Wydział: **Zarządzania i Modelowania Komputerowego**
Katedra Technologii Informatycznych
Przedmiot: **Technologie informacyjne**
Rok **I**

Wprowadzenie do programowania w języku Python

Języki programowania

Realizowany przez komputer program składa się z rozkazów – instrukcji maszynowych w kodzie zero-jedynkowym. Tak zapisany program w języku wewnętrznym jest bardzo nieczytelny.

Użytkownicy formułują algorytmy w językach programowania wyższego poziomu.

Języki programowania posiadają:

- ściśle określony alfabet tj. zbiór możliwych słów
- ściśle określone reguły składniowe (syntaktyka)
- jednoznaczne reguły interpretujące znaczenie poszczególnych napisów (semantyka).

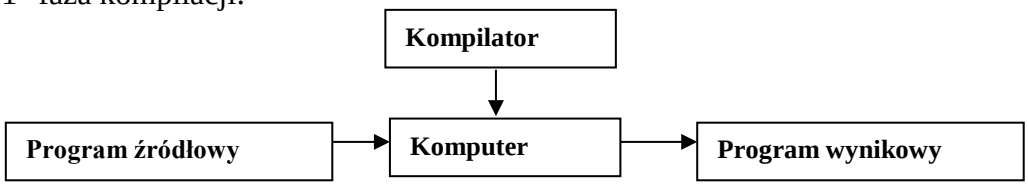
Program napisany w języku programowania może być przetłumaczony na język wewnętrzny komputera przez program nazywany translatorem.

Wyróżnia się dwa rodzaje translatorów:

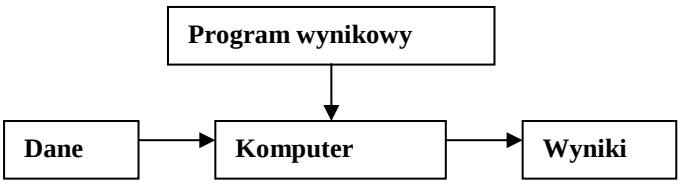
- kompilatory
- interpretatory.

A. W przypadku wykorzystania kompilatora obliczenia prowadzone są w dwu fazach:

1° faza kompilacji:



2° faza wykonania:



B. W przypadku wykorzystania do obliczeń interpretatora każda instrukcja programu źródłowego, po przetłumaczeniu, jest natychmiast wykonywana. Nie tworzona jest wersja wynikowa programu (najczęściej posiadająca rozszerzenie .exe). Za każdym razem wykonanie programu wymaga ponownego tłumaczenia.

Python jest prostym **językiem zorientowanym obiektowo (OOP –*Open Oriented Programming*)**, posiadającym czytelną składnię, łatwy w utrzymaniu i integracji z komponentami języka C, posiadającym bogaty zbiór interfejsów, bibliotek i narzędzi programistycznych.

Język Python wyposażony jest w translator typu interpretator. co daje większą łatwość modyfikacji gotowego programu, lecz obniża efektywność działania w stosunku do języków kompilowanych, takich jak C.

Program źródłowy **napisany w języku Python (podobnie jak w Java) może być najpierw** kompilowany do postaci pośredniej (byte-code**), która następnie wykonywana jest przez** Wirtualną Maszynę Pythona (PVM) **na konkretnej platformie obliczeniowej.**

Darmowa wersja jest dostępna na stronie: https://www.python.org/
Dla posiadaczy systemu Windows 10 proponujemy Python 3.9.0 , jeśli ktoś dysponuje tylko Windows 7, to Python 3.8.6 – ścieżka do instalacji poniżej.
https://www.python.org/ftp/python/3.8.6/python-3.8.6-amd64.exe

Python – elementy języka

Stałe: liczby i łańcuchy znaków

- liczby: **5, 3.24, 9.86e-3, (-5 + 2j)**
- stałe logiczne**: True, False**
- łańcuchy znaków: **'Łańcuch'**, **"Łańcuch"** –w apostrofach lub cudzysłowach
- " " " Za pomocą potrójnych cudzysłowów można oznaczać łańcuchy wielolinijkowe" " "
- w łańcuchach można używać znaków specjalnych, np. znaku nowej linii: \n "Pierwszy wiersz.\nDrugiwiersz."
- jeśli łańcuch poprzedzimy znakiem **r** lub **R**, wtedy nie są uwzględniane znaki specjalne, a tekst jest traktowany dosłownie: **R" …\n…"** –tzw. łańcuch surowy
- chcąc poprawnie wyświetlać polskie znaki diakrytyczne najlepiej posłużyć się unikodem sygnalizując to znakiem **u**lub **U**: **U"Tekstzapisaliśmy w Unikodzie"**
- Python automatycznie łączy łańcuchy obok siebie: **'Jan' 'Kot'** zostanie automatycznie przekonwertowany przez interpreter Pythona na **'JanKot'**
- łańcuchy w Pythonie są niezmienne, lecz istnieją metody operowania na nich, tworząc nowe z części starych.

Zmienne

Zmienne umożliwiają przechowywanie w pamięci komputera danych, które mogą zmieniać się w czasie, np. w wyniku obliczeń.
Zmienne posiadają swoje **nazwy**, które je identyfikują:

- pierwszym znakiem identyfikatora (nazwy) musi być mała lub duża litera alfabetu (łacińskiego) albo podkreślnik(_), lecz takie zmienne mają specjalne znaczenie.
- pozostałe znaki w nazwie mogą być małymi lub dużymi literami alfabetu łacińskiego, podkreślnikiem lub cyframi (0–9).
- wielkość znaków w identyfikatorze jest ważna, stąd **a1** i **A1** to dwie różne zmienne.
- przykłady poprawnych identyfikatorów to: **i**, **A2**, **_an**, naZwa_1, a1
- nazwy zmiennych nie mogą rozpoczynać się od cyfry, zawierać spacji ani łączników; **nie są** poprawnymi nazwami 3A, nazwa ze spacjami, nazwa-z-lacznikiem
- zmienne nie mogą posiadać nazw zastrzeżonych dla Pythona:
  and assert break class continue def del elif else except exec finally for from global if import in is lambda not or pass print raise return try while yield

Operatory

Operatory służą do definiowania operacji matematycznych, logicznych i symbolicznych:

operatory arytmetyczne

+	dodawanie
-	odejmowanie
*	mnożenie
**	potęgowanie:
/	dzielenie
//	dzielenie całkowite
%	reszta z dzielenia:

operatory relacyjne

<	mniejsze
<=	mniejsze lub równe
>	większe
>=	większe lub równe
==	równe
!=	różne

operatory logiczne

not	negacja
or	alternatywa
and	koniunkcja

operatory działające na zbiorach:

+	dodawanie zbiorów
*	multiplikacja zbiorów
|	suma zbiorów
&	iloczyn zbiorów
-	odejmowanie zbiorów

komentarz

#	znak sygnalizujący, że wszystkie znaki za nim, do końca linii, są komentarzem, ignorowanym przez translator

Tryby pracy

Praca w środowisku programistycznym Pythona może odbywać się na dwa sposoby:

- tryb interaktywny (linia poleceń) – wprowadzane są pojedyncze instrukcje i natychmiast wykonywane
- tryb skryptowy (aplikacyjny)– grupa instrukcji zapisywana jest w pliku nazywanym skryptem (rozszerzenie .py) a następnie sekwencyjnie wykonywana.

Tryb interaktywny

Można wykorzystywać jako rozbudowany kalkulator do obliczania wartości wyrażeń.

```
4 - 1
2 + 2
3 / 4
7//2
5 * 2
3**2
4 % 3
```

W trybie interaktywnym można także wykonywać instrukcje języka.

Instrukcje

Poniżej zostaną opisane podstawowe instrukcje języka Python:

1. instrukcja przypisania
2. instrukcja wyprowadzania wyników
3. instrukcja wprowadzania danych
4. instrukcja warunkowa
5. instrukcje pętli:
   a. pętla **for**
   b. pętla **while**

1. instrukcja przypisania:

zmienna = wyrażenie

może też mieć postać:

zmienna_1, zmienna_2,...,zmienna_n=wartość_1, wartość_2,...,wartość_n

nazywana jest wtedy instrukcją przypisania wielokrotnego i jest równoważna ciągowi instrukcji przypisania:

zmienna_1 =wartość_1
zmienna_2=wartość_2
…
zmienna_n=wartość_n

przy czym najpierw obliczane są wszystkie wartości po prawej stronie znaku = a dopiero wtedy ma miejsce przypisanie.

Przykładowo:

```
a=2
b=4
y=(a+1)*(b-1)**2
```

Można stosować funkcje matematyczne np.: sin, cos, ln, itp. ale po wprowadzeniu polecenia przyłączającego moduł matematyczny:

import math

Korzystając z funkcji należy poprzedzić jej nazwę prefiksem math. , przykładowo poniższe polecenie przypisze zmiennej **p** wartość pierwiastka kwadratowego z liczby **7**

```
p=math.sqrt(7)
```

Wykaz najczęściej używanych funkcji:

Funkcja	**opis**
acos(x)	arccos – wynik w radianach
asin(x)	arcsin– wynik w radianach
atan(x)	arctangens– wynik w radianach
cos(x)	cosinus – argument x w radianach
degrees(x)	zamiana kąta x w radianach na stopnie
exp(x)	potęga przy podstawie e
fabs(x)	wartość bezwzględna (moduł liczby)
log(x)	logarytm naturalny
log10(x)	logarytm dziesiętny
pow(x,y)	x do potęgi y
radians(x)	zamiana kąta x w stopniach na radiany
sin(x)	sinus – argument x w radianach
sqrt|(x)	pierwiastek kwadratowy
tan(x)	tangens – wynik w radianach

Pełny zestaw funkcji wchodzących w skład modułu **math** można obejrzeć wprowadzając polecenia:

```
import math
help (math)
```

W wyrażeniach można stosować również zdefiniowane w module stałe:

pi – wartość liczby π (3.141592653589793...)
e – wartość podstawy logarytmu naturalnego (2.718281828459045...)

Przykłady:

```
r=4
s=math.pi*r**2
x=3
y=4
d=math.sqrt(x**2+y**2)
```

Gdy użytkownik planuje wykorzystanie określonych funkcji modułu matematycznego, może przyłączyć tylko te funkcje, np.:

```
from math import sqrt, sin
```

i wtedy w wyrażeniach nie ma potrzeby poprzedzania nazwy prefiksem:

```
d=sqrt(x**2+y**2)
```

2. instrukcja wyprowadzania wyników:

print (lista-wyjścia)

gdzie **lista-wyjścia** zawiera rozdzielone przecinkami teksty i wyrażenia, których wartość ma być zaprezentowana:

```
print (math.sqrt(2**2+1))        # o ile przyłączony był pełen moduł math
print ('y =',3*5+1)
```

Wyprowadzane wyniki można sformatować, wtedy instrukcja ma postać:

print ("łańcuch formatujący" % (lista wyników))

przykładowo:

```
print ("x=%2i y=%i" % (x, x**2))
```

Znak % w łańcuchu formatującym poprzedza w zależności od wyprowadzanej wartości:
s – gdy wyprowadzany jest ciąg znaków
c – gdy wyprowadzany jest jeden znak
i – gdy wyprowadzana jest liczba całkowita
f – gdy wyprowadzana jest liczba rzeczywista
e – gdy wyprowadzana jest liczba w postaci wykładniczej

Przed symbolem literowym można (ale nie jest to konieczne) określić szerokość pola dla wyprowadzanej liczby lub tekstu.

3. instrukcja wprowadzania danych:

input ('prompt')

gdzie **prompt** jest łańcuchem zachęty (może nie występować):

Zazwyczaj instrukcja wprowadzania danych wykorzystywana jest w postaci;

zmienna=input ('prompt')

która przypisuje do podanej zmiennej **łańcuch tekstu** wprowadzony z klawiatury

Chcąc więc wykonywać działania arytmetyczne na tak wprowadzonych danych należy dokonać konwersji korzystając z odpowiedniej funkcji:

int(zmienna)-na liczby całkowite
float(zmienna)-na liczby wymierne
complex(zmienna)-na liczby zespolone

4. instrukcja warunkowa

```
if warunek1:
    instrukcja11
    instrukcja12
    …
elif warunek2:
    instrukcja21
    instrukcja22
    …
else:
    instrukcja31
    instrukcja32
    …
```

Działanie instrukcji jest następujące: jeśli spełniony jest ***warunek1*** wykonywany jest blok instrukcji poniżej warunku; jeśli ***warunek1*** nie jest spełniony instrukcje tego bloku są pomijane i badany jest ***warunek2***, ten po **elif.** Jego spełnienie powoduje wykonanie bloku po **elif,** w przeciwnym przypadku jego pominięcie. (człon **elif** może wystąpić wielokrotnie). Jeśli żaden z warunków nie jest prawdziwy, wykonywany jest blok instrukcji po **else.**
Części **elif** oraz **else** są opcjonalne, instrukcja warunkowa może mieć więc postać:

```
if warunek1:
    instrukcja11
    instrukcja12
    …
```

czyli przy prawdziwym warunku wykonywany jest blok instrukcji, jeśli jest on fałszywy, nie podejmowana jest żadna akcja;
lub postać:

```
if warunek1:
    instrukcja11
    instrukcja12
    …
else:
    instrukcja31
    instrukcja32
    …
```

Bloki instrukcji w Pythonie wyróżniane są za pomocą odpowiedniego wcięcia (tabulatora). Nie stosuje się tutaj żadnych znaków specjalnych, jak w innych językach programowania!

W sytuacji, gdy blok składa się z pojedynczej instrukcji, może ona być zapisana w linii bezpośrednio po dwukropku.

Instrukcje warunkowe można zagnieżdżać, co zaznaczane jest przez odpowiednio duże (wielokrotne) wcięcia, wyznaczające zagnieżdżone bloki instrukcji.

Warunki budowane są przy użyciu operatorów relacyjnych i logicznych.
Wynikiem każdego porównania może być wartość prawdy (**True**) lub fałszu (**False**).
Uwaga: Nie wolno mylić operatora = (operator przypisania) z == (operator równości)
Przykłady wyrażeń logicznych:

```
10>8
5<=5
2==1
3!=3
'Ola'=='Ala'
```

'Ola'>'Ala'
not 2==1
2>=2 or 3==1
3==1 and 2>3
not 3>4 or 2>3 and 4!=2

W Pythonie operatory porównań mogą występować wielokrotnie

1>2>3
2>=1>4

W pierwszej kolejności wykonywane są operatory porównania, później operator negacji, następnie iloczynu logicznego, a na końcu sumy logicznej (takie same operatory wykonywane są w kolejności od lewej do prawej)

Przykład 1

```
print ("Badanie czy liczba dodatnia")
x= float(input("Podaj liczbę:"))
if x>0:
    print ("Liczba dodatnia")
```

Przykład 2

```
print ("Badanie parzystości liczb")
x= int(input("Podaj liczbę:"))
if x%2==0:
    print ("Liczba parzysta")
else:
    print ("Liczba nieparzysta")
```

Przykład 3

```
print ("Porównywanie dwóch liczb")
x= float(input("Podaj pierwszą liczbę x="))
y= float(input("Podaj drugą liczbę      y="))
if x==y:
    print ("Liczby takie same")
elif x>y:
    print ("Liczba x większa od y")
else:
    print ("Liczba x mniejsza od y")
```

5. instrukcje pętli

a. instrukcje pętli for

for *licznik* in *sekwencja*:
instrukcja1
instrukcja2
…

Instrukcja pętli **for … in …** służy do iterowania po dowolnej *sekwencji* obiektów, gdzie *sekwencja* to uporządkowany zbiór elementów. Instrukcje bloku po **:** wykonywane są dla każdej wartości *licznika*, który przyjmuje wartości zdefiniowane przez sekwencję. Ważne, by wszystkie instrukcje bloku zapisywane były z takim samym wcięciem.

Do tworzenia sekwencji bardzo często stosuje się metodę:

range([start,] stop [, krok])

gdzie parametry **start** i **krok** są opcjonalne; pominięcie **start** jest równoznaczne z przyjęciem wartości startowej **0**, pominięcie parametru **krok** oznacza przyjęcie kroku (odległości między elementami sekwencji) równego **1**; parametr **stop** jest ograniczeniem sekwencji, już do niej nie należy.

range(10)	generuje sekwencję:	0,1,2,3,4,5,6,7,8,9
range (1,10)	generuje sekwencję:	1,2,3,4,5,6,7,8,9
range (1,10,2)	generuje sekwencję:	1,3,5,7,9
range (9,0,-2)	generuje sekwencję:	9,7,5,3,1

Przykład 4

Instrukcja, która wydrukuje kwadraty liczb od 0 do 10:

```
for x in range(11):
    print (x,x**2)
```

Jeśli wprowadzamy tę instrukcję w trybie bezpośrednim, po każdej linii i na zakończenie wprowadzania należy nacisnąć <Enter>

W celu wyrównania wyprowadzonych tabelarycznie wyników można uzupełnić instrukcję o formatowanie.

```
for x in range(11):
    print ("%4i %5i" % (x,x**2))
```

b. instrukcje pętli while

while *warunek*:
instrukcja1
instrukcja2
…

Instrukcje po : powtarzane są tak długo jak długo prawdziwy jest ***warunek***. Także w przypadku instrukcji pętli **while** wszystkie instrukcje bloku zapisywane są z takim samym wcięciem.

Przykład 5

Obliczanie sumy kwadratów liczb naturalnych do momentu gdy suma ta przekroczy 100; należy wyprowadzić wartość liczby naturalnej, dla której spełniony jest ten warunek.

```
i=0
s=0
while s<100:
    i=i+1
    s=s+i**2
print i, s
```