Język skryptowy lab3

Tablice

Tablice w Pythonie nie są takie jak tablice znane z Javy lub C ale bardziej przypominają listy. Tablica jest strukturą danych przechowującą wartości tego samego typu. Jest to główna różnica między tablicami i listami w pythonie. Aby móc używać tablic w pythonie trzeba zaimpotrować ze standardowych bibliotek pythona moduł `array` dlatego, że tablice w pythonie nie są uznawane za podstawowe typy danych jakimi są np watości całkowite czy ciągi znaków

```
from array import *
```

Poniżej przedstawiono przykładową deklarację tablicy

```
nazwa_tablicy = array(kod_typu_przechowywanych_danych, [inicjalizacja_wartościami])
```

Parametr (typ przechowywanych danych)	Opis
b	Reprezentuje liczbę całkowitą ze znakiem o rozmiarze 1
B	Reprezentuje liczbę całkowitą bez znaku o rozmiarze 1
c	Reprezentuje znak wielkości 1 bajtu
u	Reprezentuje znak Unicode o rozmiarze 2 bajty
h	Reprezentuje liczbę całkowitą ze znakiem o rozmiarze 2 bajty
H	Reprezentuje liczbę całkowitą bez znaku o rozmiarze 2 bajty
i	Reprezentuje liczbę całkowitą ze znakiem o rozmiarze 2 bajty
I	Reprezentuje liczbę całkowitą bez znaku o rozmiarze 2 bajty
w	Reprezentuje znak Unicode o wielkości 4 bajty
l	Reprezentuje liczbę całkowitą ze znakiem o rozmiarze 4 bajtów

Parametr (typ przechowywanych danych)	Opis
L	Reprezentuje liczbę całkowitą bez znaku o wielkości 4 bajty
f	Reprezentuje zmiennoprzecinkowy rozmiar 4 bajtów
d	Reprezentuje zmiennoprzecinkowy rozmiar 8 bajtów

Pojedyncze elementy pobiera się przy pomocy odwołania do indeksu na którym znajduje się element, tablice indeksowane są od 0

```
from builtins import print

my_array = array('i', [1, 2, 3, 4, 5])
print(my_array[1])
print(my_array[2])
print(my_array[0])
print()

my_array = array('i', [1, 2, 3, 4, 5])
for i in my_array:
    print(i)
```

Dodawanie kolejnego indeksu do tablicy używając metody `append`

```
my_array = array('i', [1, 2, 3, 4, 5])
my_array.append(6)
```

Dodawanie kolejnego indeksu do tablicy używając metody `insert` . Metoda `insert` wstawia wartość na odpowiednim indexie w tablicy

```
my_array.insert(0, 0)
```

Wstawiana wartość umieszczana jest w odpowiednie miejsce w tablicy przesuwając dalszą część tablicy o jeden indeks.

W pythonie jest możliwość rozszerzania tablicy przez więcej niż jedną wartość przy użyciu metody `extend`

```
my_array = array('i', [1, 2, 3, 4, 5])
my_extend_array = array("i", [6, 7, 8, 9, 10])
my_array.extend(my_extend_array)
print(my_array)
```

Do tablicy można również dodawać elementy znajdujące się w liście przy użyciu metody `fromlist`

```
my_array = array('i', [1, 2, 3, 4, 5])
c = [11, 12, 13]
my_array.fromlist(c)
```

Usuwanie elementu z listy z użyciem metody `remove'

```
my_array = array('i', [1, 2, 3, 4, 5])
my_array.remove(4)
```

Usuwanie ostatniego elementu z tablicy metodą `pop`

```
my_array = array('i', [1, 2, 3, 4, 5])
my_array.pop()
```

Sprawdzenie elementu przy użyciu metody `index` zwraca pierwszy indeks elementu na którym znajduje się podana wartość

```
my_array = array('i', [1, 2, 3, 4, 5])
print(my_array.index(5))

my_array = array('i', [1, 2, 3, 3, 5])
print(my_array.index(3))
```

Użycie metody `reverse` powoduje odwrócenie tablicy na rzecz której została wywołana ta metoda

```
my_array = array('i', [1, 2, 3, 4, 5])
my_array.reverse()
print(my_array)
```

Możliwe jest też sprawdzenie pod jakim adresem w pamięci komputera znajduje się dana tablica oraz jaką ma długość

```
my_array = array('i', [1, 2, 3, 4, 5])
print(my_array.buffer_info())
```

Sprawdzanie liczby wystąpień danego element w tablicy

```
my_array = array('i', [1, 2, 3, 3, 5])
print(my_array.count(3))
```

Konwersja tablicy do łańcucha znaków

```
print(my_array1)
print(my_array1.tounicode())
```

```
word = "spam"
my_array = array(str('u'), [])
my_array.extend(list(word))
print(my_array.tounicode())
```

Konwersja tablicy do listy z użyciem metody `tolist`

```
my_array = array('i', [1, 2, 3, 4])
list = my_array.tolist()
```

Łączenie łańcuchów znaków z tablicą znaków z użyciem metody `fromstring`

```
my_array = array('u', ['s', 'p', 'a', 'm'])
my_array.extend(" egg")
print(my_array.tounicode())
```

None

Obiekt `None` jest używany do reprezentacji braku wartości, jest on podobny do `null'a` w innych językach programowania Tak jaki inne puste wartości jak np. 0, [] i pusty ciąg znaków, jest on fałszem gdy skonwertujemy go na wartość logiczną: print("none") print(bool(None)) print("None")

Obiekt `None` jest zwracany przez funkcję gdy bezpośrednio nie zwraca ona niczego konkretnego.

Słowniki

Słowniki są strukturą danych która służy do mapowania dowolnych kluczy na wartość. Słowniki mogą być indeksowane tak samo jak listy przy użyciu kwadratowych nawiasów zawierających klucz

```
ages = {"I": 78, "You": 20, "Him": 24}
print(ages["I"])
print(ages["Him"])
```

> Każdy element w słowniku reprezentowany jest przez parę `klucz:wartość`

Próba pobrania elementu używając klucza który nie istnieje w słowniku zwróci błąd `KeyError` .

```
colors = {
    "red": [255, 0, 0],
    "green": [0, 255, 0],
    "blue": [0, 0, 255]
}
print(colors["red"])
print(colors["yellow"])
print(colors["green"])
```

> Jak widać słowniki mogą przechowywać jako wartości dane dowolnych typów

Tylko niezmienne obiekty mogą być używane jako klucze dla słowników, czyli takie których nie można zmienić. Do tej pory jedynymi możliwymi do zmienienia poznanymi obiektami są listy i tablice. Próba użycia zmiennego obiektu do słowa kluczowego powoduje błąd `TypeError`

```
bad_dict = {[1, 2, 3]: "one two three", }
```

Jak w listach klucze słownika mogą posiadać wartości różnych typów.

```
squares = {1: 1, 2: 4, 3: "spam", 4: 16}
squares[8] = 64
squares[3] = 9
squares[5] = 25
```

Do weryfikacji czy klucz znajduje się już w słowniku można wykorzystać operatory `in` i `not in` tak jak dla list.

```
nums = {1: "one",
        2: "two",
        3: "three"}
print(1 in nums)
print("three" in nums)
print(4 not in nums)
```

Metoda `get` służy do pobierania elementów ze słownika. W przypadku gdy podany argument(klucz) nie zostanie znaleziony w słowniku metoda zwróci obiekt `None`

```
pairs = {
    "orange": [2, 3, 40], 0: "spam", True: False, None: "True", 2: "apple"}
print(pairs.get("orange"))
print(pairs.get(7))
print(pairs.get(2))
print(pairs.get(True))
```

```
print(pairs.get(1235, "not in dictionary"))
print(len(pairs))
```

Odczytywanie danych ze słownika może odbywać się z wykorzystaniem metody `get`

```
for i in pairs:
    print(str(i) + "\t" + str(pairs.get(i)))
```

lub odwołania przez podanie nazwy słownika i odpowiedniego klucza w nawiasach kwadratowych.

```
for i in pairs:
    print(i, pairs[i])
 oraz w krótszej konstrukcji
print()
print([(key, pairs[key]) for key in pairs])
```

Scalanie słowników

```
fish = {'name': "Nemo", 'hands': "fins", 'special': "gills"}
dog = {'name': "Clifford", 'hands': "paws", 'color': "red"}
 #Python 3.5+
fishdog = {**fish, **dog}
print(fishdog)

 #Python 2.x
from itertools import chain

print(dict(chain(fish.items(), dog.items())))
```

Konstruktor `dict()` może być użyty do tworzenia słowników z argumentów będących kluczem z przypisana do niego wartością lub z pojedynczej iteracji par klucz-wartość lub z pojedynczego słownika i argumentów słów kluczowych

```
print(dict(a=1, b=2, c=3))
print(dict([('d', 4), ('e', 5), ('f', 6)]))
print(dict([('a', 1)], b=2, c=3))
print(dict({'a': 1, 'b': 2}, c=3))
```

Krotki `Tuples` są podobne do list z wyjątkiem tego, że nie można ich zmieniać tworzy się je z wykorzystaniem nawiasów okrągłych words = ("spam", "eggs", "saussages") Dostęp do elementu uzyskuje się przez podanie nazwy i numeru parametru objętego w nawiasy kwadratowe

Próba podmiany któregoś elementu na inny skutkuje błędem `TypeError` . Krotki podobnie jak słowniki i listy mogą się nawzajem zagnieżdżać. Krotki można też tworzyć bez użycia nawiasów

oddzielając wartości przecinkami.

```
my_tuple = "one", "two", "three"
print(my_tuple[0])
```

Krotki są szybsze niż listy lecz nie można zmieniać ich zawartości

Wycinki list

Wycinki listy zapewniają bardziej zaawansowany sposób pobierania wartości z listy. Podstawowe krojenie list polega na indeksowaniu listy dwiema liczbami całkowitymi rozdzielonymi dwukropkami. Zwraca nową listę zawierającą wszystkie wartości ze starej listy między indeksami

```
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print(squares[2:6])
print(squares[3:8])
print(squares[0:1])
```

> Pierwszy indeks podany w zakresie wycinka listy jest zawarty w wyniku, ale drugi już nie jest

Jeśli pominięto pierwszą liczbę w wycinku listy, uważa się ją za początek listy, jeśli pominięto drugą liczbę, uważa się, że jest ona końcem

```
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print(squares[:7])
print(squares[7:])
```

> Wycinanie można też wykonać na krotkach Wycinanie z list może się również odbywać z użyciem trzeciej liczby reprezentującej krok dla pobranych wartości z listy

```
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print(squares[::2])
print(squares[2:7:3])
```

Zastosowanie kroku *-1* spowoduje pobranie listy od jej końca

Wzorce list są użytecznym sposobem szybkiego tworzenia list, których zawartość jest zgodna z prostą regułą

```
cubes = [i ** 3 for i in range(5)]
print(cubes)
```

Wzorzec listy może również zawierać instrukcję `if` , która wymusi warunek na wartościach na liście

```
evens = [i ** 2 for i in range(10) if i ** 2 % 2 == 0]
print(evens)
```

Przydatne metody dla łańcuchów znaków

:small_blue_diamond: join - dołącza listę ciągów do innego ciągu znaków jako separator :small_blue_diamond: replace - zamienia jeden wycinek napisu innym :small_blue_diamond: startswith and endswith - określa, czy istnieje podłańcuch odpowiednio na początku i na końcu łańcucha :small_blue_diamond: lower upper - nazmiana napisów na małe i wielkie litery :small_blue_diamond: split - działa podobnie do `join` z pewnym separatorem w liście :small_blue_diamond: format - podstawia argumenty w ciągu znaków

```
print(",".join(["spam", "eggs", "spam"]))
print("Hello Me".replace("ME", "world"))
print("This is a sentence.".startswith("This"))
print("This is a sentence.".endswith("sentence"))
print("This is a sentence.".upper())
print("This is a sentence.".lower())
print("This is a sentence.".split(","))

nums = [4, 5, 6]
msg = "Numbers: {0} {1} {2}".format(nums[0], nums[1], nums[2])
print(msg)
print("{x},{y}".format(x=2, y=3))
```

Często używane w instrukcjach warunkowych, funkcje `all` i `any` przyjmują listę jako argument zwracają wartość `True` Jeśli odpowiednio wszystkie lub który kolwiek z jej argumentów ocenia się jako `True` . Funkcja `enumerate` może być użyta do iteracji przez wszystkie wartości i indeksy listy jednocześnie.

```
nums = [55, 44, 33, 22, 11]

if all([i > 5 for i in nums]):
    print("Wszystkie większe od 5")
if any([i % 2 == 0 for i in nums]):
    print("Przynajmniej jedna parzysta")

for v in enumerate(nums):
    print(v)
```

Zadania do wykonania

:one: Wypróbuj kody z *listingów* znajdujących się w instrukcji i sprawdź ich działanie.

:two: Napisz skrypt wypełniający tablicę znakami, a następnie wyświetl znaki w kolejności odwrotnej do wprowadzania. Dane wprowadzane z klawiatury.

:three: Wypełniający tablicę liczbami losowymi rzeczywistymi z przedziału (-5,5). Wartość tablicy zapisz do pliku *result.txt*

:four: Napisz funkcję tworzącą tablicę dwuwymiarową *(5x5)* która zostanie wypełniona kwadratami liczb z komórek z wiersza wcześniejszego. Pierwszy wiersz wypełniony wartościami 2,3,4,5,6. Do utworzenia tablicy dwuwymiarowej wykorzystaj bibiotekę NumPy. Bibliotekę można zainstalować przy pomocy polecenia:

```
python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose
```

:five: Napisz funkcję, która jako parametr przyjmuje lokalizację pliku tekstowego który zawiera dowolny tekst i zwraca histogram znaków występujących w tym napisie (czyli pary znak-liczba wystąpień). Wynikiem powinien być słownik. Przykład:

`>>> histogram("document.txt")` dokument zawiera tekst: *Ala ma kota* `{'t': 1, 'a': 3, 'l': 1, 'A': 1, 'k': 1, 'm': 1, 'o': 1}`

:six: Napisz następujące funkcje niezbędne do implementacji gry w pokera pięciokartowego dobieranego:

1. deck() - zwraca listę reprezentującą talię kart w kolejności od najmłodszej do najstarszej. Każda karta posiada 2 atrybuty, będące łańcuchem tekstowym:

- rangę - możliwe wartości: `'2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'D', 'K', 'A'` (karty od 2 do 10 oraz walet, dama, król, as)
- kolor - możliwe wartości:

  :small_blue_diamond: `c` - ♣ trefl (clubs)

  :small_blue_diamond: `d` - ♦ karo (diamonds)

  :small_blue_diamond: `h` - ♥ kier (hearts)

  :small_blue_diamond: `s` - ♠ pik (spades)

  Każdym elementem listy powinna być krotka, będąca parą (ranga, kolor). Przykładowo as pik:

  

  ---

  reprezentowany będzie jako ('A', 's'). Lista powinna zawierać 52 elementy (13 rang * 4 kolory).

2. shuffle_deck(deck) - przyjmuje listę kart, zwraca karty potasowane (permutacja). Skorzystaj z:

3. deal(deck, n) - przyjmuje talię kart (deck) oraz liczbę graczy (n), zwraca n-elementową listę 5-elementowych list z kartami rozdanymi graczom. Każda 5-elementowa lista kart gracza zawiera 5 krotek reprezentujących kartę.

:exclamation: zadania 2-6 mają zostać dodane na GitHuba :exclamation: