

# «Введение в информационные технологии»

**ЛЕКЦИЯ 2. РАБОТА СО СТРОКАМИ И ФАЙЛАМИ, СПИСКАМИ, СРЕЗЫ СПИСКОВ, ФУНКЦИИ И ДЕКОРАТОРЫ, КОЛЛЕКЦИИ, МОДУЛИ МНОЖЕСТВА, СЛОВАРИ, ОПЕРАЦИИ СО СЛОВАРЯМИ.**

## **СОДЕРЖАНИЕ ЛЕКЦИИ:**

- ▶ Работа со строками и файлами.
- ▶ Множества, словари, операции со словарями.

# 1. Работа со строками и файлами

*Экранированные последовательности -  
служебные символы*

*позволяют вставить символы, которые сложно  
ввести с клавиатуры*

Экранированная последовательность	Назначение
\n	Перевод строки
\a	Звонок
\b	Забой
\f	Перевод страницы
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\N{id}	Идентификатор ID базы данных Юникода
\uhhhh	16-битовый символ Юникода в 16-ричном представлении
\Uhhhh...	32-битовый символ Юникода в 32-ричном представлении
\xhh	16-ричное значение символа
\ooo	8-ричное значение символа
\0	Символ Null (не является признаком конца строки)

```
main.py x
1 s = 'слово_1\nслово_2\nслово_3'
2 print(s)
3
```

```
слово_1
слово_2
слово_3
```

## Работа со строками

```
S = r'C:\newt.txt'
```

Если перед открывающей кавычкой стоит символ 'r' (в любом регистре), то механизм экранирования отключается

Конкатенация (сложение)

```
>>> S1 = 'stroka 1'
>>> S2 = 'stroka 2'
>>> print(S1 + S2)
'stroka 1stroka 2'
```

Дублирование строки

```
>>> print('sp' * 3)
spspsp
```

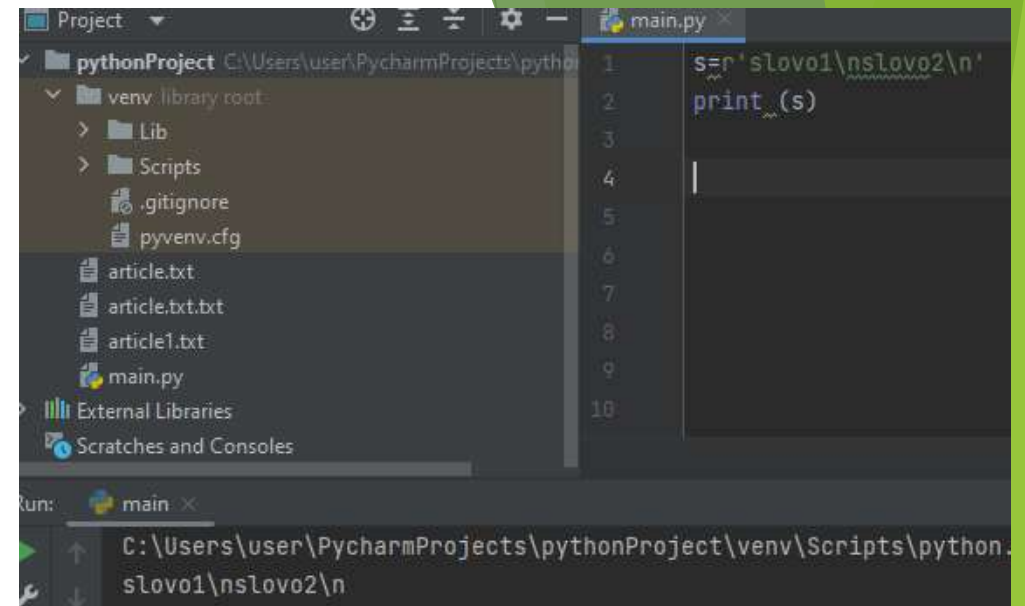
Длина строки (функция len)

```
>>> len('stroka')
6
```

возможен и доступ по отрицательному индексу, при этом отсчет идет от конца строки

Доступ по индексу

```
>>> S = 'spam'
>>> S[0]
's'
>>> S[2]
'a'
>>> S[-2]
'a'
```



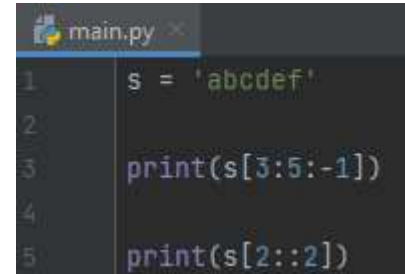
## Извлечение среза

Оператор извлечения среза: `[X:Y]`. `X` – начало среза, а `Y` – окончание; символ с номером `Y` в срез не входит. По умолчанию первый индекс равен 0, а второй – длине строки

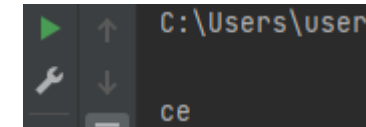
можно задать шаг, с которым нужно извлекать срез

При вызове методов необходимо помнить, что строки в Python относятся к категории неизменяемых последовательностей, то есть все функции и методы могут лишь создавать новую строку

```
>>> s = 'spameggs'
>>> s[3:5] 'me'
>>> s[2:-2] 'ameg'
>>> s[:6] 'spameg'
>>> s[1:] 'pameggs'
>>> s[:] 'spameggs'
```



```
main.py
1 s = 'abcdef'
2
3 print(s[3:5:-1])
4
5 print(s[2::2])
```



```
C:\Users\user\
ce
```

```
>>> s = 'spam'
>>> s[1] = 'b'
Traceback (most recent call last): File "", line 1, in s[1] = 'b'
TypeError: 'str' object does not support item assignment
>>> s = s[0] + 'b' + s[2:]
>>> s 'sbam'
```



## Функции и методы строк

Функция или метод	Назначение
<code>S = 'str'; S = "str"; S = """str"""; S = """str"""</code>	Литералы строк
<code>S = "s\np\ta\nbbb"</code>	Экранированные последовательности
<code>S = r"C:\temp\new"</code>	Неформатированные строки (подавляют экранирование)
<code>S = b"byte"</code>	Строка байтов
<code>S1 + S2</code>	Конкатенация (сложение строк)
<code>S1 * 3</code>	Повторение строки
<code>S[i]</code>	Обращение по индексу
<code>S[i:step]</code>	Извлечение среза
<code>len(S)</code>	Длина строки
<code>S.find(str, [start],[end])</code>	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
<code>S.rfind(str, [start],[end])</code>	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1
<code>S.index(str, [start],[end])</code>	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
<code>S.rindex(str, [start],[end])</code>	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
<code>S.replace(шаблон, замена)</code>	Замена шаблона
<code>S.split(символ)</code>	Разбиение строки по разделителю
<code>S.isdigit()</code>	Состоит ли строка из цифр
<code>S.isalpha()</code>	Состоит ли строка из букв
<code>S.isalnum()</code>	Состоит ли строка из цифр или букв
<code>S.islower()</code>	Состоит ли строка из символов в нижнем регистре
<code>S.isupper()</code>	Состоит ли строка из символов в верхнем регистре
<code>S.isspace()</code>	Состоит ли строка из неотображаемых символов (пробел, символ перевода строки ('\f'), "новая строка" ('\n'), "перевод каретки" ('\r'), "горизонтальная табуляция" ('\t') и "вертикальная табуляция" ('\v'))
<code>S.istitle()</code>	Начинаются ли слова в строке с заглавной буквы
<code>S.upper()</code>	Преобразование строки к верхнему регистру
<code>S.lower()</code>	Преобразование строки к нижнему регистру
<code>S.startswith(str)</code>	Начинается ли строка S с шаблона str

<code>S.endswith(str)</code>	Заканчивается ли строка S шаблоном str
<code>S.join(список)</code>	Сборка строки из списка с разделителем S
<code>ord(символ)</code>	Символ в его код ASCII
<code>chr(число)</code>	Код ASCII в символ
<code>S.capitalize()</code>	Переводит первый символ строки в верхний регистр, а все остальные в нижний
<code>S.center(width, [fill])</code>	Возвращает отцентрированную строку, по краям которой стоит символ fill (пробел по умолчанию)
<code>S.count(str, [start],[end])</code>	Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)
<code>S.expandtabs([tabsize])</code>	Возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от текущего столбца. Если TabSize не указан, размер табуляции полагается равным 8 пробелам
<code>S.lstrip([chars])</code>	Удаление пробельных символов в начале строки
<code>S.rstrip([chars])</code>	Удаление пробельных символов в конце строки
<code>S.strip([chars])</code>	Удаление пробельных символов в начале и в конце строки
<code>S.partition(шаблон)</code>	Возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий саму строку, а затем две пустых строки
<code>S.rpartition(sep)</code>	Возвращает кортеж, содержащий часть перед последним шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий две пустых строки, а затем саму строку
<code>S.swapcase()</code>	Переводит символы нижнего регистра в верхний, а верхнего – в нижний
<code>S.title()</code>	Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний
<code>S.zfill(width)</code>	Делает длину строки не меньшей width, по необходимости заполняя первые символы

```

main.py x
1 S1 = "Stroka_1"
2 S2 = "Stroka"
3 if S1.find(S2) != -1:
4     print('Подстрока найдена!')
5 else:
6     print('Совпадений нет!')
7 S1 = "Stroka_1"
8 S2 = "no"
9 if S1.find(S2) != -1:
10     print('Подстрока найдена!')
11 else:
12     print('Совпадений нет!')
13

```

Подстрока найдена!  
Совпадений нет!

```

1 S1 = "stroka_1"
2 S2 = "stroka"
3 print(S1.title())
4 print(S2.title())

```

Stroka\_1  
Stroka

*S1 = "stroka\_1"*  
*S2 = "stroka"*  
*print(S1.rstrip())*  
*print(S2.title())*

stroka\_1  
Stroka

```

main.py x
1 S1 = "sTroKa_1"
2 S2 = "StroKa"
3 print(S1.swapcase())
4 print(S2.swapcase())

```

C:\Users\us  
StROKA\_1  
sTROKa

# Форматирование строк

```
>>> 'Hello, {} !'.format('Vasya')
'Hello, Vasya!'
```

```
# аргументы по умолчанию
print("Hello {}, your balance is {}".format("Petr", 730.2346))

# позиционные аргументы
print("Hello {0}, your balance is {1}".format("Petr", 4430.26))

# аргументы ключевые слова
print("Hello {name}, your balance is {blc}".format(name="Petr", blc=11230.26))

# смешанные аргументы
print("Hello {0}, your balance is {blc}".format("Petr", blc=5400.46))
```

```
Hello Petr, your balance is 730.2346.
Hello Petr, your balance is 4430.26.
Hello Petr, your balance is 11230.26.
Hello Petr, your balance is 5400.46.
```

## Спецификация формата:

спецификация	::=	[[fill]align][sign][#][0][width][,][.precision][type]
заполнитель	::=	символ кроме '{' или '}'
выравнивание	::=	"<"   ">"   "="   "A"
знак	::=	"+"   "-"   " "
ширина	::=	integer
точность	::=	integer
тип	::=	"b"   "c"   "d"   "e"   "E"   "f"   "F"   "g"   "G"   "n"   "o"   "s"   "x"   "X"   "%"

Флаг	Значение
'<'	Символы-заполнители будут справа (выравнивание объекта по левому краю) (по умолчанию).
'>'	выравнивание объекта по правому краю.
'='	Заполнитель будет после знака, но перед цифрами. Работает только с числовыми типами.
'^'	Выравнивание по центру.

Флаг	Значение
'+'	Знак должен быть использован для всех чисел.
'-'	'-' для отрицательных, ничего для положительных.
'Пробел'	' ' для отрицательных, пробел для положительных.



## Работа с файлами

```
f = open('text.txt', 'r')
```

Режим	Описание
r	Только для чтения.
w	Только для записи. Создаст новый файл, если не найдет с указанным именем.
rb	Только для чтения (бинарный).
wb	Только для записи (бинарный). Создаст новый файл, если не найдет с указанным именем.
r+	Для чтения и записи.
rb+	Для чтения и записи (бинарный).
w+	Для чтения и записи. Создаст новый файл для записи, если не найдет с указанным именем.
wb+	Для чтения и записи (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
a	Откроет для добавления нового содержимого. Создаст новый файл для записи, если не найдет с указанным именем.
a+	Откроет для добавления нового содержимого. Создаст новый файл для чтения записи, если не найдет с указанным именем.
ab	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
ab+	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для чтения записи, если не найдет с указанным именем.

*Режимы могут быть объединены, то есть, к примеру, 'rb' - чтение в двоичном режиме.*

```
>>> f = open('text.txt')
>>> f.read(1) 'H'
>>> f.read()
'ello world!\nThe end.\n\n'
```

*метод read, читающий весь файл целиком, если был вызван без аргументов, и n символов, если был вызван с аргументом (целым числом n)*

```
>>> f = open('text.txt')
>>> for line in f: ... Line ...
'Hello world!\n'
'n' 'The end.\n'
'n'
```

*прочитать файл построчно*

## Запись в файл

```
This is line1.  
This is line2.  
This is line3.
```

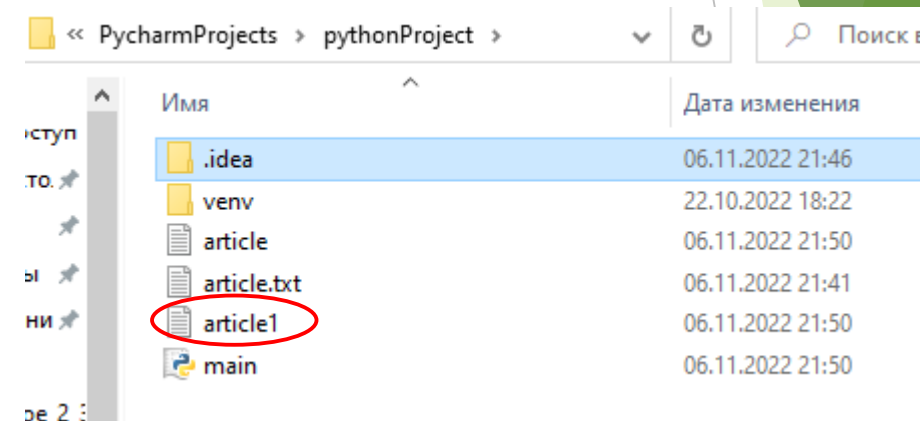
Посмотрим, как функция `readline()` работает в `test.txt`.

```
>>> x = open('test.txt','r')  
>>> x.readline() # прочитать первую строку  
This is line1.  
>>> x.readline(2) # прочитать вторую строку  
This is line2.  
>>> x.readlines() # прочитать все строки  
['This is line1.', 'This is line2.', 'This is line3.']
```

```
>>> f.close()
```

```
1 f = open('article1.txt', 'w')  
2 f.write('Hello \n World')
```

```
main.py  
1 f = open('article1.txt', 'w')
```



article1 – Блокнот

Файл Правка Формат Вид Справка

Hello  
World

## 2. Множества, словари, операции со словарями

**Словари** в Python - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами.

```
main.py x
1 dict_sample = {
2     "Company": "Toyota",
3     "model": "Premio",
4     "year": 2022
5 }
6 print(dict_sample)
```

```
{'Company': 'Toyota', 'model': 'Premio', 'year': 2022}
```

```
dict_sample = {
    "Company": "Toyota",
    "model": "Premio",
    "year": 2022
}
print(dict_sample["model"])
```

```
Premio
```

```
dict_sample = {
    "Company": "Toyota",
    "model": "Premio",
    "year": 2022
}
x=dict_sample["model"]
print(x)
```

*присвоение по новому ключу расширяет словарь,  
присвоение по существующему ключу перезаписывает его,  
а попытка извлечения несуществующего ключа  
порождает исключение*

```
>>> d = {1: 2, 2: 4, 3: 9}
>>> d[1]
2
>>> d[4] = 4 ** 2
>>> d
{1: 2, 2: 4, 3: 9, 4: 16}
>>> d['1']
Traceback (most recent call last):
  File "", line 1, in
    d['1']
KeyError: '1'
```

## Методы словарей

**dict.clear()** - очищает словарь.

**dict.copy()** - возвращает копию словаря.

classmethod **dict.fromkeys(seq[, value])** - создает словарь с ключами из seq и значением value (по умолчанию None).

**dict.get(key[, default])** - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).

**dict.items()** - возвращает пары (ключ, значение).

**dict.keys()** - возвращает ключи в словаре.

**dict.pop(key[, default])** - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).

**dict.popitem()** - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.

**dict.setdefault(key[, default])** - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ с значением default (по умолчанию None).

**dict.update([other])** - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).

**dict.values()** - возвращает значения в словаре.



# Множества

Множество в python - “контейнер”, содержащий не повторяющиеся элементы в случайном порядке

```
>>> a = set()
>>> a
set()
>>> a = set('hello')
>>> a
{'h', 'o', 'l', 'e'}
>>> a = {'a', 'b', 'c', 'd'}
>>> a
{'b', 'c', 'a', 'd'}
>>> a = {i ** 2 for i in range(10)} # генератор множеств
>>> a
{0, 1, 4, 81, 64, 9, 16, 49, 25, 36}
>>> a = {} # А так нельзя!
>>> type(a)
<class 'dict'>
```

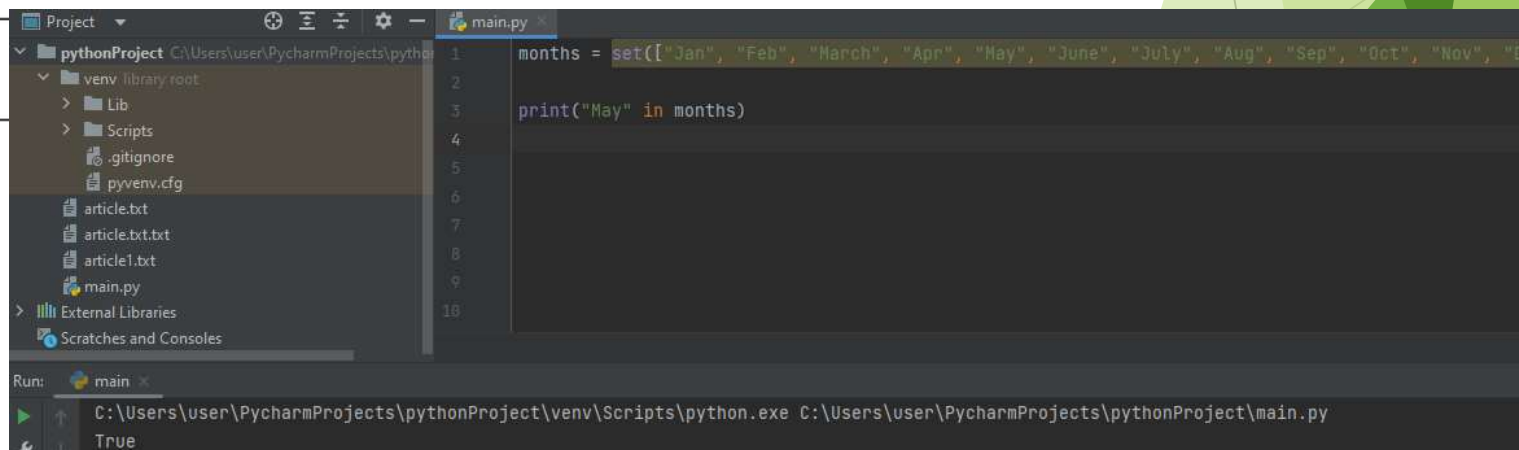
```
months = set(["Jan", "Feb", "March", "Apr", "May", "June", "July", "Aug", "Sep",
"Oct", "Nov", "Dec"])
```

```
for m in months:
    print(m)
```

```
March
Dec
Jan
Nov
July
Aug
Apr
May
Sep
Feb
June
Oct
```

Множества удобно использовать для удаления повторяющихся элементов

```
>>> words = ['hello', 'daddy', 'hello', 'mum']
>>> set(words)
{'hello', 'daddy', 'mum'}
```



- `len(s)` - число элементов в множестве (размер множества).
- `x in s` - принадлежит ли `x` множеству `s`.
- `set.isdisjoint(other)` - истина, если `set` и `other` не имеют общих элементов.
- `set == other` - все элементы `set` принадлежат `other`, все элементы `other` принадлежат `set`.
- `set.issubset(other)` или `set <= other` - все элементы `set` принадлежат `other`.
- `set.issuperset(other)` или `set >= other` - аналогично.
- `set.union(other, ...)` или `set | other | ...` - объединение нескольких множеств.
- `set.intersection(other, ...)` или `set & other & ...` - пересечение.
- `set.difference(other, ...)` или `set - other - ...` - множество из всех элементов `set`, не принадлежащие ни одному из `other`.
- `set.symmetric_difference(other)`; `set ^ other` - множество из элементов, встречающихся в одном множестве, но не встречающихся в обоих.
- `set.copy()` - копия множества.

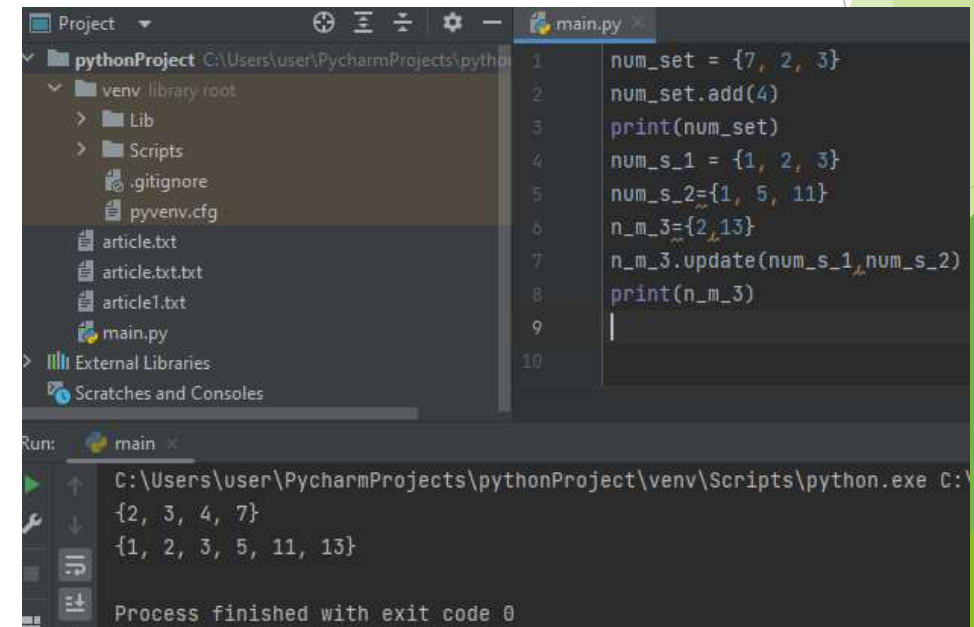
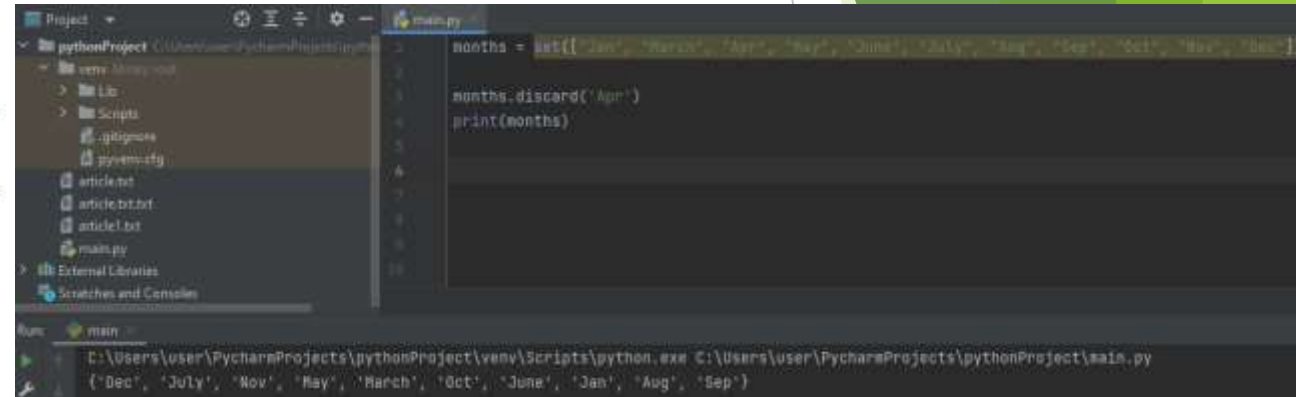
### операции, непосредственно изменяющие множество:

- `set.update(other, ...)`; `set |= other | ...` - объединение.
- `set.intersection_update(other, ...)`; `set &= other & ...` - пересечение.
- `set.difference_update(other, ...)`; `set -= other | ...` - вычитание.
- `set.symmetric_difference_update(other)`; `set ^= other` - множество из элементов, встречающихся в одном множестве, но не встречающихся в обоих.
- `set.add(elem)` - добавляет элемент в множество.
- `set.remove(elem)` - удаляет элемент из множества. `KeyError`, если такого элемента не существует.
- `set.discard(elem)` - удаляет элемент, если он находится в множестве.
- `set.pop()` - удаляет первый элемент из множества. Так как множества не упорядочены, нельзя точно сказать, какой элемент будет первым.
- `set.clear()` - очистка множества.

```
months = set(["Jan", "March", "Apr", "May", "June", "July",
              "Aug", "Sep", "Oct", "Nov", "Dec"])
```

```
months.add("Feb")
print(months)
```

```
{'Sep', 'May', 'Oct', 'Jan', 'June', 'Nov', 'Aug', 'Feb', 'Dec', 'July', 'March', 'Apr'}
```



```
1 num_set = {1, 2, 3, 4, 5, 6}
2 num_set.discard(7)
3 print(num_set)
```

Результат:

```
1 {1, 2, 3, 4, 5, 6}
```

Выдача выше показывает, что никакого воздействия на множество не было оказано. Теперь посмотрим, что выйдет из использования метода `remove()` по аналогичному сценарию:

```
1 num_set = {1, 2, 3, 4, 5, 6}
2 num_set.remove(7)
3 print(num_set)
```

Результат:

```
1 Traceback (most recent call last):
2   File "C:\Users\admin\sets.py", line 2, in <module>
3     num_set.remove(7)
4   KeyError: 7
```

## Функции и их аргументы

Функция в python - объект, принимающий аргументы и возвращающий значение. Обычно функция определяется с помощью инструкции `def`.

```
def add(x, y):  
    return x + y
```

```
>>> add(1, 10)  
11  
>>> add('abc', 'def')  
'abcdef'
```

```
>>> def newfunc(n):  
...     def myfunc(x):  
...         return x + n  
...     return myfunc  
...  
>>> new = newfunc(100) # new - это функция  
>>> new(200)  
300
```

Функция может и не заканчиваться инструкцией `return`, при этом функция вернет значение `None`:

```
>>> def func():  
...     pass  
...  
>>> print(func())  
None
```

**Декоратор** — это функция, которая позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода. Вот почему декораторы можно рассматривать как практику метапрограммирования, когда программы могут работать с другими программами как со своими данными.

**Функции высших порядков** — это функции, которые могут принимать в качестве аргументов и возвращать другие функции

```
1 >>> hello = hello_world  
2 >>> hello()  
3 Hello world!
```

можно хранить **функции в переменных**

Определять функции внутри других функций:

```
1 >>> def wrapper_function():  
2 ...     def hello_world():  
3 ...         print('Hello world!')  
4 ...     hello_world()  
5 ...  
6 >>> wrapper_function()  
7 Hello world!
```

Передавать функции в качестве аргументов и возвращать их из других функций:

```
>>> def higher_order(func):  
...     print('Получена функция {} в качестве аргумента'.format(func))  
...     func()  
...     return func  
...  
>>> higher_order(hello_world)  
Получена функция <function hello_world at 0x032C7FA8> в качестве аргумента  
Hello world!  
<function hello_world at 0x032C7FA8>
```



## Аргументы функции

Функция может принимать произвольное количество аргументов или не принимать их вовсе. Также распространены функции с произвольным числом аргументов, функции с позиционными и именованными аргументами, обязательными и необязательными

```
>>> def func(a, b, c=2): # c - необязательный аргумент
...     return a + b + c
...
>>> func(1, 2) # a = 1, b = 2, c = 2 (по умолчанию)
5
>>> func(1, 2, 3) # a = 1, b = 2, c = 3
6
>>> func(a=1, b=3) # a = 1, b = 3, c = 2
6
>>> func(a=3, c=6) # a = 3, c = 6, b не определен
Traceback (most recent call last):
  File "", line 1, in
    func(a=3, c=6)
TypeError: func() takes at least 2 arguments (2 given)
```

```
#defining the function def change_list(list1):
    list1.append(20)
    list1.append(30)
    print("list inside function = ",list1)
#defining the list list1 = [10,30,40,50]
#calling the function
change_list(list1)
print("list outside function = ",list1)
```

```
list inside function = [10, 30, 40, 50, 20, 30]
list outside function = [10, 30, 40, 50, 20, 30]
```

```
>>> def func(*args):
...     return args
...
>>> func(1, 2, 3, 'abc')
(1, 2, 3, 'abc')
>>> func()
()
>>> func(1)
(1,)
```

Функция может принимать и произвольное число именованных аргументов, тогда перед именем ставится \*\*

```
>>> def func(**kwargs):  
...     return kwargs  
...  
>>> func(a=1, b=2, c=3)  
{'a': 1, 'c': 3, 'b': 2}  
>>> func()  
{}  
>>> func(a='python')  
{'a': 'python'}
```

## Анонимные функции, инструкция lambda

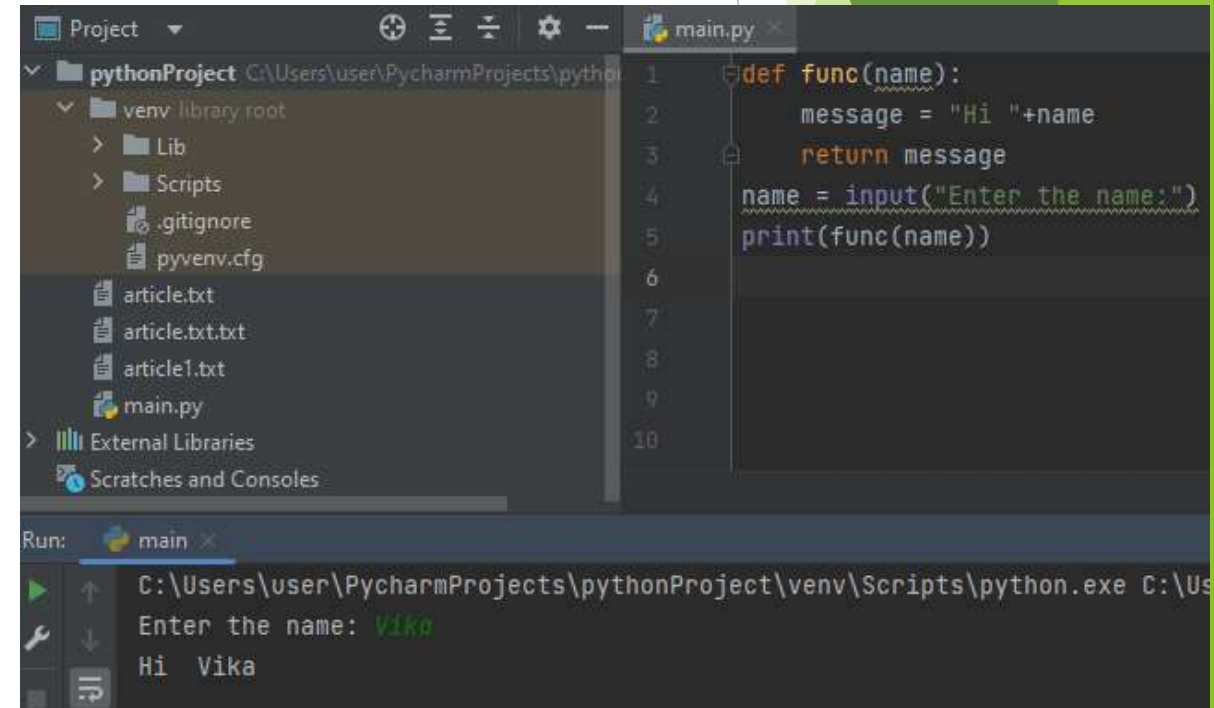
```
>>> func = lambda x, y: x + y  
>>> func(1, 2)  
3  
>>> func('a', 'b')  
'ab'  
>>> (lambda x, y: x + y)(1, 2)  
3  
>>> (lambda x, y: x + y)('a', 'b')  
'ab'
```

lambda функции, в отличие от обычной, не требуется инструкция return

## Типы аргументов:

Обязательные аргументы.  
Аргументы ключевого слова.  
По умолчанию  
.Переменной длины.

Анонимные функции могут содержать лишь одно выражение, но и выполняются они быстрее. Анонимные функции создаются с помощью инструкции lambda



```
main.py x
1 my_list = [2, 4, 8, 12, 16]
2 print(my_list[::-1])
```

```
main.py x
1 my_list = [0, 2, 4, 8, 22]
2 my_list.reverse()
3 print(my_list)
```

```
C:\Users\user\Pycharm
[16, 12, 8, 4, 2]
```

```
[22, 8, 4, 2, 0]
```

Напишите функцию `change(lst)`, которая принимает список и меняет местами его первый и последний элемент. В исходном списке минимум 3 элемента.

```
main.py x
1 def change(lst):
2     new_start = lst.pop() # Удаляем последний элемент и сохраняем его в переменную
3     new_end = lst.pop(0) # Удаляем первый элемент и сохраняем его в переменную
4     lst.append(new_end) # Добавляем к списку новый последний элемент
5     lst.insert(0, new_start) # Добавляем к списку новый первый элемент
6     return lst
7
8
9 # Тесты
10 print(change([1, 7, 3]))
11 print(change([1, 2, 9, 4, 11]))
12 print(change(['н', 'м', 'о', 'р']))
```

```
Run: main x
C:\Users\user\PycharmProje
[3, 7, 1]
[11, 2, 9, 4, 1]
['р', 'м', 'о', 'н']
```

```

1 str1 = "Программирование"
2 str2 = "Базы данных"
3 print(str1.upper())
4 print(str2.lower())

```

Ln: 4, Col: 20

Run Share Command Line Arguments

ПРОГРАММИРОВАНИЕ  
базы данных

```

1 from pprint import pprint |
2 matrix = [[0.5, 0, 0, 0, 0],
3           [ 1, 0.5, 0, 0, 0],
4           [ 1, 1, 0.5, 0, 0],
5           [ 1, 1, 1, 0.5, 0],
6           [ 1, 1, 1, 1, 0.5]]
7
8 matrix_t = list(zip(*matrix))
9
10 pprint(matrix)
11 pprint(matrix_t)

```

Run Share Comm

```

[[0.5, 0, 0, 0, 0],
 [1, 0.5, 0, 0, 0],
 [1, 1, 0.5, 0, 0],
 [1, 1, 1, 0.5, 0],
 [1, 1, 1, 1, 0.5]]

```

```

[(0.5, 1, 1, 1, 1),
 (0, 0.5, 1, 1, 1),
 (0, 0, 0.5, 1, 1),
 (0, 0, 0, 0.5, 1),
 (0, 0, 0, 0, 0.5)]

```

Дана строка в виде случайной последовательности чисел от 0 до 9. Требуется создать словарь, который в качестве ключей будет принимать данные числа (т. е. ключи будут типом `int`), а в качестве значений – количество этих чисел в имеющейся последовательности. Для построения словаря создайте функцию `count_it(sequence)`, принимающую строку из цифр. Функция должна вернуть словарь из 3-х самых часто встречаемых чисел.

Run: main

```

C:\Users\user\Pycharm
{8: 12, 1: 10, 2: 3}
{1: 5, 4: 4, 2: 1}
{9: 11, 4: 6, 7: 5}

```

```

1 from collections import Counter
2
3
4 def count_it(sequence):
5     return dict(Counter([int(num) for num in sequence]).most_common(3))
6
7
8 print(count_it('11111111112228888888888888'))
9 print(count_it('12344441111'))
10 print(count_it('76775774433116655449999999999944'))
11

```



```

a = int(input('a = ', )) # запрашиваем первый коэффициент
b = int(input('b = ', )) # запрашиваем второй коэффициент
c = int(input('c = ', )) # запрашиваем третий коэффициент

```

```

if a != 0 and b % 2 == 0 and c != 0: # решение по сокращенной формуле, т.к. b - четное
    k = b / 2
    d1 = k ** 2 - a * c
    k1 = (-k + d1 ** 0.5) / a
    k2 = (-k - d1 ** 0.5) / a
    print('так как коэффициент b - четное число, решаем по сокращенной формуле')
    print(f'k1 = {k1}')
    print(f'k2 = {k2}')

```

```

if a != 0 and b % 2 != 0 and c != 0: # решение полного уравнения
    d = b ** 2 - 4 * a * c

```

```

    if d > 0:
        k1 = (-b + d ** 0.5) / (2 * a)
        print(f'дискриминант равен: {d}')
        print(f'первый корень равен: {round(k1, 2)}')

        k2 = (-b - d ** 0.5) / (2 * a)
        print(f'второй корень равен: {round(k2, 2)}')
    elif d < 0:
        print(f'так как дискриминант меньше нуля и равен: {d}')
        print('действительных корней нет')
    else:
        k = -b / (2 * a)
        print(f'уравнение имеет один корень: {k}')

```

```

if a != 0 and c != 0 and b == 0: # решение уравнения при b = 0
    if (-c / a) >= 0:
        k1 = (-c / a) ** 0.5
        print(f'первый корень равен: {k1}')
        k2 = (-1) * ((-c / a) ** 0.5)
        print(f'второй корень равен: {k2}')
    if (-c / a) < 0:
        print(f' -c / a = : {-c / a}, т.е. < 0, поэтому действительных корней нет')

```

```

if a != 0 and c == 0 and b != 0: # решение уравнения при c = 0
    print(f'корень уравнения равен либо нулю, либо {-b / a}')

```

```

if a != 0 and b == 0 and c == 0: # решение уравнения при b = 0 и c = 0
    print(f'корни уравнения равны нулю, a*x**2 = 0')

```

a =  
3  
b =  
1  
c =  
5

так как дискриминант меньше нуля и равен: -59  
действительных корней нет

a =  
1  
b =  
-7  
c =  
12

дискриминант равен: 1  
первый корень равен: 4.0  
второй корень равен: 3.0

a =  
1  
b =  
10  
c =  
25

так как коэффициент b - четное число, решаем по сокращенной формуле  
k1 = -5.0  
k2 = -5.0

Имеется ряд словарей с пересекающимися ключами (значения - положительные числа). Напишите 2 функции, которые делают с массивом словарей следующие операции: 1-ая функция `max_dct(*dicts)` формирует новый словарь по правилу: Если в исходных словарях есть повторяющиеся ключи, выбираем среди их значений максимальное и присваиваем этому ключу (например, в словаре\_1 есть ключ "a" со значением 5, и в словаре\_2 есть ключ "a", но со значением 9. Выбираем максимальное значение, т. е. 9, и присваиваем ключу "a" в уже новом словаре). Если ключ не повторяется, то он просто переносится со своим значением в новый словарь (например, ключ "c" встретился только у одного словаря, а у других его нет. Следовательно, переносим в новый словарь этот ключ вместе с его значением). Сформированный словарь возвращаем. 2-ая функция `sum_dct(*dicts)` суммирует значения повторяющихся ключей. Значения остальных ключей остаются исходными. (Проводятся операции по аналогу первой функции, но берутся не максимумы, а суммы значений одноименных ключей). Функция возвращает сформированный словарь.

```
main.py
1 from collections import Counter
2 from functools import reduce
3 dict_1 = {1: 12, 2: 33, 3: 10, 4: 10, 5: 2, 6: 90}
4 dict_2 = {1: 12, 3: 7, 4: 1, 5: 2, 7: 112}
5 dict_3 = {2: 3, 3: 3, 4: 60, 6: 8, 7: 25, 8: 71}
6 dict_4 = {3: 1, 4: 13, 5: 31, 9: 9, 10: 556}
7 def sum_dct(*dicts):
8     return dict(reduce(lambda a, b: Counter(a) + Counter(b), dicts))
9 def max_dct(*dicts):
10    return dict(reduce(lambda a, b: Counter(a) | Counter(b), dicts))
11 print(max_dct(dict_1, dict_2))
12 print(sum_dct(dict_1, dict_4, dict_3))
13 print(max_dct(dict_1, dict_2, dict_3, dict_4))
14 print(sum_dct(dict_1, dict_2, dict_3, dict_4))
```

```
{1: 12, 2: 33, 3: 10, 4: 10, 5: 2, 6: 90, 7: 112}
{1: 12, 2: 36, 3: 14, 4: 83, 5: 33, 6: 98, 9: 9, 10: 556, 7: 25, 8: 71}
{1: 12, 2: 33, 3: 10, 4: 60, 5: 31, 6: 90, 7: 112, 8: 71, 9: 9, 10: 556}
{1: 24, 2: 36, 3: 21, 4: 84, 5: 35, 6: 98, 7: 137, 8: 71, 9: 9, 10: 556}
```

**Вопросы:**

**1. Что будет выведено в качестве результата следующей программы:**

*Str1 = 'str1--str2'*

*Str2 = 'str-str2'*

*Str3='request'*

*print(Str2 + ' '+Str3\*2)*

1. str1--str2 requestrequest
2. str-str2request request
3. str-str2 requestrequest
4. str-str1 requestrequest



**2. Что будет выведено в качестве результата следующей программы:**

*Str1 = 'str1--str2'*

*Str2 = 'str-str2'*

*Str3='request'*

*print(Str2.upper())*

1. **STR1--STR2**
2. **STR-STR2**
3. **REQUEST**
4. **str-str1 request**

### **3. Открытие файла на чтение и запись, обозначается режимом**

- 1. r**
- 2. b**
- 3. w**
- 4. +**

#### **4. Какая операция удаляет элемент из множества**

- 1. pop()**
- 2. remove()**
- 3. add()**
- 4. clear()**

**5. Что будет выведено в качестве результата следующей программы:**

```
def sum_range(start, end):  
    if start > end:  
        end, start = start, end  
    return sum(range(start, end + 1))
```

```
print(sum_range(2, 5))  
print(sum_range(-3, 3))  
print(sum_range(0, 4))
```

- |    |    |   |    |
|----|----|---|----|
| 1. | 13 | 1 | 11 |
| 2. | 10 | 0 | 14 |
| 3. | 13 | 0 | 9  |
| 4. | 14 | 0 | 10 |