PROGRAMOWANIE W PYTHONIE – OD PIERWSZYCH KROKÓW

Grażyna Szabłowicz-Zawadzka
http://metodycy.torun.pl/
m.informatyka@metodycy.torun.pl

1. Wprowadzenie do Pythona – podstawowe informacje

- Python to język programowania wysokiego poziomu, ma przejrzystą i zwięzłą składnię.
- Programy pisane są w plikach tekstowych z rozszerzenie *.py*
- Stosowane są wcięcia do wydzielania bloków kodu (stosuje się 4 spacje, nie jest to konieczne, ale musimy być konsekwentni, tzn. jeśli pierwsze wcięcie w funkcji miało trzy spacje, to kolejne wcięcia także muszą mieć trzy spacje, linia bez wcięcia znajdować się będzie poza blokiem).
- Ważna jest wielkość liter.
- IDLE to zintegrowane środowisko programistyczne tego języka dołączone do interpretera Pythona.
- Darmowy – interpretery są dostępne na wiele systemów operacyjnych.

Zintegrowane środowisko programowe IDLE to zestaw narzędzi, które ułatwiają pisanie programów w Pythonie. Po uruchomieniu IDLE zgłasza się **tryb interaktywny**. Aby przejść do edycji nowego programu należy z menu *File* wybrać polecenie *New File*. Otworzy się okno edytora, przeznaczone do pisania programów – **tryb skryptowy**.

Tabela 1 Podstawowe polecenia Pythona

`#komentarz`	jednowierszowe komentarze
`"""  ,,,`	wielowierszowe komentarze ograniczone przez ” lub ’
`print("tekst",zmienna)`	wypisanie komunikatu na ekranie
`a=float(input("podaj liczbę: "))`	wczytywanie zmiennych z klawiatury (tworzenie zmien-

`b=int(input("podaj liczbę: "))` `c=input("podaj tekst: ")`	nych przez przypisanie wartości)

`+ - * / %`
`// (dzielenie całkowite)`
`** (potęgowanie)`
	operatory arytmetyczne

`+= -= *= /= %= //= **=`	złożone operatory przypisania
`== !=< > <= >=` `1<3<7 (wielokrotne porównania)`	relacje
`and or not`	operatory logiczne

2. Instrukcje warunkowe

Tabela 2 Instrukcje warunkowe, biblioteka matematyczna i budowa funkcji

`if w: instrukcja`	instrukcja warunkowa skrócona
`if w: instrukcja` `else: instrukcja`	instrukcja warunkowa pełna

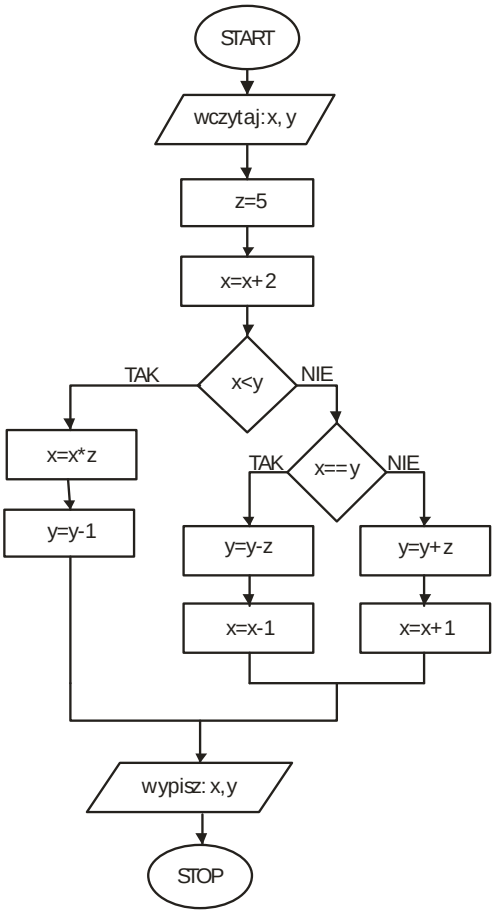
`if w1: instrukcja1`
`elif w2: instrukcja2`
`elif w3: instrukcja3`
`else: instrukcja4`
	instrukcja warunkowa zagnieżdżona (alternatywa)

`if w1:`
`  if w2: instrukcja1`
`  else: instrukcja2`
`else: instrukcja3`
	zagnieżdżenie instrukcji warunkowych (koniunkcja)

`from math import *` `a=sqrt(49)`	dostęp do funkcji biblioteki matematycznej – pierwiastek kwadratowy
`def pierwiastek(n):` `  return n**0.5`	przykład funkcji z jednym wynikiem uruchomienie funkcji: pierwiastek(9) wynik: 3.0
`def dzialania(a,b):` `  return a+b, a-b`	przykład funkcji z dwoma wynikami uruchomienie funkcji: dzialania(1,3) wyniki: (4,-2)

Zadanie 1

Poniżej przedstawiono schemat blokowy pewnego algorytmu. Napisz program realizujący ten algorytm.



Rysunek 1 Schemat blokowy algorytmu do zadania 1

Zadanie 2

Napisz program wyznaczający wartość podanej funkcji:

$$f(x)=\begin{cases} 2x & dla \quad x<1 \\ -10 & dla \quad x=1 \\ x^4 & dla \quad x=3 \\ \sqrt{x-4} & dla \quad x=6 \\ 0 & dla \quad innych \quad x \end{cases}$$

3. Instrukcje iteracyjne

Tabela 3 Instrukcje iteracyjne

`for i in range(5):` `    instrukcje`	pętla for, która powtórzy się 5 razy, dla i = 0, 1, 2, 3, 4

`range(10)`
`range(1, 10)`
`range(3, 10)`
`range(1,10,2)`
`range(9,0,-1)`
	sekwencje liczb:
	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
	[1, 2, 3, 4, 5, 6, 7, 8, 9]
	[3, 4, 5, 6, 7, 8, 9]
	[1, 3, 5, 7, 9]
	[9, 8, 7, 6, 5, 4, 3, 2, 1]

`for x in range(10):` `    print(x, "**2",x**2)`	przykładowa pętla for
`while w: instrukcja`	pętle while
`while w: instrukcja` `else: instrukcja`	

`a=7`
`while a:`
`    a-=1`
`    print(a)`
`else:`
`    print("koniec")`
	przykladowa pętla while

Zadanie 3

Skonstruuj algorytmy iteracyjne w postaci programów realizujące:

1. wypisywanie liczb całkowitych z zakresu <4, 21> podzielnych przez 3, w kolejności od najmniejszej do największej,
2. wypisywanie liczb całkowitych z zakresu <8, 20>, które nie są podzielne przez 5, w kolejności od największej do najmniejszej.

Zadanie 4

Napisz program wykonujący następujące operacje (*n* jest liczbą naturalną większą od 0):

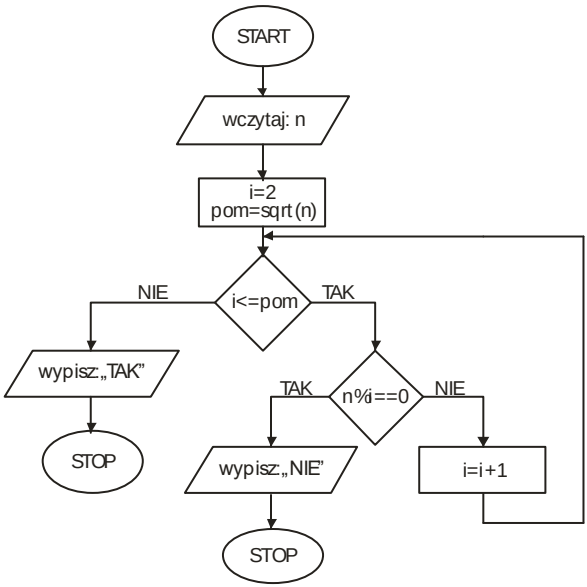
1. wyświetlanie *n*-elementowego ciągu liczb całkowitych (2, 6, 18, 54, 162...),
2. obliczanie sumy *n*-elementowego ciągu liczb całkowitych (–4, 1, 6, 11, 16...),
3. obliczanie iloczynu *n*-elementowego ciągu liczb całkowitych (–1, 2, –4, 8, –16, 32...).

4. Przykładowe proste algorytmy zapisane w Pythonie

4.1. Liczby pierwsze

Dane: Liczba naturalna: $n > 1$.

Wynik: Komunikat informujący, czy liczba n jest liczbą pierwszą.




```
def liczba_pierwsza (n):
    for i in range(2,n):
        if n%i==0:
            return False
    return True
```

4.2. Minimalna wartość

Dane: Liczba naturalna: $n > 0$ (liczba wczytywanych wartości).

Wynik: Najmniejsza z liczb rzeczywistych wprowadzonych z klawiatury: *minimum*.

Lista kroków:

Krok 1. Wczytaj pierwszą liczbę *k*.

Krok 2. Przypisz *minimum* = *k*.

Krok 3. Jeśli $n = 1$, wypisz *minimum* i zakończ algorytm.

Krok 4. Wczytaj liczbę *k*.

Krok 5. Zmniejsz *n* o 1.

Krok 6. Jeśli *k* < *minimum*, przejdź do kroku 2., w przeciwnym wypadku przejdź do kroku 3.

```
def minimum (n):
    k=float(input("podaj liczbę: "))
    min=k
    while n>1:
        k=float(input("podaj liczbę: "))
        if k<min:
            min=k
        n-=1
    return min
```

5. Funkcje rekurencyjne

Obliczanie silni liczby naturalnej

Definicja rekurencyjna algorytmu wyznaczającego n! jest następująca:

$$\begin{cases} a_0 = 1 \\ a_n = a_{n-1} \cdot n \quad dla \quad n > 0 \end{cases}$$

Stosując podaną definicję, można zapisać algorytm w postaci funkcji rekurencyjnej.

```
def silnia (n):
    if n==0:
        return 1
    return silnia(n-1)*n
```

Wyznaczanie elementów ciągu Fibonacciego

Rekurencję można zastosować również do wyznaczenia n-tego elementu ciągu Fibonacciego, którego wyrazy mają następujące wartości:

(1; 1; 2; 3; 5; 8; 13; 21; 34; 55; 89; 144; 233; 377; 610...)

Definicja rekurencyjna ciągu Fibonacciego jest następująca:

$$\begin{cases} a_1 = 1 \\ a_2 = 1 \\ a_n = a_{n-1} + a_{n-2} \quad dla \quad n > 2 \end{cases}$$

Na podstawie definicji, zapisujemy algorytm w postaci funkcji rekurencyjnej.

```
def fibo (n):
    if n==1 or n==2:
        return 1
    return fibo(n-1)+fibo(n-2)
```

Zadanie 5

Podaj definicje rekurencyjne ciągów liczbowych oraz napisz programy wyznaczające n-ty element podanego ciągu:

1. (3; 6; 9; 12; 15; 18; 21; 24...),
2. (–2; 4; –8; 16; –32; 64; –128...),
3. (8; –4; 2; –1; 0,5; –0,25; 0,125...),
4. (2; 6; 3; 7; 4; 8; 5; 9; 6; 10; 7; 11...),
5. (–1; 4; –4; –16; 64; –1024...),
6. (1; 2; –4; –3; –1; –5; –8; –9; –14...).

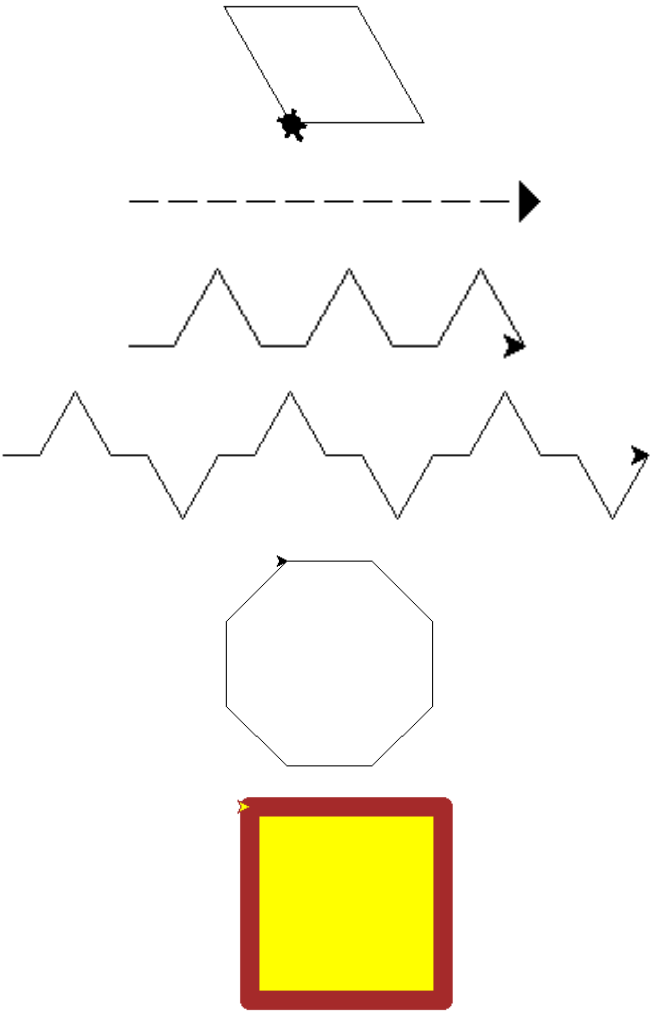
6. Grafika żółwia w Pythonie

Tabela 4 Grafika żółwia w Pythonie

`from turtle import *`	deklaracja biblioteki
`shape("turtle")`	zmiana wyglądu żółwia (turtle – żółw, circle – koło, arrow – strzałka, square – kwadrat, triangle – trójkąt, classic – wygląd klasyczny)
`speed(5)`	szybkość rysowania <1..11>
`fd(100)`	żółw idzie naprzód 100 kroków
`bk(50)`	żółw cofa się 50 kroków
`rt(90)`	żółw obraca się w prawo o 90°
`lt(45)`	żółw obraca się w lewo o 45°
`pu()`	żółw podnosi pisak
`pd()`	żółw opuszcza pisak

Zadanie 6

Korzystając z poleceń dostępnych w bibliotece turtle wykonaj poniższe rysunki.



```
from turtle import *
pencolor("brown")
pensize(15)
fillcolor("yellow")
begin_fill()
for i in range(4):
    fd(150); rt(90)
end_fill()
```

Literatura

1. Dawson M., *Python dla każdego. Podstawy programowania*, Helion, Gliwice 2013.
2. Lutz M., *Python. Wprowadzenie*, Helion, Gliwice 2011.
3. Sysło M.M., *Algorytmy*, Helion, Gliwice 2016.
4. Zawadzka G., *Informatyka. Podręcznik dla szkół ponadgimnazjalnych. Informatyka Europejczyka, część I, Wydanie III,* Helion, Gliwice 2017.