
	Politechnika Bydgoska im. J. J. Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki		
Przedmiot	Skryptowe języki programowania		
Prowadzący	mgr inż. Martyna Tarczewska		
Temat	Struktury danych języka Python		
Student			
Nr ćw.	3	Data wykonania	
Ocena		Data oddania spr.	

1. Cel ćwiczenia

Celem ćwiczenia jest podstawowych struktur danych zawartych w języku programowania *Python* przez wykonanie prostych zadań. **Funkcje i zmienne stworzone podczas zajęć powinny być odpowiednio otypowane z użyciem biblioteki *typing*. Kod powinien być napisany w języku angielskim.**

2. Informacje podstawowe

2.1. Listy

Lista jest dynamiczną strukturą danych. Stanowi ona uporządkowany, zmienny zbiór obiektów, w którym można przechowywać obiekty różnego rodzaju. Można nimi zarządzać (dodawać, usuwać, modyfikować itp.) Listy tworzone są za pomocą nawiasów kwadratowych.

Podstawowe działania przedstawiono poniżej. Pokazane zostało inicjowanie listy, sprawdzenie zawartości listy, różne sposoby wypisania elementów z listy oraz sprawdzenie jej długości.

```
list = [1,2,3,4,5,6,7,8] #definiowanie listy
print(8 in list) #czy element należy do listy
print(12 in list) #czy element należy do listy
print(list[2:5]) # 3, 4, 5
print(list[4:]) # 5, 6, 7, 8
print(list[:4]) # 1, 2, 3, 4
print(list[::2]) #lista[początek:koniec:krok] # 1, 3, 5, 7
print(sorted(list)) #sortowanie listy - 1,2,3,4,5,6,7,8
list.extend([4,5]) #rozszerzenie listy o kolejną listę
list = [1, 10, 15, 20]
list[0:3]=[1, 12] # 1, 12, 20
list[1:1]=['kot', 'pies'] # 1, kot, pies, 12, 20
list[:0]=list # 1, kot, pies, 12, 20, 1, kot, pies, 12, 20
len(list) #długość listy
list[:]=[] #pusta lista
```

Przydatne będą również poniższe funkcje:

```
list.append(x) #dodanie elementu x na końcu
list.insert(i, x) #dodanie elementu x pod indeksem i
list.remove(x) #usunięcie pierwszej pozycji, której wartość to x
del list[i] #usunięcie elementu z indeksu i
list.index(x) #indeks elementu x lub błąd
list.count(x) #zlicza wystąpienia elementu x
list.sort() #sortowanie listy
list.reverse() #odwrócenie listy
a = list.pop(i) #pobranie elementu o indeksie i zwrócenie go do a
a = list.pop() #pobranie ostatniego elementu i zwrócenie go do a
```

2.2. Warunki i pętle

Do iterowania po strukturze danych niezwykle przydatna będzie pętla FOR lub pętla WHILE, których równoważne konstrukcje zostały pokazana poniżej.

Pokazano również instrukcję warunkową IF. Aby zaprzeczać i łączyć warunki stosujemy słowa kluczowe AND, OR oraz NOT.

```
for i in range(5):
    print(i)

i = 0
while i < 5:
    print(i)
    i += 1

a = 200
b = 33

if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

2.3. Krotki

Krotki to struktury danych, które podobnie jak listy przechowują różnego typu dane. W tym przypadku również elementy mogą się powtarzać, jednak nie jest możliwa ich modyfikacja. Tworzone są za pomocą okrągłych nawiasów.

```
tuple = ("apple", "banana", "cherry") #stworzenie krotki
print (tuple[0]) # wypisanie elementu z indeksu 0
print (tuple) # wypisanie krotki
nested_tuple = tuple, (1,2,3,4,5) # definiowanie zagnieżdżonej krotki
print(nested_tuple)
```

Modyfikacja elementów krotki nie jest możliwa (także usunięcie elementu). Można to zrobić jedynie poprzez stworzenie listy z krotki, usunięcie/modyfikacja elementu i ponowne przejście do krotki.

Przykładowe działania na krotkach (**do zadania 7**):

```
tuple = ("apple", "banana", "cherry")
tuple_b = ("orange",)
tuple += tuple_b #dodawanie krotek
multi_tuple = tuple * 2 #mnożenie krotek
print(len(tuple)) #długość krotki - liczba elementów
for x in tuple: #wypisanie wszystkich elementów krotki
    print(x)
```

2.4. Zbiory

Zbiór jest w matematyce pojęciem pierwotnym, czyli jego definicja jest intuicyjna, niedefiniowalna. Jest to pewien zestaw elementów. Nowy zbiór tworzymy z wykorzystaniem funkcji set(), jako parametr możemy podać listę lub krotkę z elementami, które mają zostać na starcie dołączone do zbioru. Zbiór można też tworzyć za pomocą nawiasów klamrowych.

Pierwszą różnicą między zbiorem i poprzednimi strukturami danych jest indeksowanie. Zbiór jest tylko workiem na elementy, w przeciwieństwie do uporządkowanych krotek i list. Wartości nie posiadają swoich indeksów. Dzięki temu każda wartość występuje dokładnie jeden raz.

```
the_set = set( ["orange", "banana"] ) #definiowanie zbioru
this_set = {"apple", "banana", "cherry"} #definiowanie zbioru
print("banana" in this_set) #sprawdzenie czy element należy do zbioru
sum_set = this_set.union(the_set) #suma zbiorów
this_set.remove("banana") #usunięcie elementu
this_set.discard("banana") #usunięcie elementu
x = this_set.pop() #wyjęcie ze zbioru jakiegoś elementu
fruits = ["kiwi", "orange"]
sum_set.update(fruits) #aktualizacja listą
tropical = {"pineapple", "mango", "papaya"}
sum_set.update(tropical) #aktualizacja zbiorem
```

2.5. Słowniki

Struktury danych przechowujące pary elementów klucz – wartość. Najczęściej spotykane metody przedstawiono poniżej.

```
car_dict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964,
    "new": False
}
x = car_dict["model"] #pobranie wartości z klucza model
x = car_dict.get("model") #inny sposób pobrania wartości
print(car_dict.keys()) #lista kluczy
print(car_dict.values()) #lista wartości
print(car_dict.items()) #lista par klucz-wartość
car_dict["color"] = "white" #dodanie nowej pary klucz-wartość
car_dict["year"] = 1963 #aktualizacja wartości
if "model" in car_dict: #sprawdzenie czy klucz należy do słownika
    print("element found!")
car_dict.pop("model") #zdjęcie elementu o danym kluczu
car_dict.popitem() #zdjęcie ostatniego dodanego elementu
```

Testowanie kodu

Czym jest pytest?

pytest (<https://docs.pytest.org/en/7.4.x/>) to popularna biblioteka w języku Python służąca do tworzenia testów jednostkowych. Umożliwia ona szybkie i łatwe pisanie testów, a jej prosty składniowy format sprawia, że jest bardziej czytelna w porównaniu z innymi frameworkami testującymi w Pythonie.

Aby rozpocząć korzystanie z pytest, musisz najpierw zainstalować go. Można to zrobić za pomocą pip:

```
pip install pytest
```

Testy w pytest zaczynają się od słowa **test_**. Dla przykładu:

```
def add(a: int, b: int) -> int:
    return a + b

def test_add():
    assert add(2, 3) == 5
```

W powyższym kodzie mamy funkcję add, która dodaje dwie liczby, a następnie test **test_add**, który sprawdza, czy wynik jest poprawny.

Nazwa pliku zawierającego testy musi się zaczynać od przedrostka **test_**.

Po napisaniu testu, możemy go uruchomić w konsoli, pytest automatycznie znajduje i uruchamia wszystkie funkcje rozpoczynające się od **test_**:

pytest nazwa_pliku.py

3. Przebieg ćwiczenia

3.1. Zadanie 1.

W oddzielnej funkcji utworzyć nową pustą listę. Wykonać kolejno następujące operacje:

- a. wypisać całą listę
- b. dodać do niej 5 elementów
- c. wypisać pierwsze 2 elementy i ostatnie dwa elementy listy
- d. sprawdzić jej długość
- e. wypisać elementy z parzystych indeksów
- f. dodać element numeryczny
- g. dodać element napisowy
- h. posortować listę
- i. usunąć ostatni dodany element
- j. posortować listę odwrotnie
- k. dodać element na miejsce o indeksie 2
- l. zliczyć ile elementów o wartości 13 jest w liście

Pytania do zadania:

- a. Czy udało się wykonać wszystkie operacje?
- b. Od którego indeksu numerowane są listy w Pythonie?

3.2. Zadanie 2.

Napisać funkcję, która stworzy nową listę i wypełni ją 10 liczbami korzystając z konsoli (użytkownik podaje liczby w pętli). Za pomocą sortowania uzyskać element największy i najmniejszy. Obliczyć średnią liczb nieujemnych.

3.3. Zadanie 3.

Napisać funkcję, która utworzy dwie listy (int) i wypełni je wartościami (min. 5 elementów w każdej liście). Wypisać elementy unikatowe listy 1, czyli elementy występujące w liście 1, ale nie znajdujące się w liście 2.

3.4. Zadanie 4.

Napisać funkcję, która utworzy listę z 10 elementami (int). Wypisać z niej jedynie elementy nieparzyste. Podać najmniejszy z nich.

3.5. Zadanie 5.

Napisz funkcję, która:

1. Utworzy dwie listy — **list_A** i **list_B** — z co najmniej trzema dowolnymi elementami każda.
2. Połączy te listy na dwa różne sposoby:

- **Sposób 1:** utworzy nową listę, w której najpierw znajdą się elementy z `list_A`, a następnie z `list_B`.
 - **Sposób 2:** utworzy nową listę, w której najpierw znajdą się elementy z `list_B`, a następnie z `list_A`.
3. Wyświetli wyniki obu połączeń.



Wskazówka:

Zastanów się, jak działa wyrażenie `listA[:0] = listB`.

Co się stanie z zawartością `listA`, jeśli użyjesz tego zapisu?

3.6. Zadanie 6.

Napisać funkcję, która pobierze od użytkownika liczby i zapisze je do listy. Zero kończy wpisywanie liczb. Następnie wypisać wszystkie elementy unikatowe listy (elementy bez powtórzeń). Skorzystać ze zbioru.

3.7. Zadanie 7.

Wrócić do części teoretycznej instrukcji. Skopiować kod z instrukcji dotyczący krotek do nowej funkcji i uruchomić go. Odpowiedzieć na pytania:

- a. Jak działa dodawanie krotek?
- b. Jaki wynik otrzymamy w wyniku dodania do siebie dwukrotnie tej samej krotki?
- c. Jak działa mnożenie krotek?
- d. Jakie jest zadanie przecinka w zapisie: `tuple_B = ("orange",)` Czy można go usunąć?
- e. Czy dla krotek działa sortowanie takie jak dla list?

3.8. Zadanie 8.

Wrócić do części teoretycznej 2.4. Stworzyć w nowej funkcji definicję zbioru (5 elementów) i przetestować działania:

- a. usuwanie metodą `remove` i `discard` (sprawdzić działanie dla nieistniejącego elementu)
- b. zdejmowanie elementów metodą `pop` – Czy zawsze zdejmuje się ten sam element? Dlaczego tak się dzieje?

3.9. Zadanie 9.

Napisać funkcję, w której zdefiniowane będą dwa zbiory z liczbami. Przetestować metody (co jest wynikiem działania funkcji i jakiego jest typu):

- a. `the_set.isdisjoint(this_set)`
- b. `the_set.issubset(this_set)`
- c. `the_set.issuperset(this_set)`
- d. `the_set.union(this_set)`
- e. `the_set.difference(this_set)`
- f. `the_set.intersection(this_set)`

3.10. Zadanie 10.

Napisać funkcję, w której zostanie zdefiniowany nowy słownik z kilkoma parami klucz—wartość (można korzystać ze słownika `car_dict` z początku instrukcji). Wykonać następujące działania:

- a. dodać kolejną parę klucz-wartość
- b. zaktualizować jeden z istniejących wpisów
- c. przetestować metody `pop` i `popitem`
- d. sprawdzić, czy możliwe jest użycie w jednym słowniku kluczy o dwóch różnych typach (np. `int` i `string`)? Czy możliwe jest użycie w jednym słowniku wartości o różnych typach?

3.11. Zadanie 11.

Napisać funkcję, w której zostanie zasymulowany arkusz ocen studenckich. Stworzyć zagnieżdżony słownik, w którym kluczem są numery indeksu, a wartościami podrzędne słowniki (zawierające klucze: „imię”, „nazwisko” i „oceny” oraz wartości: imię studenta (`str`), nazwisko studenta (`str`), lista ocen (`list`). Wypisać wszystkich studentów ze słownika (wypisać imię, nazwisko, numer indeksu oraz średnią z ocen).

3.12. Zadanie 12.

Wybrać dwie z utworzonych funkcji i przy pomocy biblioteki *pytest* utworzyć po trzy testy sprawdzające różne scenariusze działania wybranych funkcji. Wyniki testów zamieścić w sprawozdaniu.

4. Sprawozdanie

Sprawozdanie z laboratorium powinno zawierać:

- wypełnioną tabelę z początku instrukcji,
- kody programów będących rozwiązaniami wszystkich zadań wraz z komentarzami,
- demonstrację działania programu,
- odpowiedzi na pytania zawarte w zadaniach,
- wnioski.