

python

popularny, interpretowany język programowania
ogólnego przeznaczenia

twórcą języka jest holenderski
programista [Guido van Rossum](#)



używany m.in. do tworzenia:

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

Python

dynamicznie
typowany

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/

Python PSF Docs PyPI Jobs Community

 [Donate](#) [GO](#) [Socialize](#)

About Downloads Documentation Community Success Stories News Events

Download the latest version for Windows

[Download Python 3.11.2](#)

Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [macOS](#), [Other](#)

Want to help test development versions of Python? [Prereleases](#), [Docker images](#)



Visual Studio Code

https://code.visualstudio.com

Visual Studio Code Docs Updates Blog API Extensions FAQ Learn

Search Docs Download

Version 1.75 is now available! Read about the new features and fixes from January.

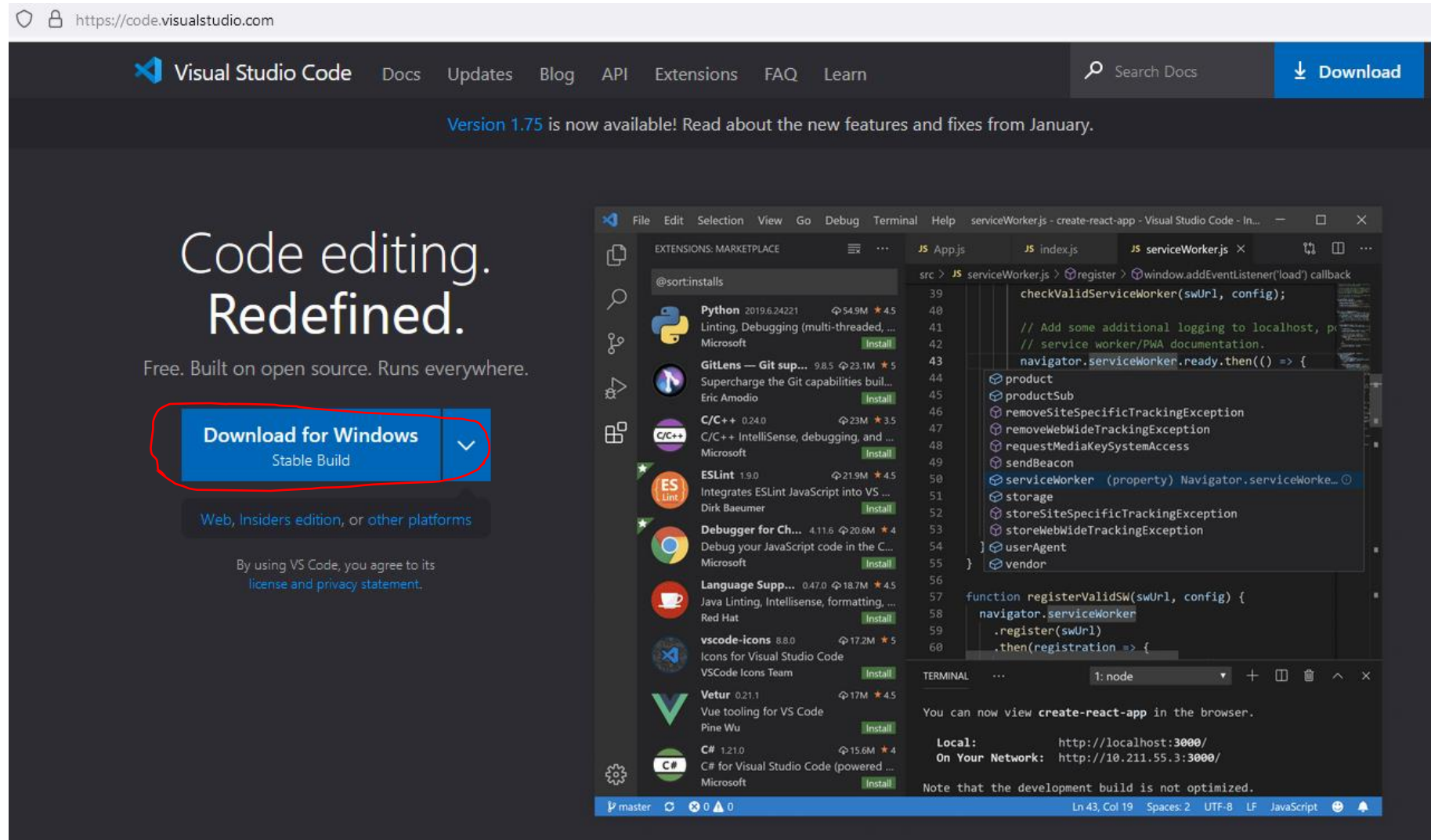
Code editing. Redefined.

Free. Built on open source. Runs everywhere.

Download for Windows
Stable Build

[Web, Insiders edition, or other platforms](#)

By using VS Code, you agree to its [license and privacy statement](#).



The screenshot displays the Visual Studio Code website with a dark theme. The main heading 'Code editing. Redefined.' is prominent, followed by the tagline 'Free. Built on open source. Runs everywhere.' A blue button labeled 'Download for Windows' with 'Stable Build' underneath is circled in red. Below it is a link for other platforms. The right side of the image shows a preview of the VS Code IDE interface. The 'EXTENSIONS: MARKETPLACE' sidebar on the left lists various extensions like Python, GitLens, C/C++, ESLint, and others, each with an 'Install' button. The main editor area shows a JavaScript file named 'serviceWorker.js' with code for a service worker. A dropdown menu is open over the 'navigator.serviceWorker' property, listing various methods and properties. At the bottom, a terminal window shows the command '1: node' and the output of a web server running on localhost:3000.

Python extension for Visual Studio Code

The screenshot displays the Visual Studio Code interface with the Extensions Marketplace open. The search bar at the top of the marketplace contains the text "python". The first search result, "Python" by Microsoft, is highlighted with a red circle. This extension is described as "IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Jupyter Note..." and has 77.4M downloads and a 4-star rating. Below it, other extensions like "Python for VSCode", "Python Extension Pack", and "Python Indent" are listed. On the right, the details page for the "Python" extension is shown, featuring the Python logo, version "v2023.2.0", and an "Install" button which is also circled in red. The interface includes a sidebar with icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The top menu bar shows File, Edit, Selection, View, Go, Run, Terminal, and Help.

EXTENSIONS: MARKETPLACE

python

Python v2023.2.0
IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Jupyter Note...
Microsoft 77.4M ★ 4

Python for VSCode
Python language extension for vscode
Thomas Haakon Townsend 5M ★ 2

Python Extension Pack
Popular Visual Studio Code extensions for Python
Don Jayamanne 4.6M ★ 4

Python Indent
Correct Python indentation
Kevin Rose 4.1M ★ 4.5

Python v2023.2.0
Microsoft 77,432,939 ★★★★★ (529)
IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Jupyter Note...

Install

DETAILS FEATURE CONTRIBUTIONS CHANGELOG EXTENSION PACK

Python extension for Visual Studio Code

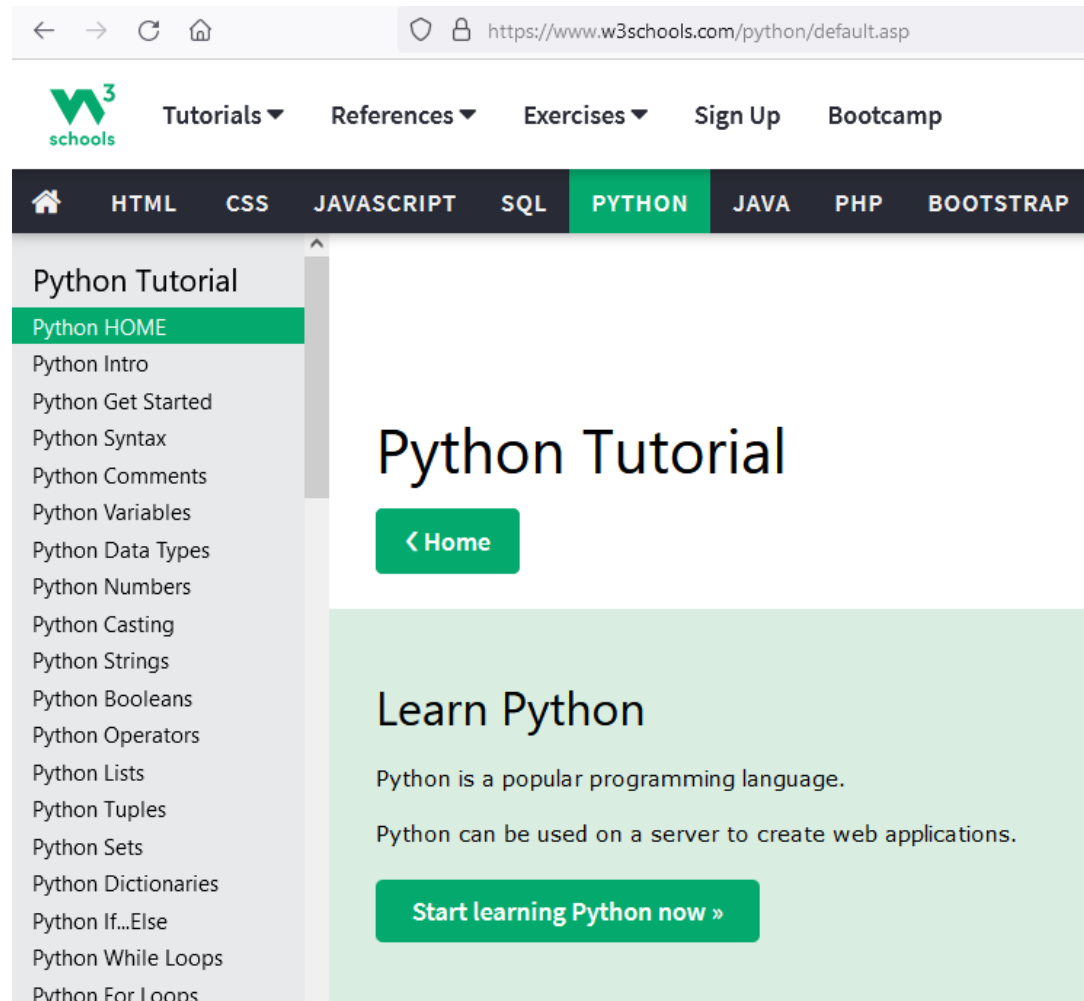
A Visual Studio Code extension with rich support for the Python language (for all actively supported versions of the language: ≥ 3.7), including features such as

Categories

Programming Languages

Linters Debuggers

tutorial



The screenshot shows the W3Schools Python Tutorial page. At the top, there is a navigation bar with links for Tutorials, References, Exercises, Sign Up, and Bootcamp. Below this is a dark navigation bar with links for HTML, CSS, JAVASCRIPT, SQL, PYTHON (highlighted), JAVA, PHP, and BOOTSTRAP. On the left, a sidebar lists various Python topics, with 'Python HOME' highlighted. The main content area features the title 'Python Tutorial', a '< Home' button, and a green box with the heading 'Learn Python' and a 'Start learning Python now »' button.

← → ↻ 🏠 <https://www.w3schools.com/python/default.asp>

W³schools Tutorials ▼ References ▼ Exercises ▼ Sign Up Bootcamp

🏠 HTML CSS JAVASCRIPT SQL **PYTHON** JAVA PHP BOOTSTRAP

Python Tutorial

Python HOME

Python Intro

Python Get Started

Python Syntax

Python Comments

Python Variables

Python Data Types

Python Numbers

Python Casting

Python Strings

Python Booleans

Python Operators

Python Lists

Python Tuples

Python Sets

Python Dictionaries

Python If...Else

Python While Loops

Python For Loops

Python Tutorial

[< Home](#)

Learn Python

Python is a popular programming language.

Python can be used on a server to create web applications.

[Start learning Python now »](#)

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

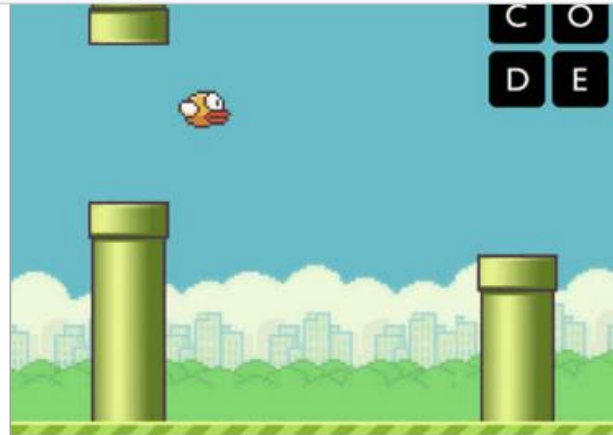
godzina kodowania

 <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

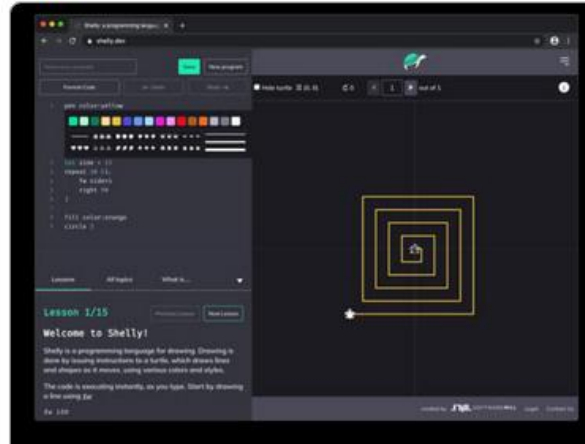
<https://hourofcode.com/pl/learn>

godzina kodowania

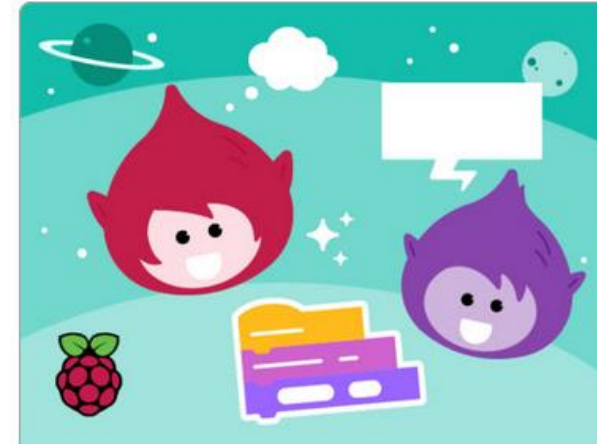
<https://hourofcode.com/pl/learn>



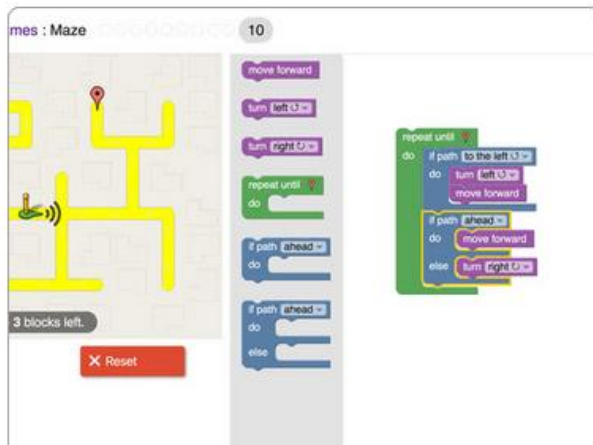
Anything is Possible! Storytelling with Scratch
Klasy 2-8 | Bloki



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



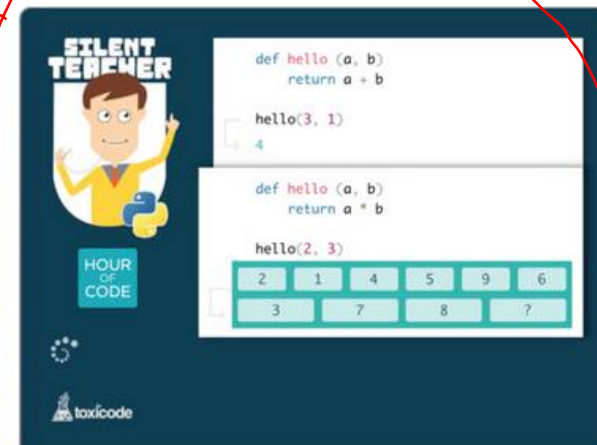
Kosmiczna rozmowa: Wystrzel w kodzenie w ...
Klasy 2-8 | Bloki, Scratch



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

klocki

zmienne

(kontenery do przechowywania danych)

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

pętla while

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

pętla for

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

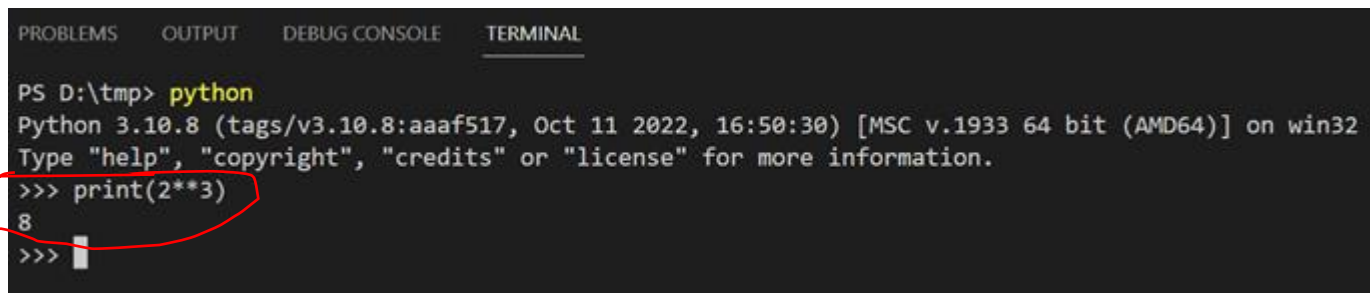
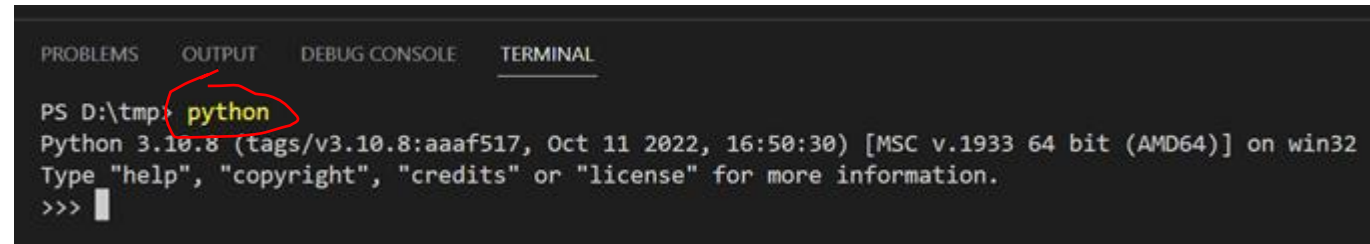
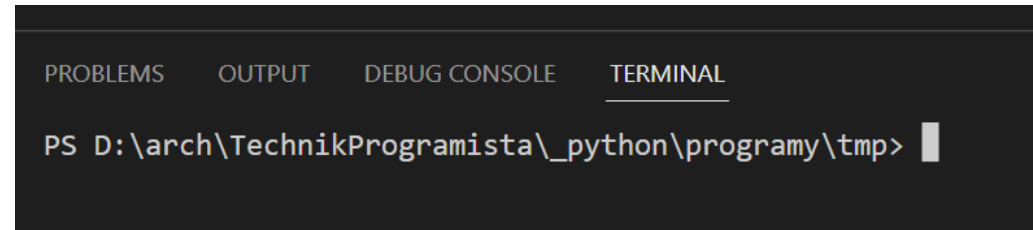
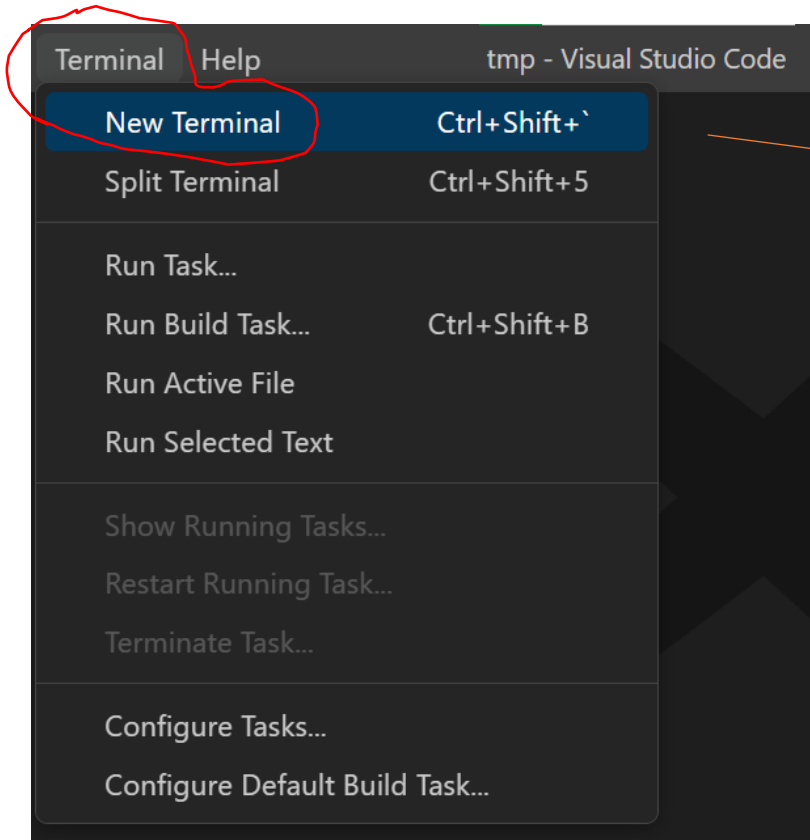
instrukcja warunkowa

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

funkcja

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

praca w terminalu



Visual Studio Code

zmienne

zmienne są pojemnikami na dane,
które używamy w naszym 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'>
```

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.

```
>>> b=3.14
>>> type(b)
<class 'float'>
```

zmienna typu *float* – liczba rzeczywista

```
>>> c="Marian"
>>> type(c)
<class 'str'>
```

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*

```
>>> type(B)
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
NameError: name 'B' is not defined. Did you mean: 'b'?
```

wielkość liter ma znaczenie

string

w potrójnym cudzysłowie możemy umieścić
ciąg tekstowy w kilku liniach

```
>>> 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)

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 (separatorom jest domyślnie
spacja, w wyniku powstaje lista słów)

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

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

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

— podział zdania na poszczególne słowa (separatorom jest przecinek)

List (lista) - właściwości

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

4

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

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

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

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

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

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

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

— zliczamy ilość liter a w stringu

— sortowanie listy m miejscu

— funkcja *sorted* zwraca nową posortowaną listę

— 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

/
shallow copy (kopiuje 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

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

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

to samo co 3*3

9 / 4 = 2 reszty 1

Operatory relacji (porównania)

wykonaj w konsoli



==	równy	<pre>>>> 3==4 False</pre>	<pre>>>> 10==10 True</pre>
!=	różny	<pre>>>> 3!=4 True</pre>	<pre>>>> 13!=13 False</pre>
>	wiekszy	<pre>>>> 12>3 True</pre>	<pre>>>> 2>30 False</pre>
<	mniejszy	<pre>>>> 2<4 True</pre>	<pre>>>> 12<3 False</pre>
>=	wiekszy lub równy	<pre>>>> 12>=12 True</pre>	<pre>>>> 1>=12 False</pre>
<=	mniejszy lub równy	<pre>>>> 30<=30 True</pre>	<pre>>>> 30<=3 False</pre>

Instrukcja warunkowa

```
if warunek:  
    instrukcja
```

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

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

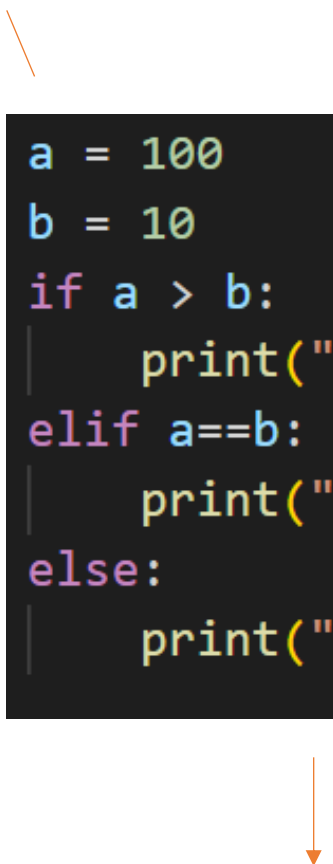
jeśli spełniony jest *warunek* (ma wartość True)
wykonywana jest *instrukcja*
w innym przypadku wykonywana jest *instrukcja2*

```
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 większa od liczby " + str(b))
elif b > a:
    print("liczba " + str(b) + " jest większa 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="")
```

```
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

Pętla for

```
for3.py > ...
1  imiona = ["Jarek", "Ania", "Maciek"]
2  for imie in imiona:
3      print(imie)
4
5
6  mojeImie = "Marek"
7  for litera in mojeImie:
8      print(litera, end=" ")
9
10 print("")
11
12 napis = "komputer"
13 for i in napis:
14     print(i, end=" ")
15     if i == 'p':
16         break
17
18 print("")
19
20 for imie in imiona:
21     if imie == "Ania":
22         continue
23     print(imie)
```

tablica imion

ciąg tekstowy
(w cudzysłowach)
jest tablicą

znak nowego wiersza

po każdej literze spacja

break przerywa pętlę

continue wraca
do początku pętli

Jarek
Ania
Maciek

M a r e k

k o m p

Jarek
Maciek

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
```

po skończonej pętli
wypisuje się napis
"Skończone!"

pętle zagnieżdżone

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

```
0
1
2
3
4
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
```

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

0 2 4 6 8

co druga liczba

liczby nieparzyste

1 3 5 7 9 11 13 15 17 19

break przerywa pętlę

continue wraca
do początku pętli

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  
  
3      1  
pole = poleKwadratu(2)  
print(pole) # 4
```

1 2 3 4

kolejność działania programu

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

4

4. zawartość zmiennej pole
jest wypisana w konsoli

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

funkcja

The screenshot displays the Visual Studio Code interface with a Python file named `function.py` open. The Explorer sidebar on the left shows the file structure with `function.py` selected. The main editor window shows the source code of the program, which defines a function `poleKwadratu(a)` and calls it with the argument `2`. The output of the program, the number `4`, is shown in the terminal window at the bottom. Red circles and arrows highlight key elements: the file name in the Explorer, the source code in the editor, the run button in the top right, and the output in the terminal.

uruchomienie programu

nazwa programu

kod źródłowy programu

wynik działania programu

```
function.py - programy - Visual Studio Code

EXPLORER
PROGRAMY
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 + - [ ] [X] ... ^ X
• /programy/function.py
  4
  PS D:\programy>
```

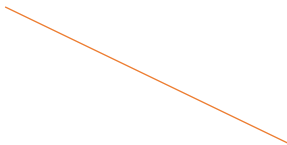
funkcja

zdefiniować następujące funkcje i je wywołać z argumentami:

a = 3

b = 4

h = 5



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

szyfr Cezara

```
cezar.py > ...
1  # szyfrowanie, bez polskich znaków
2
3  klucz = 1
4
5  def zaszyfruj(slowo, klucz):
6      for znak in slowo:
7          print(znak + ": " + str(ord(znak))) # kod ASCII
8      print("kod ASCII 98: " + chr(98)) # znak podanego kodu ASCII
9      zaszyfrowane = ""
10     for znak in slowo:
11         zaszyfrowane += chr((ord(znak) - 97 + klucz) % 26 + 97)
12     return zaszyfrowane
13
14 def odszyfruj(zaszyfrowane, klucz):
15     slowo = zaszyfruj(zaszyfrowane, 26 - klucz) # odszyfrowuje sie
16     # szyfrując kluczem
17     # dopełniającym do 26
18     return slowo
19
20 slowo = input("podaj słowo do zaszyfrowania: ")
21
22 zaszyfrowane = zaszyfruj(slowo, klucz)
23 odszyfrowane = odszyfruj(zaszyfrowane, klucz)
24
25 print("zaszyfrowane: " + zaszyfrowane)
26 print("odszyfrowane: " + odszyfrowane)
```

klasy

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

podwójny podkreślnik

```
1 class Osoba:
2     def __init__(self, imie, wiek): # konstruktor – metoda wywoływana przy tworzeniu obiektu.
3         self.imie = imie           # pole klasy
4         self.wiek = wiek           # pole klasy
5
6     def przywitaj(self):            # metoda klasy
7         print(f"Cześć, mam na imię {self.imie} i mam {self.wiek} lat.")
8
9
10 # utworzenie obiektu (instancji)
11 ala = Osoba("Ala", 12)
12
13 # wywołanie metody
14 ala.przywitaj()
```

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

Cześć, mam na imię Ala i mam 12 lat.

klasy – atrybuty prywatne

Atrybuty zaczynające się od podwójnego podkreślenia są traktowane jako prywatne.

```
1 class Konto:
2     def __init__(self, saldo):
3         self.__saldo = saldo # atrybut „prywatny”
4
5     def pokaz_saldo(self):
6         return self.__saldo
7
8     def wpłata(self, kwota):
9         self.__saldo += kwota
```

do atrybutu `__saldo`
nie można się dostać
tradycyjnym
sposobem

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

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

```
12 mojeKonto = Konto(100)
13 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)

```
1 class Konto:
2     def __init__(self, saldo):
3         self._saldo = saldo # atrybut "chroniony"
4
5     def pokaz_saldo(self):
6         return self._saldo
7
8     def wpłata(self, kwota):
9         self._saldo += kwota
10
11
12 mojeKonto = Konto(100)
13 print(mojeKonto._saldo) # działa
```

klasy – @property

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

```
1 class Konto:
2     def __init__(self, saldo):
3         self.__saldo = saldo
4
5     @property                                # getter
6     def saldo(self):
7         return self.__saldo
8
9     @saldo.setter                            # setter
10    def saldo(self, kwota):
11        if kwota < 0:
12            raise ValueError("Saldo nie może być ujemne!")
13        self.__saldo = kwota
14
15
16 mojeKonto = Konto(100)
17 mojeKonto.saldo = 200    # wywołuje setter
18 print(mojeKonto.saldo)  # wywołuje getter
```

klasy - dziedziczenie

```
1  class Zwierze:                                # klasa bazowa Zwierze
2      def __init__(self, gatunek):               # konstruktor klasy bazowej
3          self.gatunek = gatunek
4
5      def info(self):
6          print("Jestem zwierzęciem:", self.gatunek)
7
8
9  class Pies(Zwierze):                          # klasa potomna Pies dziedziczy po klasie Zwierze
10     def szczekaj(self):
11         print("Hau hau!")
12
13
14  p = Pies("Owczarek")
15  p.info()                                     # odziedziczona metoda z klasy bazowej Zwierze
16  p.szczekaj()                                # metoda z klasy Pies
```

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