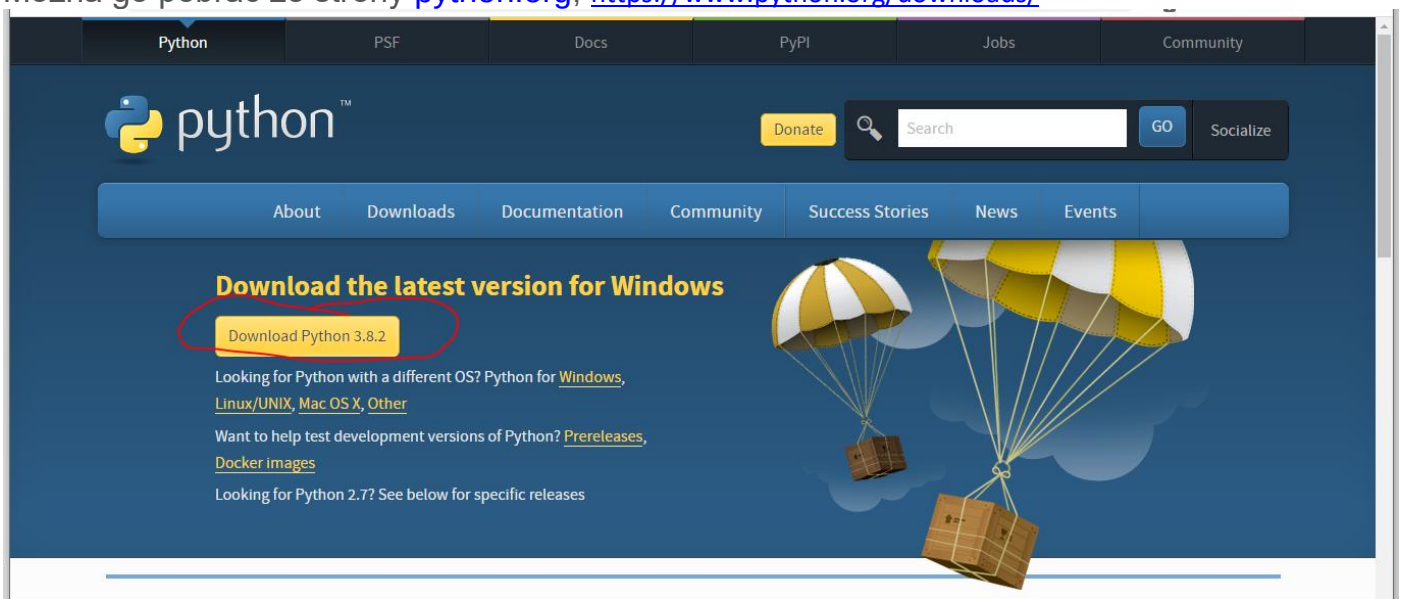
Instalacja środowiska

Żeby w ogóle zacząć swoją przygodę z Pythonem należy zainstalować środowisko. Można go pobrać ze strony python.org, https://www.python.org/downloads/



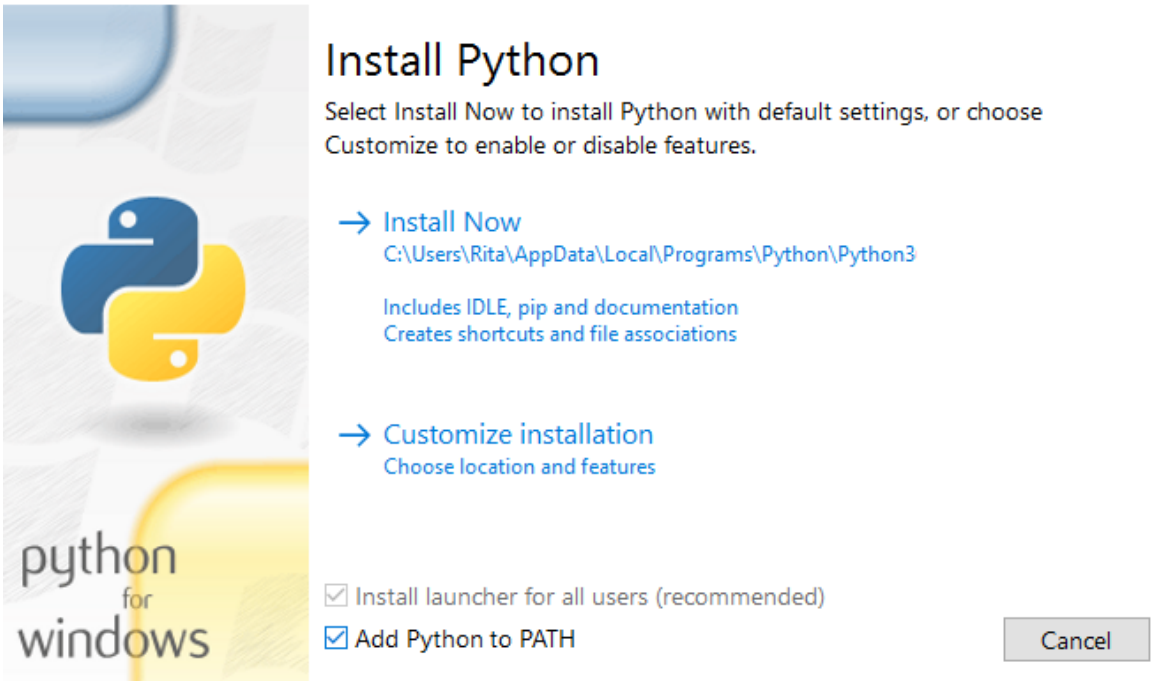
Ponieważ to Python 3.x jest przyszłością kurs jest w oparciu o aktualnie najnowszą wersję 3.8.

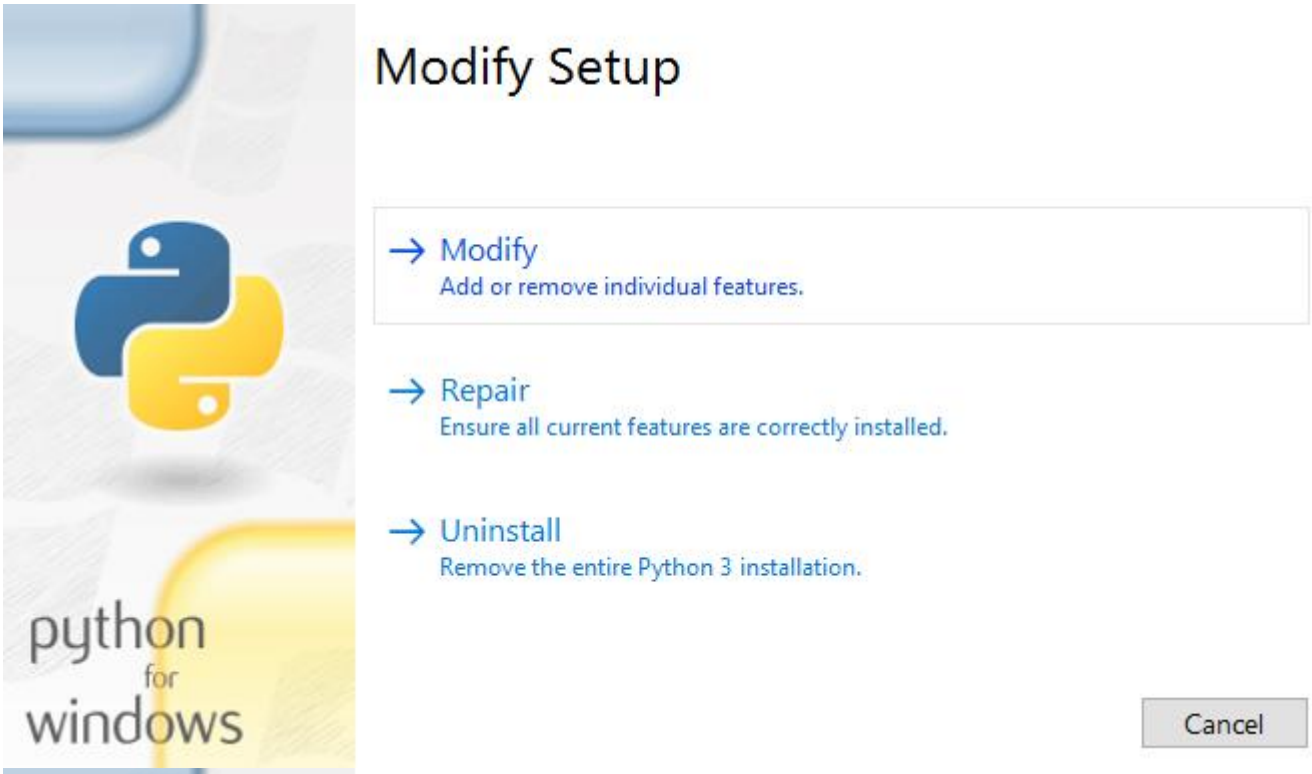
Windows

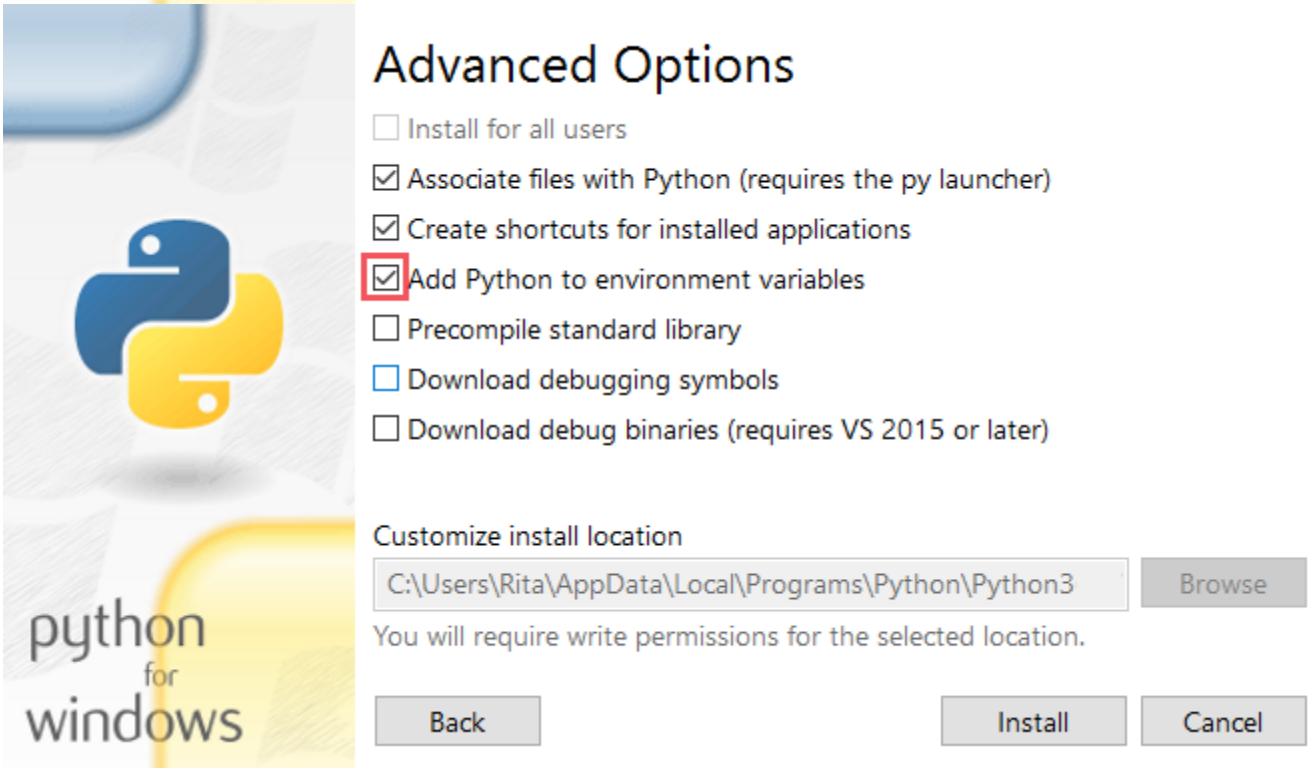
Zalecam instalowanie na ustawieniach domyślnych tzn. nie bawić się w zmianę ścieżki zapisu.

Instalując pamiętajcie zaznaczyć opcję Add Python 3.8 to path. U mnie wygląda to tak:

W razie czego, można tę opcję ustawić w Panelu Sterowania > Programy i funkcje > Python3.8, wybieramy Modify, Optional features zostawiamy bez zmian, a w Advanced dodajemy Pythona do zmiennych systemowych jak poniżej.




Modify Setup
Modify
Add or remove individual features.
Repair
Ensure all current features are correctly installed.
Uninstall
Remove the entire Python 3 installation.
python
for
windows
Cancel


Advanced Options
Install for all users
Associate files with Python (requires the py launcher)
Create shortcuts for installed applications
Add Python to environment variables
Precompile standard library
Download debugging symbols
Download debug binaries (requires VS 2015 or later)
Customize install location
C:\Users\Rita\AppData\Local\Programs\Python\Python3
Browse
You will require write permissions for the selected location.
Back
Install
Cancel
python
for
windows

Linux i Mac

W tych systemach Python już jest. Wersję jaką macie zainstalowaną można sprawdzić wklejając do terminala

```
python3.8 --version
Python 3.8.0
```

uruchamiamy konsole komendą:

```
python3.8
```

Jeśli pojawi się nieaktualna wersja wystarczy ją zaktualizować.

Mac OS X
Na systemach Appla jest domyślnie zainstalowana okrojona wersja Pythona, można przy niej zostać lub zainstalować ręcznie. Na stronie python.org wybierz wersję systemu i pobierz pakiet.

Ubuntu

```
sudo apt –y update

sudo apt –y install python3.8
```

Fedora

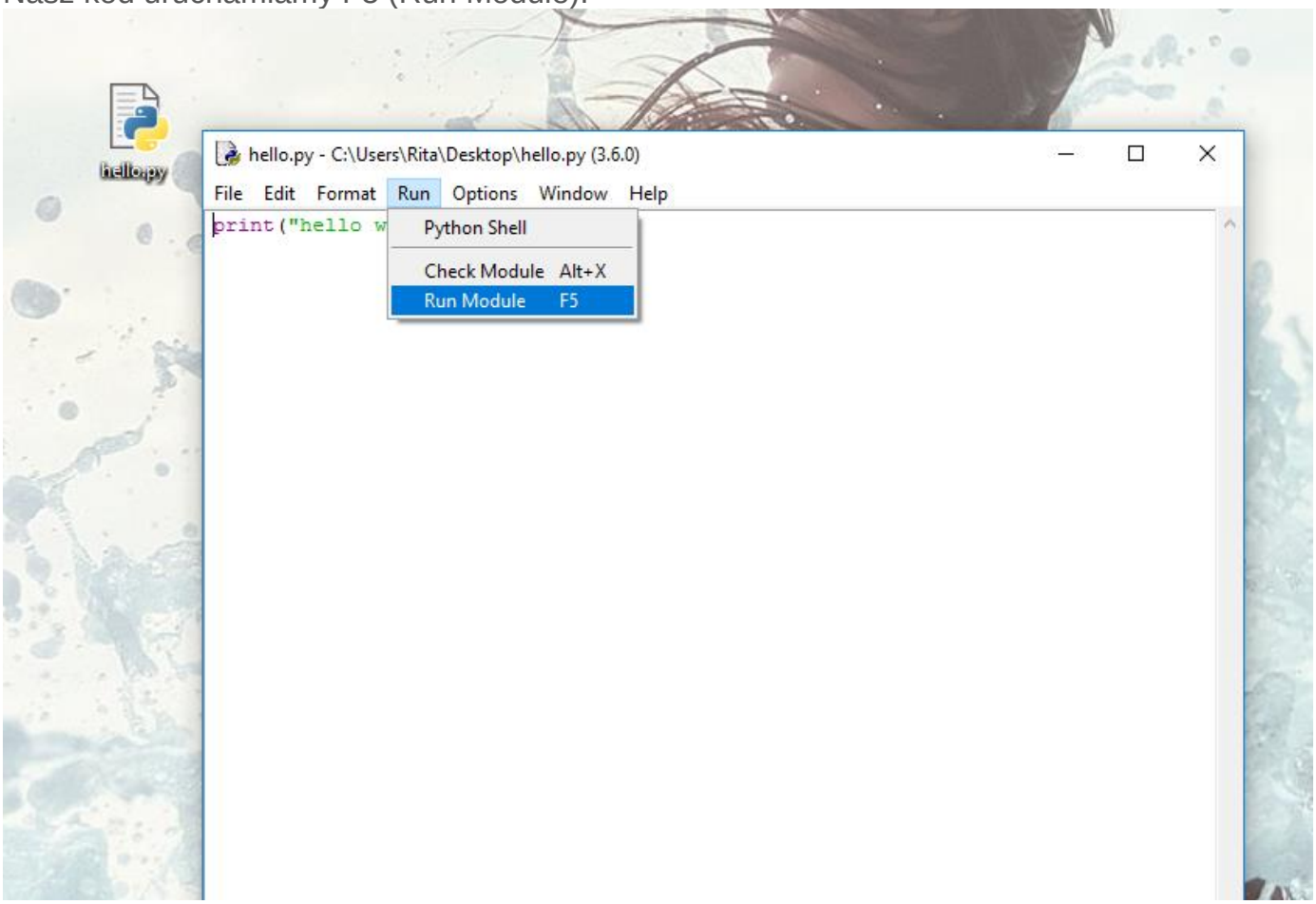
```
sudo yum check-update

sudo yum install python3.8
```

Pierwszy skrypt

W dowolnym miejscu (u mnie na pulpicie) tworzymy plik z rozszerzeniem .py – u mnie hello.py. Edytować go możemy dowolnym edytorem nawet notatnikiem, ale po kliknięciu na niego prawym przyciskiem pojawi się Edit in IDLE – jest to prosty edytor i jednocześnie środowisko. W treść wklejamy słynne „Hello World!” i wyświetlamy.

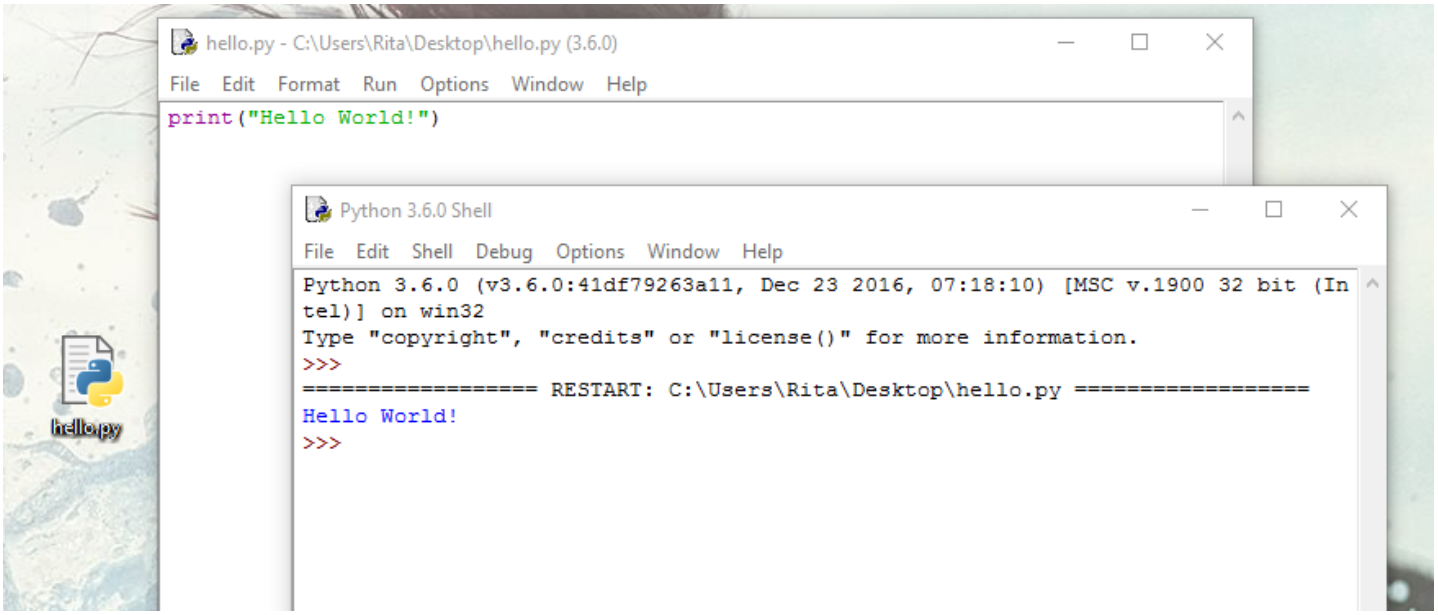
```
print("hello world!")
```

Nasz kod uruchamiamy F5 (Run Module).



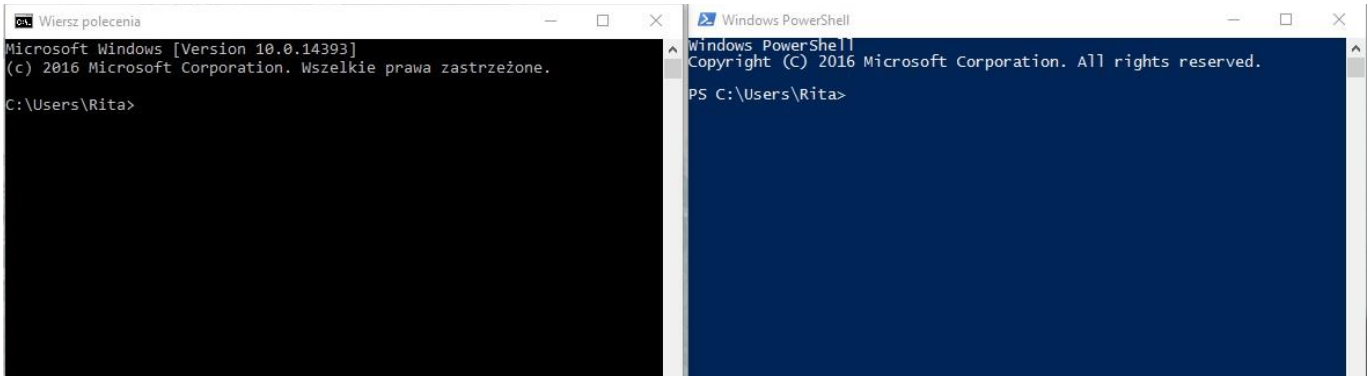
Python #1: Zabawy w konsoli

- skrypt – plik tekstowy o rozszerzeniu **.py** zawierający polecenia do wykonania
- *ang. shell* – jest interpreterem skryptów, a także interaktywną powłoką przez którą możemy (i zaraz będziemy) bezpośrednio wydawać polecenia Pythonowi.



Dalszą część dzisiejszych zadań możesz wykonywać właśnie w tym pythonowym shellu, który otworzył się po uruchomieniu pliku, ale nie po to dodawaliśmy przy instalacji Pythona do zmiennych systemowych

Windows:
Masz dwie opcje używać Wiersza poleceń (wyszukaj: cmd) lub PowerShell – zachęcam do używania tej drugiej opcji nie tylko ze względu na kolor. Na pierwszy rzut oka różnica jest niewielka, dla tego kursu Pythona też nie, ale dla osób, które w ogóle będą chciały kiedyś korzystać z konsoli Powershell może okazać się dużo lepszym rozwiązaniem.



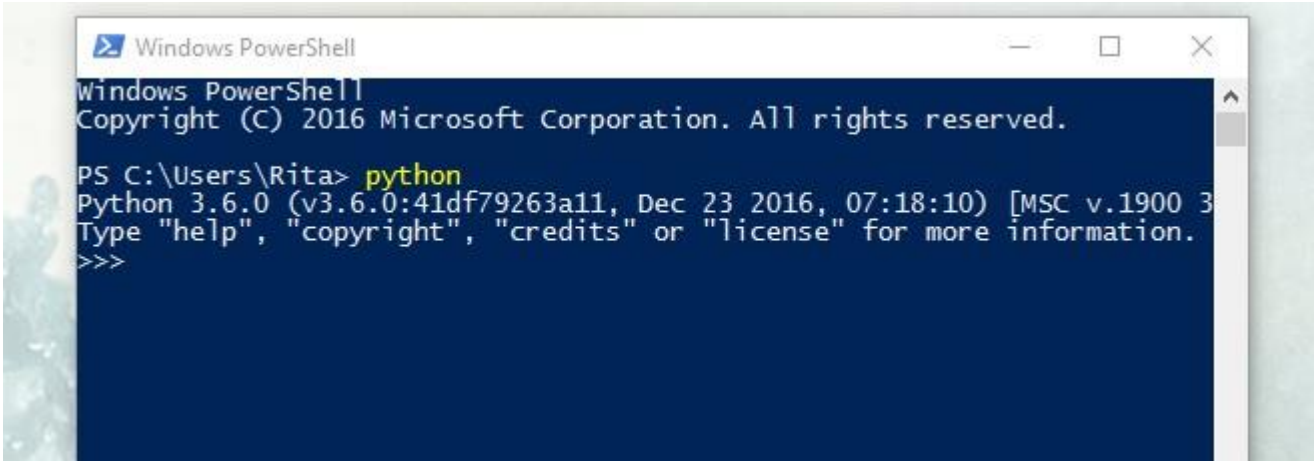
Linux i Mac OS X
Wyszukaj i uruchom Terminal.

Od tej chwili gdy piszę **konsola** lub **shell** mam na myśli w zależności od opcji, którą wybierasz: Wiersz poleceń/Powershell lub Terminal.

Twój Python w konsoli

W konsolę wpisz i kliknij enter:

```
Python
Python 3.6.0 (…)
Type „help”, „copyright”, „credits” or „license” for more information.>>>
```



Uruchomi się interpreter, który wypisał informacje o zainstalowanej wersji i prawach autorskich, a także znak zachęty(**>>>**).

Teraz możemy np. coś policzyć np. 3+3:

```
>>> 3 + 3
6
>>>
```

Możemy użyć Pythona jak kalkulator:

```
>>> 3*7
21
>>> 500 - 43 * 5
285
>>> (1 - 7) * 7
-42
>>> 8 * 3.5
28.0
>>> 8 * 3.25
26.0
>>> 4**2
16
>>> 8 / 5
1.6
>>> 8 % 5
3
>>>
```

Zwróć uwagę, że liczby dziesiętne zapisujemy z użyciem kropki, a nie przecinka jak w języku polskim. W momencie, gdy mnożymy przez liczbę dziesiętną Python domyślnie zaokrągla nam do pierwszego miejsca „po przecinku” (a dokładniej po kropce), mimo że wynik jest całkowity.

Operatory warte zapamiętania:

- ** – znak potęgowania
- / – znak dzielenia, w językach programowania nie używa się, znanego ze szkoły dwukropka ” : „
- % – znak dzielenia modulo, czyli reszta z dzielenia

Tak poznaliśmy pierwszy rodzaj zmiennych – zmienne liczbowe oraz ich dwa typy – liczby całkowite(int) i liczby zmiennoprzecinkowe (float).

Napisy

Kolejną ważną zmienną są string’i czyli napisy. Definiujemy je za pomocą apostrofów(,) lub cudzysłowów(„). Inaczej *Hello World* nie zadziała:

```
>>> Hello World
(…)
SyntaxError: invalid syntax
```

Python zwrócił błąd. Prawidłowo:

```
>>> "Hello World"
Hello World
```

Co się stanie jeśli…

```
>>> "napis"
,napis’
>>> 'napis'
,napis’
>>> "to jest 'napis'"
„to jest ,napis'”
>>> 'i to tez jest "napis"'
,i to tez jest „napis”,
```

…to bez sensu.
To spróbuj zrobić jeszcze kilka przykładów

```
>>> 'doesn't'
(…)
SyntaxError: invalid syntax
>>> "doesn't"
„doesn’t”
>>> 'doesn\'t'
„doesn’t”
```

Każdy **string musi zaczynać się i kończyć tym samym znakiem** – nie ważne czy użyjemy apostrofów (,…’) czy cudzysłowów („…”).

Pierwszy przykład zwraca **błąd** – ponieważ znak apostrofu dla Pythona kończy string w miejscu ,doesn’ a dalsza część jest poza i interpreter nie bardzo wie co z nią zrobić.

W drugim, wykorzystujemy **cudzysłów**, dzięki czemu apostrof w środku nie jest traktowany jako znak specjalny.

Natomiast w ostatnim, za pomocą ukośnika „\” zaznaczamy, że , jest **znakiem specjalnym** i w ten sposób możemy użyć apostrofu wewnątrz dwóch apostrofów oznaczających stringa – ufff… mam nadzieję, że wyjaśniłam to jasno.

Mówiąc językiem programistów, a mniej po polsku: escap’ujemy za pomocą backslash’a i będziemy to robić z wszystkimi znakami specjalnymi.

Co jeszcze możemy zrobić ze stringami?
Ciągi znaków możemy dodawać (inaczej łączyć, inaczej konkatenować) czy mnożyć:

```
>>> 'Kto to jest?'+'To Adam'
,Kto to jest?To Adam’
>>> 'Kto to jest?'+'"To Adam" odpowiedziała Magda'
,Kto to jest?”To Adam” odpowiedziała Magda’
>>> 'herbata' * 3
,herbataherbataherbata’
>>> 'herbata' + 3
(…)
TypeError: must be str, not int
```

O! Mamy nowy błąd!
Do tej pory spotkaliśmy **SyntaxError** – czyli błąd w składni.
Teraz dodanie do string’a (str) liczby (int) powoduje pojawienie się błędu w typie **TypeError**.
Rozwiązać możemy to na dwa sposoby:

```
>>> 'herbata' + "3"
,herbata3’
>>> 'herbata' + str(3)
,herbata3’
```

Jak widzisz możesz połączyć dwa napisy nawet jeśli jedne zapisujesz apostrofami, a drugie cudzysłowem. W drugim rozwiązaniu wykonano **konwersję typu int do string** przy pomocy funkcji **str()**.

Wyświetlanie napisów

Polecenie wyświetlania już znasz **print()**.

```
>>> print("Hello World!")
Hello World!
```

Możesz połączyć dwa lub więcej napisów w jednym poleceniu oddzielając je przecinkiem, a **spacja między nimi doda się automatycznie**:

```
>>> print("Witaj", "co u Ciebie?")
Witaj co u Ciebie?
```

print() pozwala na wyświetlanie różnych typów, nie tylko napisów.

```
>>> print(3)
3
>>> print(3, 6, 8)
3, 6, 8
>>> print("Ania ma:", 3, "lata")
Ania ma: 3 lata
```

Natomiast wstawienie samego print() wstawi znak nowej linii czyli da efekt entera:

```
>>> print()
>>>
```

Znaki specjalne
Gdy chcieliśmy w napisie rozpoczętym apostrofem użyć apostrofu w środku napisu, a nie jako jego zakończenie, musieliśmy poprzedzić go znakiem „\”. Oto kilka innych znaków specjalnych, które mogą się przydać:

Znak specjalny	Znaczenie
\n	znak nowej linii, dodanie „entera”
\t	dodanie tabulacji
\’	apostrof
\”	cudzysłów
\\	ukośnik

```
>>> print("To jest\nnowe")
To jest
nowe
```

Wypróbuj je wszystkie!

Przypisywanie wartości do zmiennej

Zmienne mogą mieć *prawie* dowolne nazwy i przechowywać dowolne wartości, a także typy. Przypiszmy zmiennej a wartość 5, zmiennej b wartość 10, a do c przypiszmy efekt mnożenia tych zmiennych i zobaczmy efekt:

```
>>> a = 5
>>> b = 10
>>> c = a * b
>>> c
50
```

Zadanie 1 – kalkulator BMI
Jesteś w stanie napisać to samodzielnie, w razie czego przykładowe rozwiązanie znajduje się poniżej

Po pierwsze przyda się wzór na BMI:

$$\mathrm{BMI} = \frac{\mathrm{waga}}{\mathrm{wzrost}^2}$$

Już na pierwszy rzut oka widzimy, że będą nam potrzebne tylko 3 zmienne – wzrost, waga i BMI, do którego przypiszemy wynik.

np.
wzrost = 1.60 – w metrach
waga = 53 – w kilogramach

```
>>> wzrost = 1.60
>>> waga = 53
>>> BMI = waga / (wzrost ** 2)
>>> BMI
20.703124999999996
>>> print("Twoje bmi wynosi:",BMI)
Twoje bmi wynosi: 20.703124999999996
```

Nie było trudne prawda?

W takim razie spróbuj samodzielnie całość napisać w jednej linii – wszystko wewnątrz print().

Zadanie 2 – dla chętnych
Oblicz z pomocą pythona zapotrzebowanie kaloryczne:

a) dla 25-letniej kobiety o wzroście 1.7m i wadze 63kg, która uprawia sport kilka razy w tygodniu.
b) siebie samej / samego.

Podaję wzory poniżej:

Podstawa: **10 x masa ciała + 6.25 x wzrost w cm – 5 x wiek + S**
współczynnik S: dla kobiet = -161, dla mężczyzn= +5

Podstawę, czyli liczbę kalorii zużywaną na same procesy życiowe należy pomnożyć przez rodzaj aktywności fizycznej:

Praca siedząca, brak dodatkowej aktywności fizycznej	1,2
Praca nie fizyczna, mało aktywny tryb życia	1,4
Lekka praca fizyczna, regularne ćwiczenia 3-4 razy (ok. 5h) w tygodniu	1,6
Praca fizyczna, regularne ćwiczenia od 5 razy (ok. 7h) w tygodniu	1,8
Praca fizyczna ciężka, regularne ćwiczenia 7 razy w tygodniu	2,0