

ТЕМА 2. БАЗОВЫЕ ОПЕРАЦИИ ЯЗЫКА PYTHON

2.1. Комментарии на Python

В Python комментарии - это фрагменты кода, которые игнорируются интерпретатором и не выполняются. Они используются для добавления пояснений к коду и для того, чтобы сделать его более понятным и легко читаемым для других программистов.

В Python существуют два вида комментариев: однострочные комментарии и многострочные комментарии.

Однострочные комментарии начинаются с символа "#" и продолжаются до конца строки. Например:

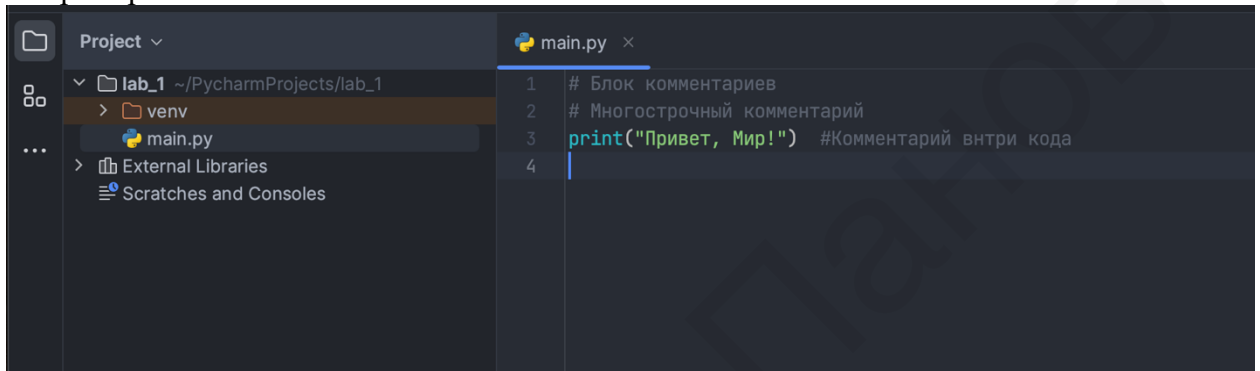


Рис. 2.1. Пример однострочных комментариев

Многострочные комментарии начинаются и заканчиваются тремя кавычками (двойными или одинарными) и могут занимать несколько строк. Например:

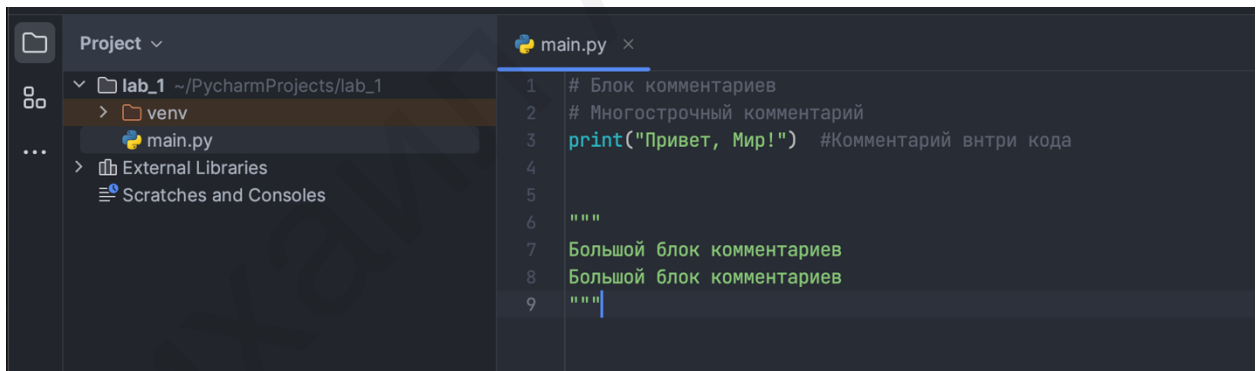


Рис. 2.2. Пример многострочных комментариев

2.2. Функция print

Функция `print()` в Python используется для вывода текста или значения переменных на консоль или в файл. Синтаксис функции `print()` очень простой:

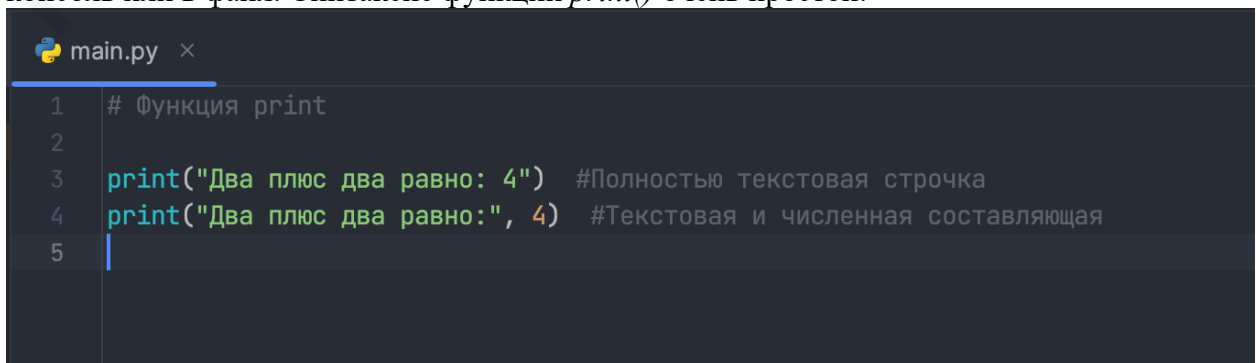
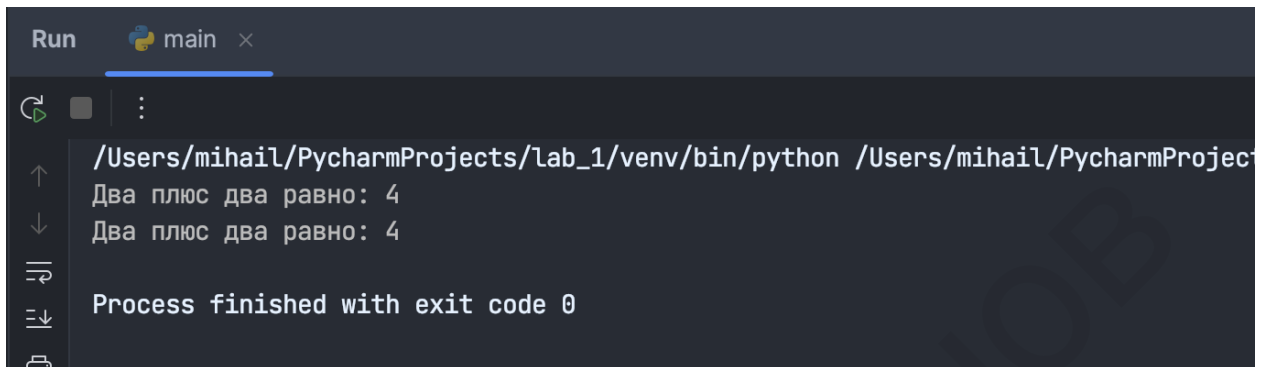


Рис. 2.3. Пример функции *print*

В 3 строчке кода нами была использована полностью текстовая информация ее можно обозначить в двойных или одинарных кавычках.

В 4 строке кода мы вывели как текст, так и число.

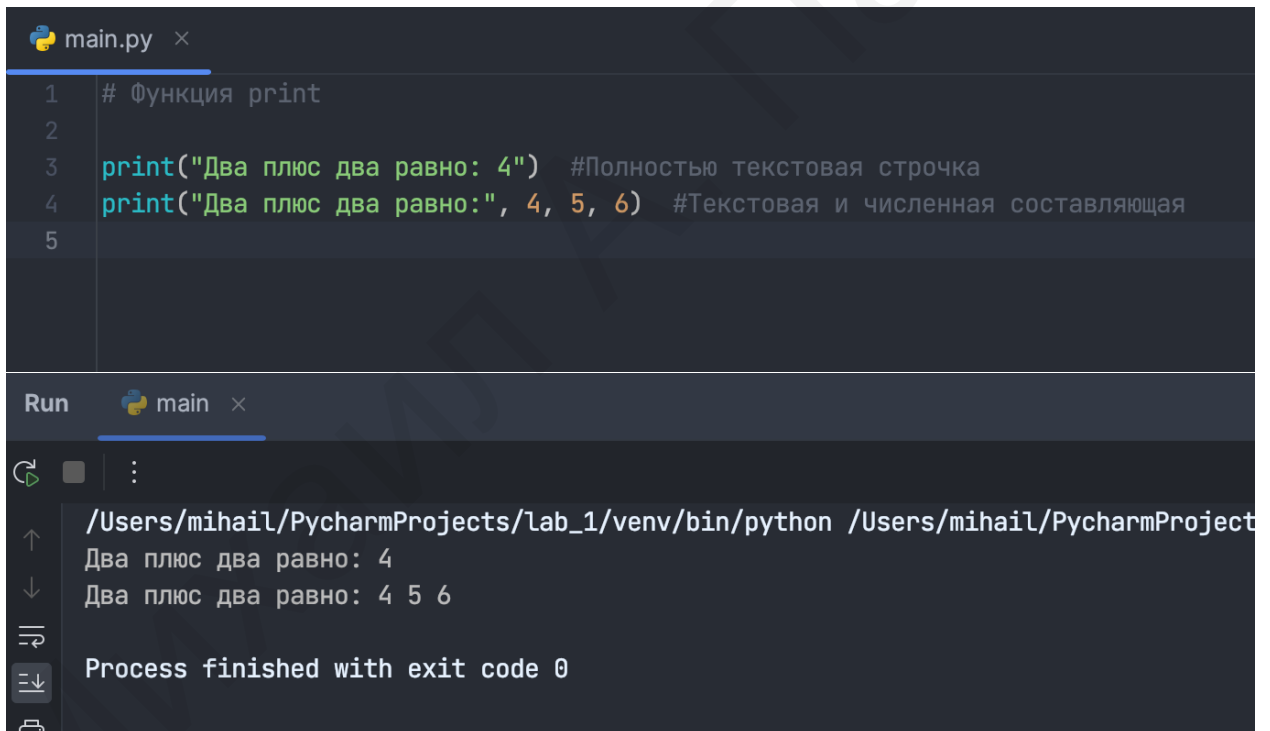


The screenshot shows a terminal window titled 'Run' with a Python icon and 'main' tab. It displays the output of a Python script. The first line of output is 'Два плюс два равно: 4'. The second line is also 'Два плюс два равно: 4'. Below the output, it says 'Process finished with exit code 0'.

```
Run  Python main x
↑
↓
↶
↷
Process finished with exit code 0
```

Рис. 2.4. Вывод функции *print*

Можно обратить внимание, на то, что пробелы также сохраняются.



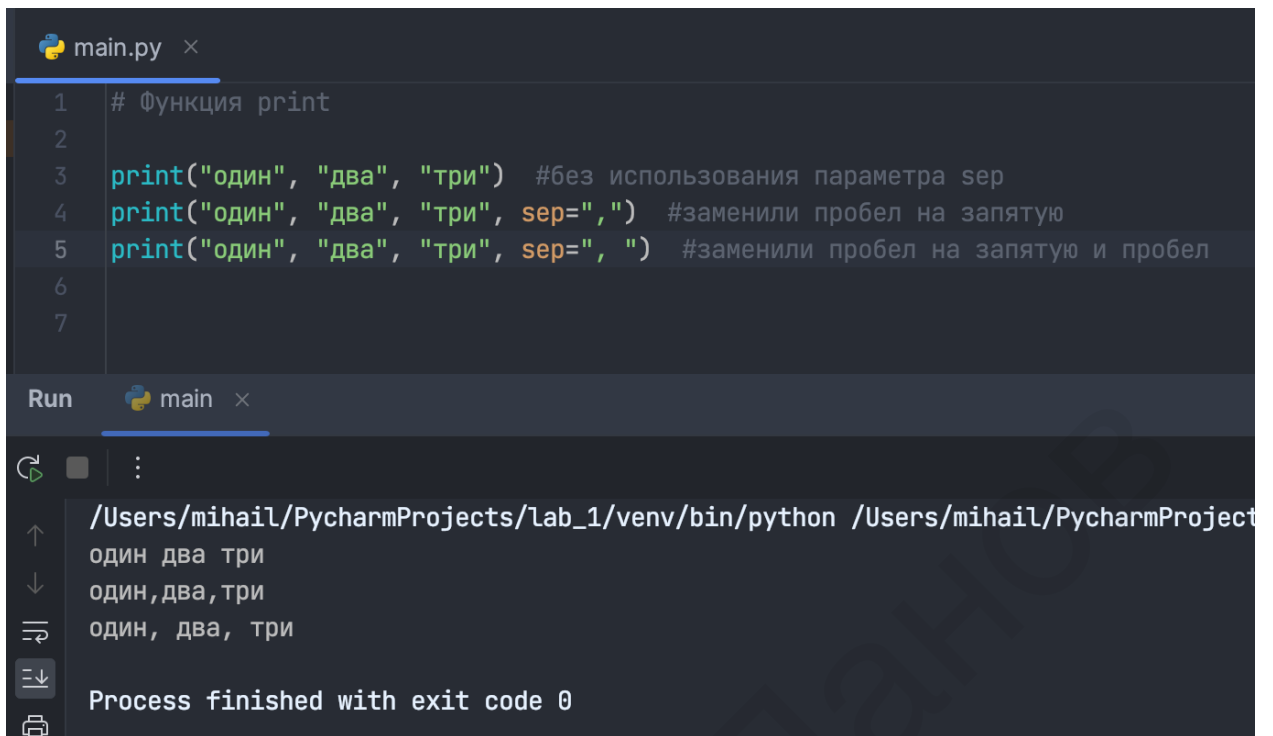
The screenshot shows a Python IDE with a file named 'main.py'. The code contains two print statements. The first is `print("Два плюс два равно: 4")` with a comment '#Полностью текстовая строка'. The second is `print("Два плюс два равно:", 4, 5, 6)` with a comment '#Текстовая и численная составляющая'. Below the code, the terminal window shows the output: 'Два плюс два равно: 4' and 'Два плюс два равно: 4 5 6'. The terminal also shows 'Process finished with exit code 0'.

```
main.py x
1 # Функция print
2
3 print("Два плюс два равно: 4") #Полностью текстовая строка
4 print("Два плюс два равно:", 4, 5, 6) #Текстовая и численная составляющая
5

Run  Python main x
↑
↓
↶
↷
Process finished with exit code 0
```

Рис. 2.5. Вывод функции *print* с использованием пробелов

sep - это параметр функции *print()* в Python, который определяет символ или строку, которая будет использоваться в качестве разделителя между элементами, выводимыми с помощью функции *print()*. По умолчанию *sep* равен пробелу. Пример использования *sep*:



```
main.py x
1 # Функция print
2
3 print("один", "два", "три") #без использования параметра sep
4 print("один", "два", "три", sep=",") #заменяли пробел на запятую
5 print("один", "два", "три", sep=", ") #заменяли пробел на запятую и пробел
6
7

Run main x

/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProject
↑
один два три
↓
один,два,три
⇨
один, два, три
⇩
Process finished with exit code 0
```

Рис. 2.6. Пример параметра *sep*

Функция *print()* в Python имеет несколько параметров, которые позволяют управлять выводом. Ниже приведены некоторые из наиболее часто используемых параметров:

sep: определяет разделитель между элементами, переданными в *print()*. По умолчанию *sep* равен пробелу.

end: определяет, какой символ будет добавлен в конец строки, переданной в *print()*. По умолчанию *end* равен символу перевода строки *\n*.

file: определяет файл, в который будет выведен результат работы функции *print()*. По умолчанию вывод осуществляется на стандартный вывод (консоль).

flush: определяет, должен ли буфер вывода быть очищен после вызова *print()*. По умолчанию *flush* равен *False*.



```
main.py x
1 # Функция print
2
3 print("один", "два", "три", sep=" ", end=".\\n")

Run main x

/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjec
один, два, три.

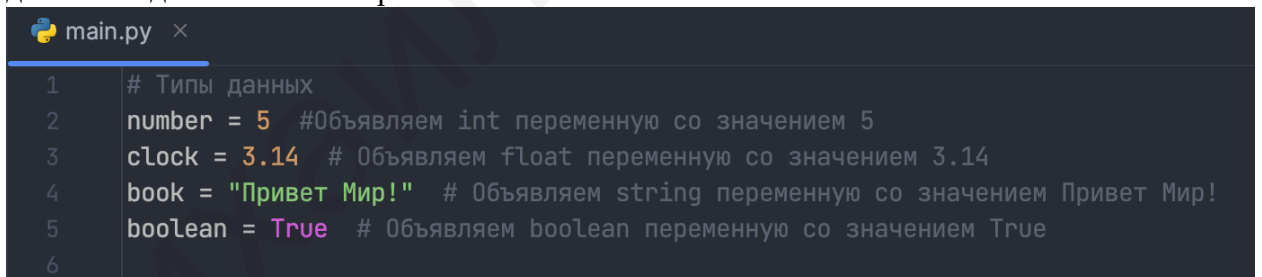
Process finished with exit code 0
```

Рис. 2.7. Пример параметра с использованием *sep* и *end*

Результатом этого кода будет вывод на консоль строки "один, два, три" где запятая и пробел используются в качестве разделителя между элементами, а точка и символ перевода строки `\\n` добавляются в конец строки.

2.3. Типы переменных в Python

В Python тип переменной определяется автоматически во время выполнения программы. Такой подход называется "динамической типизацией". Для объявления переменной мы должны задать имя этой переменной и значение.



```
main.py x
1 # Типы данных
2 number = 5 # Объявляем int переменную со значением 5
3 clock = 3.14 # Объявляем float переменную со значением 3.14
4 book = "Привет Мир!" # Объявляем string переменную со значением Привет Мир!
5 boolean = True # Объявляем boolean переменную со значением True
6
```

Рис. 2.8. Типы переменных в Python

В Python есть несколько встроенных типов данных, которые используются для хранения различных типов значений:

Числа (int, float, complex) - используются для хранения числовых значений.

Строки (str) - используются для хранения текстовых значений.

Списки (list) - используются для хранения упорядоченных коллекций элементов.

Кортежи (tuple) - используются для хранения упорядоченных неизменяемых коллекций элементов.

Словари (dict) - используются для хранения неупорядоченных коллекций пар ключ-значение.

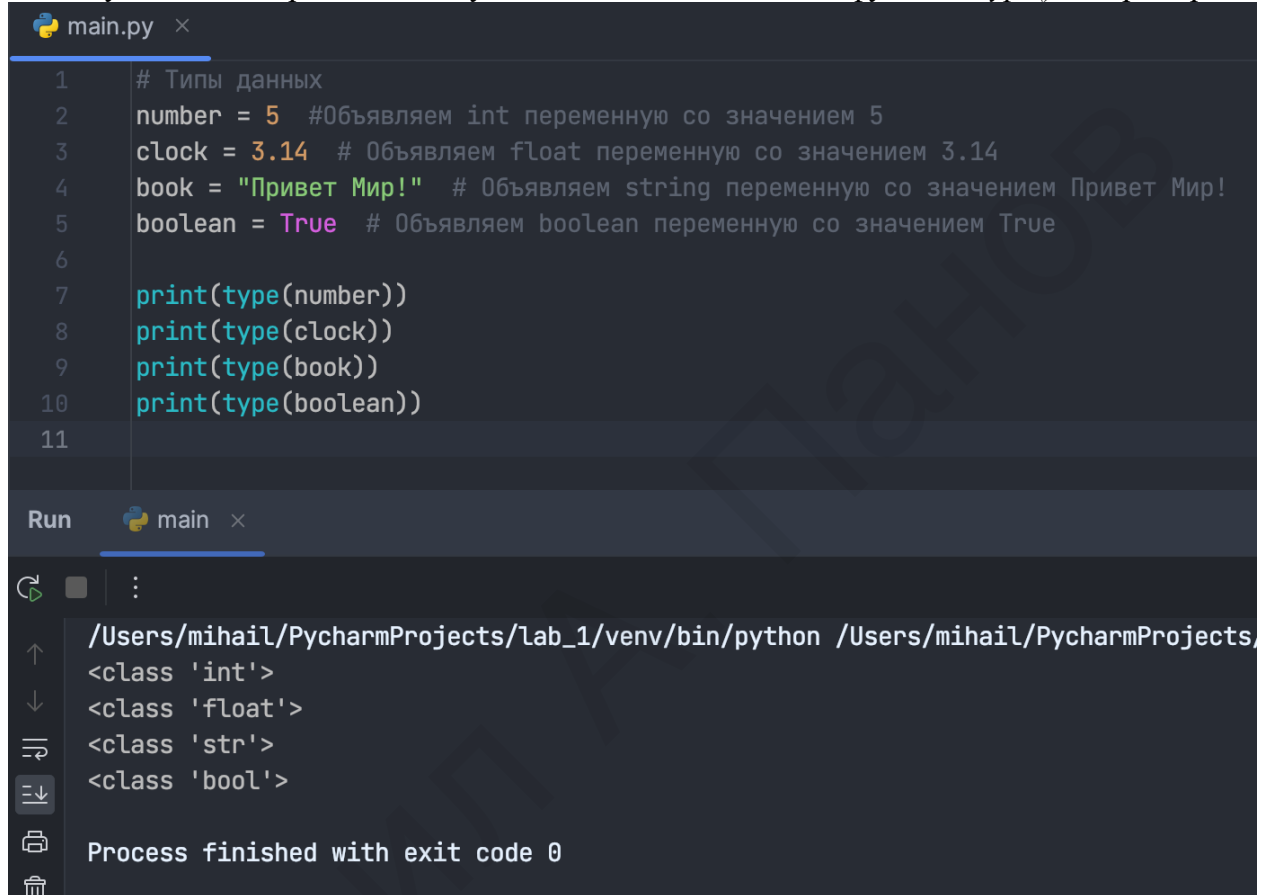
Множества (set) - используются для хранения неупорядоченных уникальных элементов.

Булев тип (bool) - используется для хранения логических значений True или False.

Также в Python есть возможность создания пользовательских типов данных с помощью классов.

NoneType – Его единственное возможное значение – `None`. Обычно `None` используется, когда вы хотите создать переменную (поскольку Python не отличает создание от присвоения: создание переменной – это просто присвоение ей значения), но пока не хотите присваивать ей какое-либо конкретное значение.

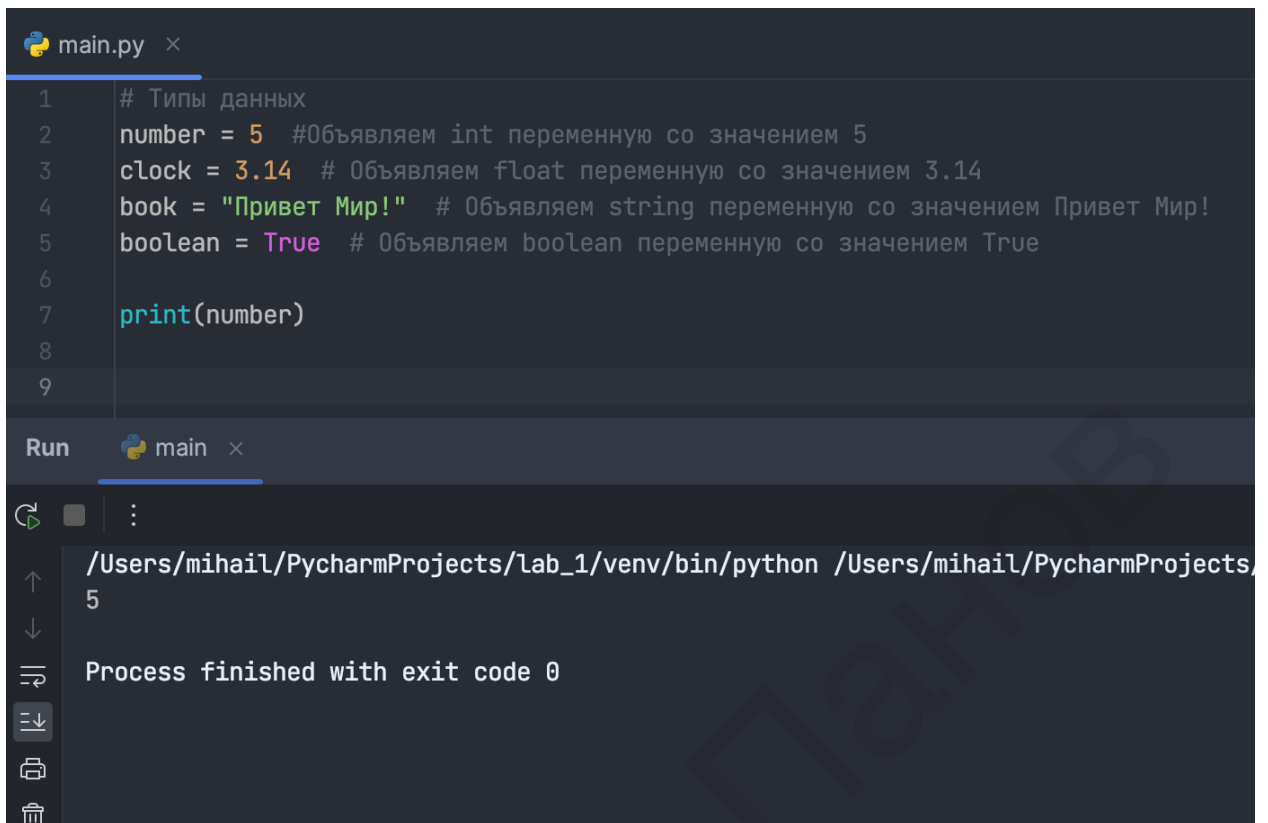
Чтобы узнать тип переменной в Python, можно использовать функцию `type()`, например:



```
main.py x
1 # Типы данных
2 number = 5 #Объявляем int переменную со значением 5
3 clock = 3.14 # Объявляем float переменную со значением 3.14
4 book = "Привет Мир!" # Объявляем string переменную со значением Привет Мир!
5 boolean = True # Объявляем boolean переменную со значением True
6
7 print(type(number))
8 print(type(clock))
9 print(type(book))
10 print(type(boolean))
11
Run main x
/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects,
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
Process finished with exit code 0
```

Рис. 2.9. Пример работы с `type()`

Мы можем вывести значение переменной через `print()`



```
main.py x
1 # Типы данных
2 number = 5 #Объявляем int переменную со значением 5
3 clock = 3.14 # Объявляем float переменную со значением 3.14
4 book = "Привет Мир!" # Объявляем string переменную со значением Привет Мир!
5 boolean = True # Объявляем boolean переменную со значением True
6
7 print(number)
8
9
```

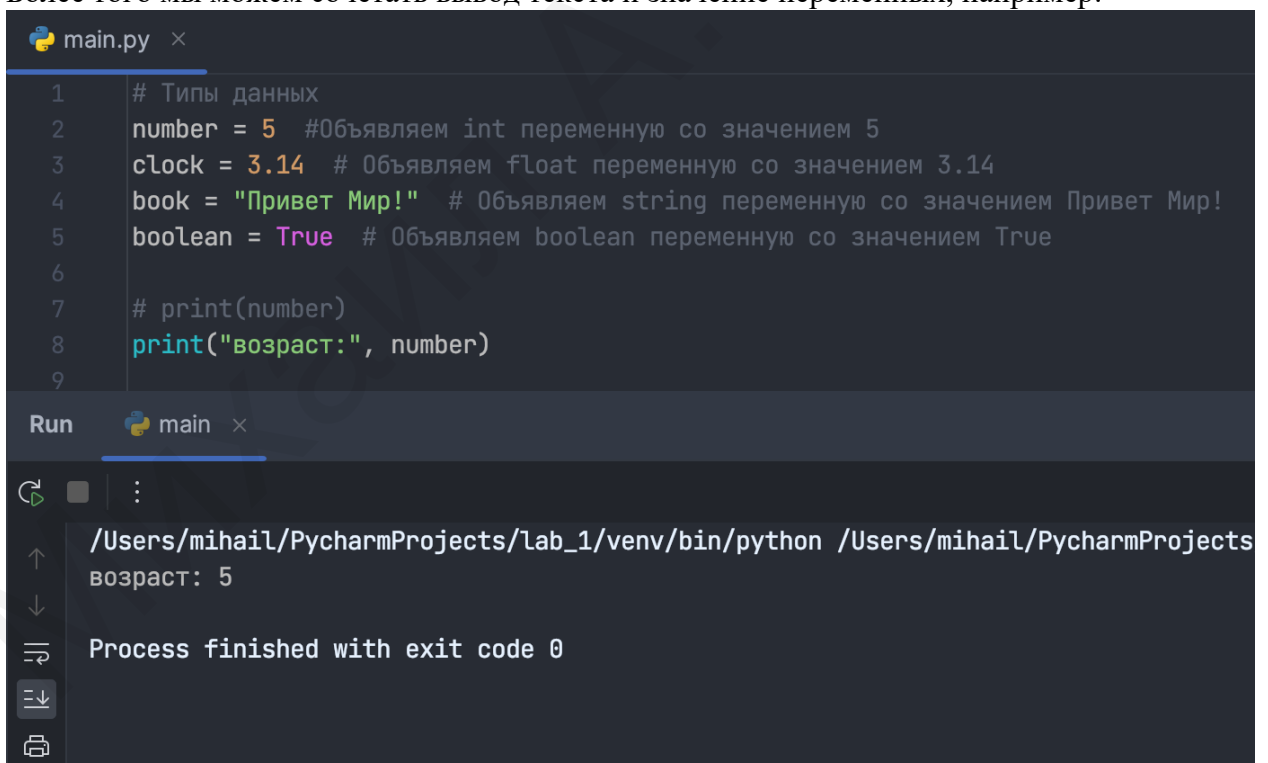
Run main x

/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects/5

Process finished with exit code 0

Рис. 2.10. Пример работы с *print()*

Более того мы можем сочетать вывод текста и значение переменных, например:



```
main.py x
1 # Типы данных
2 number = 5 #Объявляем int переменную со значением 5
3 clock = 3.14 # Объявляем float переменную со значением 3.14
4 book = "Привет Мир!" # Объявляем string переменную со значением Привет Мир!
5 boolean = True # Объявляем boolean переменную со значением True
6
7 # print(number)
8 print("возраст:", number)
9
```

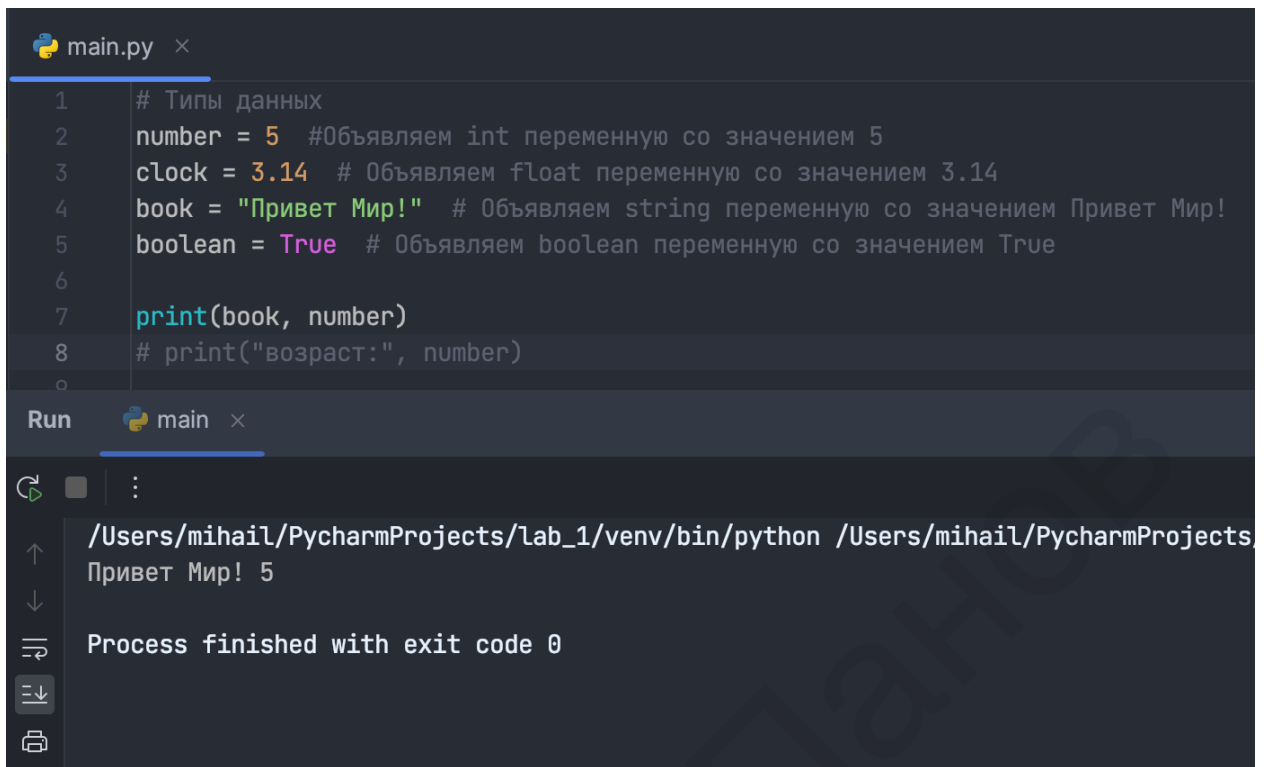
Run main x

/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects/возраст: 5

Process finished with exit code 0

Рис. 20 Пример работы с *print()* и выводом значения переменной

Или вывести переменные через запятую. Например:



```
main.py x
1 # Типы данных
2 number = 5 #Объявляем int переменную со значением 5
3 clock = 3.14 # Объявляем float переменную со значением 3.14
4 book = "Привет Мир!" # Объявляем string переменную со значением Привет Мир!
5 boolean = True # Объявляем boolean переменную со значением True
6
7 print(book, number)
8 # print("возраст:", number)
9

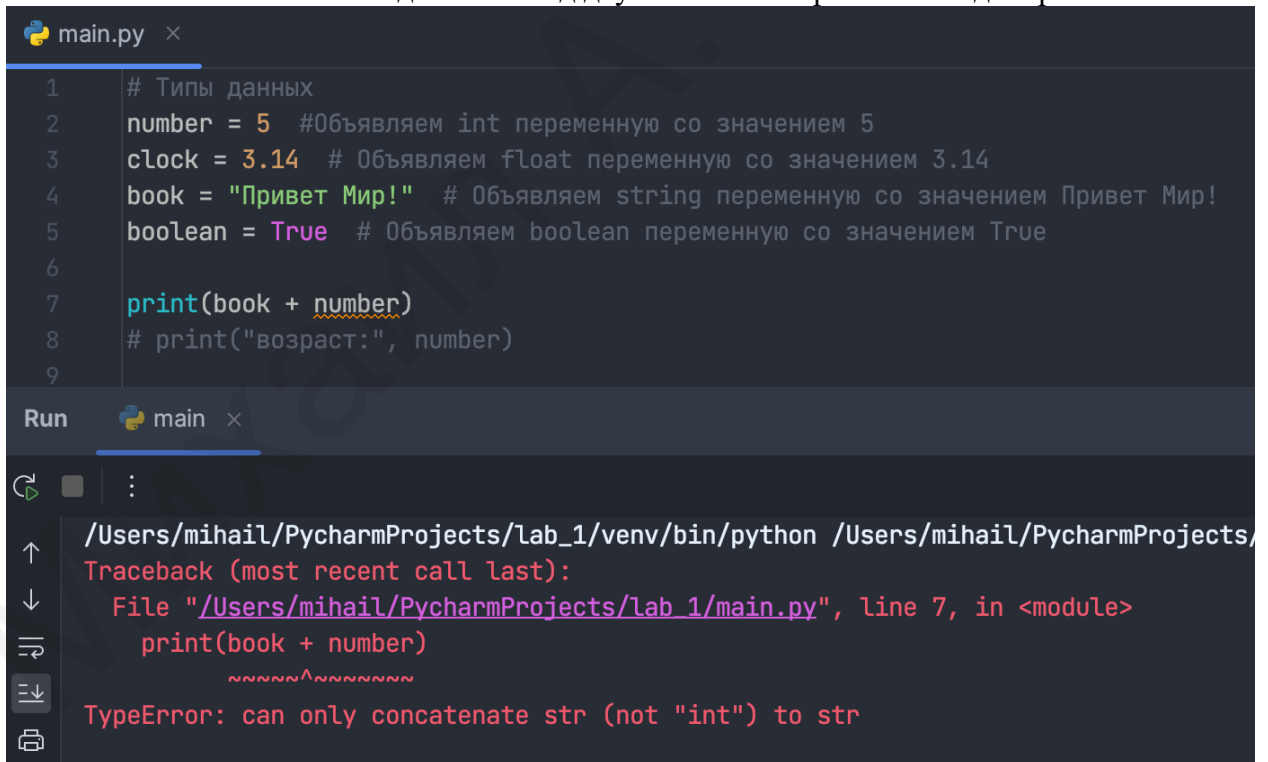
Run main x

/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects/
Привет Мир! 5

Process finished with exit code 0
```

Рис. 2.11. Пример работы с `print()` и выводом значения переменной через запятую

Но мы не можем выполнять действия над двумя типами переменных одновременно.



```
main.py x
1 # Типы данных
2 number = 5 #Объявляем int переменную со значением 5
3 clock = 3.14 # Объявляем float переменную со значением 3.14
4 book = "Привет Мир!" # Объявляем string переменную со значением Привет Мир!
5 boolean = True # Объявляем boolean переменную со значением True
6
7 print(book + number)
8 # print("возраст:", number)
9

Run main x

/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects/
Traceback (most recent call last):
  File "/Users/mihail/PycharmProjects/lab_1/main.py", line 7, in <module>
    print(book + number)
          ~~~~~^~~~~~
TypeError: can only concatenate str (not "int") to str
```

Рис. 2.12. Пример работы с `print()` и выводом значения переменной через сложение

В консоли мы получим ошибку «**TypeError: can only concatenate str (not "int") to str**» - можно выполнить конкатенацию строки к строке.

Эту ситуацию можно решить через конвертацию (приведение) переменных.

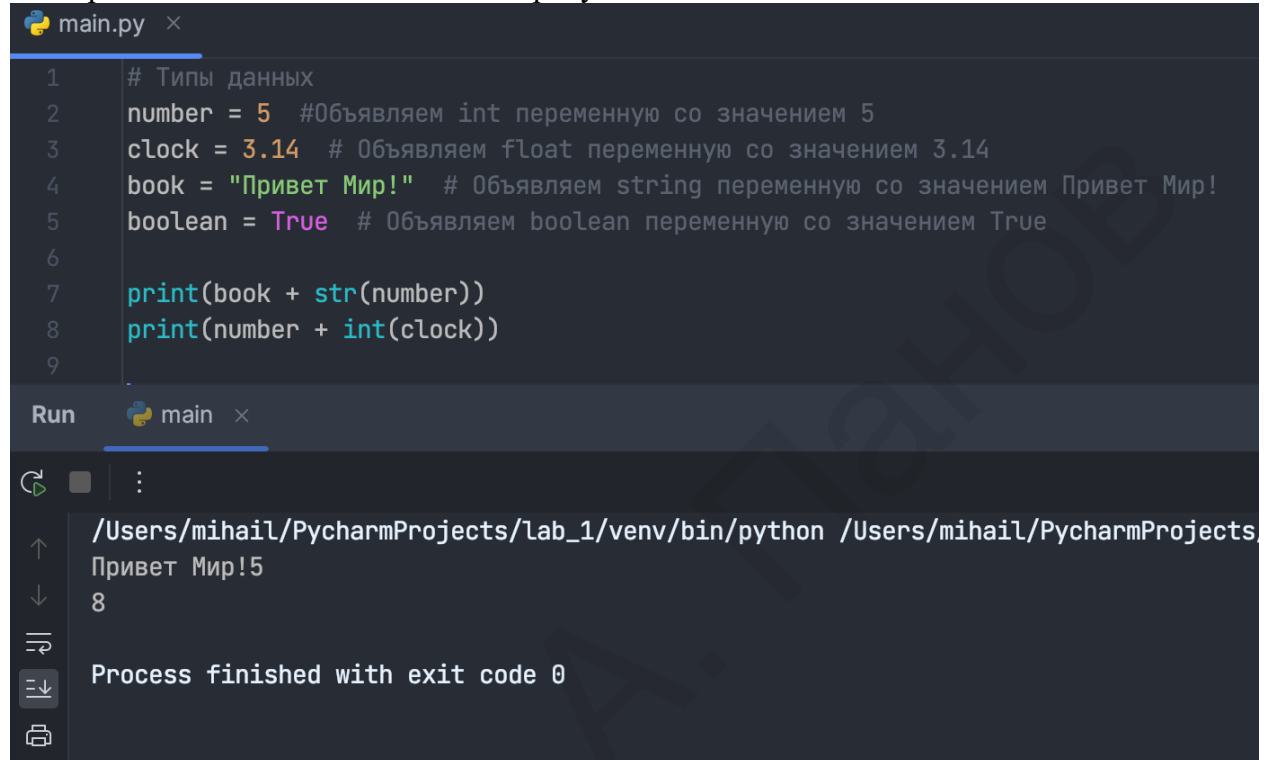
В Python можно производить конвертацию (приведение) переменных к различным типам данных с помощью встроенных функций:

Например:

int(x) - преобразует значение *x* в целочисленный тип *int*. Если значение не может быть преобразовано в целое число, то будет вызвано исключение **ValueError**.

или

str(x) - преобразует значение *x* в строковый тип *str*. Эта функция может использоваться для конвертации любого типа данных в строку.



```
main.py x
1  # Типы данных
2  number = 5 #Объявляем int переменную со значением 5
3  clock = 3.14 # Объявляем float переменную со значением 3.14
4  book = "Привет Мир!" # Объявляем string переменную со значением Привет Мир!
5  boolean = True # Объявляем boolean переменную со значением True
6
7  print(book + str(number))
8  print(number + int(clock))
9
Run main x
/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects
Привет Мир!5
8
Process finished with exit code 0
```

Рис. 2.13. Пример работы с *print()* и выводом значения через конвертацию (приведение) переменных

В Python можно производить конвертацию (приведение) переменных к различным типам данных также с помощью других встроенных функций:

float(x) - преобразует значение *x* в тип с плавающей точкой *float*. Если значение не может быть преобразовано в число, то будет вызвано исключение *ValueError*.

bool(x) - преобразует значение *x* в булевый тип *bool*. Любое значение, которое рассматривается как "ложное" (например, пустая строка, ноль, пустой список или *NoneType*), будет преобразовано в *False*. Все остальные значения будут преобразованы в *True*.

list(x) - преобразует значение *x* в список типа *list*. Если *x* является строкой, то каждый символ строки будет добавлен в список как отдельный элемент.

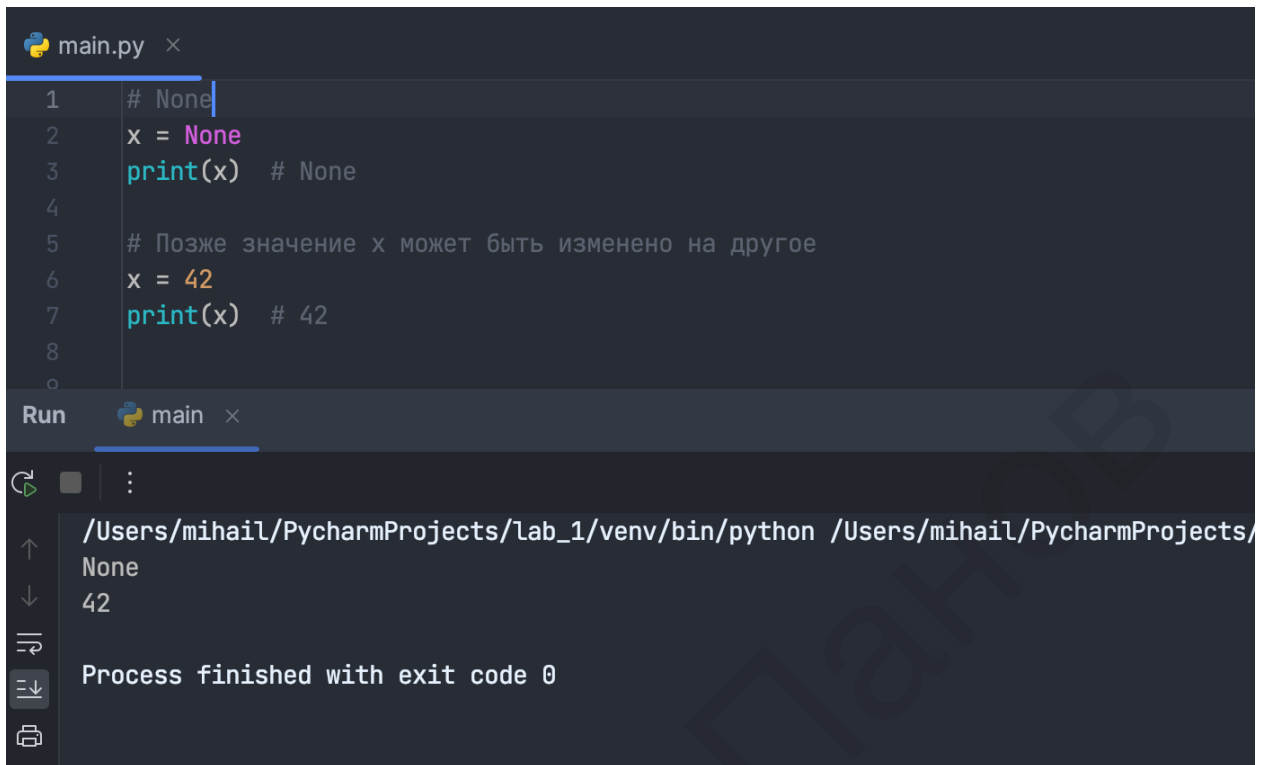
tuple(x) - преобразует значение *x* в кортеж типа *tuple*.

set(x) - преобразует значение *x* в множество типа *set*.

dict(x) - преобразует значение *x* в словарь типа *dict*. При этом *x* должен быть итерируемым объектом, содержащим пары ключ-значение.

2.4. *NoneType* объекты Python при определении переменных

None в Python можно использовать при определении переменных для задания начального значения, которое позже может быть изменено на другое значение. Такой подход может быть полезен, когда вы не знаете, какое значение должно быть присвоено переменной в начале программы или если значение переменной может быть задано позже в программе. Также *NoneType* удобно использовать в начальных сравнениях в программе.



```
main.py x
1 # None
2 x = None
3 print(x) # None
4
5 # Позже значение x может быть изменено на другое
6 x = 42
7 print(x) # 42
8
9
Run main x
/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects/
None
42
Process finished with exit code 0
```

Рис. 2.14. Пример работы с *None*

2.5. F-строки в Python

F-строки (formatted string literals) в Python позволяют создавать строки, которые могут включать значения переменных и выражений, которые вычисляются во время выполнения программы. Для создания F-строки используется префикс `f` перед строкой, а значения переменных и выражений заключаются в фигурные скобки `{}`.



```
main.py x
1 # F строки
2 name = "Михаил"
3 age = 45
4 print(f"Мое имя {name} и мне {age} лет.")
5
6 x = 5
7 y = 10
8 print(f"Сумма чисел {x} и {y} будет равна {x+y}.")
9
10 # Можно использовать любые выражения внутри фигурных скобок
11 z = 3
12 print(f"{x} + {y} * {z} = {x + y * z}") # 5 + 10 * 3 = 35
13
```

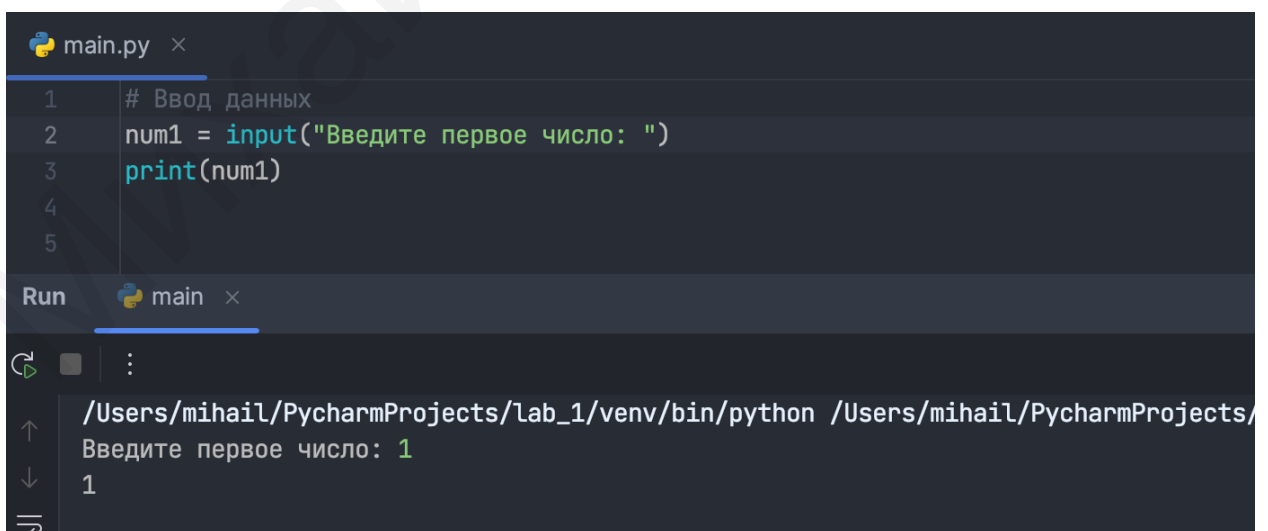
Run main x

:/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects/
Мое имя Михаил и мне 45 лет.
Сумма чисел 5 и 10 будет 15.
5 + 10 * 3 = 35
Process finished with exit code 0

Рис. 2.15. Пример работы с F-строками

2.6. Ввод данных через консоль

Для ввода данных через консоль используется функция `input()`. Но без записи ее через переменную она не имеет никакого смысла. Для того что бы на работать дальше с информацией, которую мы получили через консоль необходимо выполнить следующее:



```
main.py x
1 # Ввод данных
2 num1 = input("Введите первое число: ")
3 print(num1)
4
5
```

Run main x

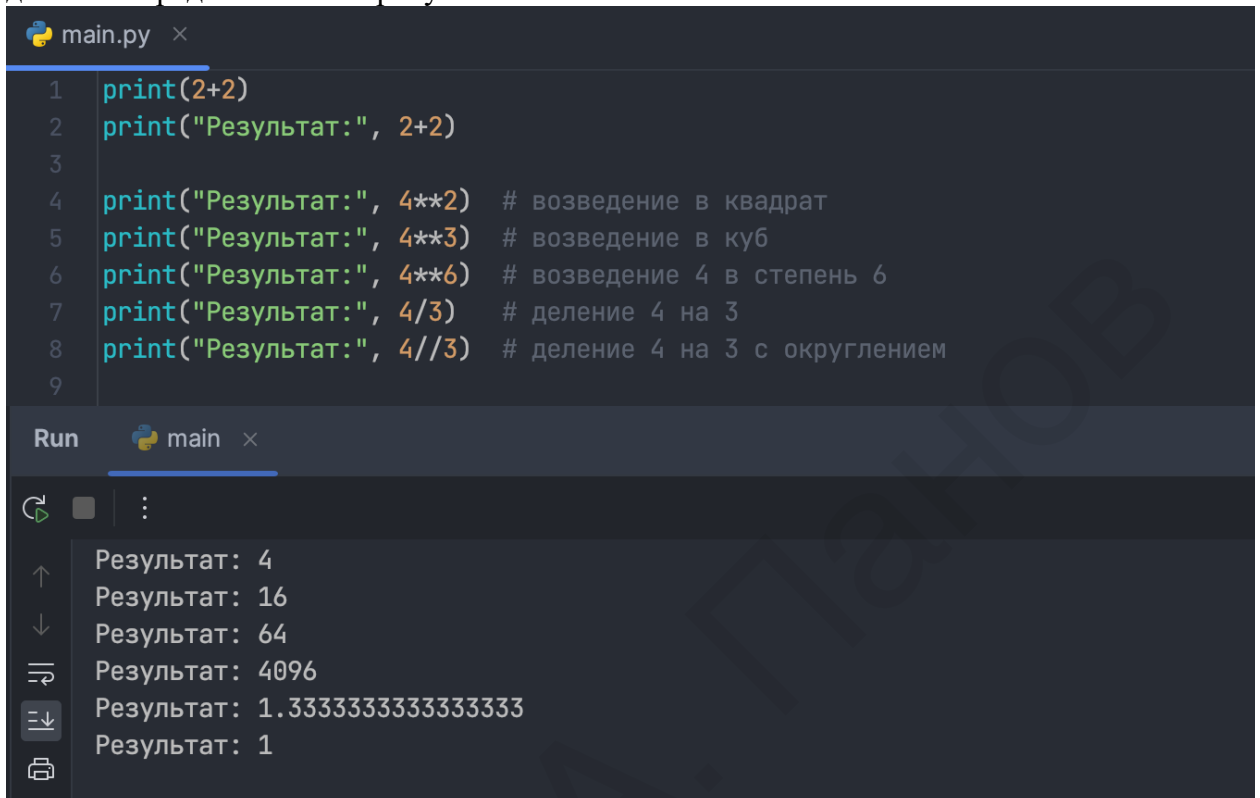
:/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects/
Введите первое число: 1
1

Рис. 2.16. Пример работы с `input()`

При использовании `input()` можно указывать какой тип значения он может принимать, например `int(input())`, тогда на ввод будут поступать только значения `int`.

2.7. Математические операции

В функции `print()` помимо вывода текстовой информации и значения переменных, можно осуществлять математические действия. Принцип работы и примеры некоторых действий представлены на рисунке 26.



```
main.py ×
1 print(2+2)
2 print("Результат:", 2+2)
3
4 print("Результат:", 4**2) # возведение в квадрат
5 print("Результат:", 4**3) # возведение в куб
6 print("Результат:", 4**6) # возведение 4 в степень 6
7 print("Результат:", 4/3)  # деление 4 на 3
8 print("Результат:", 4//3) # деление 4 на 3 с округлением
9

Run main ×

↑
↓
↺
↻
↓
📄

Результат: 4
Результат: 16
Результат: 64
Результат: 4096
Результат: 1.3333333333333333
Результат: 1
```

Рис. 2.17. Пример математических функций

Более подробное различные математические операции мы рассмотрим в наших следующих уроках.