python

tutorial version 1.15

popularny, interpretowany język programowania ogólnego przeznaczenia

twórcą języka jest holenderski programista **Guido van Rossum**

Python

dynamicznie typowany

używany m.in. do tworzenia:

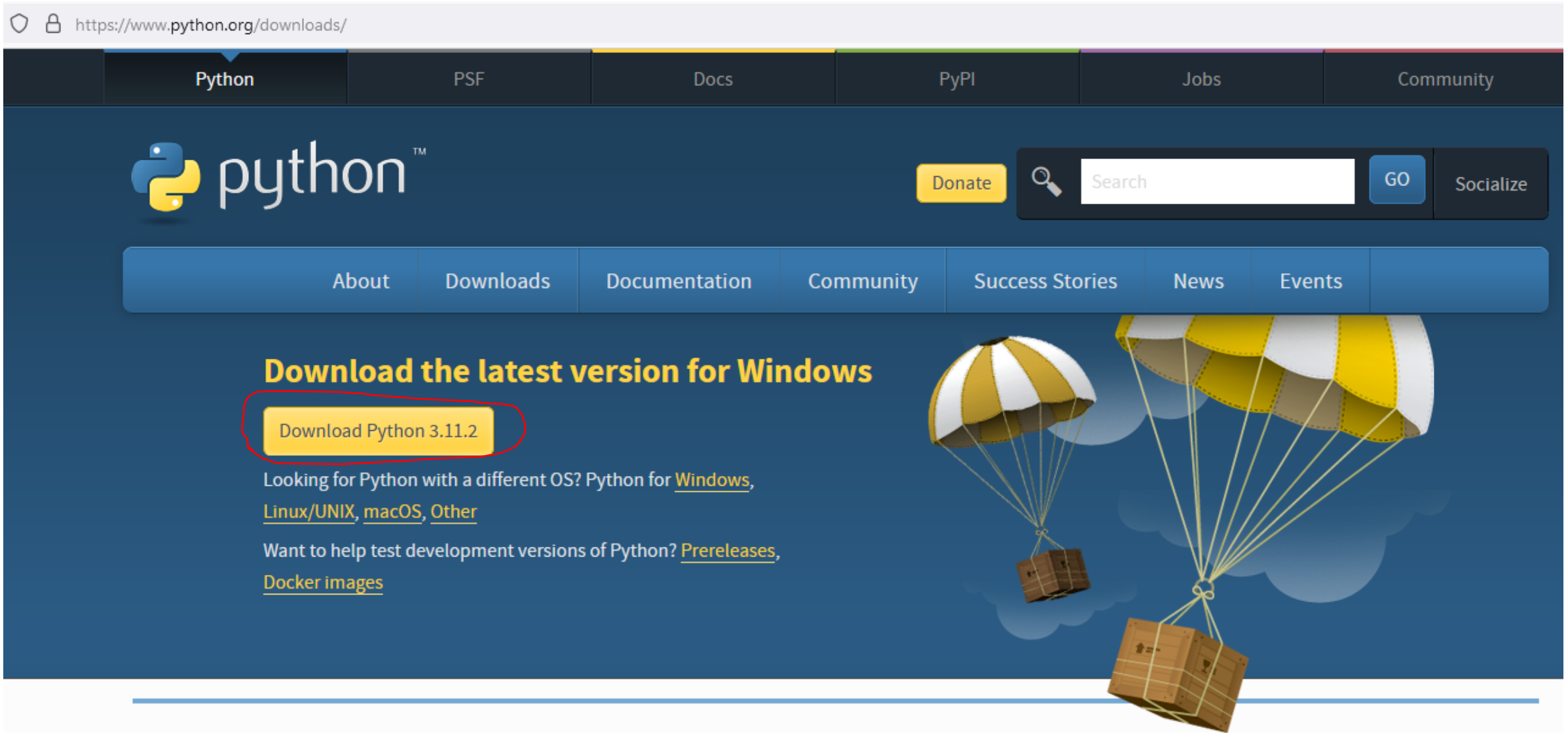
- stron internetowych (po stronie serwera),
- programów desktopowych i mobilnych,
- skryptów systemowych.

główną wersją języka jest Python 3

kładzie nacisk na czytelność kodu przy użyciu wcięć

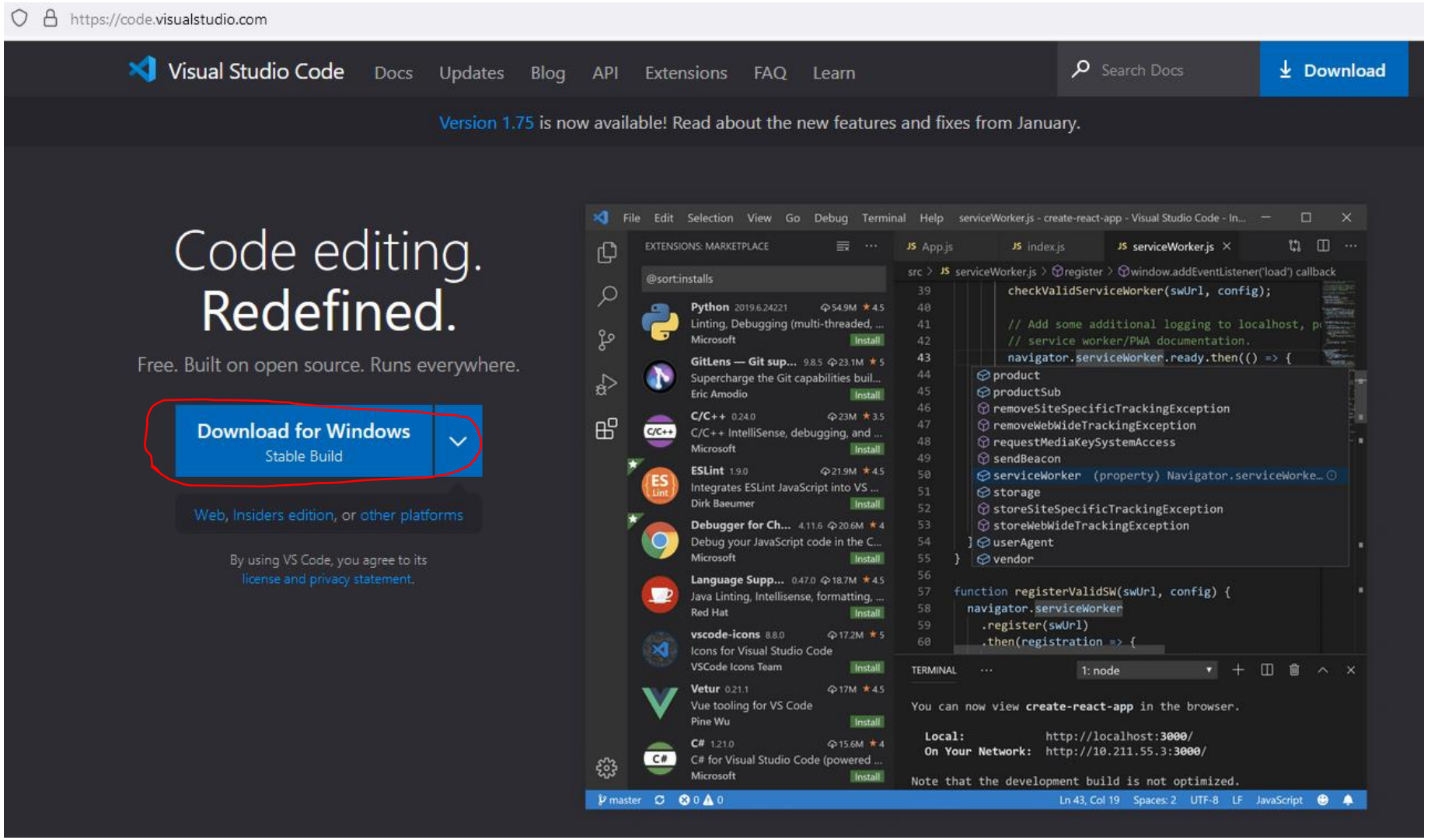
został zaprojektowany z myślą o czytelności i ma pewne podobieństwa do języka angielskiego z wpływem matematyki.

Python download



https://www.python.org/downloads/

Visual Studio Code

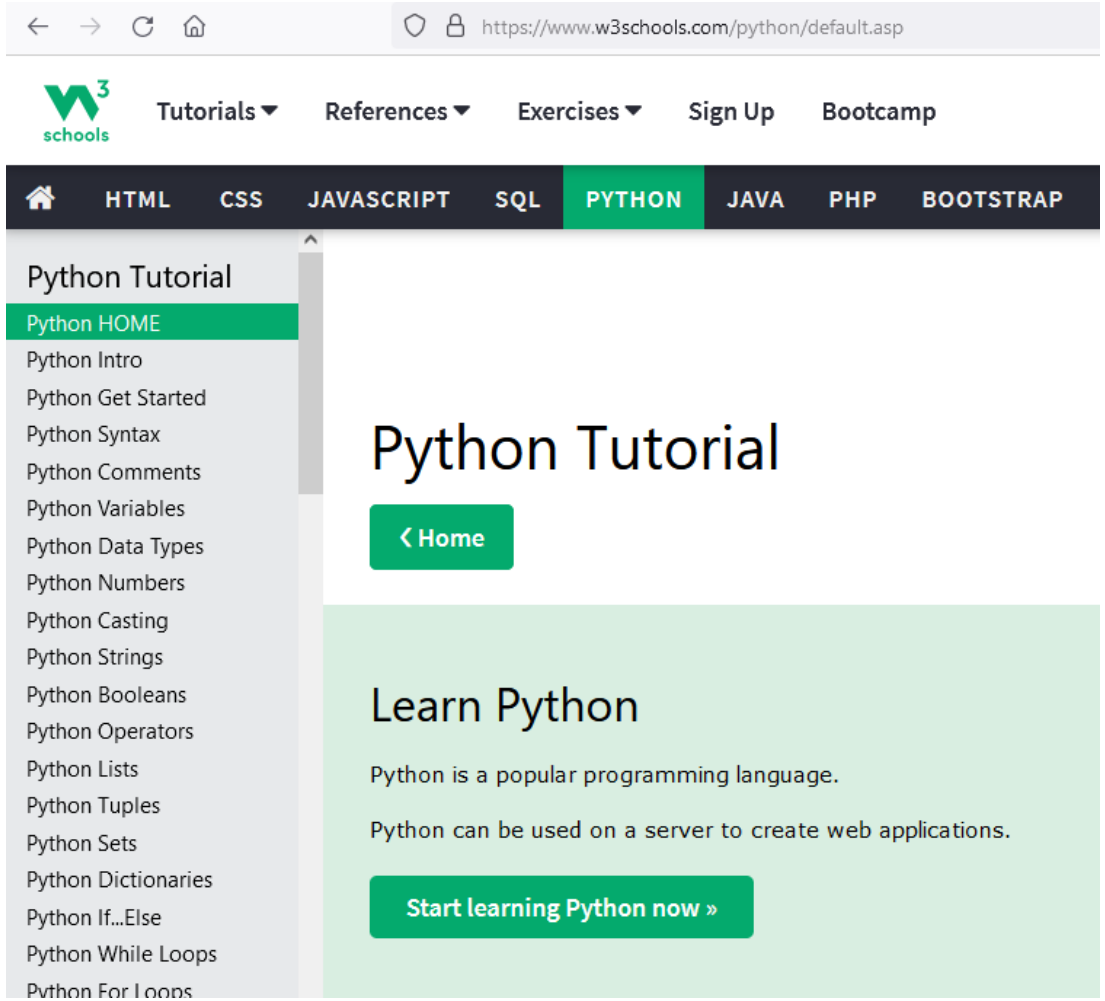


https://code.visualstudio.com/

Python extension for Visual Studio Code

https://code.visualstudio.com/docs/python/python-tutorial

tutorial

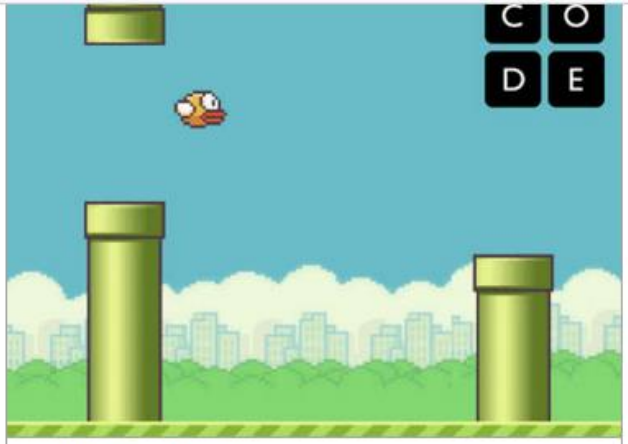


https://www.w3schools.com/python/default.asp

godzina kodowania

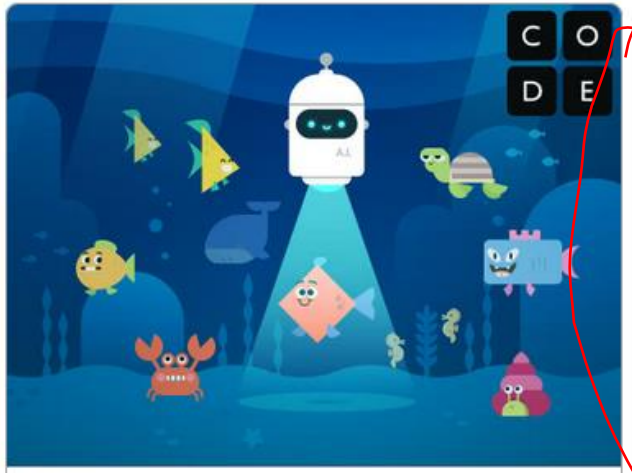
https://hourofcode.com/pl/learn

Minecraft Timecraft
Klasy 2+ | Bloki, Python



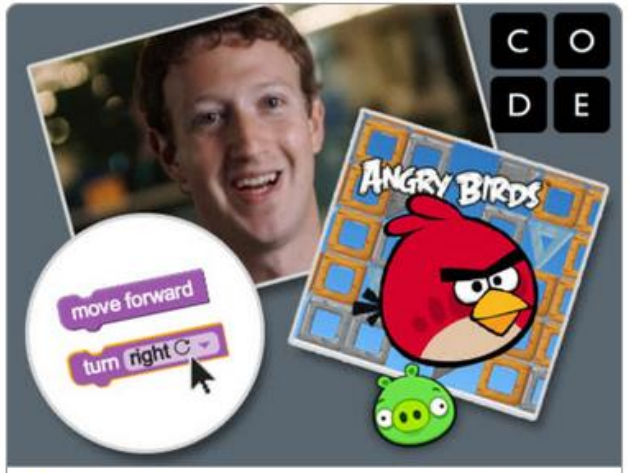
Stwórz grę Flappy
Klasy 2+ | Bloki

Gobliny i Chwała
Klasy 6-8 | JavaScript, Python



AI dla oceanów
Klasy 3+ | AI [sztuczna inteligencja] i nauka maszynowa

Odkryj Python z Compute it
Klasy 2+ | Python | Wszystkie współczesne przeglądarki



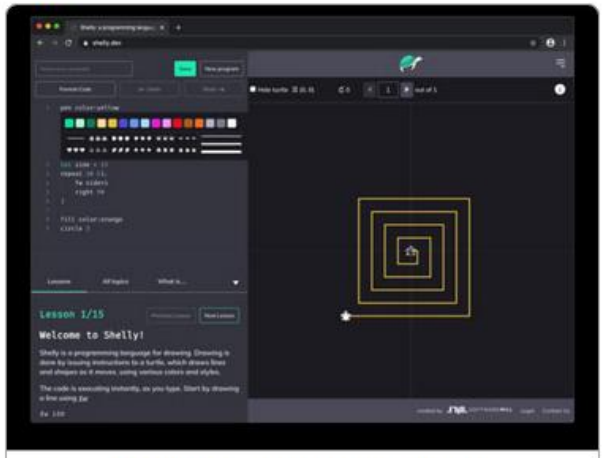
Napisz swój pierwszy program komputerowy
Klasy 2+ | Bloki

https://hourofcode.com/pl/learn

godzina kodowania

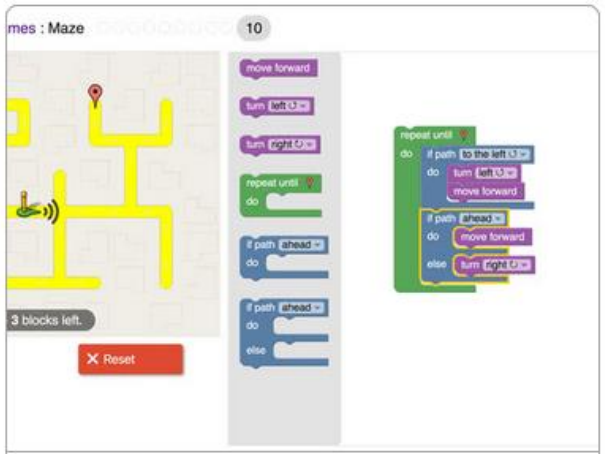
https://hourofcode.com/pl/learn

Anything is Possible! Storytelling with Scratch
Klasy 2-8 | Bloki



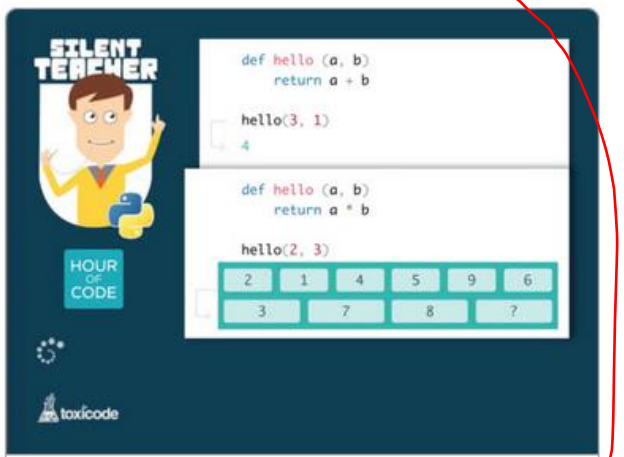
Shelly: Learn programming by drawing!
Klasy 2-8 | A Logo-like language

Kosmiczna rozmowa: Wystrzel w kodzenie w ...
Klasy 2-8 | Bloki, Scratch



Gry w Blockly
Klasy 6+ (od 11 lat) | Bloki

Galaxy Game Jam
Klasy 6+ (od 11 lat) | Bloki | Android



Odkryj Pythona z Silent Teacher
Klasy 6+ (od 11 lat) | Python

https://hourofcode.com/pl/learn

klocki

zmienne
(kontenery do przechowywania danych)

```
a = 5
imie = "Ania"
```

instrukcja warunkowa

```
a = 40
b = 30
if a > b:
  print("a jest większe")
elif a < b:
  print("b jest większe")
else:
  print("a jest równe b")
```

pętla while

```
i = 1
while i < 5:
  print(i)
  i += 1
```

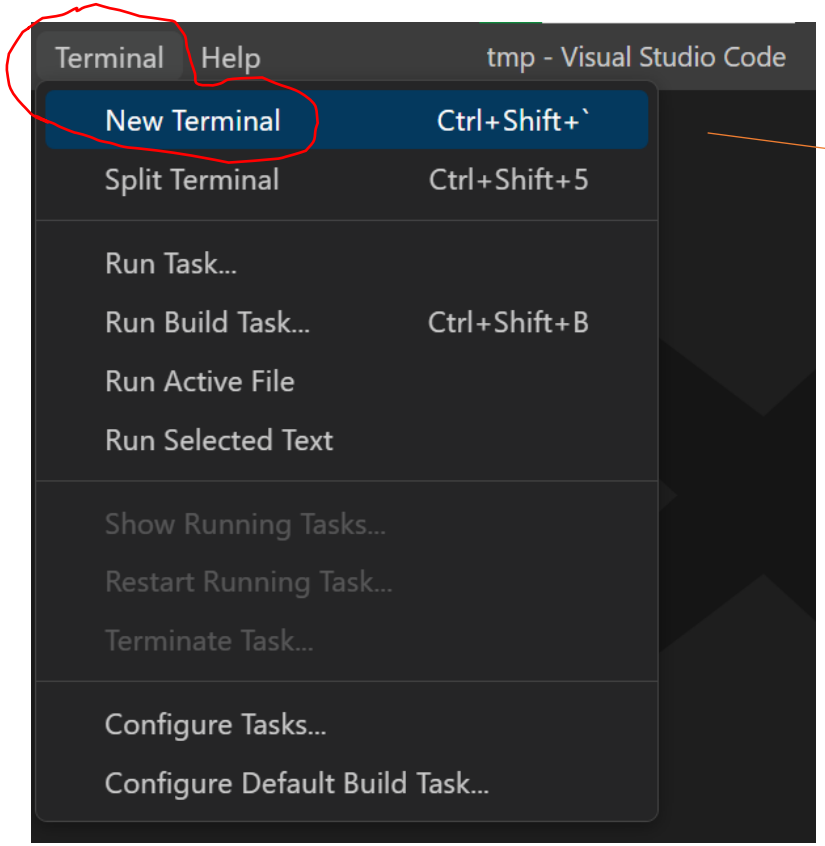
pętla for

```
for x in range(5):
  print(x)
```

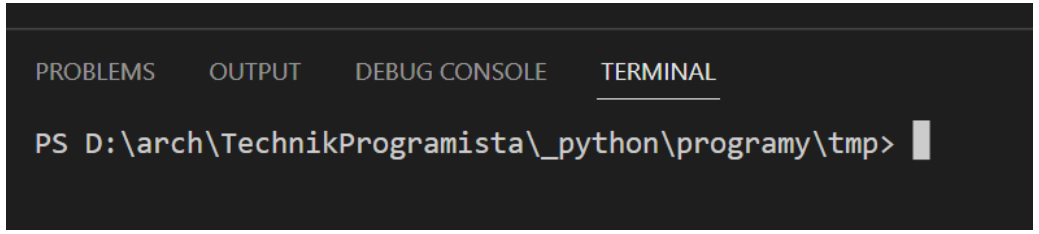
funkcja

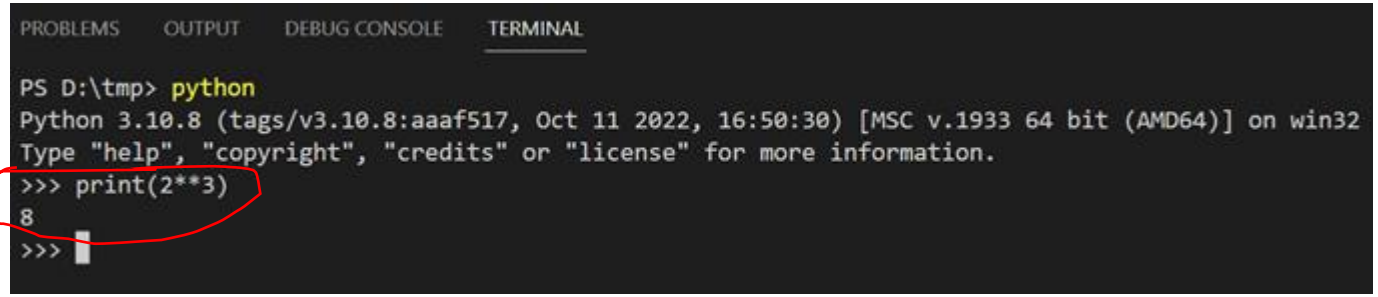
```
def fun():
  print("Hello World")
```

praca w terminalu



Visual Studio Code





zmienne

zmienne są pojemnikami na dane,
które używamy w nszym programie

operator przypisania

nie podajemy nazwy typu
przy nazwie zmiennej -
typowanie dynamiczne
(w zależności od przypisanej wartości interpreter określa typ zmiennej)

liczba = 1

wielkość liter ma znaczenie
(Liczba to nie to samo co liczba)

definicja zmiennej o nazwie *liczba*
i przypisanie jej wartości *1*

```
>>> liczba = 1
>>> liczba
1
>>> print(type(liczba))
<class 'int'>
```

w trakcie działania programu
można zmieniać typ zmiennej

typ zmiennej *int - liczba całkowita*

zmienne – typy danych

```
PS D:\tmp> python
Python 3.10.8 (tags/v3.10.8:aaaf517, Oct 11 2022, 16:50:30) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 4
>>> type(a)
<class 'int'>
>>>
>>> b=3.14
>>> type(b)
<class 'float'>
>>>
>>> c="Marian"
>>> type(c)
<class 'str'>
>>>
>>> type(B)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'B' is not defined. Did you mean: 'b'?
```

zmienna typu *int – liczba całkowita*

Python nie ma ograniczenia dla liczb całkowitych (jak to się dzieje w innych językach), które mogą być tak duże, jak duża jest pamięć, która je przechowuje.

zmienna typu *float – liczba rzeczywista*

zmienna typu *string – ciąg znaków*

ciąg znaków może być zamknięty w cudzysłowach lub apostrofach

```
>>> d=True
>>> type(d)
<class 'bool'>
```

zmienna *logiczna – True lub False*

wielkość liter ma znaczenie

https://www.w3schools.com/python/python_datatypes.asp

string

w potrójnym cudzysłowie możemy umieścić ciąg tekstowy w kilku linijkach

```
>>> komp = """
... komputer
... jest
... mocny
... """
>>> komp
'\nkomputer\njest\nmocny\n'
```

List (lista)

uporządkowana kolekcja danych różnych typów
kolejność ma znaczenie

lista może zawierać
różne typy danych
(również inne listy)

elementy listy podajemy w nawiasach kwadratowych

```
>>> lista = ["komputer", "liczba", 3.14, 26, False]
>>> lista[0]
'komputer'
>>> lista[3]
26
>>> type(lista)
<class 'list'>
```

dostęp do poszczególnych elementów
listy uzyskujemy dzięki indeksowi
(kolejnym numerze elementu z listy,
zaczynając od zera)

https://www.w3schools.com/python/python_lists.asp

List (lista) - właściwości

```
>>> lista
['komputer', 'liczba', 3.14, 26, False, 19]
```

nasza lista

```
>>> lista.append(19)
>>> lista
['komputer', 'liczba', 3.14, 26, False, 19]
```

dołożenie elementu na koniec listy

```
>>> lista[-1]
19
>>> lista[-2]
False
>>> lista[-6]
'komputer'
```

indeksy ujemne zwracają elementy od końca listy

```
>>> lista[0:5]
['komputer', 'liczba', 3.14, 26, False]
>>> lista[4:5]
[False]
>>> lista[1:3]
['liczba', 3.14]
```

zakres zwraca elementy w zakresie listy

```
>>> lista.index(3.14)
2
>>> lista.index(19)
5
>>> lista.index("komputer")
0
```

lista zwraca indeks danego elementu

```
>>> 20 in lista
False
>>> 'komputer' in lista
True
```

sprawdzenie, czy dany element jest na liście

List (lista) - właściwości

```
1  mojaLista = []
2  mojaLista.extend("napis zamienia się w literki")
3  print(mojaLista)
```

zamieniamy string na listę poszczególnych liter (w c++ typ char)

```
['n', 'a', 'p', 'i', 's', ' ', 'z', 'a', 'm', 'i', 'e', 'n', 'i', 'a', ' ', 's', 'i', 'ę', ' ', 'w', ' ', 'l', 'i', 't', 'e', 'r', 'k', 'i']
```

```
1  liczby = [1, 2, 4, 5]
2  liczby.insert(2, 3)
3  print(liczby)
```

wstawiamy element do listy w określonym miejscu:
lista.insert(indeks, element)

```
[1, 2, 3, 4, 5]
```

```
1  tekst = "ala ma kota"
2  wynik = tekst.split()
3  print(wynik)
```

podział zdania na poszczególne słowa (separatorem jest domyślnie spacja, w wyniku powstaje lista słów)

```
['ala', 'ma', 'kota']
```

```
1  mojeDane = "dane1,dane2,dane3".split(",")
2  print(mojeDane)
```

podział zdania na poszczególne słowa (separatorem jest przecinek)

```
['dane1', 'dane2', 'dane3']
```

List (lista) - właściwości

```
1  zdanie = []
2  zdanie.extend("ala ma kota")
3  print(zdanie.count("a"))
```

```
4
```

— zliczamy ilość liter a w stringu

```
1  liczby = [3, 1, 4, 2]
2  liczby.sort()
3  print(liczby)
```

```
[1, 2, 3, 4]
```

— sortowanie listy m miejscu

```
1  liczby = [3, 1, 4, 2]
2  nowaLista = sorted(liczby)
3  print(liczby)
4  print(nowaLista)
```

```
[3, 1, 4, 2]
[1, 2, 3, 4]
```

— funkcja *sorted* zwraca nową posortowaną listę

```
1  wzrost = ["najwyższy", "niski", "wyższy"]
2  print(sorted(wzrost, key=len))
```

```
['niski', 'wyższy', 'najwyższy']
```

— funkcja *sorted* sortuje listę według klucza – długości elementów

List (lista) - właściwości

```
1  liczby = [1, 2, 3, 4, 5]
2  liczby.reverse()
3  print(liczby)
```

```
[5, 4, 3, 2, 1]
```

odwracanie listy w miejscu

```
liczby = ["jeden", "dwa", "trzy", 1, 2, 3]
liczby.remove(1)
print(liczby)
```

```
['jeden', 'dwa', 'trzy', 2, 3]
```

usuwanie elementu z listy

```
1  liczby = ["jeden", "dwa", "trzy", 1, 2, 3]
2  zwroconyElement = liczby.pop()
3  print(zwroconyElement)
4  print(liczby)
```

```
3
['jeden', 'dwa', 'trzy', 1, 2]
```

usuwanie ostatniego elementu z listy

```
1  liczby = [1, 2, 3, 4, 5, 6, 7, 8]
2  print(liczby)
3  nowaLista = list(filter(lambda x: (x > 6), liczby))
4  print(nowaLista)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
[7, 8]
```

filtrowanie listy według warunku
lambda to funkcja anonimowa

List Comprehension

List comprehension to zwięzły i czytelny sposób tworzenia list z możliwością filtrowania i modyfikacji elementów.

```
1  kwadraty = [x*x for x in range(5)]
2  print(kwadraty)
```

```
[0, 1, 4, 9, 16]
```

```
1  mojaLista = [-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2  print([x ** 2 for x in mojaLista if x > 0])
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
1  parzyste = [x for x in range(10) if x % 2 == 0]
2  print(parzyste)
```

```
[0, 2, 4, 6, 8]
```

```
1  imiona = [name.upper() for name in ["ania", "janek", "zosia"] if len(name) > 4]
2  print(imiona)
```

```
['JANEK', 'ZOSIA']
```

List (lista) – uwaga na kopiowanie

```
>>> a = [1,2,3,4]
>>> b = a
>>> b
[1, 2, 3, 4]
>>> a[0] = "jeden"
>>> a
['jeden', 2, 3, 4]
>>> b
['jeden', 2, 3, 4]
```

nasza lista o nazwie a

przypisujemy naszą listę do listy b
(to nie jest kopiowanie - **lista a oraz b wskazują ten sam fragment pamięci**)

zmieniamy element na naszej liście

zmiana elementu na naszej liście
pociąga za sobą taką samą zmianę w liście b

List (lista) – uwaga na kopiowanie

```
>>> a
['jeden', 2, 3, 4]
>>> b = a[:]
>>> b
['jeden', 2, 3, 4]
>>> a[1] = "dwa"
>>> a
['jeden', 'dwa', 3, 4]
>>> b
['jeden', 2, 3, 4]
```

nasza lista o nazwie a

kopiujemy cały zakres listy a do listy b

zmieniamy drugi element w naszej liście

drugi element w kopii listy został niezmieniony

shellow copy (kpiuje referencje obiektów, a nie same obiekty, w przeciwieństwie do *deep copy*)

Tuple (krotka)

uporządkowana, **niezmienna** kolekcja danych różnych typów

elementy krotki podajemy w nawiasach okrągłych

dostęp do elementów krotki uzyskujemy dzięki indeksowi

```
>>> mojaTupla = (1, "dwa", True)
>>> mojaTupla[1]
'dwa'
>>> mojaTupla[0] = "jeden"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

nie można zmienić elementu krotki

https://www.w3schools.com/python/python_tuples.asp

Set

nieuporządkowana, niezmienna, nieindeksowana kolekcja danych różnych typów
elementy nie mogą się duplikować

elementy set podajemy w nawiasach klamrowych

```
>>> mojSet = { 1, 2, "trzy"}
>>> mojSet
{1, 2, 'trzy'}
>>> mojSet[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
```

w set nie można zmienić elementu, można go usunąć i dodać inny

w set nie ma indeksu

https://www.w3schools.com/python/python_sets.asp

od wersji Pythona 3.7 w górę

Dictionary (słownik)

uporządkowana, zmienna, nieindeksowana kolekcja par danych klucz:wartość
elementy nie mogą się duplikować (nie mogą być 2 elementy o tym samym kluczu)

```
>>> myDictionary = {
... "kot" : "cat",
... "dom" : "home",
... "rok" : 2023
... }
```

słownik to kolekcja par key:value

```
>>> myDictionary
{'kot': 'cat', 'dom': 'home', 'rok': 2023}
```

```
>>> myDictionary["kot"]
'cat'
```

podając klucz możemy dostać jego wartość

Operatory arytmetyczne dwuargumentowe

wykonaj w konsoli

wynik

+	dodawanie	*print(4+5)*	*9*
-	odejmowanie	*print(4-8)*	*-4*
*	mnożenie	*print(3*2)*	*6*
**	potęgowanie	*print(3**2)*	*9*
/	dzielenie	*print(3/2)*	*1.5*
//	dzielenie całkowite np.: 10/4=2 (dzielimy 2 liczby – iloraz jest zaokrąglany do liczby całkowitej)	*print(3//2)*	*1*
%	modulo – reszta z dzielenia liczb całkowitych	*print(9%4)*	*1*

to samo co 3*3

9 / 4 = 2 reszty 1

https://www.w3schools.com/python/python_operators.asp

Operatory relacji (porównania)

wykonaj w konsoli

==	równy	`>>> 3==4` `False`	`>>> 10==10` `True`
!=	różny	`>>> 3!=4` `True`	`>>> 13!=13` `False`
>	większy	`>>> 12>3` `True`	`>>> 2>30` `False`
<	mniejszy	`>>> 2<4` `True`	`>>> 12<3` `False`
>=	większy lub równy	`>>> 12>=12` `True`	`>>> 1>=12` `False`
<=	mniejszy lub równy	`>>> 30<=30` `True`	`>>> 30<=3` `False`

https://www.w3schools.com/python/python_operators.asp

Instrukcja warunkowa

```
if warunek:
    instrukcja
```

jeśli spełniony jest *warunek* (ma wartość True)
wykonywana jest *instrukcja*

```
if warunek:
    instrukcja
else:
    instrukcja2
```

jeśli spełniony jest *warunek* (ma wartość True)
wykonywana jest *instrukcja*
w innym przypadku wykonywana jest *instrukcja2*

```
if warunek:
    instrukcja
elif warunek2:
    instrukcja2
else:
    instrukcja3
```

jeśli spełniony jest *warunek* (ma wartość True)
wykonywana jest *instrukcja*
w innym przypadku jeśli spełniony jest *warunek2*
(ma wartość True) wykonywana jest *instrukcja2*
w innym przypadku wykonywana jest *instrukcja3*

Instrukcja warunkowa

program *instrukcjaWarunkowa.py*

```
a = 100
b = 10
if a > b:
    print("liczba " + str(a) + " jest większa od liczby " + str(b))
elif a==b:
    print("liczby są równe")
else:
    print("liczba " + str(b) + " jest większa od liczby " + str(a))
```

```
liczba 100 jest większa od liczby 10
```

Instrukcja warunkowa

program *instrukcjaWarunkowa1.py*

```
# input zwraca string
a = input("podaj pierwszą liczbę całkowitą: ")
b = input("podaj drugą liczbę całkowitą: ")
a = int(a)  # zmiana ciągu tekstowego na liczbę całkowitą
b = int(b)  # zmiana ciągu tekstowego na liczbę całkowitą
if a > b:
    print("liczba " + str(a) + " jest wieksza od liczby " + str(b))
elif b > a:
    print("liczba " + str(b) + " jest wieksza od liczby " + str(a))
else:
    print("liczby są równe")
```

konkatenacja (sklejanie) stringów

zamiana liczby na string

Pętla for

kolejne słowo zaczyna się od nowej linii

```
for.py > ...
1   for i in range(5):
2       print("pionowo")
3
4   for i in range(5):
5       print("poziomo ", end="")
6
7   print("") # enter
8
9   for i in range(3,6):
10      print(i)
11
12  for i in range(2,10,2):
13      print(str(i) + " ", end="")
```

zakres

printowanie w poziomie

od do

od do krok

```
pionowo
pionowo
pionowo
pionowo
pionowo
poziomo poziomo poziomo poziomo poziomo
3
4
5
2 4 6 8
```

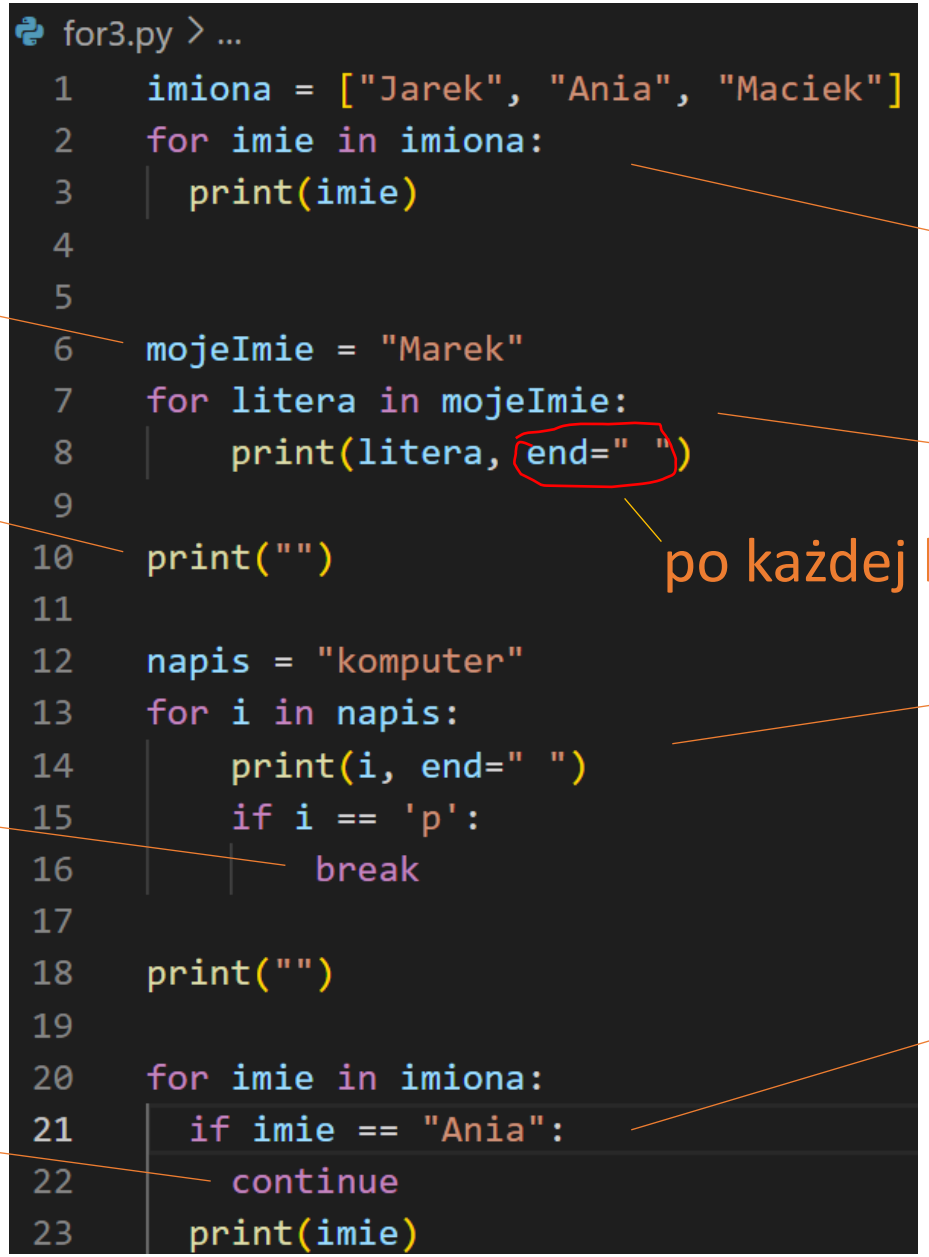
kolejne słowa po kolei

liczby parzyste

spacja pomiędzy liczbami

https://www.w3schools.com/python/python_for_loops.asp

Pętla for



tablica imion

ciąg tekstowy
(w cudzysłowach)
jest tablicą

```
Jarek
Ania
Maciek
```

znak nowego wiersza

po każdej literze spacja

```
M a r e k
```

```
k o m p
```

break przerywa pętlę

```
Jarek
Maciek
```

continue wraca
do początku pętli

Pętla for

for2.py > ...

```
1   for x in range(5):
2       print(x)
3   else:
4       print("Skończone!")
5
6
7
8   print("owocowy wierszyk :)")
9
10  opisy = ["rosną ", "coraz większe ", "smacznie zajadam "]
11  owoce = ["gruszki", "jabłka", "śliwki"]
12
13  for opis in opisy:
14      for owoc in owoce:
15          print(opis, owoc)
16
17
18  klasa = [1,2,3]
19  for x in klasa:
20      pass
```

```
0
1
2
3
4
Skończone!
```

po skończonej pętli wypisuje się napis ”Skończone!”

```
owocowy wierszyk :)
rosną  gruszki
rosną  jabłka
rosną  śliwki
coraz większe  gruszki
coraz większe  jabłka
coraz większe  śliwki
smacznie zajadam  gruszki
smacznie zajadam  jabłka
smacznie zajadam  śliwki
```

pętle zagnieżdżone

pętla nie może być pusta, dzięki pass (w pustej pętli) nie drukują się błędy

Pętla while

```
while.py > ...
1   i = 0
2   while i < 10:
3       print(i)
4       i += 1
5
6   k = 0
7   while k < 2000000:
8       print(k, end=" ")
9       k += 2
10      if k == 10:
11          break
12
13
14  print("")
15
16  m = 0
17  while m < 20:
18      m += 1
19      if m % 2 == 0:
20          continue
21      print(m, end=" ")
```

```
0
1
2
3
4
5
6
7
8
9
```

licznik pętli

break przerywa pętlę

```
0 2 4 6 8
```

co druga liczba

liczby nieparzyste

continue wraca
do początku pętli

```
1 3 5 7 9 11 13 15 17 19
```

https://www.w3schools.com/python/python_while_loops.asp

funkcja

definicja funkcji o nazwie poleKwadratu

nazwa funkcji

parametr funkcji *(w nawiasach)*

słowo kluczowe def
(z angielskiego definition)

```
def poleKwadratu(a):
    return a**2


pole = poleKwadratu(2)
print(pole)  # 4
```

wartość zwracana przez funkcję
(zmienna a do potęgi drugiej)

zmienna lokalna pole

komentarz (program tego nie widzi)

wartość zwracana przez funkcję zostaje przypisana do zmiennej pole

funkcja

2. funkcja liczy potęgę zmiennej a (dwa do potęgi drugiej wynosi 4) i zwraca wynik

2

```
def poleKwadratu(a):
    return a**2

pole = poleKwadratu(2)
print(pole)  # 4
```

3 1

4

1 2 3 4

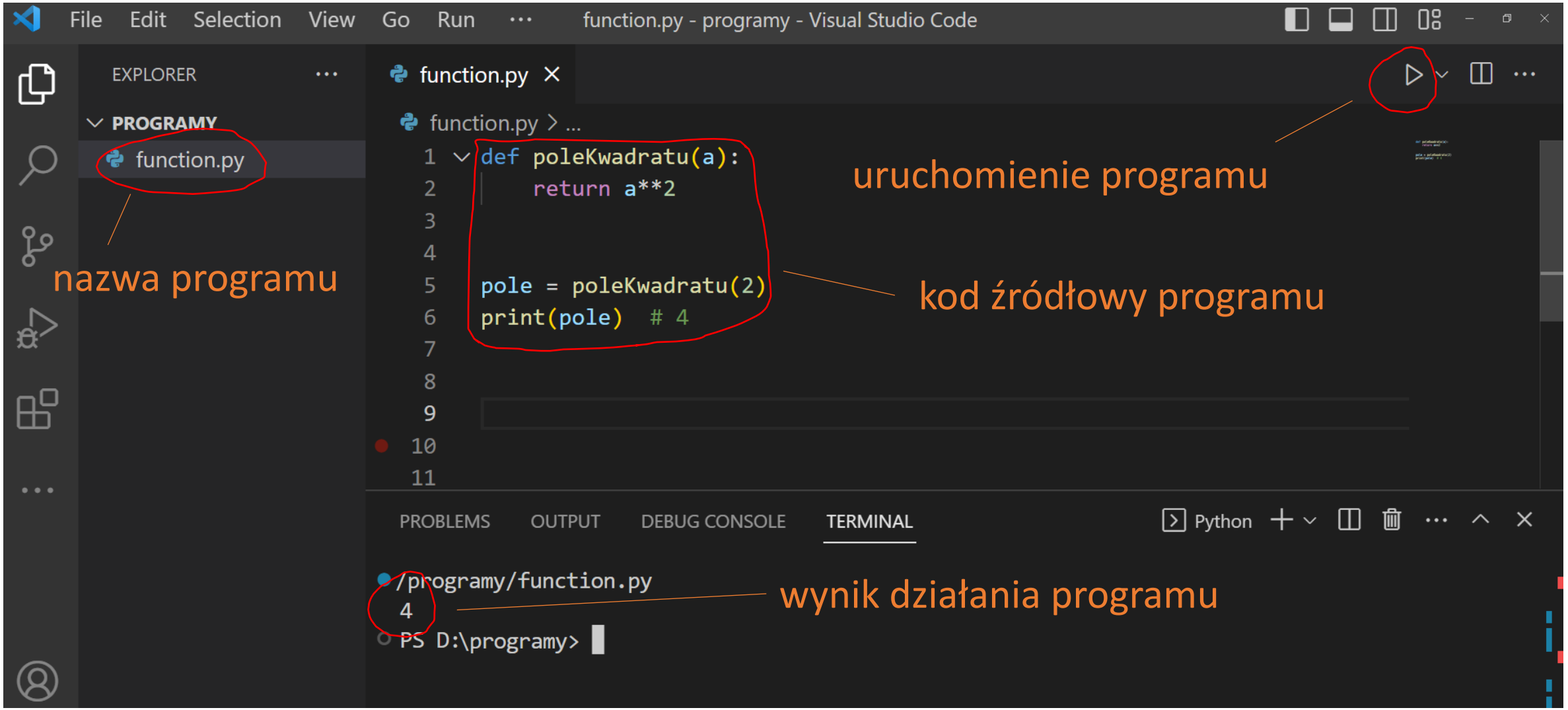
kolejność działania programu

3. wartość zwrócona przez funkcję (liczba 4) jest przypisana do zmiennej pole

1. wywołanie funkcji z argumentem 2, który zostaje skopiowany do zmiennej a (parametru funkcji)

4. zawartość zmiennej pole jest wypisana w konsoli

funkcja


File Edit Selection View Go Run ···
function.py - programy - Visual Studio Code
EXPLORER
PROGRAMY
function.py
function.py
function.py > ...
1 def poleKwadratu(a):
2 return a**2
3
4
5 pole = poleKwadratu(2)
6 print(pole) # 4
7
8
9
10
11
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Python
/programy/function.py
4
PS D:\programy>
nazwa programu
uruchomienie programu
kod źródłowy programu
wynik działania programu

funkcja

zdefiniować następujące funkcje i je wywołać z argumentami:

a = 3

b = 4

h = 5

```
poleProstakata(a,b)
poleTrojkata(a,h)
```

szyfr Cezara

cezar.py > ...

```python
# szyfrowanie, bez polskich znaków

klucz = 1

def zaszyfruj(slowo, klucz):
    for znak in slowo:
        print(znak + ": " + str(ord(znak))) # kod ASCII
    print("kod ASCII 98: " + chr(98)) # znak podanego kodu ASCII
    zaszyfrowane = ""
    for znak in slowo:
       zaszyfrowane += chr((ord(znak) - 97 + klucz) % 26 + 97)
    return zaszyfrowane

def odszyfruj(zaszyfrowane,klucz):
    slowo = zaszyfruj(zaszyfrowane, 26 - klucz) # odszyfrowuje sie
                                                # szyfrując kluczem
                                                # dopełniajacym do 26
    return slowo

slowo = input("podaj słowo do zaszyfrowania: ")

zaszyfrowane = zaszyfruj(slowo, klucz)
odszyfrowane = odszyfruj(zaszyfrowane, klucz)

print("zaszyfrowane: " + zaszyfrowane)
print("odszyfrowane: " + odszyfrowane)
```

klasy

W Pythonie klasy służą do tworzenia własnych typów danych — czyli obiektów z właściwościami (atrybutami, polami) i zachowaniem (metodami).

podwójny podkreślnik

```python
class Osoba:
    def __init__(self, imie, wiek): # konstruktor – metoda wywoływana przy tworzeniu obiektu.
        self.imie = imie            # pole klasy
        self.wiek = wiek            # pole klasy

    def przywitaj(self):            # metoda klasy
        print(f"Cześć, mam na imię {self.imie} i mam {self.wiek} lat.")


# utworzenie obiektu (instancji)
ala = Osoba("Ala", 12)

# wywołanie metody
ala.przywitaj()
```

self odnosi się do bieżącego obiektu (instancji) (jak "this" w JavieScript).

```
Cześć, mam na imię Ala i mam 12 lat.
```

klasy – atrybuty prywatne

Atrybuty zaczynające się od podwójnego podkreślenia są traktowane jako prywatne.

```python
class Konto:
    def __init__(self, saldo):
        self.__saldo = saldo  # atrybut „prywatny”

    def pokaz_saldo(self):
        return self.__saldo

    def wplata(self, kwota):
        self.__saldo += kwota
```

do atrybutu __saldo nie można się dostać tradycyjnym sposobem

```python
mojeKonto = Konto(100)
print(mojeKonto.__saldo)  # AttributeError
```

do atrybutu __saldo można się dostać przez *name mangling* (ale to nie jest zalecane)

```python
mojeKonto = Konto(100)
print(mojeKonto._Konto__saldo)  # działa, ale niezalecane
```

```
100
```

klasy – atrybuty chronione

Pojedyncze podkreślenie to konwencja, że atrybut jest przeznaczony tylko do użycia wewnętrznego (ale nie jest blokowany)

```python
class Konto:
    def __init__(self, saldo):
        self._saldo = saldo  # atrybut "chroniony"

    def pokaz_saldo(self):
        return self._saldo

    def wplata(self, kwota):
        self._saldo += kwota


mojeKonto = Konto(100)
print(mojeKonto._saldo)    # działa
```

```
100
```

klasy – @property

@property to dekorator Pythona pozwalający kontrolować dostęp do atrybutów obiektu — czyli tworzyć tzw. właściwości (properties).

```python
class Konto:
    def __init__(self, saldo):
        self.__saldo = saldo

    @property                   # getter
    def saldo(self):
        return self.__saldo

    @saldo.setter               # setter
    def saldo(self, kwota):
        if kwota < 0:
            raise ValueError("Saldo nie może być ujemne!")
        self.__saldo = kwota


mojeKonto = Konto(100)
mojeKonto.saldo = 200     # wywołuje setter
print(mojeKonto.saldo)    # wywołuje getter
```

```
200
```

klasy - dziedziczenie

```python
class Zwierze:                          # klasa bazowa Zwierze
    def __init__(self, gatunek):  # konstruktor klasy bazowej
        self.gatunek = gatunek

    def info(self):
        print("Jestem zwierzęciem:", self.gatunek)


class Pies(Zwierze):    # klasa potomna Pies dziedziczy po klasie Zwierze
    def szczekaj(self):
        print("Hau hau!")


p = Pies("Owczarek")
p.info()        # odziedziczona metoda z klasy bazowej Zwierze
p.szczekaj()    # metoda z klasy Pies
```

```
Jestem zwierzęciem: Owczarek
Hau hau!
```

klasy - dziedziczenie

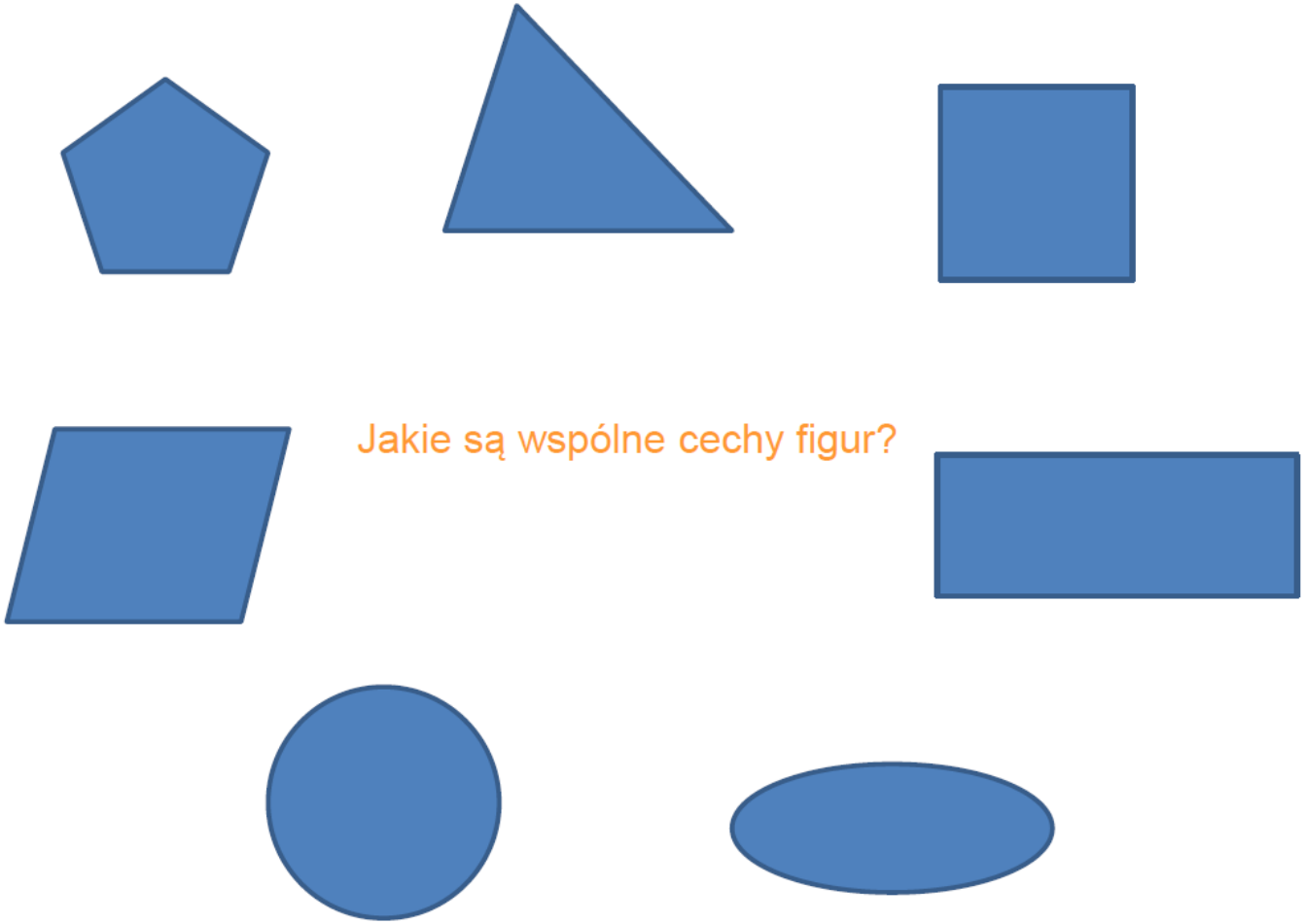
dziedziczenie1.py

klasy > dziedziczenie1.py > ...

```python
class Zwierze:
    def __init__(self, gatunek):
        self.gatunek = gatunek

    def info(self):
        print("jestem zwierzęciem:", self.gatunek)

class Pies(Zwierze):
    def __init__(self, gatunek, imie):  # klasa Pies ma własny konstruktor, dlatego
        super().__init__(gatunek)       # wywołujemy konstruktor klasy bazowej
        self.imie = imie

    def szczekaj(self):
        print("mam na imię:", self.imie,"\nHau, hau!")


p = Pies("Owczarek", "Pimpuś")  # wywołanie konstruktora klasy Pies
p.info()
p.szczekaj()
```

```
jestem zwierzęciem: Owczarek
mam na imię: Pimpuś
Hau, hau!
```

klasy - dziedziczenie



klasy - dziedziczenie

dziedziczenie3.py > ...

```python
class Figura:
    def __init__(self, nazwa):
        self.nazwa = nazwa
    pole = 0
    obwod = 0
    def toString(self):
        print(f"nazwa: {self.nazwa}\npole: {self.pole}\nobwod: {self.obwod}")

class Kwadrat(Figura):
    def __init__(self, nazwa, bok):
        super().__init__(nazwa)
        self.bok = bok
        self.pole = bok * bok
        self.obwod = 4 * bok



kwadrat = Kwadrat("kwadrat", 10)
kwadrat.toString()
```

(parameter) bok: Any

nazwa, pole, obwod, toString – wspólne dla dziedziczących klas

```
nazwa: kwadrat
pole: 100
obwod: 40
```

klasy – pole statyczne

poleStatyczne.py > ...

```python
class Samochod:
    liczba_kol = 4  # To jest pole statyczne (atrybut klasy)

    def __init__(self, marka):
        self.marka = marka  # Pole instancji

a1 = Samochod("Fiat")
a2 = Samochod("Ford")

print(a1.liczba_kol)  # 4 - dostep do pola statycznego poprzez nazwę obiektu lub klasy
print(Samochod.liczba_kol)  # 4

# Zmiana pola statycznego
Samochod.liczba_kol = 6 # - zmiana pola poprzez nazwę klasy
print(a2.liczba_kol)  # 6 - zmiana dotyczy wszystkich instancji
print(a1.liczba_kol)  # 6

a1.liczba_kol = 1       # ! zmiana pola poprzez nazwę obiektu
print(a1.liczba_kol)    # 1 - Teraz a1 ma własne pole instancji,
                        # która przesłania pole statyczne klasy.
print(a1.__class__.liczba_kol) # 6 - dostęp do pola statycznego
print(Samochod.liczba_kol) # 6 - dostęp do pola statycznego
print(a2.liczba_kol)    # 6
```

pole statyczne klasy jest wspólne dla wszystkich obiektów danej klasy

klasy – pole statyczne

zadanie – napisz klasę z licznikiem stworzonych obiektów

klasy – metody

metodaStatyczna.py > ...

```python
class Samochod:
    liczba_kol = 4  # To jest pole statyczne (atrybut klasy)

    def __init__(self, marka):
        self.marka = marka  # Pole instancji

    def getMarka(self):     # zwykła metoda (instancji)
        return self.marka

    @classmethod            # metoda klasowa
    def ile(cls):
        return cls.liczba_kol

    @staticmethod           # metoda statyczna klasy
    def ileMaLat(rocznikAuta):
        ile = 2026 - rocznikAuta
        print(f"Twoje auto ma {ile} lat")


a1 = Samochod("mercedes")
print(a1.getMarka())    # wywołanie zwykłej metody
print(Samochod.ile())   # wywołanie metody klasowej
Samochod.ileMaLat(2015) # wywołanie metody statycznej
```

```
mercedes
4
Twoje auto ma 11 lat
```

zwykła metoda (instancji) ma dostęp do argumentów (pól) i metod instancji klasy

metoda klasowa ma dostęp do zmiennych klasowych (pól statycznych)
cls – reprezentuje klasę (Samochod)

metoda statyczna klasy nie ma dostępu ani do obiektu (atrybutów, metod) ani do klasy (zmiennych klasowych)

klasy – metody statyczne

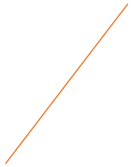
zadanie – napisz kalkulator
z wykorzystaniem metod statycznych
(w klasie powinny się znaleźć tylko te metody)

klasy - polimorfizm

polimorfizm

Polymorphism

wielopostaciowość metod klasy



przesłanianie metod

methods overriding

klasy – przesłanianie metod

przeslanianieMetod.py

```python
class Zwierze:
    def dzwiek(self):
        pass  # Metoda bazowa, może zostać zaimplementowana w klasach pochodnych

class Pies(Zwierze):
    def dzwiek(self): # Metoda dzwiek() przesłania metodę bazową o tej samej nazwie
        return "Hau hau!"

class Kot(Zwierze):
    def dzwiek(self): # Metoda dzwiek() przesłania metodę bazową o tej samej nazwie
        return "Miau!"

# Funkcja, która przyjmuje obiekt typu Zwierze i wywołuje metodę dzwiek
def wydaj_dzwiek(zwierze):
    print(zwierze.dzwiek())

# Tworzenie obiektów
pies = Pies()
kot = Kot()

# Polimorfizm - ta sama funkcja, różne obiekty, różne zachowania
wydaj_dzwiek(pies)  # Hau hau!
wydaj_dzwiek(kot)   # Miau!
```

metoda przesłaniająca musi mieć taką samą nazwę jak metoda przesłaniana.

klasa abstrakcyjna

Klasa abstrakcyjna to klasa bazowa, z której nie można utworzyć instancji (obiektu), a jej głównym celem jest definiowanie wspólnego interfejsu dla klas pochodnych.

klasaAbstrakcyjna.py

```python
# z modułu abc (Abstract Base Classes) importujemy:
# ABC - klasę bazową
# dekorator @abstractmethod
from abc import ABC, abstractmethod

class Zwierze(ABC):    # klasa Zwierze dziedziczy po klasie ABC

    @abstractmethod           # dekorator
    def wydaj_dzwiek(self):   # metoda abstrakcyjna (bez implementacji)
        pass


class Pies(Zwierze):         # klasa Pies dziedziczy po klasie abstrakcyjnej

    def wydaj_dzwiek(self):  # implementacja metody abstrakcyjnej
        return "Hau hau"


pies = Pies()
print(pies.wydaj_dzwiek())  # Hau hau
```

wzorce projektowe - fabryka abstrakcyjna

fabryka.py

```python
from abc import ABC, abstractmethod

class Chair(ABC):
    @abstractmethod
    def getProduct(self) -> str:
        pass

class ArtDecoChair(Chair):
    def getProduct(self):
        return "Jestem krzesło Art Deco"

class ModernChair(Chair):
    def getProduct(self):
        return "Jestem krzesło Modern"


class FurnitureFactory(ABC):
    @abstractmethod
    def createChair(self) -> Chair:
        pass

class ArtDecoFurnitureFactory(FurnitureFactory):
    def createChair(self):
        return ArtDecoChair()

class ModernChairFurnitureFactory(FurnitureFactory):
    def createChair(self):
        return ModernChair()


class Client:
    def __init__(self, furnitureFactory):
        self.factory = furnitureFactory
        print(f"Drogi kliencie! Wyprodukowaliśmy dla Ciebie {self.factory.createChair().getProduct()}")


artDecoFactory = ArtDecoFurnitureFactory()
artDecoClient = Client(artDecoFactory)

modernChairFactory = ModernChairFurnitureFactory()
modernChairClient = Client(modernChairFactory)
```

produkt abstrakcyjny

konkretne produkty

fabryka abstrakcyjna

konkretne fabryki

klient przyjmujący w konstruktorze typ fabryki abstrakcyjnej

tworzenie konkretnych obiektów

https://refactoring.guru/pl/design-patterns/abstract-factory

obsługa błędów

ta linijka kodu spowoduje wygenerowanie błędu (string „ala” nie jest liczbą). Program zostanie przerwany i zostanie wygenerowany błąd (wyjątek).

```
1
2   x = int("ala")
3
```

```
    x = int("ala")
        ^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'ala'
```

obsługa błędów

obsłużenie wygenerowanego błądu

```
1  try:
2      #x = int("223")
3      x = int("ala")
4  except:
5      print("Błąd!")
6  else:
7      print("Sukces:", x)
8  finally:
9      print("Ten blok wykona się zawsze")
10
11 print("program generując wyjątek nie zatrzymał się!")
```

blok try umożliwia przetestowanie bloku kodu pod kątem błędów.

blok except umożliwia obsłużenie błędu.

blok else umożliwia wykonanie kodu, jeżeli nie występuje błąd.

blok Finally pozwala na wykonanie kodu niezależnie od wyniku bloków try- i except.

```
Błąd!
Ten blok wykona się zawsze
program generując wyjątek nie zatrzymał się!
```

https://www.w3schools.com/python/python_try_except.asp

obsługa błędów

```
1   try:
2       print(x)
3   except NameError as e:
4       print("Szczegóły błędu:", e)
5   except Exception as e:
6       print("Something else went wrong")
7       print("Szczegóły:", e)
8
9   print("program generując wyjątek nie zatrzymał się!")
```

Można zdefiniować dowolną liczbę bloków wyjątków, np. specjalny blok kodu dla specjalnego rodzaju błędu

```
Szczegóły błędu: name 'x' is not defined
program generując wyjątek nie zatrzymał się!
```

https://www.w3schools.com/python/python_try_except.asp

obsługa błędów

Exception hierarchy

```
BaseException
 ├── BaseExceptionGroup
 ├── GeneratorExit
 ├── KeyboardInterrupt
 ├── SystemExit
 └── Exception
      ├── ArithmeticError
      │    ├── FloatingPointError
      │    ├── OverflowError
      │    └── ZeroDivisionError
      ├── AssertionError
      ├── AttributeError
      ├── BufferError
      ├── EOFError
      ├── ExceptionGroup [BaseExceptionGroup]
      ├── ImportError
      │    └── ModuleNotFoundError
      ├── LookupError
      │    ├── IndexError
      │    └── KeyError
      ├── MemoryError
      ├── NameError
      │    └── UnboundLocalError
```

hierarchia wbudowanych wyjątków

exception **NameError**
Raised when a local or global name is not found. This applies only to unqualified names. The associated value is an error message that includes the name that could not be found.

https://docs.python.org/pl/3/library/exceptions.html#bltin-exceptions

obsługa błędów

błędem można również „rzucić” (raise error)
raise przerywa program (chyba, ze jest w bloku try-except)

```
1  x = None
2
3  if x is None:
4      raise NameError("Zmienna x nie ma wartości!")
5
6  print("To się nie wykona")
```

```
NameError: Zmienna x nie ma wartości!
```

obsługa błędów

klasa własnego wyjątku dziedziczy po klasie Exception

```python
class InvalidAgeError(Exception):
    def __init__(self, age):
        self.age = age
        super().__init__(f"Nieprawidłowy wiek: {age}")

def sprawdz_wiek(age):
    if age < 0:
        raise InvalidAgeError(age)
    if age < 18:
        raise InvalidAgeError("Użytkownik jest niepełnoletni")
    print("wiek prawidłowy")

try:
    sprawdz_wiek(10)
except InvalidAgeError as e:
    print("Błąd:", e)
```

```
Błąd: Nieprawidłowy wiek: Użytkownik jest niepełnoletni
```

obsługa błędów

ZADANIE:
na podstawie poprzedniego slajdu
stworzyć inny własny wyjątek i go obsłużyć