ADDISON WESLEY

SERIA DANE I ANALIZY

Wprowadzenie do uczenia maszynowego opartego na chmurze

AI

Podejście pragmatyczne

NOAH GIFT

AI

Podejście pragmatyczne

Wprowadzenie do uczenia maszynowego opartego na chmurze

Noah Gift

Przekład
Marek Włodarz

APN Promise
Warszawa 2019

AI. Podejście pragmatyczne. Wprowadzenie do uczenia maszynowego opartego na chmurze

Authorized Polish translation of the English language edition entitled
Pragmatic AI. An Introduction to Cloud-Based Machine Learning, by Noah Gift,
ISBN: 978-0-13-486386-3

Polish language edition published by APN PROMISE SA Copyright © 2019

Autoryzowany przekład z wydania w języku angielskim, zatytułowanego:
Pragmatic AI. An Introduction to Cloud-Based Machine Learning, by Noah Gift,
ISBN: 978-0-13-486386-3

APN PROMISE SA, ul. Domaniewska 44a, 02-672 Warszawa
tel. +48 22 35 51 600, fax +48 22 35 51 699
e-mail: mspress@promise.pl

ISBN: 978-83-7541-363-2

Przekład: Marek Włodarz
Korekta: Ewa Swędrowska
Skład i łamanie: MAWart Marek Włodarz

Książkę tę dedykuję mojej rodzinie, która zawsze mnie wspierała:
mojej żonie Leah Gift; mojej mamie Shari Gift;
memu synowi Liamowi Gift;
oraz mojemu mentorowi, dr. Josephowi Bogen

Spis treści

Wstęp

Około dwudziestu lat temu pracowałem w Caltech w Pasadenie i marzyłem o tym, by pewnego dnia móc pracować ze sztuczną inteligencją, po prostu codziennie. W tym czasie, na początku XXI wieku, interesowanie się AI nie było zbyt popularne. A jednak jesteśmy tu, gdzie jesteśmy, zaś ta książka stanowi kulminację mojej obsesji na temat sztucznej inteligencji i science fiction. Miałem wówczas szczęście, aby pracować ramię w ramię z kilkoma czołowymi specjalistami od AI i bez wątpienia to doświadczenie wprowadziło mnie na drogę do napisania tej książki.

Jednak poza samą AI zawsze miałem również obsesję na temat automatyzacji i pragmatyzmu. Ta książka uwzględnia oba te motywy. Doświadczenie menedżera, zmuszonego do jednoczesnego tworzenia nowych produktów i unikania straszliwych raf nie dość doskonałej technologii, wyrobiło we mnie praktyczne podejście. Jeśli coś nie zostało wdrożone w produkcji, w ogóle się nie liczy. Jeśli nie zostało zautomatyzowane, jest wadliwe. Mam nadzieję, że książka ta zainspiruje czytelników do przyjęcia takiego punktu widzenia.

Kto powinien przeczytać tę książkę

Książka ta jest przeznaczona dla każdego interesującego się sztuczną inteligencją, uczeniem maszynowym, chmurą i dowolnymi kombinacjami tych zagadnień. Zarówno programiści, jak i nie-programiści powinni być w stanie znaleźć w niej użyteczne okruchy wiedzy. Wielu studentów, z którymi miałem do czynienia na warsztatach prowadzonych przeze mnie w NASA, PayPal czy na Uniwersytecie Kalifornijskim w Davis było w stanie przyswoić sobie te notatniki i koncepcje pomimo niewielkiego lub żadnego wcześniejszego doświadczenia w programowaniu. W książce wykorzystywany jest przede wszystkim Python i jest to jeden z najlepszych języków, które warto wybrać, gdy ktoś dopiero zaczyna naukę programowania.

Jednocześnie przedstawiam tu wiele zaawansowanych tematów, takich jak korzystanie z chmurowych platform obliczeniowych (a konkretnie AWS, GCP i Azure) oraz realizowaniu programowania ML i AI. Dla zaawansowanych programistów, którzy swobodnie

posługują się Pythonem, chmurą i uczeniem maszynowym, znajdzie się tu wiele użytecznych pomysłów, które można natychmiast przeszczepić do bieżącej pracy.

Organizacja tej książki

Książka jest podzielona na trzy części: Część I: „Wprowadzenie do pragmatycznej AI", Część II: „AI w chmurze" oraz Część III: „Tworzenie praktycznych aplikacji AI". Rozdziały 1–3 tworzące część pierwszą zawierają materiały wprowadzające:

- Rozdział 1, „Pragmatyczne podejście do AI" zawiera przegląd celów tej książki oraz błyskawiczny tutorial języka Python, dający dostateczne podstawy, aby użytkownik mógł zrozumieć inne przykłady użycia Pythona w tej ksiażce.
- Rozdział 2, „Narzędzia AI i ML" omawia cykl życiowy systemów kompilacyjnych, wiersz polecenia oraz zastosowanie notatników Jupyter w projekcie *data science*.
- Rozdział 3, „Spartański cykl życia AI" opisuje pragmatyczne włączanie pętli zwrotnych do projektów produkcyjnych. Omówione zostaną w nim narzędzia i schematy, takie jak Docker, AWS SageMaker oraz TensorFlow Processing Units (TPU).

Część II zawiera rozdziały 4 i 5 poświęcone środowiskom AWS i Google Cloud.

- Rozdział 4, „Chmurowe projektowanie AI w Google Cloud Platform" zajmuje się platformą GCP i niektórymi z unikatowych, przyjaznych dla dewelopera funkcjonalności przez nią oferowanych. Omówione zostaną takie usługi, jak TPU, Colaboratory i Datalab.
- Rozdział 5, „Projektowanie chmurowej AI przy użyciu Amazon Web Services", zagłębia się w takie zagadnienia pracy w AWS, jak instancje Spot, CodePipeline, wykorzystanie i testowanie Boto oraz ogólny przegląd udostępnianych usług.

Część III zawierająca rozdziały 6–11 obejmuje praktyczne zastosowania AI i działające przykłady.

- Rozdział 6, „Przewidywanie wpływów mediów społecznościowych w NBA" opiera się na pracy w hipotetycznym startupie, ale wykorzystałem w nim również kilka artykułów i wykład na konferencji Strata. Przedstawione zagadnienia obejmują: Co określa finansową wartość drużyn? Czy zwyciężanie przyciąga więcej fanów? Czy płace są skorelowane z aktywnością w mediach społecznościowych?
- Rozdział 7, „Tworzenie inteligentnego slackbota w AWS" omawia tworzenie bezserwerowego bota, który będzie przeczesywać wybrane witryny Web i przekazywać podsumowane informacje z powrotem do statku-matki w platformie Slack.
- Rozdział 8, „Wyszukiwanie pomysłów na zarządzanie projektami w organizacji GitHub" prezentuje przegląd typowego źródła informacji o zachowaniu

programistów – metadanych GitHuba. Do wydobycia drogocennych danych behawioralnych użyte zostaną takie narzędzia, jak Pandas, notatniki Jupyter oraz narzędzie wiersza polecenia *click*.

- Rozdział 9, „Dynamiczna optymalizacja instancji EC2 w AWS" przekształca sygnały zwracane przez AWS w możliwość wykorzystania technik uczenia maszynowego do optymalizowania cen.
- Rozdział 10, „Nieruchomości" bada ogólnokrajowe i lokalne trendy cen domów przy użyciu uczenia maszynowego i interaktywnych wykresów.
- Rozdział 11, „Produkcyjna AI dla treści generowanych przez użytkowników" omawia wykorzystanie AI do interakcji z treściami generowanymi przez użytkowników. Przedstawione zostały takie zagadnienia, jak analiza opinii i silniki rekomendacji.
- Dodatek A, „Akceleratory AI", stanowi krótkie omówienie rozwiązań sprzętowych specjalnie projektowanych do wykonywania obciążeń AI. Przykładem takiego akceleratora jest chip TPU opracowany przez Google.
- Dodatek B, „Wybór liczby grup", wyjaśnia, że zagadnienie to jest kwestią raczej sztuki, niż wiedzy, choć istnieją pewne techniki, które sprawiają, że proces decyzyjny staje się bardziej przejrzysty.

Kod przykładowy

Każdemu rozdziałowi książki odpowiada towarzyszący mu notatnik Jupyter lub zbiór notatników. Zostały one opracowane w ciągu kilku ostatnich lat przy okazji tworzenia artykułów lub prowadzonych przeze mnie warsztatów i wykładów.

> **Uwaga**
>
> Cały kod źródłowy odpowiadający książce można znaleźć (w postaci notatników Jupyter) w repozytorium GitHub:
>
> *https://github.com/noahgift/pragmaticai.*

Wiele przykładów zawiera również pliki Makefiles, takie jak poniższy:

```
setup:
        python3 -m venv ~/.pragai

install:
        pip install -r requirements.txt

test:
        cd chapter7; py.test --nbval-lax notebooks/*.ipynb
```

```
lint:
        pylint --disable=W,R,C *.py

lint-warnings:
        pylint --disable=R,C *.py
```

Makefile jest doskonałym sposobem wstępnego przygotowania różnych aspektów projektów *data science* realizowanych w Pythonie lub R. W szczególności są użyteczne przy konfigurowaniu środowiska, podłączaniu kodu źródłowego, wykonywaniu testów i wdrażaniu. Dodatkowo stosowanie izolowanych środowisk, takich jak `virtualenv`, pozwala wyeliminować całą klasę problemów. To zadziwiające, jak wielu spośród moich studentów boryka się z dokładnie tym samym problemem: zainstalowali coś w jednym interpreterze Pythona, ale następnie użyli innego. Albo też nie mogą doprowadzić do działania pewnego kodu, gdyż jakieś dwa pakiety są ze sobą w konflikcie.

W ogólności rozwiązaniem tych problemów jest użycie odrębnego środowiska wirtualnego dla każdego projektu i zadbanie o to, aby zawsze wybrać właściwe środowisko podczas pracy nad projektem. Odrobina planowania przy tworzeniu projektu zapewnia uniknięcie poważnych problemów w przyszłości. Połączenie Makefile, lintingu, testów w notatnikach Jupyter, systemów kompilacyjnych SaaS i testów jednostkowych to najlepsze zalecane praktyki, do których stosowania zachęcam zarówno swoich studentów, jak i czytelników.

Konwencje używane w tej książce

W książce tej stosowane są następujące konwencje typograficzne.

- `In [2]:` prezentuje wyjście terminala IPython. Często jest ono analogiczne do dołączonych przykładowych notatników Jupyter dostępnych na GitHubie.

Podziękowania

Jestem wdzięczny Laurze Lewin za możliwość opublikowania tej książki w wydawnictwie Pearson, jak również pozostałym członkom zespołu: Malobika Chakraborty i Sheri Replin. Chciałbym również podziękować recenzentom technicznym: Chao-Hsuan Shen (*https://www.linkedin.com/in/lucashsuan/*), Kennedy'emu Behrmanowi (*https://www.linkedin.com/in/kennedybehrman/*) oraz Guy'owi Ernestowi (*https://www.linkedin.com/in/guyernest/*) z Amazonu. Odegrali znaczącą rolę w nadaniu tej książce najlepszego kształtu.

Przede wszystkim podziękowania należą się Caltechowi. Moje zainteresowania w dziedzinie AI uformowały się, gdy pracowałem tam we wczesnych latach XXI wieku. Pamiętam, że jeden z profesorów przekonywał mnie, że AI jest marnowaniem czasu, ale pamiętam też, że chciałem robić to mimo wszystko. Postawiłem sobie wówczas cel, że będę tworzył znaczące programy AI, zanim osiągnę czterdziestkę i że osiągnę biegłość w wielu językach programowania – i tak się też chciało. Mówienie mi, że czegoś nie uda mi się zrobić, zawsze było dla mnie wielką motywacją, zatem dziękuję, panie profesorze!

Spotkałem tam również kilku naprawdę znaczących ludzi, w tym dr. Josepha Bogena, neurofizjologa, który był ekspertem w teorii świadomości. Przy obiedzie prowadziliśmy dyskusje o sieciach neuronowych, źródłach świadomości, rachunku różniczkowym lub pracy, którą prowadził w laboratorium Christofa Kocha. Powiedzieć, że miał on wpływ na moje życie, byłoby niedopowiedzeniem i jestem dumny, że obecnie pracuję nad sieciami neuronowymi między innymi z powodu tych rozmów, które prowadziliśmy 18 lat temu.

Byli też inni ludzie, których poznałem w Caltech, z którymi nie wiązały mnie aż tak silne związki, ale którzy też wywarli wielki wpływ na mnie i moją pracę: dr David Baltimore (dla którego pracowałem), dr David Goodstein (również), dr Fei-Fei Li oraz dr Titus Brown (którzy zarazili mnie entuzjazmem do Pythona).

Sport i treningi zawsze odgrywały wielką rolę w moim życiu. Był taki moment, gdy zupełnie poważnie myślałem o karierze zawodowego koszykarza, a nawet Ultimate Frisbee. Na Politechnice Kalifornijskiej spotkałem Sheldona Blockburgera, olimpijskiego dziesięcioboistę, który nauczył mnie, jak biegać krótkie i długie dystanse, aż osiągnąłem czas poniżej 27 sekund na 200 metrów. Nadal pamiętam, jak mówił, że „mniej niż 1 procent

populacji ma dość dyscypliny, aby zrealizować taki trening". To ćwiczenie w samodyscyplinie odegrało wielką rolę w moim rozwoju jako inżyniera oprogramowania. Oddzielne podziękowania należą się moim trenerom i kolegom od sztuk walki, a zwłaszcza brazylijskiego jiu-jitsu: Maui Jiu Jitsu, z Haiku, Maui.

Podziękować chciałbym również Zakowi Stone, menedżerowi produktu w dziedzinie TensorFlow, za wczesny dostęp do projektu TPU i pomoc w opanowaniu różnych elementów GCP. Bardzo pomocny był również Guy Ernest z AWS oraz Paul Shealy z Microsoftu, który doradzał mi w zakresie chmury Azure.

Na koniec podziękowania należą się moim przyjaciołom z czasów początku mojej kariery: Jerry Castro (*https://www.linkedin.com/in/jerry-castro-4bbb631/*) i Kennedy Behrman (*https://www.linkedin.com/in/kennedybehrman/*), którzy nadal są tak samo niezawodni, jak wówczas. Duża część materiału do tej książki powstała, gdy pracowaliśmy ramię w ramię w okopach startupu, który ostatecznie upadł. Takie doświadczenia ujawniają prawdziwe charaktery i jestem dumny z bycia ich współpracownikiem i przyjacielem.

O autorze

Noah Gift jest wykładowcą i konsultantem w UC Davis Graduate School of Management w zakresie programu MSBA. Wykłada zawansowane uczenie maszynowe i prowadzi konsultacje z zakresu ML i architektur chmurowych dla studentów i wykładowców. Jest autorem blisko 100 publikacji technicznych, w tym dwóch książek, których tematyka rozciąga się od chmurowego uczenia maszynowego po DevOps. Uzyskał certyfikat AWS Solutions Architect oraz Subject Matter Expert w dziedzinie ML, a także pomagał tworzyć system certyfikacji AWS Machine Learning. Uzyskał tytuł MBA na UC Davis, MS w dziedzinie Computer Information Systems na Uniwersytecie Kalifornijskim w Los Angeles oraz BS na Politechnice Kalifornijskiej w San Luis Obispo. Ma około 20 lat doświadczenia w programowaniu w Pythonie i ma tytuł Python Software Foundation Fellow. Pracował na rozmaitych stanowiskach, takich jak CTO, General Manager, Consulting CTO i Cloud Architect. Wśród firm, dla których pracował, wymienić można ABC, Caltech, Sony Imageworks, Disney Feature Animation, Weta Digital, AT&T, Turner Studios oraz Linden Lab. Obecnie zajmuje się konsultacjami dla startupów i innych firm w dziedzinie uczenia maszynowego i architektury chmurowej jako założyciel Pragmatic AI Labs.

Źródła ilustracji

- Rysunki 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 3.2, 3.3, 3.5, 3.6, 7.2, 7.3: Zrzut ekranu AWS © 2018, Amazon Web Services, Inc.
- Rysunki 4.1, 4.2, 4.5, 4.6: Zrzut ekranu GCP © Google LLC.
- Rysunek 4.7: © Noah Gift.
- Rysunki 5.3, 5.4: Zrzut ekranu Swagger © SmartBear Software.
- Rysunki 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.11, 6.13, 6.14: Zrzut ekranu Jupyter © 2018 Project Jupyter.
- Rysunek 7.1: Zrzut ekranu Slack © Slack Technologies.
- Rysunek 11.2: Zrzut ekranu Microsoft Azure © Microsoft 2018.

I

Wprowadzenie do pragmatycznej AI

1

Pragmatyczne podejście do AI

Nie należy mylić aktywności z osiągnięciami.

John Wooden

Skoro sięgnąłeś po tę książkę, zapewne jesteś ciekawy praktycznego i opartego na solidnych podstawach, czyli *pragmatycznego* podejścia do sztucznej inteligencji. Na rynku nie brakuje książek, kursów lub webinarów dotyczących najnowszych technik uczenia maszynowego czy uczenia pogłębionego (*deep learning*). Problem leży w tym, jak doprowadzić projekt do punktu, w którym w ogóle możliwe będzie użycie takich zaawansowanych technik. I właśnie dlatego powstała ta książka – ma stanowić most nad przepaścią oddzielającą teorię od realnych zagadnień związanych z implementacją projektów sztucznej inteligencji (*artificial intelligence* – AI).

W wielu przypadkach wytrenowanie własnego modelu nie jest wykonalne, gdyż zaangażowany zespół projektowy nie dysponuje wymaganą przez dany problem kombinacją dostępnego czasu, zasobów i umiejętności, aby wykonać implementację. Jednak istnieje lepsze rozwiązanie, niż podążanie ścieżką niemal na pewno prowadzącą do porażki. Praktyk w dziedzinie pragmatycznej AI użyje techniki odpowiedniej dla danej sytuacji. W niektórych przypadków może to oznaczać po prostu wywołanie interfejsu programowania aplikacji (API), który już zawiera wstępnie wytrenowany model. Inną pragmatyczną techniką może być utworzenie modelu, który jest zamierzenie mniej efektywny, ale za to łatwiejszy do zrozumienia i wdrożenia w produkcji.

W 2009 roku Netflix ogłosił znany konkurs z nagrodą w wysokości miliona dolarów dla rozwiązania, które pozwoli o 10 procent zwiększyć dokładność rekomendacji dla użytkowników. Konkurs był pasjonujący, a przede wszystkim wyprzedzał czas, jeśli chodzi o główny nurt analityki danych. Co jednak wiedzą tylko nieliczni, algorytm, który

zwyciężył w konkursie, nigdy nie został wdrożony z powodu kosztów implementacji (*https://www.wired.com/2012/04/netflix-prize-costs/*). Zamiast tego użyte zostały niektóre algorytmy zespołu, który osiągnął „tylko" 8,43 procent poprawy. To doskonały przykład, że prawdziwym celem wielu działań (nie tylko dotyczących AI) powinien być pragmatyzm, nie zaś perfekcja.

W książce tej koncentruję się właśnie na takim przypadku – byciu „tym drugim", ale właśnie tym, którego rozwiązanie trafia do produkcji. Ten wątek spaja całą książkę. Celem rozważań tu przedstawionych jest wysłanie kodu do produkcji – nie zaś stworzenie najlepszego rozwiązania, które nigdy nie przekształci się w finalny produkt.

Funkcjonalne wprowadzenie do Pythona

Python jest fascynującym językiem, gdyż jest dostatecznie dobry w tak wielu sytuacjach. Powtórzmy to: Python nie jest nadzwyczajnie dobry (a tym bardziej najlepszy) w żadnym przypadku, ale jest wystarczająco dobry w znakomitej większości zastosowań. Prawdziwą siłą tego języka jest zamierzony brak złożoności. W Pythonie można też programować w wielu różnych stylach. Jak najbardziej możliwe jest użycie Pythona do jawnego wykonywania instrukcji w sposób proceduralny, wiersz po wierszu. Jednak Python może również zostać użyty jako złożony, zorientowany obiektowo język wyposażony w zaawansowane funkcjonalności, takie jak metaklasy czy wielokrotne dziedziczenie.

Przy nauce Pythona, szczególnie w kontekście *data science*, pewne części języka wymagają mniej (lub wcale) uwagi. Można wręcz stwierdzić, że wiele spośród licznych elementów języka, głównie tych związanych z programowaniem zorientowanym obiektowo, nigdy nie zostaną użyte podczas pisania rozwiązań w Jupyter Notebooks (*notatnikach Jupyter* – tego terminu będziemy używać w całej książce). Zamiast tego przydatniejsze będzie podejście alternatywne, skupiające się na funkcjach. W tym rozdziale zawarłem skrótowe wprowadzenie do Pythona, traktowanego jak nowa wersja arkusza kalkulacyjnego, takiego jak Microsoft Excel.

Jeden z niedawnych absolwentów mojego kursu powiedział mi, że zanim zdecydował się na pójście na jakikolwiek wykład z uczenia maszynowego, obawiał się złożoności kodu. Po spędzeniu paru miesięcy na posługiwaniu się notatnikami Jupyter i Pythonem zauważył, że czuł się pewniej, używając Pythona do rozwiązywania problemów danych. Jestem głęboko przekonany, i jest to oparte na doświadczeniu, jakie zdobyłem podczas wykładów, że każdy, kto potrafi posługiwać się Excelem, może zostać użytkownikiem notatników Jupyter z poziomu Pythona.

Warto też zauważyć, że przy wdrażaniu kodu w produkcji Jupyter może być mechanizmem dostarczania rozwiązania, ale nie musi nim być. Istnieje wiele alternatywnych ścieżek wykorzystujących nowe platformy, takie jak Databricks, SageMaker czy Datalab, które

pozwalają na wdrożenia produkcyjne rozwiązań opartych na Jupyterze, ale często głównym zastosowaniem Jupytera jest eksperymentowanie.

> **Uwaga**
>
> Przedstawione poniżej „błyskawiczne wprowadzenie" do języka Python nie jest (i nie ma być) wyczerpującym wykładem składni i stosowania języka. Ma raczej być zachętą do bliższego poznania tego języka, pokazując funkcjonalności, które będą dalej używane w tej książce. Uważna lektura tego rozdziału zapewne pozwoli zrozumieć przykłady prezentowane w dalszej części, ale nie może zastąpić właściwej nauki języka, do czego zdecydowanie zachęcam.

Instrukcje proceduralne

Poniższe przykłady powstały przy założeniu, że zainstalowana została wersja Python 3.6. Najnowszą wersję Pythona można pobrać ze strony *https://www.python.org/downloads/*. *Instrukcje proceduralne* to dosłownie instrukcje, które można wydawać po jednym wierszu na raz. Kolejne podrozdziały prezentują różne typy instrukcji proceduralnych. Instrukcje te można uruchamiać w:

- Jupyter Notebook
- Powłoce IPython
- Interpreterze Pythona
- Skryptach Pythona

Drukowanie

Python udostępnia jedną z najprostszych form drukowania (wypisywania) wyników. `print` jest funkcją, która przyjmuje pewne wejście i przesyła je do konsoli:

```
In [1]: print("Hello world")
   ...:
Hello world
```

Tworzenie i używanie zmiennych

Zmienne tworzone są poprzez przypisanie wartości. Poniższy przykład przypisuje zmienną, po czym drukuje ją, przy czym dwie instrukcje zostały połączone w jednym wierszu przy użyciu średnika. Taki styl wykorzystujący średnik jest powszechny w środowisku Jupyter, ale byłby źle widziany w kodzie produkcyjnym i bibliotekach.

```
In [2]: variable = "armbar"; print(variable)
armbar
```

Wiele instrukcji proceduralnych

Pełne rozwiązanie jakiegoś problemu może wymagać napisania prostego kodu proceduralnego złożonego z kilku kolejnych instrukcji, jak poniżej. Taki styl jest sensowny w notatnikach Jupyter, ale raczej rzadko spotykany w kodzie produkcyjnym.

```
In [3]: attack_one = "kimura"
   ...: attack_two = "arm triangle"
   ...: print("In Brazilian jiu-jitsu a common attack is a:", attack_one)
   ...: print("Another common attack is a:", attack_two)
   ...:
In Brazilian jiu-jitsu a common attack is a: kimura
Another common attack is a: arm triangle
```

Obliczenia arytmetyczne

Języka Python można również użyć jako kalkulatora. To doskonały sposób, aby lepiej oswoić się ze środowiskiem i samym językiem; warto zacząć go używać zamiast sięgania po Excela czy aplikację kalkulatora.

```
In [4]: 7+5
   ...:
Out[4]: 12
```

Łączenie fraz (tekstów)

Ciągi (łańcuchy) znaków można dodawać do siebie.

```
In [6]: "arm" + "bar"
   ...:
Out[6]: 'armbar'
```

Złożone instrukcje

Można tworzyć bardziej złożone instrukcje wykorzystujące takie struktury danych, jak `belts` w poniższym przykładzie – w tym przypadku jest to lista.

```
In [7]: belts = ["white", "blue", "purple", "brown", "black"]
   ...: for belt in belts:
   ...:     if "black" in belt:
   ...:         print("The belt I want to earn is:", belt)
   ...:     else:
```

```
   ...:         print("This is not the belt I want to end up with:", belt)
   ...:
This is not the belt I want to end up with: white
This is not the belt I want to end up with: blue
This is not the belt I want to end up with: purple
This is not the belt I want to end up with: brown
The belt I want to earn is: black
```

Ciągi i formatowanie ciągów

Ciągi (`string`), nazywane też *łańcuchami*, są sekwencjami znaków i często są formatowane programowo. Niemal wszystkie programy w Pythonie wykorzystują ciągi, gdyż pozwala to przesyłać komunikaty do użytkowników programu. Tym niemniej jest kilka kluczowych koncepcji, które trzeba poznać.

- Ciągi można tworzyć przy użyciu pojedynczych, podwójnych i potrójnych/podwójnych cudzysłowów.
- Ciągi mogą być formatowane.
- Pewną komplikacją jest to, że ciągi mogą być kodowane na wiele sposobów, w tym przy użyciu Unicode.
- Do operowania na ciągach dostępnych jest wiele metod. W edytorze lub powłoce IPython można znaleźć te metody przy użyciu techniki dopełniania tabulatorem:

```
In [8]: basic_string = ""

In [9]: basic_string.
capitalize()   encode()       format()       isalpha()      islower()
istitle()      lower()        casefold()     endswith()     format_map()
isdecimal()    isnumeric()    isupper()      lstrip()       center()
expandtabs()   index()        isdigit()      isprintable()  join()
maketrans()    count()        find()         isalnum()      isidentifier()
isspace()      ljust()        partition()
```

Podstawowy ciąg

Najbardziej elementarną postacią ciągu jest zmienna, do której została przypisana pewna fraza ujęta w cudzysłowy. Cudzysłowy te mogą być potrójne, podwójne lub pojedyncze:

```
In [10]: basic_string = "Brazilian jiu-jitsu"
```

Dzielenie ciągów

Ciąg można przekształcić na listę, dzieląc go na podstawie spacji lub jakiegoś innego znaku separatora*:

```
In [11]: #dzielenie wg spacji (domyślnie)
    ...: basic_string.split()
Out[11]: ['Brazilian', 'jiu-jitsu']

In [12]: #dzielenie wg dywizów
    ...: string_with_hyphen = "Brazilian-jiu-jitsu"
    ...: string_with_hyphen.split("-")
    ...:

Out[12]: ['Brazilian', 'jiu-jitsu']
```

Wielkie litery

Python udostępnia wiele wbudowanych metod modyfikowania ciągów. Oto jak można przekształcić ciąg na same wielkie litery:

```
In [13]: basic_string.upper()
Out[13]: 'BRAZILIAN JIU JITSU'
```

Rozcinanie ciągów

Do zawartości ciągów można odwoływać się poprzez pozycję lub długość, wycinając pożądany fragment:

```
In [14]: #Odczytanie dwóch pierwszych znaków
    ...: basic_string[:2]
Out[14]: 'Br'
In [15]: #Odczytanie długości ciągu
    ...: len(basic_string)
Out[15]: 19
```

Łączenie (dodawanie) ciągów

Ciągi można dodawać do siebie, wykonując łączenie (konkatenację) lub przypisując ciąg do zmiennej i następnie budując dłuższe wyrażenia. Ten styl jest łatwo zrozumiały i intuicyjny w notatnikach Jupyter, ale ze względów wydajnościowych w kodzie produkcyjnym lepiej używać *f-ciągów*.

---

* W tym przykładzie (i w innych miejscach, w których treść jest znacząca) teksty w kodzie – w tym przypadku komentarze – zostały przetłumaczone, jednak w notatnikach Jupyter powiązanych z książką Czytelnik znajdzie teksty angielskie (wszystkie przypisy pochodzą od tłumacza).

```
In [16]: basic_string + " is my favorite Martial Art"
Out[16]: 'Brazilian jiu-jitsu is my favorite Martial Art'
```

Formatowanie ciągów na rozmaite sposoby

Jednym z najlepszych sposobów sformatowania ciągu w nowoczesnym Pythonie (wersje 3.x) jest użycie f-ciągów.

```
In [17]: f'I love practicing my favorite Martial Art,
         {basic_string}'
    ...:
Out[17]: 'I love practicing my favorite Martial Art,
         Brazilian jiu-jitsu'
```

Ciągi mogą używać potrójnych cudzysłowów, aby umożliwić zawijanie tekstu

Często użyteczne jest przypisanie jakiegoś fragmentu tekstu do zmiennej. Prostą metodą osiągnięcia tego w Pythonie jest użycie potrójnych cudzysłowów, aby zasygnalizować możliwość zawijania frazy.

```
In [18]: f"""
    ...: Ta fraza zawiera wiele zdań.
    ...: Fraza może zostać sformatowana jak prostsze zdania,
    ...: na przykład mogę nadal mówić o mojej ulubionej
         Sztuce walki {basic_string}
    ...: """
Out[18]: '\ Ta fraza zawiera wiele zdań.\n Fraza może
zostać sformatowana jak prostsze zdania,\n na przykład
mogę nadal mówić o mojej ulubionej Sztuce walki

Brazilian jiu-jitsu\n'
```

Można usunąć podziały wierszy przy użyciu metody replace

Pokazany wcześniej dłuższy tekst zawiera podziały wierszy, widoczne jako sekwencje znaków `\n`; można je usunąć przy użyciu metody `replace`:

```
In [19]: f"""
    ...: Ta fraza zawiera wiele zdań.
    ...: Fraza może zostać sformatowana jak prostsze zdania,
    ...: na przykład mogę nadal mówić o mojej ulubionej
         Sztuce walki {basic_string}
    ...: """
Out[18]: 'Ta fraza zawiera wiele zdań. Fraza może zostać sformatowana jak
prostsze zdania, na przykład mogę nadal mówić o mojej ulubionej Sztuce walki
Brazilian jiu-jitsu'
```

Liczby i operacje arytmetyczne

Python jest również kalkulatorem. Bez instalowania żadnych dodatkowych bibliotek można wykonywać wiele prostych i bardziej złożonych operacji arytmetycznych.

Dodawanie i odejmowanie liczb

Elastyczność języka Python objawia się też w tym, że umożliwia łączenie formatowania opartego na f-ciągach z operacjami arytmetycznymi.

```
In [20]: steps = (1+1)-1
    ...: print(f"Dwa kroki w przód: Jeden krok wstecz = {steps}")
    ...:
Dwa kroki w przód: Jeden krok wstecz = 1
```

Mnożenie liczb ułamkowych (dziesiętnych)

Język obsługuje również ułamki dziesiętne, dzięki czemu proste jest jednoznaczne formułowanie problemów.

```
In [21]:
    ...: body_fat_percentage = 0.10
    ...: weight = 200
    ...: fat_total = body_fat_percentage * weight
    ...: print(f"I weight 200lbs, and {fat_total}lbs of that is fat")
    ...:
I weight 200lbs, and 20.0lbs of that is fat
```

Potęgowanie

Biblioteka `math` pozwala na proste wykonywanie potęgowania, na przykład podniesienia liczby 2 do 3 potęgi, jak poniżej:

```
In [22]: import math
    ...: math.pow(2,3)
Out[22]: 8.0
```

Alternatywna metoda uzyskania tego samego wyniku to posłużenie się operatorem ** (dwie gwiazdki):

```
>>> 2**3
8
```

Konwertowanie pomiędzy różnymi typami numerycznymi

Trzeba mieć na uwadze, że w języku Python (podobnie jak w innych językach programowania) występuje wiele form zapisu liczb. Najbardziej powszechnie używane to

- Liczby całkowite (`int`)
- Liczby zmiennoprzecinkowe (`float`)

```
In [23]: number = 100
    ...: num_type = type(number).__name__
    ...: print(f"{number} is type [{num_type}]")
    ...:
100 is type [int]
In [24]: number = float(100)
    ...: num_type = type(number).__name__
    ...: print(f"{number} is type [{num_type}]")
    ...:

100.0 is type [float]
```

Zaokrąglanie liczb

Liczbę zawierającą wiele cyfr po kropce dziesiętnej można zaokrąglić do zadanej dokładności, jak poniżej:

```
In [26]: too_many_decimals = 1.912345897
    ...: round(too_many_decimals, 2)
Out[26]: 1.91
```

Struktury danych

Python zawiera kilka podstawowych struktur danych, z których najczęściej spotykane są

- Listy
- Słowniki

Słowniki i listy to prawdziwe konie robocze, ale w języku Python znajdziemy również inne struktury, takie jak krotki, zbiory, liczniki itp., które również warto poznać.

Słowniki

Słowniki są wykorzystywane w rozwiązywaniu bardzo różnorodnych problemów – podobnie jak sam Python. W poniższym przykładzie utworzyłem słownik zawierający nazwy ataków stosowanych w brazylijskim jiu-jitsu. „Kluczem" jest tu nazwa ataku, zaś „wartością" jest ta część ciała, względem której atak jest wykonywany.

```
In [27]: submissions = {"armbar": "upper_body",
    ...: "arm_triangle": "upper_body",
    ...: "heel_hook": "lower_body",
    ...: "knee_bar": "lower_body"}
    ...:
```

Typowy wzorzec wykorzystania słownika to iteracja przez jego zawartość przy użyciu metody `items`. W poniższym przykładzie wypisywane są klucz i wartość.

```
In [28]: for submission, body_part in submissions.items():
    ...:     print(f"The {submission} is an attack \
                on the {body_part}")
    ...:
The armbar is an attack on the upper_body
The arm_triangle is an attack on the upper_body
The heel_hook is an attack on the lower_body

The knee_bar is an attack on the lower_body
```

Słowniki mogą również posłużyć do filtrowania danych. W pokazanym niżej przykładzie wyświetlone zostaną tylko ataki kierowane na górną część ciała.

```
In [29]: print(f"These are upper_body submission attacks\
 in Brazilian jiu-jitsu:")
    ...: for submission, body_part in submissions.items():
    ...:     if body_part == "upper_body":
    ...:         print(submission)
    ...:
These are upper_body submission attacks in Brazilian jiu-jitsu:
armbar

arm_triangle
```

Można również wybierać poszczególne klucze oraz wartości słownika:

```
In [30]: print(f"These are keys: {submissions.keys()}")
    ...: print(f"These are values: {submissions.values()}")
    ...:
These are keys: dict_keys(['armbar', 'arm_triangle',
'heel_hook', 'knee_bar'])
These are values: dict_values(['upper_body', 'upper_body',

'lower_body', 'lower_body'])
```

Listy

Listy są strukturą powszechnie używaną w języku Python. Umożliwiają one tworzenie sekwencyjnych kolekcji. Warto podkreślić, że listy mogą zawierać słowniki, podobnie jak słowniki mogą zawierać listy.

```
In [31]: list_of_bjj_positions = ["mount", "full-guard",
                                  "half-guard", "turtle",
                                  "side-control", "rear-mount",
                                  "knee-on-belly", "north-south",
                                  "open-guard"]
    ...:

In [32]: for position in list_of_bjj_positions:
    ...:     if "guard" in position:
    ...:         print(position)
    ...:
full-guard
half-guard

open-guard
```

Listy mogą zostać również użyte do wybierania elementów poprzez rozdzielanie.

```
In [35]: print(f'First position: {list_of_bjj_positions[:1]}')
    ...: print(f'Last position: {list_of_bjj_positions[-1:]}')
    ...: print(f'First three positions:\
{list_of_bjj_positions[0:3]}')
    ...:
First position: ['mount']
Last position: ['open-guard']

First three positions: ['mount', 'full-guard', 'half-guard']
```

Funkcje

Funkcje są podstawowymi cegiełkami programowania analizy danych w Pythonie, ale są również sposobem na tworzenie logicznych, testowalnych struktur. Wśród użytkowników i twórców Pythona od dziesięcioleci toczy się spór o to, czy lepsze jest programowanie funkcyjne, czy zorientowane obiektowo. W tym rozdziale nie będę nawet próbować udzielenia odpowiedzi na to pytanie; pokażę jedynie użyteczność znajomości podstaw programowania funkcyjnego w Pythonie.