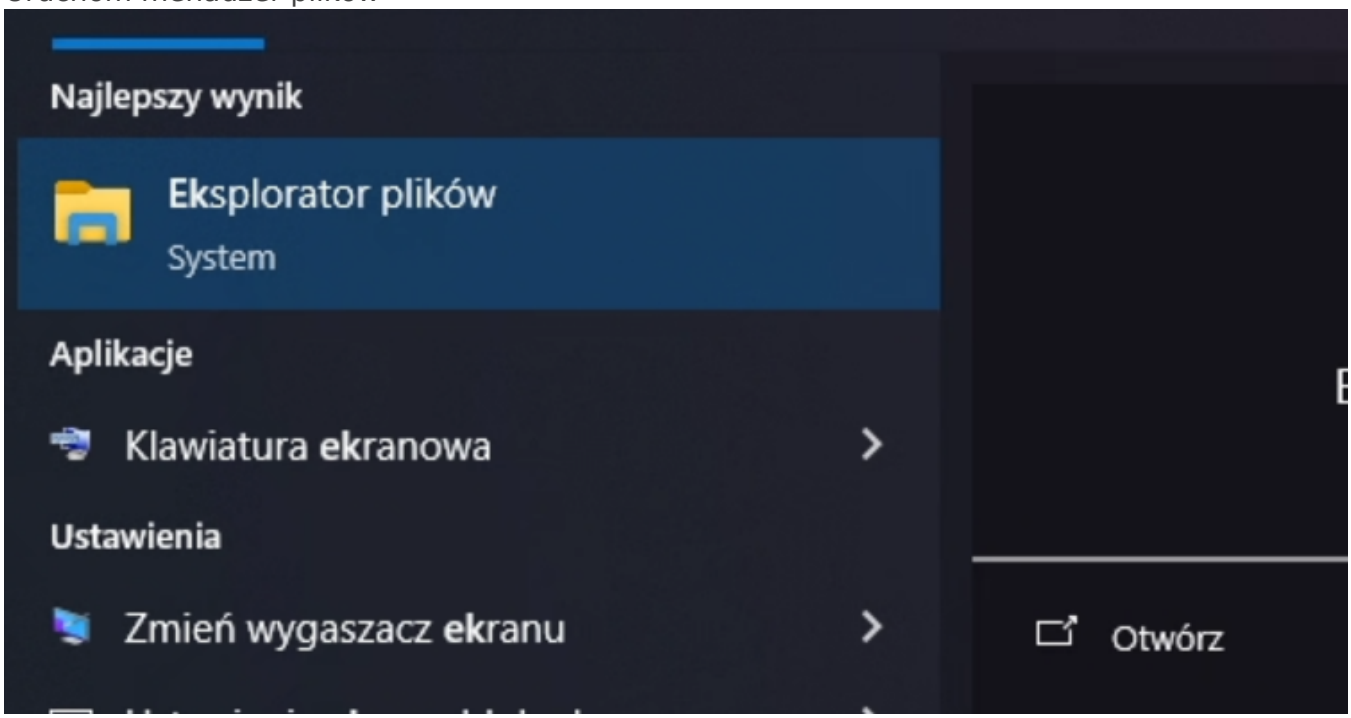
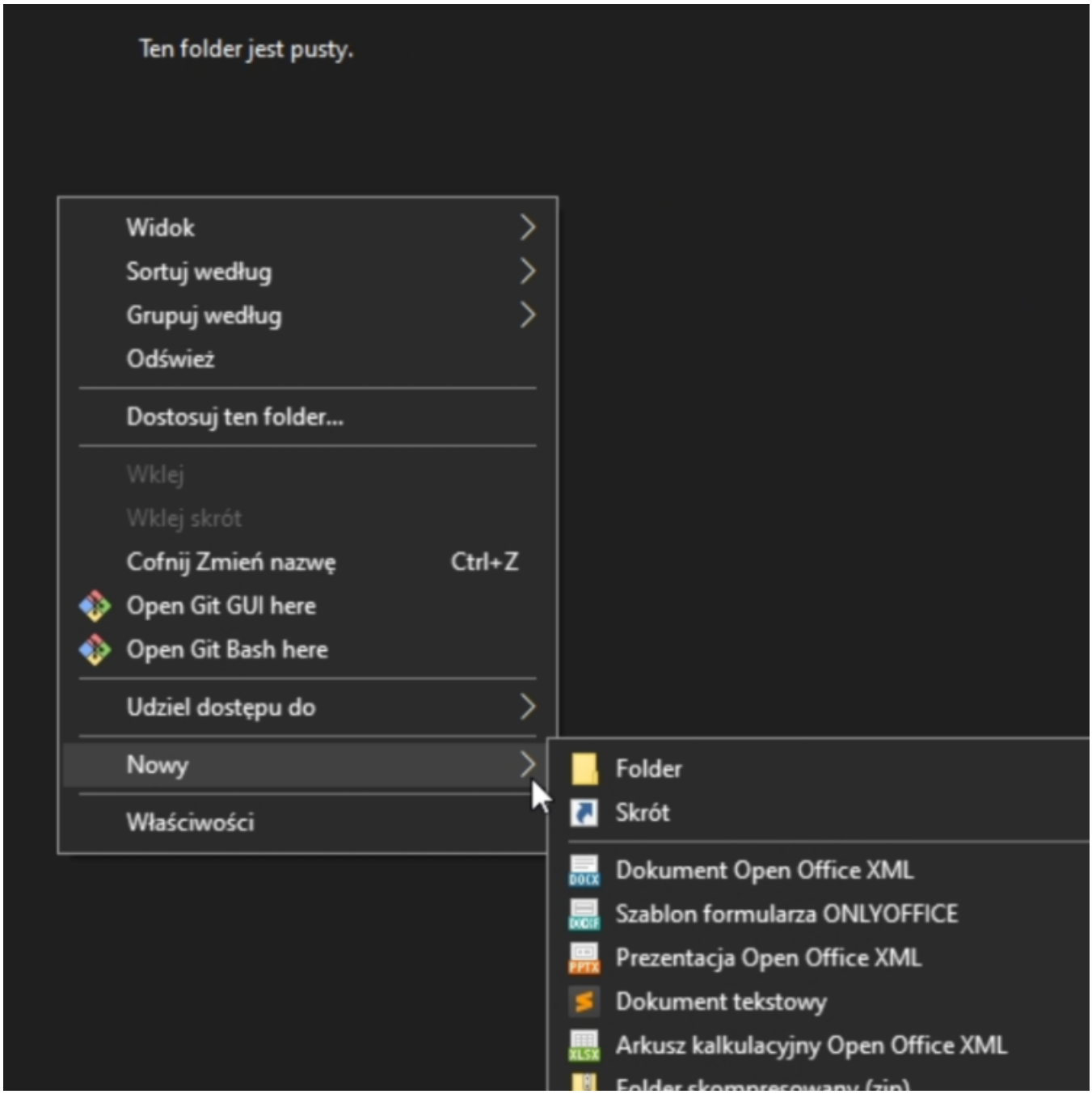
Pliki

Tworzenie nowego folderu pod projekt

1. Uruchom menadżer plików

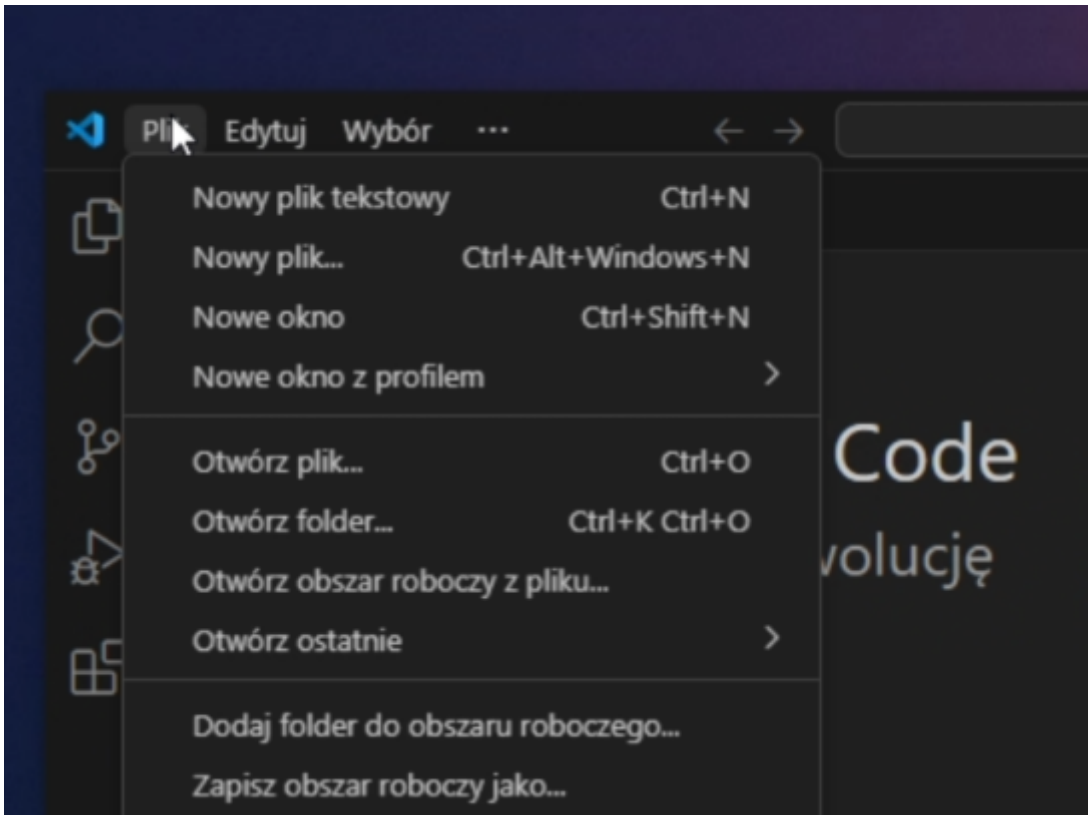


2. Przejdź do miejsca w którym chcesz utworzyć nowy folder do pracy, kliknij prawym i wybierz Nowy -> Folder

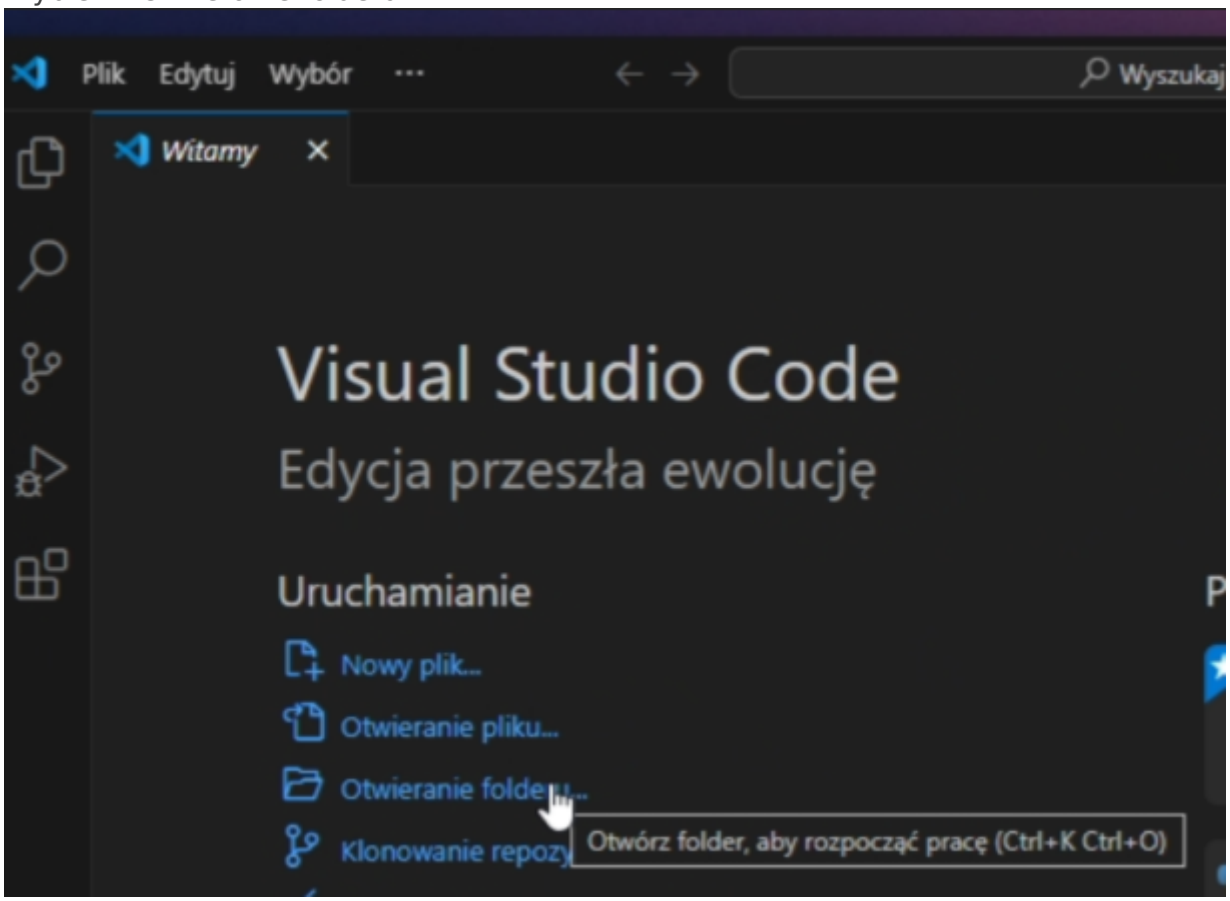


Na Windowsie 11 skorzystaj menadżer plików wygląda nieco inaczej - ma jednak identyczną opcję. Możesz kliknąć w lewym górnym rogu "Nowy" i wybrać tam folder. Nazywając foldery sugeruję unikać polskich liter, znaków specjalnych (w tym spacji).

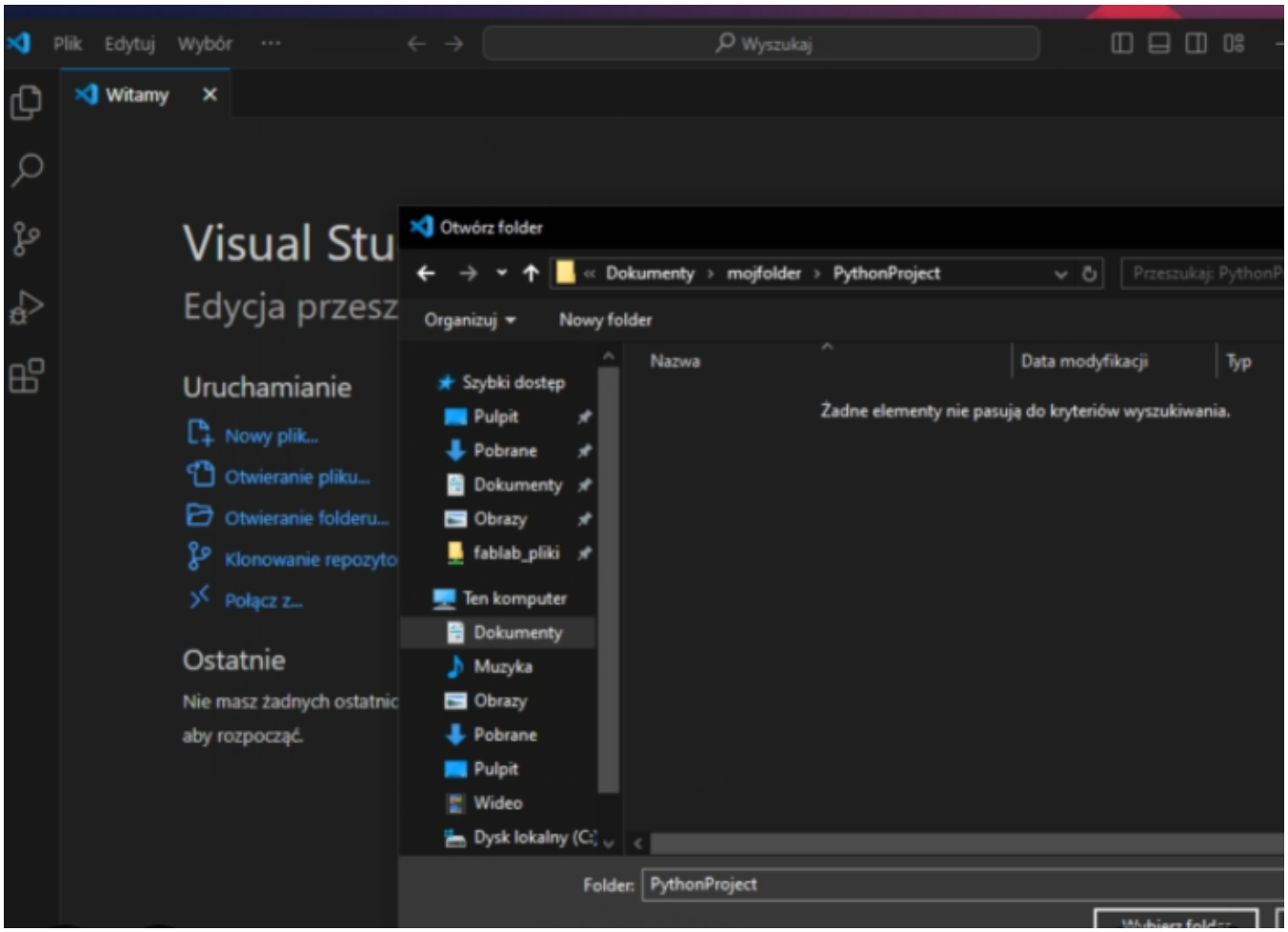
3. Uruchom nowe okno VSCode (Plik -> Nowe okno)



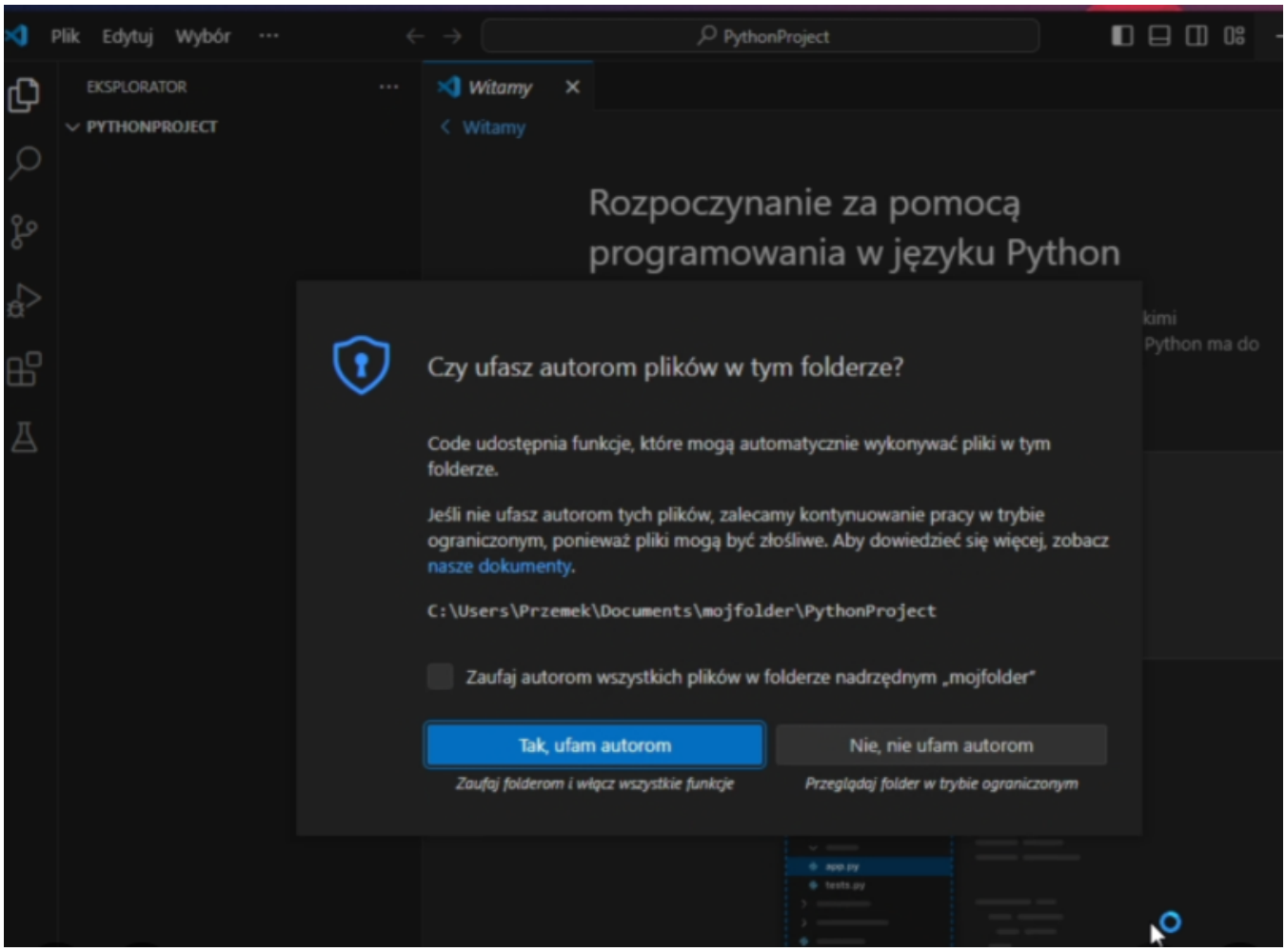
4. Wybierz "Otwieranie folderu"



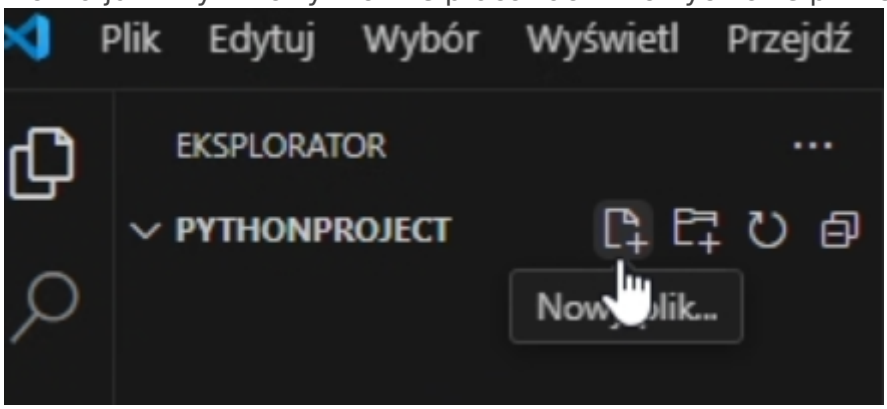
5. Odnajdź stworzony wcześniej folder i kliknij "Wybierz folder"



6. VSCode może zapytać czy ufasz autorom - wybierz "Tak"



7. Można już w tym nowym oknie pracować - tworzyć nowe pliki etc.



Obs?uga plików w Pythonie

- służy do tego słowo `open`

Sugerowane jest też dodanie słowa `with` tak aby plik był automatycznie domykany na koniec pracy (korzystamy w ten sposób z tzw. context managera).

Odczytywanie

```
with open("mojplik.txt") as moj_plik:
    print(moj_plik.read())
```

Domyślnie Python otwiera pliki w trybie do odczytu ("r") - podczas pracy w nim można jedynie odczytać zawartość pliku, nie można nic dodać.

W powyższym przykładzie na wczytanym obiekcie jest użyta metoda `.read()` umożliwiająca odczytanie całej zawartości jako ciąg znaków (string).

Zapisywanie

Chcąc zapisać coś w pliku należy otworzyć go w trybie "w":

```
with open("mojplik.txt", "w") as moj_plik:
    moj_plik.write("Nowa zawartość")
```

Parametr "w" jest dodawany do funkcji open, w nawiasach okrągłych, obok nazwy pliku, który chcemy otworzyć.

W powyższym przykładzie tekst "Nowa zawartość" zostanie zapisana do pliku `mojplik.txt`.

WAŻNE!!! - w tym trybie istniejąca zawartość pliku zostanie nadpisana(!).

Jeśli chcemy dopisać coś do końca pliku trzeba użyć trybu "a" (od append):

```
with open("mojplik.txt", "a") as moj_plik:
    moj_plik.write("Nowa zawartość")
```

Tryb binarny i tryb tekstowy

Domyślnie Python traktuje otwarte pliki jako tekst. Czasami zachodzi jednak potrzeba otwarcia ich w trybie binarnym (np. przy większości formatów obrazków) - wystarczy dodać literkę "b" do trybu.

Pliki tekstowe to np. pliki .txt, .csv, .json, .xml, .py ... i wiele innych.

Binarne to większość obrazków (.jpg, .png...), pliki wykonywalne aplikacji (.exe)...

W tym kontekście mówimy o plikach tekstowych jako o plikach, które są w stanie odczytać ludzie (otwierając je np. w Notatniku czy innym edytorze tekstu).

Tabelka trybów

Poza trybem `r` służącym do odczytu, `w` do napisania i `a` do dopisania Python oferuje nam różne kombiancje trybów.

tryb	opis
`r`	odczyt
`r+`	odczyt i zapis, błąd jeśli plik nie istnieje
`rb`	odczyt w trybie binarnym
`rb+`	odczyt i zapis w trybie binarnym, błąd gdy plik nie istnieje
`w`	zapis, nadpisuje zawartość, tworzy plik gdy nie istnieje
`w+`	odczyt i zapis, nadpisuje zawartość, tworzy plik gdy nie istnieje
`wb`	zapis w trybie binarnym, nadpisuje zawartość, tworzy plik gdy nie istnieje
`wb+`	odczyt zapis w trybie binarnym, nadpisuje zawartość, tworzy plik gdy nie istnieje
`a`	dopisanie zawartości, tworzy plik gdy nie istnieje
`a+`	odczyt i dopisanie zawartości, tworzy plik gdy nie istnieje
`ab`	dopisanie w trybie binarnym, tworzy plik gdy nie istnieje
`ab+`	odczyt i dopisanie w trybie binarnym, tworzy plik gdy nie istnieje

Na początku dobrze jest pozostać przy podstawowych trybach - bo początkowo np. `a+` i tym podobne mogą zaskakiwać (np. w `a+` jak domyślnie odczytamy zawartość to nic nie dostaniemy - co wynika z tego, że w tym trybie znacznik położenie w pliku ustawiany jest na końcu zawartości).

Sprawdzenie czy plik istnieje

```
os.path.exists("sciezka_do_pliku")
```

Pliki JSON

- to jeden z najpopularniejszych formatów do przechowywania i wymiany danych
- budową przypomina nieco pythonowy słownik (ALE nie jest identyczny!!!)
- do obsługi plików w tym formacie wykorzystamy bibliotekę json

```
import json
```

Wczytanie JSON-a

- służy do tego funkcja `load` z modułu `json`

```
import json

with open("plik_z_danymi.json") as plik_json:
    zawartosc = json.load(plik_json)
```

- wczyta to nam plik json, a następnie jego zawartość zamieni na odpowiedni obiekt pythonowy (tzw. deserializacja danych)

Zapis JSON-a

- służy do tego funkcja `dump` z modułu `json`
- oczywiście chcąc zapisać należy użyć trybu `w`

```
import json

dane = {"jabłka" : 4, "mango" : 3}

with open("plik_z_danymi.json", "w") as plik_json:
    json.dump(dane, plik_json)
```

- zamieni to nam obiekt pythonowy `dane` na poprawnie skonstruowanego json-a i zapisze do pliku

Chcąc mieć ładniej sformatowany plik JSON można dodać jeszcze parametr `indent` do `dump` np.

```
import json

dane = {"jabłka" : 4, "mango" : 3}

with open("plik_z_danymi.json", "w") as plik_json:
    json.dump(dane, plik_json, indent=4)
```

Kodowanie znaków

Domyślnie biblioteka json przy zapisie podmieni nam wszystkie "dziwne" znaki (np. polskie litery) tak aby całość dało się zapisać jako ascii.

Chcąc wymusić kodowanie utf8 należy:

- przy wczytywaniu

```
import json

with open("plik_z_danymi.json", encoding="utf8") as plik_json:
    zawartosc = json.load(plik_json)
```

- przy zapisywaniu

```
import json

dane = {"jabłka" : 4, "mango" : 3}

with open("plik_z_danymi.json", "w", encoding="utf8") as plik_json:
    json.dump(
        dane,
        plik_json,
        indent=4,
        sort_keys=True,
        ensure_ascii=False,
    )
```

CSV

- do obsługi plików csv służy biblioteka ... csv :D

```
import csv

with open("currency.csv", "r") as plik:
    reader = csv.reader(plik)

    print(reader)
```

```
    for wiersz in reader:
        print(wiersz)
```

Pandas

- kombajn do wszelkich danych statystycznych etc, w tym do obsługi plików wykorzystywanych w tej dziedzinie
- pandas natywnie operuje na tzw. ramkach danych (dataframes) i na nich wykonuje wszelkie operacje
- możemy użyć jej do deserializacji danych z plików csv, xlsx (po dodaniu biblioteki pyxml) i wielu innych

https://pandas.pydata.org/

Np. otwarcie pliku csv z pomocą Pandas:

```
import pandas as pd


currency_dataframe = pd.read_csv("currency.csv")
print(currency_dataframe)
```

Przydatne funkcje do wczytywania plików z pomocą Pandas:

- read_excel (wymaga openpyxl)
- read_csv
- read_xml
- read_json

A na koniec krótkie przypomnienie:

Importowanie bibliotek

Aby wykorzystać jakąś bibliotekę, którą mamy zainstalowaną (pobraną na nasz komputer) należy użyć słowa import w kodzie pythonowym (najlepiej na początku):

```
import nazwa_biblioteki
```

Jeśli nazwa jest za długa możemy ją podmienić np.

```
import nazwa_biblioteki as nb
```

Instalacja bibliotek

Jeśli nie mamy biblioteki w systemie możemy ją pobrać np. wykorzystując narzędzie `pip` (w terminalu):

```
pip3 install nazwa_biblioteki
```

Wersja #9
Utworzono 2024-09-23 14:29:39 UTC przez Przemek Jeske
Zaktualizowano 2025-09-16 13:10:54 UTC przez Przemek Jeske