# Podstawy pythona

## 2. Czym jest Python? Pierwsze uruchomienie

Python - jeden z języków do projektowania, który pozwala komunikować się z naszym komputerem.

IDLE - zintegrowane środowisko do programowania.

## 3. Zmienne, pierwszy skrypt oraz zmiana stanu powłoki

**Zmienne** - pozwalają zapisać wartości do siebie, można je wykorzystać/zmienić za pomocą etykiet (ich nazw). W przenośni są to pojemniki do przechowywania wartości.

Do stworzenia zmiennej potrzebujemy etykietę (nazwę), znak równości oraz wartość jaką chcemy jej przypisać np.

wiek = 28

aby wydrukować wartość zmiennej piszemy:

print(wiek)

**Shell** - powłoka interaktywna mająca za zadanie interakcje między Tobą, a interpreterem, który analizuje i wykonuje instrukcje. Pozwala na interakcję ze skryptem (kodem) napisanym i wywołanym z zewnątrz np. przez Ciebie.

Zmiana stanu powłoki zachodzi w momencie, gdy uruchomisz skrypt lub gdy np. stworzysz zmienną z pomocą interpretera. Zmiana ta istnieje dopóki nie zrestartujemy powłoki. Tzn. “stan” powłoki wraz z jej zawartością jest cały czas niezmienny. Co oznacza, że raz stworzone zmienne istnieją dopóki nie zamkniemy całego IDLE lub nie zrestartujemy powłoki np. uruchamiając ponownie nowy skrypt.

**Skrypt** - kod który zapisujemy, można go wywołać. Komputer wywołuje skrypt z komendami (instrukcjami) od góry do dołu.

## 4. Komentarze

**Komentarz** - notatka, która sprawia, że kod jest czytelniejszy. Komentarze nie są interpretowane przez komputer.

Sposób tworzenia komentarzy :

* **jednolinijkowy** (za pomocą # (hasha - SHIFT + 3) np.

#dobrzeIdzie

edytor zaznacza komentarz na kolor czerwony co oznacza, że nie zostanie interpretowany

* **wielolinijkowy** ( za pomocą potrójnego ””” ) na początku i na końcu komentarza np.

”””

to jest nieinterpretowane

i to też

i to również!

ile można!

”””

## 5. Typy zmiennych oraz nazewnictwo

Zapis zmiennych w Pythonie:

* liczb całkowitych

a = 6 #l. całkowita z angielskiego int (integer)

* liczby z ułamkiem zapisujemy z kropką

b = 5.3 #l. zmiennoprzecinkowa (float)

* ciąg znaków tzw. string np. imię, nazwisko, tekst zapisujemy za pomocą podwójnego lub pojedynczego cudzysłowia (apostrofu)

imie = "Arkadiusz" lub imie = 'Arkadiusz'

Należy pamiętać, że przy pisaniu zdania np. w języku angielskim używając apostrofa np.

zdanie = 'I'm from Poland'

musisz zastosować znak” \” (slash):

zdanie = ('I\'m from Poland'),

który mówi komputerowi, że kolejny znak jest w tym wypadku apostrofem jako znak, a nie jako zakończenie stringa.

Sprawdzenie typu zmiennych - za pomocą słowa type np.:

type(a)

type (zdanie)

type (imie)

**Nazewnictwo zmiennych jest bardzo ważne :**

* Zmienne powinny być w programowaniu samoopisujące się np.

imie = "Arkadiusz", nazwisko = "Kowalski" , wiek = 34

przez samoopisujące się rozumiemy, że czytając nazwę zmiennej wiemy co ona przechowuje

* Nie korzystamy z polskich znaków, przy pisaniu kodów
* Nie zaczynamy nazewnictwa zmiennej od liczb
* Dla czytelności warto powiększać pierwsze litery kolejnych słów w nazwie zmiennej np.: nazwaUzytkownika

# Funkcje – zawansowane aspekty

## 52. Aplikacja wielomodułowa | Jak zaimportować Twój własny moduł?

**Biblioteka** - to np. złożenie kilku modułów, czyli zestaw narzędzi (funkcji)

**Moduł** - tj. plik z kodem, to element do wykorzystania jako część czegoś większego np. biblioteki.

Moduły dzielimy na trzy typy :

* określone przez twórców Pythona zwane standardową biblioteką

(taki, z którego możemy zaimportować pliki i korzystać z nich w każdej chwili)

* z zewnętrznych źródeł (możemy je zainstalować)
* stworzone przez nas

**Przykład modułu  ''figury ''**

      import math

      def pole\_kwadratu(a):

          return a \* a

      def pole\_prostokata(a, b):

          return a \* b

      def pole\_kola(r):

          return math.pi \* r \*\* 2

      def pole\_trojkata(a, h):

          return 0.5 \* a \* h

      def pole\_trapezu(a, b, h):

          return (a + b) / 2 \* h

Aby zaimportować moduł  ''figury '' w innym pliku:

* tworzymy nowy plik, zapisujemy i nadajemy mu nazwę np. ''obliczenia''

(plik ''obliczenia'', który chcemy zaimportować musi znajdować się w tym samym folderze co plik ''figury'')

* dzięki umieszczenia obu plików w jednym folderze wystarczy w pliku ''obliczenia'' wypisać

import figury

            po wywołaniu tej instrukcji jesteśmy w stanie korzystać z każdej zapisanej funkcji np.

figury.pole\_kwadratu(4)

**Ćwiczenie**

Zapytaj użytkownika jaką figurę chce wybrać i oblicz jej pole.

**import figury**

wybor = input("""Wybierz figurę, której pole chcesz obliczyć:

1. Kwadrat

2. Prostokąt

3. Koło

4. Trójkąt

5. Trapez

""")

if (wybor == '1'):

    a = float(input("Podaj bok kwadratu: "))

    print("Pole kwadratu wynosi:", figury.pole\_kwadratu(a))

"""  - za pomocą potrójnego cudzysłowia w instrukcji **print** oraz **input** jesteśmy w stanie zastosować enter, jeżeli znajdują się one wewnątrz funkcji, nie będą traktowane jako komentarz.

**' '** - nie zapominajmy o zapisaniu wartości w pojedynczym apostrofie, ponieważ input zwraca nam ciąg znaków

## 53. Enum - co to jest i po co to używać? Ulepszamy program z poprzedniej lekcji

**enumeration - spis, wyliczenie**

wybor = input("""Wybierz figurę, której pole chcesz obliczyć:

1. Kwadrat

2. Prostokąt

3. Koło

4. Trójkąt

5. Trapez

""")

**Tworzenie spisu enumeration :**

* importujemy z biblioteki enum Enum

from enum import Enum

* korzystając z enum określamy **nazwę naszego spisu**

Enum('Menu\_Figury' , 'Kwadrat Prostokąt Koło Trójkąt Trapez')

tak powstał nasz nowy spis, który ma przypisane do poszczególnych wartości swoje

nazwy

* tworzymy zmienną aby zapisać wartość i móc się do niej odwołać

Menu\_Figury = Enum('Menu\_Figury' , 'Kwadrat Prostokąt Koło Trójkąt Trapez')

Korzystając z enumeracji używamy dużych liter, dzięki temu widzimy że korzystamy

z wartości przypisanych do spisu

* dopisujemy **int przed Enum:**

import IntEnum

Menu\_Figury = IntEnum('Menu\_Figury' , 'Kwadrat Prostokąt Koło Trójkąt Trapez')

wybor = int(input("""Wybierz figurę, której pole chcesz obliczyć:

1. Kwadrat

2. Prostokąt

3. Koło

4. Trójkąt

5. Trapez

"""))

Z POMOCĄ Menu\_Figury I NAZWY MOŻEMY ODWOŁAĆ SIĘ DO POSZCZEGÓLNYCH WARTOŚCI enum

if (wybor == Menu\_Figury.Kwadrat):

a = float(input("Podaj bok kwadratu: "))

print("Pole kwadratu wynosi:", figury.pole\_kwadratu(a))

## 68. ĆW: Ponumeruj zadania i pokaż je użytkownikowi | enumerate()

enumerate(list)

zwraca **iterator**, którego można użyć do wyodrębnienia krotki z (**indeksem**, **elementem**) każdego **elementu**na **liście**.

**Iterator**może być iterowany (zapętlany). **Iterator**, gdy się do niego odwołamy w np. pętli 'for' zwraca dane.

**enumerate**to sposób na śledzenie indeksu każdego elementu na liście. Pozwala na liczenie elementów jeden po drugim na liście.

W przypadku **enumerate**(), **iterator**zwraca **krotki**zawierające indeks i odpowiadający mu element listy iterowalnej, który został przekazany jako pierwszy argument do enumerate().

Przykładowy sposób użycia modułu wyliczającego:

#tworzymy liste

colors = ["red", "green", "blue"]

# wybieramy elementy z krotki i je wyświetlamy:

for i, color in enumerate(colors):

print(i, color)

Wylicz (ponumeruj) każde zadanie z listy i pokaż je użytkownikowi.

Input:

tasks = ["clean the kitchen", "do laundry", "pay bills"]

Output:

1 Clean the kitchen

2 Do laundry

3 Pay bills

Wykonaj poniższe kroki, aby ukończyć ćwiczenie:

1. Utwórz listę o nazwie „zadania”, która zawiera kilka elementów łańcuchowych, z których każdy reprezentuje zadanie do wykonania.

2. Użyj pętli for i funkcji `enumerate`, aby przeglądać listę zadań. Dla każdej iteracji wydrukuj indeks i zadanie.

3. Wyszukaj w Internecie, jak zmienić punkt początkowy funkcji wyliczającej. Zmodyfikuj pętlę for, aby używała indeksu początkowego 1 zamiast 0 dla wyliczenia.

4. Ponownie zmodyfikuj pętlę, aby pierwsza litera każdego zadania na liście była pisana wielką literą i wydrukuj je.

Rozwiązanie poniżej:

tasks = ["clean the kitchen", "do laundry", "pay bills"]

for i, task in enumerate(tasks, start=1):

task = task.capitalize()

print(i, task)

# Losowanie – zdarzenia losowe

## 69. Losowanie liczb - czy broń trafi przeciwnika?

**random** - (z ang. losowy). moduł po zaimportowaniu dostarcza nam kilka funkcji:

* **random** - losuje liczby od 0 do 1, która jest liczbą zmiennoprzecinkową zaczynając 0 nie licząc 1:

random 0 <= x < 1 lub [0,1)

import random

print(random.random())

* **uniform** - w tej funkcji jako pierwszy argument przesyłamy początek losowania, następnym argumentem jest koniec losowanie np. licząc od 2.5 bez uwzględnienia 10.

uniform (2.5, 10.0) 2.5<= x <10.0 lub [2.5,10)

przykład wykorzystania funkcji uniform w pętli od 0 do 100

x = 0

while x < 100:

x = x + 1

print(random.uniform(0,100))

* **randrange** - wylosuje liczby całkowite podane w zakresie puli oraz wybierze jedną z tych z wartości.

randrange(10) z puli (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

* **randrange** (0, 15, 2) parzyste z puli - podajemy liczby w zakresie od 0 do 14, następnie wypisujemy “step” - wielokrotność liczby, które mają być tworzone () przez co otrzymujemy (0,2,4,6,8,10,12,14) i losujemy jedną z nich.
* **randint** - przyjmuje pełną podaną wartość np. od 0 do 4 włącznie np.

randint(0,4) lub [0,4]

## 70. choice i choices - losowy element z listy - jak ustalić prawdopodobieństwo?

**W module random znajdziemy funkcję :**

* **Choice** - używamy tej funkcji gdy chcemy wylosować losowo jeden z elementów i mieć równe prawdopodobieństwo losowania.

movieList = ["Tytuł1", "Tytuł2", "Tytuł3", "Tytuł4"]

import random

print(random.choice(movieList)

W tym momencie szansa na wylosowanie każdego z tych elementów wynosi 25%

* **Choices** - zwraca listę elementów i pozwala ustalić prawdopodobieństwo wylosowania jednego z podanych elementów

nagrodaZeSkrzynki = ["zielona", "pomarańczowa", "purpurowa", "legendarna"]

Odwołujemy się więc do opcji random i funkcji choices po czym piszemy skąd chcemy pobrać dane, opcjonalne możemy podać drugi argument, a jako ostatni (k) liczbę losowań.

import random

random.choices(nagrodaZeSkrzynki, k = 100)

W tym kodzie szanse na wystąpienie nagród są równomierne - a więc po ok.25% szans na wylosowanie.

**Aby ustalić dokładnie jakie szanse mają wystąpić dla poszczególnych elementów musimy podać jako drugi argument listę** za pomocą średniej ważonej, ułamków lub liczb całkowitych (określane za pomocą procentów)

print(random.choices(nagrodaZeSkrzynki, [80,15,4,1], k = 100))

**Czyli szansa na wystąpienie :**

Zielony - 80 %

Pomarańcz - 15%

Purpurowy - 4 %

Legendarny - 1 %

## 71. shuffle - tasowanie kart - gra w "wojnę"

**shuffle** - przetasować, zmienić kolejność jakiegoś zbioru elementów

**Przykład listy kart do gry w wojnę**

cardList = [ "9", "9", "9", "9",

"10", "10", "10", "10",

"Jack", "Jack", "Jack", "Jack",

"King", “King", “King", “King”,

"Queen", "Queen", "Queen", "Queen"

"Ace", "Ace", "Ace", "Ace",

"Joker", "Joker"

]

**Aby zmienić kolejność kart w powyższej liście:**

* Importujemy moduł random, który posiada funkcję shuffle

import random

* Odwołujemy się do random i funkcji shuffle **i przesyłamy listę którą chcemy przetasować**

random.shuffle(cardList)

* Należy pamiętać, że funkcja ta działa na oryginale i nie zwróci nam nowej listy. Aby sprawdzić wynik losowania wypiszemy go za pomocą

print(cardList)

## 72.Sample - ĆWICZENIE: Losowanie elementów bez DUPLIKATÓW - piszemy generator 6 z 49 liczb

**sample** - próbka / przykład - z pomocą random funkcja sample wylosuje unikalne,

niepowtarzalne wartości (tylko, gdy unikalne wartości są wewnątrz listy)

import random

def choose\_random\_numbers(amount, total\_amount):

print(random.sample(range(1, total\_amount + 1), amount))

choose\_random\_numbers(6, 49)

↕

**podajemy ile liczb chcemy**

**wylosować, z jakiego zakresu**

# Praca z plikami oraz wyjątki i ich obsługa

## 75. Czym jest plik? Jak go otworzyć? Dlaczego plik trzeba zamknąć? Jak zapisać dane?

**Plik** - nazwana lokacja, która przechowuje na stałe dane na dysku

Podczas tworzenia obiektów np.

a = 5

dane z programu są zapisywane **tymczasowo w pamięci RAM**, czyli pamięci podręcznej komputera.

**Na plikach możemy wykonywać operacje:**

* **otwierania**

Podstawowe sposoby otwierania plików:

* + **r** - R ead (**czytanie**) – domyślne
  + **w** - W rite (**pisanie**) - jeżeli plik istnieje to go usunie, jeżeli nie istnieje to go utworzy
  + **a** - A ppend (**dopisywanie**)

Operacje otwierania wykonuje się przy pomocy funkcji open z ang. otwierać np.

open("test")

W nawiasie podajemy nazwę pliku.

* **pisanie/czytanie**

open("test", "w")

Stworzyliśmy w ten sposób nowy plik, bez określonego rozszerzenia(typu)

**Rozszerzenie** - to tylko tekst, nadawany po to by inne programy rozpoznawały pliki w odpowiedni sposób(np. txt rozpozna komputer jako plik tekstowy i otworzy go w notatniku)

Możemy się odwołać do pliku, jeżeli zapiszemy go do obiektu np.

file = open("test","w")

Od momentu zapisania w “file“ - tak przechowywany plik nazywamy **UCHWYTEM**

**(z ang. handle)**, ponieważ uchwycił odwołanie się do pliku.

* **zamykanie**

Uchwyt, który przetrzymuje plik musi zostać zamknięty

file.write("sample")

file.close()

## 76. Wyjątki - blok try finally - na praktycznym przykładzie

**try** - z ang. próbuj

**finally** - z ang. ostatecznie

try:

file = open("test.txt", "w")

file.write("sample")

print(0/0)

a = 5

file.write("sample")

finally:

file.close()

W **try** cokolwiek wystąpi po błędzie (error) nie zostanie wykonane. Dzięki użyciu **finally** plik zostanie zamknięty, mimo błędu, który wystąpił. Wszystko co znajduje się w finally zostanie wykonane zawsze.

## 77. Otwieranie pliku korzystając z konstrukcji: with... as...:

**Automatyczne zamykanie plików za pomocą konstrukcji with,as**

**with** - z

z otwartym plikiem… w jakim trybie, zapisz go w…

podajemy operacje, jakie mają zostać wykonan

with open("test.txt", "w") as file:

file.write("sample")

a = 5

file.write("sample")

Po wykonaniu tej konstrukcji, plik który został otwarty i zapisany pod obiektem file, bez względu na to czy wystąpi wyjątek czy też nie plik zostanie zawsze zamknięty.

## 78. Odczytywanie danych - read vs readline i poprawne odczytanie polskich znaczków

**Odczytywanie danych**

with open("oceany.txt", "r") as file:

oceany = file.read()

Możemy zapisać z pliku zawartość z pomocą funkcji “read”.

W oceany znajdują się oceany ze znakiem /n (enter).

Gdy chcemy pominąć /n użyjemy **funkcji splitlines**

with open("oceany.txt", "r") as file:

oceany = file.read().splitlines()

Na przeczytanej treści jest **wywoływana funkcja splitlines, która ostatecznie zwraca listę wszystkich poszczególnych elementów pomijając /n.**

Aby sprawdzić kodowanie pliku możemy użyć funkcji encoding

file.encoding

**Kodowanie znaków** mogą być różne przez co program może nie czytać poprawnie polskich znaków (ł, ż, ć itp.)

Aby znaki były ze sobą spójne musimy zmienić kodowanie, aby w obu miejscach były identyczne. Wystarczy, że prześlemy argument nazwany i ustalimy kodowanie, na te który ma plik.

with open("oceany.txt", "r", encoding = "UTF-8") as file:

oceany = file.read().splitlines()

**Readline** - czyta pojedynczą linię (po każdym enterze)

oceany = file.readline()

**Readlines** - przeczyta wszystkie linie i od razu stworzy listę i pozostawi /n

oceany = file.readlines()

## 79. seek, tell - zmienianie 'wskaźnika' w pliku

**Readline** - czyta i zmienia miejsce wskaźnika pliku.

Gdy wyślemy kod :

print(file.readline())

zostanie przeczytana jedna linia. Wskaźnik czyta ciąg znaków, po czym dochodzi do końca linii, czyta enter i czeka na początku nowej linii na kolejną komendę jaką będzie miał wykonać.

**Aby sprawdzić miejsce wskaźnika skorzystamy z funkcji tell**

print(file.tell())

\n - również są wliczane w pozycję wskaźnika

**Funkcja seek**

**seek** - szuka(zmienia) na miejsce wskazane przez nas np. gdy chcę zacząć od początku (0)

file.seek(0)

**Możemy też odwołać się do konkretnego miejsca przy pomocy seek**

file.seek(4)

w tym przypadku pomija pierwsze 4 znaki i wywołuje od piątego miejsca.

Zastosowanie czytania od konkretnego miejsca w plikach, są wykorzystanie w np. w pliku jpg(obrazkowym), zapisuje się w nim informacje dotyczące np. szerokości, wysokości które znajdują się w konkretnym miejscu - przez co łatwo i szybko możemy odwołać się do interesującej nas części.

## 80. append - dopisywanie nowych elementów

a - A ppend (z ang. dopisywanie)

**append** - tryb, który pozwoli na dopisanie elementu na samym końcu pliku.

with open("oceany.txt", "a", "encoding = "UTF-8") as file:

file.write("Południowy")

**Dzięki funkcji append zmieniamy miejsce wskazywania na koniec, aby dopisać do kodu nową zawartość (nie usuwając pliku, który już istnieje)**

## 81. r+ w+ a+ - mnogie tryby otwierania plików

**Mnogie tryby** otwierania plików pozwalają na wykonanie dwóch czynności jednocześnie.

**Typy mnogie:**

* **r + (do czytania i pisania)** pozwala na wykonanie file.read oraz file.write odrazu na tym pliku, sprawdza czy dany plik istnieje i nie usuwa go.

with open("oceany.txt", "r+", encoding="UTF-8") as file:

* **w+ (do pisania i czytania**) różni się od r+ tym, że usunie zawartość istniejącego pliku lub stworzy go, gdy ten nie istnieje

with open("oceany.txt", "w+", encoding="UTF-8") as file:

file.readline()

* **a+ (“wieczny tryb” dopisywanie i czytanie)** wskaźnik dopisywania jest zawsze na samym końcu, jeśli plik nie istniał stworzy go

with open("oceany.txt", "a+", encoding="UTF-8") as file:

file.write("Ocean Arka")

* + **Jeżeli chcemy w tym trybie czytać - musimy przejść do początku (0)**

file.seek(0)

print(file.readline())

print(file.tell())

file.write("Ocean Arkadiusza Wielkiego")

## 83. ĆWICZENIE: FileNotFoundError exception

def read\_content\_of\_file(path):

try:

with open(path, "r", encoding="UTF-8") as file:

return file.read()

except FileNotFoundError:

print("Nie znaleziono pliku, podaj prawidłową ścieżkę")

nameOfFile = input("Podaj nazwę pliku do otwarcia: ")

fileContent = read\_content\_of\_file(nameOfFile)

## 84. ĆWICZENIE: Częstotliwość występowania słowa w pliku

Pamiętaj o obsłudze wyjątków!

FileNotFoundError występuje, gdy plik o podanej ścieżce nie zostanie znaleziony.

PermissionError występuje, gdy brak jest uprawnień do odczytu pliku.

Gdy wypiszesz dane skorzystaj z formatted stringa

Przykład:

imie = "Jan"

wiek = 30

print(f"Cześć, nazywam się {imie} i mam {wiek} lat.") # wyświetli "Cześć, nazywam się Jan i mam 30 lat."

"f" na początku ciągu znaków oznacza, że jest to tzw. f-string (ang. formatted string).

F-string to nowa wersja ciągów znaków w Pythonie, która pozwala na szybkie i łatwe formatowanie ciągów znaków za pomocą zmiennych i wyrażeń.

F-string pozwala na oszczędzenie czasu i linii kodu, ponieważ nie trzeba już używać operatora "+" do połączenia ciągów znaków ze zmiennymi.

Uwaga: f-string dostępny jest od Python 3.6. Jeśli używasz starszej wersji Pythona, możesz użyć operatora "+" do formatowania ciągów znaków.

# JSON

## 85. Czym jest JSON? Dlaczego z niego korzystamy? Jaki problem rozwiązuje JSON?

JSON - to format danych, który posiada wspólne reguły zapisu danych dla wszystkich języków programowania.

Dzięki czemu dane raz zapisane w formacie JSON mogą być odczytane przez każdego programistę nawet jeśli programuje w innym języku programowania.

## 86. Zapis danych do pliku w formacie JSON

Aby skorzystać z JSON

import json

Od teraz możemy skorzystać z metod, np.

dump - (z ang. zsypać/zwalić/zrzucić)

* dumps - zapisuje dane JSON do postaci stringowej np.

json.dumps(film)

Aby polskie znaki zostały poprawione wyświetlone należy napisać:

json.dumps(film, ensure\_ascii=False)

* dump - służy do zapisywania danych do pliku w postaci JSON.

with open ("sample.json", "w", encoding="UTF-8") as file:

json.dump(flim, file, ensure\_ascii=False)

## 88. Pretty Printer - ładnie wyglądający zapis/odczyt JSON

Aby kod, który wczytujemy z JSON był czytelniejsze - w metodzie dump i dumps mamy możliwość przesłania dodatkowego nazwanego argumentu - indent.

indent (z ang. wcięcie), którego długość określamy po znaku równości.

encodedFilm = json.dumps(film, ensure\_ascii=False, indent=4)

Możemy posortować alfabetycznie dane JSON (może przydać się podczas tworzenia analiz, porównywania dużej ilości danych itp.) Domyślna wartość sortowania kluczy ustawiona jest na False, ale gdy chcemy aby były one sortowane należy zmienić wartość na true.

encodedFilm = json.dumps(film, ensure\_ascii = False, indent=4, sort\_keys=True)

Gdy chcemy wczytać plik (np. do wyniku) aby był czytelniejszy możemy skorzystać z dwóch metod:

* wywołanie json.dumps na wyniku

print(json.dumps(wynik, indent = 4, ensure\_ascii = False, sort\_keys = True))

* import ze standardowej biblioteki funkcję pprint (z.ang. pretty print - ładnie drukowane) i wysyłamy tam nasz wynik

import pprint

pprint.pprint(wynik)