1_Python_intro

January 27, 2019

0.0.1 Cześć! Skoro tutaj trafiłeś, to najprawdopodobniej należysz do jednej z dwóch grup: albo zastanawiasz się, w jaki sposób można przetwarzać obrazy cyfrowe, albo: co jeszcze można zrobić w Pythonie? Mamy nadzieję, że w naszym poradniku znajdziesz odpowiedź. A nawet jeśli wpadłeś na niego zupełnie przypadkowo, to i tak lektura może okazać się całkiem interesująca ;)

Poniższy przewodnik jest wynikiem pracy trzech studentów w ramach projektu zaliczeniowego z przedmiotu związanego z cyfrowym przetwarzaniem obrazów. Mamy nadzieję, że zagadnienia w nim przedstawione, zachęcą Cię do dalszego rozwoju w tym kierunku :)

1 Wstęp

1.1 Dlaczego Python?

Wybraliśmy ten język, ponieważ obecnie jest to jeden z najpopularniejszych języków programowania. Dodatkowo ma prostą składnię i należy do grupy języków wysoiego poziomu – czyli stosujemy w nim szereg gotowych funkcji, które ułatwiają pracę! Dzięki temu powstało wiele bibliotek dedykowanych różnym problemom/zadaniom, a Python stał się bardzo wszechstronnym narzędziem.

I śmieszy nas Monty Python (to od tej grupy komików wywodzi się nazwa języka) – ale spokojnie, w poradniku skupiamy się na rozwiązywaniu powszechnych zadań związanych z przetwarzaniem obrazów cyfrowych, także zapraszamy do lektury, nawet jeśli masz zupełnie inne poczucie humoru.

1.2 Python 2.x a 3.x

1.2.1 Ramy czasowe

Python 2.x jest z nami od początku XXI wieku. Wprowadził takie cechy jak list comprehension (`[str(x) for x in range(10)]`), jednak jego wsparcie zostanie porzucone od 2020 roku. Oznacza to, że poza nielicznymi przypadkami (rozwijanie starych projektów, których konwersja do 3.x może okazać się zbyt ryzykowna i/lub pracochłonna) nie warto rozważać, czy wybrać Pythona 2.x czy 3.x.

Python 3. został wydany pod koniec 2008 roku. Obecnie wykorzystuje się jego wersje 3.6 (wspacie 99% bibliotek) oraz 3.7 (ostatnia stabilna wersja, trwają pracę nad wsparciem brakujących bibliotek).

Z tych względów w tym poradniku wykorzystywać będziemy wersję 3.6.

1.2.2 Różnice

Jeśli do tej pory korzystałeś tylko z wersji 2.x, polecamy zapoznać się z artykułem: https://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html, gdzie opisane zostały główne różnice.

1.3 Podstawowe komendy

Poniżej znajdziesz przykładowe pętle. Zauważ, że bloki kodu nie są wydzielone żadnym widocznym znakiem czy słowem (pętle nie kończą się klamrą, nawiasem lub "endem") – w Pythonie wnętrze pętli (lub funkcji) zaznaczamy poprzez odpowiednie wcięcie (tabulator). Dzięki temu kod jest przejrzysty i łatwy do odczytania. Jeśli jednak po zakończeniu pętli zapomnisz wrócić kursorem do początku linii, kolejne instrukcje będą wykonywane jako dalsza część danego bloku, co może dać nieoczekiwane efekty! Lepiej o tym pamiętać.

- **wyrażenia warunkowe**

```
In [1]: python_is_easy = True # boolean True lub False
        if python_is_easy:
            print('Python is easy!')
        else:
            print('Python is hard :(')
```

```
Python is easy!
```

- **pętla while**

```
In [2]: string = r'W pythonie możemy używać pojedynczych lub podwójnych apostrofów do znaków bąd
        slowa = string.split(' ') # ta metoda dzieli nam ciąg znaków/słów wg podanego kryterium,
        licznik = 0

        while licznik<len(slowa):
            print(slowa[licznik])
            licznik+=1
```

```
W
pythonie
możemy
używać
pojedynczych
lub
podwójnych
apostrofów
do
znaków
bądź
słów
```

WAŻNE! W tym języku indeksujemy od 0.

- **pętla for (wraz z uruchamianiem paczek)**

```
In [3]: import tqdm # ta paczka pozwoli nam na wygenerowanie paska postępu (przydatne przy długi

In [4]: import urllib # pozwoli nam na przeczytanie pliku tekstowego z internetu

In [5]: with urllib.request.urlopen('http://files.catwell.info/misc/mirror/zen-of-python.txt') a
            for line in tqdm.tqdm(txt):
                print(line.decode("utf-8"))

0it [00:00, ?it/s]

The Zen of Python, by Tim Peters



Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.
```

```
If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.



20it [00:00, 199.99it/s]

Namespaces are one honking great idea -- let's do more of those!



21it [00:00, 184.20it/s]
```

Jest to znane Zem of Python. Możemy też uzyskać je poprzez użycie komendy *import this*.

Jeśli korzystasz z Jupyter Notebook komendy możemy wywoływać również z poziomu terminala/shell – należy po prostu poprzedzić komendę wykrzyknikiem. Kolejne polecenia możemy rozdzielać przecinkiem.

```
In [6]: ! python -c "import antigravity"
```

Wykorzystaliśmy tutaj tzw. *list comprehension*, które umożliwia uzyskanie listy z elementami przetworzonymi lub przefiltrowanymi wybranym przez nas kryterium. Wykorzystywane często przy prostych operacjach. Działają też dla obiektow tuple (krotka) lub dictionary (słownik).

1.4 Instalacja bibliotek

Jak wspomnieliśmy, Python posiada wiele bibliotek dedykowanych do różnych celów. Aby móc z nich skorzystać, należy je najpierw aktywować poleceniem import . Część podstawowych modułów jest wbudowana, inne trzeba zainstalować. Istnieje wiele sposobów, by to zrobić, oto przykładowe:

- pojedynczo z poziomu konsoli

```
In [ ]: ! pip install pandas
```

- "hurtowo" wykorzystując plik requirements.txt, który jest najprostszą metodą przechowywania niezbędnych paczek dla danego projektu (tutaj widoczne są biblioteki wykorzystywane przez nas)

```
In [7]: with open('../requirements.txt','r') as f:
            for req in f.readlines():
                print(req)
```

```
jupyter

numpy
```

```
pillow

matplotlib

tqdm

opencv-python
```

```
In [ ]: ! pip install -r ../requirements.txt
```

Resztę potrzebnych komend poznasz w kolejnych rozdziałach poradnika. To jak, zaczynamy?