Będziemy się uczyć

PROGRAMOWANIE:
- język z kontrolą typu (C++)
- język dynamicznie typowany (Python)
- języki deklaratywne (vs imperatywne)

piszę, co ma robić

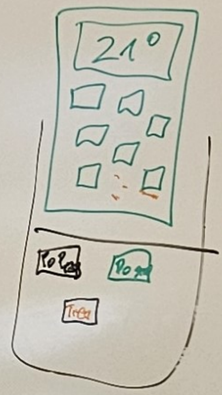
C++, Java, Python, Rust,

który lepszy

czas_programisty

sprytny

poziom abstrakcji



- kompilowane
- interpretowane ~ opowiedzieć

C++ pętle warunki

jak ma wyglądać wynik?

SQL

Arduino
Raspberry Pi

```
print('Dobry wieczór')
print(0.3+0.3+0.3)
```

1, 2, 4, 3, 2, 8

1, 2, 2, 3, 4, 8

jaroslaw.drapala@pwr.edu.pl

byes.pl

STRUKTURY DANYCH

Python : lista, słownik, krotka

```
x = 4
```

```
x = x + 1
print(x)
```

```
L = ['Brygida', 'Bożydar', 'Tamara', 'Kleofas', 4.02, 4-2j]
```

```
L2 = L[:4]
print(L2)
```

```
L3 = L[:]
L3 = L.copy()
```

```
L3[1] = 'Alcest'
```

```
~~print(L2)~~
print(L)
```

```
print(L3 is L)
print(id(L3))
id(L)
```

```
type(L[2])
len(L)
```

```
L[len(L)-1]
L[-1]
```

```
L[1:3]
L[1:]
L[0:3]
```

```
L[:3]
L[:]
```

goscie = ['Tamara', 'Lubawa', 'Rokosz', ...]

print(goscie)

[1]

W domu

and
or

[*]

```
for elem in goscie:
    print(f'Witaj {elem}')
```

```
rodzic = '....'
i = 0
while goscie[i] != rodzic:
    print(f'Witaj {goscie[i]}')
    i += 1
```

```
if elem in lista:
    print(f'{elem} jest zaproszony')
else:
    print('Nie znam')
```

print [elem in lista]

3) funkcja

fib(N)

zwraca N-ta

liczbę fibonacciego

w domu

produkty = ['lampa', 'hulajnoga', ...

...]

zamówienie = ['hulajnoga', 'tadowarka']

Czy zamówienie da się zrealizować?

1) bez funkcji

2) def dodaj(a, b):

wynik = a + b

return wynik

nauczyć się myśleć w sposób deklaratywny (jak ma „wyglądać" wynik)

SQL ← pseudokod ⇝ bo przekażę innym swoje pomysły

Python ⇝ bo szybko prototypuję

C++ ⇝ bo wszystko kontroluję

{ myślę co/jak maszyna ma robić } Python C++

program docelowy

coś banalnego

[piszę programy krok po kroku
eksperymentuję z kodem]

code snippets

```
lista = ['raz', 'dwa', 'trzy']
```

myślę w C++

```
for i in range(len(lista)):
    print(lista[i])
```

Wrocław wroctaw

```
for elem in lista:
    elem = elem.upper()
print(lista)
```

lista[1] = 'Dwa'

'string'.upper()

W domu napisać funkcję

```
def na_wielkie(lista):
```

oryginalna ma być nietknięta

```
def dodaj_jeden(x):
    return x+1
```

$ax^2 + bx + c$

```
>> dodaj_jeden(x=8)
```

```
def rkwad(x, a=2, b=1, c=1):
    wynik = a*x**2 + b*x + c
    return wynik
```

```
>> rkwad(4, 2, -1, 1)
>> rkwad(x=4, c=1, a=2)
```

jawne jest lepsze od niejawnego

code snippets

```
lista = ['raz', 'dwa', 'trzy']
```

myślę w C++

```
for i in range(len(lista)):
    print(lista[i])
```

```
lista_tmp = lista[:]
lista_tmp = lista.copy()
```

```
for elem in lista:
    elem = elem.upper()
print(lista)
```

WRocław
wroctaw

lista[1] = 'Dwa'

'string'.upper()

W domu
napisać funkcję

```
def na_wielkie(lista):
```

oryginalna ma być nietknięta

Krotka

mutowalne niemutowalne

zmienialne niezmienialny

You've made my day

```
K = ('raz', 'dwa', 'trzy')
```

```
for elem in K:
    print(elem.title())
```

int* w

void*

screen saver

Braveheart

artificial intelligence

```
K[1] = 'DWA'
```

```
K = 4
```

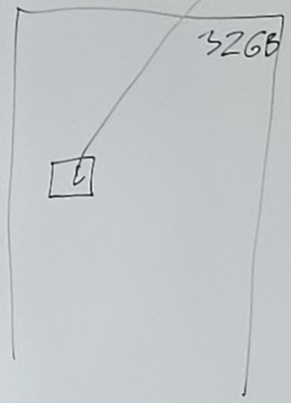
```
zielony = (0, 255, 0)
```

```
ulubiony_kolor = (126, 17, 48)
```

```
fig, ax = plt.subplots(nrows=1,
                       ncols=2,
                       figsize=(10, 5))
ax[0].plot([8, 7, 8, 4, 1])
ax[0].set_xlabel('czas')
ax[0].set_ylabel('zarobki')
ax[0].set_title('moje życie zawodowe')
ax[1].scatter([170, ....], [....])
ax[1] ... 'wzrost'
          'waga'
plt.show()
```

RAM

Untitled4.py




```
plot([1, 2, 3, 1, 2, 3])
#show()
from matplotlib.pyplot import plot, show, scatter
scatter([170, 162, 188, 177], [77, 82, 65, 71])
show()
```

```
from matplotlib.pyplot import *
```

```
import matplotlib.pyplot as plt
plt.plot([3, 8, 3, 4])
plt.show()
```

```
O=[[1, 2, 3], [4, 5, 6], [7, 8, 9],
   [10, 11, 12]]
O[1][2]
plt.imshow(O,
           cmap='inferno')
           'inferno_r')
plt.axis('off')
axes
```

w Jupyterze	w normalnym edytorze
myślimy o liniowym porządku wykonania	myślimy modularnie

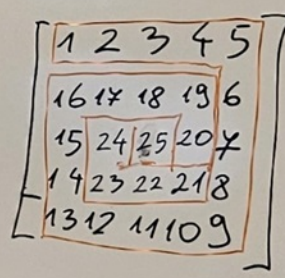
kiedy for	while
wiemy z góry ile razy coś robimy	nie wiadomo z góry ile razy coś wykonać

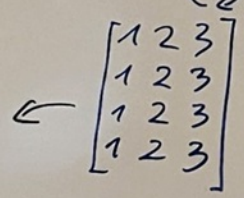
```
L = []
L.append(3)
L.append('w-w')
L.append(['a','b','c'])
L.extend(['a','b','c'])
```

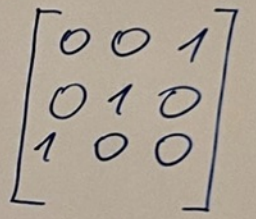
```
for elem in lista:
    print(elem)
```

%reset

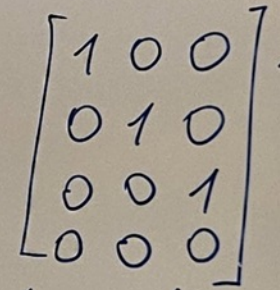
```
import matplotlib.pyplot as plt

def lista_zer(nrows=3, ncols=3):
    ...

L = lista_zer(8, 9)
plt.imshow(L, cmap='binary')
plt.show()
```







L[4][7] = 1



1. pojedynczy for
2. podwójny for

chwilowa = [. . . . .]

L = [chwilowa,
chwilowa
:
]

dlugosc_dziatki = 3

; = 3

obwód_działki = 2 * (a + b)

i

L[0][0] = 1

[[1 0 0 0],
[1 0 0 0],
[1 0 0 0],
[
]

i = [ ]

L

L[0][0] = 1
print (L)
L[4][8] = 2

N

:

```
import matplotlib.pyplot as plt
import numpy as np

lista = [1, 2, 3]
print(lista + lista)

tab = np.array([1,2,3])
print(tab + tab)
print(lista + [1])
print(tab + 1)

lista_z_tabeli = list(tab)
type(lista_z_tabeli)
tabela_z_listy = np.array(lista)
type(....)
```

W domu

eksperymenty z tabelami

- różne wymiary
- różnica w wycinkach między tabelą a listą

  lista[3:8] vs tab[3:8]

- tab ← dodaj na koniec 4

  → do kwadratu wszystkie

- co to jest .ndim
  .size
  .shape
  ?

```
import matplotlib.image as im

obrazek = im.imread('obrazek.jpg')

plt.imshow(obrazek + 20)
plt.axis('off')
plt.show()
```

plus ←

```
obrazek.ndim      3
        .size
        .shape    (...
```

```
obrazek=np.min(obrazek, 255)
```