ПРОГРАМУВАННЯ (PYTHON)

КОЛЕКЦІЇ

Вбудовані колекції (контейнери)

- Список (list) - змінюваний, упорядкований
- Кортеж (tuple) - незмінюваний, упорядкований
- Словник(dict) - змінюваний, неупорядкований
- Множина (set) - змінюваний, неупорядкований

СПИСКИ (тип list)

Список (тип list) — упорядкована (не відсортована!) змінювана послідовність (колекція) об'єктів, які називаються елементами списку.

- немає необхідності наперед вказувати кількість елементів (довжину) списку;
- елементами списку можуть бути значення будь-яких типів;
- тип елементів одного й того самого списку може бути різний
- списки змінювані (mutable), тобто підтримують додавання, вилучення та зміну елементів.

Літералом списку в Python є послідовність об'єктів, записаних через кому у квадратних дужках:

```
[3., 3.1, 3.14, 3.142, 3.1416]

['зелений', 'жовтий', 'червоний']

[1, ['Yes', 'No'], 2.1, 'Maybe']

[]
```

Список може бути записаний у кілька рядків:

```
rainbow = ['red', 'orange',
           'yellow', 'green', 'blue',
           'indigo', 'violet'
]

rainbow = [
           'red', 'orange',
           'yellow', 'green', 'blue',
           'indigo', 'violet'
           ]
```

Якщо елементом списку є вираз, то інтерпретатор обчислюватиме значення цього виразу:

```
x = 10
print([x-1, x**2-1, x**3-1])
```

```
[9, 99, 999]
```

Списки можна об'єднувати, використовуючи оператор +:

```
pi = [3., 3.1, 3.14, 3.142, 3.1416]
pi = pi + [3.14159, 3.141593]
pi += [3.1415927]
print(pi)
```

```
[3.0, 3.1, 3.14, 3.142, 3.1416, 3.14159, 3.141593, 3.1415927]
```

Оператор * надає можливість розмножувати список однаковими значеннями:

```
print([1, 'один']*3)
```

```
[1, 'один', 1, 'один', 1, 'один']
```

Список також може бути утворений з будь-якої послідовності елементів. Для цього потрібно передати цю послідовність функції list() як аргумент:

```
s = 'Це рядок.ру'
l = list(s)
print(l)
print(list(range(10)))
```

```
['Ц', 'е', ' ', 'р', 'я', 'д', 'о', 'к', '.ру']
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Кожен елемент списку має **індекс** — унікальне ціле число, яке однозначно вказує на порядковий номер елемента у списку.

Індекси можна відраховувати **з початку** списку (0, 1, 2, ...), а можна **з кінця** (-1, -2, -3, ...):

```
spysok = [1, ['Yes', 'No'], 2.1, 'Maybe']
print(spysok[1]) # ['Yes', 'No']
print(spysok[0], spysok[2], spysok[-2]) # 1 2.1 2.1
```

Щоб отримати доступ до елементів вкладеного списку, індекси записуються послідовно:

```
print(spysok[1][0], spysok[1][1]) # Yes No
```

Оскільки список — змінюваний тип даних, то його елементи можна змінювати:

```
spysok = [1, ['Yes', 'No'], 2.1, 'Maybe']
spysok[1] = 'Maybe'
print(spysok) # [1, 'Maybe', 2.1, 'Maybe']
```

Якщо елемента з вказаним індексом не існує, буде ініційована помилка:

```
print(spysok[4]) # IndexError: list index out of range
```

Зрізи, як і у випадку рядків, дають змогу отримати певну підмножину елементів списку. При цьому сам список не змінюється — результат зрізу записується у новий список.

```
rainbow = ['red', 'orange', 'yellow',
           'green', 'blue', 'indigo', 'violet']

print(rainbow[:])
print(rainbow[:3])
print(rainbow[2:4])
print(rainbow[1::2])
print(rainbow[::-1])
```

Функції для роботи зі списками

Функція len() повертає кількість елементів списку:

```
rainbow = ['red', 'orange', 'yellow',
           'green', 'blue', 'indigo', 'violet']
len(rainbow)                                  # 7
len([1, ['Yes', 'No'], 2.1, 'Maybe'])        # 4
len([])                                       # 0

numbers = [1, 2, 3]
sum(numbers)     # 6
min(numbers)     # 1
max(numbers)     # 3
```

Методи списків (змінюють список!)

```
list_ = [1, 2, 'three']
list_.pop() # вилучає елемент з кінця списку: [1, 2]
list_.append(3) # додає елемент в кінець списку: [1, 2, 3]
list_.pop(1) # вилучає елемент з індексом 1: [1, 3]
list_.insert(1, 2) # вставляє 2 на 1-е місце: [1, 2, 3]
list_.extend([2, 4]) # додає в кінець: [1, 2, 3, 2, 4]
list_.remove(2) # вилучає перше входження 2: [1, 3, 2, 4]
list_.sort() # сортує елементи списку: [1, 2, 3, 4]
list_.reverse() # в зворотньому порядку: [4, 3, 2, 1]
list_.count(3) # повертає кількість входжень елемента 3: 1
list_.index(3) # повертає індекс елемента 3: 1
list_.clear() # очищує список: []
```

```
s = '-'.join(['a','b','c'])
# утворює зі списку ['a','b','c'] рядок'a-b-c'


l = s.split('-')
# утворює з рядка 'a-b-c' список ['a','b','c']
```

Перевірка на входження

```
trafficLight = ['зелений', 'жовтий', 'червоний']

color = input('Введіть колір: ')

if color in trafficLight:
    print(color, 'є серед кольорів світлофору')
else:
    print('Цього кольору немає серед кольорів світлофору')
```

Поелементний обхід списку

```
cities = ['Kyiv', 'London', 'New York']

for city in cities:
    print(city)

for i in range(len(cities)):
    print(f'{i+1}. {cities[i]}')

for i, city in enumerate(cities):
    print(f'{i+1}. {city}')
```

Заповнення списків під час виконання програми

```
s = 'Київ – столиця України'

# створюємо порожній список для унікальних символів
uniqueSymbols = []

for symbol in s:              # обходимо рядок s посимвольно
    if symbol not in uniqueSymbols:
        uniqueSymbols.append(symbol)
print(uniqueSymbols)
```

```
['К', 'и', 'ї', 'в', ' ', '–', 'с', 'т', 'о', 'л', 'ц', 'я',
'У', 'к', 'р', 'а', 'н']
```

Генератори списків

Генератори списків призначенні для автоматизації створення списків, елементи яких утворюються згідно з певними правилами.

Загальний синтаксис генератора списку:

```
[<вираз> for <змінна> in <колекція> if <умова>]
```

```
list_1 = [n**2 for n in range(10,100) if n%2 == 0]
list_2 = [n**2 for n in range(10,100,2)]
list_3 = [x for x in range(10, 100) if '1' not in str(x)]
```

```
s = sum([_ for _ in range(0, 10000)])
s = sum(_ for _ in range(0, 10000))


import sys

mylist = (x for x in range(0, 10000))
print(sys.getsizeof(mylist))     # 112 (байт)

mylist = [x for x in range(0, 10000)]
print(sys.getsizeof(mylist))     # 87616 (байт)
```

Змінюваність списків (як і інших змінюваних типів) має побічний ефект, який називають накладенням імен (aliasing).

```
a = [1, 2, 3]
b = a
a.append(4)
print(b)          # [1, 2, 3, 4]

# Щоб уникнути цього:
b = a[:]
b = list(a)
b = copy.deepcopy(a)  # вимагає import copy
```

Оператор розпаковування (unpacking operator)

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

print(list_) # [1, 2, 3]

print(*list_) # 1 2 3
```

```
# Сума елементів рядків і стовпців матриці

matrix = [[1,2,3], [4,5,6], [7,8,9]]

rowsum = []
columnsum = [0] * len(matrix[0])

for row in matrix:
    rowsum.append(sum(row))

for index in range(len(row)):
    columnsum[index] += row[index]

print(rowsum, columnsum)
```

КОРТЕЖІ

Кортеж (тип tuple) — це, як і список, впорядкована послідовність (колекція) елементів. Але, на відміну від списків, кортежі незмінювані.

Літералом кортежа в Python є послідовність об'єктів, записаних через кому у круглих дужках:

```
('зелений', 'жовтий', 'червоний')
(1, ['Yes', 'No'], 2.1, 'Maybe')
() # це не порожній кортеж!!!
```

Оскільки круглі дужки в Python використовуються також для групування виразів, то, щоб уникнути неоднозначності, кортеж з одного елемента повинен містити кому після цього елемента:

```
(1,) # кортеж з одного елемента
```

Кортеж також може бути утворений з будь-якої послідовності елементів. Для цього потрібно передати цю послідовність функції tuple() як аргумент:

```
l = [1, 2, 3, 4, 5]

t = tuple(l)

print(t)
```

```
empty = tuple() # empty – порожній кортеж
```

Оскільки кортеж — незмінюваний тип даних, то спроба зміни його елемента призведе до помилки:

```
rainbow[2] = 'grey'
TypeError: 'tuple' object does not support item assignment
```

Проте, змінювати елементи змінюваних типів всередині кортежа допускається:

```
t = (1, ['Yes', 'No'], 2.1, 'Maybe')
t[1][1] = 'Yes'
print(t) # (1, ['Yes', 'Yes'], 2.1, 'Maybe')
```

Генерування кортежів:

```
t = tuple(i for i in range(5000))
```

СЛОВНИКИ

Словник (змінюваний тип dict) — невпорядкована колекція елементів, які організовані асоціативно.

Елементи словника називають записами.

Запис - пара ключ-значення.

Доступ до значення конкретного елемента словника здійснюється не за його позицією (порядковим номером), як у випадку списків і кортежів, а за ключем.

Літералом словника в Python є послідовність пар ключ-значення, записаних через кому у фігурних дужках, при цьому ключ відділений від значення двокрапкою.

```
person = {'name': 'Роман', 'borned': 1976}

squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
           7: 49, 8: 64, 9: 81, 10: 100}

emptyDict = {}       # порожній словник
```

Ключами зазвичай є цілі числа або рядки, але ключем може бути будь-який незмінюваний тип.

Значення можуть бути якого завгодно типу.

```
pair = {
        'she': {'name': 'Alice', 'from': 'USA'},
        'he': {'name': 'Bob', 'from': 'Canada'}
       }
```

Якщо ключем або значенням словника є вираз, то інтерпретатор обчислюватиме значення цього виразу:

```
dots1 = {'одна': '.', 'дві': '.'*2, 'три': '.'*3}
print(dots1)

# буде надруковано {'одна': '.', 'дві': '..', 'три': '...'}

dots2 = {'.': 1, '.'*2: 2, '.'*3: 3}
print(dots2)

# буде надруковано {'.': 1, '..': 2, '...': 3}
```

Доступ до значень елементів словника здійснюється подібно до списків, лише замість індекса у квадратних дужках вказується ключ:

```
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36,
           7: 49, 8: 64, 9: 81, 10: 100}

pair = {'she': {'name': 'Alice', 'from': 'USA'},
        'he': {'name': 'Bob', 'from': 'Canada'}}

print(squares[6])                    # 36
print(pair['he']['from'])            # Canada
```

Словник — змінюваний тип:

```
rectangle = {'length': 2, 'width': 5}
rectangle['width'] = 4
print(rectangle) # {'length': 2, 'width': 4}

rectangle['square'] = rectangle['length'] * rectangle['width']
print(rectangle) # {'length': 2, 'width': 4, 'square': 8}
```

Якщо звернутися до елемента словника з неіснуючим ключем:

```
print(rectangle['perimeter']) # KeyError: 'diagonal'
```

Такої помилки можна уникнути:

```
if 'perimeter' in rectangle:
    print(rectangle['perimeter'])
```

Метод get() приймає два аргументи: 1) ключ; 2) значення, яке повертатиметься у випадку відсутності у словнику елемента з таким ключем:

```
print(rectangle.get('perimeter', None))
```

Функції та методи

len() - повертає кількість елементів словника:

```
points = {'Bob': 91, 'John': 60, 'Alice': 75}
print(len(points))          # 3
```

sorted() - повертає відсортований список ключів:

```
print(sorted(points)) # ['Alice', 'Bob', 'John']
```

```
someDict.pop(key) someDict.clear()
list(someDict.keys()) list(someDict.values())
list(someDict.items())
```

Поелементний обхід словника

```
# 1-й спосіб
for key in rectangle:
    print(f'{key} = {rectangle[key]}')

# 2-й спосіб
for key,value in rectangle.items():
    print(f'{key} = {value}')
```

```
length = 2
width = 4
square = 8
```

```
# Обчислення частоти входжень елементів у список
from random import randint
# генеруємо список зі ста 0 і 1:
bits = [randint(0,1) for i in range(100)]
freq = {} # створюємо порожній словник
for bit in bits: # обходимо список поелементно
    if bit not in freq: # якщо ключа немає у словнику,
        freq[bit] = 1 # то це перше входження
    else: # інакше
        freq[bit] += 1 # збільшуємо лічильник на 1
print(freq) # буде виведено словник типу {0: 58, 1: 42}
```

Генератори словників

Синтаксис:

```
{ключ: вираз for ключ in колекція if умова}
```

Генерування словника квадратів парних двоцифрових натуральних чисел:

```
d = {n: n**2 for n in range(10,100) if n%2 == 0}
d = {n: n**2 for n in range(10,100,2)}
```

МНОЖИНИ

Множина (змінюваний тип set) — невпорядкований набір (колекція) унікальних (такі, що не повторюються) елементів незмінюваних типів.

Літералом множини в Python є послідовність об'єктів, записаних через кому у фігурних дужках:

```
digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

rgbColors = {'red', 'green', 'blue'}

assorty = {1, 'one', (1, 1.)}

emptySet = set()
```

Оскільки елементами множини можуть бути тільки незмінювані типи, то спроба створити множину, яка містить список, словник або іншу множину, викличе помилку:

```
wrongSet = {1, [2, 3]}
```

```
TypeError: unhashable type: 'list'
```

Операції над множинами

```
rgbColors = {'red', 'green', 'blue'}
trafficColors = {'red', 'yellow', 'green'}

rgbColors - trafficColors  # різниця
rgbColors | trafficColors  # об'єднання
rgbColors & trafficColors  # перетин множин
rgbColors ^ trafficColors  # симетрична різниця (елементи, які
                           # є тільки в одній з множин)
rgbColors > trafficColors  # чи є надмножиною
rgbColors < trafficColors  # чи є підмножиною другої
'blue' in rgbColors                # перевірка на входження
```

Функції та методи

```
bits = [0, 0, 1, 0, 1]
s = set(bits) # {0, 1}
len(s) # 2
s1 = set('Арктика') # {'р', 'а', 'А', 'и', 'к', 'т'}
sorted(s1) # {'А', 'а', 'р', 'к', 'т', 'и'}

A.add(x) # додавання елемента x до множини A
A.discard(x) # вилучення елемента x з множини A
A.pop() # вилучення довільного елемента з множини A
A.clear() # очищення множини A
A.union(B) # об'єднання (аналог A | B)
A.intersection(B) # перетин (аналог A & B)
A.issubset(B) # чи A є підмножиною B (аналог A < B)
```

Генератори множин

Синтаксис:

```
{expression for item in collection if condition}
```

Генерування множини квадратів парних двоцифрових натуральних чисел:

```
s = {n**2 for n in range(10, 100) if n%2 == 0}
s = {n**2 for n in range(10, 100, 2)}
```

На відміну від списків, порядок розташування елементів у множині буде порушений.

any/all

any(x) повертає True, якщо хоча б один елемент колекції x не дорівнює False.

```
>>> any([False, 2, {}]) # True
```

all(x) повертає True, якщо усі елементи колекції x дорівнюють True.

```
>>> all([False, 2, {}]) # False
```

map/filter

```
def validate(x):
    return isinstance(x, (int, float))

list_ = [2, '5', False, [1], 2.5]

check = map(validate, list_) # <map object at 0x7efefb0d64f0>
check = tuple(check) # (True, False, True, False, True)

numbers = list(filter(validate, list_)) # [2, False, 2.5]
```

zip

```
a = '12'
b = (3, 4, 5)
print(list(zip(a, b))) # [('1', 3), ('2', 4)]


from random import randint as r

m = [[r(0, 9) for j in range(2)] for i in range(3)]

mt = list(map(list, list((zip(*m)))))
```

```
test = [1, 2, 3, 4, 2, 2, 3, 1, 4, 4, 4]
print(max(set(test), key=test.count))                    # 4


from collections import Counter
Counter(test).most_common(1)          # [4: 4]
```