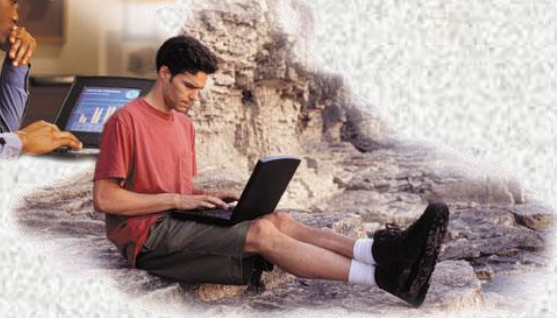




# Основи на Програмирането



## Лекция 6



## Речници, Множества, Файлове

## Какво ще научите

- ✓ Вграден тип данни речник
- ✓ Вграден тип данни множество
- ✓ Вграден тип данни файл
- ✓ Основни функции и методи за работа с файлове

# Контейнери

- Това са **обекти** които имат като елементи указатели (идентичност) на други обекти

**Пример:** [1, a, “help”, True]

- Стойността на един контейнер включва стойностите на отделните му елементи
- Ако контейнерът е **immutable**, това означава че указателите на елементите не се променят. Но ако даден елемент на контейнер е от тип **mutable**, стойността му може да се промени.

# Вградени типове

Типове	Примери за стойности
Числа	1234, 3.1415, 3+4j
Низове	'spam', "guido's"
Логически	True, False
Списъци	[1, [2, 'three'], 4]
Речници	{'food': 'spam', 'taste': 'yum'}
Редици	(1,'spam', 4, 'U')
Файлове	myfile = open('eggs', 'r')



# Речници

- ✓ Речниците са не подредено множество от обекти
- ✓ Обектите могат да бъдат от произволен тип
- ✓ Броят на обектите е неопределен и се изменя
- ✓ Всеки обект може да бъде произволен контейнер
- ✓ Всеки обект в речника се идентифицира с ключ (име)
- ✓ Достъпът до обектите става чрез ключа (името)
- ✓ Речниците са mutable (като списъците)
- ✓ Ключът (immutable) може да има само една стойност
- ✓ Различните ключове трябва да са с различна стойност

# Речници – прости команди

E = {} # празен речник

D = {'име': 'Bob', 'age': 40} # речник с 2 елемента

F = {'Num': 11, 'boss': {'name': 'Bob', 'age': 40}}

# Вложен речник в речник

D['име'] -> 'Bob'

F['boss']['age'] -> 40

>>> 'age' in D

True

>>> 40 in D

False

## Речници – прости команди 2

`D = {'име': 'Bob', 'age': 40}`    # речник с 2 елемента

`D['име'] = 'Bobby'`            # смяна на стойност

`D['pos'] = 'boss'`            # добавя нов елемент

`del(D ['age'])`            # изтрива елемент

`>>> D = dict(name='Bob', age=40)`

`>>> D`

`{'name': 'Bob', 'age': 40}`

`>>> E = dict.fromkeys(['име', 'възраст'])`

`>>> E`

`{'име': None, 'възраст': None}`

# Речници – прости команди 3

```
>>> h = {1 : 11}
```

```
>>> h[2.5] = 25
```

```
>>> h
```

```
{1: 11, 2.5: 25}
```

```
>>> h[True] = 10
```

```
>>> h
```

```
{1: 10, 2.5: 25}      # True е представено с 1 !!!
```

```
>>> h[True]
```

```
10
```

```
>>> h[False] = 0
```

```
>>> h
```

```
{1: 10, 2.5: 25, False: 0}
```



# Речници – вградени функции

```
>>> zip(['key1', 'key2'], [123, "Name"])
```

```
<zip object at 0x0017CCB0>
```

```
>>> dict(_)
```

```
{'key1': 123, 'key2': 'Name'}
```

```
>>> len(_)
```

```
2
```

```
>>> E = dict(zip(['key1', 'key2'], [123, "Name"]))
```

```
>>> del(E['key1'])
```

```
>>> print(E)
```

```
{'key2': 'Name'}
```

# Вградена функция zip()

Функцията zip има аргумент контейнер (итератор) и връща като резултат итератор, съдържащ редици - всяка поредна редица съдържа поредния елемент на итератора.

Ако аргументът не е контейнер или итератор, връща празен итератор.

Ако подадем два или повече итератора (контейнера), резултатът е един итератор, елементите на който са редици - толкова редици-елемента, колкото са елементите на контейнерите; в N-тата редица се съдържат N-тите елементи от входните контейнери (итератори).

Ако входните контейнери (итератори) не са с един и същи брой елементи, резултатния итератор ще съдържа толкова на брой редици, колкото е минималният брой елементи измежду входните контейнери (итератори).

# Вградена функция zip() – пример 1

```
test = zip()    # създаваме празен итератор
```

```
print('The type of an empty zip : ', type(test))
```

```
The type of an empty zip : <class 'zip'>
```

```
l1 = [11, 22, 33, 44, 55]
```

```
f = zip(l1)
```

```
for i in f:
```

```
    print(i)
```

```
(11,)
```

```
(22,)
```

```
(33,)
```

```
(44,)
```

```
(55,)
```

# Вградена функция zip() – пример 2

```
list1 = ['Alpha', 'Beta', 'Gamma', 'Sigma']
```

```
list2 = ['one', 'two', 'three', 'six']
```

```
test = zip(list1, list2)    # съединяваме със zip два контейнера
```

```
for values in test:
```

```
    print(values) # извеждаме всяка редица
```

```
('Alpha', 'one')
```

```
('Beta', 'two')
```

```
('Gamma', 'three')
```

```
('Sigma', 'six')
```



## Вградена функция zip() – пример 3

```
list1 = ['Alpha', 'Beta', 'Gamma', 'Sigma'] # списък с 4 елемента
list2 = ['one', 'two', 'three', 'six', 'five'] # списък с 5 елемента
list3 = [1, 2, 3] # списък с 3 елемента
test = zip(list1, list2, list3) # съединяваме списъците със zip
cnt = 0
for values in test:
    print(values) # извеждаме поредната редица от итератора
    cnt+=1 # увеличаваме брояча с 1
('Alpha', 'one', 1)
('Beta', 'two', 2)
('Gamma', 'three', 3)
print('Zip обектът съдържа ', cnt, ' елемента.');
```

Zip обектът съдържа 3 елемента.



# Разделяне на съединени контейнери (итератори)

```
list1 = ['Alpha', 'Beta', 'Gamma', 'Sigma']
```

```
list2 = ['one', 'two', 'three', 'six']
```

```
test = zip(list1, list2) # zip the values
```

```
testList = list(test)
```

```
a, b = zip( *testList )
```

```
print('Първият списък: ', list(a));
```

```
print('Вторият списък: ', list(b));
```

Първият списък: ['Alpha', 'Beta', 'Gamma']

Вторият списък: ['one', 'two', 'three']

# Речници – свойства

- ✓ Запазват реда на обектите както са въведени
- ✓ Нови елементи се добавят в края
- ✓ Елементи могат да се трият
- ✓ Елементите в речника могат да се обърнат в обратен ред

# Речници – свойства

```
>>> D = dict(name='Bob', age=40)
```

```
>>> D
```

```
{'name': 'Bob', 'age': 40}
```

```
>>> del D['name']
```

```
>>> D
```

```
{'age': 40}
```

```
>>> D['name'] = 'John'
```

```
>>> D
```

```
{'age': 40, 'name': 'John'}
```

```
>>>
```

# Речници – свойства

```
>>> D
```

```
{'age': 40, 'name': 'John'}
```

```
>>> list(reversed(D))
```

```
['name', 'age']
```

```
>>> list(reversed(D.values()))
```

```
['John', 40]
```

```
>>> D1 = {3:3, 2:2, 5:5, 1:1, 4:4}
```

```
>>> sorted(D1)
```

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

```
>>>
```

# Вградени методи и функции

D.keys()	# връща стойностите на ключовете
D.values()	# връща стойностите на обектите
D.items()	# връща двойки от стойности
D.copy()	# връща копие на речника
D.clear()	# изтрива елементите на речника
D.update(D2)	# добавя в D елементите от D2
D.popitem()	# връща и изтрива случайна 2-ка



# Вградени методи и функции

D.get(key) # връща обекта за ключа key

# ако няма такъв, връща none

D.get(key, <default>) # връща обекта за ключа key

# ако няма такъв, връща <default>

D.pop(key) # връща обекта за ключа key и изтрива

# двойката от речника (ако няма – none)

D.pop(key, <default>) # като горе, ако няма: <default>

D.setdefault(key) # аналогична на get

D.setdefault(key, <default>) # ако няма ключ key,

# вмъква двойка key:<default>

## Речници – примери

```
>>> D = {x: x*2 for x in range(10)}
```

```
>>> D
```

```
{0:0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}
```

```
>>> D = {c: c * 3 for c in 'STRING'}
```

```
>>> D
```

```
{'S': 'SSS', 'T': 'TTT', 'R': 'RRR', 'I': 'III', 'N': 'NNN', 'G':  
'GGG'}
```

## Речници – още примери

```
>>> D = {'a': 1, 'b': 2, 'c': 3}
```

```
>>> K = D.keys()
```

```
>>> V = D.values()
```

```
>>> I = D.items()
```

```
>>> K
```

```
dict_keys(['a', 'b', 'c'])
```

```
>>> list(K)
```

```
['a', 'b', 'c']
```

```
>>> list(V)
```

```
[1, 2, 3]
```

## Речници – още примери 2

```
>>> D = {'a': 1, 'b': 2, 'c': 3}
```

```
>>> del D['b']
```

```
>>> list(K)
```

```
['a', 'c']
```

```
>>> list(V)
```

```
[1, 3]
```

```
>>> list(I)
```

```
[('a', 1), ('c', 3)]
```

## Речници – още примери 3

Полезна практика за работа с речници е с използване на метода `setdefault()`. Ако трябва да се използва проверка дали даден ключ е наличен в речник и ако не е, да му се даде стойност по премълчаване, се използва метода `setdefault()`.

```
>>> workDetails = { }
```

```
>>> workDetails.setdefault('hours', 0) # Does nothing if  
'hours' exists.
```

```
0
```

```
>>> workDetails['hours'] += 10
```

```
>>> workDetails['hours']
```

```
10
```



# Обединение на речници

```
d1 = {"x": 1, "y": 4, "z": 10}
```

```
d2 = {"a": 7, "b": 9, "x": 5}
```

```
# Очакван резултат: {'x': 5, 'y': 4, 'z': 10, 'a': 7, 'b': 9}
```

```
# Важно: "x" е с променена стойност от 2-я речник
```

```
# Вариант 1
```

```
d = d1.copy() # Правим копие на 1-я речник
```

```
d.update(d2) # Променяме го "в паметта" с 2-я
```

```
# Вариант 2
```

```
d = {**d1, **d2}
```

# Нов оператор за речници в 3.9

Оператор за обединение на речници: |

```
>>> a = {1: 'one', 2: 'two', 3: 'three'}
```

```
>>> b = {4: 'four', 5: 'five'}
```

```
>>> c = a | b
```

```
>>> print(c)
```

```
{1: 'one', 2: 'two', 3: 'three', 4: 'four', 5: 'five'}
```

```
d1 = {"x": 1, "y": 4, "z": 10}
```

```
d2 = {"a": 7, "b": 9, "x": 5}
```

```
d1|d2
```

```
{'x': 5, 'y': 4, 'z': 10, 'a': 7, 'b': 9}
```

# Нов оператор за речници в 3.9

```
>>> a = {'a': 'one', 'b': 'two'}
```

```
>>> b = ((i, i**2) for i in range(3))
```

```
>>> a |= b
```

```
>>> print(a)
```

```
{'a': 'one', 'b': 'two', 0: 0, 1: 1, 2: 4}
```

```
>>> c = a | b
```

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

```
c = a | b
```

TypeError: unsupported operand type(s) for |: 'dict' and 'generator'

# Структура от данни: множество

- Отговаря на математическото понятие множество
- Константи от този тип се задават в  $\{ \}$  разделени със запетайка
- Разграничават се от речник по това, че елементите на речник са двойка ключ:стойност
- Елементите в множеството не са подредени и не се допуска повторения



# Особености на множествата в Python

- Всеки обект от тип множество е `mutable`
- Елементите на всяко множество трябва да бъдат обекти от тип `immutable`
- Това означава, че не може да имаме множество като елемент на друго множество – проблем!
- За това се въвежда специален тип данни: `immutable` множество – `frozenset`, обектите от който вече могат да са елементи на множество



## Примери

```
>>> S = {10, 3, 7, 2, 11}
```

```
>>> S
```

```
{2, 11, 3, 10, 7}
```

```
>>> T = {5, 4, 5, 2, 4, 9}
```

```
>>> T
```

```
{9, 2, 4, 5}
```

```
>>> L = [10, 13, 10, 5, 6, 13, 2, 10, 5]
```

```
>>> S = set(L)    # S = {2, 5, 6, 10, 13}
```

```
>>> T = frozenset(L)  # T = frozenset({2, 5, 6, 10, 13})
```

```
>>> S.add(T)
```

```
>>> S
```

```
{2, 5, 6, 10, 13, frozenset({2, 5, 6, 10, 13})}
```

```
>>>
```

# Операции над множества

Операция	Мат. символ	Оператор в Python	Резултат	Смисъл
Обединение	$A \cup B$	$A \mid B$	set	Елементите в А или В или двете
Сечение	$A \cap B$	$A \& B$	set	Общите за А и В елементи
Разлика	$A - B$	$A - B$	set	Елементи от А които не са в В
Симетрична разлика	$A \oplus B$	$A \wedge B$	set	Елементи от А или В, но не и в двете

## Лекция 6

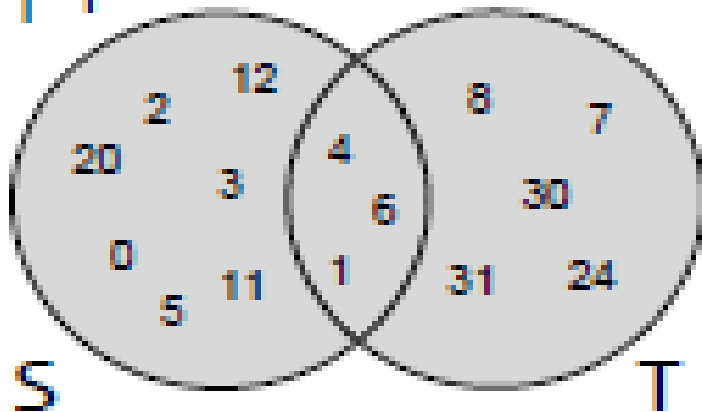
Оператор	Мат. символ	Оператор в Python	Резултат	Смисъл
Членство	$x \in A$	<code>x in A</code>	bool	x е елемент в A
Членство	$x \notin A$	<code>x not in A</code>	bool	x не е елемент в A
Равенство	$A = B$	<code>A == B</code>	bool	Множествата A и B съдържат еднакви елементи
Под-множество	$A \subseteq B$	<code>A &lt;= B</code>	bool	Всеки елемент от A е елемент и от B
Точно под-множество	$A \subset B$	<code>A &lt; B</code>	bool	A е подмножество на B, но B има поне един елемент който не е в A

## Лекция 6

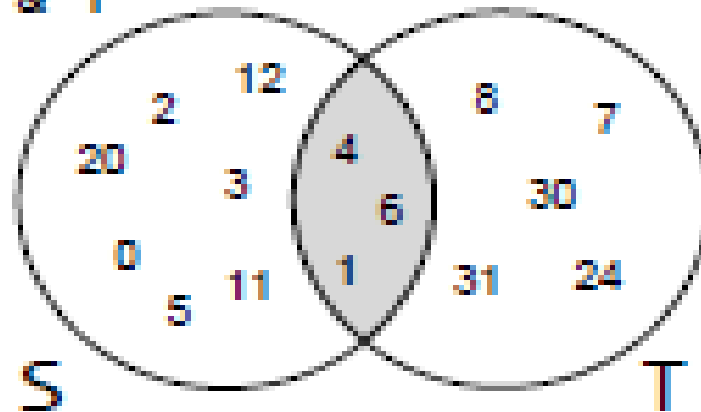
$$S = \{0, 1, 2, 3, 4, 5, 6, 11, 12, 20\}$$

$$T = \{1, 4, 6, 7, 8, 24, 31\}$$

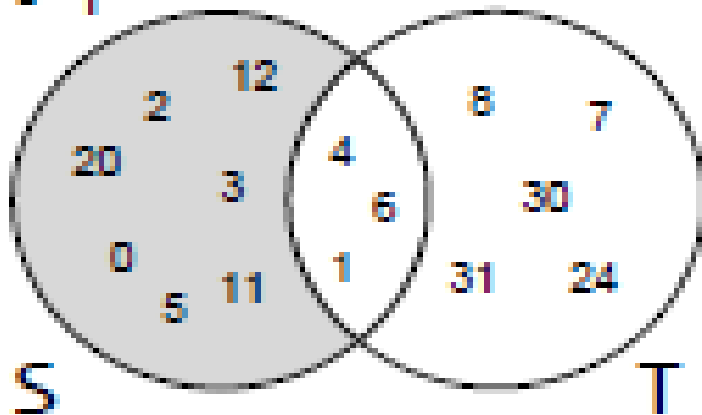
$S \mid T$



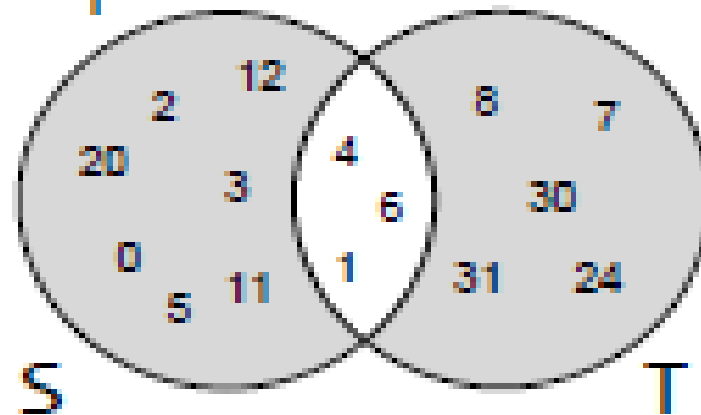
$S \& T$



$S - T$



$S \wedge T$



## Още примери за множества

```
>>> S = {x**2 for x in range(10)}
```

```
>>> S
```

```
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
```

```
>>> len(S)
```

```
10
```

```
>>> 64 in S
```

```
True
```

```
>>> L = [elem for elem in S]
```

```
>>> L
```

```
[0, 1, 64, 4, 36, 9, 16, 49, 81, 25]
```



# Още примери за множества

```
>>> b = {1,2,3,4,3,7,4,1}
```

```
>>> b
```

```
{1, 2, 3, 4, 7}
```

```
>>> c = {1, 3, [2,4]}
```

Traceback (most recent call last):

File "<pyshell#39>", line 1, in <module>

c = {1, 3, [2,4]}

TypeError: unhashable type: 'list'

```
>>> ss = {}
```

```
>>> type(ss)
```

```
<class 'dict'>
```

```
>>> ss = set()
```

```
>>> type(ss)
```

```
<class 'set'>
```

# Mutable операции за множества

- Всяка от основните операции:  $|=$  ;  $\&=$  и т.н.
- Функции като вградени методи:
  - `add(elem)`, `remove(elem)`, `discard(elem)`, `pop()`, `clear()`, `update(<iterables>)`, `copy()`

```
>>>S={2, 4, 7}
```

```
>>>S.add(9)  ->  S = {9, 2, 4, 7}
```

```
>>>S.pop()
```

```
9          # S = {1, 2, 4, 7}
```

# Още примери за множества

```
my_set = {1,3}
# add an element
my_set.add(2)
print(my_set)
{1, 2, 3}
# add multiple elements
my_set.update([2, 3, 4])
print(my_set)
{1, 2, 3, 4}
# add list and set
my_set.update([4, 5], {1, 6, 8})
print(my_set)
{1, 2, 3, 4, 5, 6, 8}
```

# Още примери за множества

```
my_set = {1, 3, 4, 5, 6}
# discard an element
my_set.discard(4)
print(my_set)
{1, 3, 5, 6}
# remove an element
my_set.remove(6)
print(my_set)
{1, 3, 5}
# not present in my_set
my_set.discard(2)
print(my_set)
# Output: {1, 3, 5}
# remove an element not present
my_set.remove(2)
KeyError
```



# Съвместно използване на двата типа множества – изменяеми и неизменяеми

- Допуска се в изрази и команди
- При смесено използване резултатът е обект от типа на първия срещнат в израза
- Сравненията не зависят от типа
- Тъй като и за двата типа няма пълна наредба: методът `sort()` не е дефиниран за множества
- Можем да получим всички елементи от едно множество `S` сортирани в списък с функцията: `sorted(S)`



# Реализация на кванторите

- Вградена функция `all(iter)`: истина ако за всяко `x` от `iter` `bool(x)` е истина. Ако обектът е празен, връща истина.
- $\text{all}(x > 0 \text{ for } x \text{ in } S) \quad \# \quad (\forall x \in S)(x > 0)$
- Вградена функция `any(iter)`: истина е ако за поне едно `x` от `iter` `bool(x)` е истина. Ако `iter` е празен, връща лъжа.
- $\text{Any}(x > 0 \text{ for } x \text{ in } S) \quad \# \quad (\exists x \in S)(x > 0)$

# Стандартни вградени функции

- `enumerate()` - Връща обект от тип `enumerate` (съдържа двойки индекс, елемент за всеки елемент от множеството)
- `len()` - Брой елементи в множеството
- `max()` - Връща максималният елемент от множеството
- `min()` - Връща минималният елемент от множеството
- `sorted()` - Връща списък с подредените по големина обекти, съдържащи се в множеството
- `sum()` - Връща сумата от елементите в множеството

# Файлове

- ✓ Използват се за постоянно съхраняване на данни
- ✓ В най-простия си вид въвеждането и извеждането на информация във файл наподобява това със стандартния вход (клавиатура) и изход (екран)
- ✓ Входно-изходните операции с файлове са много повече на брой, с много по-големи възможности и позволяват всякаква обработка на информацията съхранявана във файловете
- ✓ Стандартната команда за свързване на обект от тип данни файл с конкретен физически файл:

```
afile = open(<filename>,< mode>)
```

# Отваряне на файл

`<file_obj_name> = open(<filename>,< mode>)`

- ✓ `open()` е функция, която отваря физическия файл за вход/изход, създава входно-изходен буфер в паметта и връща като резултат адреса на този буфер
- ✓ `<file_obj_name>` е променлива, която задава името на обект от тип файл, който се свързва с току-що създадения входно-изходен буфер, т.е. тази променлива съдържа като стойност адреса на буфера на файла



# Задаване на име за файл

`<file_obj_name> = open(<filename>,< mode>)`

- ✓ И двата параметъра на функцията `open()` се задават като обекти от тип низ
- ✓ Като конкретно име на файл (параметър `<filename>`) се задава валидно за конкретната ОС име на файл заедно с път за достъп (абсолютен или относителен)
- "names.txt" – указва име на файл в текущата папка
- "C:\Python\file.txt" – указва име на файл с абсолютен път за достъп към него



# Режими за работа с файлове

Примери за стойности на параметъра <mode>:

'r' - въвеждане от файл (четене, read)

'w' - създаване и извеждане във файл (писане, write) – ако съществува съдържанието му се изтрива

'a' - добавяне в края на файл

't' - работа с текстов файл

'b' - работа с двоичен (не текстов) файл

'+' - използване на файла за писане и четене (обновяване)

'x' - създаване и извеждане във файл (писане, write) – ако съществува връща грешка

# Начини за работа с файлове

- ✓ Когато файла е отворен като двоичен, четене и записване става чрез обект данни `bytes`
- ✓ Когато файла е отворен като текстов, четене и записване става чрез обект данни `str`, като първо се прави декодиране на съдържанието (четене) или кодиране (записване)
- 'b' - работа с двоичен (не текстов) файл
- '+' - използване на файла за писане и четене (обновяване)
- 'x' - създаване и извеждане във файл (писане, `write`) – ако съществува връща грешка

# Други параметри при отваряне на файлове

`buffering` – размер на буфера за четене и писане

`encoding` – метод на кодиране (само за текстов файл)

`errors` – метод за обработка на грешките

`newline` – как да се интерпретира край на ред (само за текстов файл) – има различия в различни ОС

`closefd` – указва как и кога да се затваря физически файла

`opener` – за използване на алтернативна програма за отваряне на файла

# Вградени методи за работа с файлове

```
aString = input.read()      # чете цял файл в низ
aString = input.read(N)     # чете следващите N символа
                             # (или байта) в низ
aString = input.readline()  # чете един ред в низ
aList = input.readlines()   # чете цял файл в списък низове
                             # по редове
output.write(aString)       # записва низ от символи (или
                             # байтове) във файл
output.writelines(aList)    # записва списък от низове
                             # (редове) в цял файл
```



# Вградени методи за работа с файлове

<code>aString = input.read()</code>	<code># чете цял файл в низ</code>
<code>output.close()</code>	<code># Затваряне на файл (след # всички записи)</code>
<code>output.flush()</code>	<code># Запис от буфера във файла без # затваряне</code>
<code>anyFile.seek(N)</code>	<code># Позиционира файл до позиция # N за следващо действие</code>
<code>for line in open('data'):</code>	<code># използва line за итеративно # четене по редове от файл</code>



# Примери

## Преброяване на броя редове в един файл:

```
file1 = open("text.txt", "r")  
line_count = 0  
for line in file1:  
    line_count += 1  
print('Брой на редове във файла: ', line_count)
```

## Прочитане на цял файл:

```
File2 = open("text2.txt")  
Batch = File2.read()  
print(len(Batch))
```

## Примери 2

```
>>> lst = []
>>> for line in open("text.txt", 'r').readlines():
    lst.append(line)
>>> lst
['Chen Lin\n', 'clin@brandeis.edu\n', 'Volen 110\n', 'Office Hour:
Thurs. 3-5\n', '\n', 'Yaqin Yang\n', 'yaqin@brandeis.edu\n',
'Volen 110\n', 'Offiche Hour: Tues. 3-5\n']
```

### Игнориране на заглавен ред

```
for (i,line) in enumerate(open('text.txt','r').readlines()):
    if i == 0: continue
    print line
```

# Брой срещания на име чрез речник

```
>>> for line in open('names.txt'):
...     name = line.strip()
...     name_count[name] = name_count.get(name,0)+ 1
...
>>> for (name, count) in name_count.items():
...     print name, count
...
```


Chen 3

Ben 3

Yaqin 3


# Копиране от един файл в друг

```
input_file = open("in.txt")  
output_file = open("out.txt", "w")  
for line in input_file:  
    output_file.write(line)
```



# Използване на with

При работа с файлове е полезно да използваме обработка със запазената дума with. Предимството е, че след обработката файлът е коректно затворен и няма нужда от използване на <file>.close.



```
>>> with open('workfile') as f:  
...     read_data = f.read()
```

```
>>> # Проверка, че файлът е затворен автоматично:  
>>> f.closed  
True
```



# Заключение

- Речник - ключове и стойности, дефиниране и достъп
- Речници - вградени функции и методи
- Множества - константи, представяне, достъп
- Множества - създаване, вградени функции
- Файл - задаване и използване
- Начини на използване на файлове
- Вградени методи за работа с файлове