



Faculty of Geodesy
and Cartography

WARSAW UNIVERSITY OF TECHNOLOGY

INFORMATYKA GEODEZYJNA - WYKŁADY/ĆWICZENIA, ROK AKAD. 2021-2022

WYK. 1: PYTHON - WPROWADZENIE

Kinga Węzka

kinga.wezka@pw.edu.pl

Katedra Geodezji i Astronomii Geodezyjnej



1. Dlaczego należy znać język programowania?

- Algorytm – forma zapisu i realizacja

2. Charakterystyka języków programowania

- Poziomy wykonania programu: wysokiego i niskiego poziomu
- Sposób wykonania programu: komplowane i interpretowane
- Paradygmaty języków programowania

3. Dlaczego Python? – podstawowe cechy

- Wykonywanie programu: kompilacja czy interpreter?
- Paradygmaty
- Typowanie zmiennych i zarządzanie pamięcią
- Struktura, podstawowe konstrukcje i składnia Pythona
- Rozbudowane pakiety bibliotek

4. Ćwiczenia: instalacja i uruchomienie

- Instalacja kompilatora/interpretera
- Platforma Anaconda
- Anaconda Prompt: Menadżery pakietów - conda/pip

5. Ćwiczenia: pierwszy program, struktura, uruchomienie

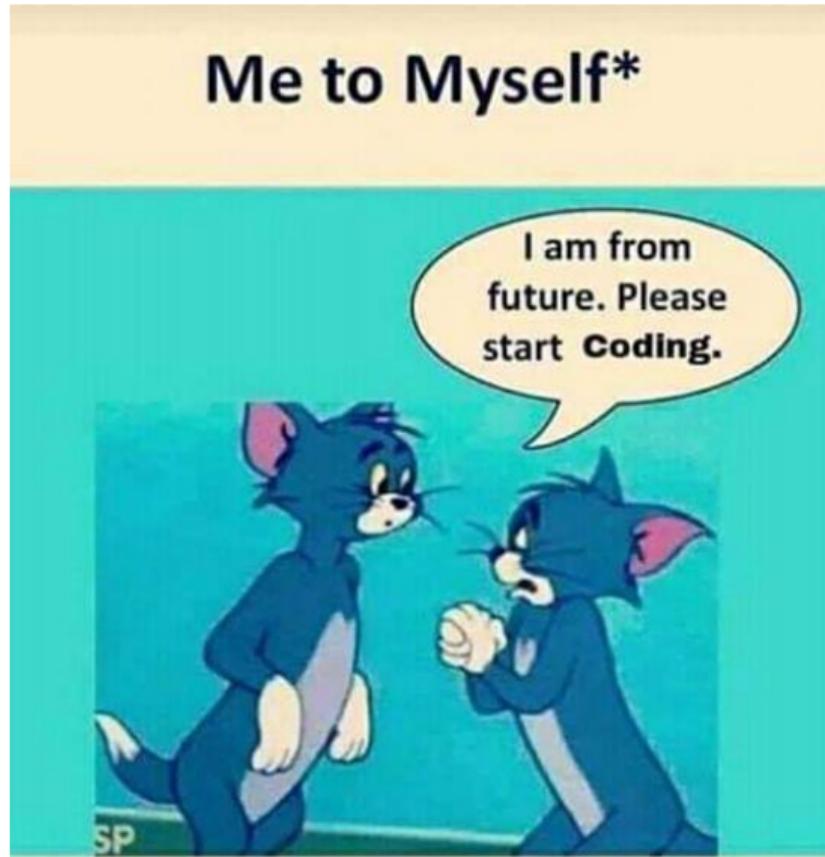
- Struktura pliku
- Importowanie i pomoc dla pakietów/modułów
- Instrukcje: operacje wyjście i wejście danych `print`/`input`



Inżynier oprócz tego, że wie jak wykonać zadanie, to potrafi je również wykonać!



Język programowania jest podstawowym narzędziem do realizacji obliczeń inżynierskich i nie tylko.

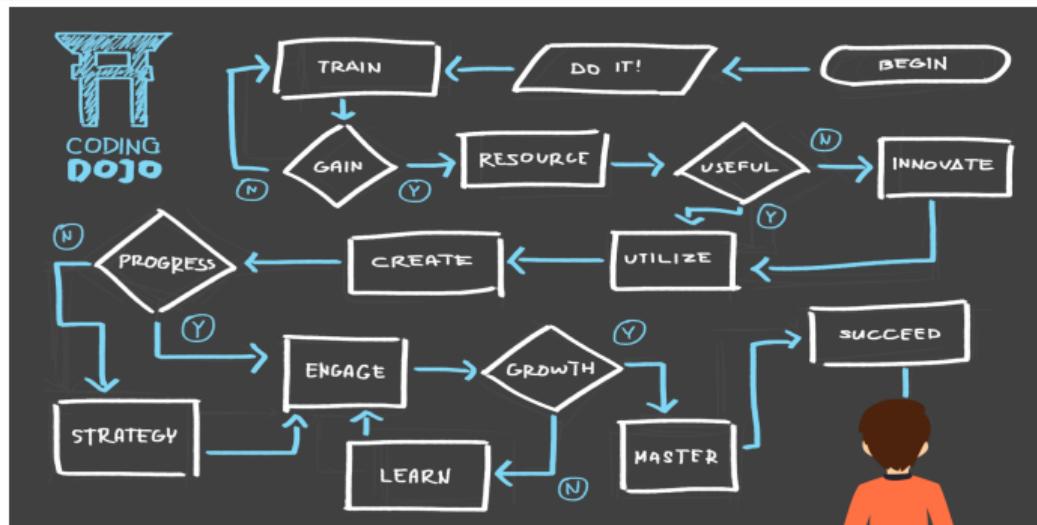




Od czego zacząć pisanie programów?

Aby napisać program, który będzie realizował rozwiązywanie zadania:

- programista powinien znać elementy składniowe wybranego języka programowania;
- programista powinien znać **sposób rozwiązywania problemu**, ten sposób określa **algorytm**



<https://www.flynerd.pl/2018/11/jak-sie-uczyc-algorytmow-i-myslenia-algorytmicznego.html>



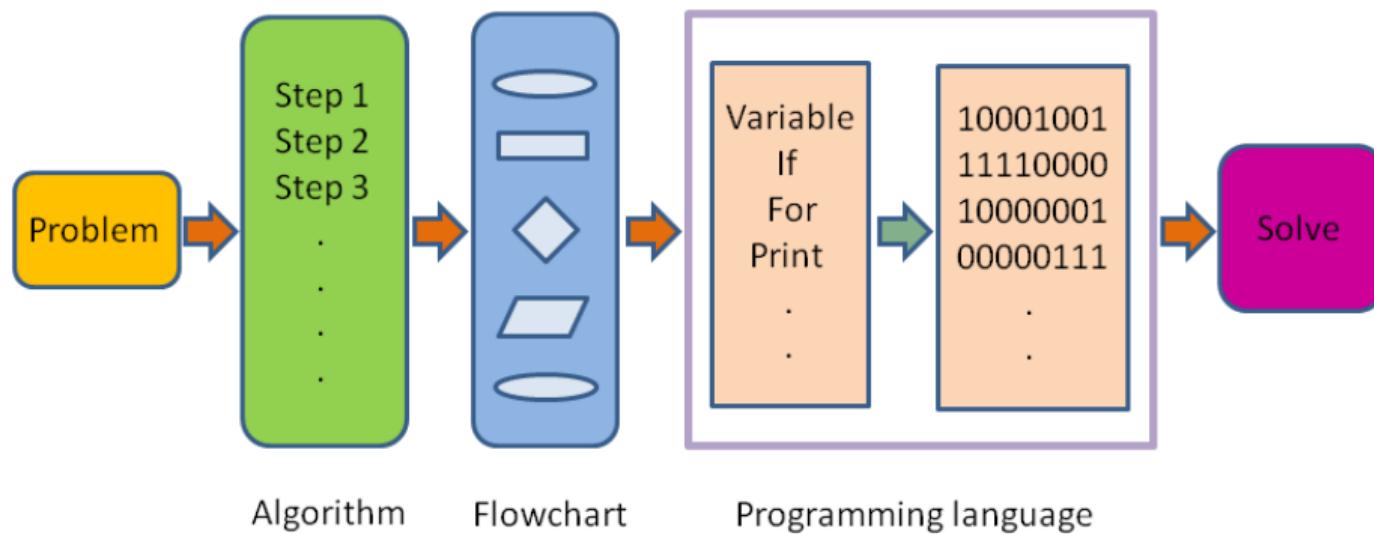
Algorytm

Algorytm określa dane oraz skończony ciąg operacji, jakie należy na tych danych wykonać, aby rozwiązać dowolny problem z określonej dziedziny.

- Poprawny – dla każdego poprawnego zestawu informacji wejściowych musi prowadzić do poprawnych rezultatów.
- Jednoznaczny – każdorazowo, dla każdego poprawnego zestawu informacji wejściowych, powinien prowadzić do tych samych rezultatów.
- Szczegółowy – aby ktoś albo coś wykonujący algorytm rozumiał dokładnie opisane operacje i potrafił je wykonać.
- Uniwersalny – pozwalał na rozwiązywanie dowolnego problemu z określonej klasy, a nie dotyczy pewnych przypadków.



- **Język naturalny** (potoczny) określający ponumerowany ciąg kroków algorytmu.
- Notacje graficzne, najpopularniejsze są **schematy blokowe** (ang.*flowchart*).
- **Pseudokod** – połączenie języka naturalnego i notacji matematycznej z elementami języków programowania.
- Zapis w postaci **kodu programu** w pewnym języku programowania.





- **Język naturalny** (potoczny) określający ponumerowany ciąg kroków algorytmu.
- Notacje graficzne, najpopularniejsze są **schematy blokowe** (ang.*flowchart*).
- **Pseudokod** – połączenie języka naturalnego i notacji matematycznej z elementami języków programowania.
- Zapis w postaci **kodu programu** w pewnym języku programowania.

SCHEMAT BLOKOWY	PSEUDOKOD	IMPLEMENTACJA W PYTHONIE
<pre> graph TD A([licznik mianownik]) --> D{Czy mianownik jest równy zero?} D -- Tak --> STOP((STOP)) D -- Nie --> B([Wykonaj dzielenie]) B --> D </pre>	<p>DANE:</p> <p>licznik mianownik</p> <p>Jeśli mianownik jest równy zero zatrzymaj program</p> <p>W pozostałych przypadkach: wynik = licznik/mianownik</p>	<pre> licznik = 8 mianownik = 4 if mianownik == 0: pass else: wynik = licznik \ mianownik </pre>

Elementy schematu blokowego: <https://en.wikipedia.org/wiki/Flowchart>



Język programowania

- Język programowania jest formalnym językiem, który składa się z zestawu instrukcji, które generują różnego rodzaju wyniki. Innymi słowy, język programowania pozwala na precyzyjny zapis algorytmów oraz innych zadań, jakie komputer ma wykonać.
- Podobnie jak języki naturalne, język programowania składa się ze zbiorów reguł **syntaktycznych** oraz **semantyki**, które opisują, jak należy budować poprawne wyrażenia oraz jak komputer ma je rozumieć (źródło: Wiki).
 - **Syntaktyka** (składnia/syntax) - zestaw reguł (wyrażenia regularne, symbole, struktury, słowa kluczowe itd.) wymaganych aby dany ciąg znaków był rozpoznawalny jako program.
 - **Semantyka** - określa precyzyjnie znaczenie poszczególnych symboli oraz ich funkcję.

Klasyfikacja języków programowania

- Poziomy wykonania programu: **wysokiego i niskiego poziomu**
- Sposób wykonania programu: **kompilowane i interpretowane**
- Paradygmaty języków programowania: **proceduralne, obiektowe, funkcyjne**



■ Języki wysokiego poziomu (ang. *high-level*):

```
class Triangle {  
    ...  
    float surface()  
        return b*h/2;  
}
```

■ Języki poziomu Assembler (ang. *assembly-level*)

```
LOAD r1,b  
LOAD r2,h  
MUL r1,r2  
DIV r1,#2  
RET
```

■ Języki niskiego poziomu (ang. *low-level*), Języki maszynowe (ang. *machine-level*)

```
0001001001000101001001  
00110110010101101001.  
..
```

■ Sprzęt (ang. *hardware*)

■ Języki wysokiego poziomu (np. C, C++, Java, Python, ...) potrzebują kompilatora lub interpretera do "komunikacji z procesorem"

■ Języki niskiego poziomu są bliskie językowi maszyny (procesora) mają bezpośredni dostęp do rejestru i pamięci.

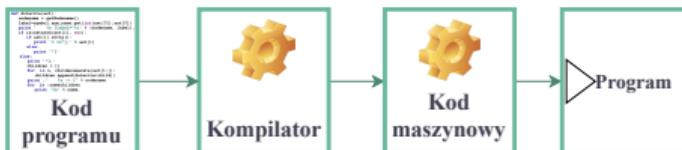
- **Assembly** język **mnemonicznym** używa **assembler** do tłumaczenia mnemonicznego kodu na kod specyficznej maszyny.

- **Język Maszynowy** jest złożony z cyfr binarnych (zer i jedynek).

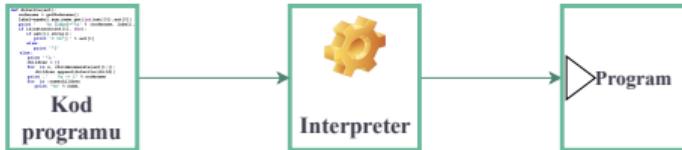




- Programy **języków kompilowanych**, przed uruchomieniem muszą zostać skompilowane do **kodu maszynowego**. **Kompilacja to tłumaczenie kodu programu na język maszynowy**, który może być wykonany przez komputer. Kompilatory zazwyczaj tworzą kod maszynowy, działający tylko na określonej architekturze systemu (np. procesory x86).



- Programy **języków interpretowanych**, są wykonywane przez interpreter linijka po linijce, nie muszą być skompilowane. Programista nie musi podczas każdej edycji kodu kompilować programu, co w pracach gdzie modyfikacje kodu są częste (**prototypowanie**) jest ogromną zaletą. **Najcenniejszy zasób - czas programisty**.



- Niektóre języki (np. Java/Python) kompilują kod źródłowy do kodu pośredniego zwanego **kodem bajtowym**, który jest wykonywany przez **maszynę wirtualną**.



Paradygmat to przyjęty sposób widzenia rzeczywistości w danej dziedzinie, doktrynie itp. lub zespół form fleksyjnych (deklinacyjnych lub koniugacyjnych), właściwy danemu typowi wyrazów; wzorzec, model deklinacyjny lub koniugacyjny

■ Jak to się ma do języków programowania?

Paradygmat języka programowania to nie tylko wzorzec pisania programu, jest to przede wszystkim zbiór mechanizmów, jakich programista używa pisząc program, sposobu w jaki program jest wykonywany przez komputer. Do najczęściej spotykanych paradygmatów języków programowania należy programowanie proceduralne, funkcyjne i obiektowe.



Najbardziej popularne i najczęściej wykorzystywane paradymaty

- **Programowanie imperatywne** – paradymat programowania, który opisuje proces wykonywania jako sekwencję instrukcji zmieniających stan programu. **Przykłady: C, C++, Java, Python**
- **Programowanie deklaratywne** – rodzina paradymatów programowania, w których programista zamiast definiowania sposobu rozwiązania, czyli sekwencji kroków prowadzących do uzyskania wyniku, opisuje samo rozwiązanie – to co nas interesuje (brak opisu przepływu sterowania).
Przykłady: Ocaml, XSLT, Prolog, SQL.



- **Programowanie proceduralne (strukturalne, imperatywne)** (np. C, Pascal, Python), opiera się na podziale kodu źródłowego programu na procedury i hierarchicznie ułożone bloki (begin ... end) z wykorzystaniem struktur kontrolnych w postaci instrukcji wyboru i pętli. Program postrzegany jest jako ciąg poleceń dla komputera.
- **Programowanie funkcyjne (deklaratywne)** (np. Haskell, Python), Filozofia i metodyka programowania w której funkcje należą do wartości podstawowych, a nacisk kładzie się na wartościowanie funkcji (rekurencja/składanie funkcji), a nie na wykonywanie poleceń. Program to po prostu złożona funkcja (w sensie matematycznym), która otrzymawszy dane wejściowe wylicza pewien wynik.
- **Programowanie obiektowe** (np. Python, Java, C++). Należy wyobrazić sobie, że obiekt to zbiór danych liczbowych i tekstowych oraz tzw. metod, które operują na tych wewnętrznych danych. Te dane i metody są ze sobą ściśle powiązane, co ułatwia pracę nad programem. Program to zbiór porozumiewających się ze sobą obiektów, czyli jednostek zawierających pewne dane i umiejących wykonywać na nich pewne operacje. Więcej w temacie OOP na kolejnych zajęciach.



Why Python?

Figure: popularities > opportunities > simplicity



Popularność języków programowania

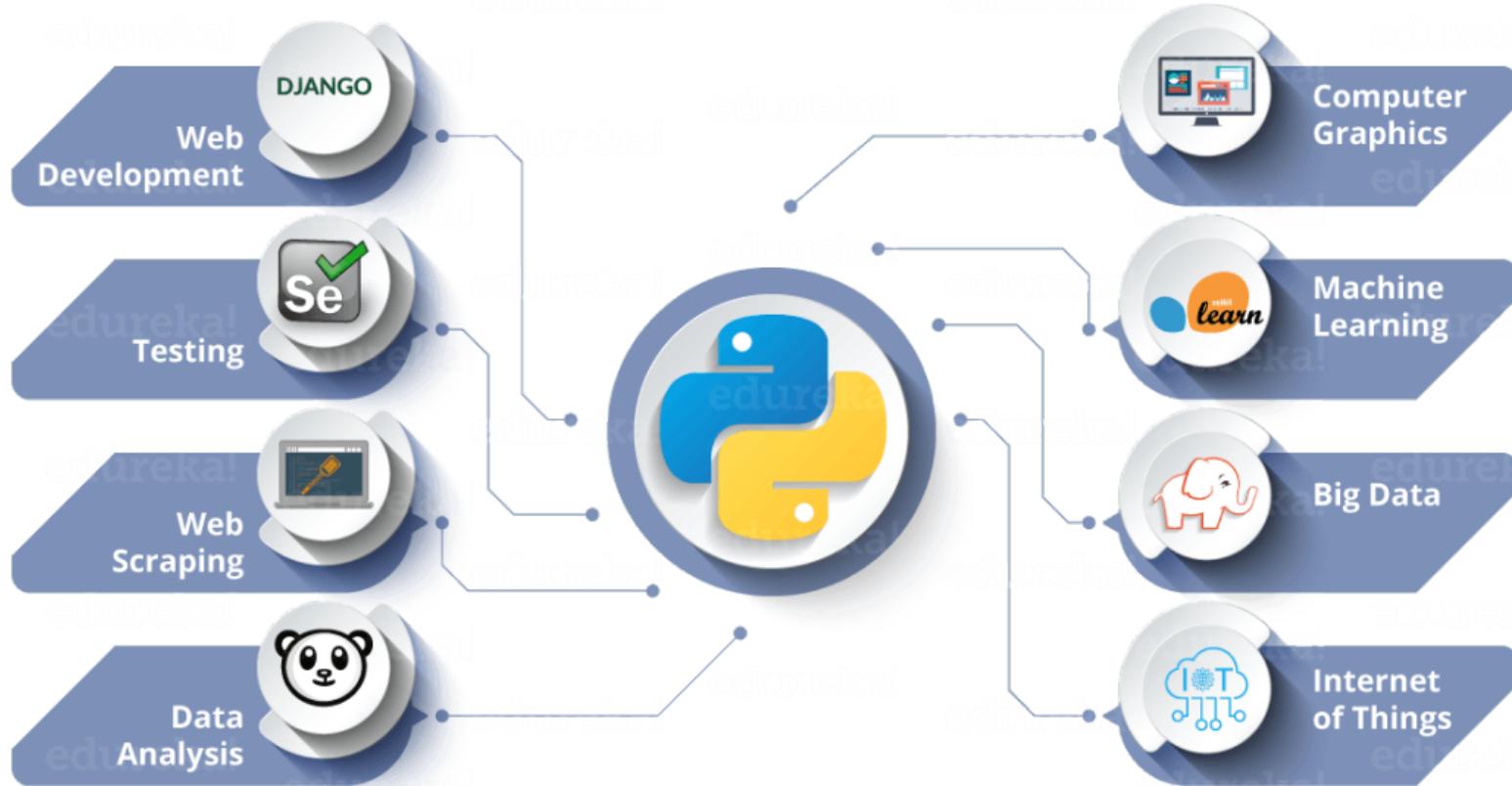
PYPL ang. *Popularity of Programming Language Index* jest tworzony na podstawie analizy częstotliwości wyszukiwania w Google pomocy i materiałów do języków programowania.

Worldwide, Oct 2020 compared to a year ago:				
Rank	Change	Language	Share	Trend
1		Python	31.02 %	+2.2 %
2		Java	16.38 %	-2.8 %
3		JavaScript	8.41 %	+0.4 %
4		C#	6.52 %	-0.6 %
5		PHP	5.83 %	-0.4 %
6		C/C++	5.56 %	-0.4 %
7		R	4.26 %	+0.4 %
8		Objective-C	3.48 %	+0.8 %
9		Swift	2.37 %	-0.1 %

Źródło: <http://pypl.github.io/PYPL.html>



edureka!



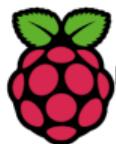


Google



mozilla
Firefox®

IBM



RaspberryPi

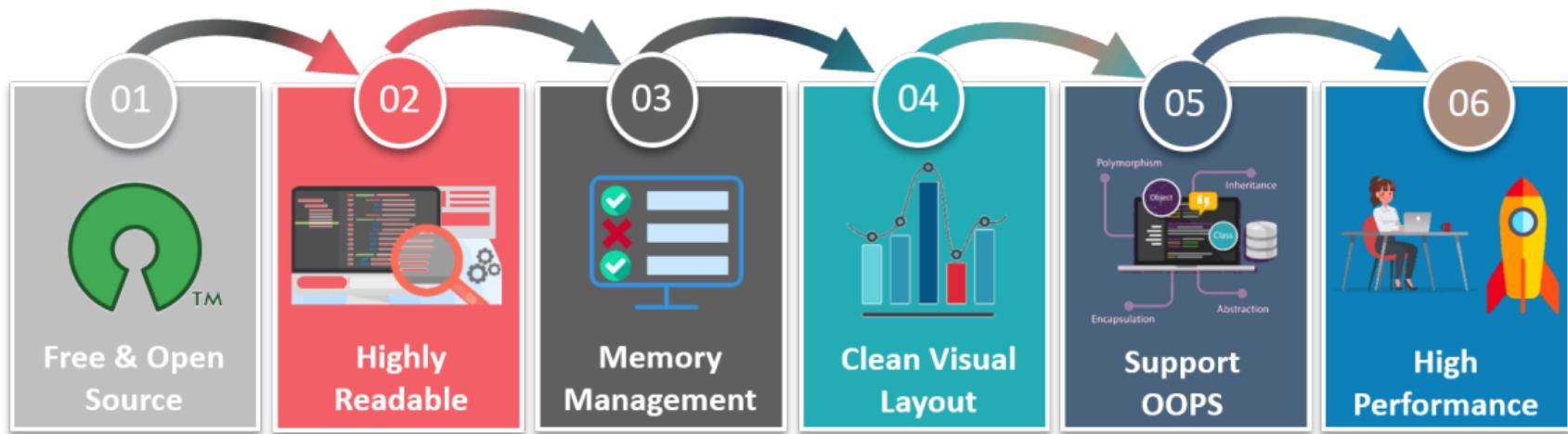


YouTube



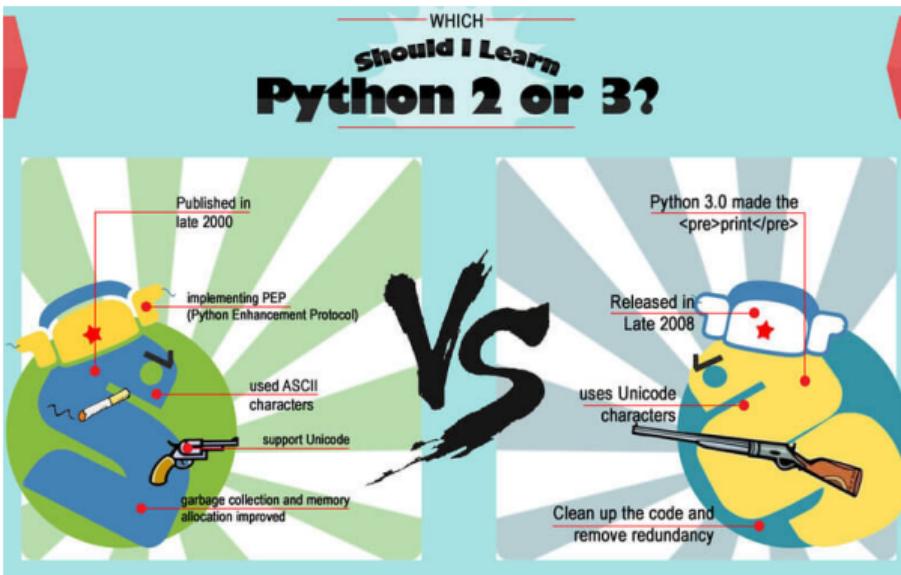
NETFLIX

Figure: “Python where we can, C++ where we must.” - Google

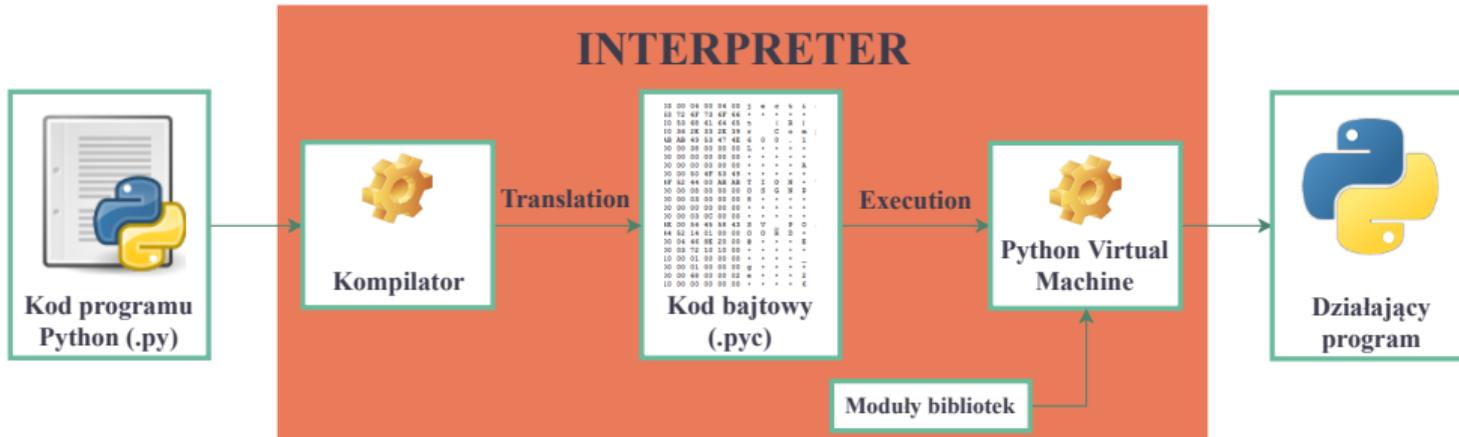




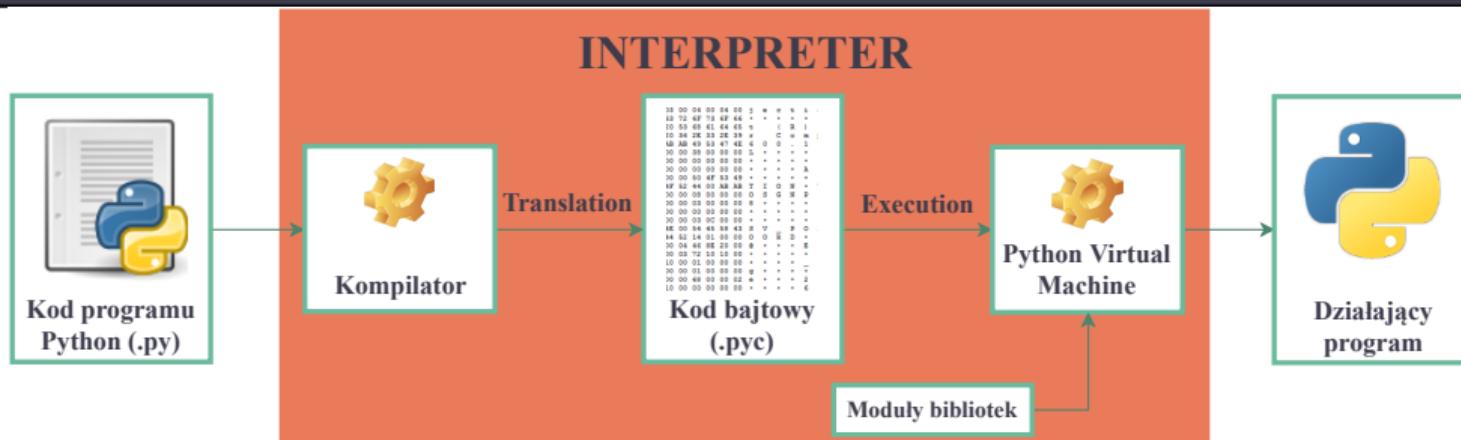
- Stworzony we wczesnych latach 90-tych przez **Guido van Rossum** jako następcę języka ABC (Rossum pracował w Centrum Matematyki i Informatyki w Amsterdamie, następnie w latach 2005 - 2013 pracował dla Google, a obecnie pracuje dla firmy Dropbox)
- Nazwa pochodzi od serialu BBC: "Monty Python's Flying Circus" (Latający cyrk Monty Pythona).
- Python jest rozwijany jako projekt **Open Source** zarządzany przez **Python Software Foundation**, która jest organizacją non-profit. <https://www.python.org/psf/>
- Standardową implementacją języka jest CPython (napisany w C), ale istnieją też inne, np. PyPy, Jython (napisany w Java), CLPython napisany w Common Lisp, IronPython (na platformę .NET) i PyPy napisany w Pythonie.



- Python 2.x (najnowsza wersja 2.7) - mnogość dostępnych bibliotek i programów.
- **Python 3.x** (najnowsza wersja 3.7) - nie uwzględnia kompatybilność w dół.
- Różnice pomiędzy 2.x i 3.x
<http://www.w3big.com/pl/python/python-2x-3x.html>

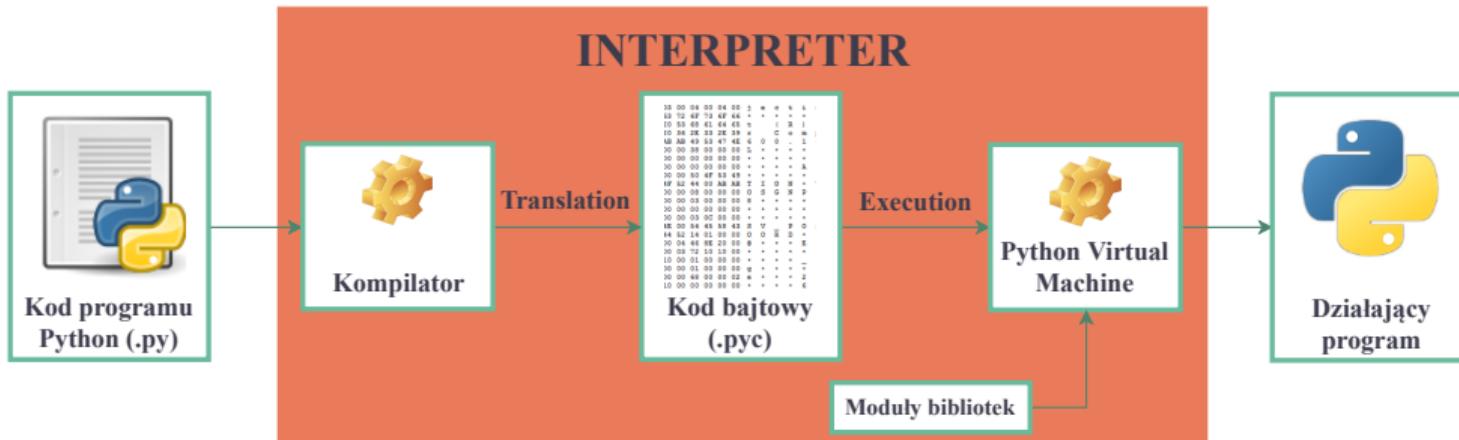


- Krok 1. Python najpierw kompiluje kod źródłowy (.py) do formatu zwanego **kodem bajtowym** (ang. *bytecode*) zapisywanego w plikach typu (.pyc). Proces kompilacji jest procesem translacji, natomiast **kod bajtowy** jest niezależnym od platformy systemowej kodem źródłowym niskiego poziomu. Program Pythona może być dystrybuowany w obu formatach plików: .py oraz .pyc.

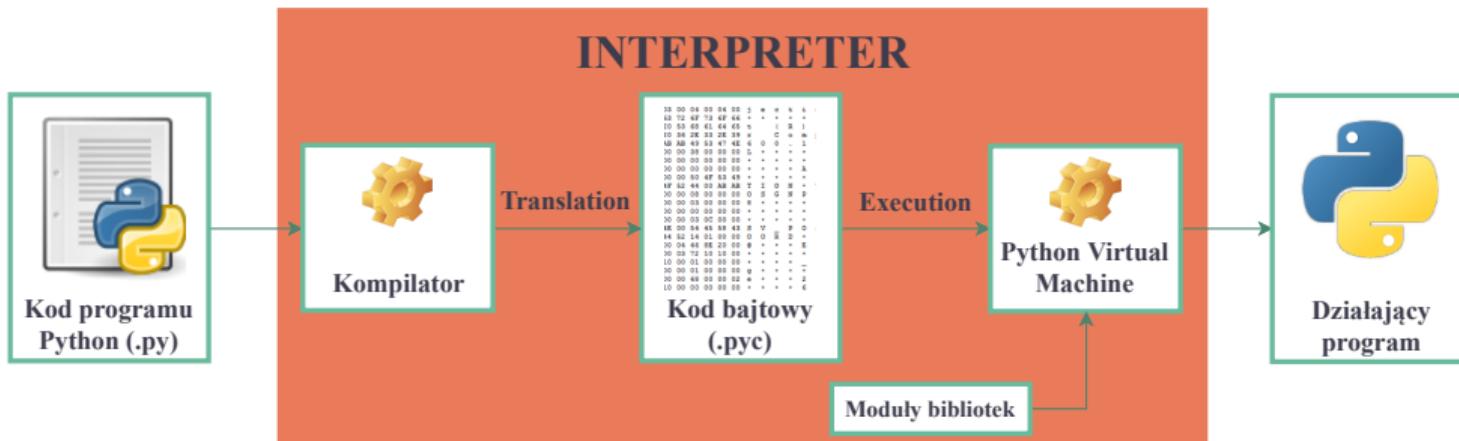


■ Krok 2. Pliki z kodem bajtowym (**.pyc**) są ładowane do środowiska wykonawczego **Python** i interpretowane przez **maszynę wirtualną Python** (ang. **Python Virtual Machine**) – silnik wykonawczy (ang. *runtime engine*) Pythona, który interpretuje nasz kod i tłumaczy go do kodu bajtowego, aby program mógł być zrozumiany przez maszynę (procesor systemu).

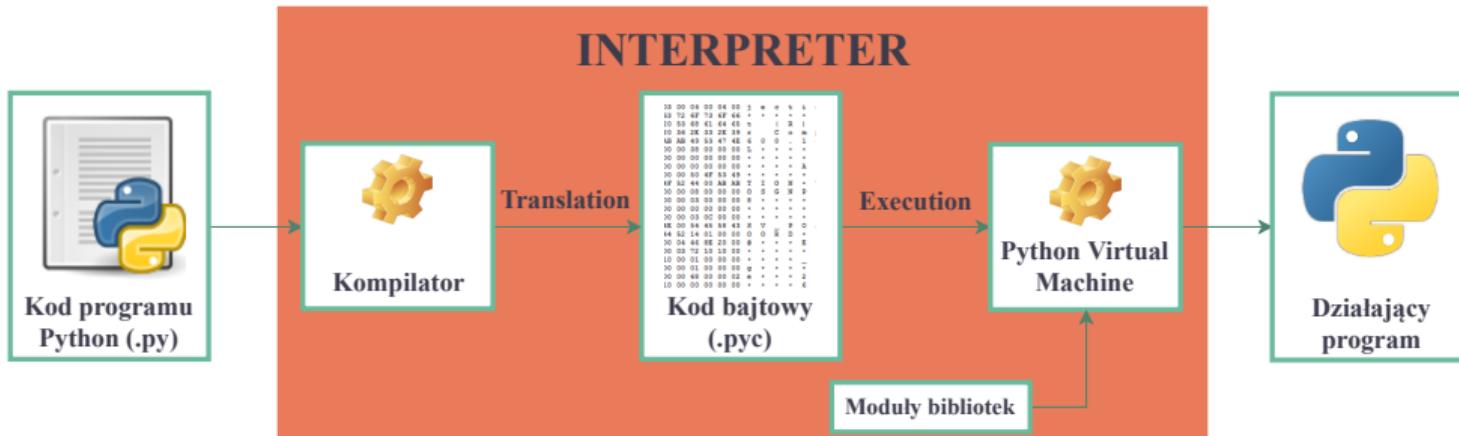
PVM to fragment kodu bajtowego, reprezentuje każdą instrukcję w kodzie bajtowym i wykonuje wskazane operacje. Kompilacja do kodu bajtowego odbywa się automatycznie, a PVM jest zawsze częścią systemu Pythona zainstalowanego na komputerze.



- Za każdym razem, gdy uruchamiany jest interpretowany program, interpreter musi konwertować kod źródłowy na kod maszynowy, a także pobierać biblioteki środowiska wykonawczego. Ten proces konwersji powoduje, że program działa wolniej niż porównywalny program napisany w skompilowanym języku.
- **Python aby poprawić swoją wydajność - robi coś sprytnego - komplikuje się do kodu bajtowego (pliki .pyc) przy pierwszym uruchomieniu pliku.**, poprawia to znacznie wykonanie kodu przy następnym importie lub wykonywaniu modułu.



- **Python nie jest typowym programem interpretowanym** poprzez komplikację do pośrednich plików bajtowych zwiększa swoją wydajność - wykonanie kodu bajtowego jest znacznie szybsze niż interpretacja kodu źródłowego.
- Kod bajtowy Pythona nie jest typowym binarnym kodem maszynowym (na przykład w postaci instrukcji dla chipa firmy Intel). To pętla maszyny wirtualnej Pythona, a nie chip procesora, musi zinterpretować kod bajtowy, a instrukcje kodu bajtowego wymagają więcej pracy od instrukcji procesora ([Lutz, 2011, p.69](#))



- Omawiana na zajęciach implementacja jest standardową i oryginalną implementacją Pythona jest **CPython**, której wiele modułów wywołuje procedury kodu C. Pozostałe implementacje mają swoje ściśle określone cele i role.
- Dostępne implementacje Python, implementują ten sam język, ale różnią się sposobem wykonania programu, na przykład:
 - **Jython** – interpreter – tworzenie w Pythonie skryptów dla aplikacji napisanych w Java.
 - **Cython** – kompilator – rozszerzenie C dla Pythona, szybkość wykonania porównywalna do C.
https://cython.readthedocs.io/en/latest/src/tutorial/cython_tutorial.html



- Python jest językiem **wielo-paradygmatowym** (ang. *multi-paradigm*)
- Podstawowym paradygmatem jest obiektowość, od samych fundamentów **Python jest językiem zorientowanym obiektywem**. **Wszystko w Pythonie jest obiektem!**. Jego model klas obsługuje zaawansowane koncepcje, takie jak: polimorfizm, przeciążanie operatorów, dziedziczenie wielokrotne ([Lutz, 2011](#)) (szczegóły na kolejnych zajęciach).
- Python jest również **językiem proceduralnym**.
- Python jest również **językiem funkcyjnym**

PROCEDURALNY



FUNKCYJNY



OBIEKTOWY





- Python posiada opcję **dynamicznego typowania zmiennych** (ang. *dynamically typed*) - to przypisywanie typów do wartości przechowywanych w zmiennych w trakcie działania programu. Typy zmiennych sprawdzane są dynamicznie w czasie wykonywania (ang. *runtime*). Nie trzeba deklarować typu przed przypisaniem jej wartości.
- Przeciwieństwem do typowania dynamicznego jest **typowanie statyczne** (ang. *static typing*) które oznacza konieczność nadawania typów zmiennym w czasie komilacji programu również w czasie komilacji zgłasza się błędy typów.

Statyczne

```
string name      - deklaracja zmiennej - określenie typu  
name = "Janek"  - inicjalizacja zmiennej - typ zgodny z deklaracją  
name = 22     - zmieni nie może zmienić typu!
```

Dynamiczne

```
name = 23.37    - deklaracja i inicjalizacja zmiennej  
name = "Janek" - wartość ma typ  
name = 22       - zmieni zmienia typ dynamicznie!
```

Figure: Różnica pomiędzy dynamicznym a statycznym typowaniem zmiennych, dla zmiennej **name**

Języki **dynamiczne** i **dynamiczne typowanie** nie są tożsamymi pojęciami, a dynamiczny język programowania nie musi zawsze posiadać mechanizmu dynamicznej zmiany typów.



- Co więcej Python wykorzystuje technikę **kaczego typowania** (ang.*Duck typing*) – rozpoznawanie typu obiektu nie na podstawie deklaracji, ale przez badanie metod udostępnionych przez obiekt. Technika ta wywodzi się z powiedzenia: „jeśli chodzi jak kaczka i kwacze jak kaczka, to musi być kaczką”.
- Kacze typowanie to koncepcja, związana z typowaniem dynamicznym, w której typ lub klasa obiektu jest mniej ważna niż metody, które określa. Używając kaczego typowania, wcale nie sprawdzamy typów, zamiast tego sprawdzamy obecności danej metody lub atrybutu.



Typowanie dynamiczne i kacze typowanie na ogół występują razem. Jednak możliwe jest stosowanie dynamicznego typowania bez kaczego typowania - podejście bardzo rzadko wykorzystywane.



- Python wykorzystuje **automatyczne czyszczenie pamięci** (ang. **garbage collection**) – to metoda automatycznego zarządzania dynamicznie przydzielaną pamięcią, w której za proces jej zwalniania odpowiedzialny jest nie programista, lecz programowy zarządca noszący nazwę **garbage collector**



- Można skorzystać z ręcznego czyszczenia pamięci co nieznacznie przyśpiesza działanie programu, ale wymaga modyfikacji kodu. **Ręczne usuwanie pamięci** (biblioteka: `gc`)



Wyrażenia (ang. expression)

3 + 5

```
map(lambda x: x*x, range(10))  
[a.x for a in some_iterable]
```

Instrukcje (ang. statement)

```
import math  
def add(a,b):  
    return(a+b)
```

- **Wyrażenie** (ang. *expression*) - wyrażenia zawierają tylko identyfikatory, literały i operatory, w których operatory obejmują operatory arytmetyczne i logiczne, operator wywołania funkcji () operator subskrypcji [] i podobne, i mogą być zredukowane do pewnego rodzaju „wartości”, którą może być dowolny obiekt Pythona. Przykłady wyrażeń w Pythonie: : **None**, **True**, **False**, **3*(7 + 7)**

- **Instrukcja** (ang. *statement*) - najmniejszy samodzielny element imperatywnego języka programowania. Instrukcja może zawierać wewnętrzne komponenty (np. wyrażenia). Instrukcje Pythona: instrukcja przypisania, instrukcja wyrażeniowa, **print** (w python. 3 jest to funkcja), **if**, **while**, **for**, **pass**, **break**, **continue**, **del**, **def**, **return**, **yield**, **global**, **nonlocal**, **import**, **from**, **class**, **try**, **raise**, **assert**, **with**.



Funkcje (ang. functions)

■ **Funkcja** w Pythonie to zbiór instrukcji pogrupowanych pod nazwą. Można jej użyć, gdy chcemy wykonać wszystkie polecenia równocześnie. Funkcję można wywoływać w dowolnym miejscu i dowolną liczbę razy. Funkcja może przyjmować argumenty i zwraca wyniki działania.

Klasy (ang. classes)

■ **Klasa** jest typem abstrakcyjnym. Innymi słowy, jest to plan dla określonego rodzaju obiektu, nie zawiera żadnych wartości. Dopiero **obiekt klasy** jest bytem świata rzeczywistego i konkretyzacją, instancją klasy.

Moduły (ang. modules)

■ **Moduł** zawiera zbiór powiązanych klas i funkcji. Plik z kodem Pythona (*.py) zawierającym funkcje, klasy itp jest modułem.

Pakiety (ang. packages)

■ **Pakiet** to zbiór powiązanych modułów. Można importować pakiet z dostępnych repozytoriów lub tworzyć własne.



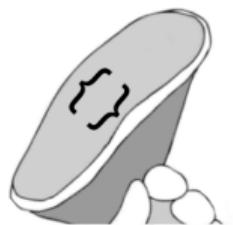
Programy napisane w Pythonie można rozbić na moduły, instrukcje, wyrażenia i obiekty – w następujący sposób:

- Programy składają się z modułów.
- Moduły zawierają instrukcje.
- Instrukcje zawierają wyrażenia.
- Wyrażenia tworzą i przetwarzają obiekty (Lutz, 2011, p.119).



- W pewnym sensie Python jest językiem typu **WYSIWYG** (ang. *what you see is what you get*) — to, co widzimy, jest tym, co otrzymujemy, ponieważ kod wykonywany jest tak, jak wygląda, bez względu na jego autora.
- Python do wyodrębniania bloków kodu stosuje wcięcia (wcięcie = tabulator lub 4 spacje)
– blok kodu zaczyna się po instrukcji z dwukropkiem np. `class:`, `def(x):`, `if x:`

Ew, I stepped in shit.



■ Silnia w C (zapisana bez wcięć):

```
1 int silnia(int x) {  
2     if (x == 0) return 1;  
3     else return x * silnia(x-1);}
```

■ Silnia w Pythonie:

```
1 def silnia(x):  
2     if x == 0:  
3         return 1  
4     else:  
5         return x * silnia(x-1)
```



- Wyrównanie kodu zgodnie z jego strukturą logiczną jest podstawowym narzędziem uczynienia go czytelnym i tym samym łatwym w ponownym użyciu oraz późniejszym utrzymywaniu — zarówno przez nas samych, jak i przez inne osoby ([Lutz, 2011](#), p.306)
- Każdy język programowania wymaga odpowiedniego wyrównania kodu. W Pythonie wyrównanie bloków kodu za pomocą indentacji jest częścią składni i jest sprawdzane przez kompilator/interpreter. Dzięki temu Python pozbywa się innych dodatkowych ograniczników bloków kodu (np. begin ... end, {...} itp.)

blok_0

wiersz_nagłówka_instrukcji:

blok_1

wiersz_nagłówka_instrukcji:

blok_2

blok_1

blok_0

```
1 x = 1
2 if x:
3     y = 2
4     if y:
5         print('blok_2')
6         print('blok_1')
7 print('blok_0')
```



- Python automatycznie wykrywa granice bloków dzięki **indentacji** wierszy – czyli pustej przestrzeni po lewej stronie kodu. Wszystkie instrukcje wcięte na tę samą odległość w prawą stronę należą do tego samego bloku kodu.
- **Instrukcje jednego bloku są ze sobą wyrównane w pionie, jak kolumna.** Blok kończy się na końcu pliku lub po napotkaniu mniej wciętego wiersza. Bloki głębiej zagnieżdżone są po prostu wcinane na większą odległość w prawą stronę w stosunku do instrukcji z bloku je zawierającego.

blok_0

wiersz_nagłówka_instrukcji:

blok_1

wiersz_nagłówka_instrukcji:

blok_2

blok_1

blok_0

```
1 x = 1
2 if x:
3     y = 2
4     if y:
5         print('blok_2')
6         print('blok_1')
7 print('blok_0')
```



- Pakiet **bibliotek wbudowanych** (ang. *Built-in Library*). Zestaw funkcji i typów wbudowanych rozpowszechniany jest razem z kompilatorem Pythona. **Nie wymagają instalacji, nie ma konieczności importu do programu.**

- *Python Built-in Functions:*

<https://docs.python.org/3/library/functions.html>

- *Python Built-in Types:*

<https://docs.python.org/3/library/stdtypes.html>

- Pakiet **bibliotek standardowych** (ang. *Standard Library*). Zestaw bibliotek standardowych rozpowszechniany jest razem z kompilatorem Pythona. **Nie wymagają instalacji, konieczny jest import do programu.**

- <https://docs.python.org/3/library/>

- **Rozbudowane repozytoria bibliotek:** Python Package Index (<https://pypi.org/>); Anaconda (repo.anaconda.com). **Wymagają instalacji, konieczny jest import do programu.**

▪ <code>completer</code> — Completion function for GNU readline
▪ <code>struct</code> — Interpret bytes as packed binary data
▪ <code>codecs</code> — Codec registry and base classes
▪ <code>Data Types</code>
▪ <code>datetime</code> — Basic date and time types
▪ <code>calendar</code> — General calendar-related functions
▪ <code>collections</code> — Container datatypes
▪ <code>collections.abc</code> — Abstract Base Classes for Containers
▪ <code>functools</code> — Higher-order functions
▪ <code>bisect</code> — Array insertion algorithm
▪ <code>array</code> — Efficient arrays of numeric values
▪ <code>weakref</code> — Weak references
▪ <code>types</code> — Dynamic type creation and names for built-in types
▪ <code>copy</code> — Shallow and deep copy operations
▪ <code>pformat</code> — Data pretty print
▪ <code>reprlib</code> — Alternate repr() implementation
▪ <code>enum</code> — Support for enumerations
▪ <code>Numeric and Mathematical Modules</code>
▪ <code>math</code> — Mathematical functions
▪ <code>cmath</code> — Mathematical functions for complex numbers
▪ <code>decimal</code> — Decimal fixed point and floating point arithmetic
▪ <code>fractions</code> — Rational numbers
▪ <code>random</code> — Generate pseudo-random numbers
▪ <code>statistics</code> — Mathematical statistics functions
▪ <code>Functional Programming Modules</code>
▪ <code>itertools</code> — Functions creating iterators for efficient looping
▪ <code>functools</code> — Higher-order functions and operations on callable objects
▪ <code>operator</code> — Standard operators as functions
▪ <code>File and Directory Access</code>
▪ <code>pathlib</code> — Object-oriented filesystem paths
▪ <code>os.path</code> — Common pathname manipulations
▪ <code>fileinput</code> — iterate over lines from multiple input streams
▪ <code>stat</code> — Interpreting stat() results
▪ <code>filecmp</code> — File and Directory Comparisons
▪ <code>tempfile</code> — Generate temporary files and directories
▪ <code>glob</code> — Unix-style pathname expansion
▪ <code>fnmatch</code> — Unix filename pattern matching
▪ <code>timecache</code> — Random access to test lines
▪ <code>shutil</code> — High-level file operations
▪ <code>macpath</code> — Mac OS 9 path manipulation functions

Figure: Pakiet bibliotek standardowych



Plik instalacyjny **Pythona** można pobrać ze strony głównej Python Software Foundation :
<https://www.python.org/downloads/>

The screenshot shows the Python Software Foundation's website homepage. The main navigation bar includes links for About, Downloads, Documentation, Community, Success Stories, News, and Events. A prominent yellow button labeled "Download the latest source release" is visible. Below it, there are links for "Download Python 3.7.4", "Python for Windows, Linux/UNIX, Mac OS X, Other", "Pre-releases, Docker images", and "Python 2.7". To the right, there is a graphic of two boxes descending from the sky on yellow and white striped parachutes. A search bar and a "Socialize" button are also present at the top.

Looking for a specific release?

Python releases by version number:

Release version	Release date	Click for more
Python 3.7.4	July 8, 2019	Download Release Notes
Python 3.6.9	July 2, 2019	Download Release Notes
Python 3.7.3	March 25, 2019	Download Release Notes



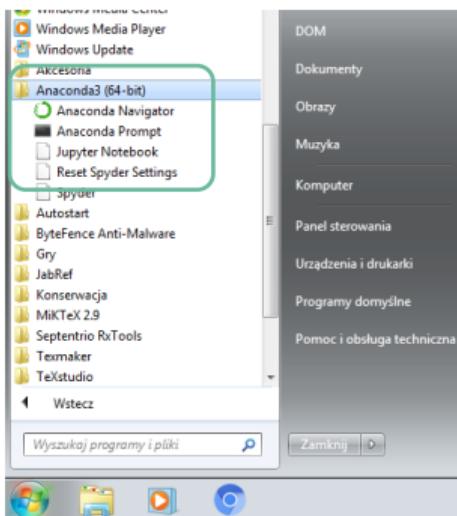
Anaconda jest rozbudowaną, darmową platformą zawierającą kompilator Pythona i kilka innych narzędzi ułatwiających pracę. Zintegrowanymi komponentami platformy są: środowisko IDE (Spyder), konsola shell Pythona, menadżer zarządzania bibliotekami (conda), możliwość definiowania wirtualnych środowisk pracy, pre-instalowane biblioteki, jupyter notebook.



Pobranie wersji zgodnej z systemem użytkownika: <https://www.anaconda.com/download>

Strona główna: <https://www.anaconda.com>

Dokumentacja Anaconda: <https://docs.anaconda.com/anaconda/>



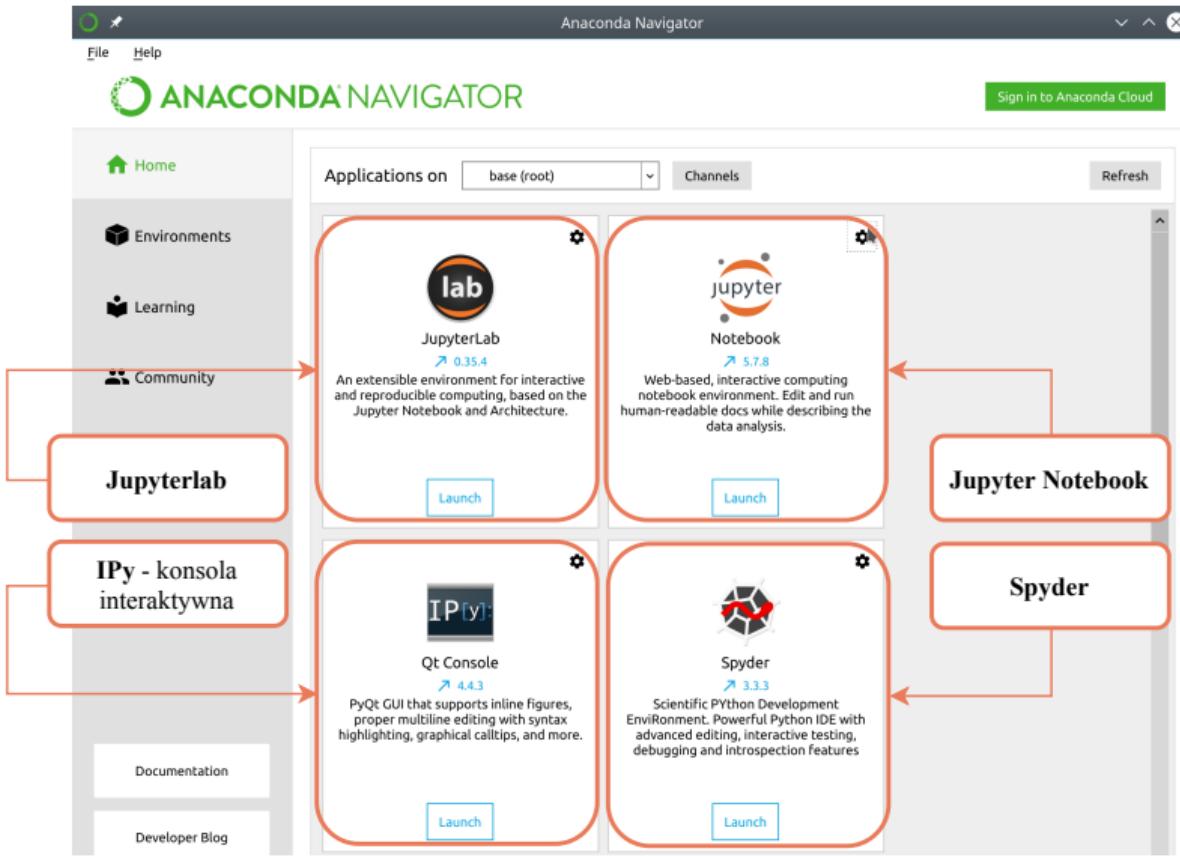
■ **Anaconda Navigator** - graficzne okno zarządzania platformą.

- **Jupyterlab** - system zarządzania dokumentami Jupyter Notebook.
- **Jupyter Notebook** - edytor dokumentów posiadający wbudowany kompilator Pythona, pozwala na uruchamianie i edycję kodu w dokumencie.
- **IPython** - konsola interaktywna Pythona.
- **Spyder** - środowisko graficzne kompilatora Python.
- **Environments** - pozwala zarządzanie pakietami, tworzenie wirtualnych środowisk pracy.

■ **Anaconda Prompt** - okno poleceń (konsola) będące powłoką Pythona dla systemów Windows (w systemi Linux rolę te pełni standardowa powłoka shell). Komponentami powłoki są:

- **konsola interaktywna Pythona** (wpisanie komendy python i naciśnięcie klawisza enter) - konsola pozwala na pisanie kodu Pythona w linii poleceń
- **conda** - menadżer pakietów/bibliotek Pythona, pozwala na instalowanie bibliotek oraz kontrolę wersji.

PLATFORMA ANACONDA: ANACONDA NAVIGATOR





ANACONDA NAVIGATOR

Sign in to Anaconda Cloud

Home Environments base (root) Search Environments Installed Channels Update index... Search Packages

Name Description Version

jpype1lab_nb_ext A configuration metapackage for enabling anaconda-bundled jupyter extensions 0.1.0

alabaster Configurable, python 2+3 compatible sphinx theme. 0.7.12

anaconda Simplifies package management and deployment of anaconda 2019.03

anaconda-client Anaconda.org command line client library 1.7.3

anaconda-project Tool for encapsulating, running, and reproducing data analysis projects 3.1.2

asn1crypto Python ASN.1 library with a focus on performance and memory efficiency 1.3.0

astroid Abstract syntax tree for python with inference support 3.1.2

estropy Community-developed python library for astronomy 3.1.0

atomicwrites Atomic file writes. 19.1.0

attrs Attrs is the python package that will bring back the joy of writing classes by relieving you from the drudgery of implementing object protocols (aka dunder methods). 19.1.0

babel Utilities to internationalize and localize python applications 2.8.0

backcall Specifications for callback functions passed in to enable function calls from other threads 0.0.0

backports Backport of new features in python's os module 3.1.0

backports.shutil_get_terminal_size A backport of the get_terminal_size function from python 3.3 3.1.0

beautifulsoup4 Python library designed for screen-scraping 4.7.1

bitarray Efficient arrays of booleans – c extension 1.1.0

bikcharts High level chart types built on top of bokeh 0.0.0

blas 3.1.0

bleach Easy, whitelist-based html-sanitizing tool 1.1.50

blosc A blocking, shuffling and loss-less compression library that can be faster than 'memcpy()' 1.1.50

Documentation Developer Blog

Create Clone Import Remove 273 packages available

Zarządzanie środowiskiem pracy.
Lista bibliotek oraz ich wersje.

Tworzenie wirtualnego środowiska pracy.
Środowisko wirtualne to częściowo izolowane środowisko Python, które pozwala instalować pakiety (ich odpowiednie wersje) do użycia przez określoną aplikację, zamiast instalować je w całym systemie.



Jupyter Notebook – dokument z interaktywnym kodem.

- Dokument tworzymy poprzez dodawanie komórek z następującym typem danych:

Code komórka z kodem

Markdown komórka z opisem (Markdown – język formatowania tekstu).

Raw komórka surowych znaków ASCII
(ang. *American Standard Code for Information Interchange*);

Heading komórka z nagłówkiem tytułem;

The screenshot shows a Jupyter Notebook interface with the title "Jupyter testJupyter_sat_orbit (automated)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a Logout button. A status bar at the bottom right indicates "Trusted" and "Python 2".

The notebook content includes a text cell with the following text:

Then, expressing the angular velocity in terms of the orbital period and then rearranging, we find Kepler's Third Law:

$$h = \sqrt{\frac{GM \cdot T^2}{4\pi^2}}$$

Below this is a list of variables and their descriptions:

- R - positioning radius
- R_E - Earth's radius
- h - altitude of satellite
- G - gravitational constant
- M_E - Earth's mass= angularvelocity=vs - linear velocity
- T - a period of revolution should be expressed in a continuous number of second

At the bottom of the text cell, the formula $T = 23^{\text{h}}56^{\text{m}}4.091^{\text{s}} = 23 \cdot 3600 + 56 \cdot 60 + 4.091 = 86164.091$ is shown.

The code cell (In [2]) contains Python code to calculate the height of a satellite given its orbital period and Earth's parameters:

```
from math import pi
def h_sat(T):
    # constants
    RE = 5.972 * 10**24      # [kg] - mass of the Earth
    G = 6.67408 * 10**-11     # [m^3 / (kg s^2)] - universal gravitational constant (Newton's constant)
    GM = 3.986005 * 10**14    # [m^3/s^2]
    RE = 6378888              # [m] mean radius of the Earth
    # solution
    GM = G * RE
    h = ((GM * T * T) / (4 * pi*pi))**(1/3.) - RE # [m] altitude of a satellite
    return h / 1000 # [km]
```

The output cell (In [5]) shows the result of running the code:

```
#input: T = 23hh 56mm 4.091 = 86164.091
h = h_sat(86164)
print("height of satellite in km =", h)
(height of satellite in km = 35670.2342967798)
```

- Przykłady formatowania tekstu w języku Markdown na dysku OneDrive:
[tutoriale_pomoce/help_jupyter_markdown.ipynb](#)
- Dokumentacja: <https://jupyter.org/documentation>
- Edytor online: <https://notebooks.azure.com>



IPython - Interaktywny wiersz poleceń

Sesja interaktywna wykonuje kod i zwraca wyniki w miarę wpisywania kodu, jednak nie zapisuje kodu w pliku. Jest to świetna opcja do eksperymentowania z językiem i testowania fragmentów programu "w locie" (Lutz, 2011, p.83).

- Pozwala na pisanie bloków kodu: funkcji, instrukcji warunkowych itp.
- Wspiera wyświetlanie wykresów w konsoli.
- Posiada mechanizm podpowiadania dla obiektów: po wpisaniu nazwy obiektu i kropki (np.: `x.`) wciskamy [Tabulator] - pojawi się okno dialogowe z nazwami dostępnych pól i metod.
- Pozwala na pisanie bloków kodu (pętle, instrukcje warunkowe). Zakończenie bloku kodu następuje po dwukrotnym naciśnięciu przycisku enter.

The screenshot shows a window titled "Jupyter QtConsole". The menu bar includes File, Edit, View, Kernel, Window, and Help. The main area displays an interactive Python session:

```
Jupyter QtConsole 4.4.3
Python 3.7.3 (default, Mar 27 2019, 22:11:17)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import math

In [2]: a = math.sin(0.5) + 145

In [3]: for x in range(0,8):
...:     if x > 4:
...:         print('tutaj x jest większe od 4:', x)
...:

tutaj x jest większe od 4: 5
tutaj x jest większe od 4: 6
tutaj x jest większe od 4: 7

In [4]:
```



Spyder - zintegrowane środowisko programistyczne

Wieloplatformowe zintegrowane środowisko programistyczne (ang. IDE, *Integrated Development Environment*), udostępnione na zasadach otwartego oprogramowania (ang. *open source*) – licencja MIT – przeznaczone do programowania naukowego w języku Python. Spyder został napisany w języku Python i integruje w sobie wiele pakietów Pythona.

ANACONDA: ŚRODOWISKO IDE (SPYDER)



Project explorer -
Spis plików projektu

The screenshot shows the Spyder IDE interface with several windows open:

- Project explorer:** Shows the directory structure of the project, including files like `__pycache__`, `data`, `projekt_2_opis`, `convert_date.py`, `convert_names.py`, `impPrintNav.py`, `main.py`, `recursive_iteration.py`, and `temp.py`.
- Editor:** Displays the Python code for `temp.py`. The code includes importing `enumerate` from `itertools`, defining a list `lista`, creating an index list `indexy` by finding indices of 5, and printing the new list `new` which contains elements from index 11 to 12.
- Variable explorer:** Shows a table of variables with their types and values. Variables include `element` (int, value 72), `i1` (int, value 6), `i2` (int, value 12), `index` (int, value 15), `indexy` (list, value [6, 9, 12]), `lista` (list, value [23, 6, 3, 76, 2, 9, 5, 2, 89, 5, 53, 68, 5, 98, 14, 72]), and `new` (list, value [5, 2, 89, 5, 53, 68, 5]).
- Outline:** A sidebar showing the outline of the current file, listing imports and function definitions.
- IPython:** An interactive console window showing command history and runfile logs.
- Help:** A help window providing documentation for functions.

Variable explorer - podgląd wartości zmiennych
Help - dokumentacja funkcji

Outline -
"Spis treści kodu programu" np. wykaz funkcji, instrukcji warunkowych itp.

Editor -
Okno edycji programów/skryptów Python

IPython -
konsola interaktywna



Wybrane opcje:

- Ustawienia główne Spydera - menu: **Tools** > opcja **Preferences**
- Ustawienie okna głównego - menu: **View** > opcja: **Panes**. Należy wybrać z opcji **Panes** interesujące nas okna, uprzednio należy odblokować ustawienia okien - menu: **View** > opcja: **Lock panes**
- Mechanizm podpowiadania dla obiektów: po wpisaniu nazwy obiektu i kropki (np.: x.) wciskamy [CTRL+Spacja] - zobaczymy wtedy okno dialogowe z nazwami pól i metod dostępnymi dla danego obiektu.
- Pomoc dla funkcji. Pomoc dla danej funkcji można uzyskać poprzez zaznaczenie nazwy funkcji w oknie edytora, następnie należy nacisnąć klawisze [Ctrl + I] po wykonaniu tej instrukcji w oknie "help" pojawi się dokumentacja dla danej funkcji.



Menadżery pakietów służą do zarządzania wersją Pythona, do instalowania (aktualizacji, utrzymywania zgodności wersji) pakietów oraz do tworzenia środowisk programistycznych.

- **PIP (package management system)** - (ang. *Pip Installs Packages*) to system zarządzania pakietami służący do instalowania i zarządzania pakietami dla Python. Wiele pakietów znajduje się w domyślnym źródle pakietów - Python Package Index (PyPI, <https://pypi.org/>) jest domyślnym menadżerem bibliotek. Komendy **PIP** uruchamiamy w konsoli Windows **cmd** lub **PowerShell**. Dokumentacja PIP: <https://pypi.org/project/pip/>
- **Conda** - to system zarządzania pakietami typu open source. Conda szybko wyszukuje, instaluje, uruchamia i aktualizuje pakiety oraz ich zależności. Tworzy i zarządza środowiskami programistycznymi. Menadżer został stworzony do programów w języku Python, ale może pakować i dystrybuować oprogramowanie dla dowolnego języka. W domyślnej konfiguracji conda zarządza tysiącami pakietów ze źródła repo.anaconda.com, które są budowane, weryfikowane i obsługiwane przez Anaconda. Dokumentacja Conda: <https://conda.io/projects/conda/en/latest/user-guide>



<https://conda.io/docs/commands.html>

- Lista zainstalowanych bibliotek:

```
1 conda list
```

- Wyszukiwanie bibliotek:

```
1 anaconda search -t conda library_name
```

- Installowanie bibliotek:

```
1 conda install library_name
```

- Pomoc na temat komendy conda:

```
1 conda install -- help
```



W edytorze tekstowym (lub w środowisku IDE) tworzymy plik, stosujący kodowanie ASCII lub UTF-8 Unicode, z rozszerzeniem **.py** (w nazwach plików unikamy spacji i polskich znaków).

- Linia 1: nosi nazwę **shebang** (od *shell execute* In[1]: czyli plik wykonywalny powłoki), informuje o ścieżce lokalizacji kompilatora Pythona.

Występuje w formach `#!/usr/bin/python` oraz `#!/usr/bin/env python3`. W systemie Windows oraz w IDE shebang nie występuje.

- Linia 2: oznacza przyjęty system kodowania znaków, definicję kodowania podajemy jedynie dla programów Python 2.x. Python 3. domyślnie stosuje kodowanie UTF-8 (większy wybór znaków), więc wiersz nie jest wymagany.

Out[1]:

- Linia 4: importowanie pakietów/bibliotek
- Linia 6-9: kod programu

```
1 #!/usr/bin/python
2 # -*- coding: UTF-8 -*-
3
4 import math
5
6 a = 7
7 b = 8
8 wynik = a + b + math.pi
9 print('wynik:', wynik)
```

1 wynik: 18.141592653589793



- Komentarze rozpoczynamy znakiem #,
- Ciągi dokumentacyjne (Multi-line Docstrings) to literał łańcuchowy występujący jako pierwsza instrukcja w definicji modułu, funkcji, klasy lub metody. Taki ciąg dokumentów staje się specjalnym atrybutem `__doc__` tego obiektu. Otoczeniem dokumentacji jest potrójny cudzysłów. Zdefiniowaną dokumentację możemy uzyskać wpisując w konsoli komendę: `help(nazwa_funk)` lub `nazwa_funk.__doc__`. Więcej w **PEP 257 – Docstring Conventions**: www.python.org/dev/peps/pep-0257/
- Dobre praktyki zapisu kodu definiuje dokument **PEP 8 – Style Guide for Python Code**: www.python.org/dev/peps/pep-0008/

In [2] :

```

1 #!/usr/bin/python
2 ﻿# -*- coding: UTF-8 -*-
3
4 """
5 Documentacja
6 Nazwa funkcji: ...
7 """
8 import math
9
10 a = 7 # zmienna a
11 b = 8 # zmienna b
12 wynik = a + b + math.pi
13 print('wynik:', wynik)

```

Out [2] :

1	wynik: 18.141592653589793
---	---------------------------





```

1 import math
2
3 a = math.sin(0)
4 print('sin(0) = ', a)
1 sin(0) = 0.0

```

```

1 import math as m
2
3 a = m.sin(0)
4 print('sin(0) = ', a)
1 sin(0) = 0.0

```

```

1 from math import sin
2
3 a = sin(0)
4 print('sin(0) = ', a)
1 sin(0) = 0.0

```

- Importowanie całej biblioteki, wszystkie funkcje biblioteki są dostępne po przedrostku `math.` (nazwa przedrostka informuje o nazwie biblioteki).

- Importowanie całej biblioteki ze skróconą nazwą, wszystkie funkcje biblioteki są dostępne po przedrostku `m.` (nazwa przedrostka informuje o nazwie biblioteki).

- Importowanie wybranych funkcji z biblioteki, wszystkie funkcje są dostępne bez podawania nazwy przedrostka. Można importować kilka funkcji np.: `from math import cos,sin`



Pomoc dla modułów/bibliotek/funkcji jest dostępna w konsoli interaktywnej Pythona, lub w edytorze IDE. Pomoc można uzyskać po uprzednim zimportowaniu biblioteki.

■ IPython console:

- `help(nazwa)`, gdzie nazwa jest nazwą funkcji lub całej biblioteki.
- `dir(nazwa)`, wyświetli metody dostępne dla danego obiektu.

■ Spyder editor: zaznaczenie nazwy funkcji i wcisnięcie Ctrl+I, pomoc pojawi się w oknie Help (w standardowych ustawieniach to prawe górne okno)

Menadżer podpowiedzi to narzędzie inteligentnego podpowiadania i auto-uzupełniania kodu (klasy, metody, funkcje, zmienne, pliki, itp.), jest narzędziem typowym dla środowisk IDE i w dużym stopniu ułatwia i przyśpiesza pisanie kodu.

- IPython console: wpisanie pierwszych liter nazwy i naciśnięcie tabulatora
- Spyder editor: wpisanie pierwszych liter nazwy i naciśnięcie tabulatora lub naciśnięcie Ctrl+Spacja



- Funkcja `input()` Wyświetla znak zachęty, jeśli ten argument zostanie przekazany, a następnie czyta wiersz ze standardowego strumienia wejściowego `stdin` (`sys.stdin`) i zwraca go w formie łańcucha znaków. Obcina końcową sekwencję `\n` na końcu wiersza oraz zgłasza wyjątek `EOFError` w przypadku osiągnięcia końca strumienia `stdin`.
- Funkcja `input()` wykorzystuje mechanizm GNU `readline` (odczytany treść jest typu tekstowego) na platformach, które go obsługują. W Pythonie 2.X ta funkcja ma nazwę `raw_input()`. (Lutz, 2015)

```
1 a = input("symbol zachęty")
```

```
1 a = input("podaj wartość która zostanie przypisana do a")
```



- Funkcja `print()` wbudowana funkcja Pythona, wyświetla tekst do standardowego strumienia wyjściowego. Domyślnie funkcja wywołuje metodę `write()` obiektu, do którego w danym momencie odwołuje się urządzenie `sys.stdout` (wyjście na konsolę) ([Lutz, 2015](#))

1 `print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)`

- `sep` – Łańcuch znaków, który ma być umieszczony pomiędzy wartościami (domyślnie spacja: `' '`).
- `end` – Łańcuch znaków do umieszczenia na końcu wyświetlonego tekstu (domyślnie znak nowego wiersza: `'\n'`).
- `file` – Obiekt postaci pliku, do którego jest zapisywany tekst (domyślnie standardowe wyjście: `sys.stdout`).
- `flush` – Wartość `true` lub `false` umożliwiająca włączenie lub wyłączenie wymuszonego opróżnienia strumienia wyjściowego (wprowadzone w Pythonie 3.3 – domyślnie `False`).



Funkcje odczytujące linię z danych wejściowych na konsoli:

```
1 text = raw_input("prompt") # Python 2  
2 text = input("prompt")    # Python 3
```

Przykład:

In[3] :

```
1 a = input("Podaj wartość a: ")
```

Out[3] :

```
1 Podaj wartość a:
```

In[4] :

```
1 a = input("Podaj wartość a: ")  
2 # rzutowanie typu string na typ liczbowy (float lub int)  
3 wynik = 5 + float(a)  
4 print('5 plus Twoja liczba daje wynik: ', wynik)
```

Out[4] :

```
1 5 plus Twoja liczba daje wynik: 10
```



- Jeśli wywołamy program bez błędów składniowych i proceduralnych to program zwróci (np. do konsoli) jedynie to co programista każe mu zwrócić (np. w funkcji print)
- Jeśli wywołamy program, który zawiera błędy składniowe i proceduralne w konsoli otrzymamy *komunikaty z błędami*.
- *Komunikaty błędów* - definują w którym miejscu programu znajduje się błąd oraz jakiego rodzaju to błąd. **Należy uważnie czytać komunikaty z błędami!!!**

Przykład:

In [5] :

```
1     a = b + 12
```

Out [5] :

```
1 File "<ipython-input-7-f157132b5f9e>", line 1, in <module>
2     a = b + 12
3 NameError: name "b" is not defined
```



M. Lutz. *Python. Wprowadzenie*. Helion, 2011.

M. Lutz. *Python, Leksykon kieszonkowy*. Helion, 2015.



ME AFTER 10 LINES OF CODING



Enough For Today!

Dziękuje za uwagę

Kinga Węzka
Gmach Główny PW – pok 38/6
kinga.wezka@pw.edu.pl