Nazwa implementacji:

Nauka języka Python – zaawansowane typy danych Autor: Piotr Fiorek

Opis implementacji:

Poznanie zaawansowanych typów danych oraz nauczenie się do czego i jak je wykorzystywać. Do tej pory używaliśmy prostych zmiennych, gdzie do jednej nazwy zawsze przypisana była tylko jedna wartość. Czas omówić inne typy zmiennych gdzie pod jedną nazwą przechowujemy więcej niż jedną wartość. Będą nam one potrzebne przy omawianiu kolejnej pętli dostępnej w Pythonie. Najpopularniejszym i najczęściej stosowanym typem zmiennych zawierającym więcej niż jedną wartość są listy (od angielskiego „list"), czasami nazywane też tablicami. Lista to uporządkowany zbiór różnych elementów. Najczęściej wewnątrz listy stosuje się jeden typ zmiennych, ale nic nie stoi na przeszkodzie, aby w jednej liście umieścić wartości zupełnie różnych typów. Zmienne tworzymy, zapisując pomiędzy nawiasami kwadratowymi („[" i „]") elementy, które chcemy, aby nasza lista przetrzymywała. W interpreterze wygląda to tak:

```
>>> lista = [0, 2, 4, 6, 8]
```

Listy można również tworzyć z już istniejących zmiennych, a więc można też nową listę stworzyć tak:

```
>>> a = 0

>>> b = 2

>>> c = 4

>>> d = 6

>>> e = 8

>>> lista = [a, b, c, d, e]
```

Tak stworzona zmienna „lista" jest typu „lista" i przechowuje wszystkie wartości, które wpisaliśmy pomiędzy nawiasami. Możemy to zweryfikować, drukując ją:

```
>>> print(lista)

[0, 2, 4, 6, 8]
```

W ten sposób interpreter wydrukował nam całą listę. A co, jeśli chcemy zobaczyć tylko jeden jej element? Na przykład pierwszy? Wtedy trzeba użyć tak zwanego indeksu listy:

```
>>> print(lista[0])

0

>>> print(lista[4])

8
```

Dlaczego wpisaliśmy zero, skoro chcieliśmy dostać pierwszy element listy? Zrobiliśmy tak, ponieważ w Pythonie, jak w większości języków programowania wszystko numerowane jest od zera. Zatem pierwszy element listy ma numer zero, a przez to ostatni ma numer cztery, a nie pięć jak mogłoby się wydawać. Jest to niezwykle ważne i należy o tym zawsze pamiętać podczas pracy z listami. Aby usunąć jakiś element z listy, należy użyć instrukcji „del

KAPITAŁ LUDZKI NARODOWA STRATEGIA SPÓJNOŚCI
ORE
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt "Strategia Wolnych i Otwartych Implementacji jako innowacyjny model zainteresowania kierunkami informatyczno-technicznymi oraz wspierania uczniów i uczennic w kształtowaniu kompetencji kluczowych" współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

” od angielskiego słowa „delete”, czyli właśnie usuwać.

```
>>> del lista[2]

>>> print(lista)

[0, 2, 6, 8]
```

Jak widać, czwórka została wyrzucona z naszej listy. Warto w tym momencie zaznaczyć, że po takim działaniu nasza lista uległa skróceniu, a więc i numeracja indeksów uległa zmianie. W tej chwili ostatni element tablicy ma numer 3. Użycie numeru 4 tak jak wcześniej spowoduje błąd w programie. Aby dodać element na koniec listy, robimy tak:

```
>>> lista.append(10)

>>> print(lista)

[0, 2, 6, 8, 10]
```

Używamy tutaj funkcji „append”, której jako parametr podajemy wartość, jaką chcemy dodać do naszej listy. Ale jak widać, nie wywołujemy funkcji tak, jak to robiliśmy do tej pory. Wynika to z tego, że funkcja „append” jest prywatną funkcją naszej zmiennej. Takie prywatne funkcje wywołuje się, dostawiając do nazwy zmiennej kropkę i później już normalnie podając nazwę funkcji i jej parametry. Jeśli chcemy dodać element w konkretne miejsce na liście, musimy znać numer elementu, przed który chcemy wstawić nową wartość. Zatem, jeśli chcemy dodać z powrotem czwórkę do naszej listy, na jej stare miejsce musimy to zrobić tak:

```
>>> lista.insert(2, 4)

>>> print(lista)

[0, 2, 4, 6, 8, 10]
```

Pierwszy parametr funkcji „insert” to indeks miejsca, przed które chcemy wstawić nowy element, a drugi to element, który chcemy dodać do naszej listy. Jak widać, listy są bardzo elastycznym typem danych, dzięki czemu nadają się do bardzo wielu zastosowań. Programując w Pythonie, bardzo często będziecie się z nimi stykać. Innym typem zmiennych, który może przetrzymywać więcej niż jedną wartość, są tak zwane krotki (angielska nazwa to „tuple”). Krotki są bardzo podobne do list z jedną bardzo ważną różnicą – dane w krotkach są niezmienne. Raz utworzona krotka już do końca ma takie elementy, jakie zostały podane przy jej stworzeniu. Krotki tworzy się tak samo jak listy, tylko zamiast nawiasów kwadratowych używa się zwykłych, tak samo jak przy funkcjach:

```
>>> krotka = (1, 3, 5, 7, 9)

>>> print(krotka)

(1, 3, 5, 7, 9)

>>> a = 1

>>> b = 3

>>> c = 5

>>> d = 7

>>> e = 9

>>> krotka = (a, b, c, d, e)
```


KAPITAŁ LUDZKI NARODOWA STRATEGIA SPÓJNOŚCI
ORE
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt “Strategia Wolnych i Otwartych Implementacji jako innowacyjny model zainteresowania kierunkami informatyczno-technicznymi oraz wspierania uczniów i uczennic w kształtowaniu kompetencji kluczowych” współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

```
>>> print(krotka)

(1, 3, 5, 7, 9)
```

W przypadku krotek, tak samo jak i w listach, możemy drukować pojedyncze elementy, również numerując je od zera:

```
>>> print(krotka[3])

7
```

I to w zasadzie wszystko, co można zrobić z krotką. Wydaje się, że to niewiele, ale krotki są bardzo przydatnym typem danych wszędzie tam, gdzie kolejność elementów ma znaczenie, a bardzo nie chcemy, żeby program mógł zmieniać zawartość naszej zmiennej. Trzecim typem zmiennych mogących posiadać więcej niż jedną wartość są zbiory (z angielskiego „set”). Nie są one używane zbyt często, ale posiadają pewną bardzo ciekawą właściwość, która może być bardzo pomocna przy rozwiązywaniu niektórych problemów – ich zawartość się nie powtarza. W liście jedna wartość może wystąpić wiele razy, można np. stworzyć listę, która będzie zawierała siedem elementów i wszystkie to będzie cyfra 3. W zbiorach jest to niemożliwe. Dodatkową cechą zbiorów jest to, że są nieuporządkowane, a co za tym idzie nie możemy drukować dowolnego ich elementu w taki sposób jak w przypadku list czy krotek. Można za to dodawać i usuwać elementy ze zbiorów. Jednak jeśli spróbujemy dodać do zbioru element, który już się w nim znajduje to nasz zbór nie zostanie zmieniony. Tworzyć zbiory możemy na dwa sposoby. Albo podając już istniejącą zmienną do funkcji tworzącej zbiór, albo podobnie jak listy i krotki tylko, że tym razem używając nawiasów klamrowych („{” i „}”):

```
>>> zbior = set(krotka)

>>> print(zbior)

{1, 3, 9, 5, 7}

>>> zbior = set(lista)

>>> print(zbior)

{0, 2, 4, 6, 8, 10}

>>> zbior = {0, 2, 4, 6, 8, 10}

>>> print(zbior)

{0, 2, 4, 6, 8, 10}

>>> zbior.add(1)

>>> zbior.add(3)

>>> print(zbior)

{0, 1, 2, 3, 4, 6, 8, 10}

>>> zbior.add(3)

>>> zbior.add(3)

>>> zbior.add(3)
```


KAPITAŁ LUDZKI NARODOWA STRATEGIA SPÓJNOŚCI
ORE
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt “Strategia Wolnych i Otwartych Implementacji jako innowacyjny model zainteresowania kierunkami informatyczno-technicznymi oraz wspierania uczniów i uczennic w kształtowaniu kompetencji kluczowych” współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

```
>>> print(zbior)

{0, 1, 2, 3, 4, 6, 8, 10}

>>> zbior.remove(10)

>>> print(zbior)

{0, 1, 2, 3, 4, 6, 8}
```

Ostatnim z nowych typów danych są słowniki (od angielskiego „dictionary”, w Pythonie używana jest skrócona forma „dict”). Słowniki są zupełnie innym typem danych od dotychczas prezentowanych. Tak samo jak w normalnym, papierowym słowniku składają się one z par elementów. Pierwszy element jest nazywany kluczem, a drugi wartością. Tak jak w papierowym słowniku gdzie jednemu elementowi (kluczowi), po którym przeszukujemy słownik, jest przypisana jakaś wartość – najczęściej odpowiednik klucza w innym języku, lub objaśnienie klucza. Kolejność elementów w słownikach nie ma znaczenia, ponieważ dane w nich i tak wyszukuje się po kluczu. Słowniki są również modyfikowalne, czyli można do nich dodawać nowe elementy i usuwać już istniejące. Jedyna zasada to taka, że klucze nie mogą się powtarzać, dodanie nowego elementu z takim samym kluczem jak jeden z już istniejących w słowniku, powoduje nadpisanie starej pary klucz-wartość nową. W słowniku wszystko może być kluczem, tak samo jak i wszystko może być wartością. Możliwe są zatem takie konstrukcje:

```
>>> na_slowa = {1:'jeden', 2:'dwa', 3:'trzy', 4:'cztery', 5:'pięć'}

>>> print(na_slowa)
{1:'jeden', 2:'dwa', 3:'trzy', 4:'cztery', 5:'pięć'}

>>> na_cyfry = {'jeden':1, 'dwa':2, 'trzy':3, 'cztery':4, 'pięć':5}

>>> print(na_cyfry)

{'jeden':1, 'dwa':2, 'trzy':3, 'cztery':4, 'pięć':5}
```

Jak widać słowniki, tak samo jak zbiory, tworzymy przy użyciu nawiasów klamrowych, ale co innego pomiędzy nie wpisujemy. Tak jak pisałem są to pary, elementy par oddzielamy od siebie dwukropkami. Wartość przypisaną do klucza sprawdzamy w następujący sposób:

```
>>> na_slowa[3]

'trzy'

>>> na_cyfry['trzy']

3
```

Usuwanie elementów ze słownika wygląda tak samo jak w przypadku list, ale zamiast indeksu elementu podajemy klucz, który chcemy usunąć:

```
>>> del na_slowa[3]

>>> print(na_slowa)

{1: 'jeden', 2: 'dwa', 4: 'cztery', 5: 'pięć'}
```


KAPITAŁ LUDZKI NARODOWA STRATEGIA SPÓJNOŚCI
ORE
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt “Strategia Wolnych i Otwartych Implementacji jako innowacyjny model zainteresowania kierunkami informatyczno-technicznymi oraz wspierania uczniów i uczennic w kształtowaniu kompetencji kluczowych” współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

```
>>> del na_cyfry['trzy']

>>> print(na_cyfry)

{'cztery': 4, 'dwa': 2, 'jeden': 1, 'pięć': 5}
```

Aby dodać nowy element do słownika, po prostu wpisujemy w nawiasy kwadratowe klucz, którego chcemy użyć i przypisujemy mu wartość:

```
>>> na_slowa[3] = 'trzy'

>>> print(na_slowa)

{1: 'jeden', 2: 'dwa', 3: 'trzy', 4: 'cztery', 5: 'pięć'}

>>> na_cyfry['trzy'] = 3

>>> print(na_cyfry)

{'cztery': 4, 'dwa': 2, 'jeden': 1, 'pięć': 5, 'trzy': 3}
```

Zadania

- **Napisz program, który stworzy tablicę z dziesięcioma kolejnymi cyframi.**

KAPITAŁ LUDZKI NARODOWA STRATEGIA SPÓJNOŚCI
UNIA EUROPEJSKA EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt “Strategia Wolnych i Otwartych Implementacji jako innowacyjny model zainteresowania kierunkami informatyczno-technicznymi oraz wspierania uczniów i uczennic w kształtowaniu kompetencji kluczowych” współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.