

ПРАКТИЧНА РАБОТА 2

РОБОТА ЗІ СЛОВНИКАМИ ТА МНОЖИНАМИ В PHYTON

Словники в Python - це набір об'єктів, доступ до яких здійснюється по ключу, а не за індексом, як в списках або масивах. Ключем може бути будь-який незмінний об'єкт, наприклад, число, рядок або кортеж. Елементами словника можуть бути об'єкти довільного типу даних. Словники не є послідовностями, і багато функцій, які використовуються для роботи з послідовностями, до них не можна застосувати. Словники відносяться до змінюваних типів даних. Тому можна не тільки отримати значення по ключу, а й змінити його.

Словники:


- змінний неупорядкований набір пар ключ-значення;
- швидкий доступ до значення по ключу;
- швидка перевірка на входження ключа в словник.

Множина в Python - це неупорядкований набір унікальних об'єктів. Вони необхідні тоді, коли присутність об'єкта в наборі важливіше порядку або скільки разів даний об'єкт там зустрічається.

Множини змінювані і найчастіше використовуються для видалення дублікатів і всіляких перевірок на входження. Використовуючи множини, можна здійснювати перевірку приналежності, визначати, чи є дана множина підмножиною іншої множини, знаходити перетину множин і так далі.

Множини:

- змінний неупорядкований набір унікальних об'єктів;
- швидка перевірка на входження;
- математичні операції над множинами.

Якщо не встановлено Python 3.6 та среда розробки IDE (наприклад Anaconda, PyCharm Community Edition), в браузері перейдіть на сайт <https://edube.org/sandbox>, вибравши в налаштуванні через  емулятор Python (рис.1).

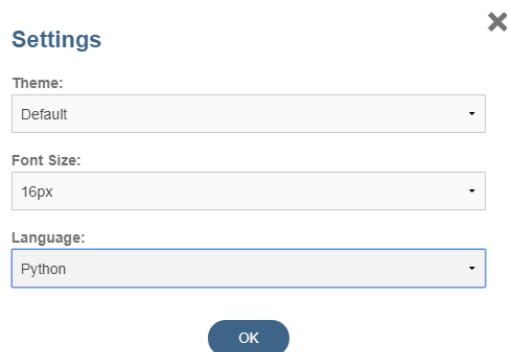


Рисунок 1 -

Для введення значень з клавіатури використовуйте функцію `input()` з параметрами. Наприклад: `n=int(input("Enter chislo"))`.

ЧАСТИНА 1. РОБОТА ЗІ СЛОВНИКАМИ

Ознайомтесь з методами списків в документації <https://docs.python.org/3/tutorial/datastructures.html#dictionaries> або `help(dict)`

1. Створення словників

Списки можна створювати за допомогою фігурних дужок (`{}`).

```
empty_dict = {}          створення словника за допомогою {}
print(empty_dict)        {}
print(type(empty_dict))  <class 'dict'>
```

Або просто викликати функцію `dict()`. Вона має кілька форматів, основним з яких є наступний:

```
dict(<Ключ1>=<Значення1>[, ... , <КлючN>=<ЗначенняN>])
empty_dict2 = dict()
print(empty_dict2)      {}
```

Аргументом функції `dict()` може бути список кортежів з парою елементів (ключ, значення).

```
dict1 = dict([("a", 5), ("b", 8)])
print("dict1=", dict1)      dict1= {'a': 5, 'b': 8}
```

Аргументом функції `dict()` може бути список списків з парою елементів (ключ, значення).

```
dict2= dict([["a", 5], ["b", 8]])
print("dict2=", dict2)      dict2= {'a': 5, 'b': 8}
```

Створити словник можна за допомогою операції привласнення, перерахувавши праворуч від знака рівності всі його елементи всередині фігурних дужок. Ключ і значення розділяється двокрапкою, а пари (ключ: значення) розділяються комами.

```
dict3={"A1":"abc","A2": [4, 5, 6]}
print("dict3=", dict3)      dict3= {'A1': 'abc', 'A2': [4, 5, 6]}
```

Створити словник можна за допомогою метода `fromkeys(seq[, value])`. Створюється новий словник з ключами з `seq` і значеннями з `value`. За замовчуванням `value` присвоюється значення `None`.

```
seq = ('name', 'age', 'sex')
dict4 = dict.fromkeys(seq)
print("dict4=", dict4)      dict4= {'name': None, 'age': None, 'sex': None}
dict5 = dict.fromkeys(seq, 10)
print("dict5=", dict5)      dict5= {'name': 10, 'age': 10, 'sex': 10}
```

Можна ще за допомогою генераторів словників, які дуже схожі на генератори списків.

```
dict6= {a: a** 2 for a in range(7)}
print("dict6=", dict6)      dict6= {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

2. Додавання і видалення елементів словника

Щоб додати елемент в словник потрібно вказати новий ключ і значення. При цьому ключ вказується всередині квадратних дужок.

```
d1 = { "Ukraine" : "Kiev" , "USA"
      : "Washington" }
d1[ "China" ] = "Beijing"
print (d1)                                {'China': 'Beijing', 'USA': 'Washington',
                                          'Ukraine': 'Kiev'}
```

Оператор *del* видаляє елементи словника.

```
del d1["USA"]
print (d1)                                {'China': 'Beijing', 'Ukraine': 'Kiev'}
del d1
print (d1)                                NameError: name 'd1' is not defined
```

Метод *clear()* очищає словник (видаляє всі елементи).

```
d={'x': 200, 'y': 33, 'z': 1000}
print(d)                                {'z': 1000, 'x': 200, 'y': 33}
dict.clear()
print(d)                                {}
```

Щоб додати якийсь ключ-значення в словник, можна використовувати вбудований метод *update([other])*, який приймає словник і оновлює існуючий.

```
d.update({'c':200})
print(d)                                {'x': 200, 'y': 33, 'z': 1000, 'c': 200}
```

3. Створення копії словників.

Для копіювання вмісту словника, а не його імені можна скористатися методом *copy()* або функцією *dict()*.

```
d = { "a": 5, "b": 8 }
d1=d.copy()
print (d1 is d)                            False
d1["b"]=25
print("d=", d)                            d= {'a': 5, 'b': 8}
print("d1=", d1)                          d1= {'a': 5, 'b': 25}
d2=dict(d)
d2["b"] = 55
print("d=", d)                            d= {'a': 5, 'b': 8}
print("d2=", d2)                          d2= {'a': 5, 'b': 55}
```

4. Робота зі словниками

Доступ до елемента словника, здійснюється як же як доступ до елемента списку, тільки в якості індексу вказується ключ. Якщо ми спробуємо отримати доступ по ключу, якого не існує, то Python видасть нам помилку *KeyError*, тому що такого ключа немає.

```
d={'x': 200, 'y': 33, 'z': 1000}
print(d['y'])                             33
```

```
print(d['c'])                                KeyError: 'c'
```

Щоб взяти значення по ключу і в разі невдачі повернути якесь дефолтне значення, є вбудований метод *get(key)* у словника, який це робить.

```
print(d.get('c', 'not found'))               not found
```

Щоб перевірити, чи міститься ключ в словнику, використовується оператор *in*.

```
print('y' in d)                             True
```

```
print('c' in d)                             False
```

Перебрати всі елементи словника можна за допомогою циклу *for*, оскільки словники підтримують ітерацію.

```
for key in d:                                x y z
    print(key, end=' ')
```

Перебрати всі елементи словника не по ключам, як за замовчуванням, а по ключам і значенням відразу, можна використовувати метод словника *items()*, який повертає ключі і значення.

```
for key, value in d.items():                 x - 200
    print('{}-{}'.format(key, value))        z - 1000
                                           y - 33
print(d.items())                            dict_items([('x', 200), ('z', 1000), ('y', 33)])
```

Якщо потрібно перебрати за значеннями, можна використовувати метод *values()*, який повертає саме значення.

```
for value in d.values():                     33
    print(value)                             1000
                                           200
print(d.values())                           dict_values([33, 1000, 200])
```

Також існує симетричний метод *keys()*, який повертає оператор ключів.

```
print(d.keys())                             dict_keys(['x', 'y', 'z'])
```

У об'єктів *dict_keys()* є одна особливість: вони підтримують операції на множинах.

```
d1,d2={'a':10, 'z':1000,
'b':'Hello'}, {'y':50, 'b':'Hello',
'x':(21,77)}
print("OR:  ", d1.keys() | OR: {'x', 'b', 'a', 'y', 'z'}
d2.keys())
print("AND:  ", d1.keys() & AND: {'b'}
d2.keys())
print("XOR:  ", d1.keys() ^ XOR: {'x', 'a', 'y', 'z'}
d2.keys())
print("- : ",d1.keys() - d2.keys()) - : {'a', 'z'}
```

Функція *len()* повертає кількість елементів (ключів) словника.

```
print(len(d))                               3
```

5. Сортуння словників

Словники є неупорядкованими наборами. Щоб вивести елементи з сортуванням по ключам, потрібно отримати набір ключів з допомогою методу *keys()*, перетворити отриманий об'єкт в список, впорядкувати його, а потім виводити елементи, використовуючи відсортований список ключів.

```
d1={'y':50, 'b':'Hello', 'x':(21,77)}
print(d1)                                {'y': 50, 'b': 'Hello', 'x': (21, 77)}
k=list(d1.keys())
k.sort()
for key in k:                             d['b']=Hello
    print("d['{0}']={1}".format(key, d1[key])) d['x']=(21, 77)
                                              d['y']=50
```

При сортуванні ключів замість методу *sort()* можна використовувати функцію *sorted()*. Функції *sorted()* можна відразу передавати об'єкт словника, а не об'єкт *dict_keys()*, що повертається методом *keys()*.

```
d2={'a':10, 'z':1000, 'b':'Hello'}
print(d2)                                {'b': 'Hello', 'z': 1000, 'a': 10}
for key in sorted(d2):                    d['a']=10
    print("d['{0}']={1}".format(key,d2[key])) d['b']=Hello
                                              d['z']=1000
```

Однак, в Python існує тип *OrderedDict*, який міститься в модулі *collections*, який гарантує, що ключі утримуються саме в тому порядку, в якому були додані в словник. Наприклад, можемо вивести пару число і рядок саме в тому порядку, в якому туди додали.

```
from collections import OrderedDict
ordered = OrderedDict()
for number in range(10):
    ordered[number] = str(number)
print(ordered)                            OrderedDict([(0, '0'), (1, '1'), (2, '2'),
(3, '3'), (4, '4'), (5, '5'), (6, '6'), (7, '7'),
(8, '8'), (9, '9')])

for key in ordered:                        0 1 2 3 4 5 6 7 8 9
    print(key, end=' ')
```

6. Інші методи словників

У словників доступні ще наступний набір методів.

pop(key[, default]) - видалити ключ-значення зі словника і повернути значення по цьому ключу, інакше буде повернуто значення *default*. Якщо *default* невідомий і запитуваний ключ відсутній в словнику, то буде викликано виняток *KeyError*.

```
vec={'x': 200, 'y': 33, 'z': 1000,
    'c':555, 'd': 888}
```

```

print(vec)                {'d': 888, 'z': 1000, 'c': 555, 'x': 200, 'y': 33}
print(vec.pop('y'))       33
print(vec)                {'d': 888, 'z': 1000, 'c': 555, 'x': 200}
print(vec.pop('w', 'error')) error
print(vec.pop('w'))       KeyError: 'w'

```

popitem() - видаляє і повертає пару (ключ, значення) зі словника. Якщо словник порожній, то буде викликано виключення `KeyError`.

```

print(vec.popitem())      ('d', 888)
print(vec)                {'z': 1000, 'c': 555, 'x': 200}
empty = {}
print(empty.popitem())    KeyError: 'popitem(): dictionary is empty'

```

setdefault() - щоб перевірити, чи існує ключ в словнику і в разі невдачі додати цю нову пару ключ-значення.

```

print(vec.setdefault('c', 555)) 555
print(vec)                    {'z': 1000, 'c': 555, 'x': 200}
print(vec.setdefault('c', 666)) 555
print(vec)                    {'z': 1000, 'c': 555, 'x': 200}
print(vec.setdefault('d', 666)) 666
print(vec)                    {'z': 1000, 'd': 666, 'x': 200, 'c': 555}

```

ЧАСТИНА 2. РОБОТА З МНОЖИНАМИ

1. Створення множин.

Множини можна створювати за допомогою фігурних дужок (`{ }`), або за допомогою функції *set()*. Але не можна створити порожню множину за допомогою двох фігурних дужок. Насправді, вони створюють порожній словник, а не множина. Якщо функції *set()* передати в якості параметра список, рядок або кортеж, то вона поверне множину, складену з елементів списку, рядку, кортежу.

```

empty_set1={}
print(type(empty_set1))    <class 'dict'>
empty_set2=set()
print("type={},           type=<class 'set'>, value=set()
value={}").format(type(empty_set2),
empty_set))
number_set={1,2,3,4,5}
print("number_set=",number_set)    number_set= {1, 2, 3, 4, 5}
str_set=set('Phyton')
print("str_set=",str_set)          str_set= {'h', 'y', 'P', 'o', 't', 'n'}

```

Кожен елемент може входити в безліч тільки один раз, порядок завдання елементів не важливий. Python гарантує, що в множині містяться тільки унікальні елементи. Це досягається за допомогою функції хешування.

```
A = {1, 2, 3}
```

```

B = {3, 2, 3, 1}
print("A={}, B={}".format(A,B))      A={1, 2, 3}, B={1, 2, 3}
print(A == B)                        True

```

2. Додавання і видалення елементів множини.

Множини є змінною структурою даних, а значить можемо додавати елементи в існуючу множину.

Додавання елемента до множини здійснюється за допомогою методу *add()*.

```

A = {1, 2, 3}
A.add(4)
print(A)                                {1, 2, 3, 4}
Щоб додати іншу множину є метод update().
A.update({5,6,7})
print(A)                                {1, 2, 3, 4, 5, 6, 7}
A.update({2,4,6,8}, {1,21,11})
print(A)                                {1, 2, 3, 4, 5, 6, 7, 8, 11, 21}

```

Для видалення елемента з множини є два методи: *discard()* і *remove()*. Їх поведінка відрізняється лише тим, коли видаляється елемент відсутній у множині. У цьому випадку метод *discard()* не робить нічого, а метод *remove()* генерує виняток *KeyError*.

```

A.discard(26)
A.remove(26)                            KeyError: 26

```

Метод *pop()* видаляє з множини один випадковий елемент і повертає його значення. Якщо ж множина порожня, то генерується виключення *KeyError*.

```

print(A.pop())                          1
print(A)                                {2, 3, 4, 5, 6, 7, 8, 11, 21}

```

Метод *clear()* видаляє всі елементи множини.

3. Робота з множинами

Дізнатися число елементів у множині можна за допомогою функції *len()*.

```

num_set={1,2,3,4,5}
print(len(num_set))                    5

```

Перебрати всі елементи множини (в невизначеному порядку!) можна за допомогою циклу *for*:

```

for i in num_set:                       1 2 3 4 5
    print(i, end=" ")

```

Перевірити, чи належить елемент безлічі можна за допомогою операції *in*, що повертає значення типу *bool*:

```

print(4 in num_set)                    True
Аналогічно є протилежна операція not in.
print(4 not in num_set)                False

```

4. Операції з множинами

Над множинами можна виконувати звичайні математичні операції.

A B A.union(B)	Повертає множину, що є об'єднанням множин A і B.
A = B A.update(B)	Додає в множину A все елементи з множини B.
A & B A.intersection(B)	Повертає множину, що є перетином множин A і B.
A &= B A.intersection_update(B)	Залишає в множині A тільки ті елементи, які є в множині B.
A – B A.difference(B)	Повертає різницю множин A і B (елементи, що входять до A, але не входять до B).
A -= B A.difference_update(B)	Видаляє з множини A все елементи, що входять до B.
A ^ B A.symmetric_difference(B)	Повертає симетричну різницю множин A і B (елементи, що входять до A або в B, але не в обидва з них одночасно).
A ^= B A.symmetric_difference_update(B)	Записує в A симетрическую різницю множин A і B.
A <= B A.issubset(B)	Повертає true, якщо A є підмножиною B.
A >= B A.issuperset(B)	Повертає true, якщо B є підмножиною A.
A < B	Еквівалентно A <= B and A! = B
A > B	Еквівалентно A >= B and A! = B

6. Незмінні множини Frozenset

Тип Frozenset гарантує унікальність елементів і поводить ся так само, як і множина, але не можна додавати туди елементи або видаляти їх.. Приблизно схожа ситуація з списками і кортежами.

Завдання для самостійного виконання

1. Написати програму, яка демонструє роботу зі словниками. В словнику зберігаються дані по таблиці 1. Передбачити створення словника з клавіатури або явно (за замовчуванням) з п'яти ключів.

За запитом користувача за ключем повинні виводитись відповідні значення. Якщо ключа не існує, то за запитом добавляти його в словник або вивести повідомлення «No key» та вивести всі значення з сортуванням по ключам.

Таблиця 1 – Данні словників

№	Ключ	Значення		
1.	Фамілія автора	назва книжки	рік видання	кількість сторінок
2.	Назва ст.групи	факультет	рік вступу	кількість студентів
3.	Назва товару	виробник	кількість	шифр товару
4.	Назва ріки	протяжність	число притоків	
5.	Назва району	кількість виборців	кількість діляниць	площа району
6.	Номер школи	кількість школярів	Назва вулиці	адреса вулиці
7.	Фірма принтера	кількість	рік придбання	модель
8.	Назва планети	діаметр	маса	віддаленість від Сонця
9.	Прізвище робітника	ім'я	стаж роботи	відділ, в якому працює
10.	Назва предмету	кафедра	викладач	кількість кредитів

2. Написати програму, яка демонструє роботу з множинами.

Програма повинна створювати дві множини з чисел різної довжини з клавіатури або явно (за замовчуванням).

№	
1.	Підраховує, скільки чисел міститься одночасно як в першій множині, так і в другій
2.	Виводить всі числа, які входять як в першу, так і в другу множину в порядку зростання.
3.	Виводить всі числа, які не входять як в першу, так і в другу множину в порядку спадання.
4.	Визначає, чи є перша множина підмножиною другої.
5.	Повертає різницю множин (елементи, що входять до другої, але не входять до першої).
6.	Повертає множину, що є перетином цих множин.
7.	Залишає в першій множині тільки ті елементи, які є в другій.
8.	Видаляє з першої множини всі елементи, що входять до другої.
9.	Повертає симетричну різницю множин (елементи, що входять до першої або до другої, але не в обидва з них одночасно).
10.	Записує в А симметрическую різницю множин А і В.