Biblioteki, pakiety i moduły

W większości wszystkie Części poprzedzające tą skupiały się na podstawowym języku Python, elementach języka, których będziesz potrzebować bez względu na to, jak zamierzasz używać Pythona w przyszłości. Ale jak widziałeś, wiele programów zaczyna się od zaimportowania jednego lub więcej modułów. Każdy moduł jest zasadniczo zbiorem wstępnie napisanego kodu, którego można użyć we własnym kodzie bez konieczności wymyślania tego koła na nowo. Dziadkiem całego tego wcześniej napisanego specjalistycznego kodu jest Standardowa Biblioteka Pythona.

Zrozumienie standardowej biblioteki Pythona

Standardowa biblioteka Pythona to w zasadzie wszystko, co otrzymujesz, gdy masz dostęp do języków Pythona. Obejmuje to wszystkie typy danych Pythona, takie jak string, integer, foat i Boolean. Każda instancja tych typów danych jest w rzeczywistości instancją klasy zdefiniowanej w bibliotece standardowej. Z tego powodu terminy typ, instancja i obiekt są często używane zamiennie. Liczba całkowita to liczba całkowita; jest to również typ danych w Pythonie. Ale istnieje, ponieważ biblioteka standardowa zawiera klasę dla liczb całkowitych, a każda liczba całkowita, którą tworzysz, jest w rzeczywistości instancją tej klasy, a zatem obiektem (ponieważ klasy są szablonami rzeczy zwanych obiektami). Funkcja type() w Pythonie zwykle identyfikuje typ fragmentu danych. na przykład uruchom te dwa wiersze kodu w wierszu poleceń Pythona, w notatniku Jupyter lub w pliku .py:

```
x = 3

print(type(x))
```

Dane wyjściowe to:

```
<class 'int'>
```

To mówi ci, że x jest liczbą całkowitą, a także, że jest instancją klasy int z biblioteki standardowej. Uruchomienie tego kodu:

```
x = 'howdy'

print(type(x))
```

Tworzy to wyjście:

```
<class 'str'>
```

Oznacza to, że x zawiera dane typu string utworzone przez klasę str Pythona. To samo działa w przypadku liczby owies (wartość liczbowa z kropką dziesiętną, na przykład 3,14) i wartości boolowskich (prawda lub fałsz).

Korzystanie z funkcji dir().

Standardowa biblioteka Pythona oferuje metodę dir(), która wyświetla listę wszystkich atrybutów powiązanych z typem. Na przykład w poprzednim przykładzie wynik <class 'str'> informuje, że dane są typu danych str. Więc wiesz, że jest to typ, a zatem w przypadku klasy o nazwie str (skrót od string). Wpisanie tej komendy:

```
dir(str)
```

Wyświetla coś takiego:

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
```

'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',

'__getnewargs__', '__gt__', '__hash__', '__init__','__init_subclass__',

'__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',

'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',

'__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize',

'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',

'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii',

'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',

'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',

'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',

'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',

'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

Nazwane elementy (nazwy otoczone podwójnym podkreśleniem) zwykle reprezentują coś, co jest wbudowane w Pythona i odgrywa pewną rolę w języku, do którego niekoniecznie masz bezpośredni dostęp. Są one często określane jako zmienne specjalne lub metody magiczne. Na przykład istnieje metoda __add__, która w rzeczywistości jest wywoływana za pomocą operatora (dodawania) + w celu dodania dwóch liczb lub połączenia dwóch ciągów znaków. Zwykłe funkcje nie mają podwójnych podkreśleń i są zwykle poprzedzone nawiasami. Na przykład spójrz na te wiersze kodu:

x = "Howdy"

print(type(x), x.isalpha(), x.upper())

Dane wyjściowe z tego kodu to:

<class 'str'> True HOWDY

Pierwsza część, <class 'str'> mówi ci, że x zawiera łańcuch. W związku z tym możesz użyć na nim dowolnego atrybutu pokazanego na wyjściu dir(str). Na przykład wartość True jest wynikiem funkcji x.isalpha(), ponieważ x zawiera znaki alfabetu. HOWDY jest wyjściem funkcji x.upper(), która konwertuje ciąg znaków na same wielkie litery. Początkujący często zastanawiają się, co dobrego, widząc kilka nazw, takich jak „wielkie litery”, „folder”, „środek”, „liczba”, „kodowanie”, „kończy się”, „rozwiń tabulatory”, „znajdź”, „formatuj” i tak dalej robi dla ciebie, gdy nie wiesz, co oznaczają nazwy lub jak ich używać. Cóż, tak naprawdę niewiele ci pomogą, jeśli nie będziesz ich dalej szukać. Możesz uzyskać bardziej szczegółowe informacje, używając help() zamiast dir.

Korzystanie z funkcji help().

Monit Pythona oferuje również funkcję help() o składni:

help(object)

Aby z niego skorzystać, zastąp object typem obiektu, z którym szukasz pomocy. Na przykład, aby uzyskać pomoc dotyczącą obiektów str (łańcuchów znaków pochodzących z klasy str), wpisz to polecenie w wierszu poleceń Pythona:

help(str)

Dane wyjściowe będą zawierały bardziej istotne informacje na dany temat w nawiasach. Na przykład dir(str) podaje nazwy atrybutów tego typu, help(dir) dostarcza więcej szczegółów o każdym elemencie. Na przykład, podczas gdy dir(str) mówi ci, że w klasie str istnieje coś takiego jak pisanie wielkimi literami, help powie ci trochę więcej na ten temat, jak następuje:

capitalize(self, /)

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

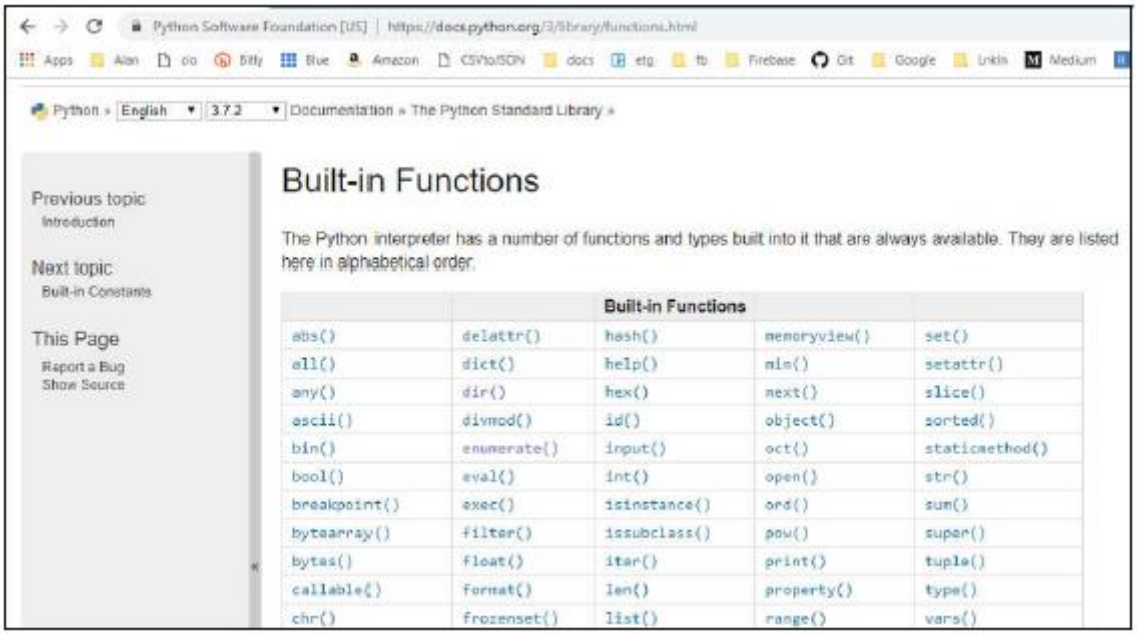
Słowo self oznacza po prostu, że każde słowo, które zapiszesz wielką literą, zostanie zapisane wielką literą. Znak / na końcu oznacza koniec parametrów tylko pozycyjnych, co oznacza, że nie można używać słów kluczowych z parametrami po nich, tak jak w przypadku definiowania własnych funkcji. To, co zwykle działa najlepiej dla większości ludzi, to bardziej szczegółowe wyjaśnienie i jeden lub więcej przykładów. Dla tych, Google lub podobna wyszukiwarka jest zwykle najlepszym wyborem. Rozpocznij wyszukiwanie od słowa Python (aby wiedziało również, czego dotyczy wyszukiwanie), po którym następuje dokładne słowo, z którym szukasz pomocy. Na przykład wyszukiwanie w Google dla

python capitalize

. . . zawiera łącza do wielu różnych zasobów umożliwiających poznanie atrybutu wielkich liter obiektu str, w tym przykładów jego użycia. Jeśli znudzi Ci się naciskanie dowolnego klawisza, aby ominąć Więcej . . . na końcu każdej strony pomocy wystarczy nacisnąć Ctrl + C. Spowoduje to powrót do zachęty Pythona. Oczywiście naprawdę dobrym (choć technicznym) zasobem dla biblioteki standardowej Pythona jest sama dokumentacja biblioteki standardowej. Jest to zawsze dostępne pod adresem https://docs.python.org/, zazwyczaj pod linkiem Library Reference. Ale nawet to sformułowanie może się zmienić, więc jeśli masz wątpliwości, po prostu googluj standardową bibliotekę Pythona. Ostrzegam tylko, że jest ogromna i bardzo techniczna. Więc nie oczekuj, że zapamiętasz, a nawet zrozumiesz wszystko od razu. Używaj jej jako stałego źródła wiedzy o rzeczach, które Cię interesują w miarę rozwoju Twojej wiedzy o Pythonie. Dokumentacja, która pojawia się na stronie docs.python.org, dotyczy zazwyczaj bieżącej stabilnej wersji. Łącza do starszych wersji oraz do wszelkich nowszych wersji, które mogą być w trakcie opracowywania podczas odwiedzania, są dostępne z linków po lewej stronie strony.

Eksplorowanie wbudowanych funkcji

Zarówno dir(), jak i help() są przykładami wbudowanych funkcji Pythona. Są to funkcje, które są zawsze dostępne w Pythonie, w każdej tworzonej aplikacji, a także w wierszu poleceń Pythona. Te wbudowane funkcje są również częścią standardowej biblioteki. W rzeczywistości, jeśli wygooglujesz wbudowane funkcje Pythona, niektóre wyniki wyszukiwania będą wskazywać bezpośrednio na dokumentację Pythona. Kliknięcie tego łącza spowoduje otwarcie tej sekcji dokumentacji biblioteki standardowej i wyświetlenie tabeli wszystkich wbudowanych funkcji, jak na rysunku . Na tej stronie możesz kliknąć nazwę dowolnej funkcji, aby dowiedzieć się więcej na jej temat.



Eksplorowanie pakietów Pythona

Język Python obsługuje programowanie modułowe, w którym chodzi o to, że nie ma potrzeby, aby każdy programista wymyślał wszystko na nowo. Nawet największe projekty można podzielić na mniejsze, łatwiejsze w zarządzaniu komponenty lub moduły. W rzeczywistości duży projekt może wymagać pewnych komponentów, które już istnieją gdzieś na świecie, ponieważ ktoś już potrzebował tej funkcjonalności i poświęcił czas na jej stworzenie, przetestowanie i debugowanie. Każdy duży projekt, niezależnie od tego, czy pracujesz sam, czy jako członek zespołu, można uprościć i usprawnić, jeśli niektóre komponenty mogą wykorzystywać wypróbowany i prawdziwy kod, który został już napisany, przetestowany, debugowany i uznany za niezawodny przez członków społeczności programistów Pythona . Pakiety, o których słyszysz w odniesieniu do Pythona, są dokładnie takim rodzajem kodu - kodem, który został opracowany i pielęgnowany, jest godny zaufania i na tyle ogólny, że można go użyć jako komponentu jakiegoś dużego projektu, nad którym pracujesz. Istnieją dosłownie tysiące pakietów dostępnych dla Pythona. Dobrym źródłem do znajdowania pakietów jest PyPi, sprytna nazwa, łatwa do zapamiętania i będąca skrótem od Python Package Index. Możesz to sprawdzić w dowolnym momencie za pomocą dowolnej przeglądarki i adresu URL https://pypi.org/. Istnieje również program o nazwie pip, inna sprytna nazwa, która oznacza pakiety instalacyjne Pip. Jest powszechnie nazywany menedżerem pakietów, ponieważ można go również używać do eksplorowania, aktualizowania i usuwania istniejących pakietów. Aby użyć pip, musisz przejść do wiersza polecenia systemu operacyjnego, którym jest Terminal na komputerze Mac lub cmd.exe lub Powershell w systemie Windows. Jeśli używasz VS Code, najprostszym sposobem, aby się tam dostać, jest otwarcie VS Code i wybranie View Terminal z paska menu. Jeśli masz już pip, wpisanie tego polecenia w wierszu polecenia powie ci, która wersja pip jest aktualnie zainstalowana:

pip –version

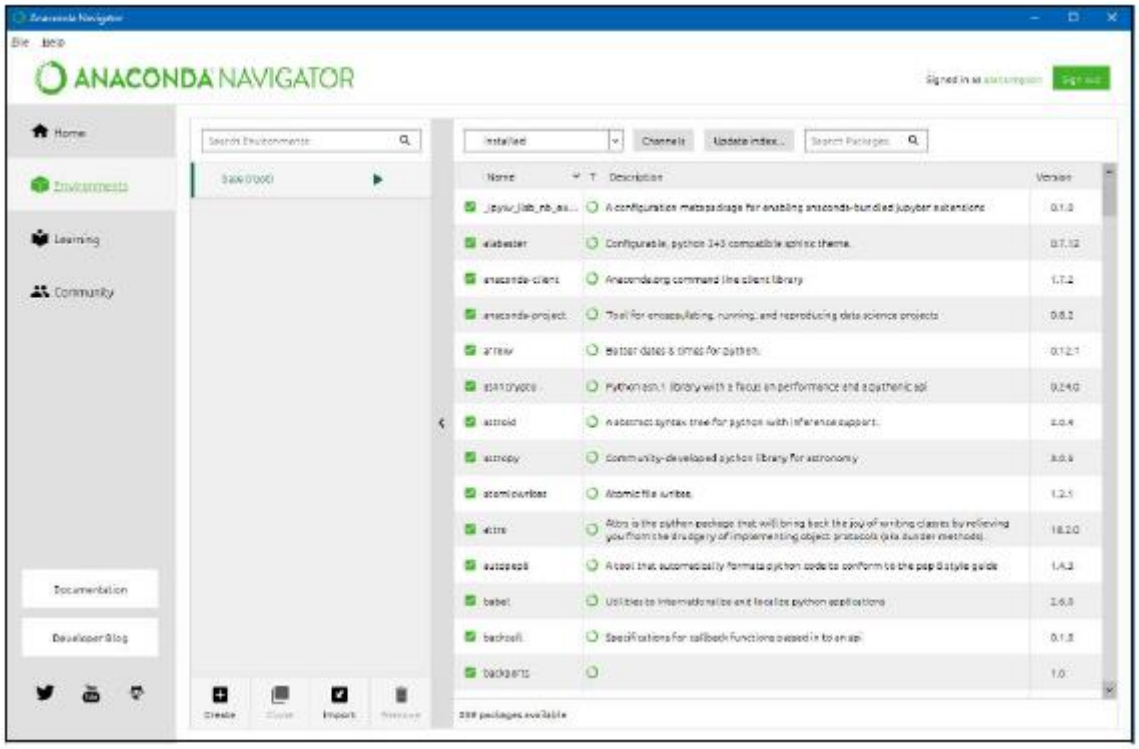
Wynik prawdopodobnie będzie wyglądał mniej więcej tak (ale z numerami i nazwami Twojej wersji):

pip 18.1 z C:\Users\...\AppData\Local\Continuum\anaconda3\lib\site-packages\pip (python 3.7)

Aby zobaczyć, jakie pakiety już zainstalowałeś, wpisz to w wierszu polecenia systemu operacyjnego:

pip list

Większość ludzi jest zaskoczona ilością zainstalowanych pakietów. To bardzo długa lista i trzeba trochę przewinąć w górę i w dół, aby zobaczyć wszystkie nazwiska. Jednak jedną z zalet instalacji Pythona z Anacondą jest to, że otrzymujesz wiele świetnych pakietów w miksie. I nie musisz polegać na liście pip, aby znaleźć ich nazwy. Zamiast tego po prostu otwórz Anaconda Navigator i kliknij Środowiska w lewej kolumnie. Zobaczysz listę wszystkich zainstalowanych pakietów wraz z krótkim opisem i numerem wersji każdego z nich, jak pokazano na przykładzie na rysunku



Chociaż używanie pip do instalowania pakietów, których jeszcze nie masz, jest całkowicie w porządku, jedną wadą jest to, że te pakiety mogą nie pojawiać się na liście zainstalowanych pakietów Anaconda Navigator. Aby to obejść, za każdym razem, gdy zobaczysz instrukcję, aby coś zainstalować, możesz spróbować zastąpić słowo pip słowem conda (skrót od Anaconda). Spowoduje to dodanie pakietu do Twojej kolekcji, więc pojawi się zarówno podczas tworzenia listy pip, jak i podczas przeglądania listy w Anaconda Navigatorze.

Importowanie modułów Pythona

Cały czas będziesz słyszeć słowo moduł używane w połączeniu z Pythonem. Jeśli myślisz o bibliotece standardowej jako o rzeczywistej bibliotece fizycznej, a pakiet jako być może jedną książąe w tej bibliotece, możesz myśleć o module jako o jednym rozdziale w jednej książce. Innymi słowy, pakiet może zawierać wiele modułów, a biblioteka może zawierać wiele pakietów. Moduł jest dużą częścią tego, co sprawia, że Python jest językiem modułowym, ponieważ kod jest pogrupowany według funkcji. Z jednej strony nie musisz importować wszystkiego, w tym zlewu kuchennego, aby użyć kodu. Z tego samego powodu, jeśli potrzebujesz użyć kilku powiązanych rzeczy, takich jak funkcje do pracy z datami i godzinami, nie musisz importować ich wszystkich pojedynczo. Zazwyczaj zaimportowanie tylko całego modułu zapewni Ci to, czego potrzebujesz. Na przykład, aby pracować z datami i godzinami, nie musisz importować każdego dostępnego modułu Pythona. Nie musisz też importować każdej możliwej małej funkcji i obiektu pojedynczo. Wystarczy zaimportować cały moduł, na przykład datetime, aby uzyskać wiele przydatnych rzeczy do pracy z datami i godzinami. W rzeczywistości istnieje kilka sposobów importowania funkcjonalności z modułów. Jednym z najczęstszych jest po

prostu zaimportowanie całego modułu. Aby to zrobić, po prostu wykonaj polecenie import z nazwą modułu, który chcesz zaimportować. Na przykład importuje to cały moduł matematyczny, który ma wiele funkcji i rzeczy do wykonywania działań matematycznych:

import math

Po zaimportowaniu modułu funkcje dir() i help() działają również na nim. Na przykład, jeśli przed importem matematyki próbowałeś wykonać komendę dir(math) lub help(math), pojawiłby się błąd. To dlatego, że ten pakiet matematyczny nie jest częścią standardowej biblioteki. Jeśli jednak najpierw zaimportujesz math, a następnie help(math), wszystko będzie działać. Może się zdarzyć, że tak naprawdę nie potrzebujesz całego zestawu i kaboodle. W takich przypadkach możesz zaimportować tylko to, czego potrzebujesz, używając takiej składni:

from math import pi

W tym przykładzie importujesz tylko jedną rzecz (pi), więc nie wprowadzasz niepotrzebnych rzeczy. Drugi przykład jest również przydatny, ponieważ w swoim kodzie możesz odnosić się do pi jako po prostu pi, nie musisz używać math.pi. Oto kilka rzeczy, które możesz wypróbować w wierszu poleceń Pythona, na przykład w oknie VS Code Terminal, aby samemu się przekonać: Wpisz polecenie print(pi) i naciśnij Enter. Najprawdopodobniej pojawi się błąd o treści:

NameError: name 'pi' is not defined

Innymi słowy, pi nie jest częścią standardowej biblioteki, która jest zawsze dostępna dla kodu Pythona. Aby z niego skorzystać, należy zaimportować moduł math. Można to zrobić na dwa sposoby. Możesz zaimportować całość, wpisując to w wierszu poleceń Pythona:

import math

Ale jeśli to zrobisz, a następnie wejdziesz

print(pi)

. . . ponownie pojawi się ten sam błąd, mimo że zaimportowałeś pakiet math. Powodem tego jest to, że gdy importujesz cały moduł i chcesz użyć jego części, musisz poprzedzić część, której chcesz użyć nazwą modułu i kropką. Na przykład, jeśli wpiszesz to polecenie:

print(math.pi)

. . . otrzymujesz poprawną odpowiedź:

3.141592653589793

Należy pamiętać, że podczas importowania tylko części modułu funkcje help() i dir() dla całego modułu nie będą działać. Na przykład, jeśli w swoim kodzie wykonywałeś tylko z importu math pi i próbujesz wykonać funkcję dir(math) lub help(math), to nie zadziała, ponieważ Python nie ma całego modułu sprzedaż. Ma do dyspozycji tylko to, co zaimportowałeś, pi w tym przykładzie.

Zwykle help() i dir() to po prostu rzeczy, których używasz w Pythonie do szybkiego wyszukiwania. Nie są to rzeczy, których prawdopodobnie będziesz używać podczas pisania aplikacji. Tak więc korzystanie z opcji from zamiast importu jest w rzeczywistości bardziej efektywne, ponieważ wprowadzasz tylko to, czego potrzebujesz. Jako dodatkowy bonus nie musisz poprzedzać nazwy funkcji nazwą modułu i kropką. Na przykład, gdy importujesz tylko pi z modułu math, tak jak poniżej:

from math import pi

Wtedy możesz odnieść się do pi w swoim kodzie to po prostu pi, a nie math.pi. Innymi słowy, jeśli wykonasz tę funkcję:

print(pi)

. . . otrzymasz właściwą odpowiedź:

3.141592653589793

Dzieje się tak, ponieważ Python „wie" teraz, że liczba pi to rzecz o nazwie pi z modułu math; nie musisz specjalnie mówić mu, aby używał math.pi. Możesz także zaimportować wiele elementów z paczki, wymieniając ich nazwy, oddzielone przecinkami, na końcu polecenia from .... Załóżmy na przykład, że potrzebujesz liczby pi i pierwiastków kwadratowych w swojej aplikacji. Możesz zaimportować tylko te do swojej aplikacji, używając tej składni:

from math import pi, sqrt

Jeszcze raz, ponieważ użyłeś składni from do importu, możesz odwoływać się do pi i sqrt() w swoim kodzie według nazwy bez wiodącej nazwy modułu. Na przykład po wykonaniu tej instrukcji from ten kod:

print(sqrt(pi))

. . . wyświetla

1.7724538509055159

. . . która, jak zapewne się domyślasz, jest pierwiastkiem kwadratowym z liczby pi. Możesz również zauważyć, że ludzie importują taki moduł:

from math import *

Gwiazdka to skrót od „wszystko". Tak więc w istocie to polecenie jest dokładnie takie samo, jak import math, które również importuje cały moduł matematyczny. Ale jest to subtelna różnica: kiedy robisz z importu math *, kojarzysz nazwę wszystkiego w module math z tym modułem. Więc możesz używać tych nazw bez matematyki. przedrostek. Innymi słowy, po wykonaniu tego:

from math import *

. . . możesz wykonać polecenie takie jak print(pi) i będzie działać, nawet bez użycia print(math.pi). Chociaż wydaje się to sprytną i wygodną rzeczą, powinniśmy zauważyć, że wielu programistów uważa, że takie rzeczy nie są zbyt Pythoniczne. Jeśli importujesz wiele modułów i używasz wielu różnych elementów każdego z nich, unikanie nazw modułów w kodzie może utrudnić innym programistom odczytanie i zrozumienie tego kodu. Więc chociaż w sensie Zen Pythona może to być mile widziane, technicznie działa i jest opcją dla Ciebie.

Tworzenie własnych modułów

Pomimo całego zamieszania wokół modułów, moduł jest w rzeczywistości całkiem prostą rzeczą. W rzeczywistości jest to po prostu plik z rozszerzeniem .py, który zawiera kod Pythona. Otóż to. Tak więc za każdym razem, gdy piszesz kod w Pythonie i zapisujesz go w pliku .py, w zasadzie tworzysz moduł. Nie oznacza to, że zawsze musisz używać tego kodu jako modułu. Z pewnością można ją traktować jako samodzielną aplikację. Ale gdybyś chciał stworzyć własny moduł z kodem, którego często potrzebujesz we własnej pracy, z pewnością mógłbyś to zrobić. W tej sekcji wyjaśniamy cały proces. Moduł to także po prostu plik z rozszerzeniem nazwy pliku .py. Nazwa modułu jest taka sama jak nazwa pliku (bez

rozszerzenia .py). Jak każdy plik .py, moduł zawiera kod Pythona. Jako przykład roboczy załóżmy, że chcesz mieć trzy funkcje upraszczające formatowanie dat i wartości walut. Możesz wymyślić dowolną nazwę dla każdej funkcji. W naszym przykładzie roboczym użyjemy tych trzech nazw:

to_date(any_str): Pozwala przekazać dowolny ciąg znaków (any_str) datę w formacie mm/dd/rr lub mm/dd/rrrr i odsyła Python datetime.date, którego można użyć do obliczenia daty.

mdy(dowolna_data): Pozwala przekazać dowolną datę lub godzinę w Pythonie i zwraca ciąg daty sformatowany w formacie mm/dd/rrrr do wyświetlenia na ekranie.

to_curr(any_num, len): Pozwala przekazać dowolną liczbę całkowitą lub całkowitą w Pythonie i zwraca ciąg z początkowym znakiem dolara, przecinkami w tysiącach miejsc i dwiema cyframi oznaczającymi grosze. Długość jest opcjonalną liczbą. Jeśli zostanie podany, zwracana wartość zostanie uzupełniona spacjami po lewej stronie, aby dopasować określoną długość

Oto cały kod do tego:

Contains custom functions for dates and currency values.

import datetime as dt

def to_date(any_str):

""" Convert mm/dd/yy or mm/dd/yyyy string to datetime.date, or None if

invalid date. """

try:

if len(any_str) == 10:

the_date = dt.datetime.strptime(any_str,'%m/%d/%Y').date()

else:

the_date = dt.datetime.strptime(any_str,'%m/%d/%y').date()

except (ValueError, TypeError):

the_date = None

return the_date

def mdy(any_date):

""" Returns a string date in mm/dd/yyyy format. Pass in Python date or

string date in mm/dd/yyyy format """

if type(any_date) == str:

any_date = to_date(anydate)

Make sure its a dateime being forwarded

if isinstance(any_date,dt.date):

s_date = f"{any_date:'%m/%d/%Y'}"

else:

```
s_date = "Invalid date"

return s_date

def to_curr(anynum, len=0):

""" Returns a number as a string with $ and commas. Length is optional """

s = "Invalid amount"

try:

x = float(anynum)

except ValueError:

x= None

if isinstance(x,float):

s = '$' f"{x:,.2f}"

if len > 0:

s=s.rjust(len)

return s
```

Możesz sam utworzyć ten sam plik i nazwać go myfunctions.py, jeśli chcesz podążać dalej. Zauważ, że plik zawiera tylko funkcje. Więc jeśli go uruchomisz, nie zrobi nic na ekranie, ponieważ nie ma tam kodu, który wywołuje którąkolwiek z tych funkcji.

Aby używać tych funkcji w dowolnej napisanej przez siebie aplikacji lub programie w języku Python, najpierw skopiuj ten plik myfunc.py do tego samego folderu, w którym znajduje się reszta kodu w języku Python, który piszesz. Następnie, kiedy tworzysz nową stronę, możesz zaimportować myfunc jako moduł, tak jak każdy inny moduł stworzony przez kogoś innego. Po prostu użyj

```
import myfunc
```

Będziesz musiał użyć nazwy modułu przed każdą funkcją, którą wywołujesz z tego modułu. Więc jeśli chcesz, aby kod był trochę bardziej czytelny, możesz zamiast tego użyć tego:

```
import myfunc as my
```

Mając to jako linię otwierającą, możesz odwołać się do dowolnej funkcji w swoim module niestandardowym za pomocą my. jako przedrostek. Na przykład my.to_date() w celu wywołania funkcji to_date. Oto strona, która importuje moduł, a następnie testuje wszystkie trzy funkcje przy użyciu mojej składni:

```
# Import all the code from myfunc.py as my.

import myfunc as my

# Need dates in this code

from datetime import datetime as dt

# Some simple test data.
```

```
string_date="12/31/2019"

# Convert string date to datetime.date

print(my.to_date(string_date))

today = dt.today()

# Show today's date in mm/dd/yyyy format.

print(my.mdy(today))

dollar_amt=12345.678

# Show this big number in currency format.

print(my.to_curr(dollar_amt))
```

Po uruchomieniu tego kodu, zakładając, że wszystko jest wpisane poprawnie i bez błędów, dane wyjściowe powinny wyglądać następująco:

```
2019-12-31

'12/27/2018'

$12,345.68
```

Wspomnieliśmy też wcześniej, że można pominąć stosowanie przedrostka, jeśli importujesz elementy według nazwy. W tym przypadku oznacza to, że możesz wywołać metody to_date() i mdy() oraz to_curr() bez użycia funkcji my. prex. Pierwsza linia kodu musiałaby być

```
from myfunc import to_date, mdy, to_curr
```

Reszta kodu byłaby taka sama jak w poprzednim przykładzie, z wyjątkiem tego, że możesz pominąć my. prefiksy jak w poniższym kodzie:

```
# Import all the code from myfunc.py by name.

from myfunc import to_date, mdy, to_curr

# Need dates in this code

from datetime import datetime as dt

# Some simple test data.

string_date="12/31/2019"

# Convert string date to datetime.date

print(to_date(string_date))

today = dt.today()

# Show today's date in mm/dd/yyyy format.

print(mdy(today))

dollar_amt=12345.678
```

```
# Show this big number in currency format.

print(to_curr(dollar_amt))
```

To tyle w przypadku bibliotek, pakietów i modułów Pythona. Cała sprawa może być dość myląca, ponieważ ludzie używają tych terminów zamiennie. A to dlatego, że wszystkie reprezentują kod napisany przez innych, którego możesz używać w dowolnym kodzie Pythona, który sam napiszesz. Jedyną prawdziwą różnicą jest rozmiar. Biblioteka może zawierać kilka pakietów. Pakiet może zawierać kilka modułów. Same moduły zazwyczaj zawierają funkcje, klasy lub inne gotowe fragmenty kodu, z których możesz swobodnie korzystać. W kolejnych książkach i rozdziałach zobaczysz wiele modułów i klas, ponieważ to właśnie one sprawiają, że Python jest tak modułowy i ma zastosowanie w wielu różnych rodzajach pracy i nauki. Pamiętaj jednak, że podstawowe zasady języka Python, których nauczyłeś się w tych trzech pierwszych książkach, mają zastosowanie wszędzie, niezależnie od tego, czy zajmujesz się nauką o danych, sztuczną inteligencją czy robotyką. Będziesz po prostu używać tego podstawowego języka do pracy z kodem, który inni już napisali dla tego jednego wyspecjalizowanego obszaru.