

Laborki – Python – PPY – PJATK 2025

Zajęcia 3

16.03.2025

1 Listy

- Lista (list):

```
x = ["Toyota", "Ford", "Mercedes"]
```

- Przykładowe listy:

```
x = [] # pusta lista
x = [1] # lista z jednym elementem typu int
x = [1, 2, 6] # lista z trzema elementami typu int
x = ['a', 'b', 'c'] # lista z trzema elementami typu str
x = ["Imię", "Nazwisko"] # lista z dwoma elementami typu str
x = [1, '2_88', [3]] # lista z elementami typu int, str oraz list
x = list("sadsfafsdf") # konwersja łańcucha znaków "dadasfgsfakj" na listę
```

- Indeksowanie list:

```
y = x[0] # wybór elementu o podanym indeksie od lewej do prawej (numerowanie od 0,
więc wybieramy pierwszy element z listy)
y = x[-1] # wybór elementu o podanym indeksie od prawej do lewej (numerowanie od 1,
więc wybieramy ostatni element z listy)
y = x[0:] # wybór wszystkich elementów od podanego indeksu
y = x[:3] # wybór wszystkich elementów do podanego indeksu
y = x[0:2] # wybór elementów z zakresu [start:stop]
y = x[0:-1:2] # wybór elementów z zakresu z krokiem [start:stop:krok]
y = x[::-1] # odwrócenie listy
x = [1, 2, 3, [4, 5, 6]]
y = x[3][0] #wybór pierwszego elementu z zagnieżdżonej listy (czyli czwórki)
```

W przeciwieństwie do min. Javy, Python nie posiada implementacji konceptu list, które przechowują dane określonego typu.

Jeżeli chcemy stworzyć listę, która przechowuje dane określonego typu, to musimy przykładowo stworzyć taki kod:

```
x: list[int] = [1, 2, 3]
```

Powyższy kod zakłada zdefiniowanie typu zmiennych w liście, gdy wpisujemy jednak:

```
x: list[str] = [1, 2, 3]
```

to otrzymamy komunikat, że nie jest to oczekiwany typ zmiennych, gdyż w liście znajdują się same "int".

Listy są "obiektami" modyfikowalnymi, więc można zmieniać ich zawartość, bez konieczności tworzenia nowych. Mogą one przechowywać obiekty dowolnego typu.

- Operacje na listach:

```
x = [1, 2, 3]
x[0] = 5 # zmiana elementu o indeksie 0 na wartość 5 typu int
y = x + x # konkatencja list
y = x * 3 # powtarzanie list przez podaną wartość
h = ["PJATK!"] * 4 # j.w., ale lista z danymi typu 'str'
z = [1] + [2, 3] # łączenie list
z.insert(0, 9) # wstawianie liczby 9 na pozycję pierwszą (0), można też wstawiać dane
innego typu
z.insert(len(z), 9) # wstawianie liczby 9 na ostatnią pozycję listy
y = x.append(9) # wstawianie nowych elementów do listy na jej koniec
x.remove("2") # usuwanie elementu "2" z listy
x.pop() # lista jako stos - usuwa ostatni element z listy
```

- Dalsze modyfikacje list:

- Łączenie dwóch list w jedną:

```
l1 = [1, 2, 3]
l2 = l1 + [4, 5, 6]
print(l2)
```

- Sortowanie list:

```
l3 = [5, 3, 7, 9, 3]
l3.sort()
print(l3)
```

```
l3.reverse()
print(l3)
```

- Dodawanie elementów na koniec listy:

```
l3.append(6)
print(l3)
```

- Dodawanie elementów w określonych indeksach listy:

```
l3.insert(5, 10)
print(l3)
```

- Usuwanie elementów z końca listy:

```
l3.pop()
print(l3)
```

- Usuwanie określonych elementów listy:

```
l3.remove(3)
print(l3)
```

- Usuwanie elementu znajdującego się pod konkretnym indeksem:

```
del l3[4]
print(l3)
```

- Usuwanie elementów z listy (wycinki):

```
del l3[1:3]
print(l3)
```

- Wyznaczanie najczęstszego elementu w liście (dominanty):

```
x = [4,4,2,3,5,6,7,8,8,4,7,5,4,3,2,4]
naj = max(set(x), key = x.count)
print(naj)
```

Uwaga! Powyższe działa prawidłowo, gdy najczęstszy element jest **jeden**, lecz, gdy np. dwa elementy listy będą powtarzały się równie często – skrypt wydrukuje na ekranie tylko jeden z nich.

- Listy wielowymiarowe (macierze):

- Tworzenie listy wielowymiarowej:

```
macierz = [1, [1, 2, 3, 4]]
print(macierz)
```

```
print(macierz[1][2])
```

- Modyfikowanie elementów listy (macierz):

```
macierz[1][3] = 6
print(macierz)
```

2 Krotki

Cechy krotek:

- Krotki definiujemy w identyczny sposób jak listę, lecz z jednym wyjątkiem – zbiór elementów jest ograniczony w nawiasach okrągłych, zamiast w kwadratowych.
- Podobnie jak w listach, elementy w krotce mają określony porządek.
- Krotki realizowane są szybciej od list.
- Krotki zajmują mniej miejsca.
- Indeksacja przebiega identycznie jak w przypadku list.
- Operacje na krotkach identyczne jak na listach.
- W krotkach nie ma modyfikowalności elementów.
- Krotka (tuple):

```
x = ("Toyota", "Ford", "Mercedes")
```

- Przykłady krotek:

```
x = () # pusta krotka
x = 5, # krotka z jednym elementem typu int
x = (1,) # j.w.
x = 3, 4 # krotka z dwoma elementami typu int
x = (1, 2) # j.w.
x = 1, (2, 3) # krotka z jednym elementem typu int i zagnieżdżoną krotką (2, 3)
x = (1, (2, 3)) # j.w.
x = 1, 'a', "Apollo", 2.0, [1, 2, 3], (1, 2, 3) # krotka złożona
z elementów różnych typów
x = (1, 'a', "Smok", 2.0, [1, 2, 3], (1, 2, 3)) # j.w.
x = tuple("oasddffghhj") # konwersja łańcucha znaków na krotkę
```

- Metody używane na krotkach:

```
y = x.index('o') # jeżeli przekazany obiekt istnieje w krotce, to podaje jego indeks
y = x.count('i') # liczy wszystkie wystąpienia przekazanego obiektu w krotce
```

3 Zbiory

Zbiór z definicji jest nieuporządkowanym zbiorem elementów, które nie mogą się powtarzać. **Lista** to elementy, które mogą się powtarzać. Dodatkowo lista ma swój określony porządek.

- Przykłady zbiorów (set):

```

x = set() # pusty zbiór
x = set([1]) # zbiór z 1 elementem typu int
x = set((1,)) # zbiór z 1 elementem typu int
x = {1} # zbiór z 1 elementem typu int
x = set("abcd") # zbiór z 4 elementami typu str {'a', 'b', 'c', 'd'}
x = {1, 2, 3} # zbiór z 3 elementami typu int
x = {1.0, "Dwa", ('T', 'r', 'z', 'y')} # zbiór z 3, niemutowalnymi obiektami różnego typu
x = {"Toyota", "Ford", "Mercedes"}

```

- Pobieranie wartości:

```

element = x.pop() # pobieramy element i usuwamy go ze zbioru
element = next(iter(x)) # pobieramy element i nie usuwamy go ze zbioru

```

- Operacje na zbiorach (**logiczne**):

```

zbior1 = set([0, 1, 2, 3, 4, 5, 6])
zbior2 = set([3, 4, 5, 6, 7, 8, 9])

```

```

suma_zbiorow = zbior1.union(zbior2) # wylicza sumę (unię) zbiorów
suma_zbiorow = zbior1 | zbior2 # j.w.

```

```

iloczyn_zbiorow = zbior1.intersection(zbior2) # wylicza iloczyn (część wspólną) zbiorów
iloczyn_zbiorow = zbior1 & zbior2 # j.w.

```

```

roznica_zbiorow = zbior1.difference(zbior2) # wylicza różnicę między zbiorem 1 a zbiorem 2
roznica_zbiorow = zbior1 - zbior2 # j.w.

```

```

roznica_zbiorow = zbior2.difference(zbior1) # wylicza różnicę między zbiorem 2 a zbiorem 1
roznica_zbiorow = zbior2 - zbior1 # j.w.

```

```

roznica_symetryczna = zbior1.symmetric_difference(zbior2) # wylicza różnicę symetryczną zbiorów
roznica_symetryczna = zbior1 ^ zbior2 # j.w.

```

- Pozostałe operacje na zbiorach:

```

x = {1, 2, 4, 6, 7}
x.remove(2) # usuwa 2 ze zbioru, jeśli nie ma tego elementu w zbiorze, metoda zwróci błąd.
x.discard(2) # usuwa 2 ze zbioru (j.w.), ale metoda ta nie zgłosi błędu, jeżeli nie ma tej cyfry w zbiorze.
x.add(5) # dodaje 5 do zbioru.
x.clear() # usuwa wszystkie elementy ze zbioru.

```

4 Słowniki

- Najciekawsze "typy zmiennych".

- Nie przypominają ani krotek, ani list, ani zbiorów.
- Możemy użyć nawiasów klamrowych do "zamknięcia" kolekcji.
- Kolekcja par zmiennych i obiektów, na które te zmienne wskazują.
- Słowniki mają bardzo szerokie zastosowanie chociażby od takich jak książka telefoniczna, kalendarz urodzinowy, po wczytywanie z bazy danych i wyświetlanie.
- Klucze podobnie jak elementy zbioru NIE mogą się powtarzać.
- W celu utworzenia słownika możemy skorzystać z klasy "dict" dedykowanej słownikom.
- Przykłady słowników (dict – dictionary):

```
x = {} # pusty słownik.
x = dict() # j.w.
x = {"Nazwisko": "Kowalski", "Wiek": 28} # słownik z dwoma parami klucz-wartość
x = dict([("Nazwisko", "Kowalski"), ("Wiek", 28)]) # j.w.
x = dict(Nazwisko="Kowalski", Wiek=28) # j.w.

y = {"Marka": "Toyota", "Data produkcji": "09-05-2022"}

z = {
    "Imię": "Agata",
    "Nazwisko": "Nowak",
    "Wiek": 46
}
# przykład:
y = dict([
    ("Nazwisko", "Kowalski"),
    ("Zarobki", 1000.0)
])
print(y)
# przykład bez "białych znaków":
y1 = dict(Nazwisko="Kowalski", Zarobki=1000.0)
print(y1)
```

- Pobieranie wartości ze słownika:

```
nazwisko = x["Nazwisko"]
wiek = x.get("Wiek")

# jeżeli podany klucz nie istnieje, to zostanie zwrócony łańcuch "Ten klucz nie istnieje!"
stanowisko = x.get("Stanowisko", "Ten klucz nie istnieje!")
```

- Dodawanie par klucz-wartość:

```
x["Stanowisko"] = "Profesor uczelni" # dodanie nowej pary klucz-wartość
nowe_pary = {"Wzrost": 162.0, "Waga": 50.0}
x.update(nowe_pary) # dodawanie wielu par klucz-wartość na raz (do słownika 'x')
```

- Modyfikacja wartości:

```
x["Wiek"] = 30 # modyfikacja wartości w jednej z par klucz-wartość
aktualizacja_par = {"Stanowisko": "Kierowca rajdowy", "Wiek": 25}
x.update(aktualizacja_par) # aktualizacja wielu wartości na raz.
```

- Usuwanie par klucz-wartość:

```
pobrane_nazwisko = x.pop("Nazwisko") # usunięcie pary klucz-wartość z pobraniem wartości

# jeżeli podany klucz nie istnieje, to zostanie zwrócony łańcuch "Nie ma takiego klucza!"
nie_ma_takiego_klucza = x.pop("Dochód", "Nie ma takiego klucza!")
del x["Wiek"] # usunięcie pary klucz-wartość
x.clear() # usunięcie wszystkich par klucz-wartość
```

- Wybrane metody:

```
x = {1: 'a', 2: 'b', 3: 'c'}
klucze = x.keys() # pobranie kluczy
wartosci = x.values() # pobranie wartości
pary = x.items() # pobranie par klucz-wartość
```

4.1 Słowniki – przykładowe kody

Ze strony <https://pl.python.org/>:

```
#!/usr/bin/env python
# -*- coding: iso-8859-2 -*-
# z pliku wczytujemy dane nazwa email
# i wpisujemy do słownika
def wyslij_do(do,tresc):
    print "wysłałeś maila do ",
    print do,
    print "o treści: ",
    print tresc
mail={}
tresc="Python jest super"
infile = file('t.dat', 'r')
# można by wczytywać z bazy danych używając np.: MySQLdb
for line in infile:
    line=line.strip('\n').split(' ')
    mail[line[0]]=line[1]
print mail.keys()
```

```
print len(mail)
for o in mail.itervalues():
    wyslij_do(o,tresc)
#Tak mógłby powstać interfejs www do odbierania/wysyłania poczty
```

Przykład użycia słownika jako kontenera do przechowywania danych programu oraz jego opcji wraz z reakcją na nie (kod autorstwa Marcina "KoD"Walkowiaka:

```
_cmdLineConfig = {
"usage": "%prog {[options] --address=ADDRESS | --config-file=CONFIGFILE}",
"version": "%prog 1.0",
"description": "",
"option_list":
[make_option("--address",action = "store",type = string,dest =
"address",help = "url to web page or site to download")
]
}
```

5 Typy wbudowane i konwersja typów

Najpopularniejsze są typy logiczne (bool).

- Typ boolean:

`x, y = True, False` # składa się z dwóch stałych oznaczających Prawdę i Fałsz
Pisane wielką literą.

- Wybrane operacje na typie boolean:

```
x = not y # x = True, gdy y == False
z = x and y # z = True, gdy x, y = True, True
z = x or y # z = False, gdy x, y = False, False

z = 2 in [1, 2, 3] # z = True, jeżeli 2 występuje w kolekcji [1, 2, 3]
z = 1 not in [1, 2, 3] # z = False, gdyż 1 występuje w kolekcji [1, 2, 3]
z = 4 not in [1, 2, 3] # z = True, jeżeli 4 nie występuje w kolekcji [1, 2, 3]
```

- Typy Decimal i Fraction:

Żeby móc z nich skorzystać, wpisujemy na początku skryptu dwie poniższe linie:

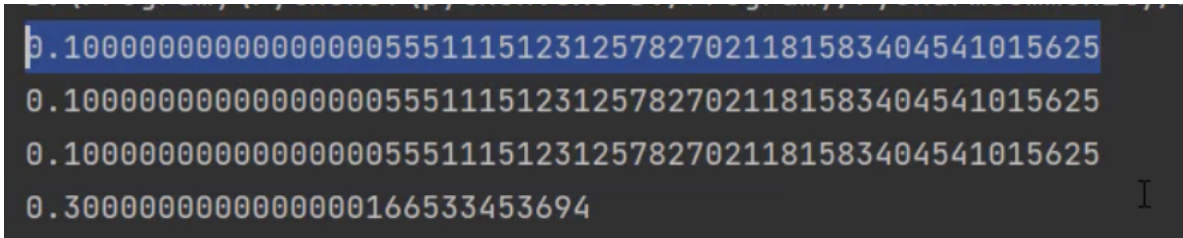
```
from decimal import Decimal
from fractions import Fraction
```

- Wybrane operacje na typach Decimal i Fraction:

```
x_decimal = Decimal('0.1') # tworzenie obiektu typu Decimal.
x_fraction = Fraction(1, 10) # tworzenie obiektu typu Fraction.
```


- Decimal i Fraction – przykłady:

```
from decimal import Decimal
from fractions import Fraction
x = Decimal(0.1)
y = Decimal(0.1)
z = Decimal(0.1)
print(x)
print(y)
print(z)
print(x + y + z)
```



```
0.10000000000000000055511151231257827021181583404541015625
0.10000000000000000055511151231257827021181583404541015625
0.10000000000000000055511151231257827021181583404541015625
0.30000000000000000166533453694
```

Wynik:

W języku python wartości zmiennoprzecinkowe mają trochę "inna twarz".

Jak obejść ten "problem"?

Przedstawiamy dane "Decimal" w formie łańcuchów:

```
from decimal import Decimal
from fractions import Fraction
x = Decimal('0.1')
y = Decimal('0.1')
z = Decimal('0.1')
print(x)
print(y)
print(z)
print(x + y + z)
```



```
0.1
0.1
0.1
0.3
```

Wynik:

Jak widać, wszystko działa.

- Tworzenie licznika i mianownika – ułamki zwykłe (Fraction):

```
x = Fraction(1, 5)
```

z dodawaniem liczby całkowitej:

```
x = Fraction(1, 5) + 4
```

- Zakres (range):

```
x = range(10) # tworzenie ciągu liczb od 0 do 9 (włącznie)
y = range(5, 10) # tworzenie ciągu liczb od 5 do 9 (włącznie)
z = range(5, 50, 5) # tworzenie ciągu liczb od 5 do 45 (włącznie) z krokiem co 5,
czyli 5, 10, 15 itd.
```

- Wypisywanie liczb **x** z podanego zakresu:

```
print(list(x), type(x))
print(list(y), type(y))
print(list(z), type(z))
```

lub:

```
for x in range(10): print(x)
```

- Obliczanie średniego wieku, np. pracowników:

```
from statistics import mean

wiek_pracownikow = [23, 34, 32, 45, 43, 23, 23]
print ("Wiek pracownikow:")
for pracownik in wiek_pracownikow:
    print (pracownik)
print ("Sredni wiek:", mean (wiek_pracownikow))
```

- Konwersja typów:

```
lancuch_znakow = "dddddddd"
lista = [1, 2, 3]
liczba_calkowita = 19
liczba_zmiennoprzecinkowa = 10.5
ciag_liczb = range(20)

print(list(lancuch_znakow)) # konwersja str na list
print(tuple(lancuch_znakow)) # konwersja str na tuple
print(tuple(lista)) # konwersja list na tuple
print(float(liczba_calkowita)) # konwersja int na float
print(int(liczba_zmiennoprzecinkowa)) # konwersja float na int
print(list(ciac_liczb)) # konwersja range na list
print(tuple(ciac_liczb)) # konwersja range na tuple
```

6 Operacje wejścia/wyjścia

Już w 1949 roku były używane na komputerach Mark II.

6.1 Funkcja wbudowana `input()`:

- Pobranie i wyświetlenie wartości od użytkownika:

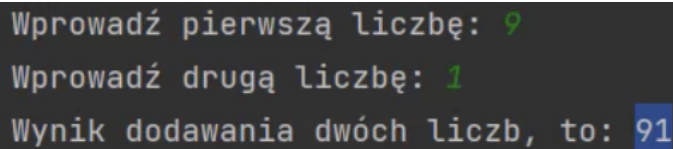
```
x = input()
print(x)
```

- Operacja dodawania na dwóch podanych wartościach całkowitoliczbowych:

```
x = int(input("Liczba 1: "))
y = int(input("Liczba 2: "))
z = x + y
print(z)
```

Przetestujmy ten program:

```
print("Dodawanie dwóch liczb.")
x = int(input("Wprowadź pierwszą liczbę: "))
y = int(input("Wprowadź drugą liczbę: "))
print("Wynik dodawania dwóch liczb, to: ", x + y)
```



```
Wprowadź pierwszą liczbę: 9
Wprowadź drugą liczbę: 1
Wynik dodawania dwóch liczb, to: 91
```

Wynik:

Ale możemy to zmienić:

```
print("Dodawanie dwóch liczb.")
x = input("Wprowadź pierwszą liczbę: ")
y = input("Wprowadź drugą liczbę: ")
print("Wynik dodawania dwóch liczb, to: ", x + y)
```

W przypadku, kiedy podamy ułamek lub literkę – wystąpi błąd, ale ten możemy rozwiązać za pomocą funkcji sterującej.

7 Pliki

Obsługa pliku wydaje się oczywista, ale warto ją powtórzyć.

- Tworzenie nowego pliku:

```
plik = open("plik.txt", 'w') # 'w' jest skrótem od słowa 'write'
plik.write("Wpisujemy jakiś tekst.")
plik.write("\n") # przejście do nowej linii
plik.write("Kolejny tekst.") # kolejny tekst, już w nowej linii.
plik.close() # zamknięcie strumienia wejściowego do pliku.
```

- Otwieranie istniejącego już pliku:

- Pierwsze podejście:

```
plik = open("plik.txt")
plik = open("plik.txt", 'r') # 'r' jest skrótem od słowa 'read'
zawartosc_pliku = plik.read()
plik.close()
```

- Drugie podejście:

```
with open("plik.txt") as plik:
    zawartosc_pliku = plik.read()
```

Powyższe (drugie) podejście może być wykorzystywane w dowolnej pracy z plikami.

- Dodawanie treści do już istniejącego pliku:

```
plik = open("plik.txt", 'a')
plik.write("\n") # Dodawanie treści od nowej linii
plik.write("Ta linia zostanie dodana do pliku.")
plik.close()
```

8 Podsumowanie

- Normy dotyczące używania nawiasów klamrowych zawarte zostały w **PEP 3100**.

9 Zadania do poćwiczenia

1. Mamy zadaną listę [4, 4, 4, 2, 2, 3, 5, 6, 7, 8, 7, 5, 4, 3, 2, 4]., Proszę **BEZ** używania pętli wyznaczyć element listy, który powtarza się najczęściej.
2. Napisz program w Pythonie, aby utworzyć krotkę.
3. Napisz program w Pythonie, aby utworzyć krotkę z różnymi typami danych.
4. Napisz program w Pythonie, który utworzy krotkę liczb i wyświetli tylko jeden element.
5. Napisz program w Pythonie, aby przekonwertować krotkę na słownik.
6. Napisz program, w którym będą wykonywane operacje dodawania i odejmowania na liczbach całkowitych.
7. Napisz program imitujący dialog dwóch postaci.
 - Zadania nie są obowiązkowe, ale warto je zrobić.
 - Będą oceny za aktywność!
 - W pierwszej linijce komentarz z imieniem, nazwiskiem oraz grupą.
 - Dane do kontaktu: **aluckiewicz@pjwstk.edu.pl** oraz **adrianewicz@gmail.com**