Moduł 3 - Podstawowe struktury danych, część 2.

1. Typ `list`.

> **Dokumentacja**: https://docs.python.org/3.12/tutorial/datastructures.html#more-on-lists

Lista w języku Python to kolekcja, którą można porównać do tablic w innych językach programowania. Ważną cechą list jest to, że mogą przechowywać różne typy danych (są więc heterogeniczne). Rozmiar tablicy ograniczony jest możliwościami sprzętu. Listę możemy zainicjalizować w poniższy sposób:

Listing 1

```
lista = []
lista2 = list()
lista3 = [1, 2, 3]
lista4 = ['a', 5, 'Python', 7.8]
```

Do elementów listy odwołujemy się tak samo jak do elementów ciągu tekstowego, czyli poprzez odwołanie do indeksu elementu listy. Obiekty typu `list` w odróżnieniu od tablic znaków (typ `str`) są mutowalne, co oznacza, że możemy zamieniać wartość danego elementu.

Listing 2

```
lista = []
lista.append([1, 2, 3])
lista.append(['a', 5, 'Python', 7.8])
# efektem będzie dodanie list jako elementów listy (lista dwupoziomowa)

print(lista)

# wyjście
# [[1, 2, 3], ['a', 5, 'Python', 7.8]]


lista[0][0] = 0
# nowa postać listy
# [[0, 2, 3], ['a', 5, 'Python', 7.8]]
```

W Pythonie można też w łatwy sposób łączyć ze sobą listy:

Listing 3

```
# rozszerzanie listy o elementy innej listy (taki append() w pętli), zachowując
kolejność
lista3.extend(lista4)
print(lista3)
```

```
# wyjście
# [1, 2, 3, 'a', 5, 'Python', 7.8]

# lub w inny sposób
lista6 = lista3 + lista4
print(lista6)

# wyjście
# [1, 2, 3, 'a', 5, 'Python', 7.8]
```

Obie metody różnią się od siebie tym, że pierwsza modyfikuje już istniejącą listę, a druga wymaga podstawienia połączonej listy pod zmienną, gdyż sama arytmetyczna operacja „+" nie spowoduje zmiany pierwotnej listy, a zawróci nowy obiekt typu `list` z elementami z obu list.

Niektóre metody, które można wykonać na obiekcie listy, wykonywane są jako operacje **in-place** co oznacza, że operacja wykonywana jest bez zwracania nowej wartości (co przy próbie przypisania wartości zwracanej przypisze zmienną typu **NoneType**), nadpisując zawartość oryginalnej zmiennej. Poniżej przykład z sortowaniem.

Listing 4

```
lista7 = [7, 9, 3, 1]

# operacja sort działa w trybie in-place
posortowana = lista7.sort()
print(lista7)
print(posortowana)

# wyjście
# [1, 3, 7, 9]
# None
```

Wartość **None** (typ **NoneType**) w Pythonie odpowiada wartości **Null** w innych językach programowania i oznacza **nic**, specjalny typ danych. Nie możemy też posortować tablicy, w której znajdują się niejednorodne wartości np. liczby oraz ciągi tekstowe.

Listy mogą być „cięte" (ang. **sliced**) tak jak ciągi tekstowe przedstawione w poprzednim rozdziale. Dodawanie, usuwanie i zmiana wartości elementów listy może być wykonywana na wiele sposobów. Poniżej listing z niektórymi z nich.

Listing 5

```
# wstawianie i usuwanie elementów listy
skala = [1, 2, 3, 4, 5]
# dodajemy element na końcu listy
skala.append(6)
print(skala)
```

```
# znamy już sposób odwoływania się do elementu listy poprzez indeks więc można
# wartości listy ustawiać w ten sposób, ale nie możemy odwołać się'
# do indeksu, który nie istnieje
skala[6] = 7
# zostanie zwrócony błąd: IndexError: list assignment index out of range
# ale można zrobić to np. tak
skala[6:] = [7]
# lub tak - z dynamicznym sprawdzaniem aktualnego rozmiaru listy
skala[len(skala):] = [7]

# alternatywnym sposobem jest wywołanie metody insert
skala.insert(6, 7)
print(skala)

# usuwamy element z końca listy co powoduje, że z wykorzystaniem
# tych metod osiągamy funkcję stosu
skala.pop()
print(skala)

# pop może również przyjmować indeks elementu do usunięcia.
# Metoda pop zwraca również wartość elementu usuwanego
skala.pop(2)
print(skala)
```

Wartości listy można rozpakować (bardziej popularne jest to dla krotek) do zmiennych, gdzie możemy się wspomóc poprzez użycie symbolu *. Przykład poniżej.

Listing 6

```
order = [1, 2, 3]

first, second, third = order

# jeżeli liczba elementów i zmiennych nie jest zgodna
# możemy elementy, które "zostają" przypisać do jednej zmiennej
first, *second = order

# o ile first przyjmie wartość, która w liście na danej pozycji występuje
# o tyle pozostałe wartości wciąż pozostaną w postaci obiektu typu list

# ten mechanizm zostanie jeszcze omówiony przy okazji zagadnienia związanego
# z funkcjami z nieokreśloną ilością atrybutów pozycyjnych
```

Do usuwania elementów listy można wykorzystać również wbudowaną funkcję **del()**. Za jej pomocą można również usuwać zmienne.

W tym rozdziale zostały zaprezentowane podstawowe operacje na listach. Python oferuje wiele bardziej rozbudowanych możliwości generowania, wybierania, sortowania list jednak zostaną one zaprezentowane

razem z przykładami pętli oraz instrukcji warunkowej.

Ćwiczenia!

> Wykonaj zadania numer 1,2 oraz 3 znajdujące się na końcu tego dokumentu.

2. Słowniki, czyli typ `dict`.

> **Dokumentacja**: https://docs.python.org/3.12/tutorial/datastructures.html#dictionaries

Słownik to tablica mieszająca lub inaczej tablica z haszowaniem, którą można porównać do tablic asocjacyjnych znanych z innych języków programowania. Słowniki przechowują pary **klucz: wartość** i właśnie poprzez odwołanie do klucza odbywa się dostęp do elementu.

> **Do wersji 3.6 języka Python słowniki nie gwarantowały porządku elementów zgodnie z kolejnością ich dodawania. Od wspomnianej wersji słowniki są uporządkowane zgodnie z kolejnością dodawania elementów.**

Kluczem w słowniku może być każdy niezmienny typ (niemutowalny, ang. immutable type) danych np., string lub liczba. Kluczem może być również krotka (jeden z typów danych języka Python), jeżeli zawiera typy niezmienne (string, liczba, krotka).

Poniżej fragmenty kodu tworzące słownik oraz pokazujące jak uzyskać dostęp do jego danych.

Listing 7

```
# tworzenie słownika
slownik = {}
slownik = dict([('jeden', 1), ('dwa', 2), ('trzy', 3)])
slownik = dict(jeden=1, dwa=2, trzy=3)
slownik = dict({'jeden': 1, 'dwa': 2, 'trzy': 3})
slownik = {'jeden': 1, 'dwa': 2, 'trzy': 3}

print(slownik['jeden'])

# sprawdzenie, czy klucz jest w słowniku czy nie
print('jeden' in slownik)

# wypisanie wszystkich kluczy
print(slownik.keys())

# wypisanie wszystkich wartości
print(slownik.values())

# można również sprawdzić, czy klucz występuje w słowniku
# w przedstawiony poniżej sposób, ale jest on wolniejszy
print('jeden' in slownik.keys())

# dodanie elementu do słownika
slownik['cztery'] = 4
print(slownik.keys())
```

Ćwiczenia!

> Wykonaj zadania numer 4,5 oraz 6 znajdujące się na końcu tego dokumentu.

3. Krotki, czyli typ `tuple`.

> **Dokumentacja**: https://docs.python.org/3.12/tutorial/datastructures.html#tuples-and-sequences

Krotki (ang. tuples) są bardzo podobne do list z tą różnicą, że są typem niezmiennym i deklaracja zmiennych zapisywana jest w nawiasach zwykłych, a nie kwadratowych. Również mogą przechowywać wiele typów danych jednocześnie.

Listing 8

```
# tworzymy krotkę
krotka = (1, 2, 'Jacek', 'ma')
krotka_liczb = krotka[:2]
print(krotka_liczb)
krotka_stringow = krotka[2:]
print(krotka_stringow)

nowa_krotka = tuple()
najnowsza_krotka = tuple([1, 2, 3])

# możemy również rzutować typy krotka - lista
lista = [1, 2, 'Ala', 'też', 'ma']
krotka_z_listy = tuple(lista)
nowa_lista = list(krotka_z_listy)

# krotki mogą być zagnieżdżane
duza_krotka = krotka_stringow, krotka_liczb, tuple(nowa_lista)
print(duza_krotka)

# a jeżeli zagnieździmy listę w krotce ?
listokrotka = krotka_z_listy, lista
# to nadal możemy modyfikować elementy listy
listokrotka[1][0] = 0
print(listokrotka)

# pakowanie krotki (tuple packing)
t = 5, 6, 7
print(t)
x, y, z = t
# i rozpakowywanie krotki (tuple unpacking)
# inny sposób łączenia zmiennych różnego typu w string
print('x = ' + str(x))
print('y = ' + repr(y))
print('z = ' + str(z))

# na wyjściu
(1, 2)
('Jacek', 'ma')
```

```
(('Jacek', 'ma'), (1, 2), (1, 2, 'Ala', 'też', 'ma'))
((1, 2, 'Ala', 'też', 'ma'), [0, 2, 'Ala', 'też', 'ma'])
(5, 6, 7)
x = 5
y = 6
z = 7
```

4. Zbiory, czyli typ `set`.

> **Dokumentacja**: https://docs.python.org/3.12/tutorial/datastructures.html#sets

Zbiór (ang. set) to nieuporządkowana kolekcja, której ważną cechą jest to, że znajdują się w niej unikalne elementy (czyli bez powtórzeń). Zbiory obsługują również matematyczne operacje, które znane są z teorii zbiorów: suma, przecięcie, różnica oraz różnica symetryczna.

Listing 9

```
# inicjalizacja zbiorów
klasa = {'Marek', 'Janek', 'Ania', 'Ewa', 'Marek', 'Ania'}
print(klasa) # duplikatów już nie ma

# a teraz zbiór znaków ze stringa
czar = set('czabunagunga')
print(czar)
inny_czar = set('abrakadabra')
print(inny_czar)
print(czar - inny_czar) # są w czar, ale nie ma w inny_czar
print(czar.difference(inny_czar)) # to samo, ale inaczej
print(inny_czar - czar) # to nie to samo, jak wiadomo z teorii

print(czar | inny_czar) # znaki w czar lub inny_czar lub obu
print(czar & inny_czar) # przecięcie zbiorów, czyli część wspólna
print(czar.intersection(inny_czar)) # można tak
print(czar ^ inny_czar) # różnica symetryczna
```

Bardzo przydatna staje się własność unikalnych elementów zbioru jeżeli chcemy wyeliminować duplikaty z listy, gdyż wystarczy rzutować listę na zbiór. Rzutowanie to konwersja jednego typu danych na inny. Nie zawsze jest ona możliwa. Następnie, jeżeli na wyjściu potrzebujemy znowu listy, to rzutujemy w odwrotną stronę.

Ćwiczenia!

> Wykonaj zadanie numer 7 znajdujące się na końcu tego dokumentu.

5. Funkcja `range`.

> **Dokumentacja**: https://docs.python.org/3.12/library/functions.html#func-range

Funkcja (chociaż nazywając precyzyjniej to niezmienna sekwencja) `range` służy do generowania ciągu liczb według zadanych parametrów. Często przydaje się w pętlach lub podczas tworzenia list lub zbiorów liczb. Funkcja range() i sposób jej użycia zmieniał się w trakcie rozwoju języka i jej zastosowanie w wersji Python 2.x różni się od tego, co będzie zaprezentowane tutaj. Zapoznanie się ze szczegółami dotyczącymi tych różnic pozostawiam czytelnikowi.

Poniższe fragmenty kodu zaprezentują sposób działania funkcji range.

Listing 10

```
# jakim typem jest range ?
liczby = range(5)
print(type(liczby)) # range jest obiektem typu range (w Python 2.x była to lista)

# range może przyjmować 1 parametr, wtedy jest to parametr stop
for i in range(10):
    print(i)

# range może też przyjmować 2 parametry (start, stop)
for i in range(4, 10):
    print(i)

# lub 3 parametry (start, stop, step)
for i in range(4, 10, 2):
    print(i)

# możemy również generować wartości ujemne
for i in range(-5, -1):
    print(i)

for i in range(-5, -10, -2):
    print(i)

# lista z elementów funkcji range
lista = list(range(10))
print(lista)

# funkcja range nie generuje wartości zmiennoprzecinkowych, ale można
# dość łatwo taką funkcję stworzyć - poniżej funkcja generująca (yield)
def frange(start, stop, step):
    i = start
    while i < stop:
        yield i
        i += step

for i in frange(0.1, 0.5, 0.1):
    print(i)
```

Dla osób chcących przenieść swoją wiedzę na temat list i krotek na nieco wyższy poziom, zachęcam do przeczytania artykułu na temat mechanizmu alokacji pamięci dla tych dwóch typów w implementacji CPython:

https://www.opensourceforu.com/2021/05/memory-management-in-lists-and-tuples/ Można tam znaleźć również informacje o złożoności obliczeniowej wybranych metod dla klasy `list` co może pomóc w procesie optymalizacji kodu wykorzystującego te kolekcje.

Ćwiczenia

> Wykonaj zadanie numer 8 znajdujące się na końcu tego dokumentu.

Zadania

Zadanie 1
Stwórz listę z wartościami od 1 do 10. Następnie podziel listę tak, aby pierwsze 5 liczb zostało w oryginalnej liście a pozostałe 5 znalazło się w nowej liście.

Zadanie 2
Połącz te listy ponownie. Dodaj do listy wartość „0” na początku. Utwórz kopię połączonej listy i wyświetl listę posortowaną malejąco.

Zadanie 3
Napisz skrypt, który pobierze dowolny tekst ze standardowego wejścia poprzez funkcję `input()`. Następnie wyświetl ciąg unikalnych znaków z wczytanego zdania, zapisanych alfabetycznie małymi literami.*

* wykorzystaj rzutowanie typu `str` na `set` oraz `set` na `list` i użyj funkcji sortującej listę

Zadanie 4
Stwórz słownik gdzie kluczami będą numery miesięcy (rozpoczynając od 1) a wartościami nazwy polskich miesięcy.

Zadanie 5
Stwórz podobny słownik jak w zadaniu 4, ale z angielskimi nazwami miesięcy. Połącz teraz słowniki tak, żeby przykładowo dla kwietnia, dostać się poprzez wyrażenie: months['pl'][4] a dla wersji angielskiej poprzez months['en'][4].

Zadanie 6
Wykorzystując ciąg tekstowy 'Marianna' oraz metodę **fromkeys()** dla słowników stwórz słownik, który będzie zawierał jako klucze unikalne litery w/w imienia a jako wartość każdy klucz będzie miał przypisaną wartość 1. Poprawne wyjście: `{'M': 1, 'a': 1, 'r': 1, 'i': 1, 'n': 1}`

Zadanie 7
Wykorzystaj moduł `string` (dodaje się go poprzez instrukcję `import string` zapisaną zazwyczaj na początku skryptu) i następnie:

- wczytaj ze standardowego wejścia dowolny łańcuch znaków,
- używając formatowania znaków wyświetl ile znaków oraz jaki procent (zamienionych na małe litery) z nich pokrywa się ze zbiorem znaków z: `string.ascii_lowercase, string.digits` (podpowiedź: operator `in`)

Przykład:
Wejście (input):

```
Ala ma kota.
```

Wyjście (output):

```
W zdaniu 'Ala ma kota.' występuje 6 znaków wspólnych ze zbiorem
string.ascii_lowercase,  co stanowi  23.00 % tego zbioru.
W zdaniu 'Ala ma kota.' występuje 0 znaków wspólnych ze zbiorem string.digits,  co
stanowi  0.00 % tego zbioru.
```

Zadanie 8

Napisz kod, w którym pobierzesz za pomocą funkcji `input()` 3 wartości przypisując je do zmiennych: start, stop oraz step. Następnie użyj ich jako parametrów funkcji `range` i wykorzystując przykłady z listingu 10 wypisz wszystkie wartości ciągu wygenerowane przez tę funkcję. Zwróć uwagę na typ danych, który zwraca funkcja `input()` - będzie konieczna konwersja (rzutowanie).