wygląda binarna wartość `0`?

Odpowiedź: Można to ustalić za pomocą funkcji `bin`.

```
>>> bin(0)
'0b0'
```

Wartość `0` w formacie binarnym wygląda tak jak wszystkie inne zera.

Pytanie: Jak wygląda binarna wartość `1`?

Odpowiedź: Możemy to zobaczyć, przekazując `1` do funkcji `bin`.

```
>>> bin(1)
'0b1'
```

Wartość `1` w formacie binarnym wygląda dokładnie tak jak każda inna jedynka, z którą spotkałeś się wcześniej. Dotychczas nie wydaje się, aby w wartościach binarnych było coś szczególnego.

Pytanie: Jak wygląda binarna wartość 2?

Odpowiedź: Można to ustalić, przekazując wartość 2 do funkcji `bin`.

```
>>> bin(2)
'0b10'
```

Tym razem wynik jest inny. Aby zrozumieć, co to znaczy, zacznijmy od przeanalizowania dziesiętnej liczby `10`. W tej liczbie cyfra `1` mówi nam, ile dziesiątek mieści się w liczbie. Format binarny działa w ten sam sposób, z tym że cyfra 1 mówi nam, ile w liczbie mieści się dwójek. Zatem wartość `10` w systemie binarnym (dwójkowym) oznacza 2.

Pytanie: Jak myślisz, co oznacza wartość binarna 11?

Odpowiedź: Binarna wartość 11 odpowiada dziesiętnej wartości 3 (1 plus 2). Aby to sprawdzić, możemy użyć funkcji `bin`.

```
>>> bin(3)
'0b11'
```

Trzeci bit oznacza liczbę czwórek, jakie mieszczą się w liczbie, czwarty bit oznacza liczbę ósemek itd. Aby zobaczyć, jakie wzorce bitów reprezentują poszczególne liczby, możesz poeksperymentować z funkcją `bin`, a następnie sprawdzić wyniki.

Pytanie: W jaki sposób binarna wartość reprezentująca liczbę 86 różni się od binarnej wartości reprezentującej liczbę 87?

Odpowiedź: Można to ustalić, ponownie korzystając z funkcji `bin`.

```
>>> bin(86)
'0b1010110'
>>> bin(87)
'0b1010111'
```

Jeśli porównasz te dwa wzorce binarne, zauważysz, że skrajny prawy bit zmienił wartość z jeden na zero, ponieważ ten bit wskazuje, czy ta wartość binarna zawiera jedności. Każda liczba zawierająca jedynkę na skrajnym prawym bicie jest liczbą nieparzystą. Jeśli mi nie wierzysz, poeksperymentuj z kilkoma wartościami.

Czego się nauczyłeś?

W tym rozdziale dowiedziałeś się, jak faktycznie działają komputery i co to jest programowanie. Odkryłeś, że komputer postrzega cały wszechświat jako wzorce jedynek i zer reprezentujących dane, które komputer przetwarza. Komputer przeprowadza przetwarzanie danych dzięki przekształceniu jednego wzorca bitów (reprezentującego dane wejściowe) na inny wzorzec bitów (reprezentujący dane wynikowe).

Kiedy ludzie zaczynają przeglądać dane wyjściowe i podejmują działania na ich podstawie, to dane stają się *informacjami*. Komputery nie są świadome znaczenia przetwarzanych przez siebie wzorców bitów, co oznacza, że mogą robić z danymi „głupie” rzeczy.

Program informuje komputer, co on ma zrobić ze wzorcami bitów. Sam komputer rozumie tylko bardzo proste instrukcje, które muszą być zapisane w specjalnych językach, zwanych

językami programowania. Python jest językiem programowania i działa jako program komputerowy w tym sensie, że pobiera instrukcje programu, a następnie przetwarza je w taki sposób, w jaki aktor wykonuje rolę.

Zadaniem programisty jest stworzenie programu jako sekwencji instrukcji opisujących zadania do wykonania. Aby umożliwić skuteczne rozwiązanie, programista musi nie tylko napisać dobry program, ale także zadbać o to, aby ten program faktycznie robił to, czego chce użytkownik. Oznacza to, że zanim programista napisze jakikolwiek kod, musi dobrze zrozumieć, jakie wymagania powinien spełnić ten kod. Rozmowa z ludźmi w celu ustalenia ich wymagań jest bardzo cenną umiejętnością. Niezbędną, jeśli chcesz stać się odnoszącym sukcesy programistą.

Aby lepiej zrozumieć tę treść, spróbuj przeanalizować poniższe pytania dotyczące komputerów, programów i programowania.

Czy komputer „wie", że to głupie, aby ktoś miał wiek -20?

Nie. Jeśli chodzi o komputer, wartość wieku jest po prostu wzorcem bitów reprezentującym liczbę. Jeśli chcemy, aby komputer odrzucał ujemne reprezentacje wieku osób, powinniśmy wbudować to zrozumienie do programu.

Jeśli wyjście z programu służy do ustawiania układu wtrysku paliwa w samochodzie, to czy są to dane wyjściowe, czy informacje?

Gdy tylko coś zaczyna przetwarzać dane, stają się one informacjami. Człowiek nie robi niczego z tymi wartościami, ale powodują one zmianę obrotów silnika, co może mieć wpływ na ludzi. Sądzę więc, że to sprawia, że są to informacje wyjściowe, a nie dane.

Czy komputer jest „głupi", ponieważ nie rozumie polskiego lub angielskiego?

Trudno napisać coś po polsku lub angielsku w taki sposób, aby było to całkowicie jednoznaczne. Duża część zawodu prawnika bazuje na precyzyjnej interpretacji znaczenia tekstów i sposobu zastosowania tego znaczenia w określonych sytuacjach. Ponieważ my, ludzie, nie możemy dojść do zgody, jak coś należy rozumieć, nie jest uczciwe nazywanie komputera głupim tylko dlatego, że także nie potrafi tego zrobić.

Jeśli nie wiem, jak rozwiązać problem, to czy mogę napisać program, który to zrobi?

Nie. Możesz napisać kilka instrukcji i sprawdzić, co się stanie, gdy spróbujesz je uruchomić, ale jest mało prawdopodobne, abyś osiągnął taki wynik, jakiego oczekujesz. Byłoby to tak, jakbyś rzucił pod ścianę pewną liczbę kół, zębatek i silnik i oczekiwał, że po „wylądowaniu" utworzą działający samochód. W rzeczywistości najlepszym sposobem napisania programu jest oderwanie się od klawiatury i zastanowienie się nad tym, co program powinien robić.

Czy rozsądne jest założenie, że klient mierzy wszystko w centymetrach?

Nigdy nie jest rozsądne zakładanie czegokolwiek na temat projektu. Skuteczny programista musi dbać o to, aby wszystko to, co robi, bazowało na solidnym zrozumieniu tematu. Każde przyjęte założenie zwiększa prawdopodobieństwo niepowodzenia.

Jeśli program robi coś źle, to czy jest to moja wina, czy wina klienta?

To zależy:

- Specyfikacja prawidłowa, program nieprawidłowy — wina programisty.
- Niepoprawna specyfikacja, program poprawny — wina klienta.
- Niepoprawna specyfikacja, błędny program — wina obu stron.

Skorowidz

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

Ż

Wszechstronny, wydajny, elastyczny. Python — wybór idealny!

Python jest znakomitym językiem do tworzenia wydajnego kodu. Nadaje się do różnych, również bardzo nietypowych zastosowań. Samo programowanie natomiast, choć jest w nim coś magicznego, stanowi umiejętność, którą każdy może opanować pod warunkiem odrobiny zaangażowania. Naturalnie, nauka kodowania nie zawsze przebiega bezproblemowo. Trzeba zrozumieć sposób działania komputera i nauczyć się nieco inaczej myśleć o rozwiązywaniu problemów. Napisanie dobrze działającego programu jest jednak niezwykle satysfakcjonującym doświadczeniem, a programowanie samo w sobie jest jedną z najbardziej kreatywnych umiejętności.

Ta książka jest przeznaczona dla osób, które nie mają doświadczenia w programowaniu. Została pomyślana jako podręcznik, który maksymalnie ułatwia uczenie się skutecznego kodowania. Najpierw omówiono niskopoziomowe instrukcje programowania, aby stopniowo przejść do przedstawienia i analizy profesjonalnych konstrukcji programistycznych. Książka jest przyjazna w odbiorze, a przy tym pełna innowacji, takich jak choćby opis korzystania z wbudowanych gadżetów czy projekty „zrób to sam". Dzięki temu Czytelnik bardzo szybko odkryje, że programowanie jest świetną, ekscytującą i porywającą zabawą!

W tej książce:

- wprowadzenie do Pythona i przygotowanie środowiska do pracy
- podstawowe konstrukcje kodu
- projektowanie konstrukcji bardziej złożonych aplikacji
- tworzenie dokumentacji aplikacji w Pythonie
- korzystanie z różnych bibliotek Pythona

Rob Miles przez ponad 30 lat wykładał programowanie na Uniwersytecie Hull w Wielkiej Brytanii. Został uhonorowany tytułem Microsoft MVP. Gdyby miał choć trochę wolnego czasu, spędziłby go na kodowaniu. Uwielbia tworzyć programy, uruchamiać je i śledzić ich działanie. Uważa, że programowanie to budowanie przyszłości. Jego pasją są kodowanie (co chyba naturalne), tworzenie nowych rzeczy oraz świetne dowcipy.

ISBN 978-83-283-4654-3

Cena: 89,00 zł

Microsoft Press