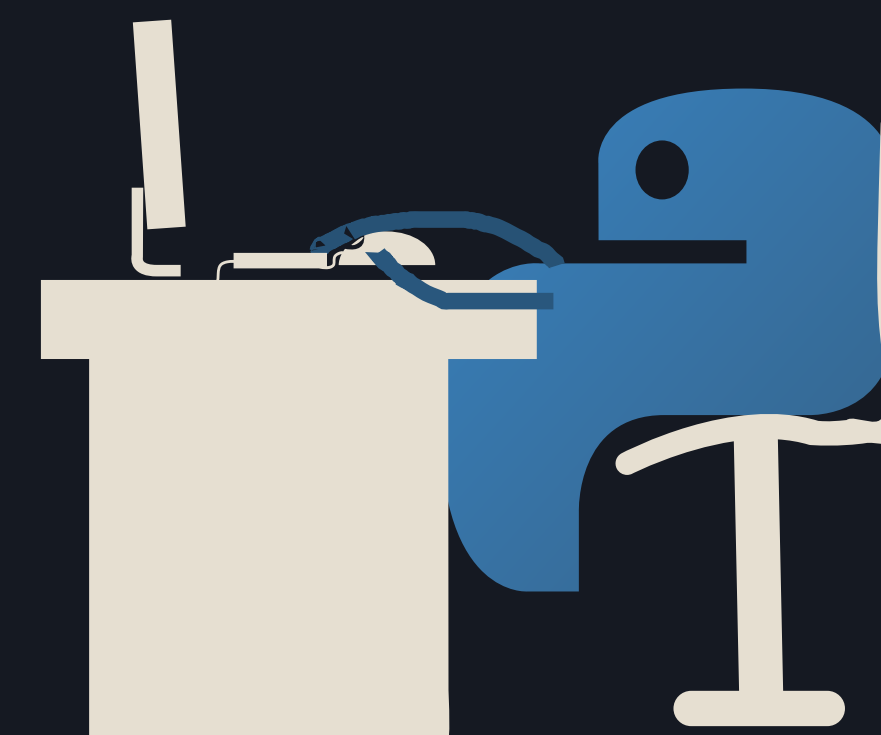




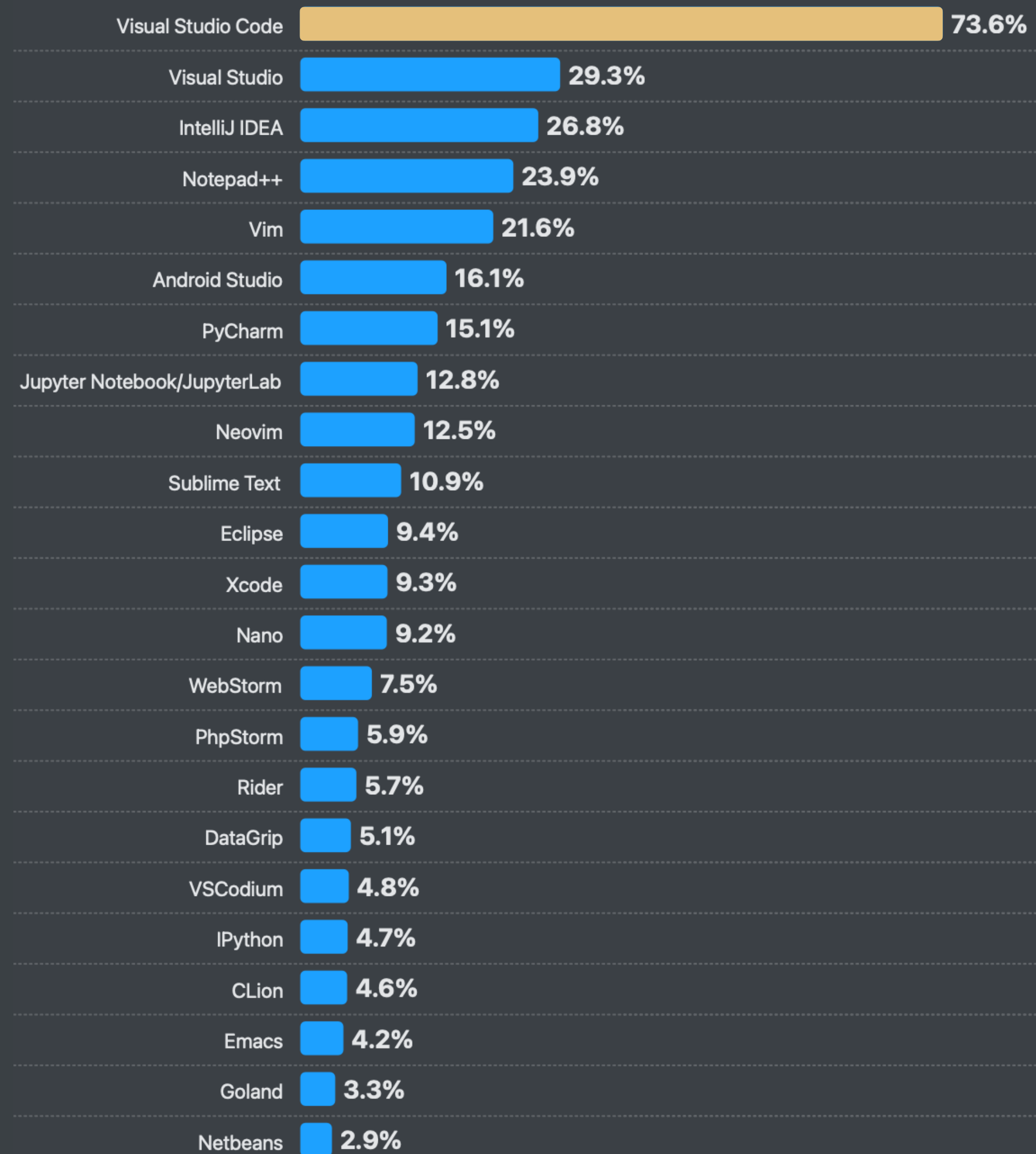
Python

Nyelvi alapok

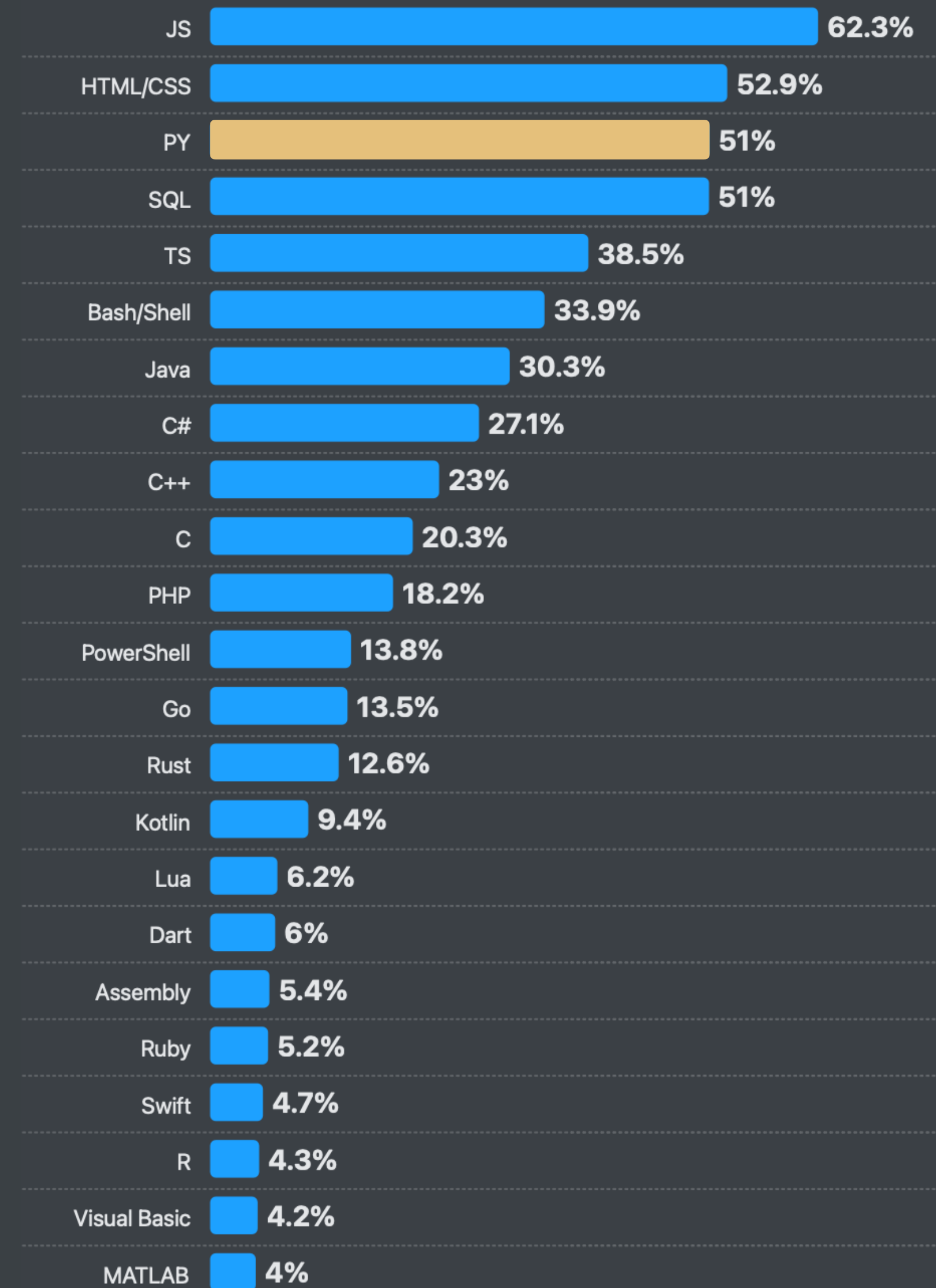


Csuzdi Domonkos, BME KJIT

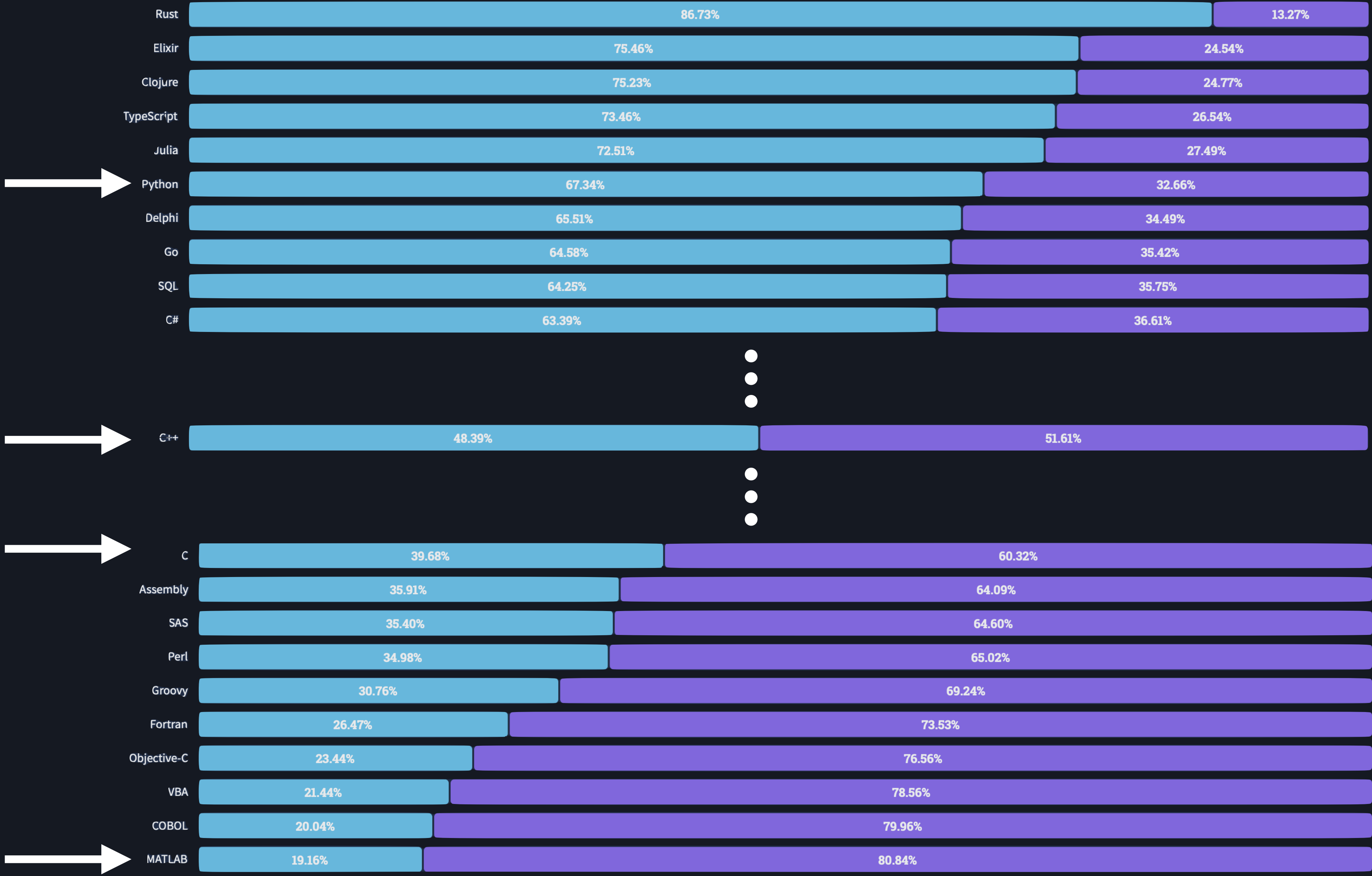
Legnépszerűbb szerkesztő



Legnépszerűbb nyelv



Loved vs. dreaded



Bevezetés

Miért Python?

nyílt forráskódú

magas szintű

ingyenes

objektumorientált

moduláris

gyengén típusos

cross-platform

interpretált¹



Bevezetés

VS Code

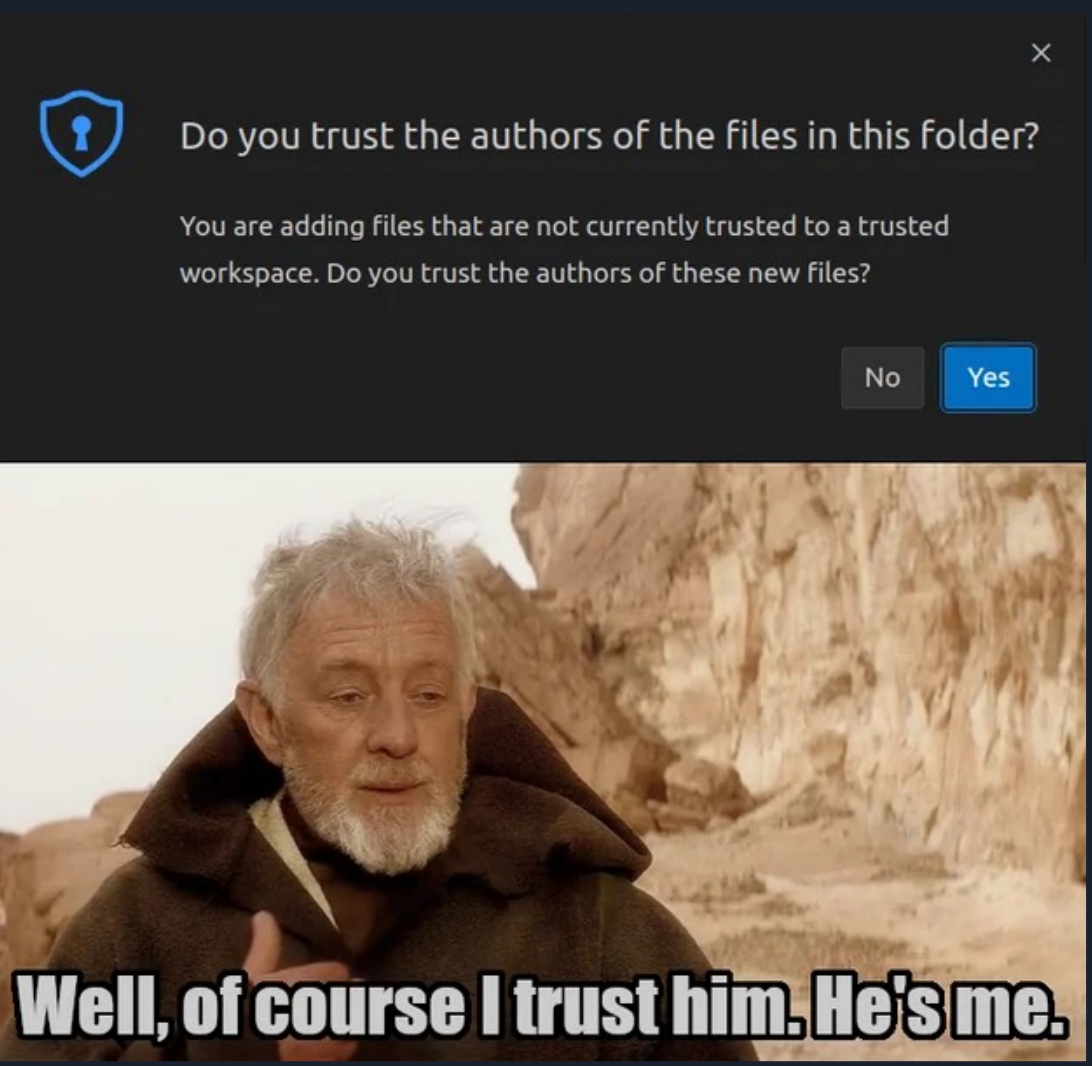
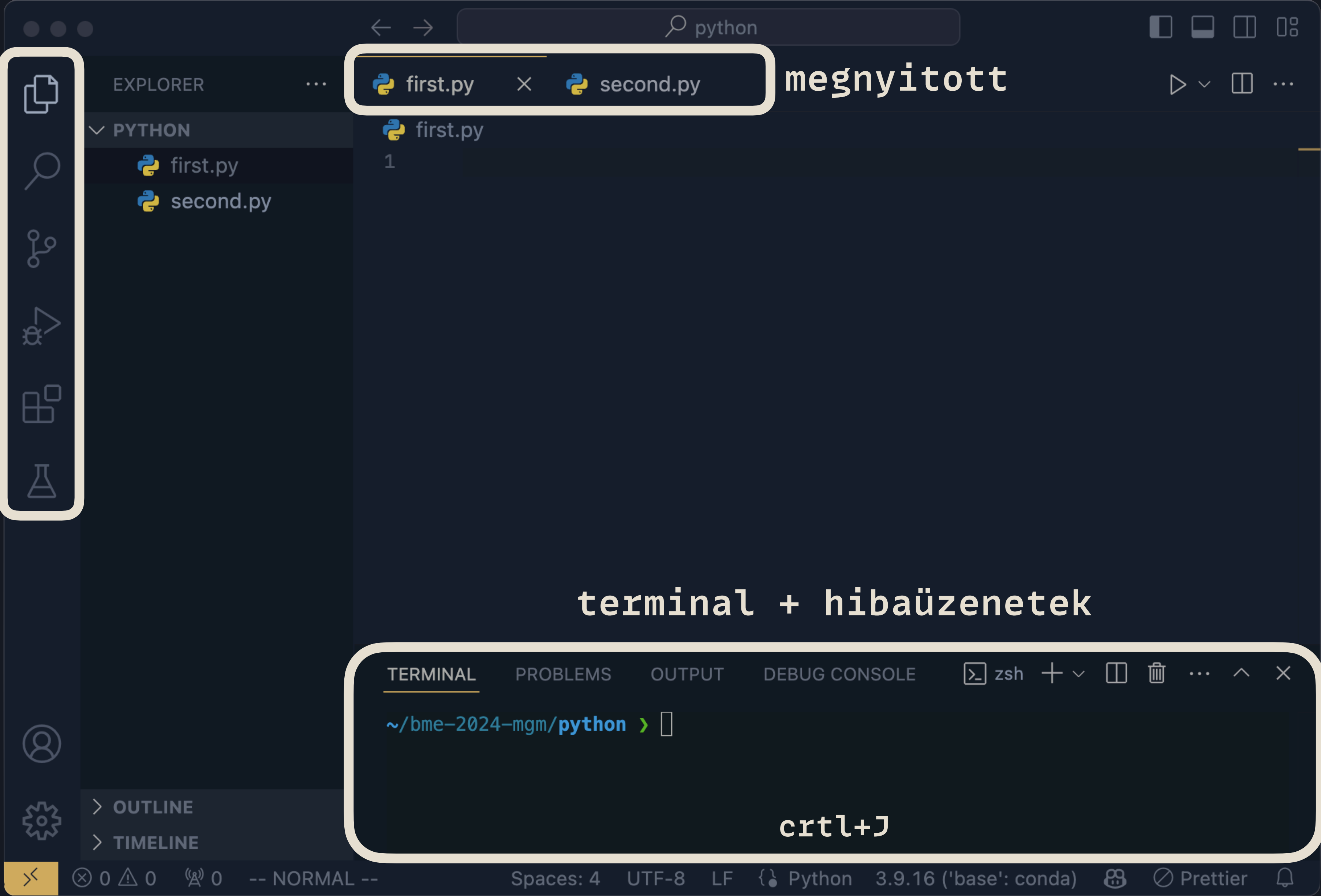
fájlok (workspace)

keresés

git

debug

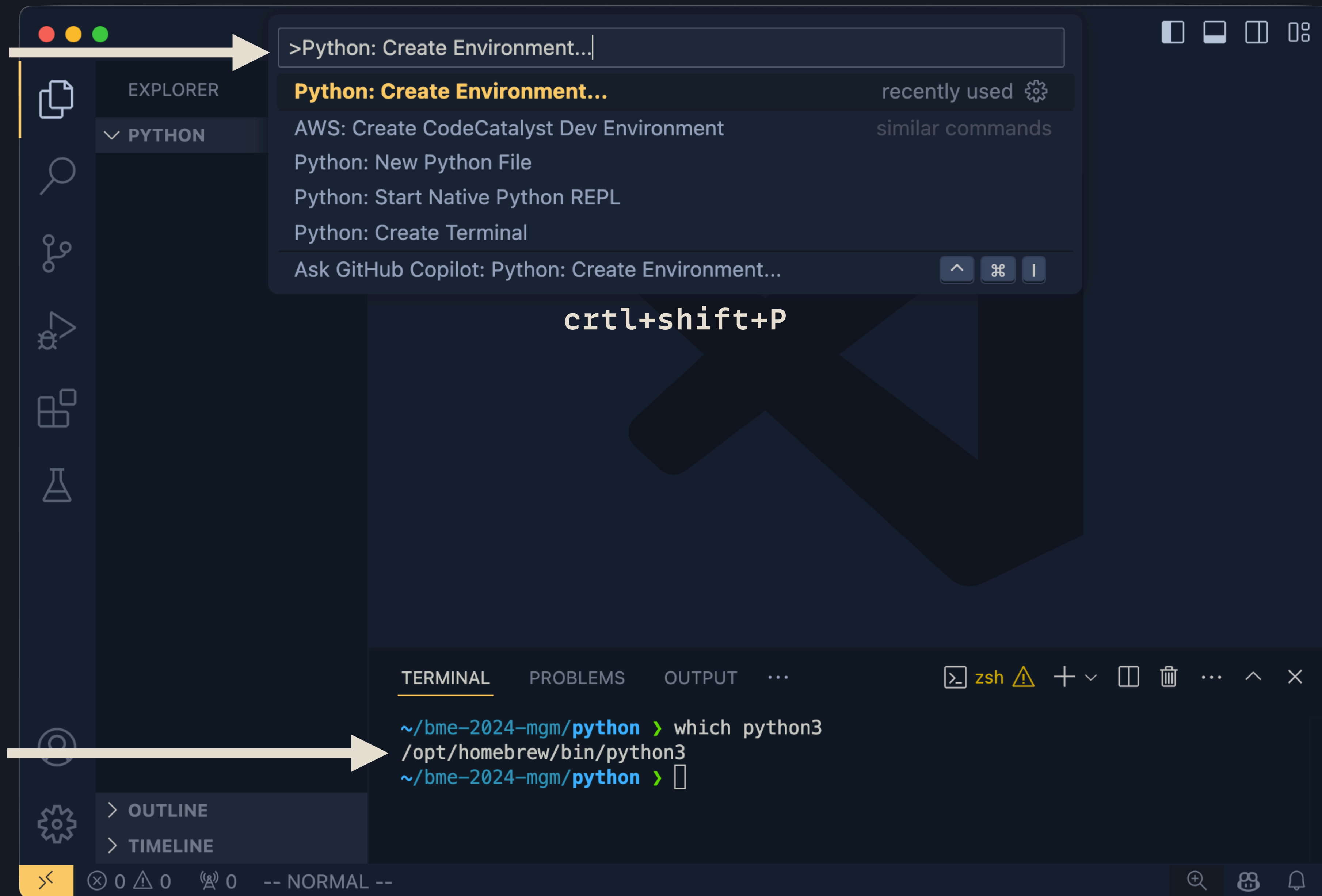
kiegészítők



Python virtual environment

2. csináljunk
egy lokált

1. ez a
globál Python

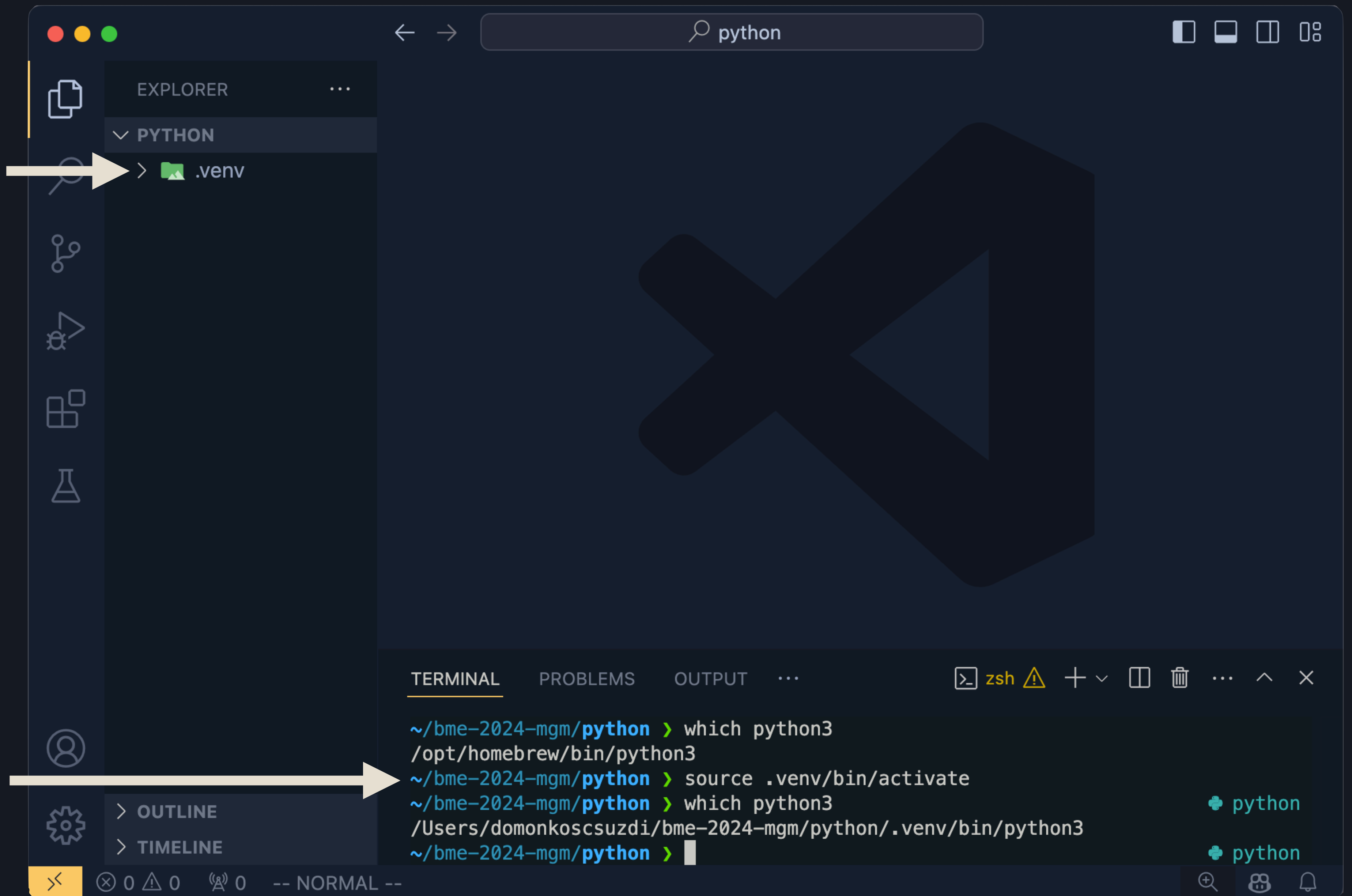


Python virtual environment

Igazából csak
egy mappa

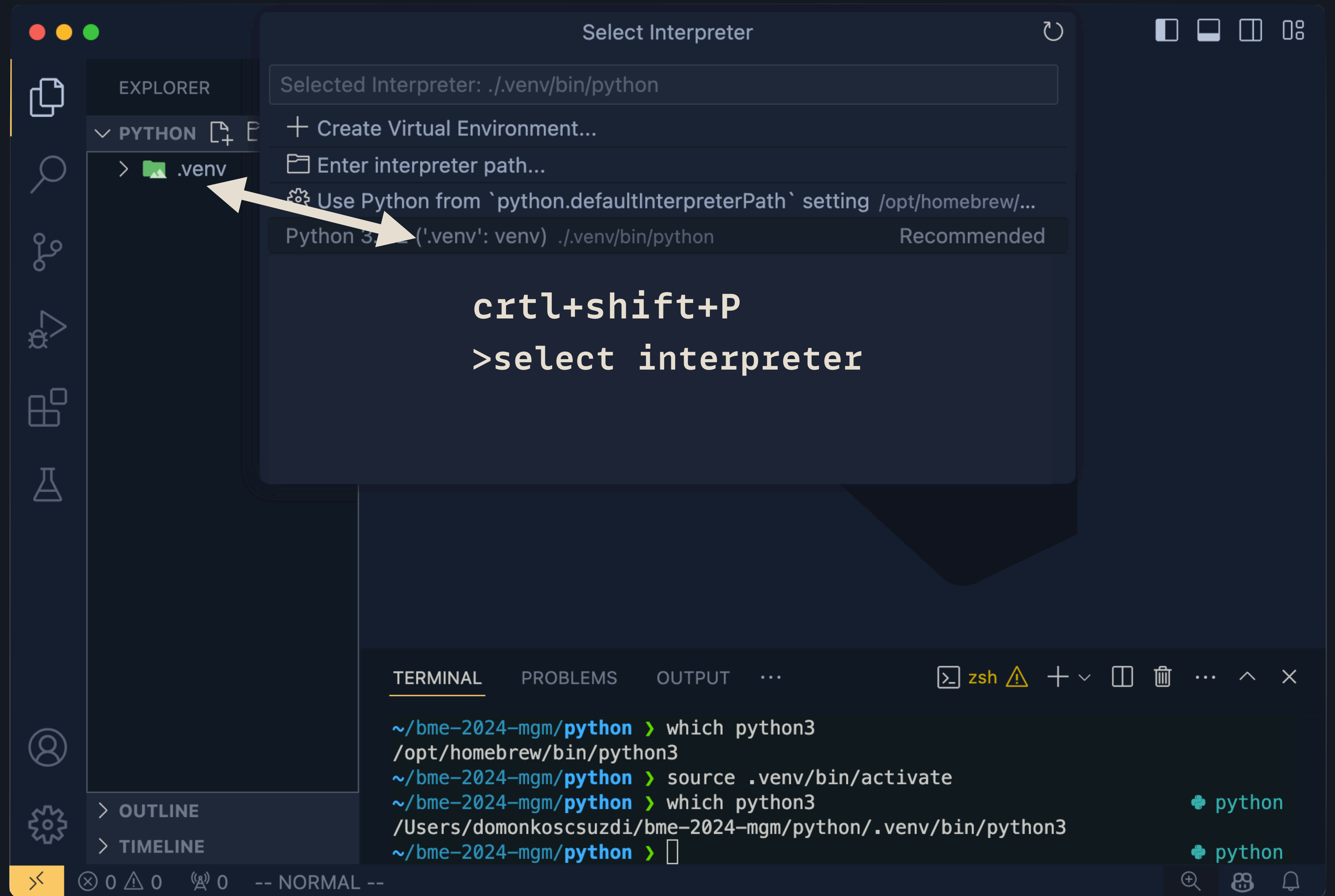
Itt lesznek a
feltelepített
csomagjaink

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 Library¹

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

Külső csomagok

The Python Package Index²
(PyPi)

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

#TODO

- anaconda

Saját csomagok

- átláthatóság
- modularitás
- feltölthetjük a PyPi-re³
- 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)
```



teljes namespace-t

Beépített csomagok



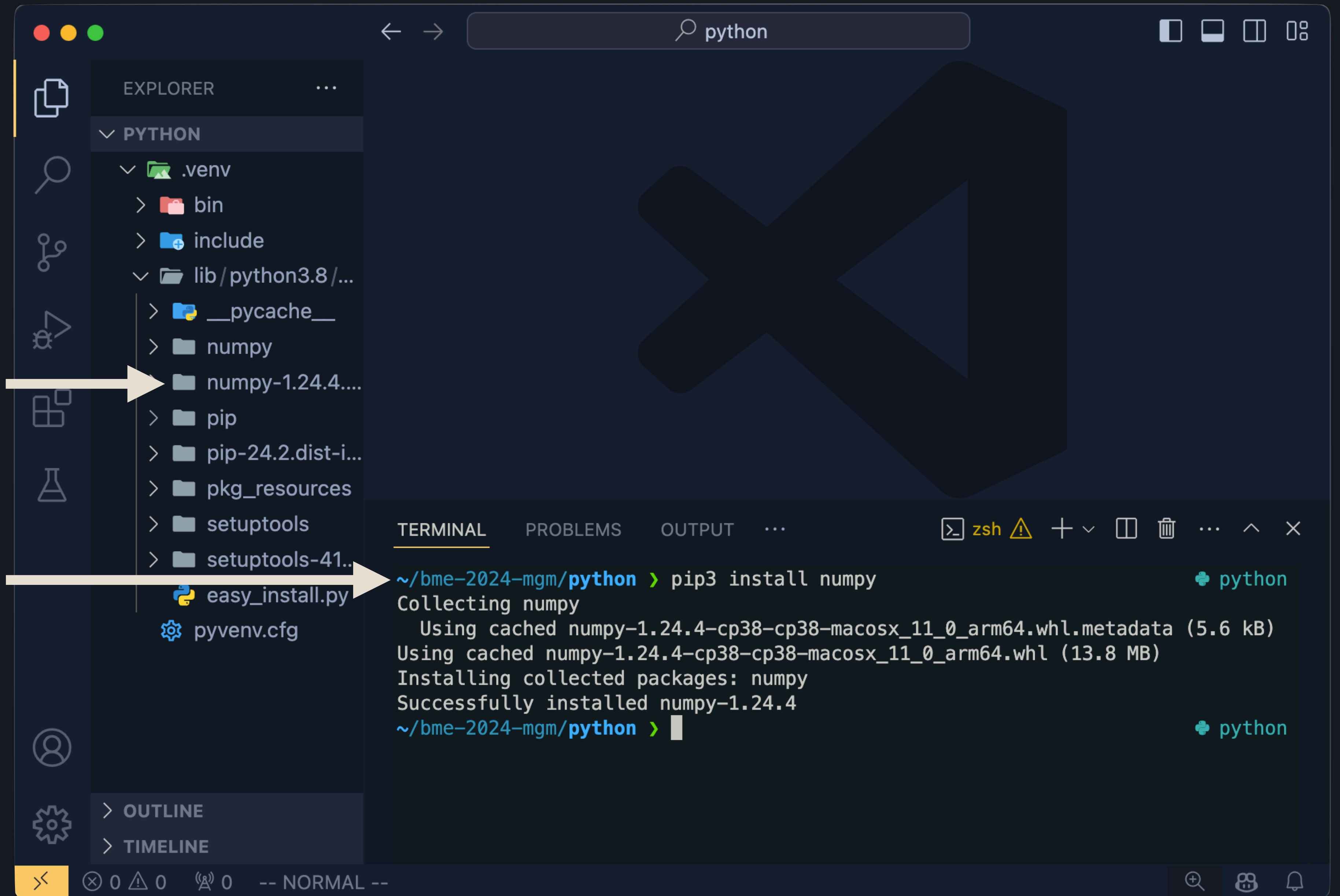
terminal

```
~/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

2. itt is van

1. telepítjük



Külső csomagok



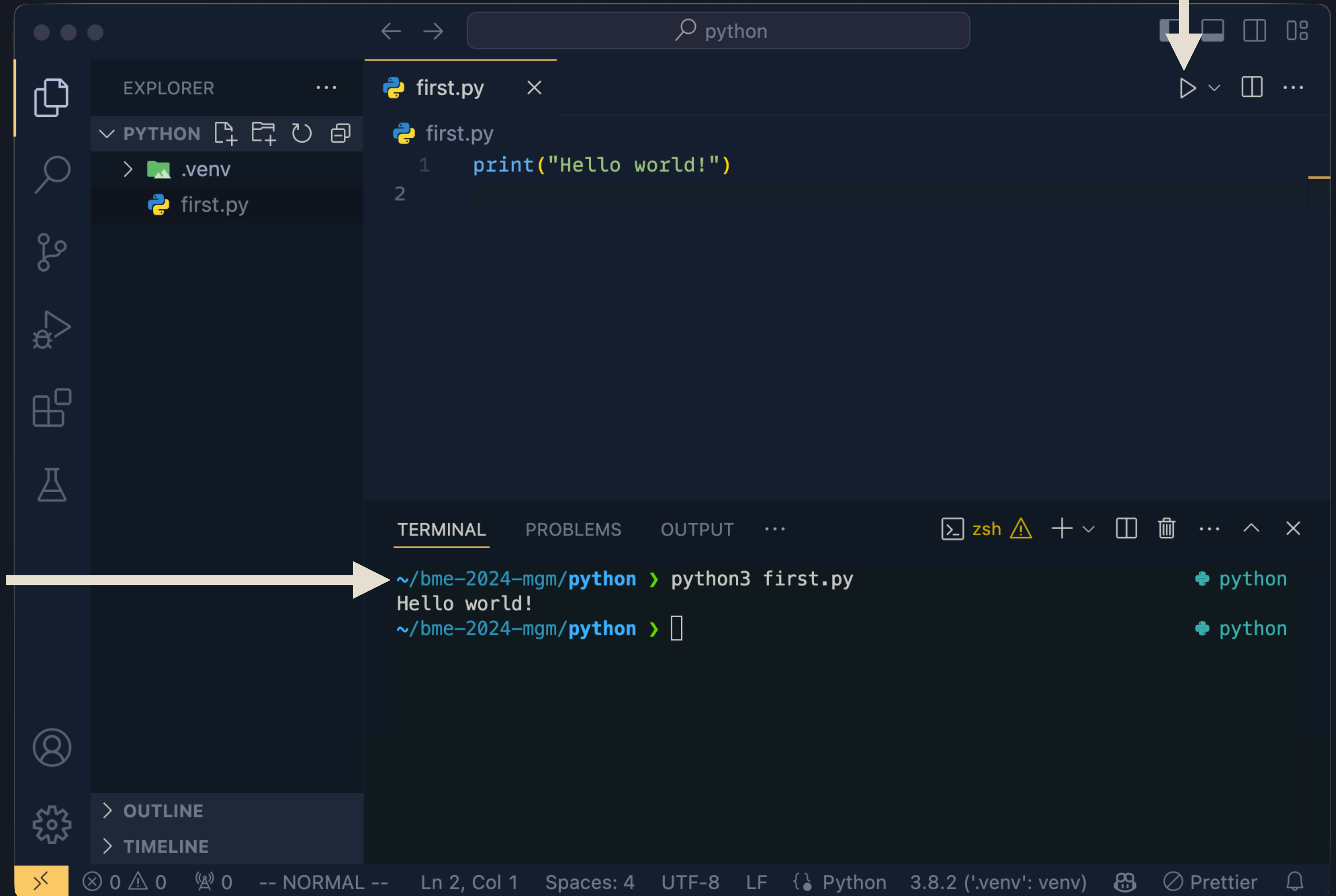
terminal

```
~/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



Python debug

folytat léptet

Run and Debug

vagy

F5

- #TODO
- watch
 - conditional breakpoint

first.py

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

VARIABLES

- Locals
 - special variables
 - math = <module 'ma...>
 - r = 3
- Globals

WATCH

CALL STACK Paused o...

<module>	first.py
----------	----------

DEBUG CONSOLE

Filter (e.g. text, !exclude, \escape)

```
→ 2*r
6
```

Spaces: 4 UTF-8 LF Python 3.8.2 ('.venv': venv) Prettier

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`

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]
```

Mátrixok, vektorok, transzponált

```
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()

Adatstruktúra

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

unchangeable¹, unordered, no duplicates¹

Miért használjuk őket?

$\mathcal{O}(1)$ vs. $\mathcal{O}(n)$

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

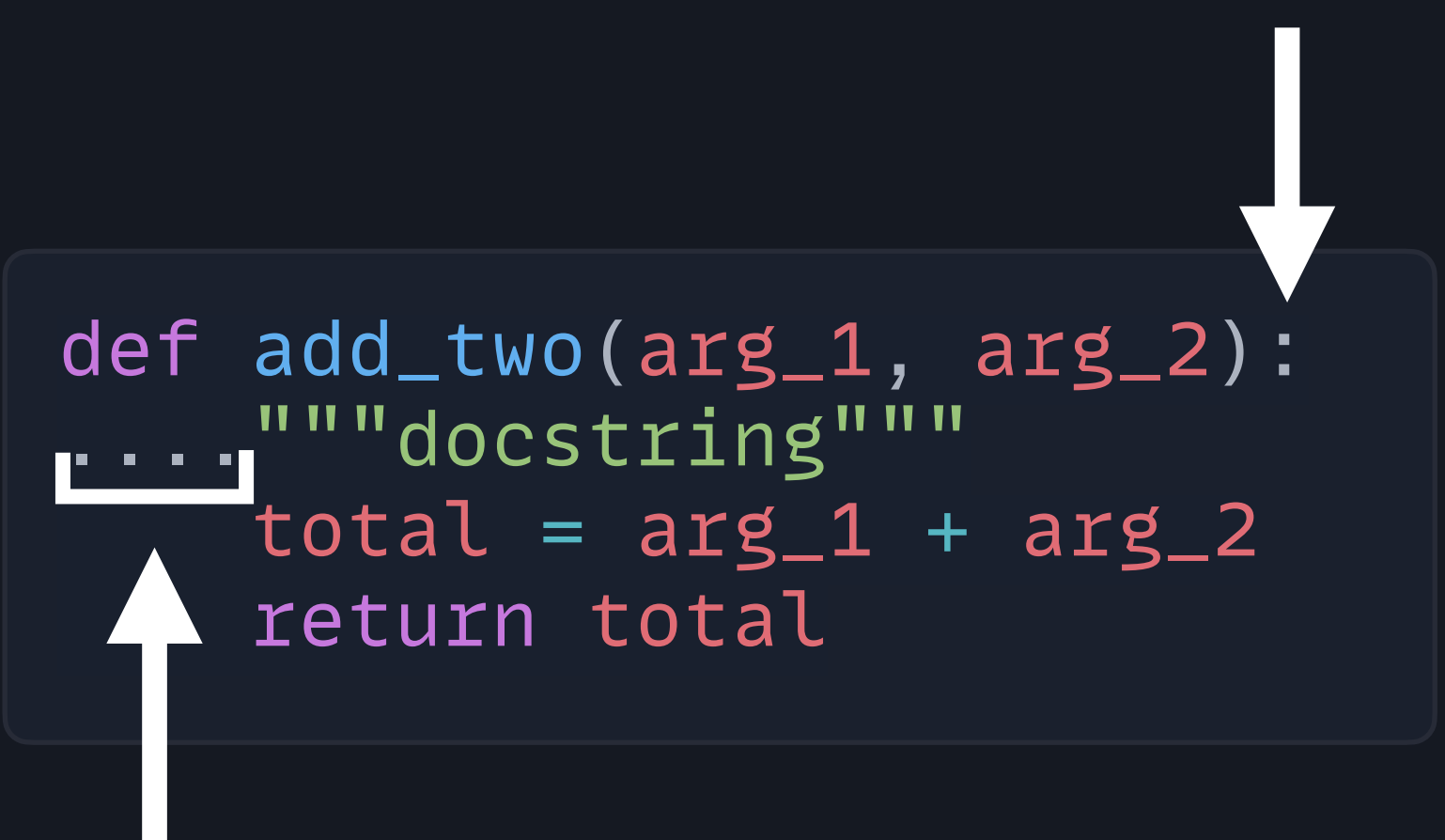
```
import time
```

```
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



A diagram illustrating the evolution of a Python function. It shows a simple function `def add_two(arg_1, arg_2):` with a docstring and a return statement. An arrow points from this function to a more complex version, `def add_nums(arg_1: int, *args: int, **kwargs) → int:`, which includes type hints, variable arguments, keyword arguments, and a return type annotation. Another arrow points from the title 'Függvények' to the first function.

```
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
- no return?

```
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)]
```

 The Pythonic Way® 

```
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"
```