Skrypt do zajęć "Kodowanie jest przydatne - PYTHON"

Bożena Wieczerzycka
Liceum Ogólnokształcące
św. Marii Magdaleny
w Poznaniu

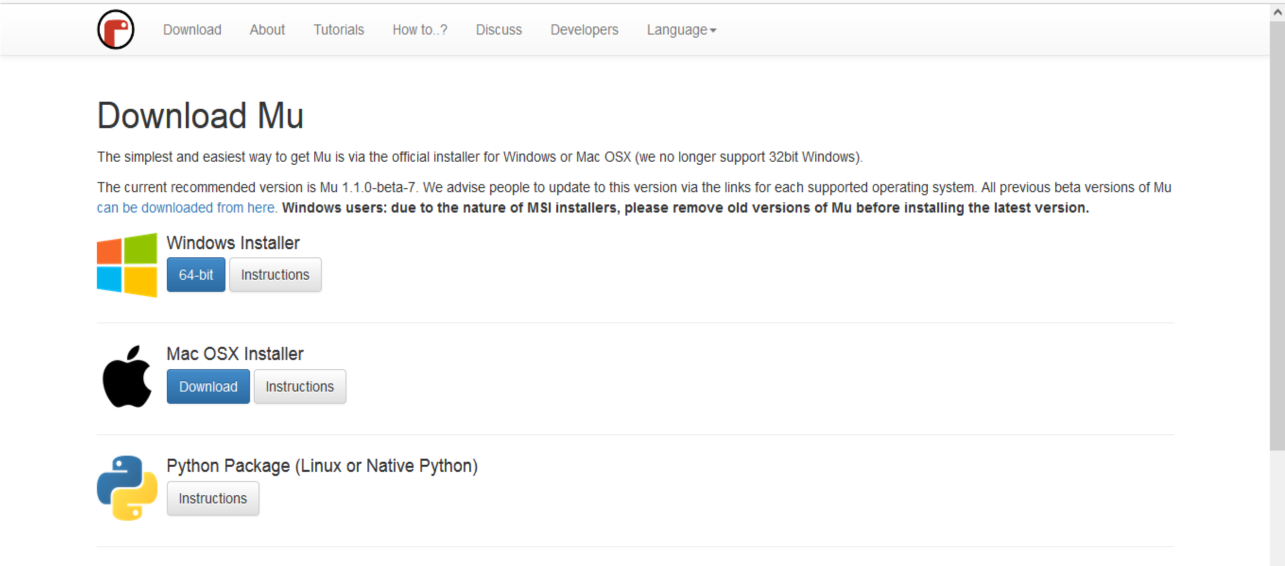
Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań”
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

Wprowadzenie (1h)

- Przygotowanie środowiska do pracy
- Nasz pierwszy program
- Jak będziemy pracowali

Przygotowanie środowiska do pracy

1. Przejdź na stronę https://codewith.mu i kliknij Download. Na ekranie pojawi się okno:



2. Wybierz wersję odpowiednią do systemu operacyjnego, pobierz i uruchom.

   Okno środowiska Mu wygląda następująco:

Nasz pierwszy program (pierwszy.py)

Wykonaj następujące czynności:

1. uruchom środowisko

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań"
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

2. wpisz następujący kod źródłowy:

Program (pierwszy.py):

```
# pierwszy program i funkcja wyjścia  - print

print("Twój pierwszy program")
print('Dzień dobry - mam nadzieję, \n że dowiem się za chwilę, jak masz na imię \n')
```

3. przed uruchomieniem zapisz plik na dysku - polecenie Zapisz
4. sprawdź, czy nie ma błędów składniowych - polecenie Sprawdź
5. uruchom program - polecenie Wykonuj

```
Wykonywanie: pr.py
Twój pierwszy program
Dzień dobry - mam nadzieję,
 że dowiem się za chwilę, jak masz na imię

>>>
```

Pliki źródłowe w języku Python mają rozszerzenie .py

Funkcja PRINT służy między innymi do wyświetlania tekstu na ekranie. Tekst musi być ujęty w znaki cudzysłowia lub zamknięty między apostrofami.

Znak "\n" jest znakiem specjalnym i powoduje przejście kursora do nowego wiersza.

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań"
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

Podstawowe typy danych (2h)

- Komentarze
- Operacje arytmetyczne
- Zmienne i wartości
- Liczby int oraz float
- String
- Input w konsoli Podgląd

Komentarze

Komentowanie kodu źródłowego jest bardzo pomocne w nauce programowania - ułatwiają analizę kodu źródłowego.

Komentarz jednowierszowy rozpoczyna się od znaku #.

Komentarz wielowierszowy musi być ujęty w potrójny apostrof lub w potrójny cudzysłów.

Operatory arytmetyczne

Operator	**Nazwa operatora**
+, -, *, /	- dodawanie, odejmowanie, mnożenie, dzielenie
**	- potęgowanie
//	- dzielenie całkowite - zwraca część całkowitą ilorazu
%	- dzielenie modulo - zwraca resztę z dzielenia

Program (operatory_arytm.py):

```
# podstawowe operacje arytmetyczne
print(5 + 10.5)  # dodawanie
print(-5 - 3, 5 * 2)  # odejmowanie i mnożenie
print(3 ** 3)  # potęgowanie
print(37 / 3)  # iloraz
print(37 // 3)  # dzieleniem całkowite-zwraca tylko część całkowitą ilorazu
print(37 % 3)  # dzielenie modulo-zwraca resztę z dzielenia dwóch liczb całkowitych
```

```
Wykonywanie: oper_arytm.py
15.5
-8 10
27
12.333333333333334
12
1
>>>
```

Zmienne i wartości

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań"
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

Zmienna to wydzielone miejsce w pamięci komputerowej, któremu nadajemy nazwę i w którym możemy przechowywać potrzebną informację. **Zmienna** w języku Python jest tworzona dopiero w momencie, gdy przypiszemy do niej **wartość**.

Nazwa zmiennej:

- musi zaczynać się litery;
- nie możemy używać polskich znaków oraz spacji;
- wielkość liter ma znaczenie.

Zmienna nie może istnieć bez wartości, ponieważ to wartość, a nie zmienna, posiada typ.

Zmienne typu całkowitego (int), typu rzeczywistego (float) oraz łańcuchy znaków (string)

Podstawowe typy danych to liczby całkowite, rzeczywiste, łańcuchy znaków.

Zmienna typu string jest inicjowana za pomocą znaków cudzysłowie lub apostrofów.

Różne łańcuchy znaków można składać poprzez użycie operatora + lub powielać (mnożyć) poprzez użycie operatora *.

Program (zmienne.py):

```
""" zmienne typu int(liczby całkowite), float(liczby rzeczywiste)
oraz typu string(łańcuch znaków)"""
calkowita = 5
print("mamy zmienną calkowitą, której wartość= ", calkowita)
rzeczyw1 = 5.0
rzeczyw2 = float(5)
print("mamy zmienną rzeczyw1, której wartość= ", rzeczyw1)
print("mamy zmienną rzeczyw2, której wartość= ", rzeczyw2)
tekst1 = 'Ala ma'
tekst2 = " kota"
wynik = tekst1 + ' ' + tekst2
print(wynik)
wynik = tekst2 * 3
print(wynik)
```

```
Wykonywanie: zm.py
mamy zmienną calkowitą, której wartość=  5
mamy zmienną rzeczyw1, której wartość=  5.0
mamy zmienną rzeczyw2, której wartość=  5.0
Ala ma  kota
 kota kota kota
>>>
```

Funkcja Input

Funkcja Input służy do wczytywania danych podanych przez użytkownika.

Program (funkcja_input.py)

 # wczytywanie danych - funkcja input

imie = input("Jak masz na imię? ")

wiek = input('Ile masz lat? ')

zdanie = imie + " masz " + wiek + " lat. "

print(zdanie)

nowy_wiek = wiek+10

print('Za dziesięć lat bedziesz miała ', nowy_wiek)

Wykonywanie: funkcja_input1.py

```
Jak masz na imię? Ala
Ile masz lat? 23
Ala masz 23 lat.
Traceback (most recent call last):
  File "d:\użytkownik\dokumenty\0rok 2021 2022\enigma\programy\zaj2 2022-02-15\funkcja_input1.py", line 6, in <module>
    nowy_wiek = wiek+10
TypeError: can only concatenate str (not "int") to str

>>> |
```

UWAGA!

Jest błąd w linii 6 (*nowy_wiek = wiek+10)*

Funkcja input zwraca wartość typu string (łańcuch znaków), czyli na tej wartości nie można wykonać żadnej operacji matematycznej.

Poprawiony program (funkcja_input.py)

 # wczytywanie danych - funkcja input

imie = input("Jak masz na imię: ")

wiek = int(input('Ile masz lat: '))

 # funkcja input wczytała łancuch znaków, a int zmieniła go na wartość liczbową

print(imie, ' masz ', wiek, ' lat.')

nowy_wiek = wiek + 10

print('Nowy wiek= ', nowy_wiek)

Wynik działania kodu:

Wykonywanie: funkcja_input.py

```
Jak masz na imię: Ala
Ile masz lat: 23
Ala  masz  23  lat.
Nowy wiek=  33
>>> |
```

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań"
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

Instrukcje warunkowe (2h)

- Instrukcja If
- Operatory porównania
- Operatory logiczne

Instrukcja If

Instrukcja warunkowa to jedna z podstawnych instrukcji w każdym języku programowania. Instrukcje lub blok instrukcji wykonuje się tylko, gdy określony warunek (lub zestaw warunków) jest spełniony.

Składnia instrukcji warunkowej IF:

1. instrukcja if...else

```
if  warunek :
    instrukcja_10
    instrukcja_11
    ………
else:
    instrukcja_20
    instrukcja_21
    ………
```

2. zagnieżdżona instrukcja if...else

```
if  warunek1:
    instrukcja1
    instrukcja...
    if  warunek2:
        instrukcja2
        instrukcja....
    else:
        instrukcja3
        instrukcja...
else:
    instrukcja4
    instrukcja....
```

3. instrukcja warunkowa if...elif...else

```
if   warunek1:
     instrukcja1
     instrukcja....
```

```
elif  warunek2:
      instrukcja2
      instrukcja....
elif  warunek3:
      instrukcja3
      instrukcja....
else:
      instrukcja4
      instrukcja....
```

4. prosta instrukcja if

```
if  warunek:
      instrukcja_10
      instrukcja_11
      ….
```

Operatory porównań

Operator porównania	**Znaczenie**	**Przykład**
==	równe	a == b
!=	różne	a != b
<	mniejsze	a < b
<=	mniejsze lub równe	a <= b
>	większe	a > b
>=	większe lub równe	a >=

Operatory logiczne

Nazwa operatora	**Operator**
Negacja NOT	!
Koniunkcja AND	&&
Alternatywa OR	||

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań”
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

Operatory porównania w Pythonie można pogrupować w następujący sposób: a == b == c lub x <= y >= 10.

Program (pole_kwadratu.py) - instrukcja if...else

```
# Oblicz pole kwadratu o boku a
a = int(input("Podaj bok kwadratu: "))
if a > 0:
   print('Pole kwadratu= ', a ** 2)
else:
   print('Taki kwadrat nie istnieje')
```

Program (imie_i_nazwisko.py) - instrukcja if...else

```
# Program sprawdza, czy użytkownik podał poprawne imię i nazwisko
imie = input("Podaj Swoje imię: ")
nazwisko = input('Podaj Swoje nazwisko: ')
if imie == 'Ala' and nazwisko == "Kot":
   print('Dane sa prawidłowe')
else:
   print('Błędne dane')
```

Program (pole_prostokata.py) - instrukcja if...else

```
# Oblicz pole prostokąta o bokach a i b
a = float(input('Podaj długość pierwszego boku prostokąta: '))
b = float(input('Podaj długość drugiego boku prostokata: '))
if a > 0 and b > 0:
   print("Pole prostokata = ", a*b)
else:
   print('Podałeś błędne dane')
```

Program (kiedy_imieniny.py) - instrukcja if...elif...else

```
# program podaje termin imienin
imie = input('Podaj Swoje imię: ')
if imie == 'Ewa' or imie == 'Adam' or imie == 'ewa' or imie == 'adam':
   print('24 grudnia')
elif imie == 'Jakub' or imie == 'jakub':
   print('5 lutego')
```

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań”
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

```
elif imie == 'Maria' or imie == 'maria':
   print('3 maja')
else:
   print('Niestety, zgubiłam pozostałe kartki z kalendarza')
```

Program (czy_parzysta.py) - instrukcja if...elif...else

```
# program sprawdza czy liczba jest parzysta
liczba = int(input('Podaj liczbę: '))
if liczba == 0:
   print("Liczba jest równa 0")
elif liczba % 2 == 0:
   print("Liczba jest parzysta")
else:
   print("Liczba jest nieparzysta")
```

Każdy z powyższych programów proszę przetestować dla różnych wartości danych.

Pętle (2h)

- Pętla While
- Pętla For
- Funkcja range
- Break oraz Continue

Instrukcje iteracyjne (WHILE, FOR) służą do powtarzaniu określonych czynności.

Instrukcja iteracyjna WHILE

Pętlę stosujemy wtedy, gdy nie znamy z góry liczby powtórzeń.

Pętla wykonuje się dopóki warunek logiczny jest spełniony, czyli przyjmuje wartość logiczną prawda.

Składnia instrukcji WHILE:

```
while warunek:
        instrukcja_10
        instrukcja_11
        instrukcja….
```

Program (while1.py)

```
# pętla while - powtarzaj dopóki spełniony jest warunek
odp = input('Czy mam rozwiązać zadania (t/n)? ')
while odp == 't' or odp == 'T':
    print('rozwiązuję zadanie')
print('Czas na odpoczynek')
```

Pętla działa w nieskończoność.

Modyfikacja programu while1 (while1.py)

```
# pętla while - powtarzaj dopóki spełniony jest warunek
odp = input('Czy mam rozwiązać zadania (t/n)? ')
while odp == 't' or odp == 'T':
    print('rozwiązuję zadanie')
    odp = input('Czy mam rozwiązać zadania (t/n)? ')
print('Czas na odpoczynek')
```

Program (while2.py)

```
''' pętla while - powtarzaj dopóki spełniony jest warunek
 i policz ile razy wykona się pętla'''
ile = 0  # inicjalizacja zmiennej - nadanie wartości początkowej
```

```
odp = input('Czy mam rozwiązać zadanie (t/n)? ')
while odp == 't' or odp == 'T':
   ile = ile + 1
   ''' instrukcja przypisania wartości zmiennej (najpierw prawa strona i wartość
   wyrażenia staje się nową wartością zmiennej'''
   print('rozwiązuję zadanie')
   odp = input('Czy mam rozwiązać zadanie (t/n)? ')
print('Czas na odpoczynek ', ile, ' zadania')
```

Instrukcja iteracyjna FOR

Pętlę FOR stosujemy wtedy, gdy chcemy by określona czynność lub grupa czynności wykonała się z góry określoną liczbę razy.

Składnia instrukcji FOR:

for licznik **in** wartości po których będziemy iterować:

instrukcja_10

instrukcja_11

instrukcja....

Pętla FOR zawiera słowo kluczowe **for**, nazwę zmiennej, która odpowiada za kolejne powtórzenie, słowo kluczowe **in**, wartości, po których będziemy iterować (np. lista, krotka, słownik, string) oraz polecenie lub polecenia, które będą wykonywały się w pętli.

Funkcja range

Do utworzenia w pętli FOR listy wartości służy wbudowana **funkcja range**. Umożliwia ona generowanie ciągów liczb całkowitych.

Funkcja range może być wywoływana w następujących wariantach:

- range(n) - funkcja generuje listę kolejnych liczb od 0 do n-1
- range(k, n) - funkcja generuje listę kolejnych liczb od k do n-1
- range(k, n, krok) - funkcja generuje listę kolejnych liczb od k do n-1. Kolejne wygenerowane wartości będą zmieniały się o wartość kroku

Program (for1.py)

```
# pętla for - wykonaj trzy zadania
for zadania in range(1, 4):
   print('rozwiązuję zadanie')
print('Czas na odpoczynek')
```

Program (for2.py)

```
# pętla for - wykonaj n zadań (wartość n podaje użytkownik)
```

```
n = int(input('Ile zadań mam wykonać: '))
for zadania in range(1, n + 1):
    print('rozwiązuję zadanie')
print('Czas na odpoczynek', n, ' zadań')
for zadania in range(n):  # iteracja zaczyna się od zera
    print('rozwiązuję zadanie')
print('Czas na odpoczynek', n, ' zadań')
```

Program (wypisz_parzyste.py)

```
# wypisz w jednej linii liczby parzyste od 2 do 20
for liczba in range(2, 21, 2):
    print(liczba, ' ', end='') # end='' - kursor nie przechodzi do nowego wiersza
# wypisz w następnej linii parzyste liczby ujemne od -20 d0 -2
print('\n') # znak sterujący - kursor przechodzi do nowego wiersza
for liczba in range(-20, 0, 2):
    print(liczba, ' ', end='')
```

Program (suma_n_parzystych.py)

```
'''napisz program, który obliczy sumę n kolejnych
liczb parzystych (wartość n podaje użytkownik)'''
n = int(input('Podaj n: '))
suma = 0
skladnik = 0
for kolejna_parzysta in range(1, n+1):
    skladnik = skladnik + 2
    print(skladnik, ' ', end='')
    suma = suma + skladnik
print('\n', suma)
```

Instrukcja skoku break

Polecenie **break** jest informacją dla języka Python, aby natychmiast przerwać wykonywanie pętli (kończy działanie pętli). Interpreter przechodzi do dalszej części instrukcji następujących po bloku pętli.

Może być stosowana w pętli **while** i w pętli **for**.

Możemy przykładowo, wykorzystać to do przetwarzania listy w pętli for, po czym wyjść z niej, gdy zostanie znaleziona konkretna wartość.

Program (break.py)

```
# break
ile_zadan = int(input('Ile zadań ma rozwiązać? '))
for zadanie in range(1, ile_zadan+1):
    if zadanie == 3:
        break
    else:
        print('rozwiązuję zadanie', zadanie)
print('Ilość zadań = ', zadanie)
print('Czas na odpoczynek')
```

Instrukcja skoku continue

Polecenie **continue** kończy iterację bieżącej pętli. Interpreter wraca do początku pętli i wyrażenie warunkowe jest ponownie sprawdzane, aby określić, czy pętla zostanie wykonana ponownie, czy zakończyć i przejść dalej.

Program (continue.py)

```
# continue
ile_zadan = int(input('Ile zadań ma rozwiązać? '))
for zadanie in range(1, ile_zadan+1):
if zadanie == 3:
continue
print('rozwiązuję zadanie', zadanie)
print('Czas na odpoczynek')
```

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań”
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

Struktury danych (4h)

W języku Python możemy wyróżnić cztery główne struktury danych: listy, krotki, zbiory i słowniki . Każda z nich ma odmienną budowę i zastosowania, jednak wszystkie są w pewien sposób połączone.

Listy

Listy to złożone struktury danych, które służą do przechowywania danych różnego typu. Poszczególne elementy listy są przedzielone przecinkiem. Wszystkie wartości listy muszą być ujęte w nawiasy kwadratowe.

Tworzenie list:

lista1 = [0, ‘Ala’, -12.45, False]

Lista może być pusta:

pusta_lista1 = [ ]

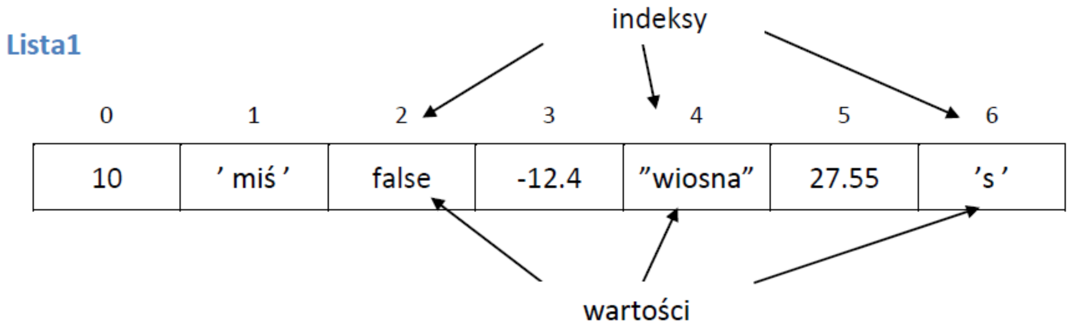
pusta_lista2 = list()

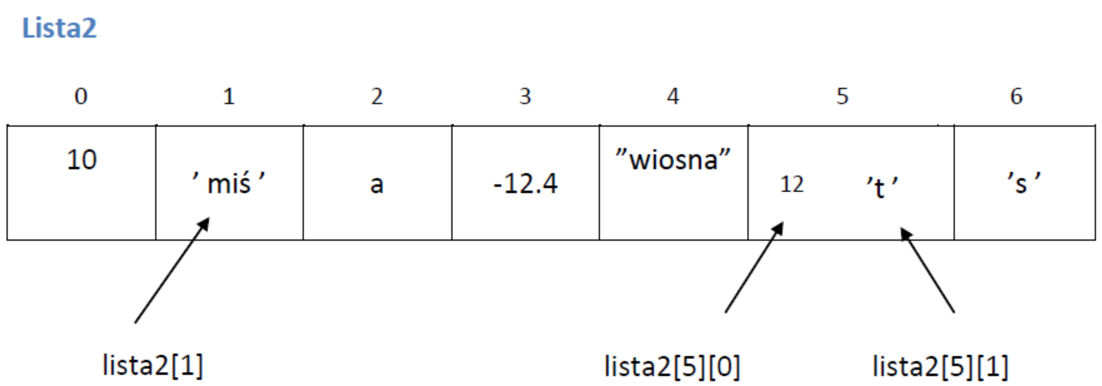
Listy mogą być zagnieżdżone, np.:

lista2 = [' wiosna ', 2.0, 5, [10, 20]]

Każda wartość zapisana na liście ma określone miejsce, które nazywamy indeksem. Listy są indeksowane (numerowane) od 0 – oznacza to, że żeby dostać się do pierwszego elementu musimy podać nazwę listy oraz w nawiasie kwadratowym indeks elementu np. lista[0].

Jeśli indeks ma wartość ujemną, to oblicza się go wstecz od końca listy. Jeśli odwołamy się do wartości, która nie istnieje, pojawi się komunikat o błędzie IndexError (IndexError: list index out of range)





Operacje na listach:

- tworzenie listy
- wywołanie elementu listy za pomocą indeksu
- modyfikacja elementu listy, czyli podmiana elementu listy na inny
- wycinanie ('slicing') wybranego ciągu elementów z listy

  lista[1:] - elementy od pierwszego do ostatniego

  lista[1:3] - fragment listy obejmujący elementy od drugiego do trzeciego

  lista[:3] – elementy od pierwszego (domyślnie jeśli nie podano) do trzeciego

  lista[2:] – od trzeciego do ostatniego

  lista[:] – wszystkie elementy listy

  lista[-2:] – dwa ostatnie elementy

  lista[:-1] – od pierwszego do przedostatniego

  lista[3:-3] – od czwartego do czwartego od końca

  lista[::2] – wyświetl wszystko, ale co drugi element listy

  lista[::-1] – odwrócenie kolejności listy – wyświetlenie elementów od końca do początku
- łączenie (dodawanie) list
- mnożenie (powielanie) listy

Program (lista_operacje.py)

```
# tworzenie listy
zmienna = 3
lista1 = [2, False, -10.2, 'atom', 's', zmienna]
lista2 = []  # lista pusta
print(lista1)
print(lista2)
print(lista1[3])  # odwołanie do elementu listy o indeksie 3
print(lista1[-3])  # odwołanie do elementu listy - licząc od ostatniego elementu
lista1[4] = 'a'  # modyfikacja elementu listy
print(lista1)
print(lista1[3:])  # wycinanie fragmentu listy - od  elementu o indeksie 3 do końca
print(lista1)
lista2 = lista1[2:4] # wycinanie z listy - od indeksu 2 do indeksu 3 print(lista2)
lista2 = lista2 + [1, 2]  #  dodawanie list
print(lista2)
lista2 = lista2 * 3  # powielanie (mnożenie listy)
```

print(lista2)

del(lista2)

print(lista2)

Metody:

- funkcja .append() - dodawanie elementów na koniec listy
- funkcja .insert() - dodawanie elementów w dowolnie wybranym miejscu
- funkcja len() - sprawdzenie długości listy
- funkcje .pop() oraz .remove() - usuwanie elementów

  Funkcja .pop() usuwa element po numerze indeksu, natomiast .remove() wyszukuje pierwsze wystąpienia podanej zmiennej w liście.
- funkcja .index() - zwraca indeks pierwszego wystąpienia szukanego elementu w liście
- funkcja .count() - zwraca liczbę wystąpień wybranego elementu listy
- max() - zwraca największy element listy
- min() - zwraca najmniejszy element listy
- funkcja .reverse()- odwraca kolejność danej listy
- funkcja .sort() - sortuje elementy na liście
- funkcja .clear() - czyszczenie całej listy

Program (lista_metody.py)

 # listy - metody

zmienna = 3

lista1 = ['l', -10.2, zmienna, 'l', 'l']

lista2 = [] # lista pusta

print(len(lista1)) # zwraca długość listy

print(len(lista2))

lista1.append('nowy na końcu') # dodanie elementu na koniec listy

print(lista1)

lista1.insert(1, 'nowy drugi element') # dodanie elementu w miejsce o podanym indeksie

print(lista1)

lista1.remove('l') # usunięcie pierwszego wystąpienia danego elementu

print(lista1)

print(lista1.index('l')) # zwraca indeks pierwszego wystąpienia szukanego elementu w liście

print(lista1.count('l')) # zwraca liczbę wystąpień wybranego elementu listy

lista2 = [3, 5, 2, 56, -67]

```
print(max(lista2))  # zwraca największy element listy
print(min(lista2))  # zwraca najmniejszy element listy
lista2.reverse()  # odwraca kolejność danej listy
print(lista2)
lista2.sort()  # sortuje elementy na liście
print(lista2)
lista2.clear() - czyści listę
print(lista2)
```

Program (wczytywanie_do_listy_for.py)

```
# wczytywanie n wartości do pustej listy - pętla for
lista2 = []
n = int(input('Podaj ile chcesz liczb na liście: '))
lista2 = [0] * n
for el in range(n):
    liczba = int(input('Podaj liczbę: '))
    lista2[el] = liczba
print(lista2)
# lub z wykorzystaniem .append
lista3 = []
for el in range(n):
    lista3.append(int(input('Podaj liczbę: ')))
print(lista3)
```

Program(wczytywanie_do_listy_while.py)

```
# wczytywanie wartości do pustej listy - pętla while
lista = []
liczba = int(input("Podaj liczbę, 0 - zakończ: "))
while liczba != 0:
    lista.append(liczba)
    liczba = int(input("Podaj liczbę, 0 - zakończ: "))
print(lista)
```

Program (losowanie_do_listy.py)

```
# losowanie n danych do pustej listy
```

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań”
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

```
import random
lista = []  # pusta lista
n = int(input('Podaj ile chcesz liczb na liście: '))
for i in range(n):
    lista.append(random.randint(0, 100))
print(lista)
```

Program (spr_czy_jest_element_na_liście.py)

```
# sprawdzenie, czy na liście znajduje się dany element
lista1 = [2, 'wiosna', -10.2, 'atom', 's', 'Kasia']
if "wiosna" in lista1:
    print("Jest na liście!")
    print(lista1.index('wiosna'))
else:
    print("Nie ma na liście!")
```

Program (iterowanie_po_liście.py)

```
# iterowanie po elementach listy-każdy element listy w nowej linii:
zmienna = 3
lista1 = [2, False, -10.2, 'atom', 's', zmienna]
for i in lista1:
    print(i)
```

Krotki (Tuples)

Krotki zwane czasem tuplami są podobne do list, ale nie są modyfikowalne. W związku z tym wiele operacji, które działają na listach, będą także działały na krotkach, ale nie te, które wiążą się z ich modyfikacją. Nie możemy zatem dodawać, usuwać czy podmieniać elementów krotek, które zostały już wcześniej utworzone.

Tworzy się je podobnie jak listy, z tą różnicą, że zamiast nawiasów kwadratowych [] używamy nawiasów okrągłych ().

Podobnie jak w listach, elementy w krotce mają określony porządek. Są one indeksowane od 0. Ujemne indeksy idą od końca krotki, tak samo jak w listach.

Krotki można nadpisywać lub wycinać - dostajemy nową krotkę.

Program (krotka_operacje.py)

```
# operacje na krotkach
krotka1 = (1, 2, 3, ['a', 'k'], (-2.0, -5.0))
```

```
# dostęp do elementu poprzez wskazanie indeksu
print(krotka1[3])
print(krotka1[4][1])
# odczyt indeksu wskazanego elementu
print(krotka1.index((-2.0, -5.0)))
krotka1 = krotka1 + (4, 5)  # nadpisywanie
print(krotka1)
krotka1 = krotka1 + (6, )
print(krotka1)
krotka1 = krotka1[:3] + (1.5, -2, 2.5, ) + krotka1[:3]
print(krotka1)
krotka1 = krotka1 * 2
print(krotka1)
```

Program (wczytywanie_do_krotki_for.py)

```
# wczytywanie n wartości do pustej krotki - pętla for
krotka = ()
n = int(input('Podaj ile chcesz liczb w krotce: '))
for el in range(n):
   liczba = int(input('Podaj liczbę: '))
   krotka = krotka + (liczba,)
print(krotka)
```

Program (iterowanie_po_krotce.py)

```
# iterowanie po elementach krotki-każdy element krotki w nowej linii:
zmienna = 5
krotka1 = (2, False, -10.2, 'atom', 's', zmienna-1)
for i in krotka1:
   print(i)
```

Program (konwersja_listy_krotki.py)

```
# tworzenie z krotki listy i na odwrót
zmienna = 3
krotka1 = (2, False, -10.2, 'atom', 's', zmienna)
lista1 = list(krotka1)  # z krotki1 powstała lista1
```

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań”
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

```
print(lista1)
print(type(lista1))
lista1.append('ostatni')
print(lista1)
krotka2 = tuple(lista1)  # z lista1 powstała krotka2
print(krotka2)
print(type(krotka2))
```

Zadanie 1: Napisz program, który obliczy sumy poszczególnych krotek według elementów.

Dane wejściowe:

(2, 2, 5, 4)

(3, 5, 10, -10)

(5.3, 2, 3, 1)

Dane wyjściowe:

(10.3, 9, 18, -5)

Program (krotki_zadanie1.py)

```
kr1 = (2, 2, 5, 4)
kr2 = (3, 5, 10, -10)
kr3 = (5.3, 2, 3, 1)
print("dane wejściowe:")
print(kr1)
print(kr2)
print(kr3)
krotka = ()
for el in range(4):
   suma = kr1[el] + kr2[el] + kr3[el]
   krotka = krotka + (suma,)
print(krotka)
```

Zadanie2: Napisz program, który rozpakuje krotkę do pojedynczych zmiennych.

Program (krotki_zadanie2.py)

```
krotka = (4, 8, False, 'a')
print(krotka)
```

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań"
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

```
z1, z2, z3, n4 = krotka
print(z1, z2, z3, n4)
print(z1+z2)
```

Zadanie 3: Napisz program przechowujący w strukturze listy kwadraty 20 kolejnych liczb naturalnych. Elementem listy o indeksie 0 powinna być liczba 0, elementem o indeksie 1 - liczba 1, elementem 0 indeksie 2 - liczba 4, elementem o indeksie 3 - liczba 9 itd.

Program (lista_kwadraty_liczb.py)

```
# program przechowujący w strukturze listy kwadraty 20 kolejnych liczb naturalnych.
kwadraty = [0] * 20


for i in range(20):
   kwadraty[i] = i * i
   print("Dla [",i,"]=",kwadraty[i])
```

Zadanie 4: Napisz program, który wypisze liczby z danej listy po usunięciu z niej liczb parzystych.

Program (lista_bez_parzystych.py)

```
import random
lista = []  # pusta lista
n = int(input('Podaj ile chcesz liczb na liście: '))
for i in range(n):
   lista.append(random.randint(0, 100))
print(lista)

for i in range(n - 1, -1, -1):
      if lista[i] % 2 == 0:
         lista.pop(i)
print(lista)
```

lub

```
import random
lista = []  # pusta lista
n = int(input('Podaj ile chcesz liczb na liście: '))
for i in range(n):
   lista.append(random.randint(0, 100))
print(lista)
```

```
for i in range(n - 1, -1, -1):
    if lista[i] % 2 == 0:
        lista.pop(i)
print(lista)
```

Zbiór (set)

{ 31, ’Elżbieta’, -34.62, (’s’, 2) }

Jest to struktura danych charakteryzuje się unikalnością elementów. Zbiór może składać się z różnych elementów liczb, łańcuchów, krotek. Kolejność elementów w zbiorze jest niezdefiniowana. Elementami zbioru nie mogą być listy i inne zbiory.

Zbiory reprezentują zbiory matematyczne, dlatego wyposażone są w metody związane z algebrą zbiorów.

Działania na zbiorach:

- tworzenie zbioru
- łączenie dwóch zbiorów - operator | (funkcja .union)
- część wspólna dwóch zbiorów - operator & (funkcja .intersection)
- różnica dwóch zbiorów - operator (funkcja .difference)
- różnica symetryczna dwóch zbiorów - operator ^
- funkcja .add() - dodawanie pojedynczego elementu do zbioru
- funkcje .remove(), .pop() oraz .discard() - usuwanie pojedynczego elementu zbioru

Funkcja .remove() - zwraca błąd, jeżeli dana wartość nie istnieje w zbiorze

Funkcja .discard() - usuwa dany element, jeśli istnieje on w zbiorze, w przeciwnym wypadku nie sygnalizuje błędu

Funkcja -.pop() - zwraca błąd, jeżeli zbiór jest pusty

- funkcja .issubset - sprawdzenie, czy jeden zbiór jest podzbiorem drugiego
- funkcja len() - sprawdzenie liczebności (wielkości) zbioru
- funkcja .copy() - kopiowanie zbioru
- funkcja .clear() - czyszczenie zbioru

Program (operacje_zbiór.py)

```
# zbiory - tworzenie
zbior1 = {0, 1, 2, 3, 4}
zbior2 = ()  # zbiór pusty
print(zbior1)
print(zbior2)
```

```
# zbior3 = {[1, 2, 3], 's'} - lista nie może być elementem zbioru
# print(zbior3)
#zbior = {1, 2, {3, 4}} - zbiór nie może być elementem zbioru
#print(zbior4)
zbior1.add(5)  # dodawanie elementu do zbioru - .add()
print(zbior1)
zbior1.add(5)  # zbiór elementów unikatowych
print(zbior1)
zbior1.remove(0)  # usuwanie elementu ze zbioru - .remove(), discard(), .pop()
print(zbior1)
print(1 in zbior1)  # sprawdzenie czy dana wartość jest w zbiorze
print('a' in zbior1)
# operacje na zbiorach
zbior1 = {0, 1, 2, 3, 4}
zbior2 = {3, 4, 5, 6}
print(zbior1 | zbior2)  # suma
print(zbior1 & zbior2)  # iloczyn
print(zbior1 - zbior2)  #  różnica
print(zbior2 - zbior1)
print(zbior1 ^ zbior2)  # różnica symetryczna
```

Napisz program, który usunie przecięcie dwóch zbiorów z pierwszego zbioru.

```
zbior1 = {0, 1, 2, 3, 4, 12, 13, 15}
zbior2 = {0, 1, 2, 3, 4, 7, 8, 9}
print(zbior1)
print(zbior2)
zbior1 = zbior1 - (zbior1 & zbior2)
print(zbior1)
print(zbior2)
```

Napisz program, który znajdzie w zbiorze pierwszym elementy, których nie ma w drugim zbiorze.

```
zbior1 = {0, 1, 2, 3, 4, 12, 13, 15}
zbior2 = {0, 1, 2, 3, 4, 7, 8, 9}
print(zbior1)
```

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań”
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

```
print(zbior2)
zbior1 = zbior1 - zbior2
print(zbior1)
print(zbior2)
```

Napisz program który sprawdzi, czy dwa podane zbiory nie mają wspólnych elementów.

```
zbior1 = {0, 11, 22, 23, 24, 12, 13, 15}
zbior2 = {0, 1, 2, 3, 4, 7, 8, 9}
print(zbior1)
print(zbior2)
if len(zbior1 & zbior2) == 0:
   print('nie ma')
else:
   print('ma')
```

Słownik (dict)

Słownik jest to nieuporządkowany zbiór par (klucz: wartość), przy czym klucze muszą być różne. Słownik jest strukturą modyfikowalną i nieposortowaną.

Indeksem słownika jest klucz. <u>Klucz musi być unikatowy - nie może się powtórzyć</u>. Słownik tworzymy "wkładając" pary klucz: wartość w nawiasy klamrowe, oddzielając je przecinkiem.

klucz 1 **wartość 1** **klucz 2** **wartość 2**

słownik = { ‘styczeń’ : 1, ‘luty’ : 2, ‘marzec’ : 3}

Słownik

klucz	wartość
styczeń	1
luty	2
marzec	3

Operacje na słownikach:

- tworzenie słowników
- dodawanie nowej pary klucz-wartość
- modyfikacja wartości w słowniku
- usuwanie elementu

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań”
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

- funkcja len() - sprawdzenie liczebność par „klucz – wartość" w słowniku
- funkcja .update - aktualizować słownika w oparciu o inny słownik
- funkcje. keys() / .values() - zwraca listę wszystkich kluczy słownika / zwraca listę wszystkich wartości
- funkcja .items - zwraca listę elementów (klucz, wartość).
- funkcja .clear - wyczyszczenie słownika (usunięcie wszystkich par)

UWAGA! Możesz dostać się do wartości za pomocą klucza, ale nie możesz dostać się do klucza za pomocą wartości.

Program (operacje_słownik.py)

```
# słowniki - tworzenie
slownik1 = {1 : ["Jan", "Kos", 4.7],
            2 : ["Lech", "Las", 4],
            3 : ["Ola", "Mik", -2]}
slownik2 = {}
print(slownik1)
print(slownik2)
slownik1[5] = ['Jaś', 'Bąk', 1]  # dodanie nowej pary klucz-wartość
print(slownik1)
# odczyt wartości  - odwołanie się do wartości poprzez klucz
print(slownik1[3])
print(slownik1[3][1])
# odwołanie do nieistniejącego klucza - zabezpieczenie funkcją .get()
# print(slownik1[4])
klucz = 'a'
print(slownik1.get(klucz, 'podany klucz nie istnieje'))
# sprawdzenie, czy dany klucz jest w słowniku
klucz = 4
if klucz in slownik1:
   print(' - klucz ', klucz, 'jest w słowniku')
else:
   print(' - klucz ', klucz, 'nie istnieje w słowniku')
# modyfikowanie wartości słownika o danym kluczu -
slownik1[3] = ['Ela', 'Mak', 45]
print(slownik1)
```

```
slownik1[3][2] = (slownik1[3][2] - 15)/5
print(slownik1[3])
slownik1[3] = slownik1[3] + ['s'] + ['Al', 1]
print(slownik1)
slownik1[2] = slownik1[2] * 2  # powielanie elementu słownika
print(slownik1)
del slownik1[3]  # usunięcie pary klucz-wartość
print(slownik1)
slownik2 = {'10': (10, 2)}  # utworzenie drugiego słownika
slownik1.update(slownik2) # aktualizacja słownika1 o słownik2
print(slownik1)
print(len(slownik1))  # sprawdzanie ile par jest w słowniku
slownik1.clear()  #usuwa wszystkie pary
print(slownik1)
```

Program (iterowanie_po_słowniku.py)

```
# iterowanie po słowniku
slownik1 = {"P1" : ["Jan", "Kos", 4.7],
           'P2' : ["Lech", "Las", 4],
          'P3' : ["Ola", "Mik", -2]}
print(slownik1)
for klucz in slownik1.keys():  #iterowanier po kluczach
   print(klucz)
for wartosc in slownik1.values():  #iterowanier po wartościach
   print(wartosc)
for klucz, wartosc in slownik1.items():  # iterowanie po parach: klucz, wartość
   print(klucz, '    ',wartosc)
for klucz in slownik1.keys():  # iterowanie po kluczach i aktualizacja wartości
   slownik1[klucz][2] = slownik1[klucz][2] + 10
   print(slownik1[klucz][0], ' wiek= ', slownik1[klucz][2])
```

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań"
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

Funkcje (3h)

Funkcja to fragment kodu, który odpowiada za wykonanie konkretnego zadania i który można wielokrotnie używać w różnych miejscach programu.

Dzięki stosowaniu funkcji kod programu jest bardziej przejrzysty i łatwiejszy do modyfikowania w przyszłości orazz unikamy powtórzeń.

Funkcja musi być zdefiniowana:

def *nazwa_funkcji (lista parametrów):*
    blok instrukcji

UWAGA! lista parametrów jest opcjonalna - nie musi wystąpić.

W treści funkcji umieszczamy instrukcje, które odpowiadają za wykonanie zadania (stanowią pewną całość).

Efekt działania funkcji albo wyświetlamy na ekranie albo za pomocą polecenia return przekazujemy z powrotem do instrukcji, która wywołała funkcję.

Program (pole_prostokata_funkcje.py)

```
# Oblicz pole prostokąta o bokach a i b - funkcja
def pole_prostokata(b1,b2):
    return b1 * b2
a = float(input('Podaj długość pierwszego boku prostokąta: '))
b = float(input('Podaj długość drugiego boku prostokata: '))
def spr_danych(b1,b2):
    if b1 >0 and b2 > 0:
        return True
    else:
        return False
if spr_danych(a, b) != False:
    print("Pole prostokata = ", pole_prostokata(a, b))
else:
    print('Podałeś błędne dane')
```

Program (usuwa_parzyste_z_listy.py)

```
# program, który z danej listy utworzy nową listę bez liczb parzystych.
import random
lista = []  # pusta lista
n = int(input('Podaj ile chcesz liczb na liście: '))
```

```
for i in range(n):
    lista.append(random.randint(0, 100))    # losowanie liczb z przedziału (0,100)
print(lista)
nowa_lista =[]
for i in lista:
    if i%2 != 0:    # sprawdzamy czy reszta z dzielenia przez 2 jest różna od 0
        nowa_lista.append(i)   # dodajemy liczbę nieparzystą do nowej listy
print(nowa_lista)
```

Program (usuwa_parzyste_z_listy_funkcje.py)

```
# program usuwa parzyste z listy - funkcje
def losuj_do_listy():
    import random
    lista = []  # pusta lista
    n = int(input('Podaj ile chcesz liczb na liście: '))
    for i in range(n):
        lista.append(random.randint(0, 100))
    return (lista)
def usun_parzyste (lista):
    nowa_lista = []
    for i in lista:    # sprawdzamy każdy element z listy przekazanej poprzez parametr
        if i%2 != 0:    # sprawdzamy czy reszta z dzielenia przez 2 jest różna od 0
            nowa_lista.append(i)    # dodajemy liczbę nieparzystą do nowej listy
    return nowa_lista
l1 = losuj_do_listy()
print (l1)
print (usun_parzyste(l1))
```

Program (spr_podzielności.py)

```
# sprawdzenie podzielności danej liczby
def reszta_z_dzielenia (liczba, dzielnik):
    return liczba % dzielnik
a = int(input('Podaj liczbe: '))
b = int(input('Podaj podzielnik: '))
if b != 0:
```

```
    if reszta_z_dzielenia (a, b) == 0:
        print('Liczba ', a, 'jest podzielna przez ', b)
    else:
        print('Liczba ', a, 'nie jest podzielna przez ', b)
else:
    print('Podzielnik nie może być = 0')
```

SCOPE (zasięg)

Wyróżniamy:

- local scope – zasięg lokalny, który ogranicza zasięg zmiennych do obszaru wewnątrz funkcji, instrukcji warunkowych czy pętli - czyli jeśli zadeklarujemy zmienną w funkcji, instrukcji warunkowej czy pętli, to tylko i wyłącznie tam będzie dostępna
- global scope – zasięg globalny, w którym każda funkcja, instrukcja warunkowa, pętla itd. ma dostęp do tej zmiennej. Domyślnie zmienne globalne są dostępne w trybie „tylko do odczytu”. By umożliwić modyfikację zmiennej musimy zadeklarować ją z użyciem polecenia global.

Program (zasieg_zmiennych.py)

```
# scope-zasięg zmiennych
miesiac = 'maj'          # zmienna globalna-dostępna w całym skrypcie
def lubie():
    print('Lubie miesiąc: ', miesiac)
def bardzo_lubie():
    print('Bardzo lubię: ', miesiac)
lubie()
bardzo_lubie()
```

Nie mamy dostępu do zmiennych lokalnych poza scopem, w którym zostały zadeklarowane.

Program (zasieg_zmiennych2.py)

```
def lubie():
    miesiac = 'czerwiec'
    print('Lubie miesiąc: ', miesiac)
lubie()
print(miesiac)
```

Po uruchomieniu programu ostatnie polecenie skryptu - *print(miesiac)* - spowoduje wygenerowanie komunikatu o błędzie: NameError: name 'miesiac' is not defined

Zmienna lokalna, nie ma nic wspólnego ze zmienną globalną, mimo że może mieć taką samą nazwę.

Program (zasieg_zmiennych3.py)

```
# scope-zasięg zmiennych
miesiac = 'maj'          # zmienna globalna-dostępna w całym skrypcie
def lubie():
   miesiac = 'czerwiec'   # zm lokalna-mimo tej samej nazwy zmienne są różne
   print('Lubie miesiąc: ', miesiac)
def bardzo_lubie():
   print('Bardzo lubię: ', miesiac)
lubie()
bardzo_lubie()
```

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań”
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020

Moduły (3h)

W **Pythonie moduły** są po prostu plikami z rozszerzeniem py , w których zawarto pewien zestaw funkcji, zmiennych. **Moduły** importujemy do swojego programu za pomocą komendy import .

Moduły mogą importować inne moduły. Najczęściej wszystkie instrukcje importowania modułów umieszczane są na początku modułu lub skryptu.

Moduł własny

Oprócz korzystania z gotowych modułów możemy tworzyć własne. Najprościej jest stworzyć plik z rozszerzeniem .py, który będzie zawierał funkcje i zmienne.

Moduł 1 (plik moj_modul1.py)

```
# własny moduł 1 - plik moj_modul1.py
def zlicz(tekst):
    ile_wyrazow = 1
    for i in tekst:
        if i == ' ':
            ile_wyrazow += 1
    return ile_wyrazow
```

Program (program1_moj_modul1.py)

```
# program1 z importem moj_modul1.py
import moj_modul1
zdanie = input("Napisz zdanie: ")
print ('Liczba wyrazów = ', moj_modul1.zlicz(zdanie))
```

Z funkcji zawartej w naszym moj_module1 możemy skorzystać w dowolnie innym programie.

Projekt „ENIGMA - Wsparcie nauczania matematyki i informatyki w szkołach podstawowych i ponadpodstawowych Metropolii Poznań”
współfinansowany przez Unię Europejską z Europejskiego Funduszu Społecznego
w ramach Wielkopolskiego Regionalnego Programu Operacyjnego na lata 2014-2020