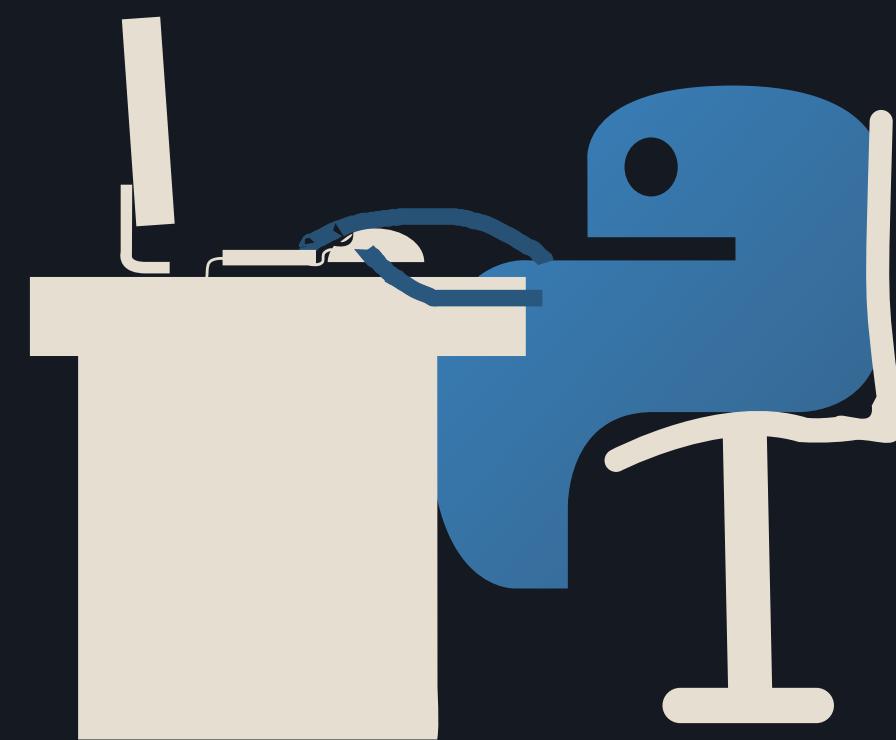


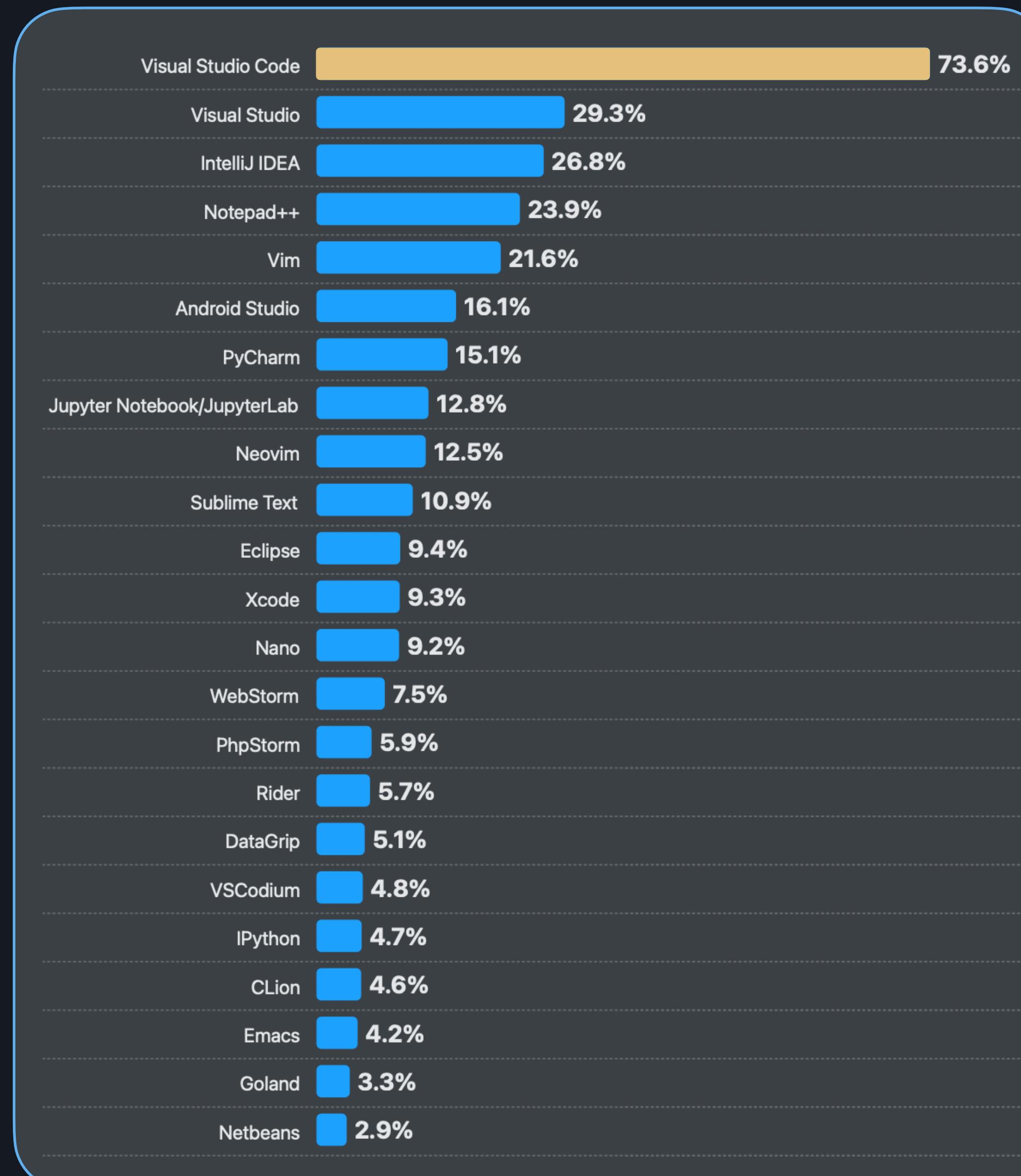
# Python

## Nyelvi alapok

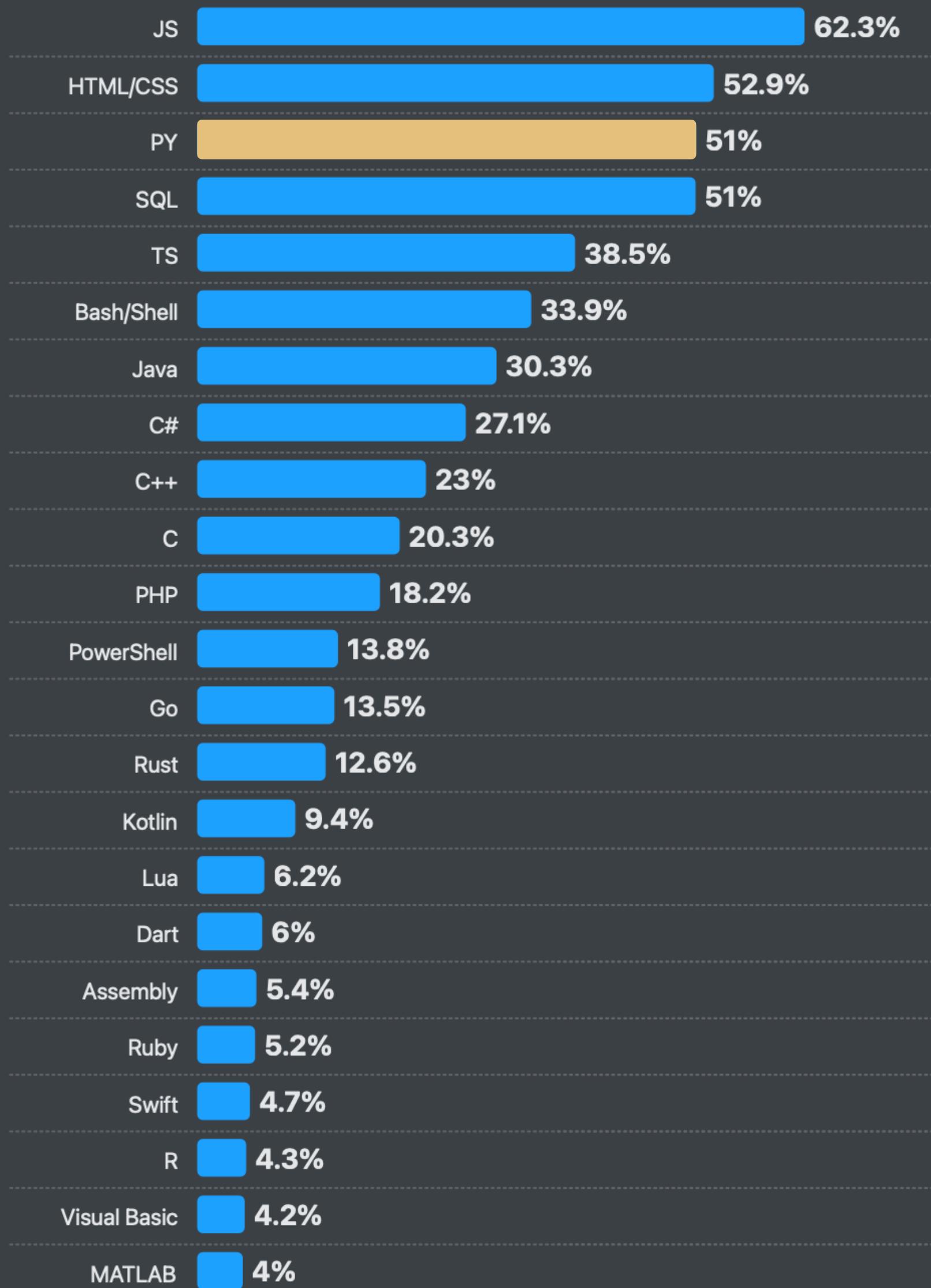


Csuzdi Domonkos, BME KJIT

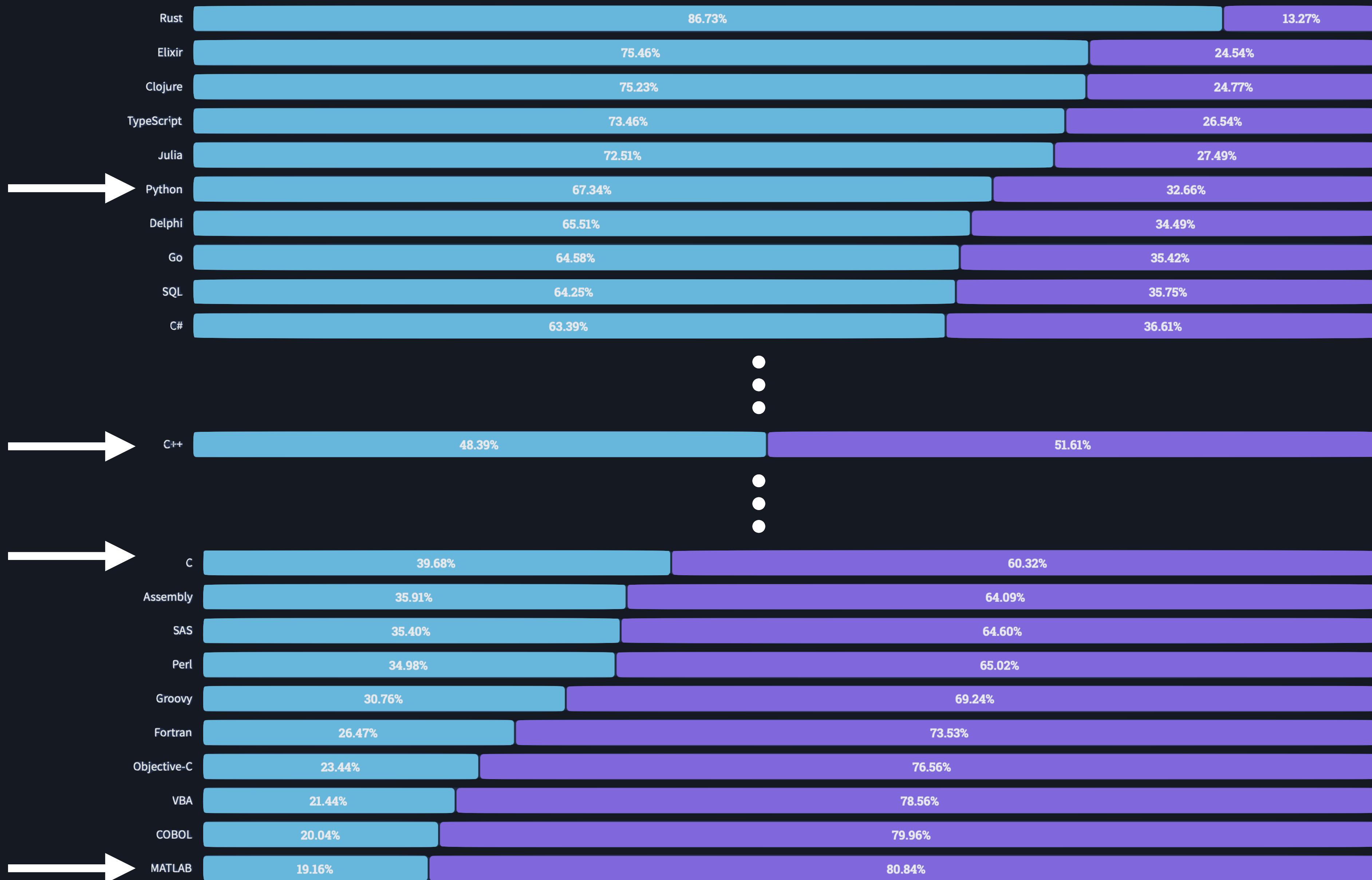
# Legnépszerűbb szerkesztő



# Legnépszerűbb nyelv



# Loved vs. dreaded



# Bevezetés

## Miért Python?

nyílt forráskódú

magas szintű

ingyenes

objektumorientált

moduláris

gyengén típusos

cross-platform

interpretált<sup>1</sup>



# Bevezetés

## VS Code

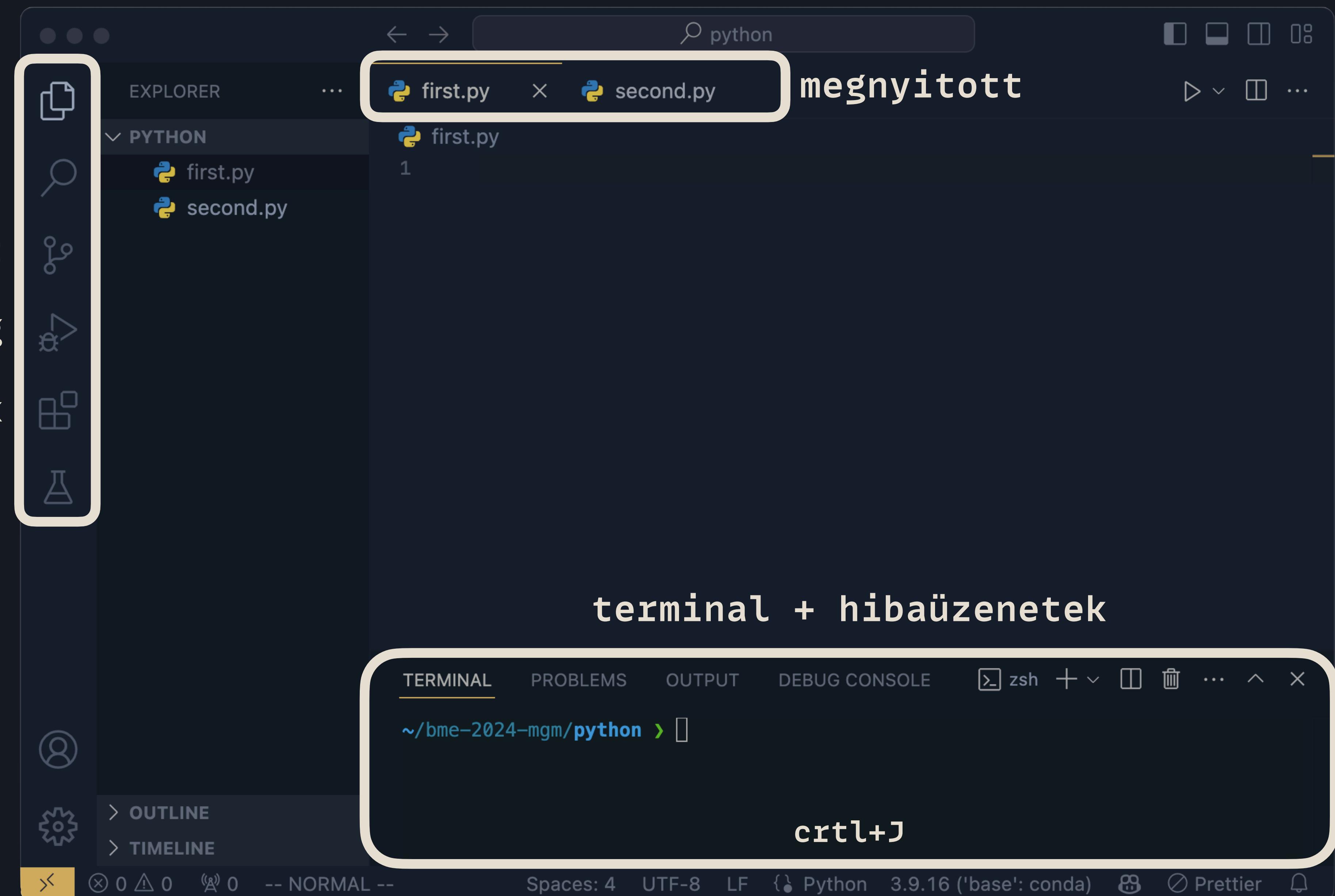
fájlok (workspace)

keresés

git

debug

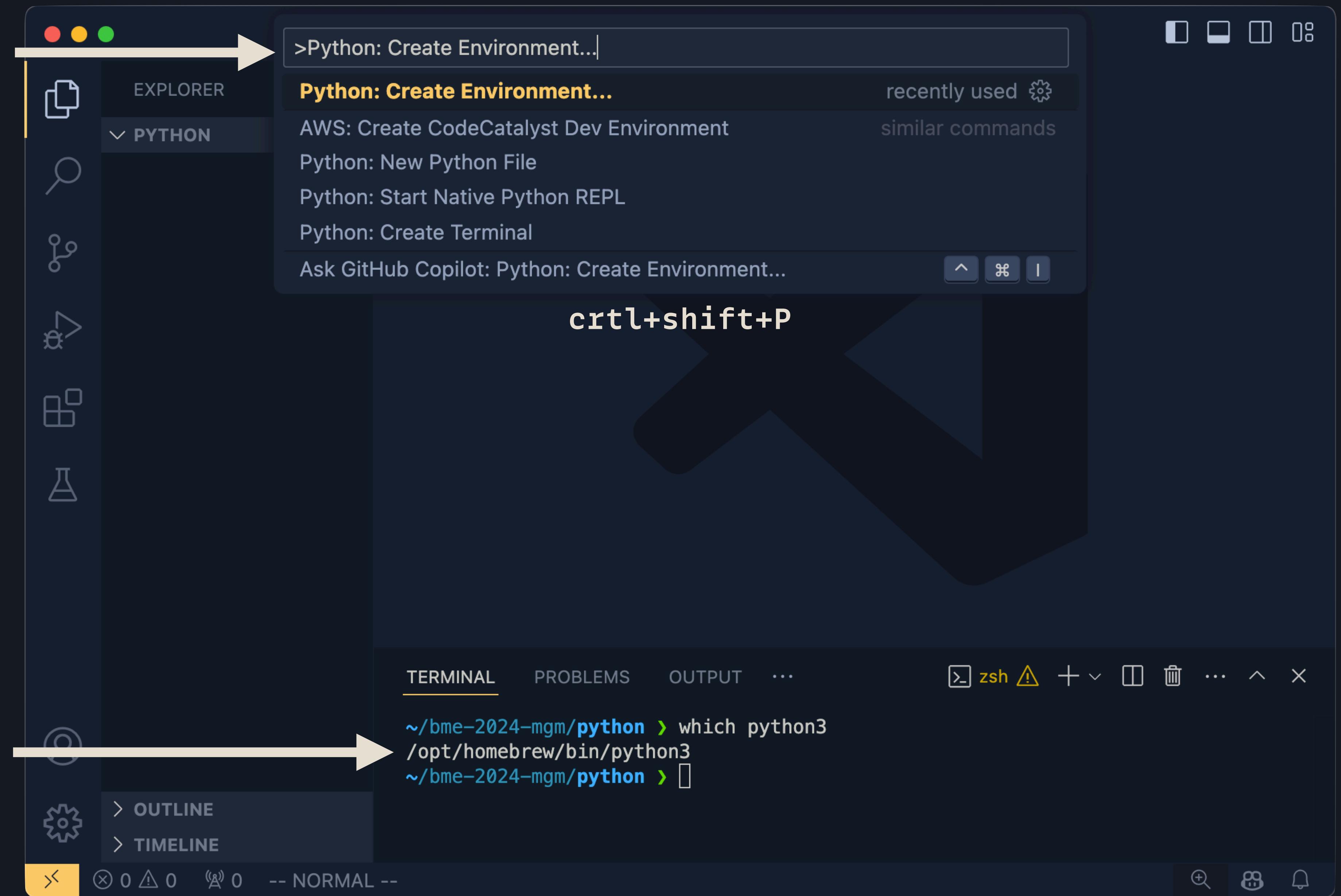
kiegészítők



# Python virtual environment

2. csinálunk  
egy lokált

1. ez a  
globál Python

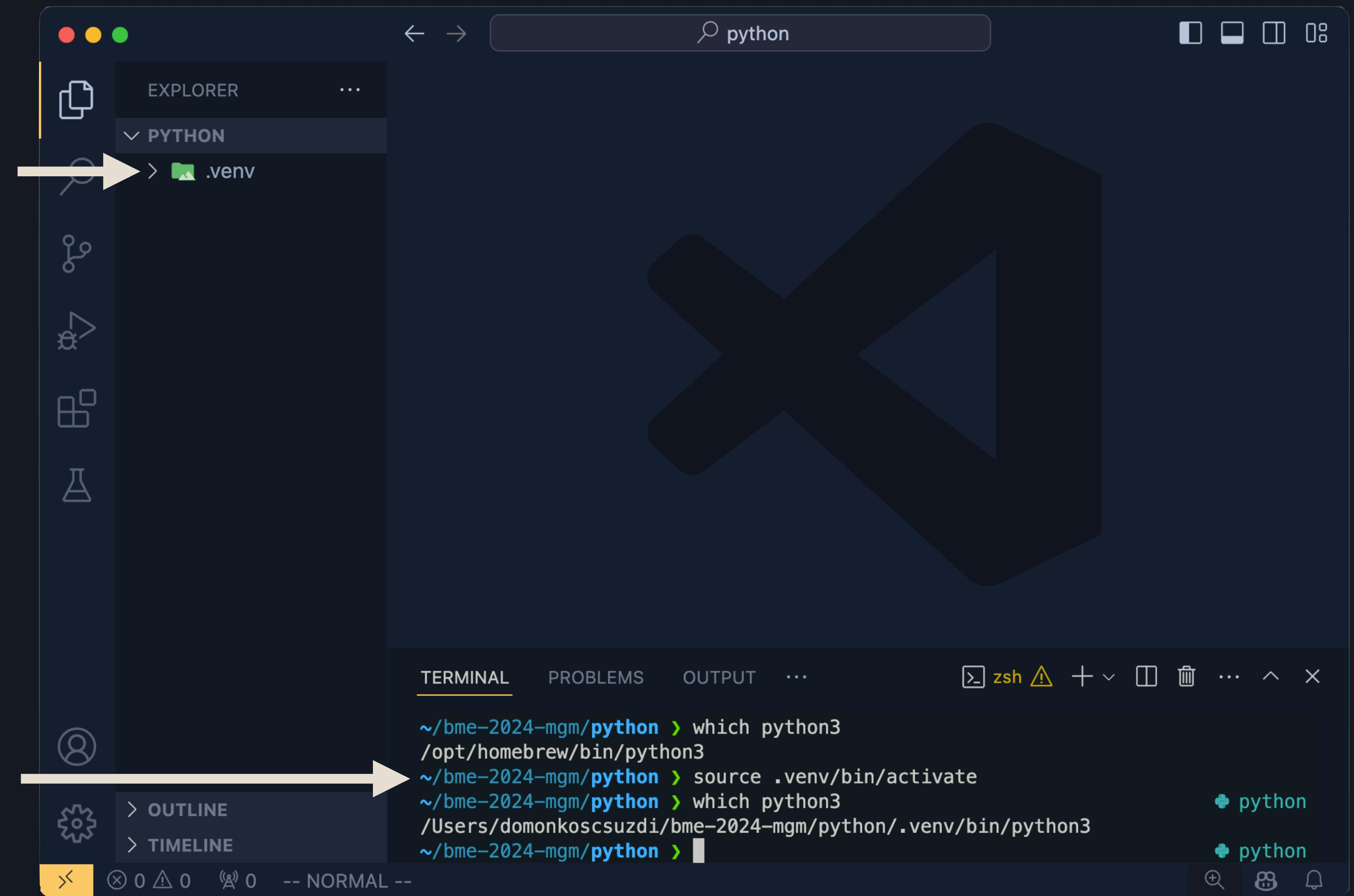


# Python virtual environment

Igazából csak  
egy mappa

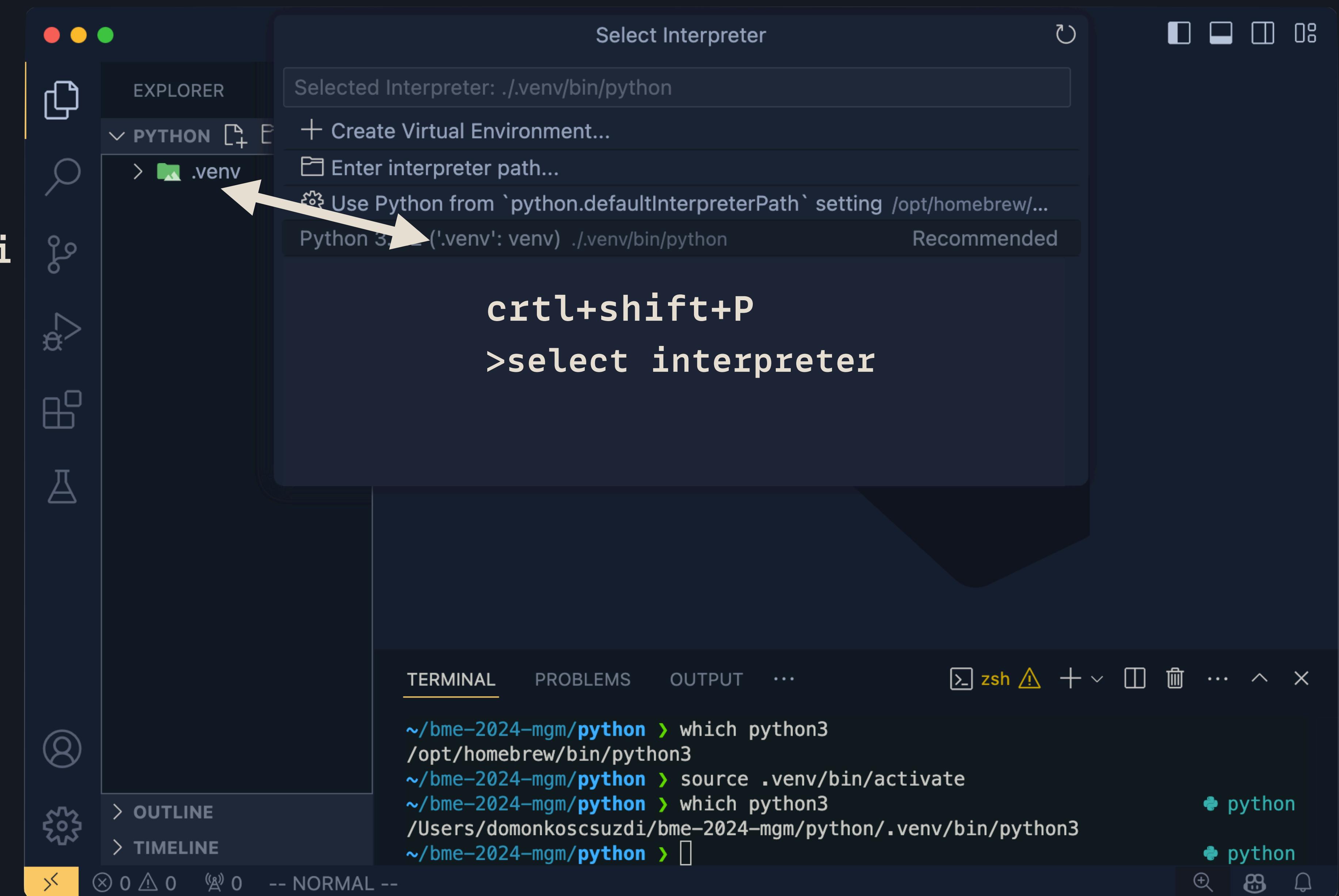
Itt lesznek a  
feltelepített  
csomagjaink

használjuk a  
terminálban



# Python interpreter

Ha az editorból  
futtatunk,  
akkor itt is  
ki kell választani



# Python

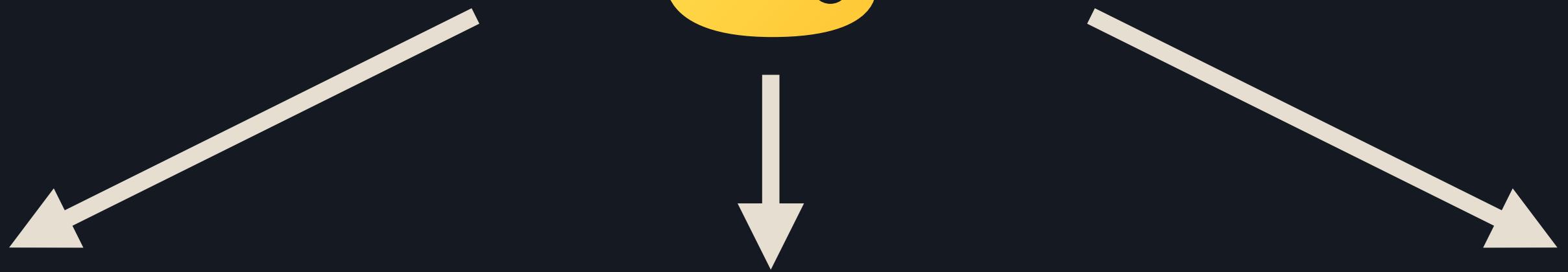
## Read Evaluate Print Loop



terminal

```
~/bme-2024-mgm/python > python3
Python 3.8.2 (default, Apr  8 2021, 23:19:18)
[Clang 12.0.5 (clang-1205.0.22.9)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> r = 3
>>> r
3
>>> 2*r*pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
>>> exit()
```

# Python csomagok/modulok



## Adatelemzés

- numpy
- pandas
- matplotlib
- ...

## Gépi tanulás

- pytorch/tensorflow
- scikit-learn
- scipy
- ...

## Webfejlesztés

- django
- flask
- pyramid
- ...

és még sok más...

# Csomagok használata, telepítése

## Beépített csomagok

The Python Standard Library<sup>1</sup>

- os: operárendszer spec.
- sys: parancssor
- time: pillanatnyi idő
- json: JSON fájlok
- math: matek
- antigravity: ☢
- ...

## Külső csomagok

The Python Package Index<sup>2</sup>  
(PyPi)

- letöltés és telepítés: pip
- példa (terminal):  
pip install numpy

#TODO

- anaconda

## Saját csomagok

- átláthatóság
- modularitás
- feltölthetjük a PyPi-re<sup>3</sup>
- lokális importok

# Csomag import



teljes csomagot

```
import numpy  
numpy.__version__
```

csak egy részét

```
import matplotlib.pyplot
```

alias

```
import numpy as np  
np.__version__
```



egy függvényt/változót

```
from numpy import pi  
print(pi)
```

kevésbé olvasható a kód



minden fv-t,változót

```
from numpy import *  
print(pi)
```



telerakjuk namespace-t

# Beépített csomagok



```
terminal
~/bme-2024-mgm/python > python3
Python 3.8.2 (default, Apr  8 2021, 23:19:18)
[Clang 12.0.5 (clang-1205.0.22.9)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> r = 3
>>> import math
>>> 2*r*math.pi
18.84955592153876
>>> exit()
```

# Külső csomagok

1. telepítjük
2. itt is van

The screenshot shows the Visual Studio Code interface with the Python extension installed. The Explorer sidebar on the left has a 'PYTHON' section expanded, showing subfolders like '.venv', 'bin', 'include', 'lib/python3.8...', '\_\_pycache\_\_', 'numpy', 'numpy-1.24.4...', 'pip', 'pip-24.2.dist-i...', 'pkg\_resources', 'setuptools', 'setuptools-41...', and 'easy\_install.py'. A large white arrow points from the text '1. telepítjük' to the 'easy\_install.py' file. Another large white arrow points from the text '2. itt is van' to the 'numpy' folder in the Explorer. The terminal at the bottom shows the command 'pip3 install numpy' being run, with output indicating the package is being collected and installed successfully.

```
~/bme-2024-mgm/python > pip3 install numpy
Collecting numpy
  Using cached numpy-1.24.4-cp38-cp38-macosx_11_0_arm64.whl.metadata (5.6 kB)
  Using cached numpy-1.24.4-cp38-cp38-macosx_11_0_arm64.whl (13.8 MB)
Installing collected packages: numpy
Successfully installed numpy-1.24.4
~/bme-2024-mgm/python >
```

# Külső csomagok



terminal

```
~/bme-2024-mgm/python > python3
Python 3.8.2 (default, Apr  8 2021, 23:19:18)
[Clang 12.0.5 (clang-1205.0.22.9)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> r = 3
>>> import numpy as np
>>> 2*r*np.pi
18.84955592153876
>>> exit()
```

# Python modul (fájl)

2. vagy az editorból

1. futtathatjuk  
a terminálból

The screenshot shows the Visual Studio Code interface with the following details:

- Code Editor:** Shows a file named `first.py` with the content:

```
1 print("Hello world!")
```
- Terminal:** Shows the output of running the script:

```
~/bme-2024-mgm/python > python3 first.py
Hello world!
~/bme-2024-mgm/python >
```
- Explorer:** Shows a Python environment named `.venv` and a file named `first.py`.
- Status Bar:** Shows file statistics ( $\otimes 0 \triangle 0 \text{ 0}$ ), mode (`-- NORMAL --`), cursor position (`Ln 2, Col 1`), and encoding (`UTF-8 LF`).

# Python debug

folytat      léptet

Run and Debug

vagy

F5

#TODO

- watch
- conditional breakpoint

The screenshot shows the Visual Studio Code interface during a Python debugging session. A yellow callout box labeled "breakpoint" points to a red circular breakpoint marker on line 5 of the code. The code itself calculates the area of a circle with radius 3.

```
4 import math
5
6 r = 3
7 area = r**2 * math.pi
8 print(f"A(z) {r} sugarú kör területe: {round(area,3)}")
```

A large white arrow points from the "REPL a változókkal" text towards the "DEBUG CONSOLE" tab, which displays the output of the print statement: "2\*r" and "6".

At the top, there are two large arrows pointing downwards: one labeled "folytat" (continue) and one labeled "léptet" (step).

On the left, the "Variables" sidebar shows the local variables: "math" (module), "r" (value 3), and "area" (calculated value 6). The "Breakpoints" sidebar shows a single breakpoint at line 5.

At the bottom, the status bar shows "Spaces: 4" and "Python 3.8.2 ('.venv': venv)".

# Adatstruktúrák

## list

changeable, ordered, allow duplicates

```
my_list = [1, 2.0, 1j, "five", [4, 5]]  
my_list.append(6) # [1, 2.0, 1j, "five", [4, 5], 6]  
removed = my_list.pop() # removed: 6, my_list: [1, 2.0, 1j, "five", [4, 5]]  
del my_list[0] # [2.0, 1j, "five", [4, 5]], no return val  
  
start = 1 # inclusive  
end = 3 # exclusive  
step = 2  
new_list = my_list[start:end:step] # [1j]  
reversed_list = my_list[::-1] # [[4, 5], "five", 1j, 2.0]
```

### #TODO

- sort, sorted
- kapcsolat a stringekkel
- egyéb műveletek

# Adatstruktúrák kitérő: numpy array

## Mátrixok, vektorok, transzponált

### Motiváció

```
my_list = [1, 2, 3]
my_list *= 2 # [1, 2, 3, 1, 2, 3]
```

```
import numpy as np

my_array = np.array([1, 2, 3])
# or np.array(my_list)
my_array *= 2 # [2 4 6]
```

```
my_vector = np.array([5, 6])
my_vector.T # [5 6], not as expected!
my_vector.shape # (2,)
my_vector.ndim # 1
```

```
my_vector = np.array([[5, 6]])
my_vector.T # [[5], [6]]
my_vector.shape # (1,2)
my_vector.ndim # 2
my_matrix = np.array([[1, 2], [3, 4]])
my_matrix @ my_vector.T # [[17], [39]]
```

# Adatstruktúrák

## tuple

unchangeable, ordered, allow duplicates

```
my_tuple = (1, 2, 3)
a, b, c = my_tuple # unpacking, a: 1, b: 2, c: 3
a, _, _ = my_tuple # do not waste memory
a, *rest = my_tuple # a: 1, rest: [2, 3]
my_tuple[0] # 1
my_tuple[1] = 4 # unchangeable! → TypeError

# tuple with one element
new_tuple = (1) # 1, does not work!
new_tuple = 1, # (1,)
```

### #TODO

- függvények hol használják?
- enumerate()

# Adatstruktúrák

## dictionary

changeable, ordered, no duplicates

```
my_string = "hello"

char_dict = {}
for char in my_string:
    if char in char_dict:
        char_dict[char] += 1
    else:
        char_dict[char] = 1
print(char_dict) # {'h': 1, 'e': 1, 'l': 2, 'o': 1}

del char_dict["l"] # {'h': 1, 'e': 1, 'o': 1}
for key,value in char_dict.items():
    print(key, value) # h 1, e 1, o 1
```

#TODO

- melyik Py. verzió óta ordered?

# Adatstruktúrák

## set

unchangeable<sup>1</sup>, unordered, no duplicates<sup>1</sup>

### Alapok

```
my_set = {1, 2, 3}  
my_set.add(4) # {1, 2, 3, 4}  
my_set.remove(3) # {1, 2, 4}  
my_set[0] # unordered → TypeError
```

### Apró trükk

```
# discard duplicates  
my_list = [1, 2, 2, 4, 5]  
unique_numbers = len(list(set(my_list))) # 4
```

#TODO

- set halmazműveletek

Miért használjuk őket?  
 $\mathcal{O}(1)$  vs.  $\mathcal{O}(n)$

```
import time  
  
my_list = list(range(1000000))  
my_set = set(my_list)  
  
start = time.time()  
999999 in my_list  
print(f"List lookup: {time.time() - start}")  
# 0.007 s  
  
start = time.time()  
999999 in my_set  
print(f"Set lookup: {time.time() - start}")  
# 2.86e-6 s
```

# Függvények

```
def add_two(arg_1, arg_2):  
    """docstring"""  
    total = arg_1 + arg_2  
    return total
```

4 db space/1 db tab

#TODO

- \*args, \*\*kwargs
- type hint
- if \_\_name\_\_ == "\_\_main\_\_"

```
def add_nums(arg_1: int, *args: int, **kwargs) → int:  
    """
```

Adds one or more numbers and prints any additional keyword arguments.

Args:

- arg\_1 (int): The first number.
- \*args (int): Additional numbers to add.
- \*\*kwargs: Additional keyword arguments.

Returns:

- int: The sum of all provided numbers.

```
    """
```

```
    total = arg_1 + sum(args)
```

```
    print(f"Additional keywords: {kwargs}")
```

```
    return total
```

```
print(add_nums(2, 3, 4, 5, extra1="val1", extra2="val2"))  
# Additional keywords: {'extra1': 'val1', 'extra2': 'val2'}  
# 14
```

# Vezérlési szerkezetek

```
x = 5
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is less than 5")
```

```
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    if num == 2:
        continue
    elif num == 3:
        pass
    elif num == 4:
        break
    else:
        print(num)
```

```
i = 0
while i < 5:
    print(i)
    i += 1
```

## #TODO

- exception-ök (`ZeroDivisionError`)
- `finally`
- do-while
- `match-case`
- walrus operator :=

```
try:
    x = 1 / 0
except Exception as e:
    print(f"Encountered a problem: {e}")
```

# The Pythonic Way

## list comprehension, ternary operator

```
squares = []
for x in range(5):
    squares.append(x**2)
```

```
list(range(5))
# [0, 1, 2, 3, 4]
```

```
even_squares = []
for x in range(5):
    if x % 2 == 0:
        even_squares.append(x**2)
```

#TODO

- nested list comprehension
- dictionary comprehension

```
squares = [x**2 for x in range(5)]
```



```
even_squares = [x**2 for x in range(5) if x % 2 == 0]
```

## ternary operator

```
num = 4
result = "Even" if num % 2 == 0 else "Odd"
print(result) # Output: "Even"
```

# Objektumorientált programozás 1. alapok

A Pythonban minden egy objektum

**objektum = adattagok + metódusok**

```
a = 3 + 4j # complex number
print(type(a)) # <class 'complex'>
real = a.real # property
imag = a.imag # property
conj = a.conjugate() # method
(komplex konjugált)
```

```
a = 3 + 4j
b = 2 + 5j
c = a.__add__(b)
# c = a + b (egyszerűbben)
```

# Objektumorientált programozás 2. osztály létrehozása

```
class Pose2D:  
    """A type to represent 2D pose (x, y, rot)"""  
  
    def __init__(self, x, y, theta):  
        self.x = x # adattagok  
        self.y = y  
        self.theta = theta  
  
    # metódusok  
    def to_degrees(self):  
        """Return the rotation angle in degrees."""  
        return np.degrees(self.theta)  
  
    def add(self, other):  
        x = self.x + other.x  
        y = self.y + other.y  
        theta = self.theta + other.theta  
        return Pose2D(x, y, theta)
```

```
pose_1 = Pose2D(1, 2, np.pi/4)  
pose_2 = Pose2D(0.5, 0.5, np.pi/4)  
pose_3 = pose_1.add(pose_2)  
print(pose_3.x, pose_3.y, pose_3.theta)
```

#TODO

- “dunder” metódusok
- privát adattagok

# Objektumorientált programozás 2. osztály létrehozása

```
class Pose2D:  
    """A type to represent 2D pose (x, y, rot)"""  
  
    def __init__(self, x, y, theta):  
        self.x = x # adattagok  
        self.y = y  
        self.theta = theta  
  
    # metódusok  
    def to_degrees(self):  
        """Return the rotation angle in degrees."""  
        return np.degrees(self.theta)  
  
    def add(self, other):  
        x = self.x + other.x  
        y = self.y + other.y  
        theta = self.theta + other.theta  
        return Pose2D(x, y, theta)
```

```
pose_1 = Pose2D(1, 2, np.pi/4)  
pose_2 = Pose2D(0.5, 0.5, np.pi/4)  
pose_3 = pose_1.add(pose_2)  
print(pose_3.x, pose_3.y, pose_3.theta)
```

# Objektumorientált programozás 2. osztály létrehozása

```
class Pose2D:  
    """A type to represent 2D pose (x, y, rot)"""  
  
    def __init__(self, x, y, theta):  
        self.x = x # adattagok  
        self.y = y  
        self.theta = theta  
  
    # metódusok  
    def to_degrees(self):  
        """Return the rotation angle in degrees."""  
        return np.degrees(self.theta)  
  
    def add(self, other):  
        x = self.x + other.x  
        y = self.y + other.y  
        theta = self.theta + other.theta  
        return Pose2D(x, y, theta)
```

```
pose_1 = Pose2D(1, 2, np.pi/4)  
pose_2 = Pose2D(0.5, 0.5, np.pi/4)  
pose_3 = pose_1.add(pose_2)  
print(pose_3.x, pose_3.y, pose_3.theta)
```

# Objektumorientált programozás 3. öröklődés

```
class Pose2D:  
    def __init__(self, x, y, theta):  
        self.x = x  
        self.y = y  
        self.theta = theta  
    def to_degrees(self):  
        ...  
    def add(self, other):  
        ...
```

```
class Pose2DStamped(Pose2D):  
    """2D pose with timestamp"""  
  
    def __init__(self, x, y, theta, timestamp):  
        super().__init__(x, y, theta)  
        self.timestamp = timestamp  
  
    def add(self, other):  
        n_p = super().add(other)  
        n_stamp = max(self.timestamp, other.timestamp)  
        return Pose2DStamped(n_p.x, n_p.y, n_p.theta, n_stamp)
```

```
pose_4 = Pose2DStamped(  
    1,  
    2,  
    np.pi / 4,  
    12.13)  
pose_5 = Pose2DStamped(  
    2,  
    3,  
    np.pi,  
    23)
```

# Objektumorientált programozás 3.

## öröklődés: `super()`

```
class Pose2D:  
    def __init__(self, x, y, theta):  
        self.x = x  
        self.y = y  
        self.theta = theta  
    def to_degrees(self):  
        ...  
    def add(self, other):  
        ...
```

```
class Pose2DStamped(Pose2D):  
    """2D pose with timestamp"""  
  
    def __init__(self, x, y, theta, timestamp):  
        super().__init__(x, y, theta)  
        self.timestamp = timestamp  
  
    def add(self, other):  
        n_p = super().add(other)  
        n_stamp = max(self.timestamp, other.timestamp)  
        return Pose2DStamped(n_p.x, n_p.y, n_p.theta, n_stamp)
```

```
pose_4 = Pose2DStamped(  
    1,  
    2,  
    np.pi / 4,  
    12.13)  
pose_5 = Pose2DStamped(  
    2,  
    3,  
    np.pi,  
    23)
```

# Objektumorientált programozás 3.

## öröklődés: metódus öröklés

```
class Pose2D:  
    def __init__(self, x, y, theta):  
        self.x = x  
        self.y = y  
        self.theta = theta  
    def to_degrees(self):  
        ...  
    def add(self, other):  
        ...
```

```
class Pose2DStamped(Pose2D):  
    """2D pose with timestamp"""  
  
    def __init__(self, x, y, theta, timestamp):  
        super().__init__(x, y, theta)  
        self.timestamp = timestamp  
  
    def add(self, other):  
        n_p = super().add(other)  
        n_stamp = max(self.timestamp, other.timestamp)  
        return Pose2DStamped(n_p.x, n_p.y, n_p.theta, n_stamp)
```

```
pose_4 = Pose2DStamped(  
    1,  
    2,  
    np.pi / 4,  
    12.13)  
pose_5 = Pose2DStamped(  
    2,  
    3,  
    np.pi,  
    23)  
print(pose_4.to_degrees())
```

# Objektumorientált programozás 3.

## öröklődés: metódus felüldefiniálás

```
class Pose2D:  
    def __init__(self, x, y, theta):  
        self.x = x  
        self.y = y  
        self.theta = theta  
    def to_degrees(self):  
        ...  
    def add(self, other):  
        ...
```

```
class Pose2DStamped(Pose2D):  
    """2D pose with timestamp"""\n\n    def __init__(self, x, y, theta, timestamp):  
        super().__init__(x, y, theta)  
        self.timestamp = timestamp  
  
    def add(self, other):  
        n_p = super().add(other)  
        n_stamp = max(self.timestamp, other.timestamp)  
        return Pose2DStamped(n_p.x, n_p.y, n_p.theta, n_stamp)
```

```
pose_4 = Pose2DStamped(  
    1,  
    2,  
    np.pi / 4,  
    12.13)  
pose_5 = Pose2DStamped(  
    2,  
    3,  
    np.pi,  
    23)  
print(pose_4.to_degrees())  
print(pose_4.add(pose_5))
```

meglévő logika

új logika

# Modulok és csomagok (package)

## Osztályok elkülönítése

Csomag (mappa) neve

Modul (fájl) neve

The screenshot shows a dark-themed interface of the Visual Studio Code (VS Code) code editor. On the left is the Explorer sidebar, which displays a file tree under the 'PYTEST' folder. Inside 'PYTEST', there is a '.venv' folder, a 'geometry' folder, and a '\_\_pycache\_\_' folder containing an 'init\_\_.py' file. In the main editor area, the 'main.py' file is open, showing Python code that imports from the 'geometry.pose' module. A red arrow points from the 'init\_\_.py' file in the Explorer to the 'from geometry import pose' line in the 'main.py' code. The status bar at the bottom indicates the file is a Python file (Python 3.13.3 (.venv)) and shows the current line and column (Ln 11, Col 1).

```
10 import geometry.pose
9 n_p = geometry.pose.Pose2D(1, 2, 3)
8
7 # vagy
6 from geometry import pose
5 n_p = pose.Pose2D(1, 2, 3)
4
3 # vagy
2 from geometry.pose import Pose2D
1 n_p = Pose2D(1, 2, 3)
```

TERMINAL OUTPUT DEBUG CONSOLE ...

~/pytest > [ ]

Python + - ×

pytest base

11 Ln 11, Col 1 Spaces: 4 UTF-8 LF {} Python 3.13.3 (.venv) Go Live

# Modulok és csomagok (package) importálás

The screenshot shows a Python project structure in the Explorer sidebar. The file `pose.py` is selected in the tree view. A white arrow points from the file icon in the Explorer to the corresponding line of code in the main editor area. The editor displays the following code:

```
10 import geometry.pose
 9 n_p = geometry.pose.Pose2D(1, 2, 3)
 8
 7 # vagy
 6 from geometry import pose
 5 n_p = pose.Pose2D(1, 2, 3)
 4
 3 # vagy
 2 from geometry.pose import Pose2D
 1 n_p = Pose2D(1, 2, 3)
```

The terminal below shows the current working directory as `~/pytest`.