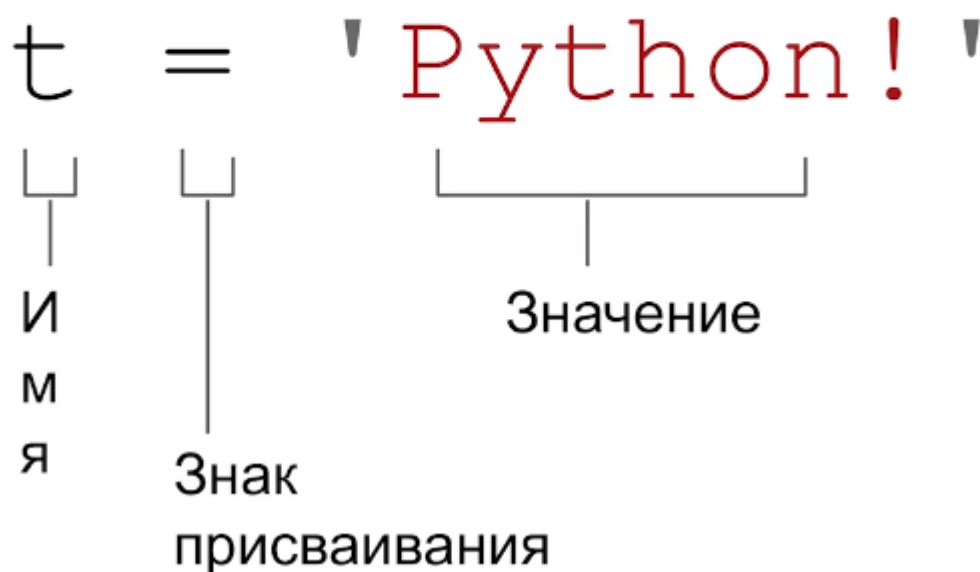


## Самое важное

Создание переменных

# Инициализация переменной



### \*Дополнительная информация

На самом деле переменные — это ссылки на объект, а не сам объект. Объект (например, строка или число) создаётся и хранится в памяти компьютера, а переменная лишь ссылается на место в компьютере, где хранятся эти данные.

### Нейминг (названия для переменных)

Следуйте паре простых правил, чтобы ваш код подходил под мировые стандарты стиля кода:

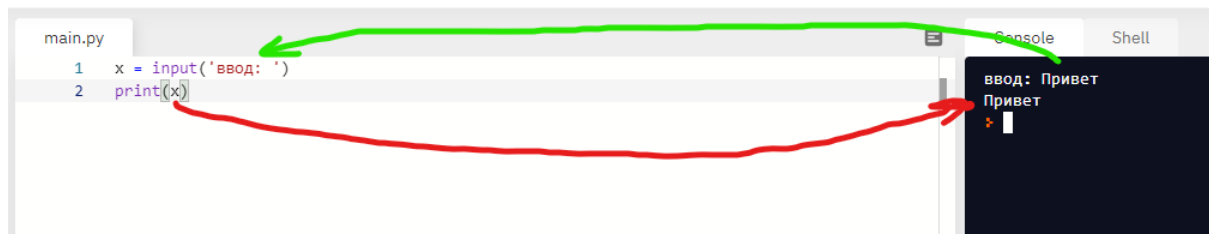
1. используйте только латинские символы;
2. не используйте спец символы в названиях переменных;
3. не используйте пробелы в названиях, вместо них используйте нижнее подчёркивание.

### Оператор ввода `input`

`Input` — функция, которая позволяет нам запрашивать ввод из консоли.

- Если мы напишем:  
`x = input('ввод :')`  
и выполним код, то в консоли появится надпись «ввод», и будет ожидать ввод данных.
- Если мы что-либо введем и нажмем enter — этот ввод сперва превратится в строку, а затем будет записан в переменную, которую мы использовали.

Пример:



```
main.py
1 x = input('ввод :')
2 print(x)
```

Console Shell

ввод: Привет  
Привет

То, что мы ввели вернулось результатом выполнения функции и было записано в переменную «x».

После этого мы распечатали переменную «x», чтобы убедиться, что всё сработало так, как мы задумали. Советуем выполнять такие проверки.

### Обратите внимание

Мы передали в функцию `input` параметр-строку «ввод :» — эта строка является приглашением ко вводу и нужна для того, чтобы подсказать пользователю, какой ввод вы от него ожидаете.

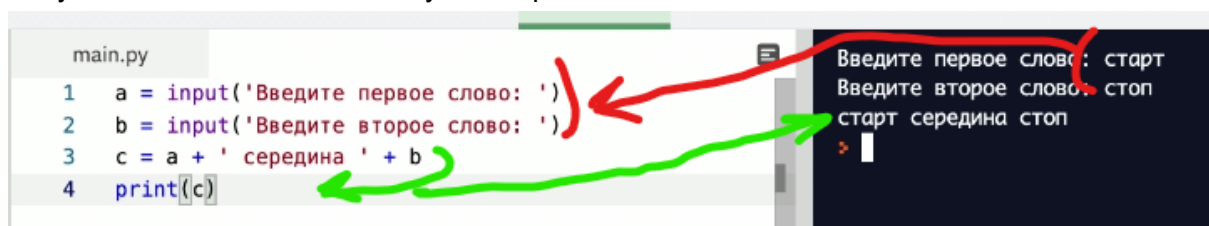
Например, это может быть запрос логина/пароля или даты рождения (вводы могут быть разными), поэтому важно писать понятное сообщение, чтобы пользователь понимал, что ему нужно ввести.

При этом важно понимать, что `input` (в отличие от `print`) принимает только одну единственную строку. В `print` же можно передать множество объектов, и он сам составит из них единую строку (добавив при этом свой стандартный разделитель-пробел между элементами), которую отобразит в консоли.

### Конкатенация строк

Это «склеивание» строк между собой при помощи суммирования. Приём часто используется для подстановки каких-либо изменяемых значений.

Например, мы можем запросить у пользователя какой-либо ввод, затем добавить к нему что-то своё и вывести ему это обратно на консоль:



```
main.py
1 a = input('Введите первое слово: ')
2 b = input('Введите второе слово: ')
3 c = a + ' середина ' + b
4 print(c)
```

Введите первое слово: старт  
Введите второе слово: стоп  
старт середина стоп

В этом примере мы:

- 1) запросили два ввода;
- 2) записали их в переменные a,b;
- 3) составили из этих переменных и своей строки «середина» строку;
- 4) вывели эту строку обратно на консоль через print.

Не забывайте, что пробел — такой же символ, как и все остальные. Если он вам нужен — добавляйте его, как показано в примере выше, иначе ваши строки склеятся без разделителя, а сообщение превратится в «кашу» из букв.

### Правильный нейминг

- Используйте только латинские слова.
- Старайтесь придумывать такие названия, которые будут отражать содержимое переменной.

Например, если вы спрашиваете у пользователя возраст — можете назвать переменную age:

```
age = input('введите ваш возраст: ')
```

Тогда дальше в коде будет ясно, что переменная относится к возрасту.

### Множественное присваивание

Если нужно присвоить несколько схожих объектов разным переменным, мы можем использовать множественное присваивание:

```
a, b = 'Привет', 'Мир'
```

Слева через запятую мы перечисляем переменные, а справа через запятую, в том же порядке, что и переменные — объекты. От порядка зависит, в какую переменную попадёт объект.

# Множественное присваивание

```
name1, name2, name3 = 'Вова', 'Петя', 'Лена'
```

Эквивалентом такой записи будет привычная нам запись:

```
a = 'Привет'
```

```
b = 'Мир'
```

Однако не стоит переусердствовать с этим приёмом. Помните, что длинные строки снижают читаемость кода. Кроме того, если элементы будут разного типа, будет сложно воспринимать такое присваивание.

Например:

```
first_car, important_documents, surnames = 'Toyota', ('doc1.txt', 'doc_2.xls', 'doc-3.docx'),  
['Петрова', 'Иванов', 'Смирнов']
```

Такой код читается на порядок сложнее, чем

```
first_car = 'Toyota'
```

```
important_documents = ('doc1.txt', 'doc_2.xls', 'doc-3.docx')
```

```
surnames = ['Петрова', 'Иванов', 'Смирнов']
```

### **Как не допустить ошибку**

Не пишите названия переменных в кавычках, если вы хотите использовать объект, на который ссылается переменная.

```
x = 5
```

`print('x')` — так Python не поймёт, что вы хотите распечатать число 5. Программа подумает, что вы хотите распечатать строку «x» и распечатает её.

`print(x)` — правильный способ обращения к переменной.

Не забывайте, что Python — регистрозависимый язык, поэтому большие и маленькие буквы он будет воспринимать как совершенно разные символы:

```
x = 5
```

`print(X)` — приведёт к ошибке, т. к. «X» создан при помощи маленького символа «x», а в данном обращении использован большой символ «X».