

Programowanie Python 1

(CP1S02005)

Politechnika Białostocka - Wydział Elektryczny
Cyfryzacja przemysłu, sem. II, studia stacjonarne I stopnia
Rok akademicki 2024/2025

Wykład nr 3 (01.04.2025)

dr inż. Jarosław Forenc

Plan wykładu nr 3

- Słownik
 - implementacja
 - metody tworzenia
 - operacje
- Zbiór
 - implementacja
 - metody tworzenia
 - operacje
- Kolokwium nr 1
 - rozwiązanie przykładowego zadania

Python - słownik

- **Słownik** (ang. dictionary) - kolekcja par **klucz-wartość**, gdzie każdy klucz jest unikalny i przypisany jest do określonej wartości

```
rysopis = {"płeć" : "K", "wzrost" : 170, "oczy" : "szare"}  
komputer = {"procesor" : "AMD", "dysk" : "SSD"}
```

- elementy **słownika** umieszcza się w nawiasach klamrowych
- połączenie klucza z wartością odbywa się za pomocą dwukropka, a poszczególne pary klucz-wartość są rozdzielone przecinkami
- za pomocą klucza można uzyskać dostęp do powiązanej z nim wartości
- wartością klucza może być liczba, ciąg tekstowy, lista lub inny słownik (dowolny obiekt możliwy do utworzenia w Pythonie)
- w **słowniku** można przechowywać dowolną liczbę par klucz-wartość

Python - słownik, metody tworzenia

- użycie pary nawiasów klamrowych do utworzenia pustego słownika

```
mój_słownik = {}  
print(mój_słownik)
```

```
{}
```

- pusty słownik tworzymy wtedy, kiedy chcemy przechowywać dane wprowadzane przez użytkownika lub kiedy kod programu automatycznie generuje pary klucz-wartość
- pusty słownik można utworzyć stosując konstruktor bezargumentowy: `dict()`

```
mój_słownik = dict()
```

Python - słownik, metody tworzenia

- użycie nawiasów klamrowych, oddzielenie par klucz-wartość przecinkami, połączenie klucza z wartością za pomocą dwukropka

```
komputer = {"procesor" : "Intel", "dysk" : "HDD"}  
litery = {"A" : 1, "B" : 2, "C" : 0}  
cyfry = {0 : 1, 1 : 1, 2 : 0}  
print(komputer)  
print(litery)  
print(cyfry)
```

```
{'procesor': 'Intel', 'dysk': 'HDD'}  
{'A': 1, 'B': 2, 'C': 0}  
{0: 1, 1: 1, 2: 0}
```

Python - słownik, metody tworzenia

- definicję słownika można umieścić w kilku wierszach kodu

```
komputer = {  
    "procesor" : "Intel",  
    "dysk" : "HDD",  
    "klawiatura" : "A4Tech",  
    "mysz" : "Logitech",  
}
```

```
{'procesor': 'Intel', 'dysk': 'HDD', 'klawiatura':  
'A4Tech', 'mysz': 'Logitech'}
```

- dobrą praktyką jest umieszczanie przecinka po ostatniej wartości
- zapewnia to jednolitość składniową i czytelności kodu

Python - słownik, unikalność kluczy

- w słowniku każdy klucz musi być **unikalny**
- jeśli nadamy wartość istniejącemu kluczowi, to wartość ta zastąpi poprzednią

```
wiek = {"Jan" : 21, "Ela" : 19, "Jan" : 18, "Ola" : 23}  
print(wiek)
```

```
{'Jan': 18, 'Ela': 19, 'Ola': 23}
```

```
litery = {"A" : 1, "B" : 1, "B" : 2, "A" : 2}  
print(litery)
```

```
{'A': 2, 'B': 2}
```

Python - słownik, metody tworzenia

- ❑ zastosowanie **słownika składanego** (ang. dict comprehension)
- ❑ oznacza wykorzystanie składni umożliwiającej tworzenie nowego słownika poprzez iterację i zastosowanie warunków do istniejących danych
- ❑ ogólna struktura słownika składanego

```
{key_expr: value_expr for item in iterable if condition}
```

- ❑ **key_expr** - wyrażenie określające klucz dla każdej pary klucz-wartość
- ❑ **value_expr** - wyrażenie określające wartość dla każdego klucza
- ❑ **item** - zmienna używana do iteracji po elemencie w iterowalnym obiekcie
- ❑ **iterable** - obiekt, po którym można iterować (np. lista, krotka)
- ❑ **condition** - opcjonalny warunek, który musi być spełniony, aby dodać element do słownika

Python - słownik, metody tworzenia

- przykład: stworzenie **słownika** na podstawie **listy**

```
liczby = [1, 2, 3, 4, 5]
kwadraty = {nr: nr**2 for nr in liczby}
print(kwadraty)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

- przykład: stworzenie **słownika** na podstawie **krotki**

```
owoce = ('jabłko', 'banan', 'kiwi')
długości = {owoc: len(owoc) for owoc in owoce}
print(długości)
```

```
{'jabłko': 6, 'banan': 5, 'kiwi': 4}
```

Python - słownik, metody tworzenia

- przykład: stworzenie **słownika** na podstawie **listy** z zastosowaniem warunku

```
liczby = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
parzyste = {nr: nr**2 for nr in liczby if nr % 2 == 0}
print(parzyste)
```

```
2: 4, 4: 16, 6: 36, 8: 64, 10: 100}
```

Python - słownik, dostęp do wartości

- do **pobrania wartości** powiązanej z kluczem należy podać nazwę słownika oraz, w nawiasach kwadratowych, nazwę klucza

```
nazwa_słownika[nazwa_klucza]
```

- przykład:

```
komputer = {"procesor" : "AMD", "dysk" : "SSD"}  
print(f"Procesor to: {komputer["procesor"]}")  
print(f"Dysk to:      {komputer["dysk"]}")
```

```
Procesor to: AMD  
Dysk to:      SSD
```

Python - słownik, dostęp do wartości

- podanie błędnej nazwy klucza spowoduje wyświetlenie błędu kompilacji

```
komputer = {"procesor" : "AMD", "dysk" : "SSD"}  
print(f"Procesor to: {komputer["Procesor"]}")  
print(f"Dysk to:      {komputer["dysk"]}")
```

```
Traceback (most recent call last):  
  File "d:\MyApp.py", line 2, in <module>  
    print(f"Procesor to: {komputer["Procesor"]}")  
                                ~~~~~~^~^~^~^~^~^~^~^~^  
KeyError: 'Procesor'
```

Python - słownik, operacje

- słownik jest strukturą dynamiczną, można w każdej chwili **dodawać** i **usuwać** pary klucz-wartość
- **dodanie** nowej pary klucz-wartość polega na podaniu nazwy słownika, nazwy nowego klucza (w nawiasach kwadratowych) oraz wartości przypisywanej do danego klucza

```
komputer = {"procesor" : "AMD"}  
print(komputer)  
komputer["dysk"] = "HDD"  
komputer["mysz"] = "A4Tech"  
print(komputer)
```

```
{'procesor': 'AMD'}  
{'procesor': 'AMD', 'dysk': 'HDD', 'mysz': 'A4Tech'}
```

- słownik zachowuje kolejność, w której były dodawane pary klucz-wartość

Python - słownik, operacje

- jeśli klucz o podanej nazwie istnieje, to jego wartość jest **aktualizowana**

```
komputer = {"procesor" : "AMD", "dysk" : "HDD"}  
komputer["dysk"] = "SSD"  
print(komputer)
```

```
{'procesor': 'AMD', 'dysk': 'SSD'}
```

- do **dodawania** nowych par klucz-wartość lub **aktualizacji** istniejących wartości można zastosować metodę **update()**

```
komputer = {"procesor" : "AMD"}  
komputer.update({"dysk" : "SSD", "procesor" : "Intel"})  
print(komputer)
```

```
{'procesor': 'Intel', 'dysk': 'SSD'}
```

Python - słownik, operacje

- metoda `setdefault(key, default_value)` dodaje nowy klucz ze wskazaną wartością domyślną, jeśli klucz nie istnieje w słowniku

```
komputer = {"procesor" : "AMD"}  
print(komputer)  
komputer.setdefault("dysk", "SSD")  
print(komputer)  
komputer.setdefault("procesor", "Intel")  
print(komputer)
```

```
{'procesor': 'AMD'}  
{'procesor': 'AMD', 'dysk': 'SSD'}  
{'procesor': 'AMD', 'dysk': 'SSD'}
```

Python - słownik, operacje

- do **usunięcia** konkretnego klucza ze słownika służy funkcja **del**

```
komputer = {"procesor" : "AMD", "dysk" : "HDD"}  
del komputer["procesor"]  
print(komputer)
```

```
{ 'dysk' : 'HDD' }
```

- do **usunięcia** klucza i zwrócenia odpowiadającej mu wartości można zastosować metodę **pop()**

```
komputer = {"procesor" : "AMD", "dysk" : "HDD"}  
wynik = komputer.pop("dysk")  
print(komputer)  
print(wynik)
```

```
{ 'procesor' : 'AMD' }  
HDD
```


Python - słownik, operacje

- w metodzie `pop()` można podać wartość domyślną jako drugi argument
- jeśli klucz nie istnieje, to ta wartość zostanie zwrócona

```
komputer = {"procesor" : "AMD", "dysk" : "HDD"}
wynik = komputer.pop("dysk","Klucz nie istnieje")
print(komputer)
print(wynik)

wynik = komputer.pop("mysz","Klucz nie istnieje")
print(komputer)
print(wynik)
```

```
{'procesor': 'AMD'}
HDD
{'procesor': 'AMD'}
Klucz nie istnieje
```

Python - słownik, operacje

- do **usunięcia** ostatniej pary klucz-wartość ze słownika służy metoda **popitem()**
- metoda ta usuwa i zwraca ostatnią parę klucz-wartość jako krotkę

```
komputer = {"procesor" : "AMD", "dysk" : "HDD"}  
klucz, wartość = komputer.popitem()  
print(komputer)  
print(klucz, wartość)
```

```
{'procesor': 'AMD'}  
dysk HDD
```

- każda operacja usunięcia pary klucz-wartość jest nieodwracalna

Python - słownik, operacje

- do **pobrania** wartości z słownika na podstawie podanego klucza można zastosować metodę **get()**

```
komputer = {"procesor" : "AMD", "dysk" : "HDD"}  
wartość = komputer.get("dysk")  
print(wartość)
```

HDD

- jeśli klucz nie istnieje, to metoda **get()** zwraca wartość domyślną (**None**)

```
komputer = {"procesor" : "AMD", "dysk" : "HDD"}  
wartość = komputer.get("mysz")  
print(wartość)
```

None

Python - słownik, operacje

- sprawdzenie wartości zapisanych w słowniku - operator porównania (==)

```
komputer = {"procesor" : "AMD", "dysk" : "HDD"}  
if komputer["procesor"] == "AMD":  
    print("Masz procesor AMD")  
if komputer["procesor"] == "Intel":  
    print("Masz procesor Intel")
```

```
Masz procesor AMD
```

Python - słownik, operacje

- iteracja przez wszystkie pary klucz-wartość - stosujemy pętlę `for` i metodę `items()`

```
komputer = {  
    "procesor" : "Intel",  
    "dysk" : "HDD",  
    "mysz" : "Logitech",  
}  
  
for key, value in komputer.items():  
    print(f"Klucz: {key}, wartość: {value}")
```

```
Klucz: procesor, wartość: Intel  
Klucz: dysk, wartość: HDD  
Klucz: mysz, wartość: Logitech
```

- iteracja taka zwraca listę par klucz-wartość w kolejności ich wstawiania do słownika

Python - słownik, operacje

- metod `keys()` pozwala otrzymać widok (ang. view) zawierający **klucze**

```
komputer = {  
    "procesor" : "Intel",  
    "dysk" : "HDD",  
    "mysz" : "Logitech",  
}  
  
for name in komputer.keys():  
    print(name)
```

```
procesor  
dysk  
mysz
```

- to samo otrzymamy pomijając `keys()`, ale kod jest wtedy mniej oczywisty

```
for name in komputer:  
    print(name)
```

Python - słownik, operacje

- metod `values()` pozwala otrzymać widok (ang. view) zawierający wartości

```
komputer = {  
    "procesor" : "Intel",  
    "dysk" : "HDD",  
    "mysz" : "Logitech",  
}  
  
for name in komputer.values():  
    print(name)
```

```
Intel  
HDD  
Logitech
```

- widok można zamienić na listę

```
lista = list(komputer.values())  
print(lista)
```

Python - słownik, operacje

- klucze lub wartości słownika mogą być przy iteracji posortowane

```
komputer = {  
    "procesor" : "Intel",  
    "dysk" : "HDD",  
    "mysz" : "Logitech",  
}  
  
for name in sorted(komputer.values()):  
    print(name)
```

```
HDD  
Intel  
Logitech
```

- aby wyświetlane wartości nie powtarzały się należy zastosować **zbiór**

```
for name in set(komputer.values()):  
    print(name)
```


Python - słownik, operacje

- zagnieżdżanie - słownik zawierający listy

```
programowanie = {  
    "Jan" : ["C", "Python"],  
    "Karol" : ["C", "C++", "Python"],  
    "Piotr" : ["Python"],  
}  
  
for imię, języki in programowanie.items():  
    if len(języki) == 1:  
        print(f"{imię} zna język:")  
    else:  
        print(f"{imię} zna języki:")  
        for język in języki:  
            print(f"{język}", end = ", ")  
        print("")
```

```
Jan zna języki:  
C, Python,  
Karol zna języki:  
C, C++, Python,  
Piotr zna język:  
Python,
```

Python - słownik, operacje

- zagnieżdżanie - lista zawierająca słowniki

```
PC1 = {"procesor" : "AMD", "dysk" : "SSD"}
PC2 = {"procesor" : "Intel", "dysk" : "HDD"}
PC3 = {"procesor" : "Intel", "dysk" : "SSD"}

komputery = [PC1, PC2, PC3]

for PC in komputery:
    print(PC)
```

```
{'procesor': 'AMD', 'dysk': 'SSD'}
{'procesor': 'Intel', 'dysk': 'HDD'}
{'procesor': 'Intel', 'dysk': 'SSD'}
```

- wszystkie słowniki na liście powinny mieć identyczną strukturę, tak aby można było przeprowadzić iterację listy

Python - słownik, operacje

- operacje na słowniku i liście

```
cars = {  
    "Jan" : "Opel",  
    "Piotr" : "BMW",  
    "Olek" : "Audi",  
    "Bartek" : "BMW",  
}  
koledzy = ["Olek", "Paweł", "Piotr", "Marian"]  
  
for name in koledzy:  
    if name not in cars.keys():  
        print(f"{name} - nie ma samochodu")  
    if name in cars.keys():  
        print(f"{name} ma samochód: {cars[name]}")
```

```
Piotr ma samochód: BMW  
Marian - nie ma samochodu  
Olek ma samochód: Audi  
Paweł - nie ma samochodu
```

Python - słownik, wczytywanie danych

- wczytywanie danych z klawiatury bezpośrednio do słownika

```
kierowcy = {}
wczytaj = True

while wczytaj:
    imię = input("Podaj imię kierowcy: ")
    marka = input("Podaj markę samochodu: ")
    kierowcy[imię] = marka

    odp = input("Dalej? (tak/nie): ")
    if odp == "nie":
        wczytaj = False

print("Dane: ")
for imię, marka in kierowcy.items():
    print(f"{imię} ma samochód {marka}")
```

Python - słownik, wczytywanie danych

- wczytywanie danych z klawiatury bezpośrednio do słownika

```
kierowcy = {}  
wczytuj = True  
  
while wczytuj:  
    imię = input("Podaj imię  
    marka = input("Podaj mar  
    kierowcy[imię] = marka  
  
    odp = input("Dalej? (tak  
    if odp == "nie":  
        wczytuj = False  
  
print("Dane: ")  
for imię, marka in kierowcy.  
    print(f"{imię} ma samoch
```

```
Podaj imię kierowcy: Jan  
Podaj markę samochodu: Fiat  
Dalej? (tak/nie): tak  
Podaj imię kierowcy: Anna  
Podaj markę samochodu: Skoda  
Dalej? (tak/nie): tak  
Podaj imię kierowcy: Piotr  
Podaj markę samochodu: BMW  
Dalej? (tak/nie): nie  
Dane:  
Jan ma samochód Fiat  
Anna ma samochód Skoda  
Piotr ma samochód BMW
```

Python - słownik, zagnieżdżanie

- zagnieżdżanie słownika w słowniku

```
student = {  
    "imię" : "Jan",  
    "nazwisko" : "Kowalski",  
    "oceny" : {  
        "Metrologia" : 4.5,  
        "Fizyka" : 3.0,  
        "Matematyka" : 4.0  
    }  
}  
  
print(f"Imię: {student["imię"]}")  
print(f"Nazwisko: {student["nazwisko"]}")  
  
print("Oceny:")  
for przedmiot, ocena in student["oceny"].items():  
    print(f"{przedmiot}: {ocena}")
```

```
Imię: Jan  
Nazwisko: Kowalski  
Oceny:  
Metrologia: 4.5  
Fizyka: 3.0  
Matematyka: 4.0
```

Python - zbiór

- **Zbiór** (ang. set) - kolekcja unikalnych elementów, które są nieuporządkowane
 - zbiory można tworzyć za pomocą nawiasów klamrowych `{}` lub przy użyciu wbudowanej funkcji `set()`
 - utworzenie pustego zbioru:

```
pusty1 = {}  
print(pusty1)
```

```
{}
```

```
pusty2 = set()  
print(pusty2)
```

```
set()
```

Python - zbiór, metody tworzenia

- utworzenie zbioru za pomocą nawiasów klamrowych

```
litory = {"A", "B", "C", "D", "E"}  
print(litory)
```

```
{'D', 'A', 'B', 'E', 'C'}
```

- utworzenie zbioru z listy elementów za pomocą funkcji `set()`

```
litory = set(["A", "B", "C", "C", "B"])  
print(litory)
```

```
{'A', 'C', 'B'}
```

- powtarzające się elementy zostaną automatycznie usunięte

Python - zbiór, operacje

- **dodanie** jednego elementu do zbioru - metoda **add()**

```
liczby = {1, 2, 3, 4}
liczby.add(0)
print(liczby)
```

```
{0, 1, 2, 3, 4}
```

- **dodanie** wielu elementów do zbioru - metoda **update()**

```
liczby = {1, 2, 3, 4}
liczby.update([0, 5, 6])
print(liczby)
```

```
{0, 1, 2, 3, 4, 5, 6}
```

Python - zbiór, operacje

- **usunięcie** elementu ze zbioru - metoda `remove()`

```
liczby = {1, 2, 3, 4}
liczby.remove(1)
print(liczby)
```

```
{2, 3, 4}
```

- jeśli element nie istnieje w zbiorze, `remove()` spowoduje wyjątek `KeyError`

```
liczby = {1, 2, 3, 4}
liczby.remove(0)
print(liczby)
```

```
Traceback (most recent call last):
  File "d:\MyApp.py", line 2, in <module>
    liczby.remove(0)
KeyError: 0
```

Python - zbiór, operacje

- **usunięcie** elementu ze zbioru - metoda `discard()`

```
liczby = {1, 2, 3, 4}
liczby.discard(1)
print(liczby)
```

```
{2, 3, 4}
```

- jeśli element nie istnieje w zbiorze, `discard()` nie zgłasza błędu

```
liczby = {1, 2, 3, 4}
liczby.discard(0)
print(liczby)
```

```
{1, 2, 3, 4}
```

Python - zbiór, operacje matematyczne

- **suma** (unia) dwóch zbiorów - metoda **union()**

```
liczby1 = {1, 2, 3, 4}
liczby2 = {3, 4, 5, 6}
suma = liczby1.union(liczby2)
print(suma)
```

```
{1, 2, 3, 4, 5, 6}
```

Python - zbiór, operacje matematyczne

- część wspólna (przecięcie) dwóch zbiorów - metoda `intersection()`

```
liczby1 = {1, 2, 3, 4}
liczby2 = {3, 4, 5, 6}
wspolne = liczby1.intersection(liczby2)
print(wspolne)
```

```
{3, 4}
```

Python - zbiór, operacje matematyczne

- różnica dwóch zbiorów - metoda `difference()`

```
liczby1 = {1, 2, 3, 4}
liczby2 = {3, 4, 5, 6}
różnica = liczby1.difference(liczby2)
print(różnica)
```

```
{1, 2}
```

Python - zbiór, operacje matematyczne

- różnica symetryczna dwóch zbiorów - metoda `symmetric_difference()`

```
liczby1 = {1, 2, 3, 4}
liczby2 = {3, 4, 5, 6}
różnica = liczby1.symmetric_difference(liczby2)
print(różnica)
```

```
{1, 2, 5, 6}
```

- metoda ta zwraca zbiór zawierający elementy, które są obecne tylko w jednym z dwóch zbiorów, ale nie w obu jednocześnie

Python - zbiór, operator in

- operator **in** służy do sprawdzenia czy element znajduje się w zbiorze

```
liczby = {1, 2, 3, 4, 5, 6}
nr = int(input("Podaj liczbe: "))

if nr in liczby:
    print(f"Element {nr} znajduje się w zbiorze")
else:
    print(f"Element {nr} nie znajduje się w zbiorze")
```

```
Podaj liczbe: 6
Element 6 znajduje się w zbiorze
```

```
Podaj liczbe: 0
Element 0 nie znajduje się w zbiorze
```


Python - zbiór, operator not in

- operator **not in** służy do sprawdzenia czy element nie znajduje się w zbiorze

```
liczby = {1, 2, 3, 4, 5, 6}
nr = int(input("Podaj liczbę: "))

if nr not in liczby:
    print(f"Element {nr} nie znajduje się w zbiorze")
else:
    print(f"Element {nr} znajduje się w zbiorze")
```

```
Podaj liczbę: 6
Element 6 znajduje się w zbiorze
```

```
Podaj liczbę: 0
Element 0 nie znajduje się w zbiorze
```

Python - kolokwium (zadanie 1, wersja 1)

- Umieść imię i nazwisko, nr grupy, nr albumu, datę, kod przedmiotu (CP1S02005), formę zajęć (Pracownia specjalistyczna) i nazwę środowiska programistycznego (Visual Studio Code) na początku kodu źródłowego programu (jako komentarz). [5 pkt.]

```
# Autor: Jan Kowalski, gr. PS1  
# Album: 123456  
# Data: 01-04-2025  
# Kod: CP1S02005  
# Forma: Pracownia specjalistyczna  
# IDE: Visual Studio Code
```

Python - kolokwium (zadanie 1, wersja 2)

- Umieść imię i nazwisko, nr grupy, nr albumu, datę, kod przedmiotu (CP1S02005), formę zajęć (Pracownia specjalistyczna) i nazwę środowiska programistycznego (Visual Studio Code) na początku kodu źródłowego programu (jako komentarz). [5 pkt.]

```
"""  
Autor: Jan Kowalski, gr. PS1  
Album: 123456  
Data: 01-04-2025  
Kod: CP1S02005  
Forma: Pracownia specjalistyczna  
IDE: Visual Studio Code  
"""
```

- zamiast trzech cudzysłowów można zastosować trzy apostrofy

Python - kolokwium (zadanie 2, wersja 1)

- Oblicz obwód elipsy L o parametrach c i d . W przypadku liczby π użyj odpowiedniej stałej. Wartości parametrów c i d wczytaj z klawiatury. Wyświetl wynik z dokładnością do 3 cyfr po kropce. [15 pkt.]

$$L = \pi(1,5(c + d) - \sqrt{cd})$$

```
import math
c = float(input("Podaj wartość parametru c: "))
d = float(input("Podaj wartość parametru d: "))
L = math.pi * (1.5 * (c + d) - math.sqrt(c * d))
print(f"Obwód elipsy wynosi: {L:.3f}")
```

```
Podaj wartość parametru c: 2.5
Podaj wartość parametru d: 5
Obwód elipsy wynosi: 24.236
```

Python - kolokwium (zadanie 2, wersja 2)

- Oblicz obwód elipsy L o parametrach c i d . W przypadku liczby π użyj odpowiedniej stałej. Wartości parametrów c i d wczytaj z klawiatury. Wyświetl wynik z dokładnością do 3 cyfr po kropce. [15 pkt.]

$$L = \pi(1,5(c + d) - \sqrt{cd})$$

```
import math
try:
    c = float(input("Podaj wartość parametru c: "))
    d = float(input("Podaj wartość parametru d: "))
except ValueError:
    print("Błędne dane wejściowe!")
else:
    L = math.pi * (1.5 * (c + d) - math.sqrt(c * d))
    print(f"Obwód elipsy wynosi: {L:.3f}")
```

Python - kolokwium (zadanie 3)

- Koszt ubezpieczenia samochodu stanowi określony procent jego wartości i jest zależny od wieku kierowcy. Kierowcy dzieleni są na cztery grupy (tabela).

Grupa	Wiek	Procent
0	od 18 do 25 lat	8,25%
1	powyżej 25 do 35 lat	7,25%
2	powyżej 35 do 45 lat	6,25%
3	powyżej 45 lat do 55 lat	6,00%
4	powyżej 55 lat	6,50%

- Wczytaj **wartość samochodu** oraz **wiek kierowcy**. Na podstawie wieku kierowcy, określ i wyświetl **grupe**, do której zostanie on zaliczony oraz oblicz i wyświetl **koszt** ubezpieczenia samochodu. Koszt wyświetl z dokładnością do 2 cyfr po kropce dziesiętnej, dodaj symbol waluty. **[15 pkt.]**

Python - kolokwium (zadanie 3)

```
wartość = float(input("Podaj wartość samochodu (w PLN): "))
wiek = int(input("Podaj wiek kierowcy: "))

if 18 <= wiek <= 25:
    grupa = 0
    koszt = 8.25 / 100 * wartość
elif 26 <= wiek <= 35:
    grupa = 1
    koszt = 7.25 / 100 * wartość
elif 36 <= wiek <= 45:
    grupa = 2
    koszt = 6.25 / 100 * wartość
elif 46 <= wiek <= 55:
    grupa = 3
    koszt = 6.00 / 100 * wartość
else:
    grupa = 4
    koszt = 6.50 / 100 * wartość

print(f"Grupa: {grupa}")
print(f"Koszt ubezpieczenia: {koszt:.2f} PLN")
```

Python - kolokwium (zadanie 3)

```
wartość = float(input("Podaj wartość samochodu (w PLN): "))
wiek = int(input("Podaj wiek kierowcy: "))
if 18 <= wiek <= 25:
    grupa = 0
    koszt = 8.25 / 100 * wartość
elif 26 <= wiek <= 35:
    grupa = 1
    koszt = 7.25 / 100 * wartość
elif 36 <= wiek <= 45:
    grupa = 2
    koszt = 6.25 / 100 * wartość
elif 46 <= wiek <= 55:
    grupa = 3
    koszt = 6.00 / 100 * wartość
else:
```

```
Podaj wartość samochodu (w PLN): 35000
Podaj wiek kierowcy: 23
Grupa: 0
Koszt ubezpieczenia: 2887.50 PLN
```


Python - kolokwium (zadanie 4)

- Utwórz listę zawierającą n liczb pseudolosowych z zakresu od 1 do 50. Wyświetl zawartość listy na ekranie. Wartość n wczytaj z klawiatury. [6 pkt.]
- Oblicz i wyświetl **średnią arytmetyczną** wszystkich liczb znajdujących się na liście. [4 pkt.]
- Utwórz dwie nowe listy. Do pierwszej skopiuj liczby **mniejsze lub równe** średniej, zaś do drugiej liczby **większe** od średniej. Wyświetl zawartość obu list na ekranie. [10 pkt.]

Suma punktów: [20 pkt.]

Python - kolokwium (zadanie 4)

```
import random

n = int(input("Podaj liczbę n: "))

liczby = [random.randint(1, 50) for _ in range(n)]

print("Liczby pseudolosowe:")
print(liczby)

srednia = sum(liczby) / n
print(f"Średnia arytmetyczna: {srednia:.2f}")

mniejsze = [nr for nr in liczby if nr <= srednia]
wieksze = [nr for nr in liczby if nr > srednia]

print("Liczby mniejsze lub równe średniej:")
print(mniejsze)

print("Liczby większe od średniej:")
print(wieksze)
```

Python - kolokwium (zadanie 4)

```
import random

n = int(input("Podaj liczbę n: "))

liczby = [random.randint(1, 50) for _ in range(n)]

print("Liczby pseudolosowe:")
print(liczby)

srednia = sum(liczby) / n
print(f"Średnia arytmetyczna: {srednia:.2f}")

mniejsze = [nr for nr in liczby if nr <= srednia]
```

```
Podaj liczbę n: 10
Liczby pseudolosowe:
[26, 46, 13, 18, 42, 28, 37, 3, 33, 17]
Średnia arytmetyczna: 26.30
Liczby mniejsze lub równe średniej:
[26, 13, 18, 3, 17]
Liczby większe od średniej:
[46, 42, 28, 37, 33]
```

Python - kolokwium (zadanie 5)

- Grupa osób odwiedziła w wakacje kilka miast w Europie. Zdefiniuj słownik, w którym klucze zawierają imiona osób, zaś wartości to lista miast, które odwiedziły te osoby. Jedna osoba mogła odwiedzić kilka miast. [4 pkt.]
- Zdefiniuj drugi słownik, w którym klucze zawierają nazwy miast, zaś wartości to odległość danego miasta od Białegostoku. Wykorzystaj poniższe miasta i odległości: Berlin - 659 km, Paryż - 1528 km, Londyn - 1588 km, Rzym - 1482 km, Praga - 692 km, Oslo - 1070 km. [4 pkt.]
- Dla każdej osoby wyświetl w jednym wierszu: imię osoby, odwiedzone miasta, sumę odległości z Białegostoku do tych miast. [10 pkt.]
- Wyświetl na ekranie imię osoby, dla której suma odległości jest największa. [7 pkt.]

Suma punktów: [25 pkt.]

Python - kolokwium (zadanie 5)

```
wyjazdy = {  
    "Kamil": ["Berlin", "Paryż", "Rzym"],  
    "Anna": ["Londyn", "Praga"],  
    "Ola": ["Oslo", "Berlin"],  
    "Szymon": ["Praga", "Rzym", "Berlin", "Londyn"],  
}  
odleglosci = {  
    "Berlin": 659,  
    "Paryż": 1528,  
    "Londyn": 1588,  
    "Rzym": 1482,  
    "Praga": 692,  
    "Oslo": 1070  
}
```

Python - kolokwium (zadanie 5)

```
max_osoba = ""
max_odleglosc = 0

for osoba, miasta in wyjazdy.items():
    tekst = osoba + ": "
    suma = 0

    for miasto in miasta:
        tekst += (miasto + ", ")
        suma += odleglosci[miasto]

    tekst += ("odległość: " + str(suma))
    print(tekst)

    if suma > max_odleglosc:
        max_odleglosc = suma
        max_osoba = osoba

print(f"Największa suma odległości: {max_osoba}")
```

Python - kolokwium (zadanie 5)

```
max_osoba = ""
max_odleglosc = 0

for osoba, miasta in wyjazdy.items():
    tekst = osoba + ": "
    suma = 0

    for miasto in miasta:
        tekst += (miasto + ", ")
        suma += odleglosci[miasto]

    tekst += ("odległość: " + str(suma))
    print(tekst)

    if suma > max_odleglosc:
```

```
Kamil: Berlin, Paryż, Rzym, odległość: 3669
Anna: Londyn, Praga, odległość: 2280
Ola: Oslo, Berlin, odległość: 1729
Szymon: Praga, Rzym, Berlin, Londyn, odległość: 4421
Największa suma odległości: Szymon
```

Python - kolokwium (zadanie 6)

- Trzy listy zawierają imiona osób, które brały udział w trzech rundach zawodów sportowych. Te same osoby mogą występować na kilku listach. [6 pkt.]
- Wyświetl na ekranie alfabetyczną listę unikalnych osób, które wzięły udział przynajmniej w jednej rundzie zawodów. Imiona nie mogą się powtarzać. [14 pkt.]

Suma punktów: [20 pkt.]

Python - kolokwium (zadanie 6, wersja 1)

```
zawody1 = ["Anna", "Piotr", "Jan", "Ela", "Marcin"]
zawody2 = ["Jan", "Ola", "Paweł", "Ela"]
zawody3 = ["Marcin", "Jan", "Kamila", "Mariusz"]

wszyscy = set()

wszyscy.update(zawody1)
wszyscy.update(zawody2)
wszyscy.update(zawody3)

wszyscy = sorted(wszyscy)

print("Lista uczestników:")
for osoba in wszyscy:
    print(osoba)
```

```
Lista uczestników:
Anna
Ela
Jan
Kamila
Marcin
Mariusz
Ola
Paweł
Piotr
```

Python - kolokwium (zadanie 6, wersja 2)

```
zawody1 = ["Anna", "Piotr", "Jan", "Ela", "Marcin"]
zawody2 = ["Jan", "Ola", "Paweł", "Ela"]
zawody3 = ["Marcin", "Jan", "Kamila", "Mariusz"]

wszyscy = sorted(set(zawody1 + zawody2 + zawody3))

print("Lista uczestników:")
for osoba in wszyscy:
    print(osoba)
```

```
Lista uczestników:
Anna
Ela
Jan
Kamila
Marcin
Mariusz
Ola
Paweł
Piotr
```

Koniec wykładu nr 3

Dziękuję za uwagę!