

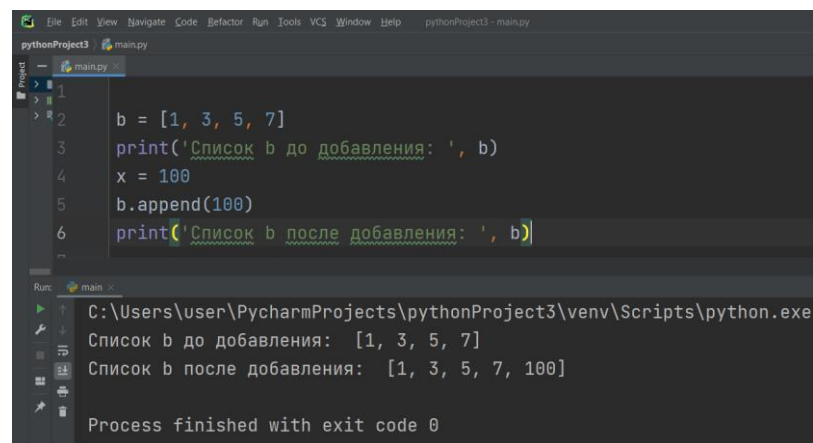
Методы списков в Python

В таблице 1 приведены основные методы, которые можно применять к спискам.

Таблица 1. Методы списков в языке Python

Метод	Для чего используется
<code>list.append(x)</code>	Добавление элемента в конец списка
<code>list.extend(L)</code>	Расширения списка <code>list</code> , добавление в конец всех элементов списка с именем <code>'L'</code>
<code>list.insert(i, x)</code>	Вставка на <code>i</code> -ый элемент значение <code>'x'</code>
<code>list.remove(x)</code>	Удаление первого элемента в списке, имеющего значение <code>'x'</code> . <code>ValueError</code> , если такого элемента не существует
<code>list.pop([i])</code>	Удаление <code>i</code> -ого элемента и возвращение его. Если индекс не указан, то удаляется последний элемент
<code>list.index(x, [start [, end]])</code>	Возвращение положения первого элемента со значением <code>'x'</code> (при этом поиск ведется от <code>start</code> до <code>end</code>)
<code>list.count(x)</code>	Возвращение количества элементов со значением <code>'x'</code>
<code>list.sort([key = функция])</code>	Сортировка списка на основе функции
<code>list.reverse()</code>	Разворачивание списка
<code>list.copy()</code>	Поверхностная копия списка
<code>list.clear()</code>	Очищение списка

Покажем реализацию метода **list.append()** (рис. 1):

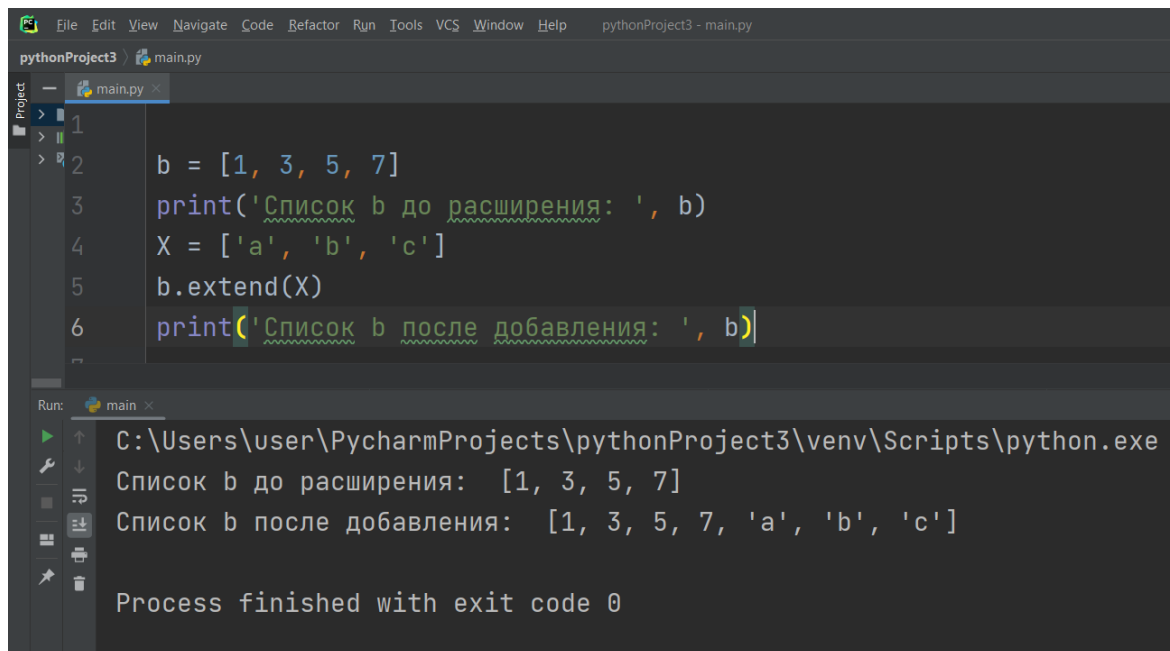


```
File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject3 - main.py
pythonProject3 - main.py
1
2 b = [1, 3, 5, 7]
3 print('Список b до добавления: ', b)
4 x = 100
5 b.append(100)
6 print('Список b после добавления: ', b)

Run: main
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe
Список b до добавления: [1, 3, 5, 7]
Список b после добавления: [1, 3, 5, 7, 100]
Process finished with exit code 0
```

Рисунок 1 - Реализация метода `list.append()`

Расширим список `'b'` при помощи метода **list.extend()** (рис. 2):

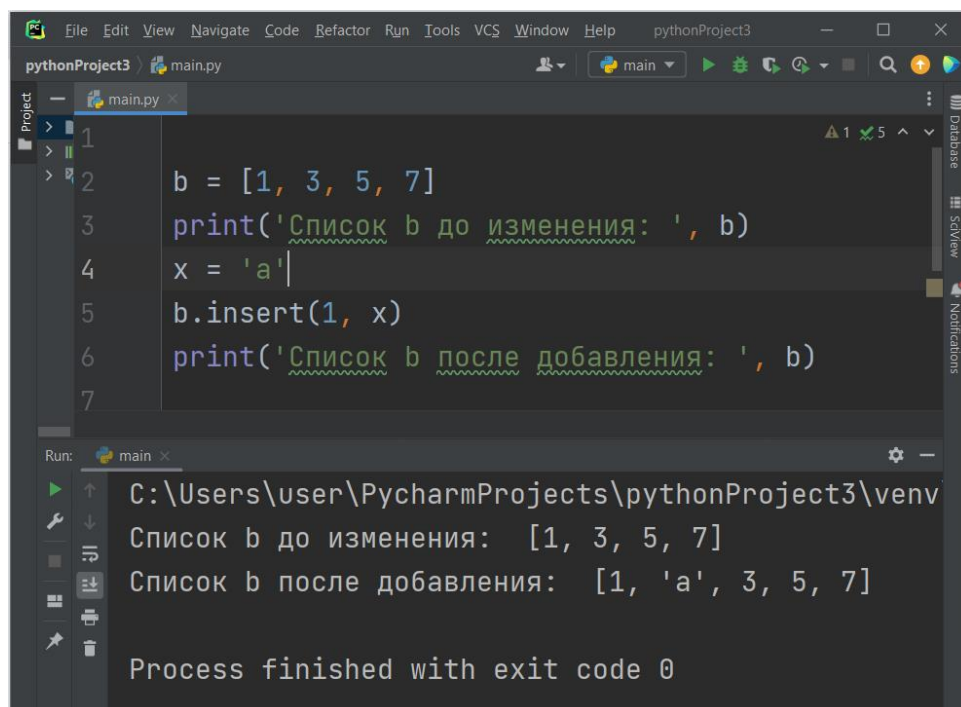


```
pythonProject3 - main.py
main.py
1
2 b = [1, 3, 5, 7]
3 print('Список b до расширения: ', b)
4 X = ['a', 'b', 'c']
5 b.extend(X)
6 print('Список b после добавления: ', b)

Run: main
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe
Список b до расширения: [1, 3, 5, 7]
Список b после добавления: [1, 3, 5, 7, 'a', 'b', 'c']
Process finished with exit code 0
```

Рисунок 2 - Реализация метода list.extend()

Теперь добавим значение на конкретную позицию при помощи специального метода **list.insert()** (рис. 3):



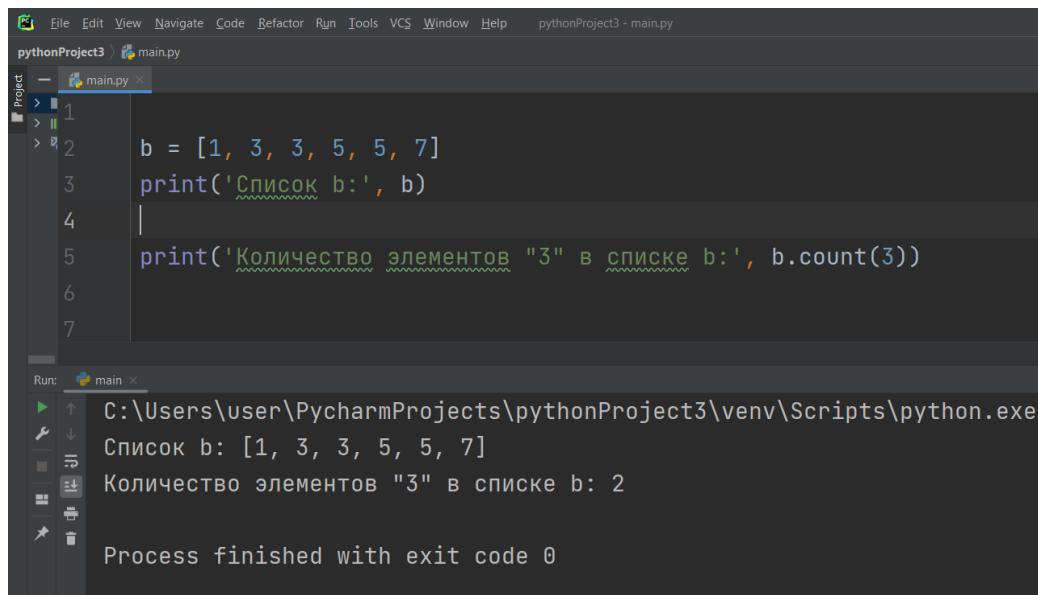
```
pythonProject3
main.py
1
2 b = [1, 3, 5, 7]
3 print('Список b до изменения: ', b)
4 x = 'a'
5 b.insert(1, x)
6 print('Список b после добавления: ', b)
7

Run: main
C:\Users\user\PycharmProjects\pythonProject3\venv
Список b до изменения: [1, 3, 5, 7]
Список b после добавления: [1, 'a', 3, 5, 7]
Process finished with exit code 0
```

Рисунок 3 - Реализация метода list.insert()

Как можно заметить, элемент **‘a’** добавился на вторую позицию в список **‘b’** (в строке 5 указан индекс 1, так как первый элемент списка имеет индекс 0).

При помощи метода **list.count()** можно посчитать количество элементов, указанного в круглых скобках (рис. 4):



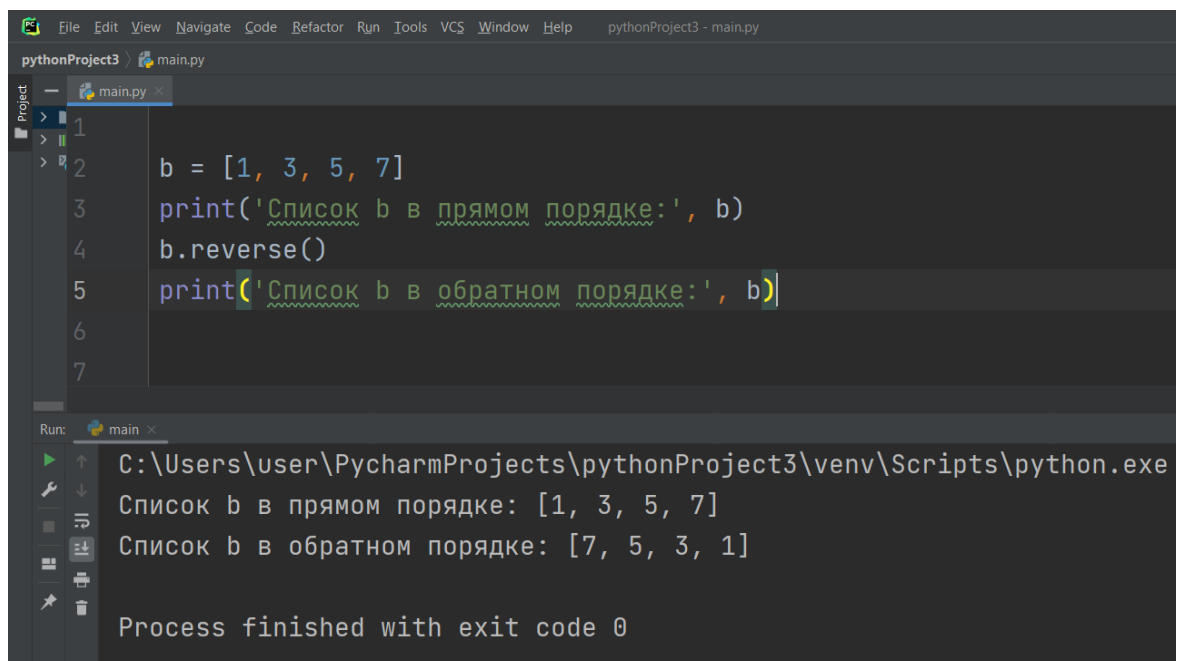
```
pythonProject3 - main.py
main.py
1
2 b = [1, 3, 3, 5, 5, 7]
3 print('Список b:', b)
4
5 print('Количество элементов "3" в списке b:', b.count(3))
6
7

Run: main
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe
Список b: [1, 3, 3, 5, 5, 7]
Количество элементов "3" в списке b: 2
Process finished with exit code 0
```

Рисунок 4 - Реализация метода list.count()

В данном случае, в строке 5 подсчитывается количество цифр ‘3’ в списке ‘b’.

При помощи метода **list.reverse()** можно “развернуть” список и вывести его элементы в обратном порядке (рис. 5):



```
pythonProject3 - main.py
main.py
1
2 b = [1, 3, 5, 7]
3 print('Список b в прямом порядке:', b)
4 b.reverse()
5 print('Список b в обратном порядке:', b)
6
7

Run: main
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe
Список b в прямом порядке: [1, 3, 5, 7]
Список b в обратном порядке: [7, 5, 3, 1]
Process finished with exit code 0
```

Рисунок 5 - Реализация метода list.reverse()

При использовании метода **list.reverse()** можно не объявлять новую переменную, достаточно использовать команду **list.reverse()**.

Метод **list.copy()** позволяет копировать список со всеми элементами. Для реализации данного метода необходима новая переменная (рис. 6):

```
pythonProject3 > main.py
1
2 b = [1, 3, 5, 7]
3 print('Оригинал списка b:', b)
4 c = b.copy()
5 print('Копия списка b:', c)
6
7

Run: main
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe
Оригинал списка b: [1, 3, 5, 7]
Копия списка b: [1, 3, 5, 7]

Process finished with exit code 0
```

Рисунок 6 - Реализация метода list.copy()

Метод **list.clear()** позволяет удалить все элементы. У данного метода нет необходимости в создании новой переменной (рис. 7):

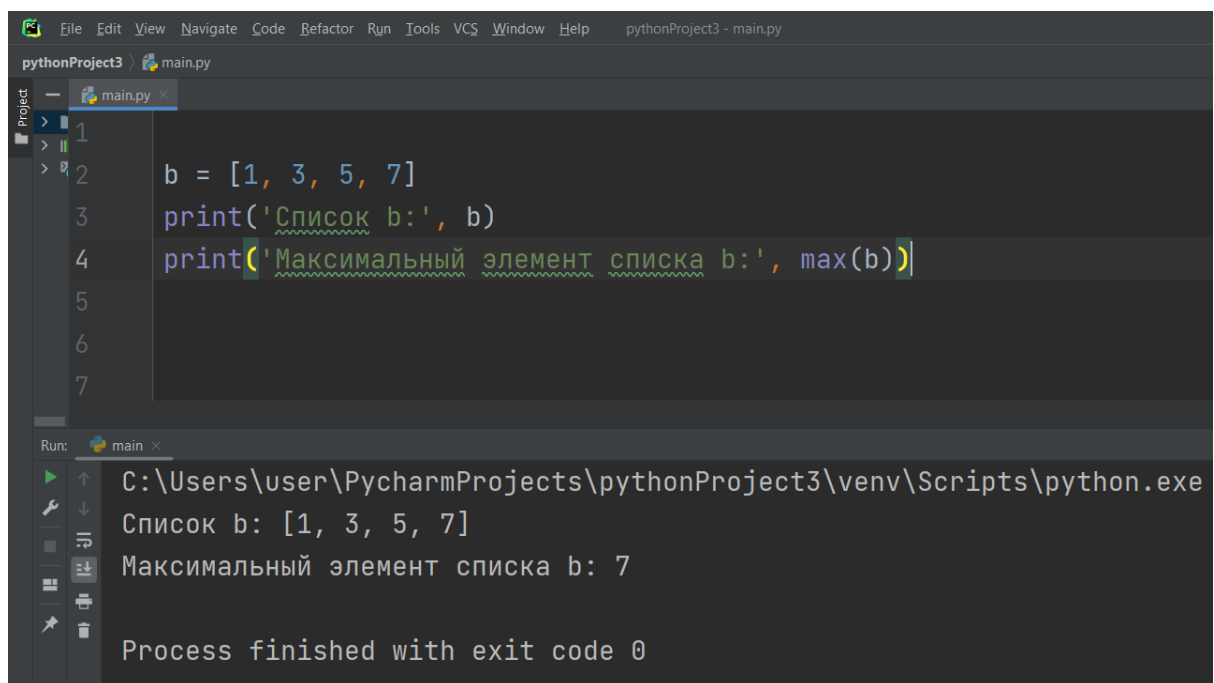
```
pythonProject3 > main.py
1
2 b = [1, 3, 5, 7]
3 print('Список b:', b)
4 b.clear()
5 print('Пустой список b:', b)
6
7

Run: main
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe
Список b: [1, 3, 5, 7]
Пустой список b: []

Process finished with exit code 0
```

Рисунок 7 - Реализация метода list.clear()

Команды max() и min(). Данные команды позволяют найти максимальный и минимальный элементы в списке. Найдем максимальный элемент в списке 'b' при помощи команды **max()** (рис. 8):



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The main editor window displays a file named `main.py` with the following code:

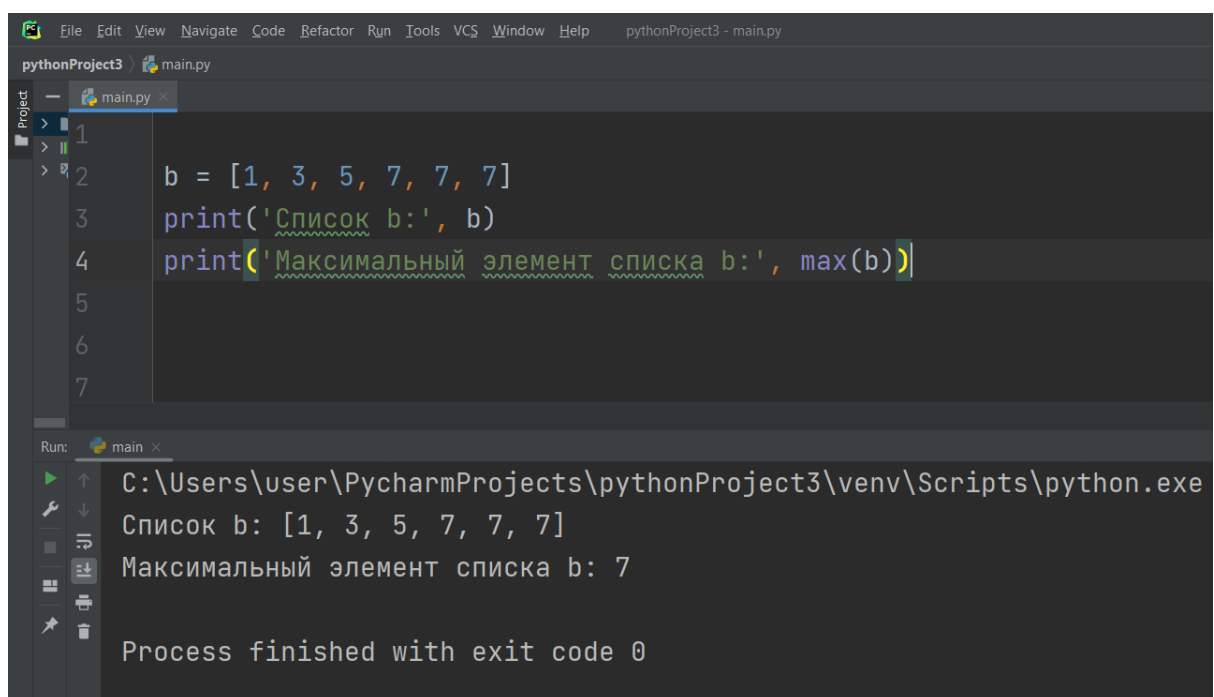
```
1  
2 b = [1, 3, 5, 7]  
3 print('Список b:', b)  
4 print('Максимальный элемент списка b:', max(b))  
5  
6  
7
```

Below the editor, the Run console shows the output of the script:

```
Run: main ×  
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe  
Список b: [1, 3, 5, 7]  
Максимальный элемент списка b: 7  
  
Process finished with exit code 0
```

Рисунок 8 - Реализация команды `max()` с одним максимальным элементом

Стоит отметить, что если максимальных элементов в списке будет несколько, то вывод команды `max()` не изменится (рис. 9):



The screenshot shows the PyCharm IDE interface. The main editor window displays a file named `main.py` with the following code:

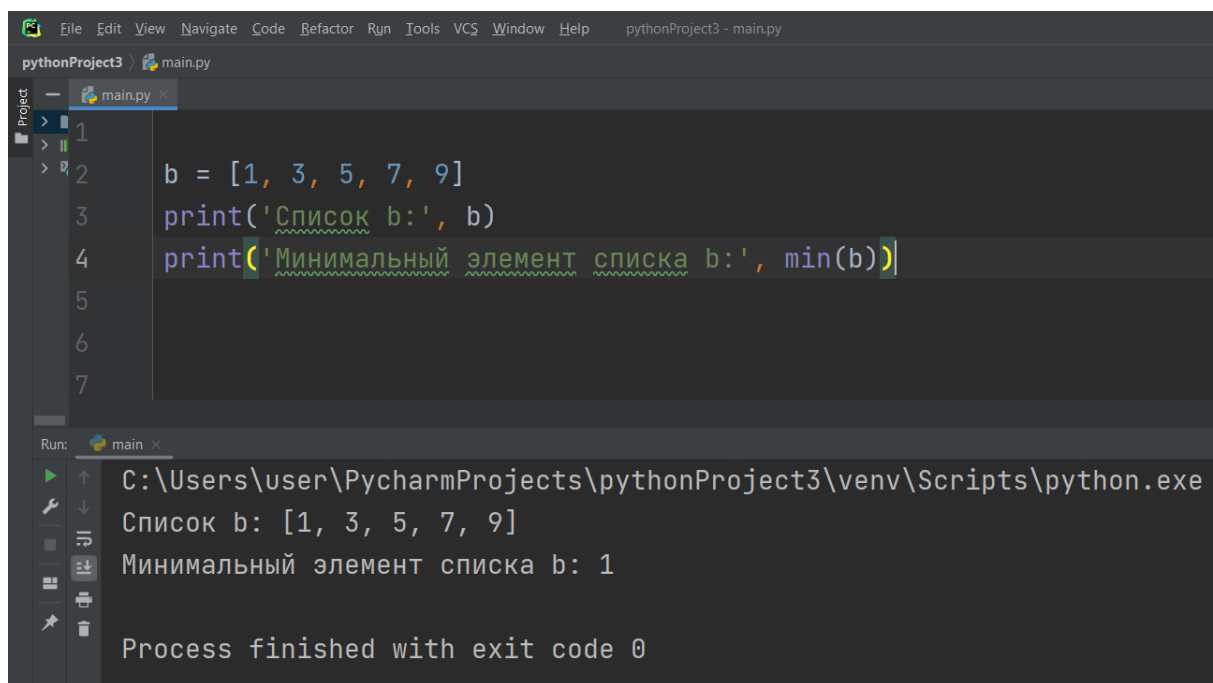
```
1  
2 b = [1, 3, 5, 7, 7, 7]  
3 print('Список b:', b)  
4 print('Максимальный элемент списка b:', max(b))  
5  
6  
7
```

Below the editor, the Run console shows the output of the script:

```
Run: main ×  
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe  
Список b: [1, 3, 5, 7, 7, 7]  
Максимальный элемент списка b: 7  
  
Process finished with exit code 0
```

Рисунок 9 - Реализация команды `max()` с несколькими максимальными элементами

Команда `min()` работает аналогично, только ищет не максимальный элемент в списке, а минимальный (рис. 10):



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The main editor window displays a file named `main.py` with the following code:

```
1  
2 b = [1, 3, 5, 7, 9]  
3 print('Список b:', b)  
4 print('Минимальный элемент списка b:', min(b))  
5  
6  
7
```

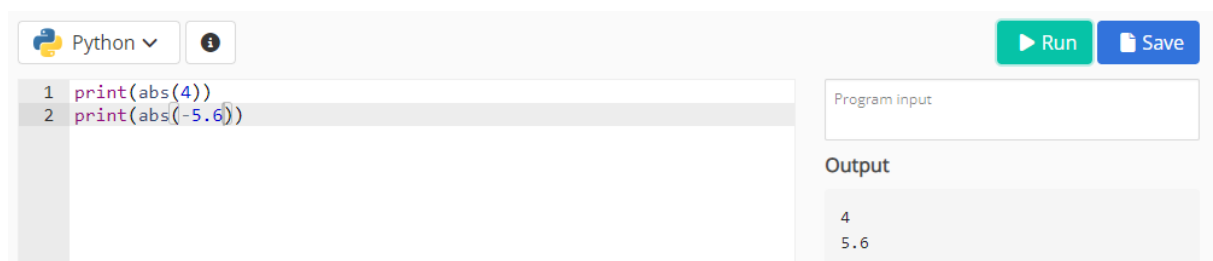
Below the editor is the Run console. It shows the command `C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe` and the output:

```
Список b: [1, 3, 5, 7, 9]  
Минимальный элемент списка b: 1  
  
Process finished with exit code 0
```

Рисунок 10 - Реализация команды `min()`

Самые популярные встроенные функции

Функция **`abs()`** в языке *Python* возвращает абсолютное значение числа. Если это комплексное число, то абсолютным значением будет величина целой и мнимой частей. Посмотрим на пример использования такой функции (рис. 11):



The screenshot shows an online Python interpreter interface. The code editor contains the following code:

```
1 print(abs(4))  
2 print(abs(-5.6))
```

On the right side, there are buttons for 'Run' and 'Save'. Below them is a 'Program input' field and an 'Output' section. The output section displays the results of the code execution:

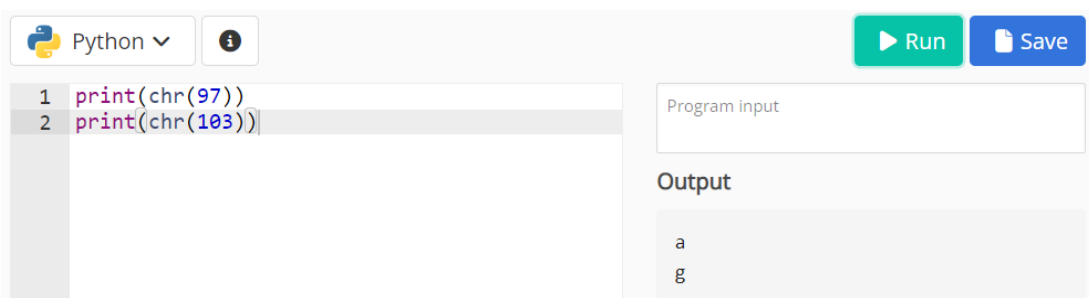
```
4  
5.6
```

Рисунок 11 - Синтаксис встроенной функции `abs()`

Как можно заметить, при вводе положительного числа, в консоли выводится то же самое число (первая строка *Output*), а при вводе отрицательного - выводится число, обратное введенному (число без минуса) (последняя строка *Output*).

Функция **`chr()`** возвращает строку, представляющую символ *Unicode* для переданного числа.

Она является противоположностью функции **`ord()`**, которая принимает символ и возвращает его числовой код (рис. 12):



The screenshot shows an online Python interpreter interface. The code editor contains the following code:

```
1 print(chr(97))  
2 print(chr(103))
```

On the right side, there are buttons for 'Run' and 'Save'. Below them is a 'Program input' field and an 'Output' section. The output section displays the results of the code execution:

```
a  
g
```

Рисунок 12 - Синтаксис встроенной функции `chr()`

С отображением других символов можно ознакомиться в таблице 2.

Таблица 2. Кодирование букв и других символов в *Unicode*

0	NUL	1	SOH	2	STX	3	ETX	4	EOT	5	ENQ	6	ACK	7	BEL
8	BS	9	HT	10	LF	11	VT	12	FF	13	CR	14	SO	15	SI
16	DLE	17	DC1	18	DC2	19	DC3	20	DC4	21	NAK	22	SYN	23	ETB
24	CAN	25	EM	26	SUB	27	ESC	28	FS	29	GS	30	RS	31	US
32	SP	33	!	34	"	35	#	36	\$	37	%	38	&	39	'
40	(41)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[92	\	93]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124		125	}	126	~	127	DEL

Функция **complex()** принимает целые числа или строки и возвращает соответствующее комплексное число (число, представленное в форме $a + bi$). Если передать неподходящее значение, то вернется ошибка (рис. 13):

The screenshot shows a Python IDE with the following code in the editor:

```
1 print(complex(3))
2 print(complex(-3, -2))
3 print(complex(1, 2, 3, 4, 5))
```

The output pane shows the following error message:

```
(3+0j)
(-3-2j)
Traceback (most recent call
last):
  File "/tmp/main.py", line 2, in
<module>
    import user_code
  File "/tmp/user_code.py", line
3, in <module>
    print(complex(1, 2, 3, 4, 5))
TypeError: complex() takes at
most 2 arguments (5 given)
```

Рисунок 13 - Синтаксис встроенной функции *complex()*

Функция **enumerate()** в качестве параметра принимает последовательность. После этого она перебирает каждый элемент и возвращает его вместе со счетчиком в виде перечисляемого объекта. Основной особенностью таких объектов является возможность размещать их в цикле для перебора (рис. 14):

The screenshot shows a Python IDE with a code editor on the left and an output panel on the right. The code editor contains two lines of Python code: `1 x = "Python"` and `2 print(list(enumerate(x)))`. The output panel shows the result of the execution: `[(0, 'P'), (1, 'y'), (2, 't'), (3, 'h'), (4, 'o'), (5, 'n')]`. Above the output panel, there is a text input field labeled "Program input" and a status message: "[Execution complete with exit code 0]".

Рисунок 14 - Синтаксис встроенной функции *enumerate()*

Следующая функция, о которой мы будем говорить - **eval()**. Данная функция обрабатывает переданное в нее выражение и исполняет его как выражение на языке *Python*.

После этого возвращается значение. Чаще всего, данная функция используется для выполнения математических функций (рис. 15):

The screenshot shows a Python IDE with a code editor on the left and an output panel on the right. The code editor contains three lines of Python code: `1 print(eval('2+4'))`, `2 print(eval('3*10'))`, and `3 print(eval('5/2.5'))`. The output panel shows the results of the execution: `6`, `30`, and `2.0`. Above the output panel, there is a text input field labeled "Program input" and buttons for "Run" and "Save".

Рисунок 15 - Синтаксис встроенной функции *eval()*

Функция **float()** конвертирует число или строку в число с плавающей точкой и возвращает результат. Если из-за некорректного ввода конвертация не происходит, то вернется соответствующая ошибка (рис. 16):

The screenshot shows a Python IDE with a code editor on the left and an output panel on the right. The code editor contains three lines of Python code: `1 print(float('596'))`, `2 print(eval('26'))`, and `3 print(eval(2.5))`. The output panel shows the results of the execution: `596.0` and `26`, followed by a `TypeError: eval() arg 1 must be a string, bytes or code object`. Above the output panel, there is a text input field labeled "Program input" and buttons for "Run" and "Save".

Рисунок 16 - Синтаксис встроенной функции *float()*

Функция **help()** предоставляет простой способ получения доступа к документации языка *Python* без использования интернета для любой функции, ключевого слова или модуля (рис. 17):

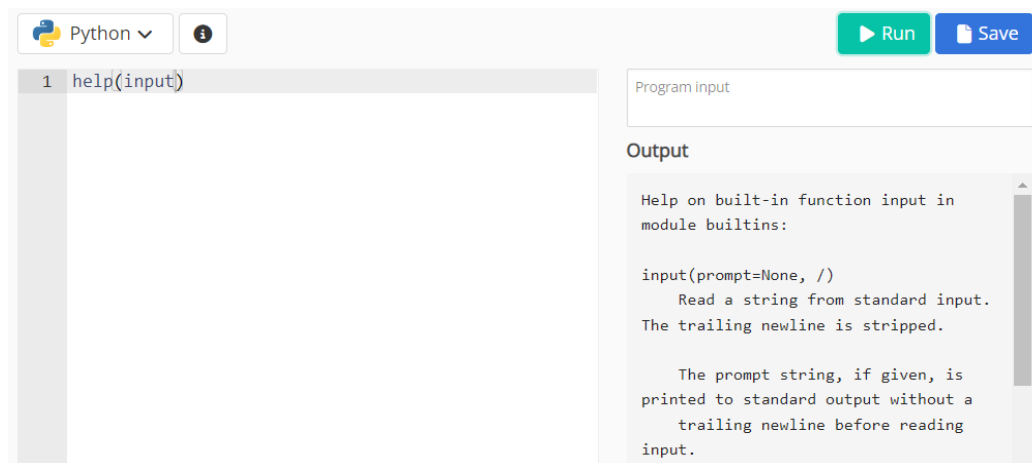


Рисунок 17 - Синтаксис встроенной функции *help()*

Функция **input()** позволяет пользователю вводить данные разного типа. Вызов данной функции предоставляет пользователю возможность ввести на экране текст. Затем, он конвертируется в строку и возвращается в программу (рис. 18):



Рисунок 18 - Синтаксис встроенной функции *input()*

Функция **int()** возвращает всегда целое число из объекта, переданного в параметре. Данная функция может конвертировать числа с разным основанием (шестнадцатеричные, двоичные и т.д.) и целые в том числе. Посмотрим как использовать данную функцию (рис. 19):

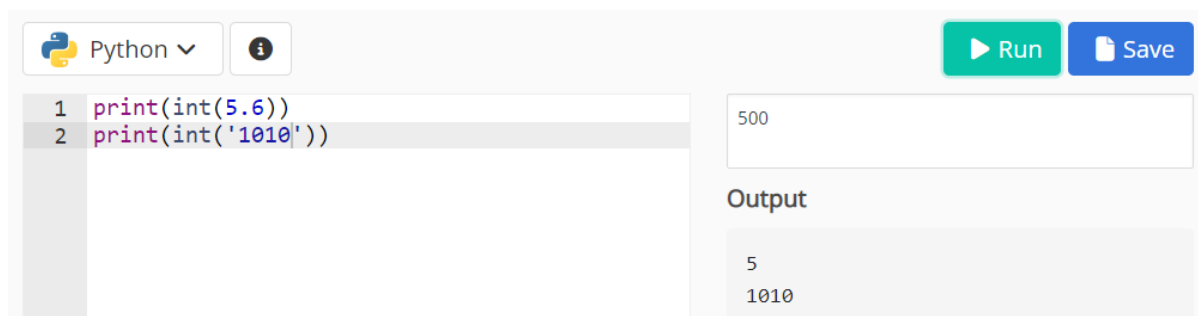
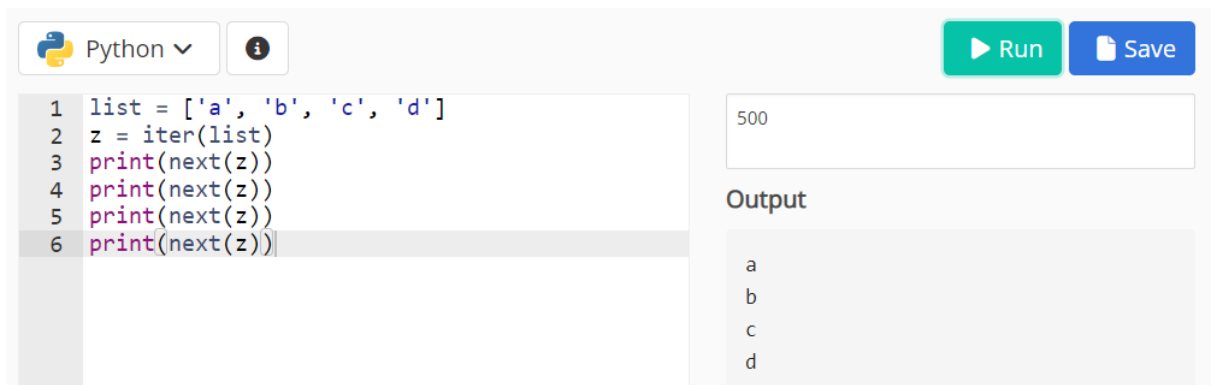


Рисунок 19 - Синтаксис встроенной функции *int()*

Следующая функция, о которой мы будем говорить - **iter()**. Данная функция принимает объект и возвращает итерируемый объект. Например, используя данную функцию, можно перебирать различные объекты, посмотрим на наглядный пример использования данной функции (рис. 20):



```
1 list = ['a', 'b', 'c', 'd']
2 z = iter(list)
3 print(next(z))
4 print(next(z))
5 print(next(z))
6 print(next(z))
```

500

Output

a
b
c
d

Рисунок 20 - Синтаксис встроенной функции *iter()*

В вышеуказанном примере, изначально, была задана последовательность чисел английского алфавита под общим именем **list**.

Затем, при участии переменной **z** была использована данная встроенная функция - **iter(list)**. После этого, при помощи ключевого слова **next**, выводились по очереди первый, второй и следующие элементы последовательности **list**.

Помимо данных функций, есть еще много полезных. Одна из них - **max()**. Данная функция используется для нахождения максимального значения в последовательности, итерируемом объекте и так далее. В параметрах можно менять способ вычисления максимального значения (в данном случае, как это будет удобно пользователю) (рис. 21):



```
1 a = [1, 3, 5, 7, 9]
2 b = [2, 4, 6, 8, 10]
3 print("Максимальное число в последовательности a: ", max(a))
4 print("Максимальное число в последовательности b: ", max(b))
```

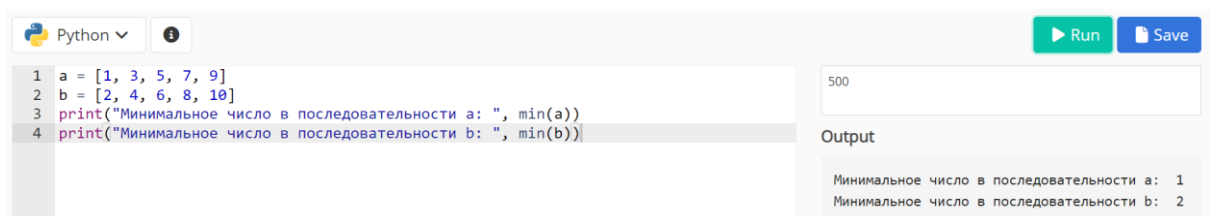
500

Output

Максимальное число в последовательности a: 9
Максимальное число в последовательности b: 10

Рисунок 21 - Синтаксис встроенной функции *max()*

Аналогичным образом работает функция **min()** (рис. 22):



```
1 a = [1, 3, 5, 7, 9]
2 b = [2, 4, 6, 8, 10]
3 print("Минимальное число в последовательности a: ", min(a))
4 print("Минимальное число в последовательности b: ", min(b))
```

500

Output

Минимальное число в последовательности a: 1
Минимальное число в последовательности b: 2

Рисунок 22 - Синтаксис встроенной функции *min()*

Иногда, при работе с большими массивами данных, необходимо знать длину того или иного объекта. Как раз для таких задач используется встроенная функция **len()**. Посмотрим как ее можно использовать (рис. 23):



```
1 x = [1, 3, 5, 7, 9]
2 y = [2, 4, 6, 8, 10, 12]
3 print("Количество элементов в последовательности x: ", len(x))
4 print("Количество элементов в последовательности y: ", len(y))
```

500

Output

Количество элементов в последовательности x: 5
Количество элементов в последовательности y: 6

Рисунок 23 - Синтаксис встроенной функции *len()*

Практические задачи

Задача 1.

Используя встроенные функции `max()` и `sum()`, найти максимальное значение в последовательности `x = [1, 3, 5, 10, 11]` и посчитать сумму элементов в последовательности `y = [10, 55, 12, 100]`.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



```
Python
1 x = [1, 3, 5, 10, 11]
2 y = [10, 55, 12, 100]
3 print("Максимальное число в последовательности x: ", max(x))
4 print("Сумма элементов в последовательности y: ", sum(y))
```

Run Save

Output

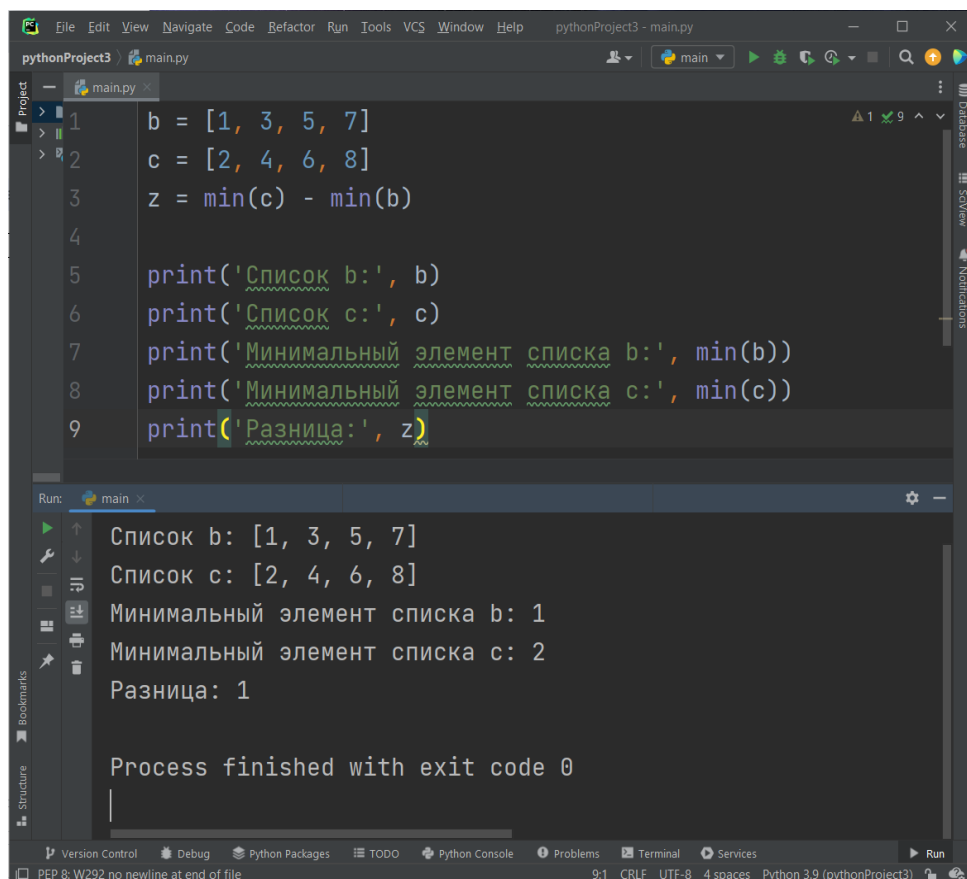
Максимальное число в последовательности x: 11
Сумма элементов в последовательности y: 177

Задача 2.

Создать программу, в которой вводятся списки `'b' = [1, 3, 5, 7]` и `'c' = [2, 4, 6, 8]`. Необходимо найти минимальные элементы в списках `'b'` и `'c'`. Вывести списки и минимальные элементы на консоль. Затем найти произведение данных элементов и вывести его на консоль.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



```
pythonProject3 - main.py
1 b = [1, 3, 5, 7]
2 c = [2, 4, 6, 8]
3 z = min(c) - min(b)
4
5 print('Список b:', b)
6 print('Список c:', c)
7 print('Минимальный элемент списка b:', min(b))
8 print('Минимальный элемент списка c:', min(c))
9 print('Разница:', z)
```

Run: main

Список b: [1, 3, 5, 7]
Список c: [2, 4, 6, 8]
Минимальный элемент списка b: 1
Минимальный элемент списка c: 2
Разница: 1



Process finished with exit code 0

Задача 3.

Используя встроенные функции **min()** и **reversed()**, найти минимальное значение в последовательности **a** = [-2, 0, 4, 8, 10] и развернуть порядок элементов в последовательности **b** = [6, 4, 2, 0].

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:

```
Python   Run Save
```

```
1 a = [-2, 0, 4, 8, 10]
2 b = [6, 4, 2, 0]
3 c = reversed(b)
4 print("Минимальное число в последовательности a: ", min(a))
5 print("Обратный порядок следования элементов в последовательности b: ", list(c))
```

Output

```
Минимальное число в последовательности a: -2
Обратный порядок следования элементов в
последовательности b: [0, 2, 4, 6]
```

Задача 4.

Имеется некая числовая последовательность **x** = [3, 7, 2, 5, 10]. Необходимо возвести каждый элемент данной последовательности в третью степень и посчитать сумму элементов после данной операции.

Решение.

Для решения данной задачи нам понадобятся несколько инструментов - функция **sum()**, операция по возведению в степень «**» и цикл **for**:

```
Python   Run Save
```

```
1 x = [3, 7, 2, 5, 10]
2
3 print("Данная последовательность x:", x)
4 print("Результат:", sum(n**3 for n in x))
```

Output

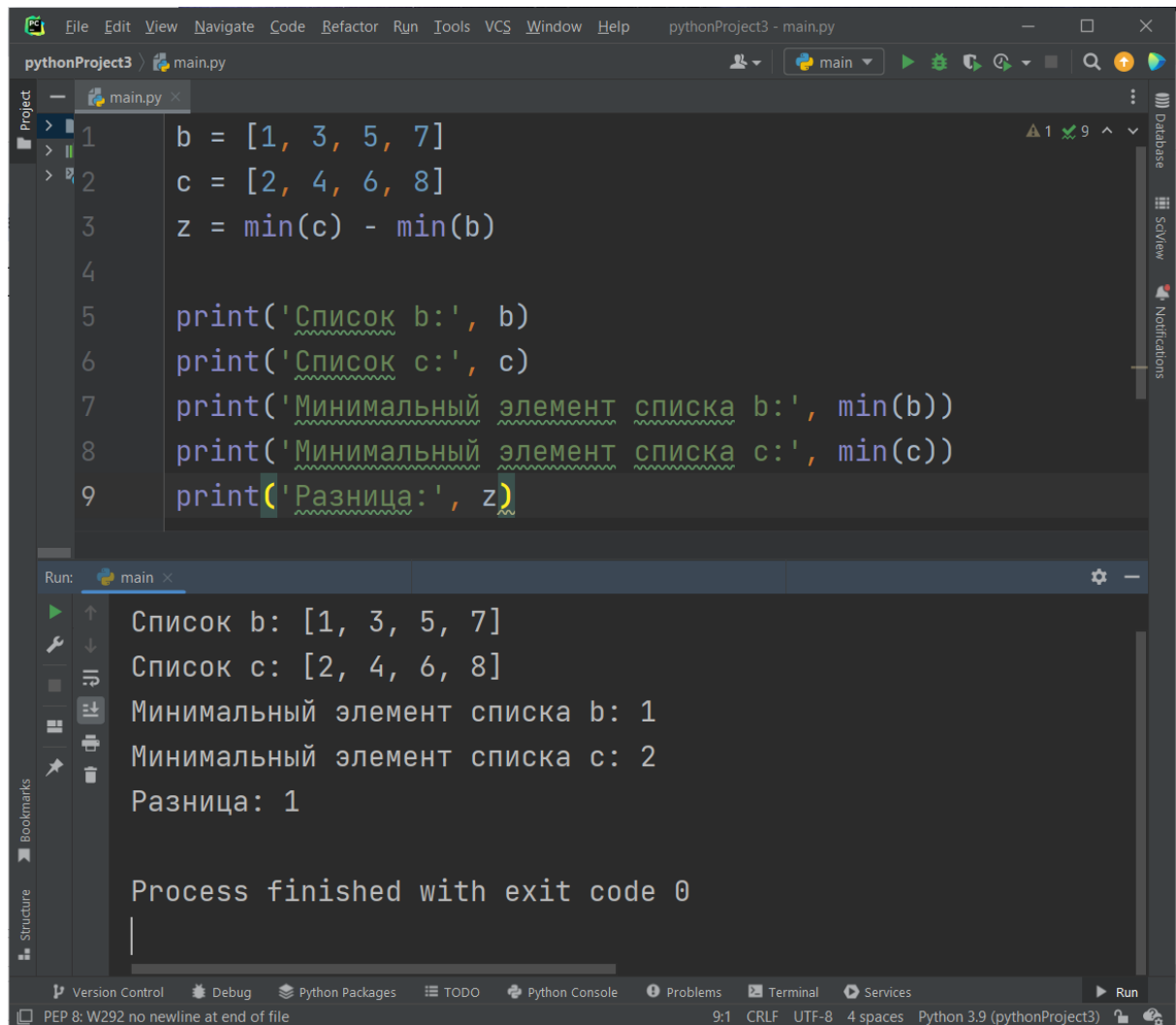
```
Данная последовательность x: [3, 7, 2, 5, 10]
Результат: 1503
```

Задача 5.

Создать программу, в которой вводятся списки **'b'** = [1, 3, 5, 7] и **'c'** = [2, 4, 6, 8]. Необходимо найти минимальные элементы в списках **'b'** и **'c'**. Вывести списки и минимальные элементы на консоль. Затем найти произведение данных элементов и вывести его на консоль.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



The screenshot shows an IDE window titled 'pythonProject3 - main.py'. The editor displays the following Python code:

```
1 b = [1, 3, 5, 7]
2 c = [2, 4, 6, 8]
3 z = min(c) - min(b)
4
5 print('Список b:', b)
6 print('Список c:', c)
7 print('Минимальный элемент списка b:', min(b))
8 print('Минимальный элемент списка c:', min(c))
9 print('Разница:', z)
```

Below the editor, the 'Run' panel shows the output of the program:

```
Список b: [1, 3, 5, 7]
Список c: [2, 4, 6, 8]
Минимальный элемент списка b: 1
Минимальный элемент списка c: 2
Разница: 1

Process finished with exit code 0
```

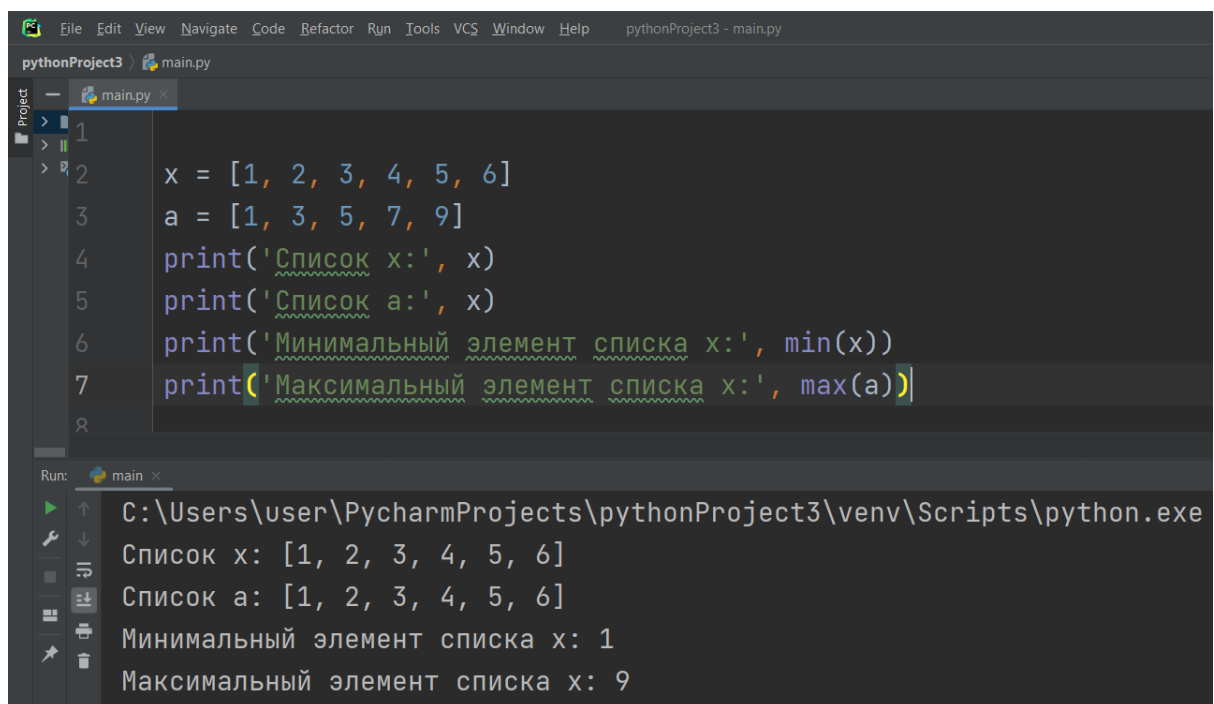
The IDE interface includes a menu bar (File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help), a toolbar with icons for running and debugging, and a sidebar with 'Project', 'Database', 'ScoutView', and 'Notifications' panels. The status bar at the bottom indicates 'PEP 8: W292 no newline at end of file', '9:1 CRLF UTF-8 4 spaces Python 3.9 (pythonProject3)', and a 'Run' button.

Задача 6.

Создать программу, в которой вводятся списки 'x' = [1, 2, 3, 4, 5, 6] и 'a' = [1, 3, 5, 7, 9]. Необходимо найти максимальный элемент в списке 'a' и минимальный элемент в списке 'x'. Данные значения вывести на консоль.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The project name is 'pythonProject3' and the active file is 'main.py'. The code editor contains the following Python code:

```
1 x = [1, 2, 3, 4, 5, 6]
2 a = [1, 3, 5, 7, 9]
3 print('Список x:', x)
4 print('Список a:', x)
5 print('Минимальный элемент списка x:', min(x))
6 print('Максимальный элемент списка x:', max(a))
```

The Run window at the bottom shows the execution output for 'main.py':

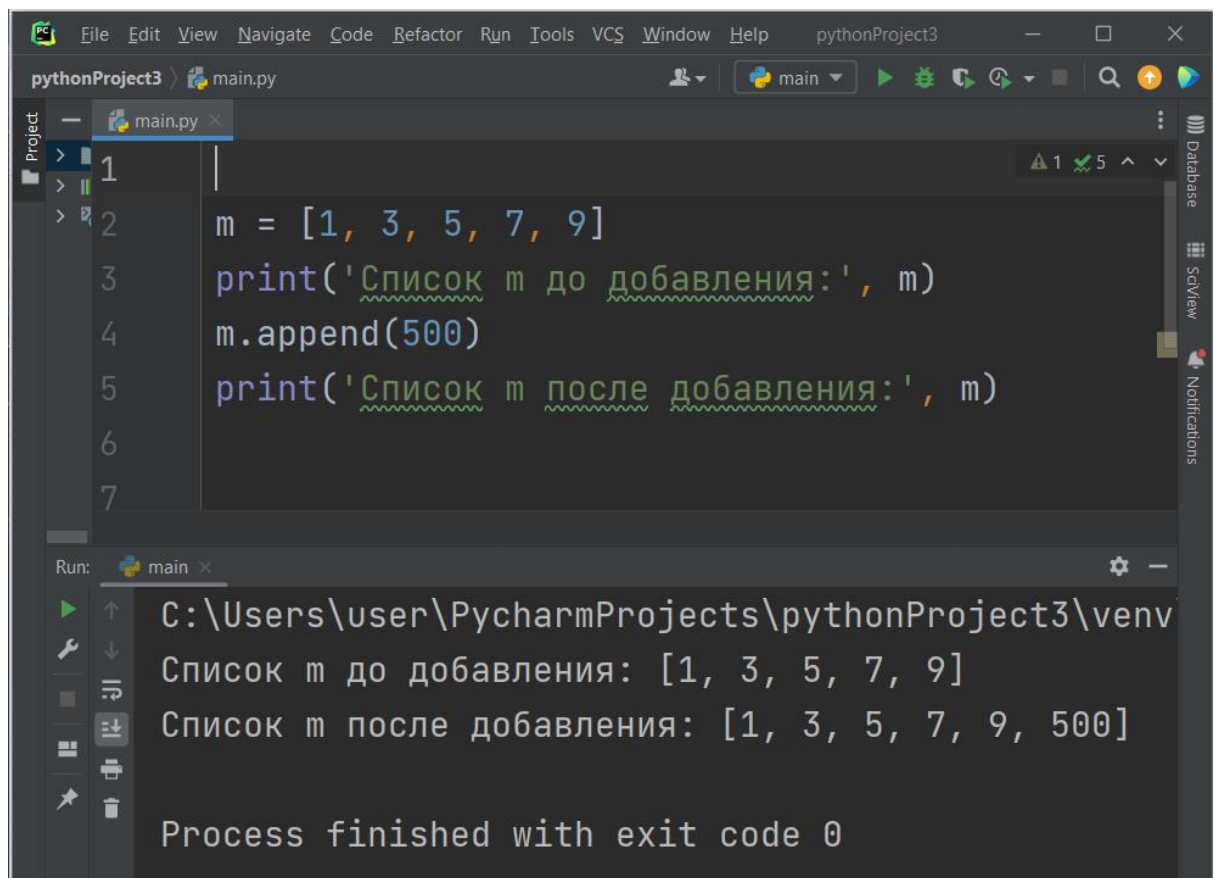
```
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe
Список x: [1, 2, 3, 4, 5, 6]
Список a: [1, 2, 3, 4, 5, 6]
Минимальный элемент списка x: 1
Максимальный элемент списка x: 9
```

Задача 7.

Создать программу, в которой вводится список `'m' = [1, 3, 5, 7, 9]`. Необходимо в конец данного списка добавить число 500 и вывести модифицированный список на консоль.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The project name is 'pythonProject3' and the active file is 'main.py'. The code editor contains the following Python code:

```
1 m = [1, 3, 5, 7, 9]
2 print('Список m до добавления:', m)
3 m.append(500)
4 print('Список m после добавления:', m)
```

The Run window at the bottom shows the execution output for 'main.py':

```
C:\Users\user\PycharmProjects\pythonProject3\venv
Список m до добавления: [1, 3, 5, 7, 9]
Список m после добавления: [1, 3, 5, 7, 9, 500]


Process finished with exit code 0
```

Задача 8.

Даны две числовые последовательности: **a** = [3, 6, 8, 10] и **b** = [11, 15, 25, 32]. Необходимо сравнить максимальный элемент последовательности **a** и минимальный элемент последовательности **b**. После сравнения вывести наименьший из них.

Решение.

Для решения данной задачи нам понадобятся функции **max()**, **min()**, а также условный оператор **if**:



```
1 a = [3, 6, 8, 10]
2 b = [11, 15, 25, 32]
3
4 print("Максимальный элемент последовательности a:", max(a))
5 print("Минимальный элемент последовательности b:", min(b))
6
7 if max(a) < min(b):
8     print("Наименьшее из них:", max(a))
9 if min(b) < max(a):
10    print("Наименьшее из них:", min(b))
```

Output

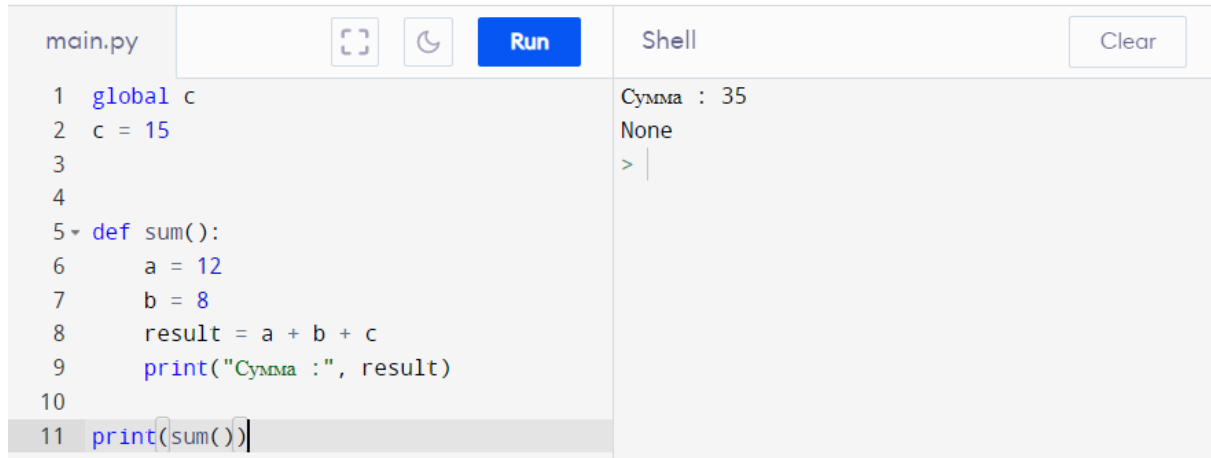
```
Максимальный элемент последовательности a: 10
Минимальный элемент последовательности b: 11
Наименьшее из них: 10
[Execution complete with exit code 0]
```

Задача 9.

Написать программу с одной глобальной переменной **x**, которая считает произведение трех целых чисел.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



```
main.py
1 global c
2 c = 15
3
4
5 def sum():
6     a = 12
7     b = 8
8     result = a + b + c
9     print("Сумма :", result)
10
11 print(sum())
```

Shell

```
Сумма : 35
None
> |
```

Задача 10.

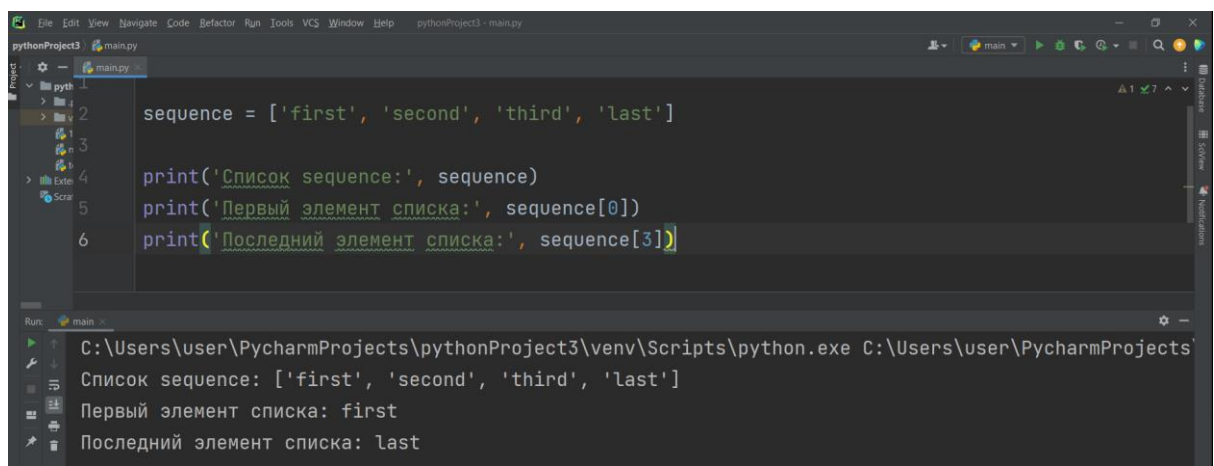
Написать программу, в которой создается следующий список:

'sequence' = ['first', 'second', 'three', 'last'].

Вывести данный список полностью, а затем только первый и последний элементы.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



```
pythonProject3 - main.py
1 sequence = ['first', 'second', 'third', 'last']
2
3
4 print('Список sequence:', sequence)
5 print('Первый элемент списка:', sequence[0])
6 print('Последний элемент списка:', sequence[3])

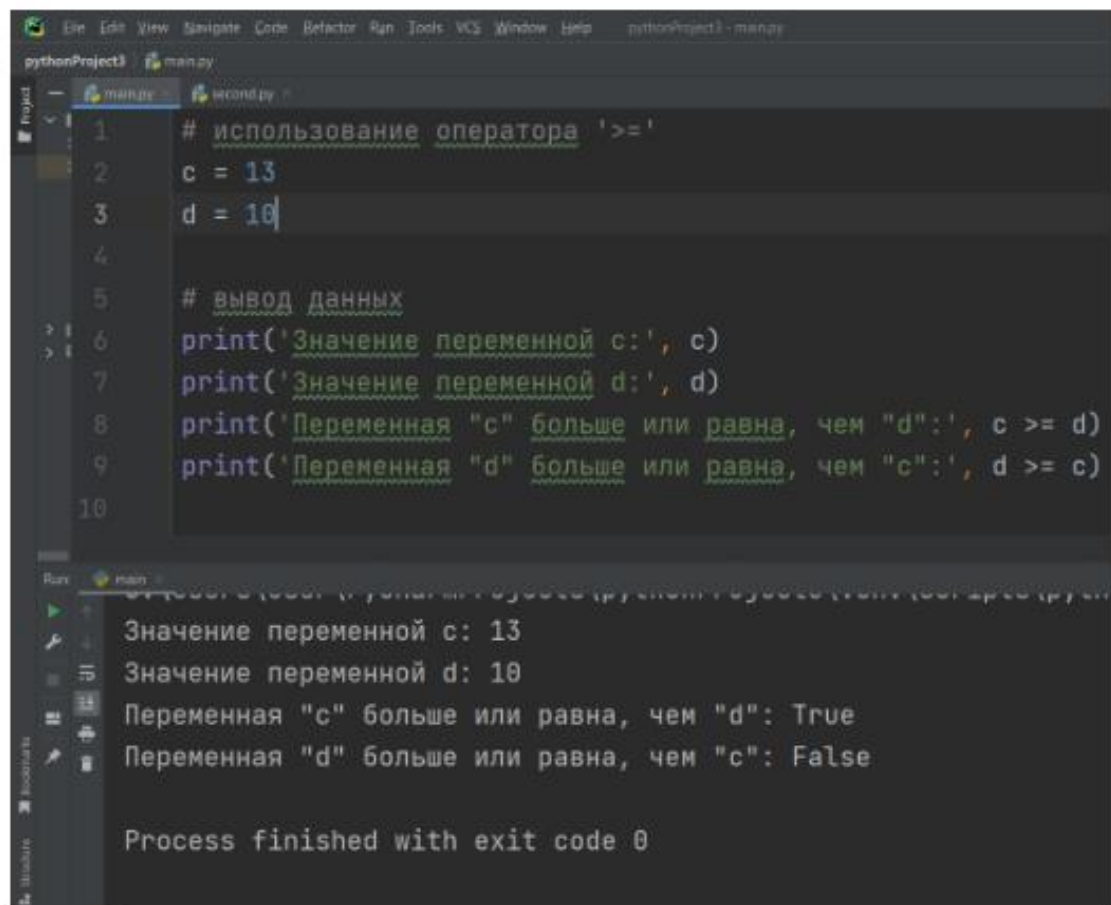
Run: main
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe C:\Users\user\PycharmProjects\pythonProject3\main.py
Список sequence: ['first', 'second', 'third', 'last']
Первый элемент списка: first
Последний элемент списка: last
```

Задача 11.

Создать программу, в которой используются переменные с именами ‘с’ и ‘d’. Необходимо использовать специальный оператор ‘>=’, относительно данных переменных. Результаты продемонстрировать на консоли.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



```
pythonProject3 - main.py
1 # использование оператора '>='
2 c = 13
3 d = 10
4
5 # вывод данных
6 print('Значение переменной c:', c)
7 print('Значение переменной d:', d)
8 print('Переменная "c" больше или равна, чем "d":', c >= d)
9 print('Переменная "d" больше или равна, чем "c":', d >= c)
10

Run: main
Значение переменной c: 13
Значение переменной d: 10
Переменная "c" больше или равна, чем "d": True
Переменная "d" больше или равна, чем "c": False

Process finished with exit code 0
```

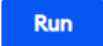

Задача 12.

Написать программу с двумя глобальными переменными (имена задаются произвольно),

которая вычисляет разность данных двух чисел.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:

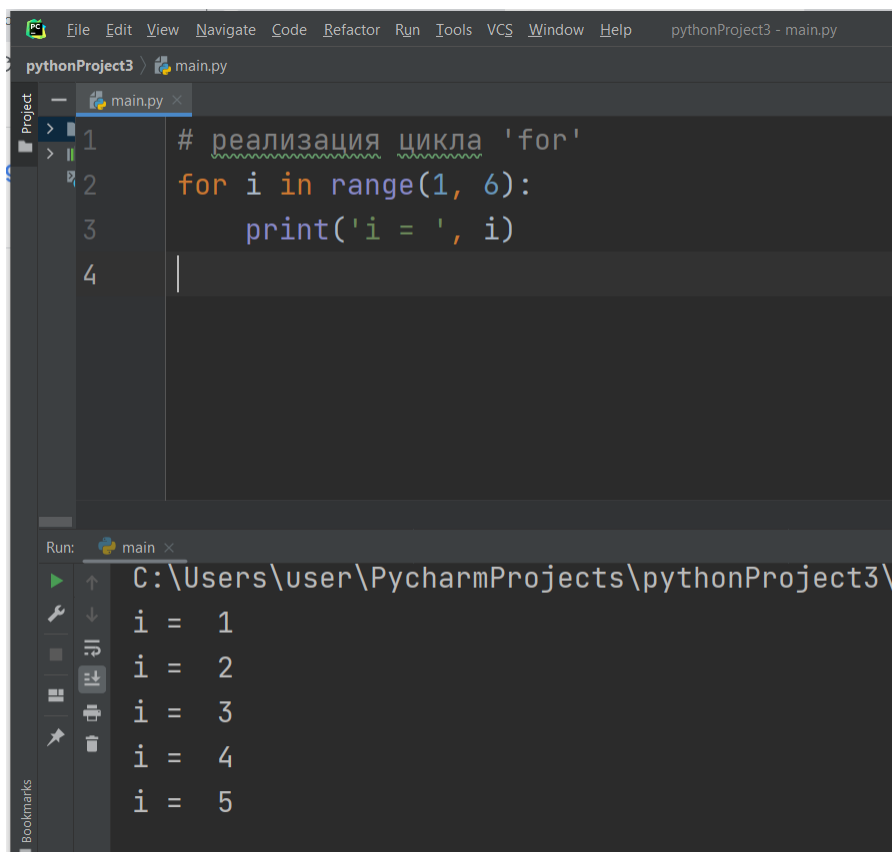
main.py		Shell	
<pre>1 global c 2 c = 1000 3 global x 4 x = 500 5 6 7 def difference(): 8 result = c - x 9 print("Разность:", result) 10 11 print(difference())</pre>		<pre>Разность: 500 None > </pre>	

Задача 13.

Создать программу с использованием цикла **for**. Данный цикл перебирает значения переменной 'i' от 1 до 5 и выводит на консоль.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



The screenshot shows the PyCharm IDE interface. The top toolbar includes buttons for File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The main editor window displays a Python script in 'main.py' with the following code:

```
1 # реализация цикла 'for'
2 for i in range(1, 6):
3     print('i = ', i)
4
```

Below the editor, the 'Run' console shows the output of the script:

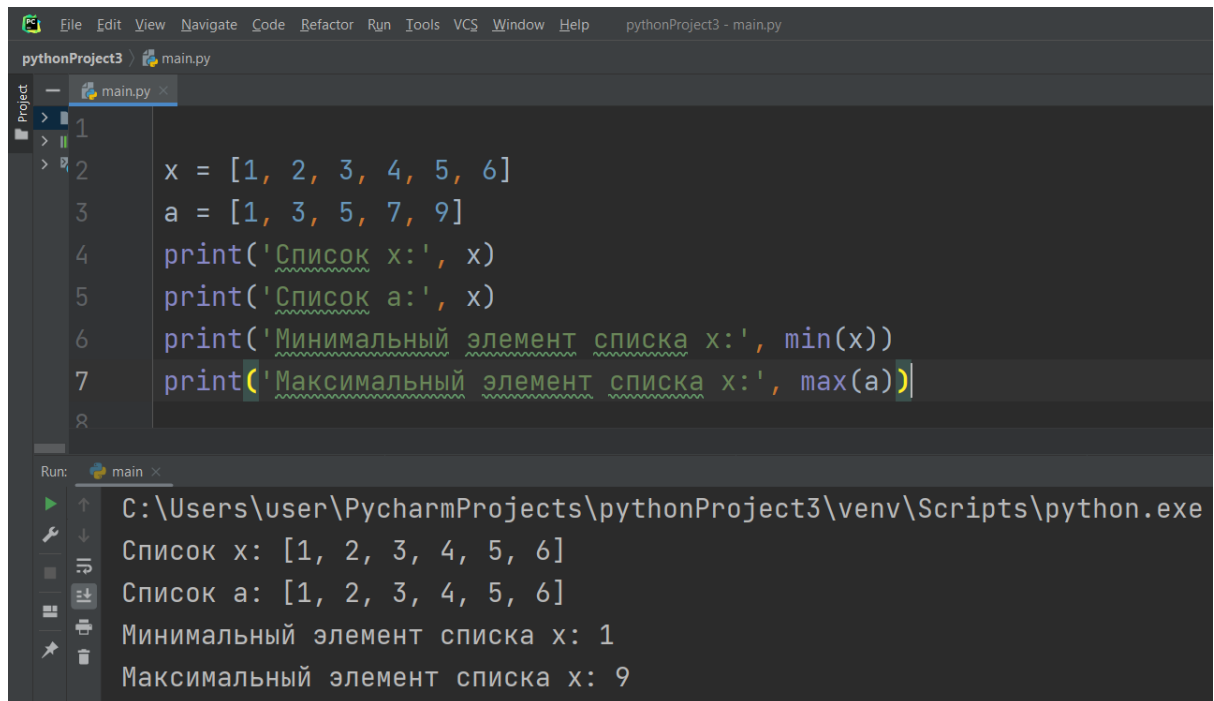
```
C:\Users\user\PycharmProjects\pythonProject3\
i = 1
i = 2
i = 3
i = 4
i = 5
```

Задача 14.

Создать программу, в которой вводятся списки 'x' = [1, 2, 3, 4, 5, 6] и 'a' = [1, 3, 5, 7, 9]. Необходимо найти максимальный элемент в списке 'a' и минимальный элемент в списке 'x'. Данные значения вывести на консоль.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



The screenshot shows the PyCharm IDE interface. The top part is the editor window with a file named 'main.py'. The code in the editor is as follows:

```
1 x = [1, 2, 3, 4, 5, 6]
2 a = [1, 3, 5, 7, 9]
3 print('Список x:', x)
4 print('Список a:', x)
5 print('Минимальный элемент списка x:', min(x))
6 print('Максимальный элемент списка x:', max(a))
```

The bottom part of the screenshot shows the 'Run' console. It displays the output of the program:

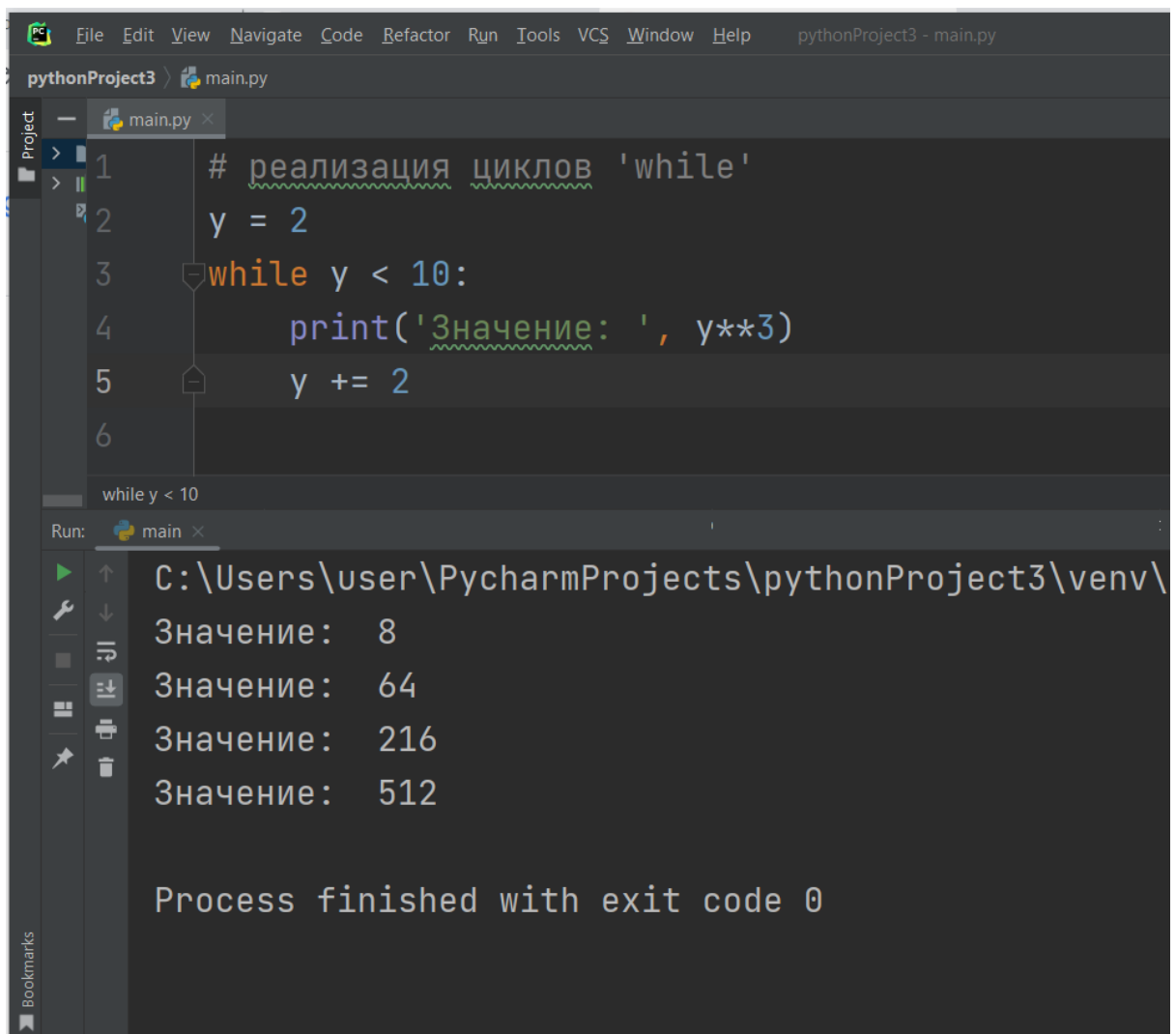
```
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe
Список x: [1, 2, 3, 4, 5, 6]
Список a: [1, 2, 3, 4, 5, 6]
Минимальный элемент списка x: 1
Максимальный элемент списка x: 9
```

Задача 15.

Создать программу с использованием цикла **while** и переменной 'y', значение которой, изначально, равно 2. Данный цикл выводит значения переменной 'y' в третьей степени до тех пор, пока значение переменной 'y' не станет равным 10. При каждой итерации значение переменной 'y' увеличивается на 2.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



Функции (повторение)

Пример 1

```
def rectangle():
    a = float(input("Ширина %s: " % figure)) # обращение к глобальной
    b = float(input("Высота %s: " % figure)) # переменной figure
    print("Площадь: %.2f" % (a*b))
def triangle():
    a = float(input("Основание %s: " % figure))
    h = float(input("Высота %s: " % figure))
    print("Площадь: %.2f" % (0.5 * a * h))
figure = input("1-прямоугольник, 2-треугольник: ")
if figure == '1':
    rectangle()
elif figure == '2':
    triangle()
```

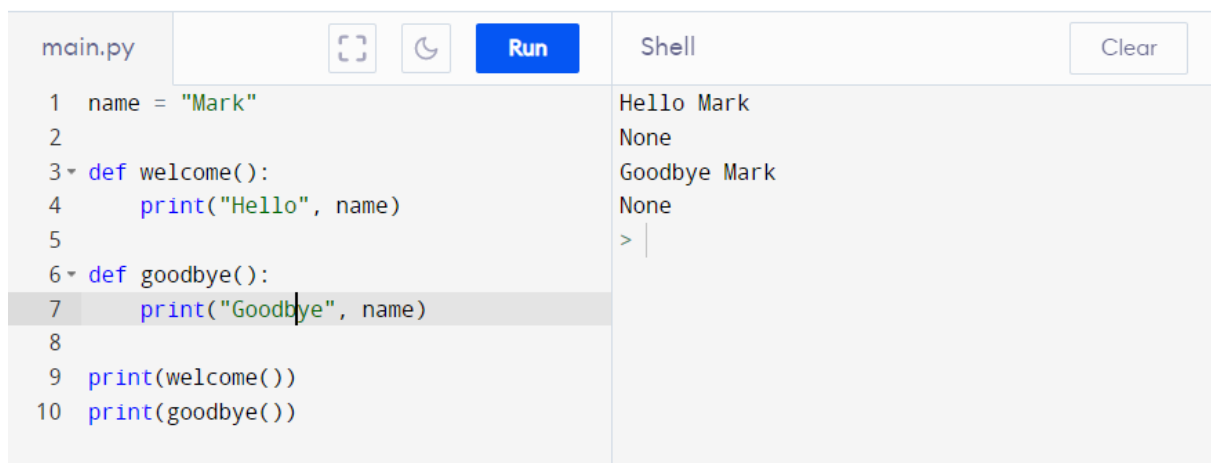
Пример 2

```
def duple(a, b):
    ploch = a * b
    perim = 2 * (a + b)
    return ploch, perim

width = float(input('Введи ширину: '))
height = float(input('Введи высоту: '))
g_ploch, g_perim = duple(width, height)
print('Площадь прямоугольника: ', g_ploch)
print('Периметр прямоугольника: ', g_perim)
```

Глобальный контекст

Глобальный контекст подразумевает собой, что некая переменная является глобальной, которая определена вне любой из функций и доступна любой функции в программе.



The screenshot shows a code editor with a file named 'main.py'. The code defines a global variable 'name' and two functions, 'welcome()' and 'goodbye()', both of which use 'name'. The 'welcome()' function prints 'Hello' and 'goodbye()' prints 'Goodbye'. The main execution calls both functions. The output in the 'Shell' pane shows 'Hello Mark', 'None', 'Goodbye Mark', and 'None'.

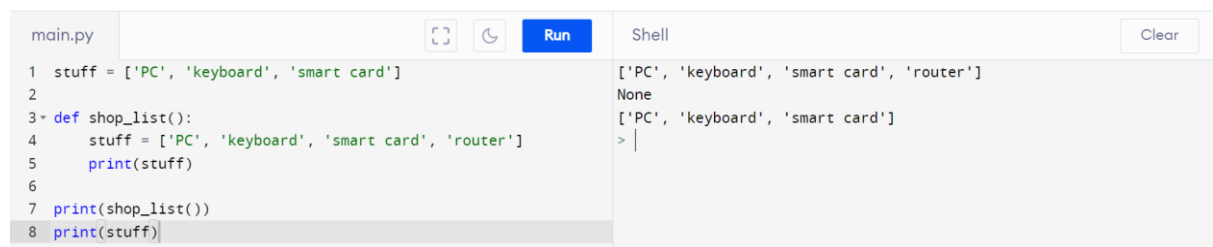
```
main.py
1 name = "Mark"
2
3 def welcome():
4     print("Hello", name)
5
6 def goodbye():
7     print("Goodbye", name)
8
9 print(welcome())
10 print(goodbye())
```

```
Shell
Hello Mark
None
Goodbye Mark
None
> |
```

Рисунок 18 - Листинг кода с использованием глобального контекста

В данной ситуации, переменная **name** является глобальной и имеет глобальную область видимости. Кроме того, определенные здесь функции могут свободно ее использовать. Если вместо первоначального определения переменной внутри функции переместить ее на внешний уровень и инициализировать ее, то тогда можно ссылаться на нее вне функции.

Рассмотрим другой пример - имеется некий список товаров **stuff = [PC, keyboard, smart card, router]**. Но, если попытаться переопределить переменную **stuff** внутри функции **shop_list()**, данные изменения не будут обновлены до исходной глобальной переменной, а будут изолированы локально (рис. 19):



```
main.py
1 stuff = ['PC', 'keyboard', 'smart card']
2
3- def shop_list():
4     stuff = ['PC', 'keyboard', 'smart card', 'router']
5     print(stuff)
6
7 print(shop_list())
8 print(stuff)
```

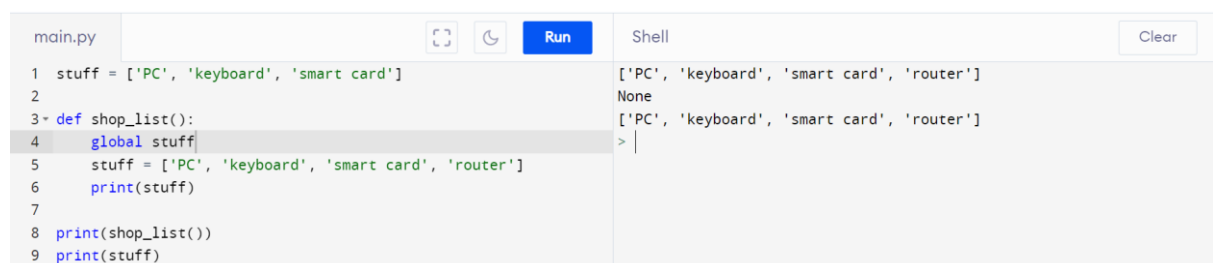
```
Shell
['PC', 'keyboard', 'smart card', 'router']
None
['PC', 'keyboard', 'smart card']
> |
```

Рисунок 19 - Листинг кода с использованием списка товаров *stuff*

Как можно заметить, в первый раз (1 строка вывода) выводится переменная **stuff**, определенная внутри функции **shop_list()**, а затем измененная переменная **stuff**, которая была инициализирована вне функции **shop_list()** (в этом случае она является глобальной). Это происходит из-за того, что данная переменная была изменена внутри функции **shop_list()**, путем создания локальной переменной с тем же именем.

Ключевое слово **global**

В том случае, если разработчик-программист хочет, чтобы данные изменения отражались в глобальной переменной, вместо того, чтобы создавать новую локальную, все, что ему нужно - добавить ключевое слово **global**. Данное ключевое слово позволит сообщить о том, что переменная **stuff** действительно является глобальной (рис. 20):



```
main.py
1 stuff = ['PC', 'keyboard', 'smart card']
2
3- def shop_list():
4     global stuff
5     stuff = ['PC', 'keyboard', 'smart card', 'router']
6     print(stuff)
7
8 print(shop_list())
9 print(stuff)
```

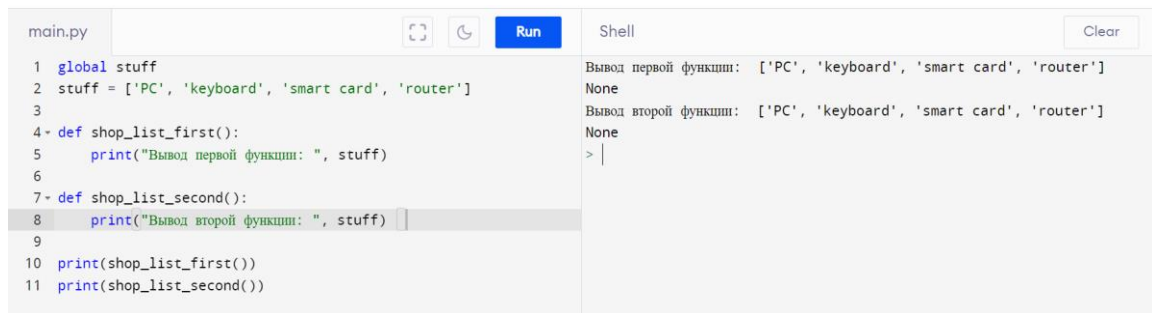
```
Shell
['PC', 'keyboard', 'smart card', 'router']
None
['PC', 'keyboard', 'smart card', 'router']
> |
```

Рисунок 20 - Листинг кода с использованием ключевого слова *global*

В данном случае, глобальная переменная модифицируется новыми значениями, в связи с этим, при вызове **print(stuff)**, новое значение (здесь это слово **router**) будет выведено.

Определив контекст переменной **stuff**, которая истинно является глобальной, можно переопределить и изменить ее по своему усмотрению, зная, что абсолютно все изменения,

которые будут внесены в функцию, будут перенесены. Кроме этого, в объектно-ориентированном языке программирования *Python* можно определять глобальную переменную в самой функции и иметь возможность ссылаться на нее и получать к ней доступ в любом другом участке кода (рис. 21):



```
main.py
1 global stuff
2 stuff = ['PC', 'keyboard', 'smart card', 'router']
3
4 def shop_list_first():
5     print("Вывод первой функции: ", stuff)
6
7 def shop_list_second():
8     print("Вывод второй функции: ", stuff)
9
10 print(shop_list_first())
11 print(shop_list_second())
```

Shell

```
Вывод первой функции: ['PC', 'keyboard', 'smart card', 'router']
None
Вывод второй функции: ['PC', 'keyboard', 'smart card', 'router']
None
> |
```

Рисунок 21 - Листинг кода с использованием разных функций и ключевого слова *global*

Как можно заметить, переменная **stuff** является глобальной и определена вне какой-либо функции, хотя, при выводе (в строке 10 файла *main.py* используется функция **shop_list_first()**, а в строке 11 - **shop_list_second()**) данная переменная отображается корректно в обоих случаях.

С другой стороны, возможность локально изменять глобальную переменную - небольшой удобный инструмент, к нему нужно относиться с некоторой осторожностью. Чрезмерное переписывание и повторное определение области видимости - рецепт катастрофы, которая заканчивается ошибками и неожиданным поведением. Помимо этого, разработчику-программисту нужно быть уверенным в том, что он манипулирует переменной только в том контексте, который нужен в данной ситуации.



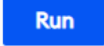

Практические задачи

Задача 1.

Написать программу с одной глобальной переменной **x**, которая считает произведение трех целых чисел.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:





main.py	  	Shell	
<pre>1 global c 2 c = 15 3 4 5 def sum(): 6 a = 12 7 b = 8 8 result = a + b + c 9 print("Сумма :", result) 10 11 print(sum())</pre>		<pre>Сумма : 35 None > </pre>	

Задача 2.

Написать программу с двумя глобальными переменными (имена задаются произвольно), которая вычисляет разность данных двух чисел.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:

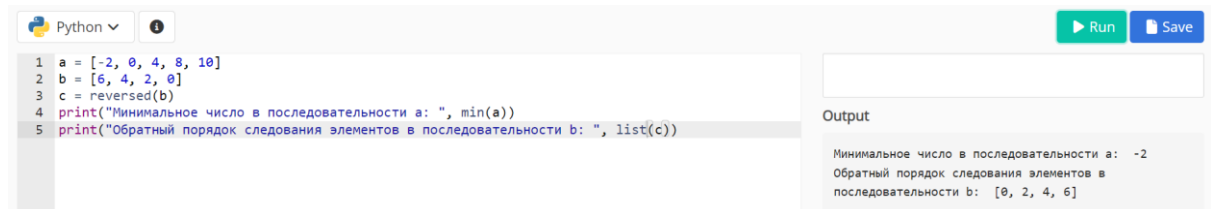
main.py	  	Shell	
<pre>1 global c 2 c = 1000 3 global x 4 x = 500 5 6 7 def difference(): 8 result = c - x 9 print("Разность:", result) 10 11 print(difference())</pre>		<pre>Разность: 500 None > </pre>	

Задача 3.

Используя встроенные функции `min()` и `reversed()`, найти минимальное значение в последовательности `a = [-2, 0, 4, 8, 10]` и развернуть порядок элементов в последовательности `b = [6, 4, 2, 0]`.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



```
1 a = [-2, 0, 4, 8, 10]
2 b = [6, 4, 2, 0]
3 c = reversed(b)
4 print("Минимальное число в последовательности a: ", min(a))
5 print("Обратный порядок следования элементов в последовательности b: ", list(c))
```

Output

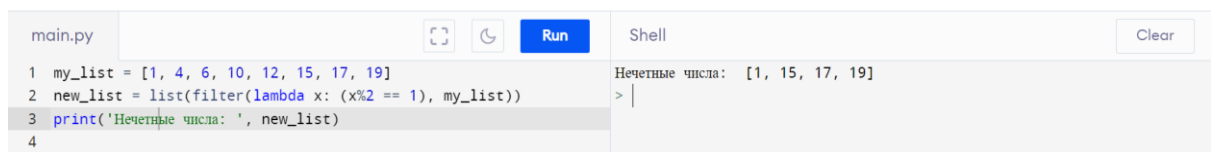
Минимальное число в последовательности a: -2
Обратный порядок следования элементов в последовательности b: [0, 2, 4, 6]

Задача 4.

Дан следующий список чисел: `my_list = [1, 4, 6, 10, 12, 15, 17, 19]`. Используя функцию `filter()`, отобрать из данного списка только нечетные числа.

Решение.

Напишем код для решения данной задачи и посмотрим на вывод:



```
main.py  Run  Shell  Clear
1 my_list = [1, 4, 6, 10, 12, 15, 17, 19]
2 new_list = list(filter(lambda x: (x%2 == 1), my_list))
3 print('Нечетные числа: ', new_list)
4
```

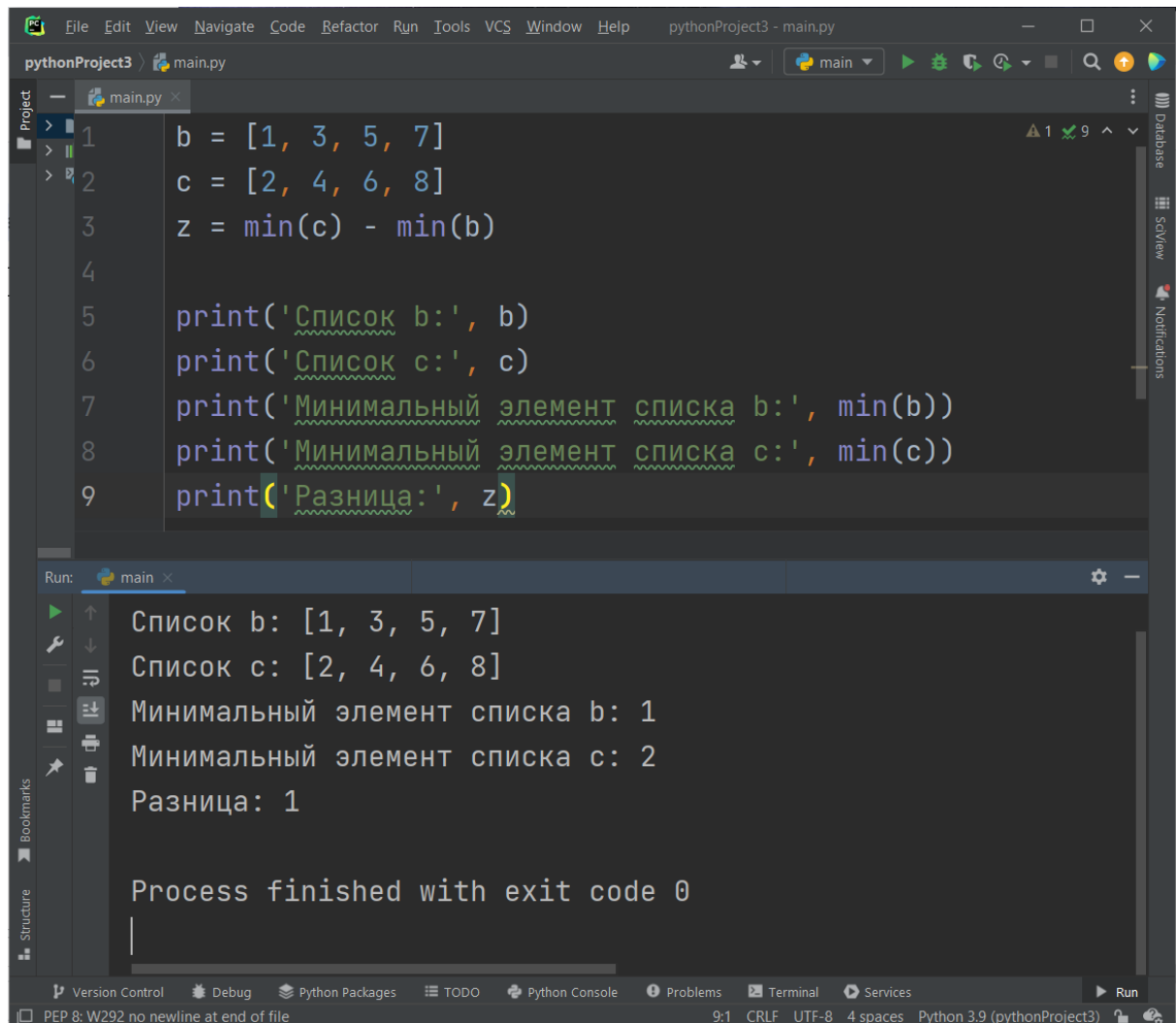
Нечетные числа: [1, 15, 17, 19]
> |

Задача 5.

Создать программу, в которой вводятся списки **'b'** = [1, 3, 5, 7] и **'c'** = [2, 4, 6, 8]. Необходимо найти минимальные элементы в списках **'b'** и **'c'**. Вывести списки и минимальные элементы на консоль. Затем найти произведение данных элементов и вывести его на консоль.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



The screenshot shows an IDE window titled 'pythonProject3 - main.py'. The code in 'main.py' is as follows:

```
1 b = [1, 3, 5, 7]
2 c = [2, 4, 6, 8]
3 z = min(c) - min(b)
4
5 print('Список b:', b)
6 print('Список c:', c)
7 print('Минимальный элемент списка b:', min(b))
8 print('Минимальный элемент списка c:', min(c))
9 print('Разница:', z)
```

Below the code editor is a 'Run' console window showing the output of the program:

```
Список b: [1, 3, 5, 7]
Список c: [2, 4, 6, 8]
Минимальный элемент списка b: 1
Минимальный элемент списка c: 2
Разница: 1

Process finished with exit code 0
```

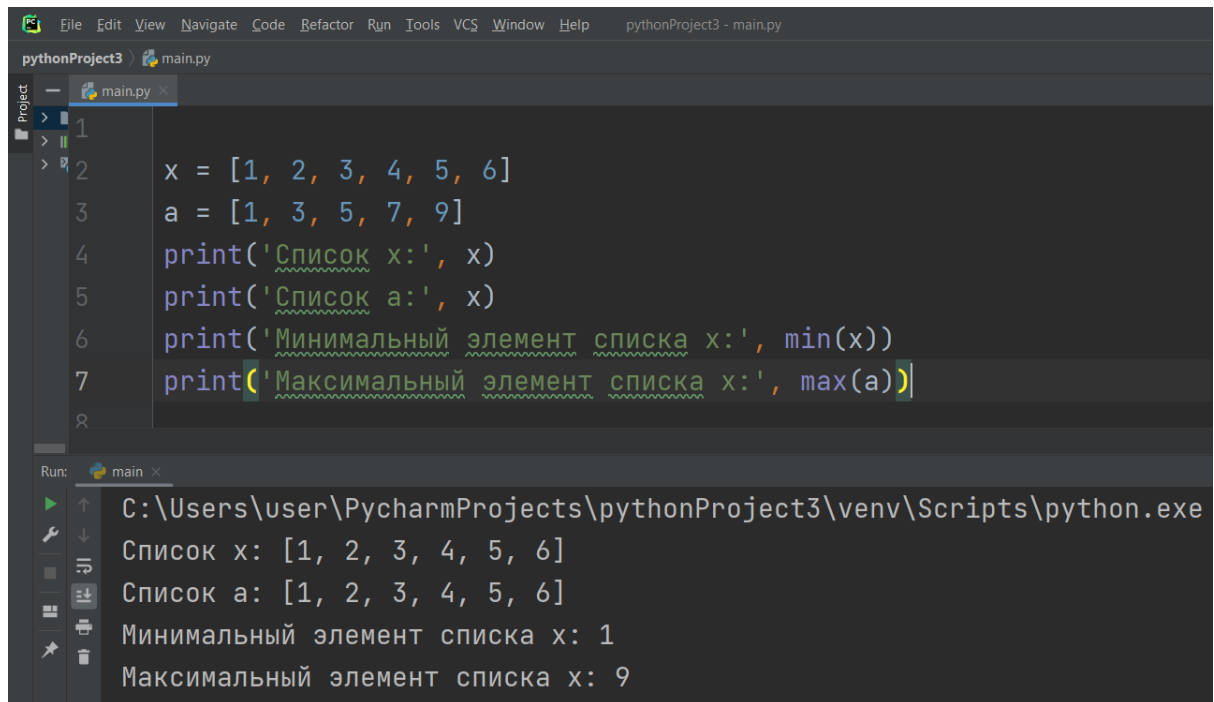
The IDE interface includes a sidebar with 'Project', 'Structure', and 'Bookmarks' views, and a bottom status bar showing 'PEP 8: W292 no newline at end of file', '9:1 CRLF UTF-8 4 spaces Python 3.9 (pythonProject3)', and a 'Run' button.

Задача 6.

Создать программу, в которой вводятся списки 'x' = [1, 2, 3, 4, 5, 6] и 'a' = [1, 3, 5, 7, 9]. Необходимо найти максимальный элемент в списке 'a' и минимальный элемент в списке 'x'. Данные значения вывести на консоль.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The main editor window displays a Python file named 'main.py' with the following code:

```
1
2 x = [1, 2, 3, 4, 5, 6]
3 a = [1, 3, 5, 7, 9]
4 print('Список x:', x)
5 print('Список a:', x)
6 print('Минимальный элемент списка x:', min(x))
7 print('Максимальный элемент списка x:', max(a))
```

Below the editor, the 'Run' console shows the output of the program:

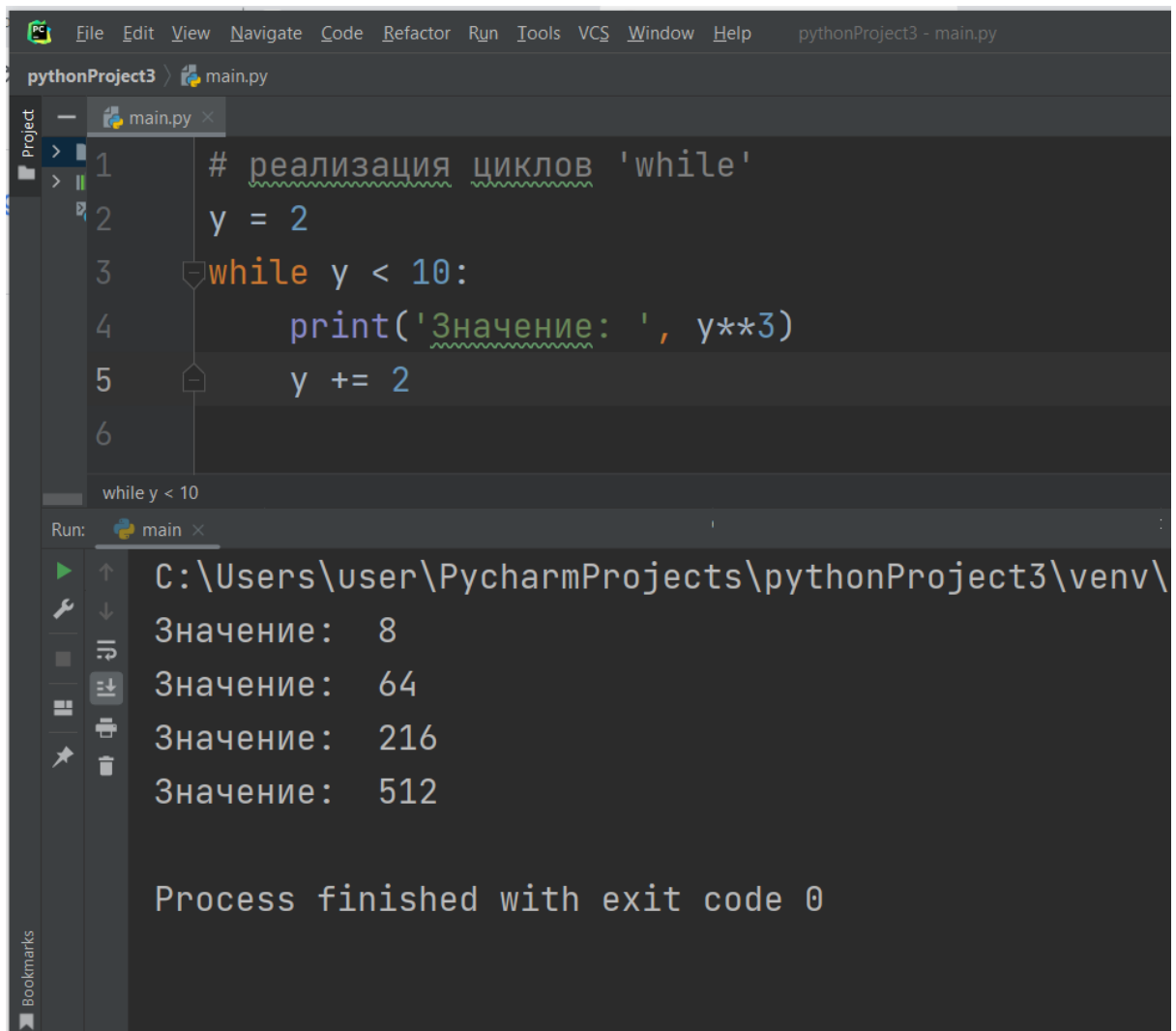
```
Run: main x
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe
Список x: [1, 2, 3, 4, 5, 6]
Список a: [1, 2, 3, 4, 5, 6]
Минимальный элемент списка x: 1
Максимальный элемент списка x: 9
```

Задача 7.

Создать программу с использованием цикла **while** и переменной 'y', значение которой, изначально, равно 2. Данный цикл выводит значения переменной 'y' в третьей степени до тех пор, пока значение переменной 'y' не станет равным 10. При каждой итерации значение переменной 'y' увеличивается на 2.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The main editor window displays a Python file named 'main.py' with the following code:

```
1 # реализация циклов 'while'
2 y = 2
3 while y < 10:
4     print('Значение: ', y**3)
5     y += 2
6
```

Below the editor, the 'Run' console shows the output of the program:

```
C:\Users\user\PycharmProjects\pythonProject3\venv\
Значение: 8
Значение: 64
Значение: 216
Значение: 512

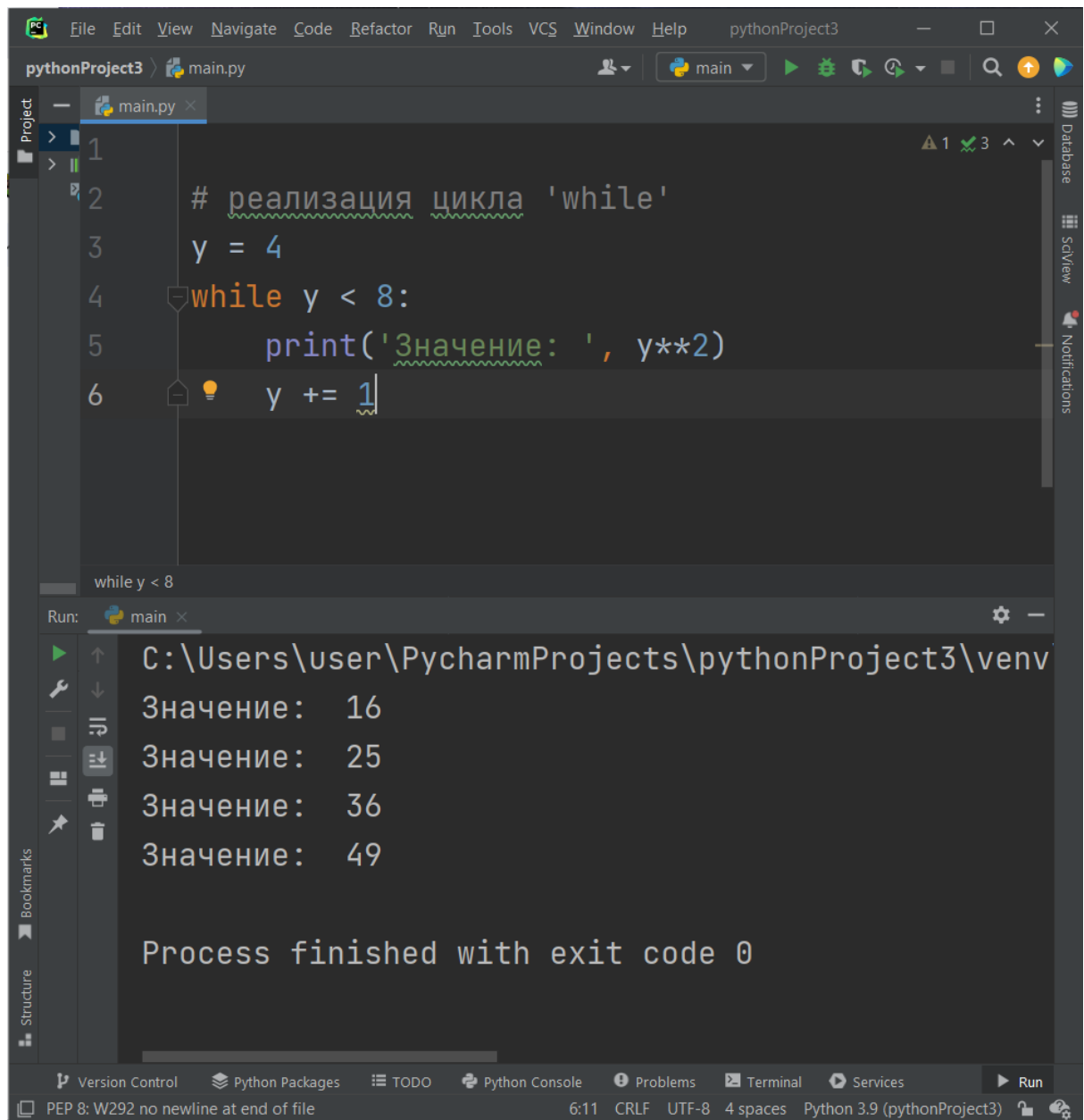
Process finished with exit code 0
```

Задача 8.

Создать программу с использованием цикла **while** и переменной 'y', значение которой, изначально, равно 4. Данный цикл выводит значения переменной 'y' во второй степени до тех пор, пока значение переменной 'y' не станет равным 8. При каждой итерации значение переменной 'y' увеличивается на единицу.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



The screenshot shows the PyCharm IDE interface. The main editor window displays a Python script in `main.py` for a project named `pythonProject3`. The script implements a `while` loop that prints the square of `y` until `y` reaches 8. The code is as follows:

```
1  
2 # реализация цикла 'while'  
3 y = 4  
4 while y < 8:  
5     print('Значение: ', y**2)  
6     y += 1
```

Below the editor, the Run console shows the output of the program. The loop executes four times, printing the squares of 4, 5, 6, and 7, as the condition `y < 8` becomes false when `y` reaches 8. The output is:

```
C:\Users\user\PycharmProjects\pythonProject3\venv  
Значение:  16  
Значение:  25  
Значение:  36  
Значение:  49  
  
Process finished with exit code 0
```

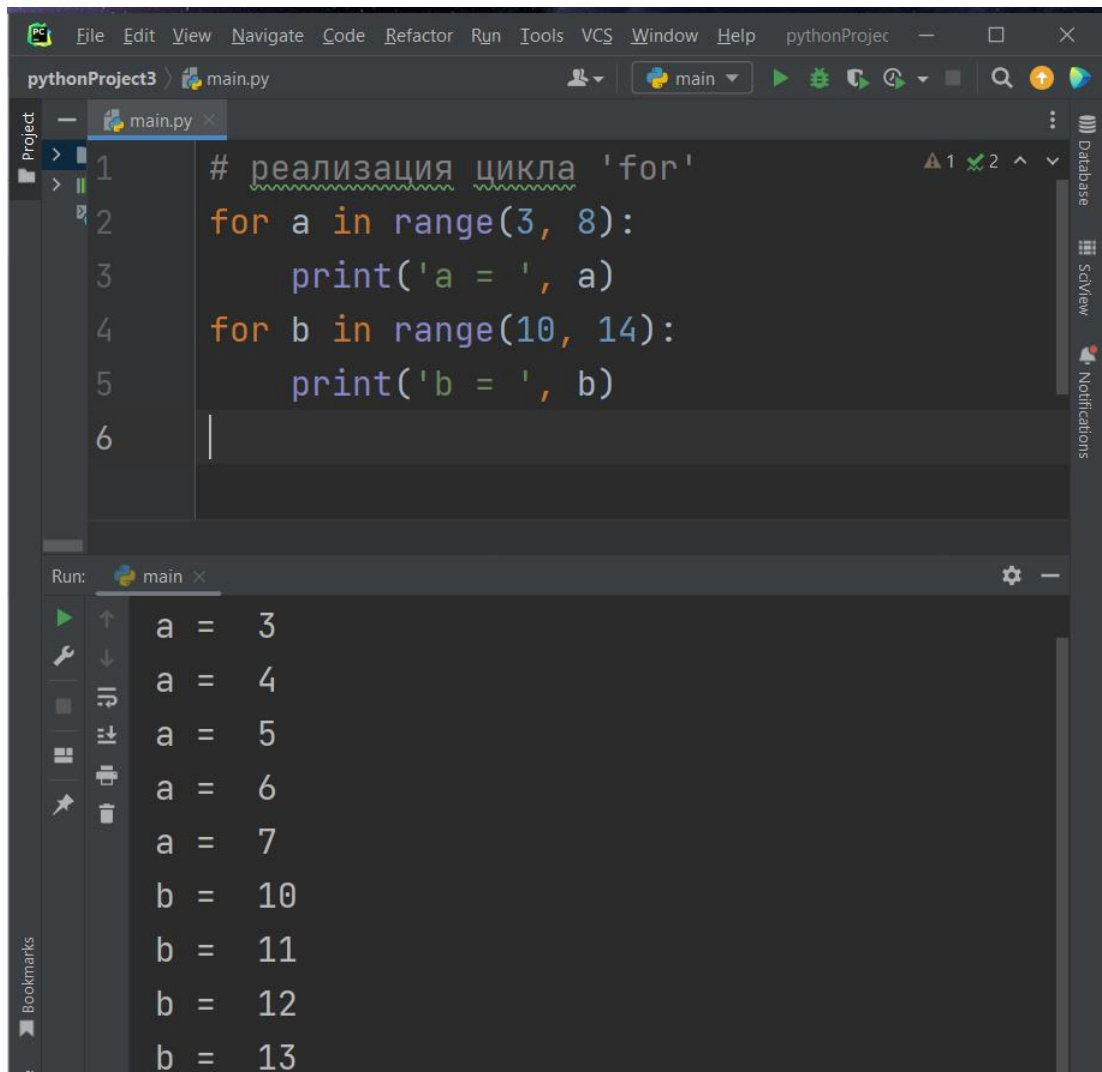
The status bar at the bottom indicates the file encoding is UTF-8, the line ending is CRLF, and the indentation is 4 spaces. The Python version is 3.9.

Задача 9.

Создать программу с использованием цикла **for**. Данный цикл перебирает значения переменной 'a' от 3 до 7 и переменной 'b' от 10 до 13 и выводит соответствующие значения на консоль.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



The screenshot shows an IDE window titled 'pythonProject3' with a file named 'main.py'. The code in the editor is as follows:

```
1  # реализация цикла 'for'
2  for a in range(3, 8):
3      print('a = ', a)
4  for b in range(10, 14):
5      print('b = ', b)
6
```

Below the editor, the 'Run' console shows the output of the program:

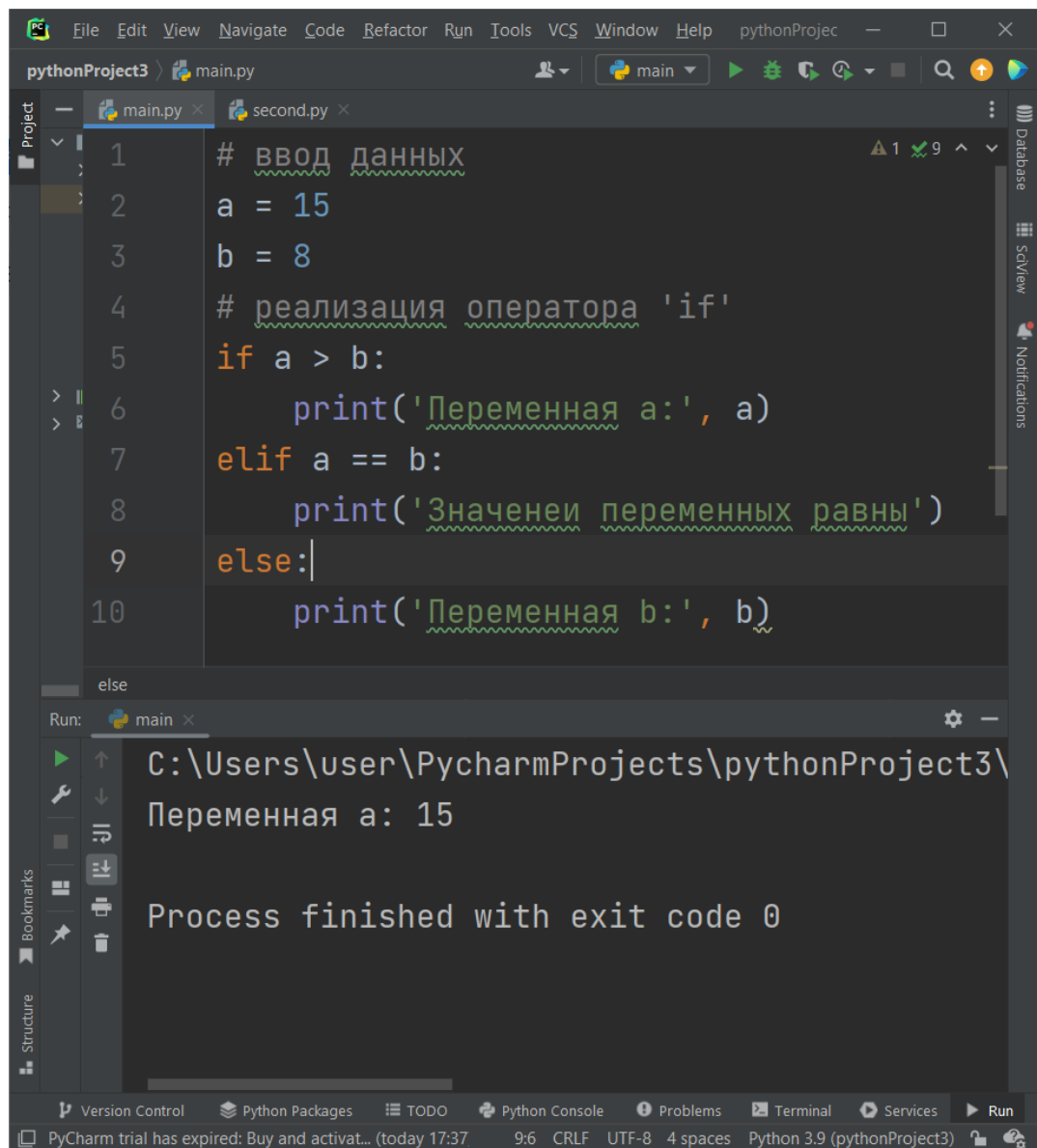
```
Run: main
a = 3
a = 4
a = 5
a = 6
a = 7
b = 10
b = 11
b = 12
b = 13
```

Задача 10.

Создать программу, в которой используются переменные 'a' и 'b'. Реализовать в данной программе оператор 'if' со следующим условием: если значение переменной 'a' больше, чем значение переменной 'b', то выводится значение переменной 'b'; если значения переменных равны, то выводится сообщение 'Значения переменных равны'; в противном случае, выводится значение переменной 'b'. Реализовать все три итерации (в каждой итерации выполняется свое условие). Результаты продемонстрировать на консоли.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



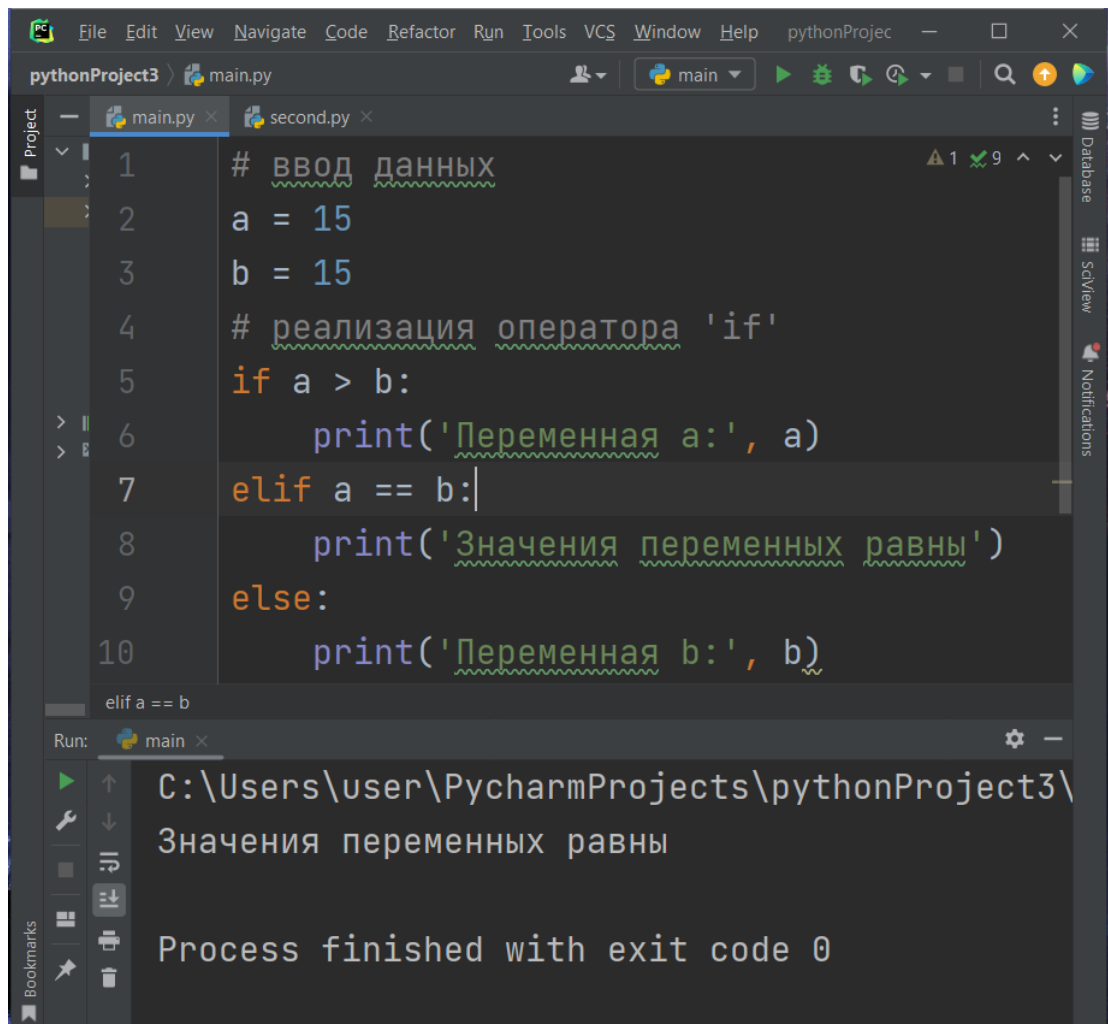
The screenshot shows the PyCharm IDE interface. The main editor window displays a Python script in a file named `main.py`. The script is as follows:

```
1 # ввод данных
2 a = 15
3 b = 8
4 # реализация оператора 'if'
5 if a > b:
6     print('Переменная a:', a)
7 elif a == b:
8     print('Значения переменных равны')
9 else:
10    print('Переменная b:', b)
```

Below the editor, the Run window shows the output of the program. The output is:

```
C:\Users\user\PycharmProjects\pythonProject3\
Переменная a: 15
Process finished with exit code 0
```

The status bar at the bottom of the IDE indicates the current configuration: Python 3.9 (pythonProject3).

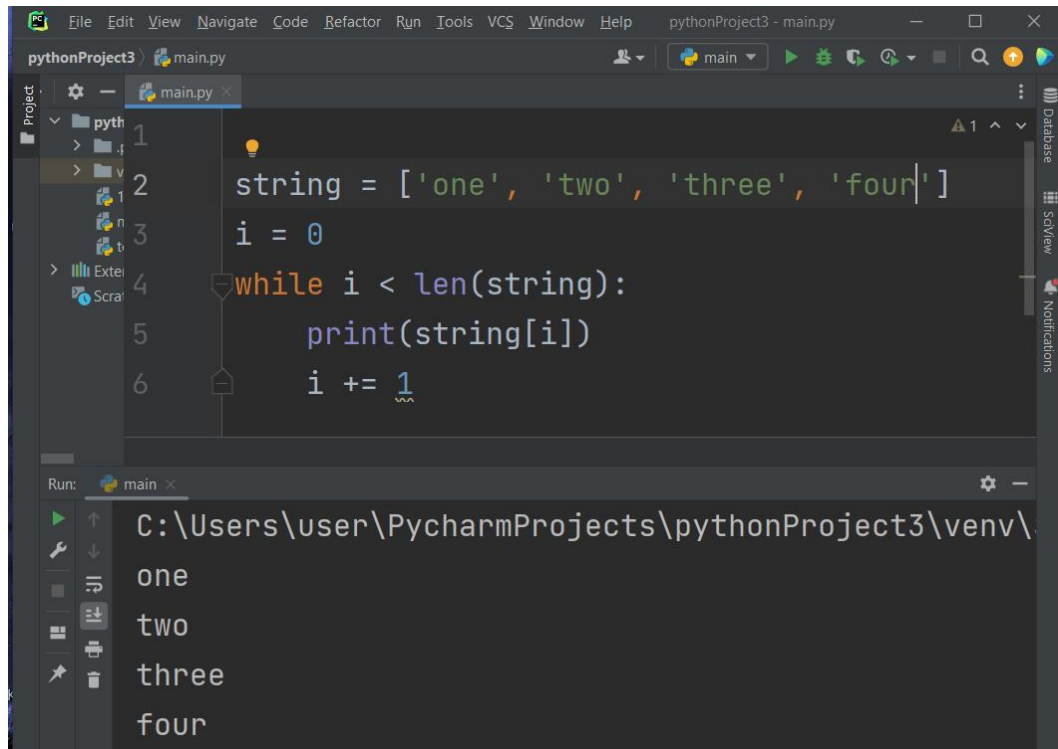


Задача 11.

Создать программу, в которой происходит перебор элементов списка с именем **'words'** при помощи цикла **while**.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



The screenshot shows the PyCharm IDE interface. The main editor window displays a Python script in `main.py` for a project named `pythonProject3`. The script defines a list `string` with elements `'one', 'two', 'three', 'four'` and uses a `while` loop to iterate through the list, printing each element. The `Run` button is visible in the top toolbar. Below the editor, the `Run` console shows the output of the script: `one`, `two`, `three`, and `four` on separate lines.

```
pythonProject3 - main.py
main.py
1 string = ['one', 'two', 'three', 'four']
2
3 i = 0
4 while i < len(string):
5     print(string[i])
6     i += 1

Run: main
C:\Users\user\PycharmProjects\pythonProject3\venv\
one
two
three
four
```

Задача 12.

Написать программу с двумя глобальными переменными (имена задаются произвольно), которая вычисляет разность данных двух чисел.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:

main.py	Shell
<pre>1 global c 2 c = 1000 3 global x 4 x = 500 5 6 7 def difference(): 8 result = c - x 9 print("Разность:", result) 10 11 print(difference())</pre>	<pre>Разность: 500 None > </pre>

Задача 13.

Используя встроенные функции `max()` и `sum()`, найти максимальное значение в последовательности `x = [1, 3, 5, 10, 11]` и посчитать сумму элементов в последовательности `y = [10, 55, 12, 100]`.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



```
Python
1 x = [1, 3, 5, 10, 11]
2 y = [10, 55, 12, 100]
3 print("Максимальное число в последовательности x: ", max(x))
4 print("Сумма элементов в последовательности y: ", sum(y))
```

Output

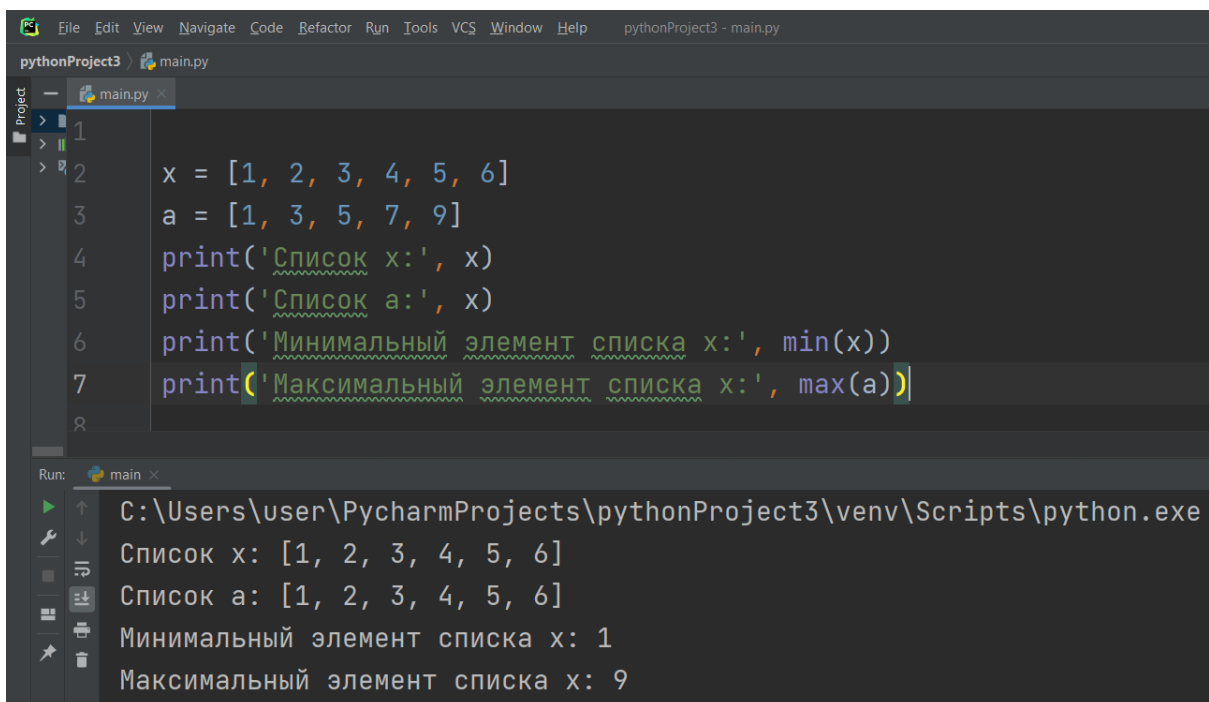
```
Максимальное число в последовательности x: 11
Сумма элементов в последовательности y: 177
```

Задача 14.

Создать программу, в которой вводятся списки `'x' = [1, 2, 3, 4, 5, 6]` и `'a' = [1, 3, 5, 7, 9]`. Необходимо найти максимальный элемент в списке `'a'` и минимальный элемент в списке `'x'`. Данные значения вывести на консоль.

Решение.

Напишем код для решения данной практической задачи и посмотрим на вывод:



```
pythonProject3 > main.py
1
2 x = [1, 2, 3, 4, 5, 6]
3 a = [1, 3, 5, 7, 9]
4 print('Список x:', x)
5 print('Список a:', a)
6 print('Минимальный элемент списка x:', min(x))
7 print('Максимальный элемент списка a:', max(a))
```

Run: main

```
C:\Users\user\PycharmProjects\pythonProject3\venv\Scripts\python.exe
Список x: [1, 2, 3, 4, 5, 6]
Список a: [1, 3, 5, 7, 9]
Минимальный элемент списка x: 1
Максимальный элемент списка a: 9
```

Словари (повторение)

Словари (тип dict) представляют собой изменяемый упорядоченный набор элементов с доступом к ним по ключу. Данные в словаре хранятся в формате "ключ:значение".

Правила организации словаря:

1. В словаре не может быть двух элементов с одинаковыми ключами. Однако могут быть одинаковые значения у разных ключей.
2. Ключом может быть любой неизменяемый тип данных. Значением – любой тип данных.

Создание пустого словаря

Раздаточный материал № 86

```
>>> d1 = dict()
>>> print(type(d1))
<class 'dict'>
```

```
>>> d2 = {}
>>> print(type(d2))
<class 'dict'>
```

Создание словаря с заранее подготовленным набором данных

Раздаточный материал № 87

```
>>> d1 = dict(Ivan="менеджер", Mark="инженер")
>>> print(d1)
{'Mark': 'инженер', 'Ivan': 'менеджер'}
```

```
>>> d2 = {"A1": "123", "A2": "456"}
>>> print(d2)
{'A2': '456', 'A1': '123'}
```

```
>>> a = {'cat': 'кошка', 'dog': 'собака', 'bird': 'птица', 'mouse': 'мышь'}
```

Создание словаря через вложенный список

```
a = [['cat', 'кошка'], ['dog', 'собака'], ['bird', 'птица'], ['mouse', 'мышь']]
s = dict(a)
print(s)
{'cat': 'кошка', 'dog': 'собака', 'bird': 'птица', 'mouse': 'мышь'}
```

В словаре доступ к значениям осуществляется по ключам, которые заключаются в квадратные скобки

Раздаточный материал № 88

```
>>> a['cat']
'кошка'
>>> a['bird']
'птица'
```

Удаление и добавление элемента (пары "ключ:значение")

Раздаточный материал № 89

```
>>> a['elephant'] = 'бегемот'      # добавляем
>>> a['table'] = 'стол'           # добавляем
>>> a
{'dog': 'собака', 'cat': 'кошка', 'mouse': 'мышь', 'bird': 'птица', 'table': 'стол', 'elephant': 'бегемот'}
```

```
>>> a['elephant'] = 'слон'          # изменяем
>>> del a['table']                  # удаляем
>>> a
{'dog': 'собака', 'cat': 'кошка', 'mouse': 'мышь', 'bird': 'птица', 'elephant': 'слон'}
```

Элементы словаря перебираются в цикле for также, как элементы других сложных объектов.

```
Раздаточный материал № 91
nums = {1: 'one', 2: 'two', 3: 'three'}
```

```
# извлекаются ключи
for i in nums:
    print(i)
```

Результат

```
1
2
3
```

```
# извлекаются значения:
for i in nums:
    print(nums[i])
```

Результат

```
one
two
three
```

Лямбда функции

Лямбда функции в Python – это такие функции, которые не имеют названия. Их также называют анонимными. Слово «lambda» является служебным, и не отражает сути конкретной функции. Не требуют return записываются в одной строке. Используются в коде единожды, может входить в состав других языковых конструкций.

В отличие от обычных функций, анонимные не могут содержать return, pass, assert и raise.

Создание лямбда функций происходит с помощью ключевого слова lambda следующим образом:

```
lambda <аргумент(ы)>: <выражение>
```

Лямбда функции могут иметь сколько угодно аргументов или не иметь их вовсе, но обязательно должны содержать лишь одно выражение.

Лямбда функции лучше использовать в связке с обычными функциями, например, для работы с итерируемыми объектами (map(), reduce(), zip(), filter()).

map() — это встроенная функция Python, принимающая в качестве аргумента функцию и последовательность. Она работает так, что применяет переданную функцию к каждому элементу.

Предположим, есть список целых чисел, которые нужно возвести в квадрат с помощью map.

```
# список целых чисел, которые нужно возвести в квадрат
L = [1, 2, 3, 4]
print(list(map(lambda x: x**2, L)))
```

filter() — отфильтровывает некоторые элементы итерируемого объекта (например, списка) на основе какого-то критерия. Критерий определяется за счет передачи функции в качестве аргумента. Она же применяется к каждому элементу объекта.

```
print(list(filter(lambda x: x % 2 == 0, [1, 3, 2, 5, 20, 21])))
```

reduce() принимает два параметра — функцию и список. Сначала она применяет стоящую первым аргументом функцию для двух начальных элементов списка, а затем использует в качестве аргументов этой функции полученное значение вместе со следующим элементом списка и так до тех пор, пока весь список не будет пройден, а итоговое значение не будет возвращено. Для того, чтобы использовать `reduce()`, ее необходимо сначала импортировать ее из модуля `functools`.

```
from functools import reduce
print(reduce(lambda x,y: y-x, L)) # работа reduce
# 3 - 1 = 2
# 2 - 2 = 0
# 5 - 0 = 5
# 20 - 5 = 15
# 21 - 15 = 6
```

Лямбда-функции можно включать в тело обычных функций. Эта функция будет удваивать все получаемые значения:

```
def my_function(n):
    return lambda a : a * n

double = my_function(2)
print(double(15))
```

Вывод: 30

В анонимных функциях можно использовать условия и операторы сравнения. Здесь анонимная функция добавляет слово гласная или согласная в список в зависимости от того, является ли буква в строке `st` гласной или согласной:

```
st = 'яжертыуиопшщасдфгчйклзхцвбнм'
vowels = 'аиеёоуыэюя'
result = list(map(lambda x: 'гласная' if x in vowels else 'согласная', st))
print(result)
```

Вывод:

```
['гласная', 'согласная', 'гласная', 'согласная', 'согласная', 'гласная', 'гласная', 'гласная',
'гласная', 'согласная', 'согласная', 'согласная', 'гласная', 'согласная', 'согласная', 'согласная',
'согласная', 'согласная', 'согласная', 'согласная', 'согласная', 'согласная', 'согласная', 'согласная',
'согласная', 'согласная', 'согласная', 'согласная', 'согласная']
```

Здесь анонимная функция выбирает из исходного списка четные числа, превосходящие 6:

```
numbers = [7, 8, 6, 9, 4, -6, 2, 0, -3, -12, -5, -2, 12, 77, 32]
print(list(filter(lambda x: x > 6 and x % 2 == 0, numbers)))
```

В этом примере внутренняя лямбда-функция принимает список и возвращает минимальный или максимальный элемент в зависимости от соответствующего параметра внешней функции:

```
def get_min_or_max(value='max'):
    return eval(f'lambda x: {value}(x)')

lst = [3, 5, -22, -15, 100, 7, 8, 9, 12, 98]

max_func = get_min_or_max()
min_func = get_min_or_max('min')
print(max_func(lst))
print(min_func(lst))
```

Вывод:

100

-22

Лямбда-функции для проведения последовательных операций можно хранить в списках:

```
ops = [(lambda x, y: x + y), (lambda x, y: x - y),
       (lambda x, y: x * y), (lambda x, y: x / y)]

x, y = int(input("Введи x: ")), int(input("Введи y: "))
for i in range(len(ops)):
    print(ops[i](x, y))
```

Пример вывода для x = 12, y = 15:

27

-3

180

0.8

А также в словарях:

```
ops = {'neg': lambda x: abs(x), 'pos': lambda x: x / 2, 'zero': lambda x: x + 5}
lst = list(map(int, input("введи через пробел длину списка и его элементы: ").split()))
for i in lst:
    if i < 0:
        print(ops['neg'](i))
    elif i > 0:
        print(ops['pos'](i))
    else:
        print(ops['zero'](i))
```

Пример вывода для 4 -5 0 6 -3:

2.0

5

5

3.0

3

Задание 1

Напишите программу, которая получает от пользователя список, состоящий из целых чисел, и находит произведение его элементов. Решите задачу тремя способами:

с помощью reduce и лямбда-функции;

с math.prod() (используется для возврата произведений элементов) ;

с использованием пользовательской функции.

Пример ввода: 3 5 6 7 8 2 4 3 4

Вывод: 483840

Решение 1:

```
from functools import reduce
lst = list(map(int, input("введи через элементы списка: ").split()))
print(reduce(lambda x, y: x * y, lst))
```

Решение 2:

```
import math
lst = list(map(int, input("введи через элементы списка: ").split()))
print(math.prod(lst))
```

Решение 3:

```
def prod(lst):
    prod = 1
    for i in lst:
        prod *= i
    return prod

lst = list(map(int, input("введи через элементы списка: ").split()))
print(prod(lst))
```

Задание 2

Напишите программу для сортировки значений словаря в алфавитном порядке.

Словарь:

```
unsorted_dict = {'фрукт': 'яблоко', 'цвет': 'антрацит', 'артикул': 'в5678',
                 'модель': 'бабочка', 'наименование': 'книга', 'жанр': 'триллер'}
```

Ожидаемый результат:

```
{'цвет': 'антрацит', 'модель': 'бабочка', 'артикул': 'в5678', 'наименование': 'книга', 'жанр':
'триллер', 'фрукт': 'яблоко'}
```

Решение:

```
unsorted_dict = {'фрукт': 'яблоко', 'цвет': 'антрацит', 'артикул': 'в5678',
                 'модель': 'бабочка', 'наименование': 'книга', 'жанр':
'триллер'}
print(dict(sorted(unsorted_dict.items(), key=lambda item: item[1])))
```

ФАЙЛЫ

В случае, когда программа должна принимать в обработку и/или выдавать большой объем данных используют файлы.

С точки зрения программы файл можно рассматривать, как последовательность заранее неизвестного количества элементов данных, располагающихся за пределами оперативной памяти компьютера, например, на его жестком диске.

Поскольку программа может непосредственно оперировать только с объектами в оперативной памяти компьютера, для взаимодействия с файлами она вынуждена обращаться к операционной системе. Для связи файлов операционной системы с программой используется объект файлового типа.

Традиционный процесс работы с файлами строится по следующей схеме:

- определяется переменная файлового типа;
- эта переменная связывается с реальным файлом;
- файл открывается с целью его чтения, либо записи или дозаписи в него;
- выполняются операции по обмену данными с файлом;
- файл закрывается; при этом операционная система фиксирует сделанные в нем изменения.

Обычно файлы делят на текстовые и байтовые (бинарные).

Текстовые файлы рассматриваются как содержащие символьные данные, строки. Данные в текстовом файле хранятся в символьном виде, поэтому такой файл может обрабатываться в текстовом редакторе. Программа рассматривает текстовый файл как последовательность физических записей (строк), в которой к записи с номером k можно обратиться, только просмотрев предыдущие $k-1$ записи, поэтому такой доступ к данным называется последовательным.

При записи или чтении в/из текстового файла всегда передается именно строка, поэтому, если нужно записать числа, данные других типов, то их предварительно нужно конвертировать в строку.

Байтовые файлы рассматриваются как поток байтов. Побайтово считываются, например, файлы изображений.

Открытие файла выполняется с помощью, встроенной в Python функции `open()`.

У функции `open` много параметров. Пока важны 3 аргумента: первый, это имя файла. Путь к файлу может быть относительным или абсолютным. Второй аргумент, это режим, в котором мы будем открывать файл.

Обычно используются режимы чтения ('r') и записи ('w'). Если файл открыт в режиме чтения, то запись в него невозможна. Можно только считывать данные из него. Если файл открыт в режиме записи, то в него можно только записывать данные, считывать нельзя.

Если файл открывается в режиме 'w', то все данные, которые в нем были до этого, стираются. Файл становится пустым. Если не надо удалять существующие в файле данные, тогда следует использовать вместо режима записи, режим дозаписи ('a').

Если файл отсутствует, то открытие его в режиме 'w' создаст новый файл. Бывают ситуации, когда надо гарантировано создать новый файл, избежав случайной перезаписи данных существующего. В этом случае вместо режима 'w' используется режим 'x'. В нем всегда создается новый файл для записи. Если указано имя существующего файла, то будет выброшено исключение. Потери данных в уже имеющемся файле не произойдет.

Если при вызове `open()` второй аргумент не указан, то файл открывается в режиме чтения как текстовый файл. Чтобы открыть файл как байтовый, дополнительно к букве режима чтения/записи добавляется символ 'b'. Буква 't' обозначает текстовый файл. Поскольку это тип файла по умолчанию, то обычно ее не указывают.

Нельзя указывать только тип файла, то есть `open("имя_файла", 'b')` есть ошибка, даже если файл открывается на чтение. Правильно – `open("имя_файла", 'rb')`. Только текстовые файлы можно открыть командой `open("имя_файла")`, потому что и 'r' и 't' подразумеваются по- умолчанию.

Режимы могут быть объединены, то есть, к примеру, 'rb' - чтение в двоичном режиме. По умолчанию режим равен 'rt'.

И последний аргумент, encoding, нужен только в текстовом режиме чтения файла. Этот аргумент задает кодировку.

Функция open() возвращает объект файлового типа. Его надо либо сразу связать с переменной (объект файлового типа), чтобы не потерять, либо сразу прочитать.

С помощью файлового метода read() можно прочитать файл целиком или только определенное количество байт. Текстовый файл можно читать по байтам.

Раздаточный материал № 110

```
f1 = open('new_file_1.txt')
print('Читаем и выводим на экран первые 10 байт или символов')
print(f1.read(10))
f1.close()

f1 = open('new_file_1.txt')
print('Читаем и выводим на экран весь файл')
print(f1.read())
f1.close()

f1 = open('new_file_1.txt')
print('Читаем и выводим на экран весь файл с помощью for')
for line in f1:
    print(line, end='')
print(type(f1.read()))
f1.close()
```

Пояснения:

1. Объект файлового типа, т.е. переменная f1, после выполнения операции считывания становится пустым, поэтому файл необходимо открывать еще раз, предварительно закрыв его после очередной операции.
2. Объект файлового типа, т.е. переменная f1, относится к итераторам, т.е. возможно последовательное извлечение элементов, т.е. сразу в цикле без использования методов чтения.
3. Данные, прочитанные из файла, всегда возвращаются в сценарий в виде строки. Поэтому строку всегда нужно преобразовывать в другой тип, если строка неуместна.
4. Параметр encoding="utf-8" задает кодировку читаемого файла.

Раздаточный материал № 111

Метод	Описание
readline()	Чтение файла построчно
readlines()	Считывает сразу все строки и создает список

Запись в файл выполняется с помощью методов write() и writelines(). Во второй можно передать структуру данных. Метод write() возвращает количество записанных символов. Методы write() и writelines() автоматически не ставят символ переноса строки, и это программисту нужно контролировать самостоятельно.

Раздаточный материал № 112

```
print('Запишем в файл структуру данных - список')
l = ['tree', 'four']
f2 = open('data.txt', 'w')
f2.write('one')
```



```
f2.write(' two')
f2.writelines(1)
f2.close()
f2 = open('data.txt')
print(f2.read())
print(type(f2.read())) # получаем тип - строка
f2.close()
```

После того как работа с файлом закончена, важно не забывать его закрыть, чтобы освободить место в памяти. Если файл открывается в заголовке цикла, то интерпретатор его закрывает при завершении работы цикла или через какое-то время. Закрывается файл с помощью файлового метода `close()`. Свойство файлового объекта `closed` позволяет проверить закрыт ли файл.

Раздаточный материал № 113

```
# содержимое файла data_2.txt:
# зима
# весна
# лето
# осень

nums = []
for i in open('data_2.txt', encoding='UTF-8'):
    nums.append(i[:-1])
print(nums)
print('получаем тип', type(nums))
```

Результат

```
['зима', 'весна', 'лето', 'осень']
```

Существует удобный способ записи в файл - использование функции `print()`. Если передать в необязательный аргумент `file` объект файлового типа, то поток вывода функции `print()` перенаправляется из консоли в файл. Преимущество такого подхода заключается в том, что в `print()` можно передавать не обязательно строковые аргументы — при необходимости функция сама их преобразует к строковому типу.

Использование оператора «with»

В Python встроенный инструмент, который помогает упростить чтение и редактирование файлов. Оператор `with` создает диспетчер контекста в Пайтоне, который автоматически закрывает файл по окончании работы в нем.

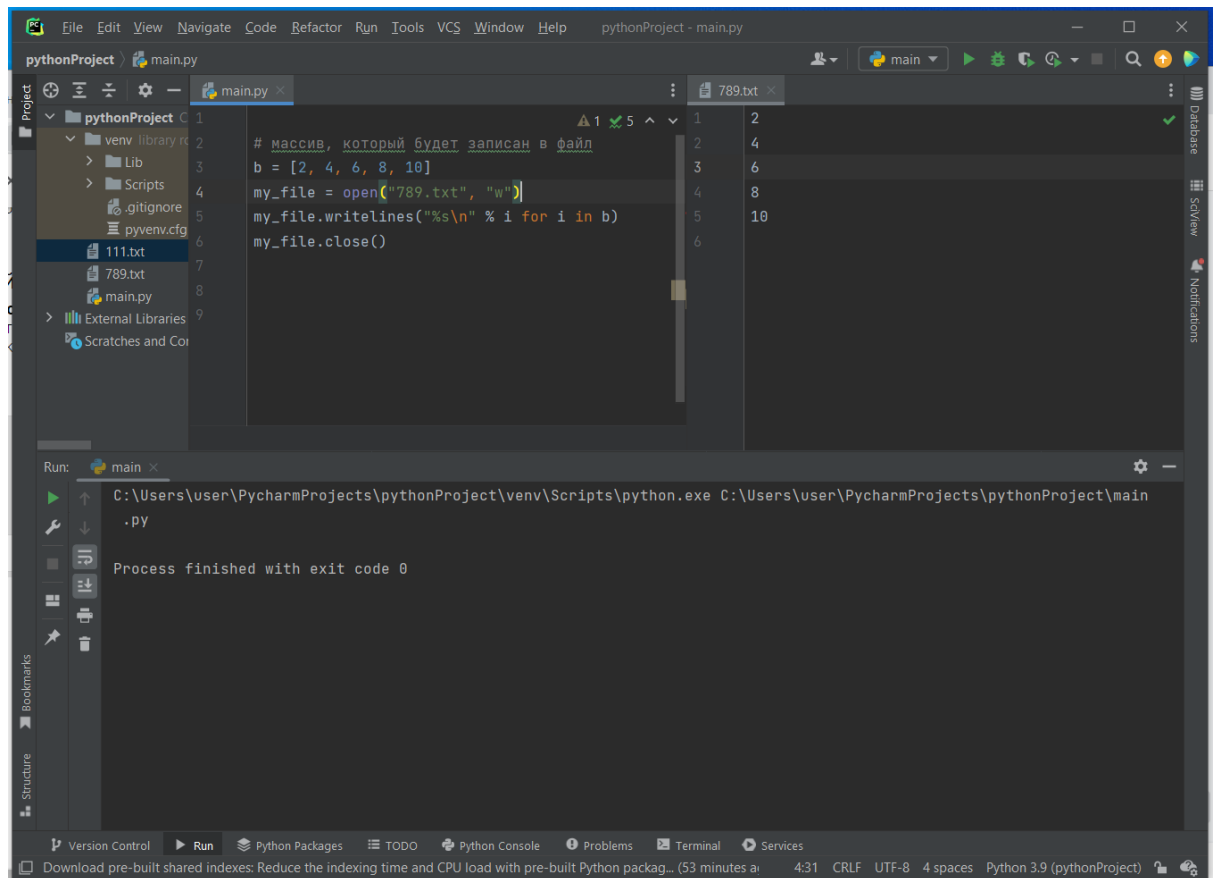
Т.е. можно выполнять все стандартные операции ввода\вывода, в привычном порядке в пределах блока кода. После ухода из блока кода, файловый дескриптор закроет его, и его уже нельзя будет использовать.

Задача 1.

Используя встроенные команды языка *Python*, записать в произвольный текстовый файл следующую строку: **b = [2, 4, 6, 8, 10]**. После этого закрыть файл.

Решение.

Напишем код для решения данной практической задачи:



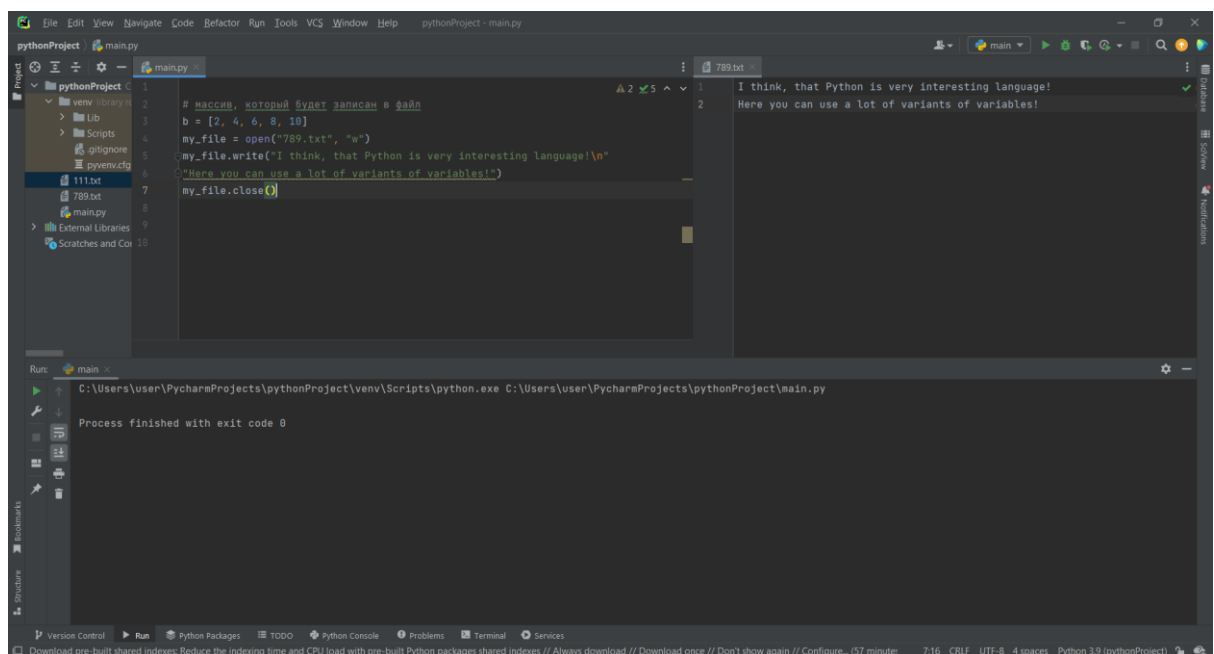
Задача 2.

Используя встроенные команды языка *Python*, записать в произвольный текстовый файл данную информацию: «I think, that Python is very interesting language!

Here you can use a lot of variants of variables!». Каждое предложение начинается с новой строки. После этого закрыть файл.

Решение.

Напишем код для решения следующей практической задачи:

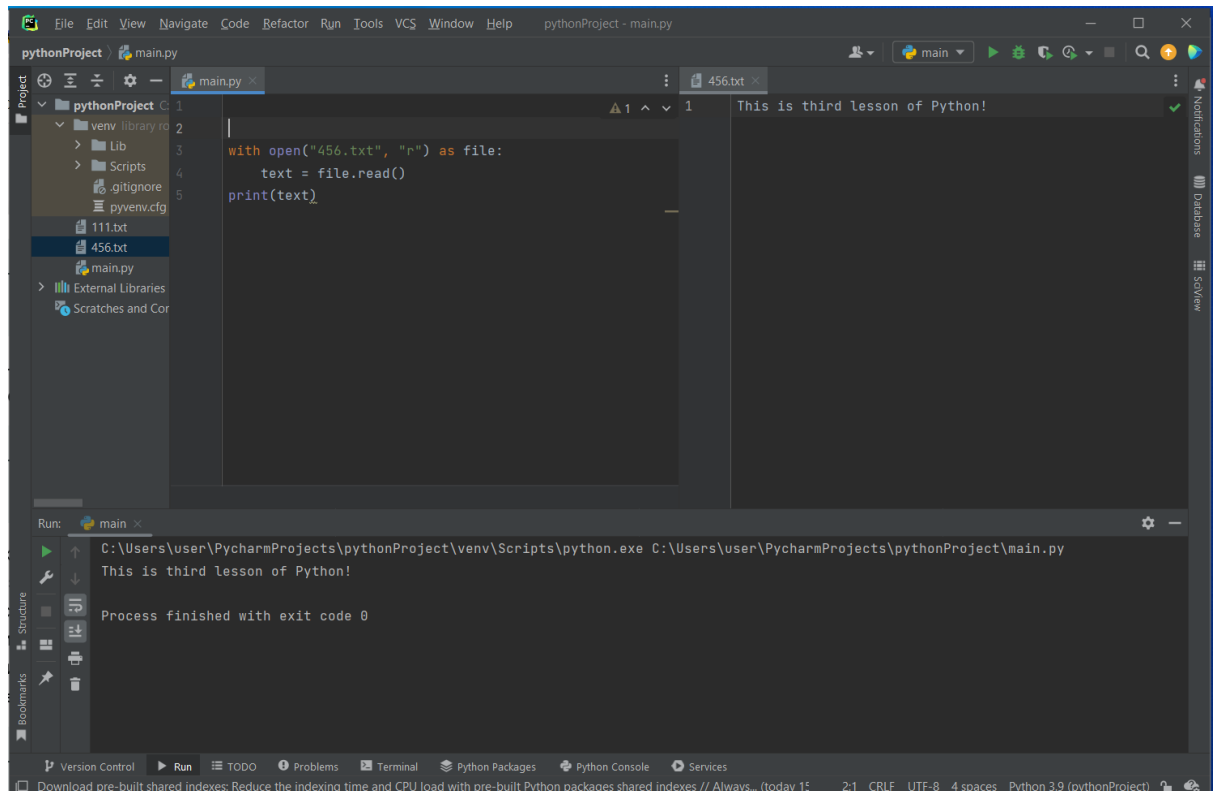


Задача 3.

Используя встроенные команды языка *Python*, создать пустой файл с именем *456.txt* и записать в него (вручную) предложение: «This is third lesson of Python!». После этого вывести содержимое данного файла целиком на консоль.

Решение.

После создания файла запишем в него предложение и выведем содержимое на консоль:



Задача 4.

```
# Средствами языка Python сформировать текстовый файл (.txt) содержащий
# последовательность из целых положительных и отрицательных чисел.
# Сформировать новый текстовый файл (.txt) следующего вида,
# предварительно выполнив требуемую обработку элементов:
# Исходные данные:
# Количество элементов:
# Максимальный элемент:
# Количество отрицательных элементов:

# Запишем в файл data_3.txt структуру данных - список
l = ['-99 6 12 -36 20 45 100 -15']
f3 = open('data_3.txt', 'w')
f3.writelines(l)
f3.close()

# Дублируем список в новый файл data_4.txt
f4 = open('data_4.txt', 'w')
f4.write('Исходные данные: ')
f4.write('\n')
f4.writelines(l)
f4.close()

# разбиваем строку и ее значения преобразуем в числа
f3 = open('data_3.txt')
k = f3.read()
k = k.split()
for i in range(len(k)):
    k[i] = int(k[i])
f3.close()

# Ищем максимальный элемент и количество отрицательных элементов
# в файле data_3.txt и записываем в файл data_4.txt
f3 = open('data_3.txt')
max, t = 0, 0
for i in range(len(k)):
    max = max if max > k[i] else k[i]
    if k[i] < 0:
        t += 1

f4 = open('data_4.txt', 'a') # открываем файл для дозаписи
f4.write('\n')
print('Количество элементов: ', len(k), 'Максимальный элемент: ', max,
file=f4)
f4.close()
```

Задача 5.

Из предложенного текстового файла (text18-1.txt) вывести на экран его содержимое,
посчитать и вывести количество строк и количество букв 'ж'.
Сформировать новый файл, в который поместить текст, предварительно поменяв
местами первую и четвертую строки.

```
t = 0
d = 0
for i in open('text18-1.txt', encoding='UTF-8'):
    print(i, end='')
    t += 1
    for j in i:
        if j == 'ж':
            d += 1
print(end='\n')
print('Количество строк: ', t, end='\n')
print('Количество букв "ж" : ', d, end='\n')
```

```
f1 = open('text18-1.txt', encoding='UTF-8')
l = f1.readlines()
l[0], l[3] = l[3], l[0]
f1.close()
```

```
f2 = open('text18-2.txt', 'w')
f2.writelines(l)
f2.close()
```

Работа с файлами в Python с помощью модуля OS

Обработка файлов в Python с помощью модуля os включает создание, переименование, перемещение, удаление файлов и папок, а также получение списка всех файлов и каталогов и многое другое.

Модуль встроенный, поэтому для работы с ним не нужно ничего устанавливать.

Вывод текущей директории

Для получения текущего рабочего каталога используется os.getcwd():

```
import os

# вывести текущую директорию
print("Текущая директория:", os.getcwd())
```

os.getcwd() возвращает строку в Юникоде, представляющую текущий рабочий каталог.

Создание папки

Для создания папки/каталога в любой операционной системе нужна следующая команда:

```
# создать пустой каталог (папку)
os.mkdir("folder")
```

После ее выполнения в текущем рабочем каталоге тут же появится новая папка с названием «folder».

Если запустить ее еще раз, будет вызвана ошибка FileExistsError, потому что такая папка уже есть. Для решения проблемы нужно запускать команду только в том случае, если каталога с таким же именем нет. Этого можно добиться следующим образом:

```
# повторный запуск mkdir с тем же именем вызывает FileExistsError,
# вместо этого запустите:
if not os.path.isdir("folder"):
    os.mkdir("folder")
```

Функция os.path.isdir() вернет True, если переданное имя ссылается на существующий каталог.

Изменение директории

```
# изменение текущего каталога на 'folder'
os.chdir("folder")
```

Еще раз выведем рабочий каталог:

```
# вывод текущей папки
print("Текущая директория изменилась на folder:", os.getcwd())
```

Создание вложенных папок

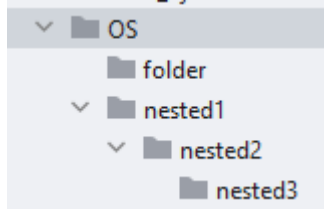
Предположим, вы хотите создать не только одну папку, но и несколько вложенных:

```
# вернуться в предыдущую директорию
os.chdir("..")

# сделать несколько вложенных папок
os.makedirs("nested1/nested2/nested3")
```

!!! Внимание!!! После выполнения – закомментировать этот код.

Это создаст три папки рекурсивно, как показано на следующем изображении:



Для дальнейшей работы создадим файл

```
# создать новый текстовый файл
text_file = open("text.txt", "w")
# запись текста в этот файл
text_file.write("Это текстовый файл")
text_file.close()
```

!!! Внимание!!! После выполнения – закомментировать этот код.

Переименование файлов

```
# переименовать text.txt на renamed-text.txt
os.rename("text.txt", "renamed-text.txt")
```

!!! Внимание!!! После выполнения – закомментировать этот код.

Функция `os.rename()` принимает 2 аргумента: имя файла или папки, которые нужно переименовать и новое имя.

Перемещение файлов

Функцию `os.replace()` можно использовать для перемещения файлов или каталогов:

```
# заменить (переместить) этот файл в другой каталог
os.replace("renamed-text.txt", "folder/renamed-text.txt")
```

!!! Внимание!!! После выполнения – закомментировать этот код.

Стоит обратить внимание, что это перезапишет путь, поэтому если в папке `folder` уже есть файл с таким же именем (`renamed-text.txt`), он будет перезаписан.

Список файлов и директорий

```
# распечатать все файлы и папки в текущем каталоге
print("Все папки и файлы:", os.listdir())
```

Функция `os.listdir()` возвращает список, который содержит имена файлов в папке. Если в качестве аргумента не указывать ничего, вернется список файлов и папок текущего рабочего каталога.

Если нужно узнать состав папок, то используем функцию `os.walk()`:

```
# распечатать все файлы и папки рекурсивно
for dirpath, dirnames, filenames in os.walk("."):
    # перебрать каталоги
    for dirname in dirnames:
        print("Каталог:", os.path.join(dirpath, dirname))
    # перебрать файлы
    for filename in filenames:
        print("Файл:", os.path.join(dirpath, filename))
```

`os.walk()` — это генератор дерева каталогов. Он будет перебирать все переданные составляющие. Здесь в качестве аргумента передано значение «.», которое обозначает верхушку дерева.

Метод `os.path.join()` был использован для объединения текущего пути с именем файла/папки.

Удаление файлов

```
# удалить созданный файл
os.remove("folder/renamed-text.txt")
```

!!! Внимание!!! После выполнения — закомментировать этот код.

`os.remove()` удалит файл с указанным именем (не каталог).

Удаление директорий

```
# удалить папку
os.rmdir("folder")
```

!!! Внимание!!! После выполнения — закомментировать этот код.

Для удаления каталогов рекурсивно необходимо использовать `os.removedirs()`. Это удалит только пустые каталоги.

```
# удалить вложенные папки
os.removedirs("nested1/nested2/nested3")
```

!!! Внимание!!! После выполнения — закомментировать этот код.

Получение информации о файлах

Для получения информации о файле в ОС используется функция `os.stat()`, которая выполняет системный вызов `stat()` по выбранному пути:

```
# вывести некоторые данные о файле
print(os.stat("text.txt"))
```

Вернется кортеж с отдельными метриками. В их числе есть следующие:

`st_size` — размер файла в байтах

`st_atime` — время последнего доступа в секундах (временная метка)

`st_mtime` — время последнего изменения

`st_ctime` — в Windows это время создания файла, а в Linux — последнего изменения метаданных

Для получения конкретного атрибута нужно писать следующим образом:

```
# например, получить размер файла
print("Размер файла:", os.stat("text.txt").st_size)
```


Практические задачи

Задача 1.

Выберите любую папку на своем компьютере, имеющую вложенные директории. Выведите на печать в терминал ее содержимое, как и всех подкаталогов при помощи функции `print_docs(directory)`.

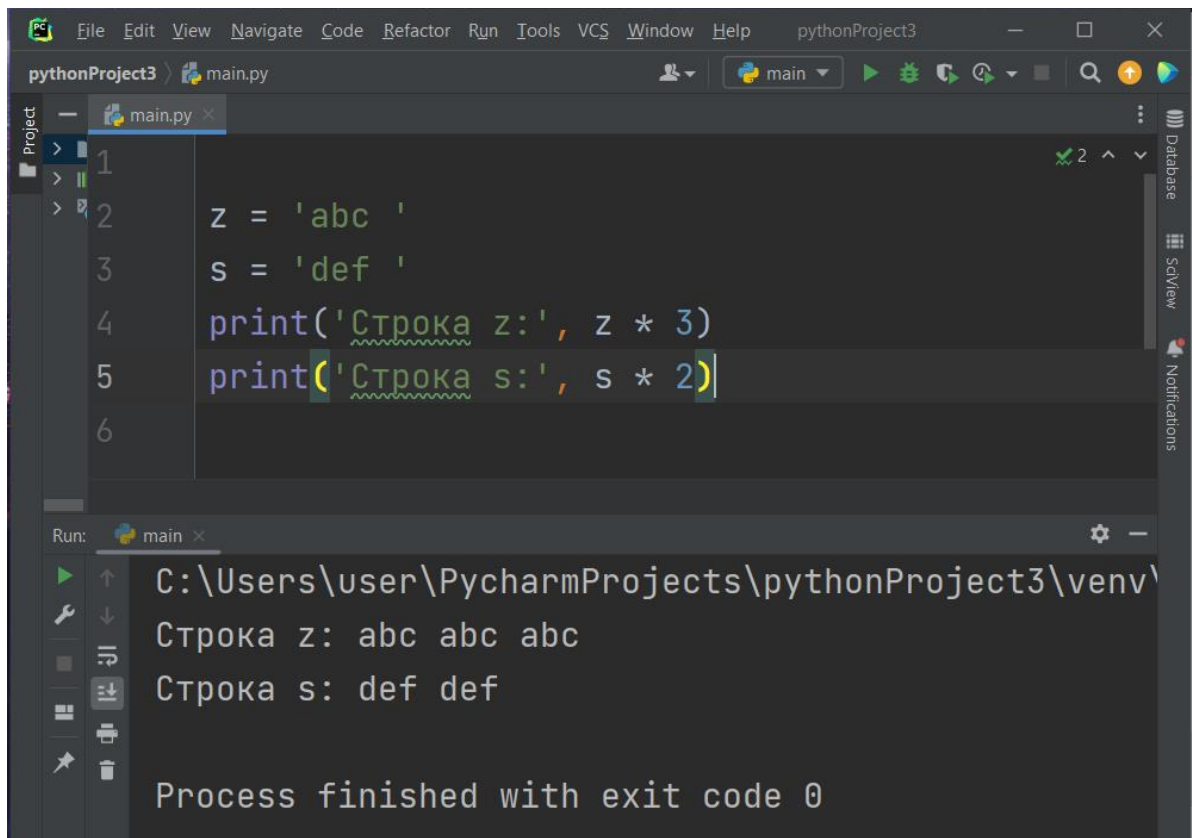
```
import os

def print_docs(directory):
    all_files = os.walk(directory)
    for catalog in all_files:
        print(f'Папка {catalog[0]} содержит:')
        print(f'Директории: {", ".join([folder for folder in catalog[1]])}')
        print(f'Файлы: {", ".join([file for file in catalog[2]])}')
        print('-' * 40)

print_docs('C:/Program Files')
```

Задача 2.

Создать программу, в которой есть две строки: `'z' = 'abc'` и `'s' = 'def'`. Помимо этого, в данной программе реализован оператор `*`. Необходимо вывести три раза подряд строку `'z'`, а затем строку `'s'` два раза подряд.



The screenshot shows the PyCharm IDE interface. The top pane displays a Python script in `main.py` with the following code:

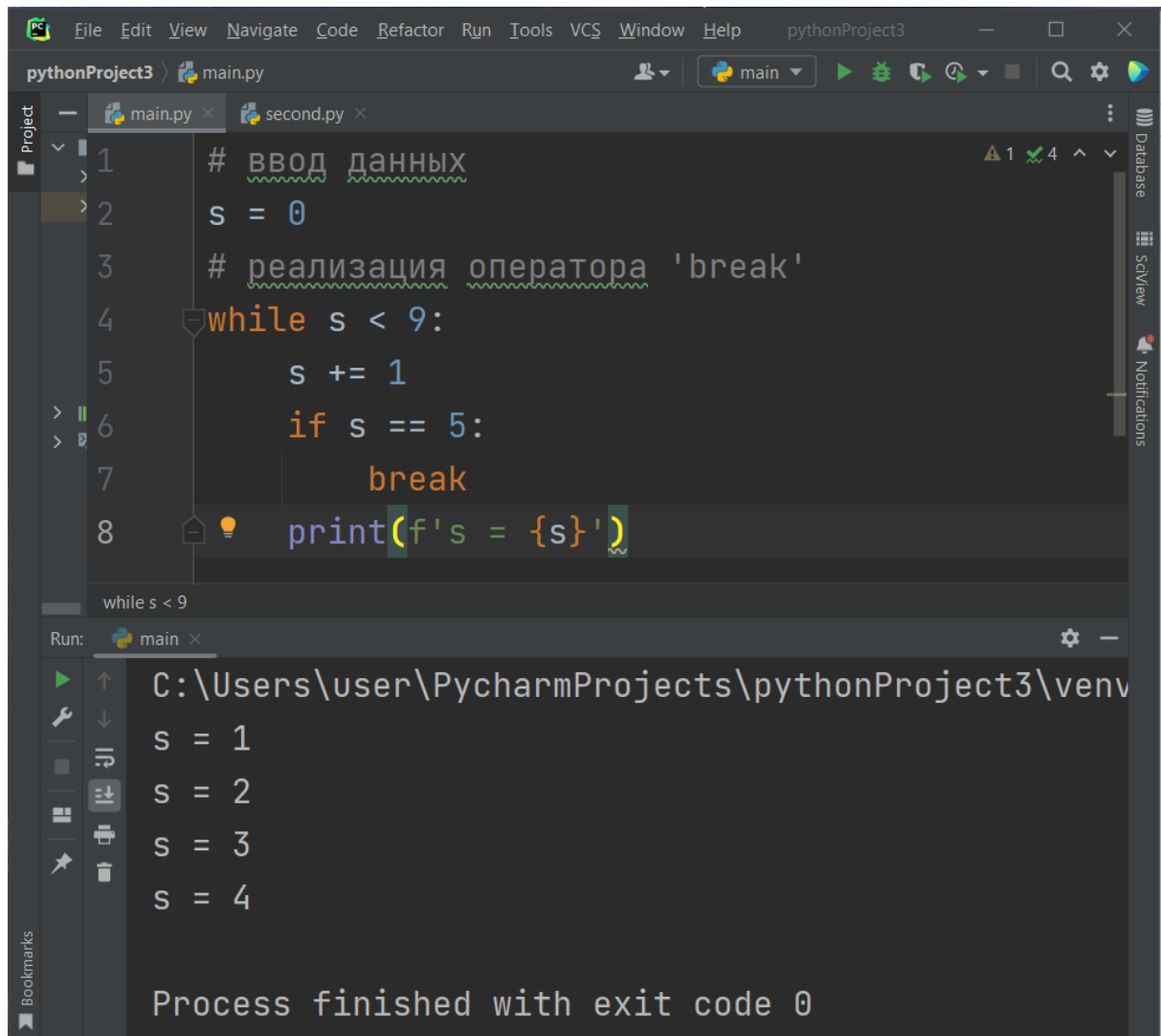
```
1 z = 'abc '
2 s = 'def '
3
4 print('Строка z:', z * 3)
5 print('Строка s:', s * 2)
6
```

The bottom pane shows the output of the script execution:

```
C:\Users\user\PycharmProjects\pythonProject3\venv\
Строка z: abc abc abc
Строка s: def def
Process finished with exit code 0
```

Задача 3.

Создать программу, в которой реализован оператор выхода из цикла **break**. Изначально вводится переменная 's' со значением 0. При каждой итерации цикла значение переменной 's' увеличивается на единицу. Когда значение переменной 's' становится равным 5, оператор **break** завершает работу цикла (на консоли должны быть выведены значения с 1 по 4 включительно).



```
# ВВОД ДАННЫХ
s = 0
# реализация оператора 'break'
while s < 9:
    s += 1
    if s == 5:
        break
    print(f's = {s}')
```

The screenshot shows the PyCharm IDE with a file named `main.py` open. The code in the editor is a Python script that initializes `s = 0` and enters a `while s < 9:` loop. Inside the loop, `s` is incremented by 1, and if it reaches 5, a `break` statement is executed. Otherwise, the current value of `s` is printed. The Run window at the bottom shows the output: `s = 1`, `s = 2`, `s = 3`, and `s = 4`, followed by the message "Process finished with exit code 0".

Задача 4.

Создать программу, в которой используются переменные с именами 'first_one', 'second_one' и 'res'. Необходимо использовать оператор **and**. Реализовать две итерации: когда значение переменной 'res' является **True** (например, когда значение переменной 'first_one' сравнивается с числом 20) и когда значение является **False**. Результаты обеих итераций продемонстрировать на консоли.

The screenshot shows the PyCharm IDE with a Python file named `main.py`. The code is as follows:

```
1 # ВВОД ДАННЫХ
2 first_one = 25
3 second_one = 65
4
5 # ИСПОЛЬЗОВАНИЕ ЛОГ. ОПЕРАТОРА 'and'
6 res = first_one > 20 and second_one == 65
7
8 # ВЫВОД ДАННЫХ
9 print('Переменная "first_one":', first_one)
10 print('Переменная "second_one":', second_one)
11 print('Переменная "res":', res)
```

The Run console shows the output:

```
Переменная "first_one": 25
Переменная "second_one": 65
Переменная "res": True

Process finished with exit code 0
```

The status bar at the bottom indicates: Python 3.9 (pythonProject3).

The screenshot shows the PyCharm IDE with a modified Python file named `main.py`. The code is as follows:

```
1 # ВВОД ДАННЫХ
2 first_one = 25
3 second_one = 65
4
5 # ИСПОЛЬЗОВАНИЕ ЛОГ. ОПЕРАТОРА 'and'
6 res = first_one > 50 and second_one == 66
7
8 # ВЫВОД ДАННЫХ
9 print('Переменная "first_one":', first_one)
10 print('Переменная "second_one":', second_one)
11 print('Переменная "res":', res)
```

The Run console shows the output:

```
Переменная "first_one": 25
Переменная "second_one": 65
Переменная "res": False

Process finished with exit code 0
```

The status bar at the bottom indicates: Python 3.9 (pythonProject3).