**Utilizing Packages**

Do tej pory używaliśmy tylko funkcji i typów, które są zawsze dostępne globalnie, ale istnieje wiele funkcji, których możemy użyć, jeśli importujemy je ze standardowej biblioteki. Importowanie pakietów można wykonać na kilka różnych sposobów, ale najprostszym jest użycie instrukcji importu. Oto, jak możemy zaimportować pakiet czasowy do użytku:

>>> import time

>>>

Importowanie pakietu pozwala nam uzyskać dostęp do funkcji i klas, które definiuje. Możemy to zrobić, odłączając nazwę pakietu. Wywołajmy funkcję localtime zapewnianą przez pakiet czasu:

>>> now = time.localtime()

>>> now

time.struct\_time(tm\_year=2018, tm\_mon=1, tm\_mday=26, tm\_hour=15, tm\_min=32, tm\_sec=43, tm\_wday=4, tm\_yday=26, tm\_isdst=0)

Wywołanie tej funkcji zwraca time.struct\_time do użycia, który ma pewne atrybuty, z którymi możemy wchodzić w interakcje za pomocą kropki (.):

>>> now.tm\_hour

15

Oto nasza pierwsza interakcja z atrybutem obiektu, który nie jest funkcją. Czasami musimy uzyskać dostęp do danych z obiektu, a do tego nie musimy używać nawiasów.

#### Working with Environment Variables

Importując pakiet os, możemy uzyskać dostęp do wielu różnych atrybutów i funkcji na poziomie systemu operacyjnego, z których najmniejszą jest obiekt środowiskowy. Ten obiekt zachowuje się jak słownik, więc możemy użyć operacji indeksowania do odczytu z niego.

Stwórzmy prosty skrypt, który odczyta zmienną środowiskową „STAGE” i wydrukujemy, na jakim etapie aktualnie się znajdujemy:

#!/usr/bin/env python3.6

import os

stage = os.environ["STAGE"].upper()

output = f"We're running in {stage}"

if stage.startswith("PROD"):

output = "DANGER!!! - " + output

print(output)

Możemy ustawić zmienną środowiskową, gdy uruchomimy skrypt, aby przetestować różnice:

$ STAGE=staging running

We're running in STAGING

$ STAGE=production running

DANGER!!! - We're running in PRODUCTION

Co się stanie, jeśli zmienna środowiskowa „STAGE” nie jest ustawiona?

$ running

Traceback (most recent call last):

File "/home/user/bin/running", line 5, in

stage = os.environ["STAGE"].upper()

File "/usr/local/lib/python3.6/os.py", line 669, in \_\_getitem\_\_

raise KeyError(key) from None

KeyError: 'STAGE'

Ten potencjalny KeyError to największy upadek korzystania z os.environ i powód, dla którego zwykle będziemy używać os.getenv.

#### Handling A Missing Environment Variable

Jeśli zmienna środowiskowa „STAGE” nie jest ustawiona, chcemy ustawić domyślnie „DEV” i możemy to zrobić za pomocą funkcji os.getenv:

#!/usr/bin/env python3.6

import os

stage = os.getenv("STAGE", "dev").upper()

output = f"We're running in {stage}"

if stage.startswith("PROD"):

output = "DANGER!!! - " + output

print(output)

Jeśli teraz uruchomimy nasz skrypt bez „ETAPU”, nie będziemy mieć błędu:

$ running

We're running in DEV

#### Interacting with Files

Bardzo często trzeba czytać zawartość pliku w skrypcie, a Python sprawia, że jest to dla nas całkiem łatwe. Zanim zaczniemy, utwórzmy plik tekstowy, który możemy odczytać z pliku xmen\_base.txt:

Storm

Wolverine

Cyclops

Bishop

Nightcrawler

Teraz, gdy mamy plik do pracy, poeksperymentujmy z REPL przed napisaniem skryptów wykorzystujących pliki.

#### Opening and Reading a File

Zanim będziemy mogli odczytać plik, musimy otworzyć połączenie z plikiem. Otwórzmy plik xmen\_base.txt, aby zobaczyć, co może zrobić obiekt pliku:

>>> xmen\_file = open('xmen\_base.txt', 'r')

>>> xmen\_file

<\_io.TextIOWrapper name='xmen\_base.txt' mode='r' encoding='UTF-8'>

The [open](https://docs.python.org/3/library/functions.html#open) function allows us to connect to our file by specifying the path and the mode. We can see that our xmen\_file object is an \_io.TextIOWrapper so we can look at [the documentation](https://docs.python.org/3/library/io.html#io.TextIOWrapper) to see what we can do with that type of object.

There is a read function so let’s try to use that:

>>> xmen\_file.read()

'Storm\nWolverine\nCyclops\nBishop\nNightcrawler\n'

>>> xmen\_file.read()

''

read gives us all of the content as a single string, but notice that it gave us an empty string when we called the function as second time. That happens because the file maintains a cursor position and when we first called read the cursor was moved to the very end of the file’s contents. If we want to reread the file we’ll need to move the beginning of the file using the seek function like so:

>>> xmen\_file.seek(0)

0

>>> xmen\_file.read()

'Storm\nWolverine\nCyclops\nBishop\nNightcrawler\n'

>>> xmen\_file.seek(6)

6

>>> xmen\_file.read()

'Wolverine\nCyclops\nBishop\nNightcrawler\n'

By seeking to a specific point of the file, we are able to get a string that only contains what is after our cursor’s location.

Another way that we can read through content is by using a for loop:

>>> xmen\_file.seek(0)

0

>>> for line in xmen\_file:

... print(line, end="")

...

Storm

Wolverine

Cyclops

Bishop

Nightcrawler

>>>

Notice that we added a custom end to our printing because we knew that there were already newline characters (\n) in each line.

Po zakończeniu pracy z plikiem importujemy zamknięcie połączenia z plikiem przy użyciu funkcji zamknięcia:

>>> xmen\_file.close()

>>> xmen\_file.read()

Traceback (most recent call last):

File "", line 1, in

ValueError: I/O operation on closed file.

>>>

#### Creating a New File and Writing to It

Znamy teraz podstawy czytania pliku, ale będziemy również musieli wiedzieć, jak zapisywać zawartość do plików. Stwórzmy kopię naszego pliku xmen, który możemy dodać do:

>>> xmen\_base = open('xmen\_base.txt')

>>> new\_xmen = open('new\_xmen.txt', 'w')

Musimy ponownie otworzyć nasze poprzednie połączenie z xmen\_base.txt, abyśmy mogli je ponownie przeczytać. Następnie tworzymy połączenie z plikiem, który jeszcze nie istnieje i ustawiamy tryb na w, co oznacza „pisz”. Przeciwieństwem funkcji odczytu jest funkcja zapisu i możemy użyć obu do wypełnienia naszego nowego pliku:

>>> new\_xmen.write(xmen\_base.read())

>>> new\_xmen.close()

>>> new\_xmen = open(new\_xmen.name, 'r+')

>>> new\_xmen.read()

'Storm\nWolverine\nCyclops\nBishop\nNightcrawler\n'

Zrobiliśmy tam całkiem sporo, przełammy to:

1. Czytamy z pliku bazowego i używamy wartości zwracanej jako argumentu do zapisu naszego nowego pliku.

2. Zamknęliśmy nowy plik.

3. Ponownie otworzyliśmy nowy plik, używając trybu r +, który pozwoli nam czytać i zapisywać zawartość do pliku.

4. Czytamy zawartość z nowego pliku, aby upewnić się, że napisał poprawnie.

Teraz, gdy mamy plik, który możemy odczytać i napisać, dodajmy jeszcze kilka nazw:

>>> new\_xmen.seek(0)

>>> new\_xmen.write("Beast\n")

6

>>> new\_xmen.write("Phoenix\n")

8

>>> new\_xmen.seek(0)

0

>>> new\_xmen.read()

'Beast\nPhoenix\ne\nCyclops\nBishop\nNightcrawler\n'

Co tu się stało? Ponieważ używamy r +, nadpisujemy plik na podstawie każdego znaku, ponieważ staraliśmy się wrócić do początku pliku. Jeśli ponownie otworzymy plik w trybie w, wcześniej istniejąca zawartość zostanie obcięta.

#### Appending to a File

Dość częstą rzeczą, którą chcesz zrobić, jest dołączenie do pliku bez czytania jego bieżącej zawartości. Można to zrobić za pomocą trybu a. Zamknijmy plik xmen\_base.txt i otwórzmy go ponownie w trybie, aby dodać inną nazwę, nie martwiąc się o utratę oryginalnej treści. Tym razem użyjemy instrukcji with do tymczasowego otwarcia pliku i automatycznego zamknięcia go po wykonaniu naszego bloku kodu:

>>> xmen\_file.close()

>>> with open('xmen\_base.txt', 'a') as f:

... f.write('Professor Xavier\n')

...

17

>>> f = open('xmen\_base.txt', 'a')

>>> with f:

... f.write("Something\n")

...

10

>>> exit()

Aby przetestować to, co właśnie zrobiliśmy, wypiszmy zawartość tego pliku:

$ cat xmen\_base.txt

Storm

Wolverine

Cyclops

Bishop

Nightcrawler

Professor Xavier

Something