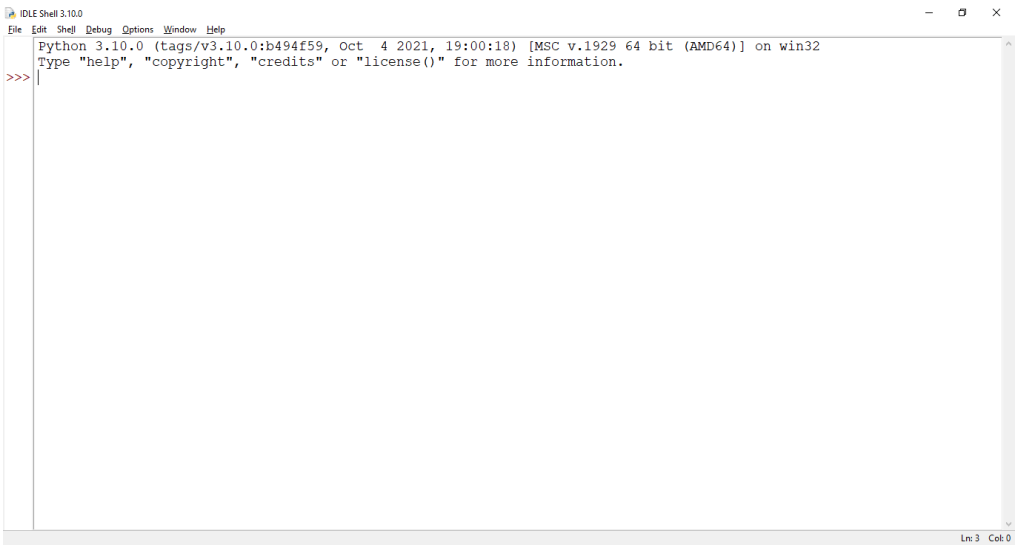
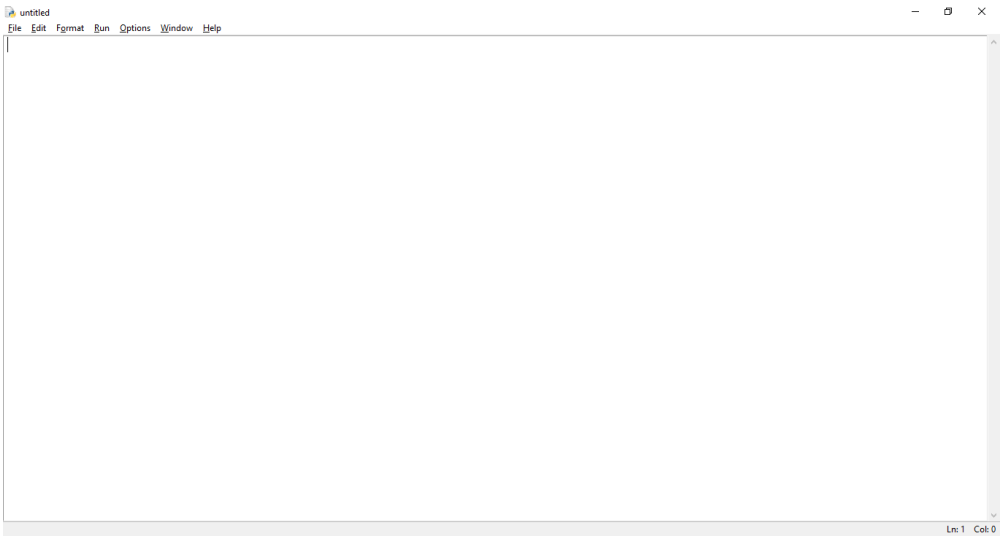
Wprowadzenie do języka Python

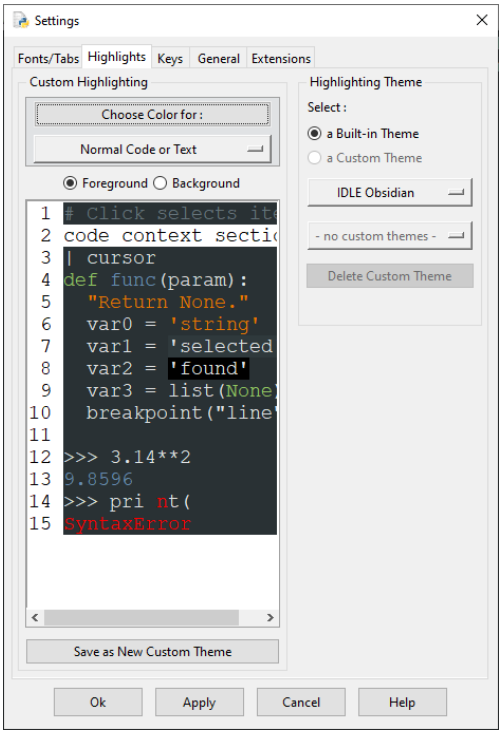
Python realizuje jednocześnie kilka paradygmatów programowania: imperatywny, obiektowy oraz funkcyjny. Jego składnia jest prosta i przejrzysta. Python rozwijany jest jako projekt Open Source, co również stanowi jego zaletę. Olbrzymi wpływ na popularność ma powszechność stosowania Pythona w takich obszarach, jak eksploracja danych (ang. *data mining*), przetwarzanie i analiza wielkich zbiorów danych (ang. *big data*), sztuczna inteligencja (ang. *artificial intelligence* – AI), tworzenie stron internetowych (zarówno po stronie serwera, jak i klienta), programowanie narzędzi dla systemów operacyjnych, analiza numeryczna, automatyzacja zadań, uczenie maszynowe, robotyka itp. W USA już od kilku lat jako najlepszy zawód uznawany jest „data scientist", czyli specjalista od analizy danych, zaś tutaj język Python jest na razie niekwestionowanym liderem, choć przez krótki czas dobrze zapowiadał się także język R. Wniosek końcowy może być tylko jeden – każdy szanujący się programista powinien poznać język Python.

Po ściągnięciu odpowiedniego pliku (w skrypcie obowiązuje stabilna wersja Pythona 3.10.0 z października 2021 roku) ze strony https://www.python.org/downloads/ należy go zainstalować, najlepiej korzystając z opcji Customize i pamiętając m.in. o zaznaczeniu opcji zaawansowanej Add Python to environment variables. Po zainstalowaniu interpretera języka Python możliwe są dwa typy pracy: w interaktywnej powłoce IDLE albo w edytorze.





Po zainstalowaniu Pythona warto też dodać ulubione motywy (ang. *themes*), które uprzyjemniają programowanie, lecz nade wszystko oszczędzają wzrok. Standardowa instalacja umożliwia bowiem wybór między Classic, Dark i New, które można odnaleźć w Options → Configure IDLE → Highlights → Highlighting Theme. Wielu programistów lubi jednak inne motywy np. `Obsidian` czy `Monokai`.



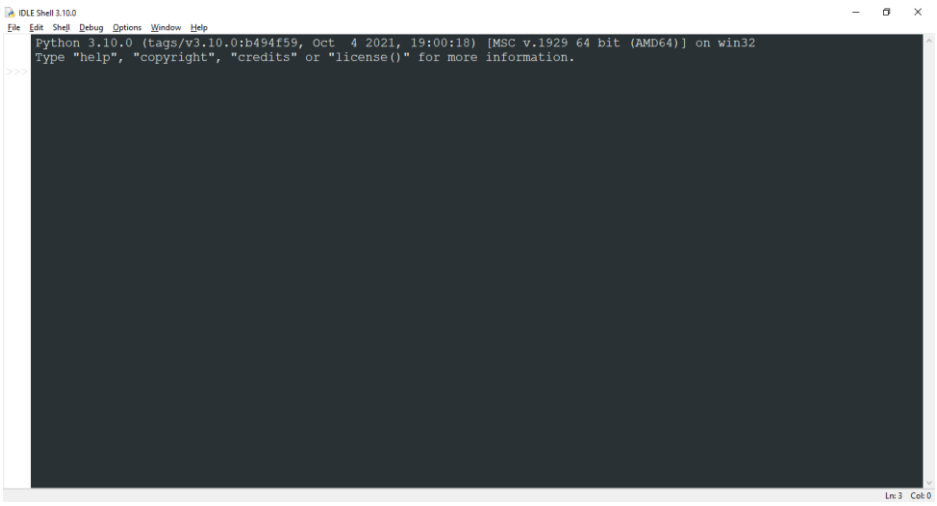
Aby dodać nowy motyw, należy odnaleźć plik konfiguracyjny `config-highlight.def`, w którym znajdują się standardowe motywy i dopisać na końcu nową konfigurację (dotyczy motywów `Monokai` oraz `Obsidian`). Dla przykładu, w systemie operacyjnym MS Windows 10 do pliku tego wiedzie domyślna ścieżka dostępu:

```
c:\Program Files\Python310\Lib\idlelib\
```

Po wyedytowaniu pliku `config-highlight.def` (niezbędne są uprawnienia administratora) należy na samym końcu dopisać (wkleić):

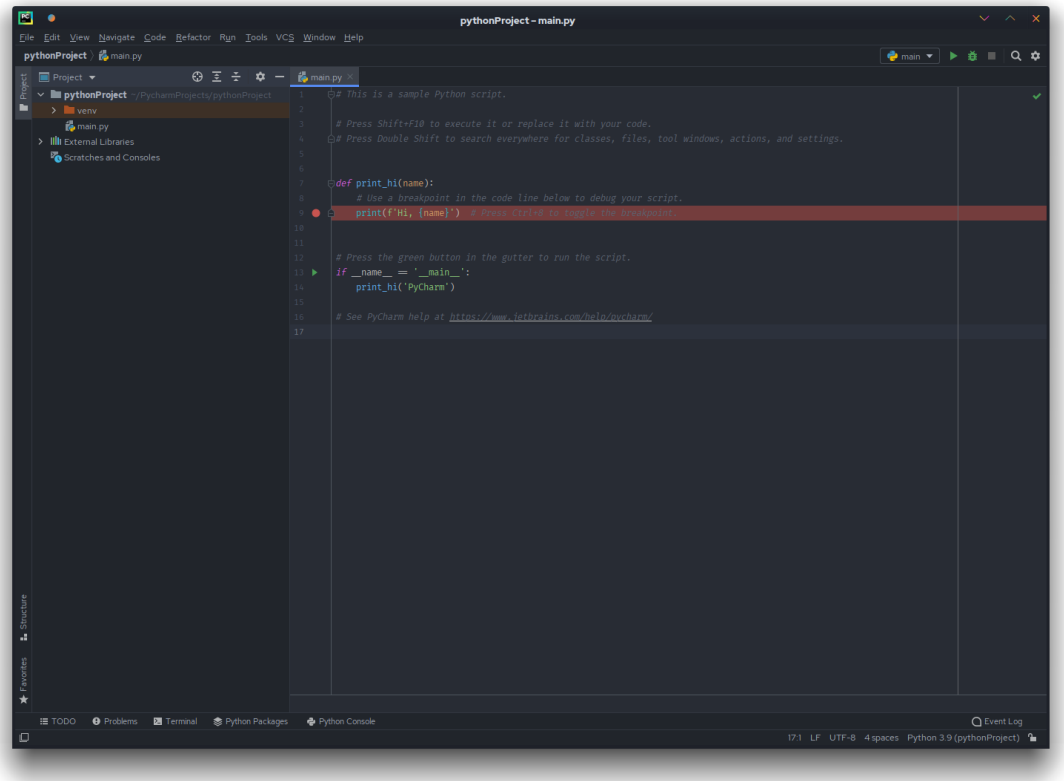
```
[IDLE Monokai]
cursor-foreground    = #ffffff
normal-background    = #272822
normal-foreground    = #f8f8f2
keyword-background   = #272822
keyword-foreground   = #66d9ef
builtin-background   = #272822
builtin-foreground   = #f92672
definition-background = #272822
definition-foreground = #a6e22e
string-background    = #272822
string-foreground    = #e6db74
comment-background   = #272822
comment-foreground   = #75715e
stdout-background    = #272822
```

```
stdout-foreground     = #e6db74
stderr-background     = #272822
stderr-foreground     = #f92672
console-background    = #272822
console-foreground    = #bca3a3
hilite-background     = #3e3d32
hilite-foreground     = #e6db74
error-background      = #ff7777
error-foreground      = #000000
hit-background        = #000000
hit-foreground        = #ffffff
break-background      = #000000
break-foreground      = #ffffff

[IDLE Obsidian]
definition-foreground = #678CB1
error-foreground = #FF0000
string-background = #293134
keyword-foreground = #93C763
normal-foreground = #E0E2E4
comment-background = #293134
hit-foreground = #E0E2E4
builtin-background = #293134
stdout-foreground = #678CB1
cursor-foreground = #E0E2E4
break-background = #293134
comment-foreground = #66747B
hilite-background = #2F393C
hilite-foreground = #E0E2E4
definition-background = #293134
stderr-background = #293134
hit-background = #000000
console-foreground = #E0E2E4
normal-background = #293134
builtin-foreground = #E0E2E4
stdout-background = #293134
console-background = #293134
stderr-foreground = #FB0000
keyword-background = #293134
string-foreground = #EC7600
break-foreground = #E0E2E4
error-background = #293134
```

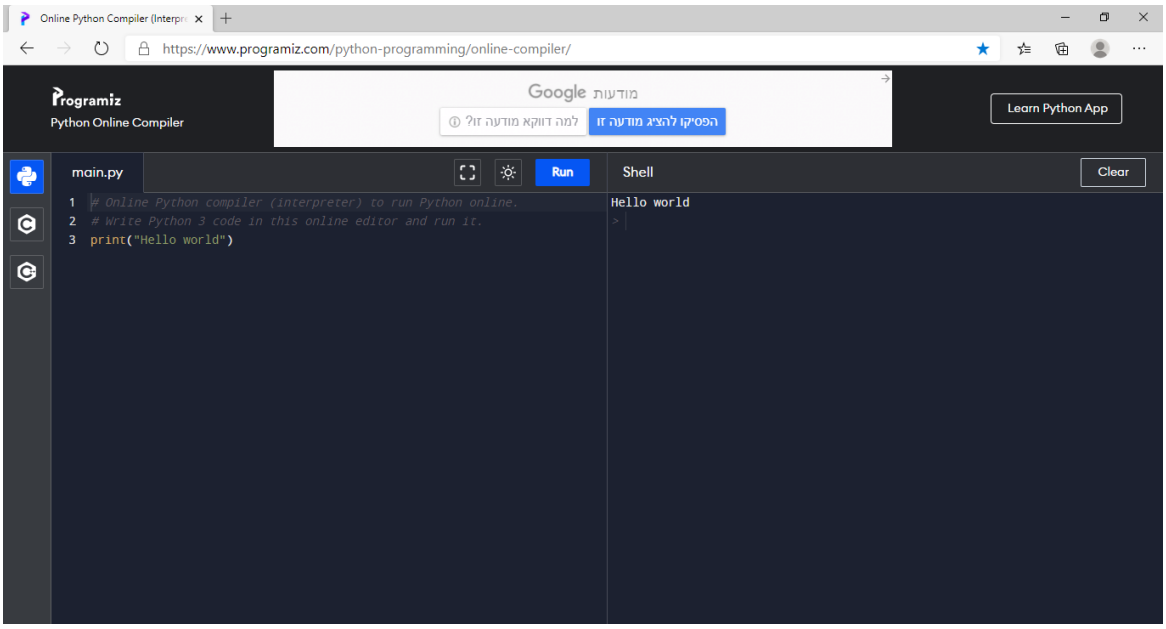


Po zapoznaniu się z językiem Python warto również zastanowić się nad instalacją bardziej profesjonalnego środowiska programistycznego IDE (ang. *Integrated Development Environment*). Do jednych z najlepszych zalicza się m.in. PyCharm firmy JetBrains, które w wersji Community jest bezpłatne i można je pobrać ze strony:

https://www.jetbrains.com/pycharm/download/?source=google&medium=cpc&campaign=14124132618&gclid=EAIaIQobChMI4aHKy6-l9AIVH0CRBR0SOwTREAAYASABEgLivPD_BwE#section=windows



Można również korzystać ze środowisk online, np. https://www.programiz.com/python-programming/online-compiler/, ale należy pamiętać o ich ograniczonych możliwościach.



Praca w interaktywnej powłoce IDLE

Ten tryb pracy wykorzystujemy tylko w celach poznawczych.

```
>>>2 + 2
4


>>>2 + 3 * 6
20

>>>(2 + 3) * 6
30

>>>2 ** 8
256


>>>23 / 7
3.2857142857142856

>>>23 // 7
3

>>>23 % 2
2

>>>


>>>'Adam' + 'Ewa'
```

```
'AdamEwa'

>>>'Adam' + 7
TypeError

>>>'Adam' * 7
'AdamAdamAdamAdamAdamAdamAdam'

>>>jajka = 10
>>>jajka
10

>>>chleb = 5
>>>jajka + chleb
15
```

Nieprawidłowe nazwy zmiennych	**Prawidłowe nazwy zmiennych**
aktualny-bilans (myślnik)	aktualnyBilans
aktualny bilans (spacja)	aktualny_bilans
4konto (cyfra na początku)	konto4
'spam' (apostrofy)	_spam
TOTAL&RESULT (znak specjalny &)	TOTAL_RESULT
żółć (polskie znaki diakrytyczne)	zolc

Operatory porównań w języku Python

Operator	**Znaczenie**	**Przykład użycia**	**Przyjmowana wartość**
==	równa się	**3 == 3**	true
!=	nie równa się	**7 != -5**	true
>	większe niż	**1 > 9**	false
<	mniejsze niż	**3 < 2**	false
>=	więsze lub równe	**6 >= 6**	true
<=	mniejsze lub równe	**11 <= 10**	false

Uwaga:
Nie wolno porównywać zmiennych różnych typów, tzn. nie wolno dokonać np. porównania 1 >= "jeden"! Takie działanie wygeneruje komunikat o błędzie w programie.

```
>>> 42 == '42'
False

>>> 42 == 42.0
True
```

Funkcje logiczne w Pythonie

W Pythonie, jak we wszystkich językach programowania, dostępne są instrukcje zgodne z algebrą Boole'a, w wyniku których otrzymamy wartość logiczną, czyli prawdę (*True*) lub fałsz (*False*). Najlepiej bardzo uważnie przeanalizować je na przykładach, ponieważ bardzo łatwo tutaj o błędy. W dodatku trzeba pamiętać o tzw. operacjach bitowych, których wyniki mogą zaskakiwać, ponieważ te same operatory występują w podwójnych rolach.

Trzeba też pamiętać, że wartość logiczna *False* może być reprezentowana przez wartość liczbową 0, a True przez liczby różne od zera (na przykład 1). Tak więc otrzymujemy:

```
>>> int(True)
1

>>> int(False)
0
>>> bool(0)
False

>>> bool(1)
True

>>> bool(3.14)
True
```

Operator różności:

```
>>> 3 != 2 # lub bool(3 !=2)
True
```

Operator równości

```
>>> 3 == 3
True
```

Operator większości

```
>>> 3 > 3
False

>>> False + False
0
```

(to nie jest suma logiczna, lecz arytmetyczna)

```
>>> True – False
1
```

(to jest operator odejmowania, czyli wykonujemy działanie arytmetyczne 1 – 0)

```
>>> False – True
-1
```

(to jest operator odejmowania, czyli wykonujemy działanie arytmetyczne 0 - 1)

Operator not

```
>>> not False
True
```

```
>>> not 0
True
```
(negacja wartości logicznej False)

```
>>> not -1
False
```
(negacja wartości logicznej różnej od zera, czyli wartości logicznej True)

```
>>> not True
False
```

Operator and (lub &)

```
>>> False and False
False

>>> False & False
False

>>> False and True
False

>>> False & True
False

>>> True and False
False

>>> True & False
False

>>> True and True
True

>>> True & True
True
```

Operator or (lub |)

```
>>> False or False
False

>>> False | False
False

>>> False or True
True

>>> False | True
True

>>> True or False
True
```

```
>>> True | False
True

>>> True or True
True

>>> True | True
True
```

Operator ^ (XOR)

```
>>> False ^ False
False

>>> False ^ True
True
>>> True ^ False
True

>>> True ^ True
False
```

Złożone warunki logiczne

```
>>> 10 > 1 and 10 < 13 != 10
True

>>> x = 2
>>> a = 6
>>> x <= 3 | x > 11 | a != 6 & a > 12
False
```

Operacje bitowe

W języku Python istnieją następujące operacje bitowe, które znakomicie przyspieszają niektóre obliczenia (kolejność operatorów ustawiona jest w tabeli w kierunku rosnącego priorytetu):

Operacja	Opis operacji	Przykład użycia	Wynik
x | y	alternatywa bitowa	42 | 27	59
x & y	koniunkcja bitowa	42 & 27	10
x ^ y	różnica symetryczna bitowa	42 ^ 27	49
x << n	przesunięcie bitowe w lewo o *n* bitów	42 << 1	84
x >> n	przesunięcie bitowe w prawo o *n* bitów	42 >> 1	21
~x	uzupełnienie bitowe zmiennej *x*	~42	-43

Uwagi:

1. Liczby przed użyciem zamieniane są na postać binarną i uzupełniane zerami z lewej strony (do zrównania z większą), a potem bity sprawdzane są odpowiadającymi sobie parami funkcją logiczną OR (po jednym z każdej liczby). Zatem:

$$(42)_{10} = (101010)_2$$

$$(27)_{10} = (011011)_2$$

Zatem

101010 | 011011 = 111011

Zaś $(111011)_2 = (59)_{10}$ i stąd taki wynik.

Wynik działania koniunkcji (funkcja logiczna AND) oraz różnicy symetrycznej (funkcja logiczna XOR, która jednak nie występuje w Pythonie) sprawdź „ręcznie” sam.

2. Przesunięcie bitowe w lewo o 1, 2, 3 itd. oznacza przesunięcie bitów w lewo i uzupełnienie powstałych po przesunięciu pustych miejsc zerami po prawej stronie, co odpowiada operacji mnożenia przez 2, 4, 8 itd. Natomiast przesunięcie bitowe w prawo oznacza operację odwrotną, czyli dzielenie przez 2, 4, 8 itd.
3. Ponieważ w Pythonie (w większości innych języków programowania również) obowiązuje równanie x + ~x = -1, to uzupełnienie bitowe oznacza wykonanie operacji ~x = -1 – x, co wyjaśnia otrzymany wynik. Stąd też np. działanie ~True prowadzi do wyniku -2.

Praca w edytorze IDLE

Przykład 1.
Napisz program obliczający pole prostokąta o bokach podanych przez użytkownika.

Listing:

```
'''Program oblicza
pole prostokąta'''
a = float(input("Wczytaj a: "))    #wczytywanie zmiennej a
b = float(input("Wczytaj b: "))    #wczytywanie zmiennej b
P = a * b                                   #instrukcja przypisania
print(P)                                    #wypisanie wyniku na ekranie
```

Uwagi:

1. Znak # rozpoczyna tzw. komentarz jednowierszowy (obowiązujący wyłącznie do końca linii). Komentarze służą wyłącznie programistom i są całkowicie ignorowane przez kompilatory oraz interpretery. Absolutnie nie należy przesadzać z komentarzami!
2. Aby napisać komentarz wielowierszowy (zawierający więcej niż jedną linię), należy tekst objąć znakami potrójnego apostrofu '''Tutaj wstaw komentarz''' albo znakami potrójnego cudzysłowu """Tutaj wstaw komentarz""".
3. Znak = oznacza instrukcję przypisania, jedną z najważniejszych w każdym języku programowania.
4. Słowo kluczowe float (ang. *floating point*) oznacza, że liczbę, którą wprowadzi użytkownik należy traktować jako zmiennoprzecinkową (rzeczywistą). Liczbę całkowitą deklaruje się za pomocą słowa int (ang. *integer*).
5. Słowo kluczowe input oznacza oczekiwanie na wpisanie przez użytkownika czegokolwiek z klawiatury i wciśnięcie klawisza ENTER. Sekwencję input() można więc wykorzystać też do zatrzymania wykonania programu po to, by np. użytkownik zapoznał się z wynikiem działania programu w konsoli MS Windows.

Przykład 2.
Napisz program, który pyta użytkownika o imię i wiek, a następnie wyświetla dane na ekranie.

```
#program ilustrujący interakcję z użytkownikiem

imie = input("Podaj swoje imię: ")
wiek = int(input("Ile masz lat? "))

print("Masz na imię", imie, "i masz", wiek, "lat")
input("\nAby zakończyć program, naciśnij klawisz Enter.")
```

Przykład 3.
Napisz program obliczający wynik podniesienia dowolnej liczby naturalnej do dowolnej potęgi naturalnej.

```
#program oblicza potęgi liczb naturalnych

x = int(input("Podaj podstawę: "))
y = int(input("Podaj wykładnik: "))

wynik = x ** y          #znaki ** oznaczają potęgowanie

print(x, "do potęgi", y, "=", wynik)
input("\nAby zakończyć program, naciśnij klawisz Enter.")

' ' '
dwuznak: \n
oznacza przejście do nowej linii
' ' '
```

Przykład 4.
Napisz program obliczający pole dowolnego trójkąta korzystając ze wzoru Herona

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

gdzie: S – pole trójkąta; a, b i c – długości boków, p – połowa obwodu trójkąta.

```
#program oblicza pole trójkąta za pomocą wzoru Herona
from math import *

a = float(input("Wprowadź bok a: "))
b = float(input("Wprowadź bok b: "))
c = float(input("Wprowadź bok c: "))

p = (a + b + c) / 2                    #p - połowa obwodu
pole = sqrt(p * (p - a) * (p - b) * (p - c))    #sqrt – pierwiastek kwadratowy

print("Połowa obwodu =", p)
print("Pole trójkąta wynosi:", pole)
input("\nAby zakończyć program, naciśnij klawisz Enter.")
```

"""
math import * – dołączenie do programu całej biblioteki, zawierającej różne funkcje matematyczne (w tym pierwiastkowanie)
"""

Przykład 5.
Napisz program imitujący grę dwoma kośćmi i obliczający sumę wyrzuconych oczek.

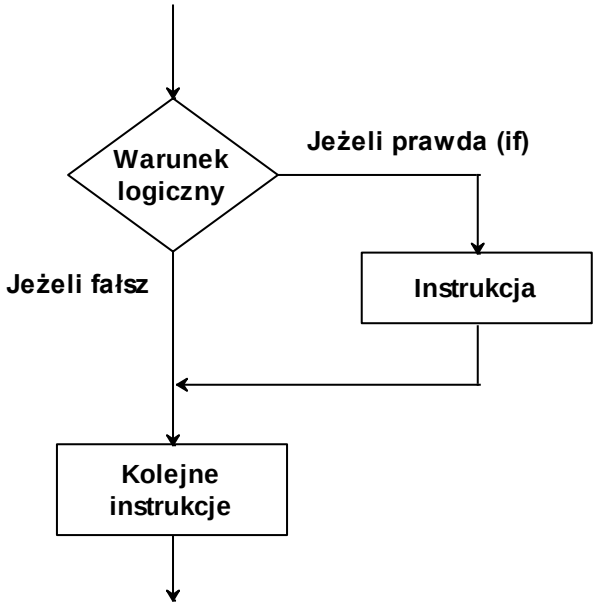
```
import random
kostka_1 = random.randint(1, 6)
kostka_2 = random.randrange(6) + 1

wynik = kostka_1 + kostka_2

print("Wyrzuciłeś", kostka_1, "oraz", kostka_2, "i uzyskałeś wynik =", wynik)
input("\nAby zakończyć program, naciśnij klawisz ENTER")
```

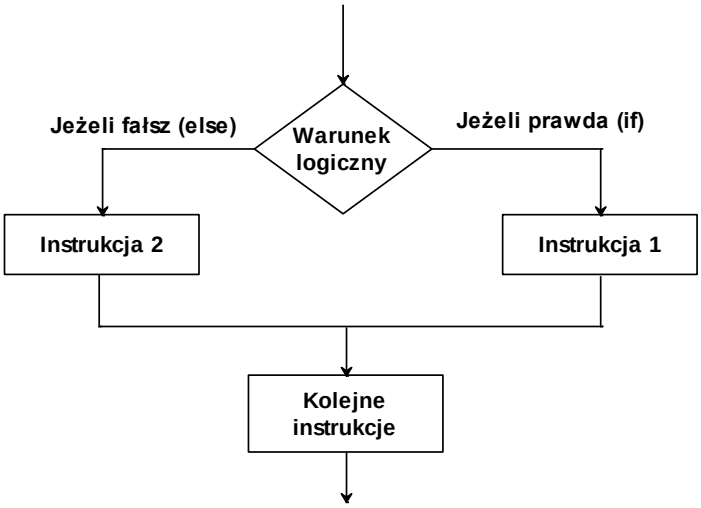
"""
math random – dołączenie do programu modułu o nazwie **random**, zawierającego funkcje związane z generowaniem liczb pseudolosowych

random.randinit(1, 6) – wywołanie funkcji (metody) o nazwie **randinit** z wnętrza modułu **random** w celu wygenerowania liczby z zakresu od 1 do 6

random.randrange(6) + 1 – wywołanie funkcji (metody) o nazwie **randrange** z wnętrza modułu **random** w celu wygenerowania liczby z zakresu od 0 do 5 (dlatego trzeba do niej dodać jeszcze 1), czyli efekt jest taki sam, jak przy użyciu funkcji (metody) **randinit**
"""

Instrukcje warunkowe (decyzyjne) – niepełna i pełna





Przykład 1.
Napisz program, który sprawdza poprawność wprowadzonego hasła (o nazwie „tajne”).

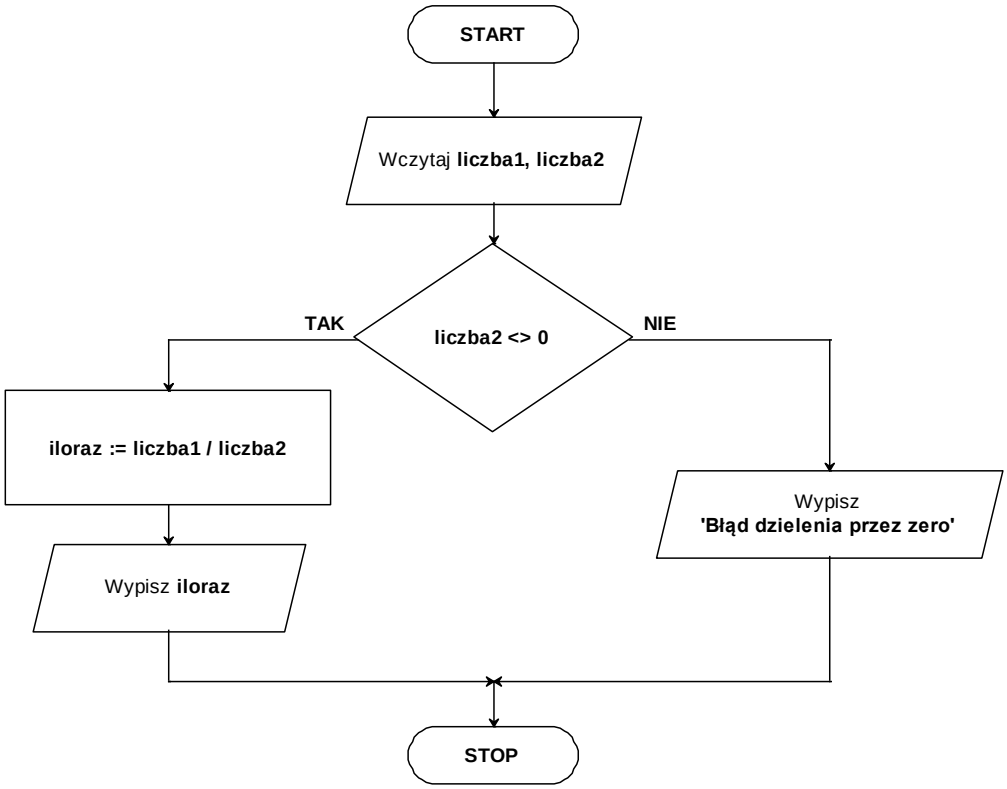
```
#Sprawdzenie poprawności hasła

password = input("Wprowadź hasło: ")

if password == "tajne":
   print("Hasło prawidłowe")
else:
   print("Złe hasło")

"""
biorąc do porównania łańcuch znakowy (ang. string) należy użyć cudzysłowu
"""
```

Przykład 2.
Napisz program umożliwiający dzielenie całkowite z resztą.

```
print("")                                                    #"" oznacza wprowadzenie linii od-
stępu
print("Dzielenie z resztą")
dzielna = int(input("Podaj dzielną: "))          #int - liczba całkowita
dzielnik = int(input("Podaj dzielnik: "))
wynik = dzielna // dzielnik                              #znak // - wynik dzielenia całkowitego
reszta = dzielna % dzielnik                              #znak % - reszta z dzielenia całkowitego

print(dzielna,":",dzielnik,"=",wynik,"r",reszta)

print("\n")                    #dodatkowa linia odstępu dzięki znakowi specjalnemu \n

print("Sprawdzenie poprawności działania:")

dividend = dzielna             #angielskie odpowiedniki
divisor = dzielnik
remainder = reszta
result = wynik
```

```
dividend = result * divisor + remainder          #sprawdzenie poprawności działania

if dividend == dzielna:                          #co ma zrobić, gdy zmienne są równe
  print("Wynik poprawny")
else:                                                    #co ma zrobić w przeciwnym wypadku
   print("Zły wynik")
```

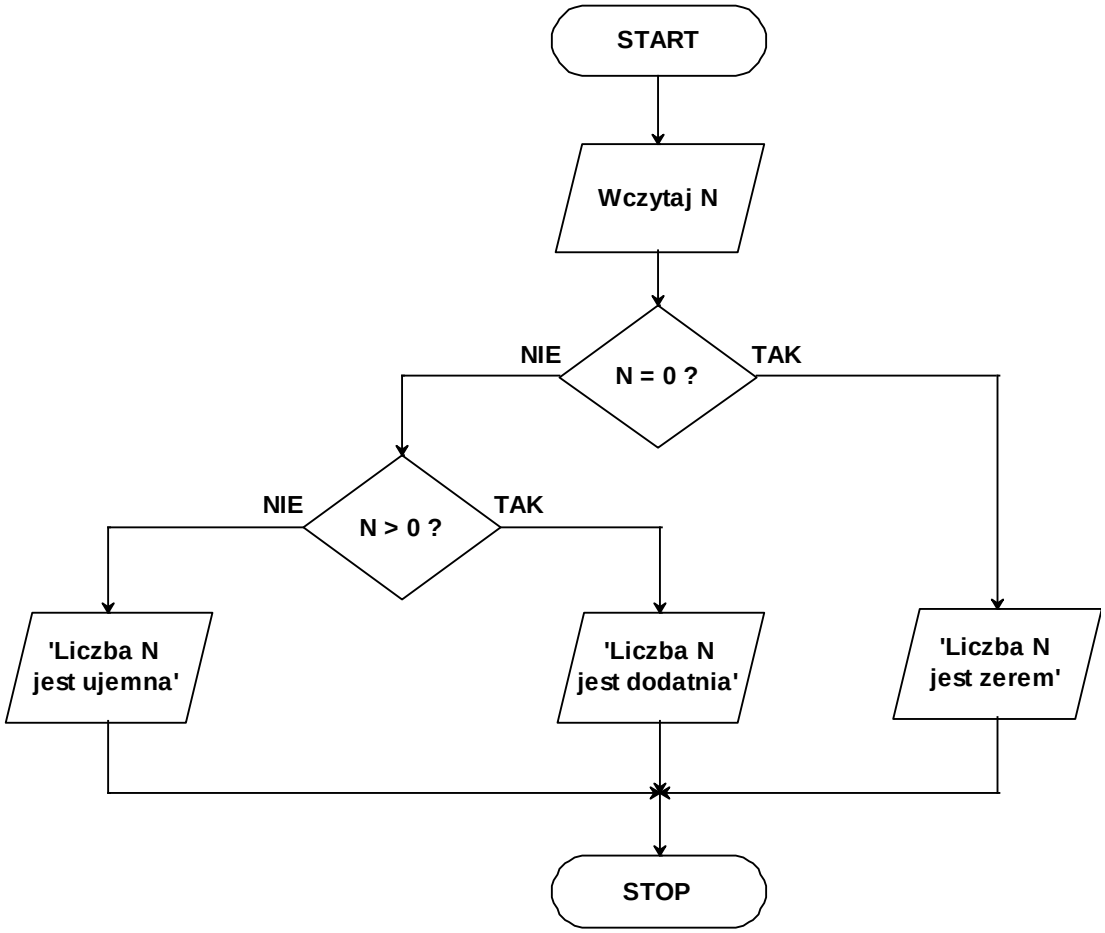
Przykład 3.
Na podstawie schematu blokowego napisz program, który dzieli przez siebie dwie liczby rzeczywiste. Zabezpiecz program przed dzieleniem przez zero.



Rozwiązanie:

```
#dzielenie dwóch liczb rzeczywistych

print("Program dzieli przez siebie dwie liczby rzeczywiste")
liczba1 = float(input("Podaj pierwszą liczbę: "))
liczba2 = float(input("Podaj drugą liczbę: "))

if liczba2 != 0:
   iloraz = liczba1 / liczba2
   print(liczba1, "/", liczba2,"=", iloraz)
else:
   print("Błąd dzielenia przez zero")

input("\nAby zakończyć program, naciśnij klawisz Enter.")
```

Przykład 4.
Opracuj schemat blokowy algorytmu, który wczytuje z klawiatury dowolną liczbę całkowitą N i bada jej znak, tzn. określa, czy jest ona dodatnia, ujemna lub jest zerem. Sprawdź działanie algorytmu w w Pythonie.



Listing w Pythonie:

```
#program bada znak liczby
N = int(input("Podaj liczbę: "))

if N == 0:
   print("Liczba", N, "jest zerem")
elif N > 0:                    #jeżeli w przeciwnym wypadku
   print("Liczba", N, "jest dodatnia")
else:                          #w przeciwnym wypadku
   print("Liczba", N, "jest ujemna")
input("\nAby zakończyć program, naciśnij klawisz Enter.")
```

Przykład 5.
Skonstruuj algorytm znajdujący minimalną z trzech liczb *a, b, c* oraz zapisz algorytm w postaci listy kroków i schematu blokowego. Przetestuj jego działanie za pomocą przykładowego zestawu liczb. Następnie samodzielnie napisz program w języku Python realizujący ten algorytm.

Rozwiązanie:
Na ogół proste problemy algorytmiczne mają niewiele różnych rozwiązań, trudniejsze można rozwiązać na dużo sposobów. Oto najlepszy z nich, wykorzystujący tzw. zmienną pomocniczą.

Lista kroków może mieć postać:

a) podaj trzy dowolne liczby *a, b, c,*

b) przypisz zmiennej *x* wartość równą *a*,

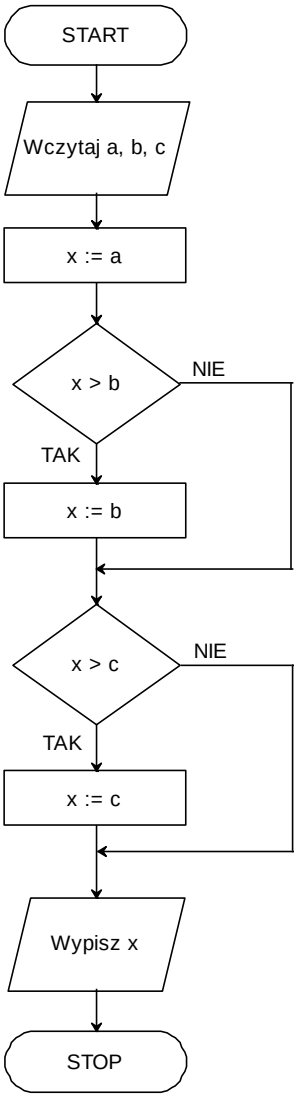
c) porównaj *x* z *b*,

d) jeśli *x* > *b*, to zmiennej *x* przypisz wartość równą *b*. W przeciwnym wypadku wartość zmiennej *x* nie ulega zmianie,

e) porównaj *x* z *c*,

f) jeśli *x* > *c*, to zmiennej *x* przypisz wartość równą *c*. W przeciwnym wypadku wartość zmiennej *x* nie ulega zmianie,

g) koniec postępowania. Wartość zmiennej *x* oznacza rozwiązanie zadania.



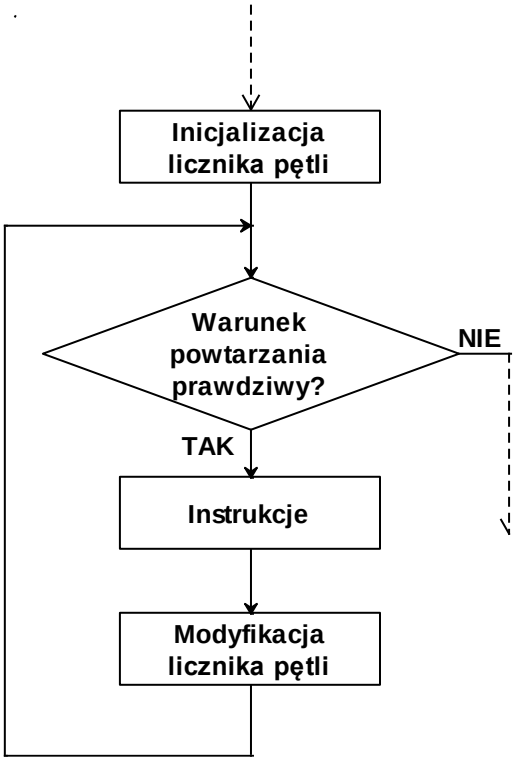
Przykładowy zestaw liczb do testowania algorytmu musi zawierać wszystkie możliwe relacje, jakie zachodzą między liczbami. Tylko w ten sposób można sprawdzić, czy algorytm daje poprawne wyniki niezależnie od zestawu wprowadzonych liczb.

Relacje zachodzące między liczbami a, b, c	Przykładowe zestawy liczb a, b, c
$a < b \wedge a < c$	{-1, 2, 3}
$a < b \wedge c = a$	{-1, 3, -1}
$a < b \wedge c > a \wedge c < b$	{3, 7, 6}
$a < b \wedge c = b$	{1, 5, 5}
$a < b \wedge b < c$	{1, 2, 3}
$a = b \wedge c < a$	{5, 5, -2}
$a = b \wedge c = a$	{4, 4, 4}
$a = b \wedge c > a$	{-1, -1, 0}
$a > b \wedge c < b$	{4, 2, 1}
$a > b \wedge c = b$	{4, 2, 2}
$a > b \wedge c > b \wedge c < a$	{5, 1, 3}
$a > b \wedge c = a$	{7, 3, 7}
$a > b \wedge c > a$	{4, 2, 8}

Program w języku Python:

```
#wyszukiwanie najmniejszej liczby z trzech
a = int(input("Podaj pierwszą liczbę: "))
b = int(input("Podaj drugą liczbę: "))
c = int(input("Podaj trzecią liczbę: "))

x = a

if x >= b:
   x = b
if x > c:
   x = c
print("Najmniejsza z wprowadzonych liczb to ", x)

input("\nAby zakończyć program, naciśnij ENTER")
```

Pętle programowa for



Przykład 1.
Napisz program, który ilustruje działanie pętli for.

```
#ilustracja działania pętli for
for i in range(1, 11):
   print("Pętla nr ", i)
```

Przykład 2.
Napisz program, który wypisze na ekranie poziomo liczby od 0 do 10, od 10 do 1 oraz od 5 do 50 z krokiem równym 5.

```
#wypisywanie liczb w pętli for

print("\nLiczby od 0 do 10")
for i in range(11):
   print(i, end=" ")
#range(11) - wypisuje liczby całkowite z zakresu <0; 11)

print("\n\nLiczby od 10 do 1")
for i in range(10, 0, -1):
   print(i, end=" ")
'''
range(10, 0, -1)
wypisuje liczby całkowite z zakresu <10; 0) z krokiem -1
'''
```

```
print("\n\nLiczby od 0 do 50, co 5")
for i in range(5, 51, 5):
   print(i, end=" ")
'''
range(5, 51, 5)
wypisuje liczby całkowite z zakresu <5; 51) z krokiem 5
'''
```

Przykład 3.
Napisz program, który sumuje liczby naturalne od 1 do zadanej przez użytkownika wartości n.

```
#sumowanie liczb naturalnych – pętla for

zakres = int(input("Podaj górną granicę sumowania: "))
suma = 0

for i in range(1, zakres + 1):
   suma += i                                   #to samo, co suma = suma + i

print("Suma liczb od 1 do", zakres, "=", suma)
```

Przykład 4.
Napisz program, który oblicza silnię liczby naturalnej n ($n! = 1 \cdot 2 \cdot 3 \cdots n$). Z definicji $0! = 1$.

```
#Iteracyjne obliczanie silni

print("Program oblicza silnię liczby naturalnej n")
n = int(input("Podaj liczbę naturalną n: "))
silnia = 1

for i in range(1, n + 1):
   silnia = silnia * i
   i += 1                    #i += 1 skrócony zapis i = i + 1

print("Silnia liczby", n, "=", silnia)
```

Przykład 5.
Napisz program, który wypisuje na ekranie litery wprowadzonego przez użytkownika słowa („rozstrzelone” w poziomie oraz w pionie).

```
#wypisywanie liter w pętli for

word = input("Wprowadź jakieś słowo: ")

print("\nOto poszczególne litery w słowie (w poziomie i w pionie)")

for letter in word:
   print(letter, end=" ")
'''
```

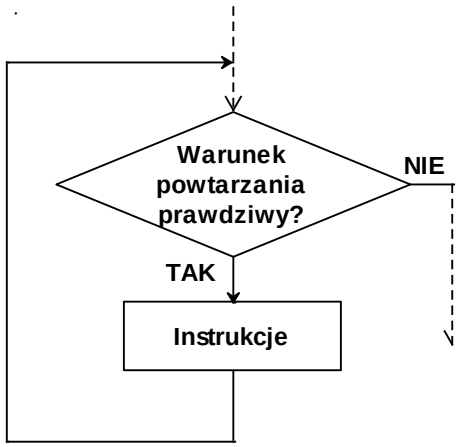
```
pętla wykonana się tyle razy, ile jest liter w wyrazie,
a na końcu każdej litery dopisana zostanie spacja
'''

print("\n") #wiersz odstępu

for letter in word:
   print(letter)
```

Przykład 6.
Napisz program ilustrujący działanie pętli for zagnieżdżonej trzykrotnie, tj. pętli for w pętli for.

```
#ilustracja zagnieżdżonej pętli for
for i in range(1, 4):
   print("Jestem w pętli zewnętrznej nr", i)
   for j in range(1, 4):
      print("\tJestem w pętli wewnętrznej nr", j)
```

Ćwiczenie
Napisz program realizujący tabliczkę mnożenia 10 x 10.

Pętla programowa while



Przykład 1.
Napisz program, w którym użytkownik ma zgadnąć liczbę losową z zakresu od 1 do 100. Po zgadnięciu program ma wypisać na ekranie, za którym razem użytkownikowi udało się zgadnąć.

```
#gra w zgadywanie liczby

import random

liczba_wylosowana = random.randint(1, 100)
```

```
liczba = 0 #nadanie wartości początkowej liczbie podanej przez użytkownika
proba = 0 #nadanie wartości początkowej liczbie prób zgadnięcia

while liczba != liczba_wylosowana:
   proba += 1                              #to samo, co proba = proba + 1
   liczba = int(input("Zgadnij liczbę z zakresu od 1 do 100: "))
   if liczba == liczba_wylosowana:
      print("Zgadłeś za", proba, "razem")
   elif liczba > liczba_wylosowana:
      print("Za duża")
   else:
      print("Za mała")
```

Przykład 2.
Zastąp pętlę for pętlą while w programie do sumowania liczb naturalnych od 1 do zadanej przez użytkownika liczby n.

```
#sumowanie liczb naturalnych – pętla while

zakres = int(input("Podaj górną granicę sumowania: "))
suma = 0
i = 1

while i < 101:
   suma += i
   i += 1
print("Suma liczb od 1 do", zakres, "=", suma)
```

Przykład 3.
Wykorzystując pętlę for zmodyfikuj program w zgadywanie liczby w ten sposób, by użytkownik sam wybrał ile może wykonać prób. Zastosuj odpowiednie komunikaty dla użytkownika (także w wypadku niepowodzenia).

```
#gra w zgadywanie liczby

import random

liczba_wylosowana = random.randint(1, 100)
print(liczba_wylosowana)

ile_prob = int(input("Ile prób?"))

proba = 0 #nadanie wartości początkowej liczbie prób odgadnięcia liczby

for i in range(1, ile_prob + 1):
   proba += 1
   liczba = int(input("Zgadnij liczbę z zakresu od 1 do 100: "))
   if liczba > liczba_wylosowana:
      print("Za duża")
   elif liczba < liczba_wylosowana:
      print("Za mała")
```

```
    else:
        break      #przerwanie pętli w wypadku zgadnięcia liczby

if liczba == liczba_wylosowana:
    print("Gratulacje! Zgadywana liczba to rzeczywiście", liczba_wylosowana)
    print("Zgadłeś za", proba, "razem")
else:
    print("Niestety, niepowodzenie")
```

Przykład 4.
Napisz program, który wypisze na ekranie poziomo liczby od 0 do 10, od 10 do 1 oraz od 5 do 50 z krokiem równym 5.

```
#wypisywanie liczb w pętli for

print("\nLiczby od 0 do 10")
for i in range(11):
    print(i, end=" ")
#range(11) - wypisuje liczby całkowite z zakresu <0; 11)

print("\n\nLiczby od 10 do 1")
for i in range(10, 0, -1):
    print(i, end=" ")
'''
range(10, 0, -1)
wypisuje liczby całkowite z zakresu <10; 0) z krokiem -1
'''

print("\n\nLiczby od 0 do 50, co 5")
for i in range(5, 51, 5):
    print(i, end=" ")
'''
range(5, 51, 5)
wypisuje liczby całkowite z zakresu <5; 51) z krokiem 5
```

Przykład 5.
Napisz program rozwiązujący równanie liniowe.

```
a = int(input("Podaj a: "))
b = int(input("Podaj b: "))
if a != 0:
    x = -b / a
    print("x = ", x)
elif b == 0:
    print("nieskończenie wiele rozwiązań")
else: print("równanie sprzeczne")
```

Przykład 6.
Napisz program, który oblicza średnią ocen. Wpisanie przez użytkownika dowolnej liczby ujemnej kończy obliczenia.

```
liczba_ocen = 0
srednia = 0
ocena = 0
while ocena >= 0:
   ocena = float(input("Podaj ocenę: "))
   if ocena > 0:
      liczba_ocen += 1
      srednia += ocena
print("Średnia Twoich ocen wynosi:", round(srednia / liczba_ocen, 2))
```

Przykład 7.
Napisz program rozwiązujący układ dwóch równań z dwiema niewiadomymi.

```
a11 = float(input("Podaj współczynnik a11: "))
a12 = float(input("Podaj współczynnik a12: "))
b1 = float(input("Podaj współczynnik b1: "))
a21 = float(input("Podaj współczynnik a21: "))
a22 = float(input("Podaj współczynnik a22: "))
b2 = float(input("Podaj współczynnik b2: "))
W = a11 * a22 - a12 * a21
Wx = b1 * a22 - a12 * b2
Wy = a11 * b2 - b1 * a21

if W == Wx == Wy == 0:
   print("Układ nieoznaczony")
elif W == 0:
   print("Układ sprzeczny")
else:
   print("x = ", round(Wx / W, 2))
   print("y = ", round(Wy / W, 2))
input()
```

Przykład 8.
Napisz program, który odlicza od 10 do 1 (ang. *final countdown*).

```
import time
a = 10
while a > 0:
   print(a)
   time.sleep(1)
   a -= 1
print("Koniec odliczania")
```

Przykład 9.
Napisz program, który na podstawie kodu ASCII przedstawi kolejne litery litery alfabetu w porządku naturalnym i odwróconym.

```
print("Alfabet w porządku naturalnym:")
x = 0
for i in range(65, 91):
```

```
    litera = chr(i)
    x += 1
    tmp = litera + " => " + litera.lower()
    if i > 65 and x % 5 == 0:
        x = 0
        tmp += "\n"
    print(tmp, end=" ")

x = -1
print("\nAlfabet w porządku odwróconym:")
for i in range(122, 96, -1):
    litera = chr(i)
    x += 1
    if x == 5:
        x = 0
        print("\n", end=" ")
    print(litera.upper(), "=>", litera, end=" ")
```

Ćwiczenia

Ćwiczenie 1.
Napisz program, który oblicza pierwiastki dowolnego stopnia z liczby rzeczywistej (bez korzystania z dodatkowych funkcji języka Python).

Ćwiczenie 2.
Używając liczb całkowitych napisz program przeliczający temperaturę wyrażoną w stopniach Celsjusza T_C w zakresie od $0°$ do $100°$ na skalę Fahrenheita T_F z krokiem $10°$ według wzoru:

$$T_F = 32 + \frac{9}{5} \cdot T_C$$

Ćwiczenie 3.
Napisz program, którego zadaniem jest wygenerowanie *n-tego* wyrazu ciągu Fibonacciego.

Ciąg Fibonacciego opisany jest następującym wzorem matematycznym:

$$F_n = \begin{cases} 0 & dla\ n = 0 \\ 1 & dla\ n = 1 \\ F_{n-1} + F_{n-2} & dla\ n > 1 \end{cases}$$

Pierwszych dziesięć wyrazów ciągu Fibonacciego to:

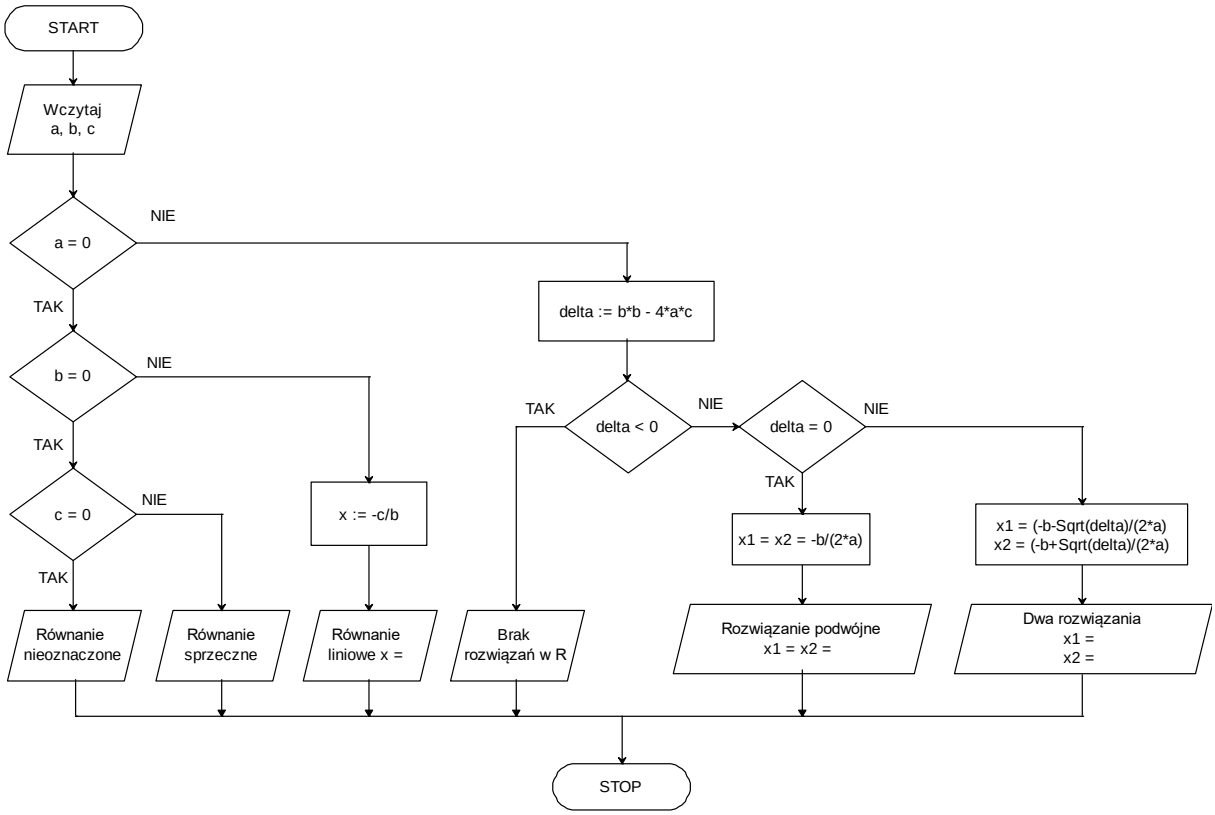
F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9
0	**1**	**1**	**2**	**3**	**5**	**8**	**13**	**21**	**34**

Zadania domowe

Z1.
Napisz program, który generuje wszystkie liczby doskonałe w zakresie od 1 do granicy podanej przez użytkownika. Liczba doskonała to taka liczba naturalna, która jest sumą wszystkich swoich dzielników właściwych, tj. mniejszych od niej samej. Przykładowo, taką liczbą jest 28 $(28 = 14 + 7 + 4 + 2 + 1)$.

Z2.
Napisz program, który generuje wszystkie liczby zaprzyjaźnione w zakresie od 1 do granicy podanej przez użytkownika. Liczby zaprzyjaźnione stanowi para liczb naturalnych spełniających warunek, że suma dzielników właściwych każdej z tych liczb równa się drugiej liczbie samej (np. 220 i 284 są zaprzyjaźnione). Napisz program sprawdzający, czy dane dwie liczby naturalne są zaprzyjaźnione.

Z3.
Napisz program, który rozwiązuje równanie kwadratowe dokładnie według poniższego schematu blokowego.



Z4.
Napisz program wyznaczający pierwiastek kwadratowy z danej liczby P korzystając z opisanego poniżej algorytmu Herona (ok. $10 - 70$ n.e.), który w założeniu polega na poszukiwaniu boku kwadratu, gdy dana jest jego powierzchnia P.

Algorytm Herona
W pierwszym kroku zakładamy na przykład, że bok kwadratu P wynosi po prostu $a_1 = \frac{P}{2}$. Jeśli $a_1 < \sqrt{P}$, to $\frac{P}{a_1} > \sqrt{P}$ i na odwrót.

Zatem dokładna wartość pierwiastka leży między wartościami a_1 i $\frac{P}{a_1}$, czyli kolejne przybliżenie pierwiastka może być wzięte jako średnia arytmetyczna obu krańców przedziału: $a_2 = \frac{a_1+\frac{P}{a_1}}{2}$. Zaś następne przybliżenia można opisać ogólnym wzorem $a_{n+1} = \frac{a_n+\frac{P}{a_n}}{2}$. W obliczeniach numerycznych konieczne jest określenie dokładności obliczeń, do której używa się litery ε (gr. *epsilon*), np. $\varepsilon = 10^{-3}$. Zatem po każdym kroku n należy sprawdzić, czy zadana dokładność jest już osiągnięta poprzez spełnienie warunku $\left|a_n - \frac{P}{a_n}\right| < \varepsilon$.