Команда input()

1. Команда input()

Ранее наши программы выводили на экран только фразы, известные в момент написания кода. Но программы могут работать и с данными, которые станут известны только во время выполнения: например, их будет вводить пользователь с клавиатуры. Реализовать это можно так:

In [3]:
```
print('Как тебя зовут?')
name = input()
print('Привет,', name)
```

```
Как тебя зовут?
Anna
Привет, Anna
```

Команда input() всегда записываеться с круглыми скобками. Работает так: когда программа доходит до места, где есть input(), она ждет, пока пользователь введет строку с клавиатуры (ввод завершается нажатием клавиши Enter). Введенная строка подставляется на место input(). Т.е. input() получает от пользователя какие-то данные и в место вызова подставляет строковое значение, в нашем случае записывает его в качестве значения переменной **name**.

Примечание. Текст, который мы печатаем перед считыванием данных в некоторых случаях нужен чтобы пользователь, который будет вводить данные понимал, что именно от него требуется. Поскольку это достаточно распространённый сценарий, то в языке Python можно выводить текст, передавая его в качестве параметра в команду input(). Предыдущий код можно переписать так:

In [2]:
```
name = input('Как тебя зовут? ')
print('Привет,', name)
```

```
Как тебя зовут? Anna
Привет, Anna
```

2. Переменные

name - это имя переменной. Оно может состоять из латинских букв, цифр и символа подчеркивания. Имя не может начинаться с цифры. Имя переменной должно отражать ее назначение.

PEP8: Для именования переменных принято использовать стиль lower_case_with_underscores (слова из маленьких букв с подчеркиваниями).

3. Список зарезервированных слов

В каждом языке есть зарезервированные слова. Такие слова имеют специальное значение, и поэтому запрещены для использования в качестве имён переменных. Вот список зарезервированных слов для Python:

False, class, finally, is, return, None, continue, for, lambda, try, True, def, from, nonlocal, whileand, del, global, not, with, as, elif, if, or, yield, assert, else, import, pass, break, except, in, raise.

4. Значение переменной

Значение переменной — то, что сохраняет в себе переменная.

Знак «=» обозначает команду под названием «оператор присваивания». Оператор присваивания присваивает значение, которое находится справа от знака равно, переменной, которая находится слева от знака равно.

В нашем случае это — строка. То есть переменная сохраняет в себе строковое значение. Говорят, что переменная строкового типа.

PEP8: Всегда окружайте оператор присваивания одним пробелом с каждой стороны:

5. Как устроены переменные внутри

Каждый элемент данных в Python является объектом определенного типа или класса. Например:

In [1]:
```
name = 'Anna'
print(type(name))    # строковая переменная
print(id(name))      # адрес объекта

a = 123
print(type(a))       # целочисленная переменная
print(id(a))         # адрес объекта
```

```
<class 'str'>
86422000
<class 'int'>
8790909825232
```

Когда, в процессе выполнения программного кода, появляется новое значение, интерпретатор выделяет для него область памяти – то есть создаёт объект определенного типа (число, строка и т.д.). После этого Python записывает в свой внутренний список адрес этого объекта.

Но как теперь получить доступ к созданному объекту? Для этого и существуют переменные – они дают возможность удобно работать с объектами используя имена вместо адресов. Т.е. переменная в Python – это просто имя, прикрепленное к определенному объекту

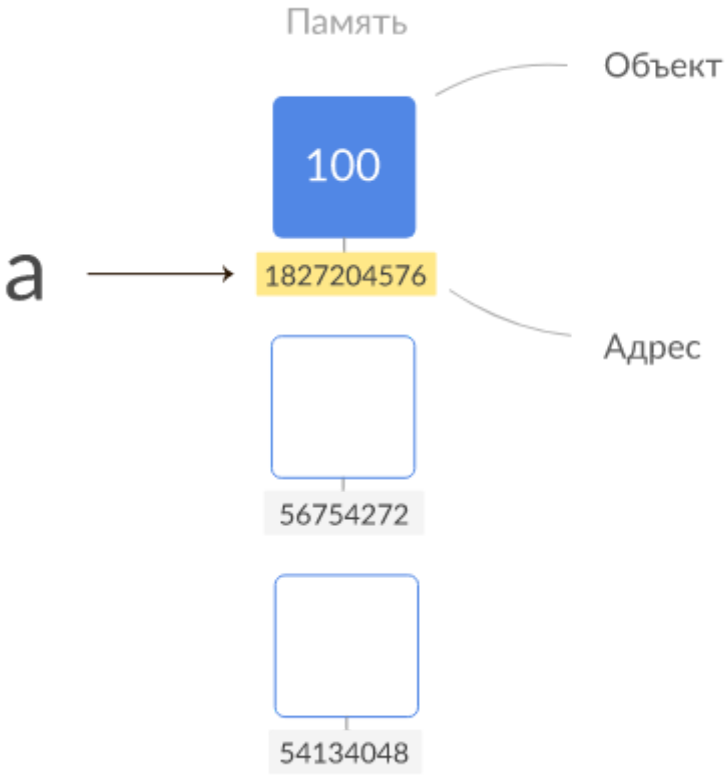
Принцип работы переменных

Для понимания работы переменных, разберемся, что происходит, когда мы создаем новую переменную и присваиваем ей значение?

In [ ]:
```
a = 100
```

В данном примере происходит следующее:

- создается объект типа int со значение 100;
- создается переменная a;
- в переменной a сохранится адрес (ссылка) на объект;



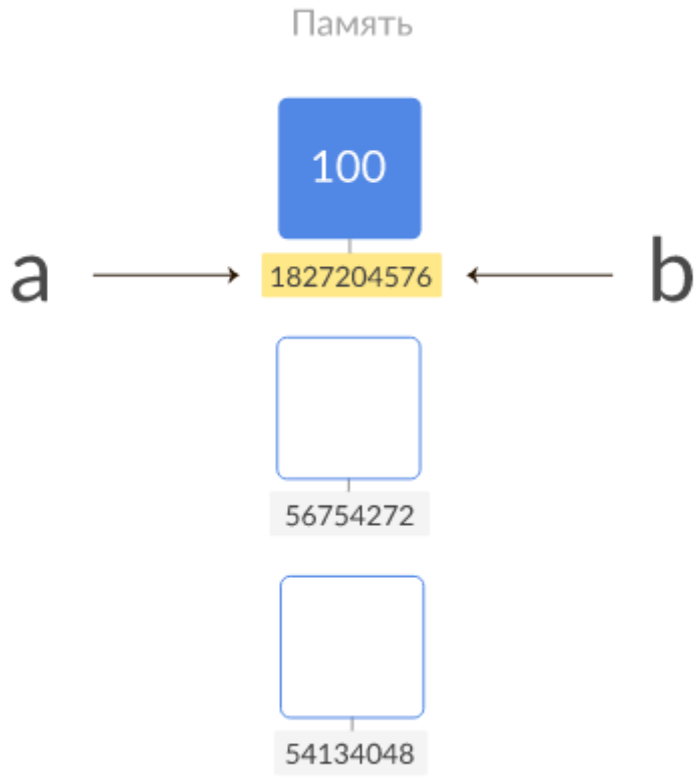
Теперь посмотрим, что произойдет, если одной переменой присвоить другую переменную:

In [4]:
```
b = a

print(id(a))
print(id(b))
```

```
8790909825232
8790909825232
```

В данном примере Python не создает новый объект – он просто создает переменную, которая ссылается на тот же объект, что и переменная a.



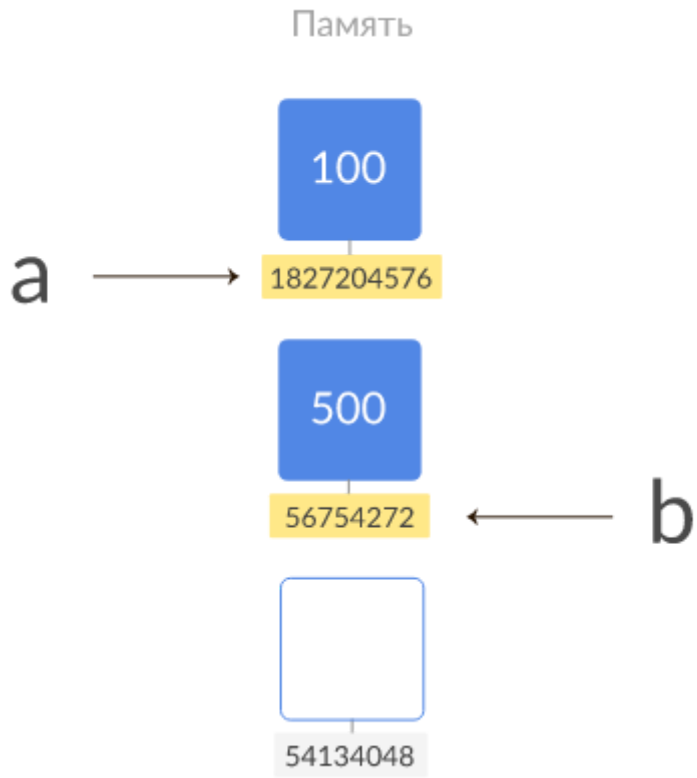
Предположим, что в какой-то момент мы захотели поменять значение переменной b.

In [5]:
```
b = 500

print(id(a))
print(id(b))
```

```
8790909825232
86663408
```

В данном примере Python создал новый объект типа int, и теперь переменная b ссылается на новый объект.



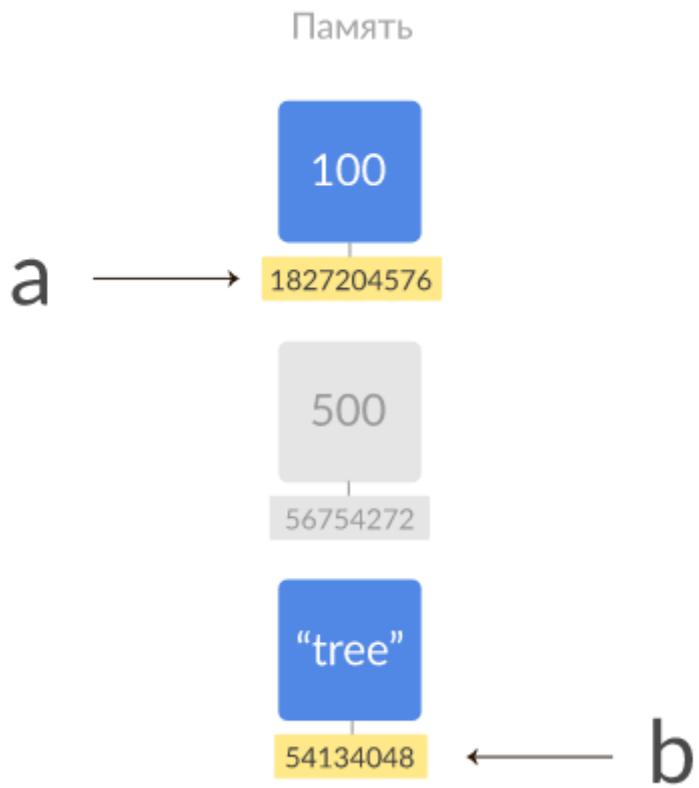
Рассмотрим еще один пример:

In [6]:
```
b = "tree"

print(id(b))
```

```
49806768
```

В этом примере создается новый объект типа str, и переменная b ссылается на новый объект.



b = "tree", переменная b теперь ссылается на новый объект строкового типа

На объект типа int со значением 500 больше никто не ссылается. Следовательно, он больше не доступен и будет удален сборщиком мусора (тем самым освободим немного памяти).

6. Идентификатор объекта (Object Identity)

Идентификатор объекта – это адрес объекта в памяти.

В примерах выше мы вызывали функцию **id()**. Эта функция возвращает число, которое является неизменным и уникальным для каждого объекта на протяжении его жизненного периода:

In [7]:
```
a = b = 1

print(id(a))

print(id(b))

print(id(1))
```

```
8790909821328
8790909821328
8790909821328
```

Видно, что объект здесь всего один. А **a** и **b** – по-разному названные переменные, которые на него ссылаются. Проверить равенство идентификаторов можно с помощью оператора **is**

In [8]:
```
print(a is b)
```

```
True
```