

Python. Хранение ДАННЫХ В ПАМЯТИ

А что там внутри?

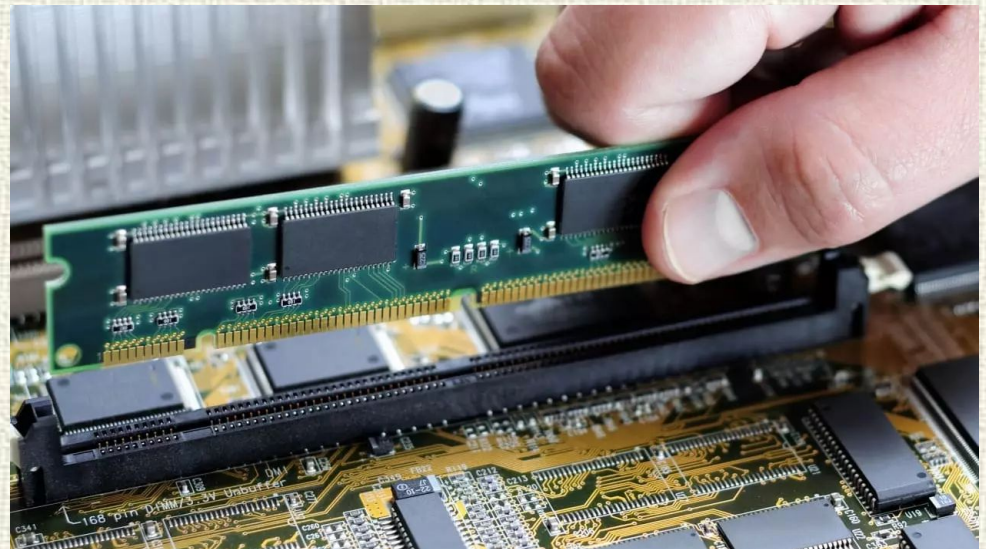
Если все операции с данными выполняет процессор, используя оперативную память, то что произойдет, когда в программе инициализируется переменная?

$$a = 5$$

Переменная – поименованная область памяти, хранящая данные в ходе выполнения программы.

Не питон

a	
адрес	#10055680
значение	5



А дальше?

$a = 5$

a	
адрес	#10055680
значение	5

$a = a + 1$

a	
адрес	#10055680
значение	6

$b = a$

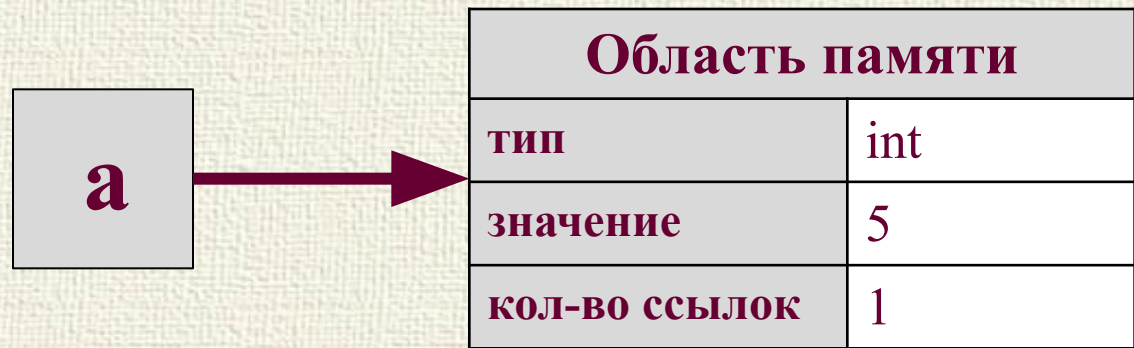
a	
адрес	#10055680
значение	6

b	
адрес	#11199751
значение	6

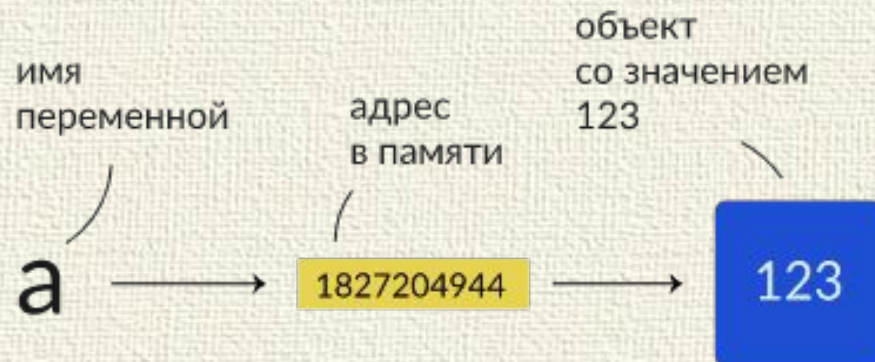
А в питоне?

В других языках программирования переменная обозначает ячейку памяти с хранящимся в ней значением. В Python переменная является ссылкой на область памяти.

`a = 5`



Ссылка - это адрес объекта в памяти

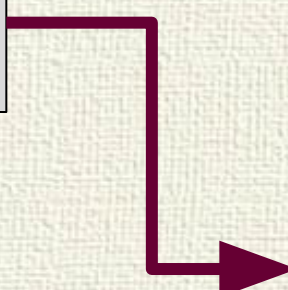


А дальше?

$a = a + 1$



Область памяти	
тип	int
значение	5
кол-во ссылок	0



A thick, dark red arrow originates from the right side of the 'a' box, moves right, then turns down, and finally turns right again to point at the top-left corner of the second memory block table.

Область памяти	
тип	int
значение	6
кол-во ссылок	1

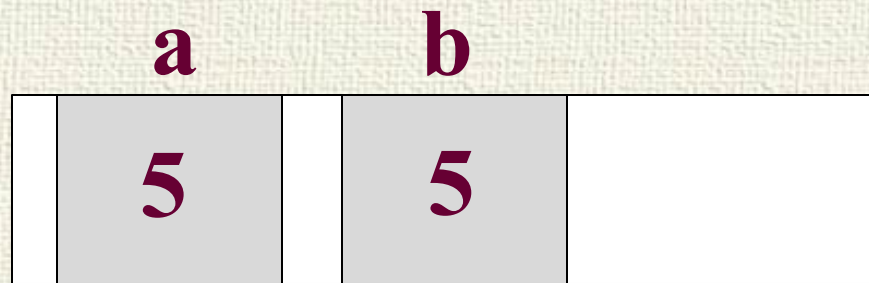
При изменении значения переменной будет выделяться новый участок памяти для новых данных, а предыдущий участок будет очищен.

Атомарные и ссылочные типы

Атомарные - при присваивании одного объекта другому копируется их значение.

$a=5$

$b=a$



Память

Ссылочные - при присваивании копируется только указатель на объект, таким образом, обе переменные после присваивания используют одно и то же значение.

+ экономия памяти;

- при изменении исходной переменной меняются все, ссылающиеся на неё.

ССЫЛОЧНЫЕ

$b = a$



А можно новый объект?

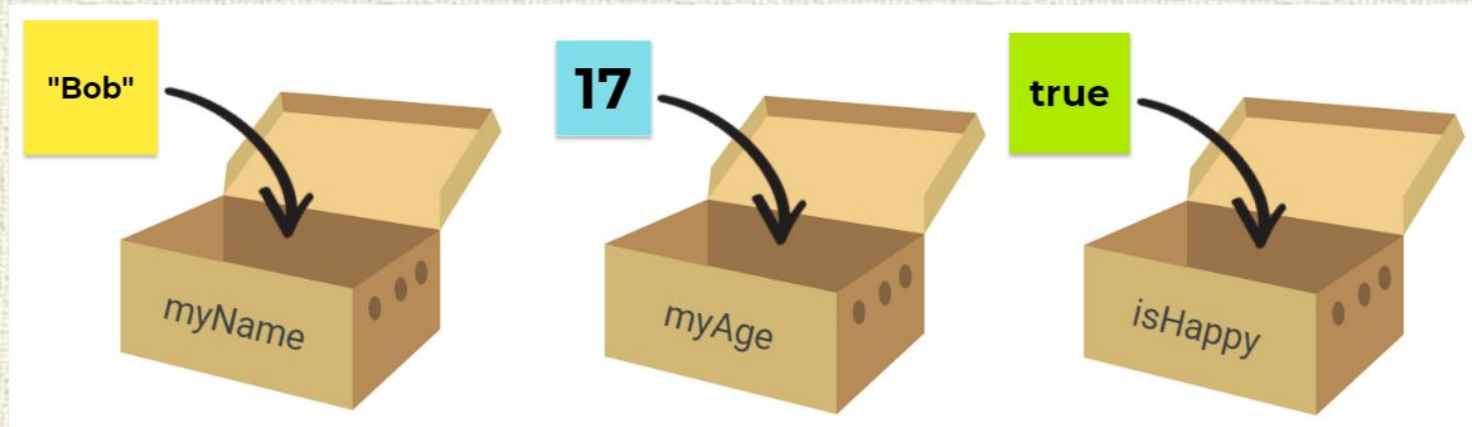
Если при присваивании переменной ссылочного типа необходимо, чтобы обе переменные хранились в разных участках памяти, то необходимо явно создавать новый объект в памяти.

Создание нового объекта в памяти:

```
L = [1,2,3]
```

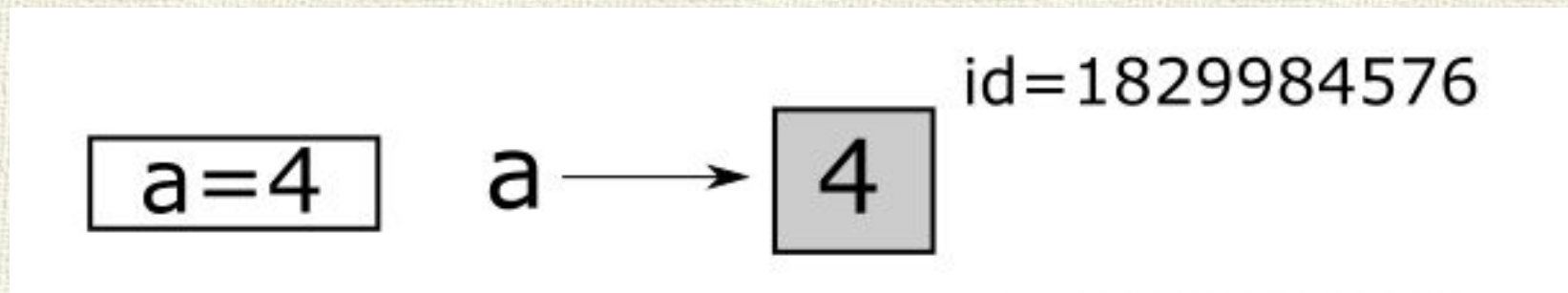
```
L2 = list(L)
```

Также можно использовать функцию `copy()` и полный срез `[:]` объекта.



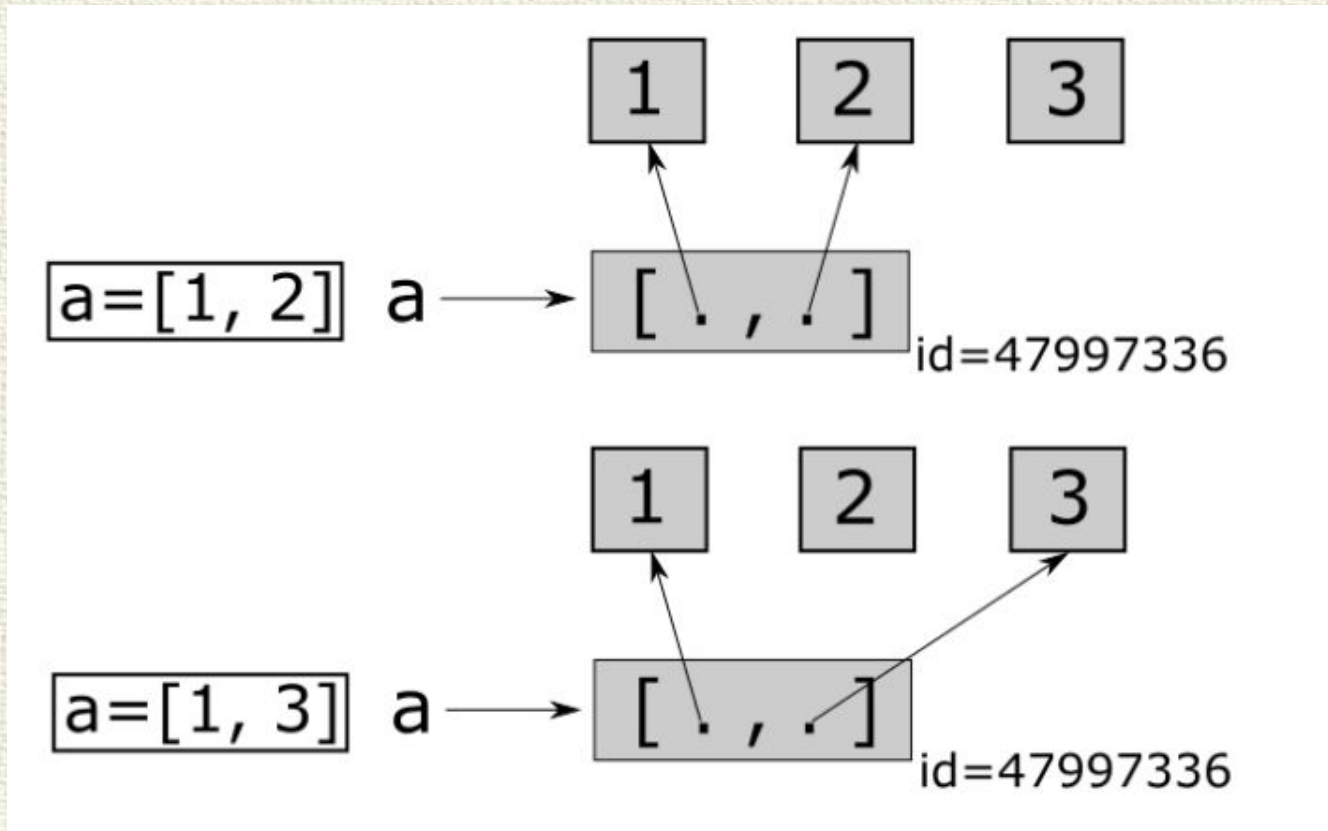
Изменяемые и неизменяемые типы

При создании переменной в памяти сначала создается объект, который имеет уникальный идентификатор, тип и значение, после этого переменная может ссылаться на созданный объект.



Неизменяемость типа данных означает, что созданный объект больше не изменится и мы можем только поменять ссылку на участок в памяти.

Изменяемые и неизменяемые типы



Если тип данных изменяемый, то можно менять значение объекта. Т.к. в переменной хранятся ссылки на объекты со значениями, а не непосредственно сами значения.

Кто есть кто?

Оператор “==” проверяет равенство значений объектов.

Оператор is проверяет идентичность самих объектов.

```
# ссылочный объект типа list
l1 = [1, 2, 3] # создадим объект списка l1
l2 = [1, 2, 3] # создадим объект списка l2
l3 = l1 # присвоим ссылку на объект l1 переменной l3
print(l1 == l2) # True -- списки равны
print(l1 == l3) # True -- списки равны
print(l1 is l2) # False -- ссылаются на два разных объекта
print(l1 is l3) # True -- содержат ссылки на один и тот же объект
```

Функция id() возвращает адрес ячейки памяти переменной

```
n = [5,6,10]
print(n)           # [5, 6, 10]
print(id(n))       # 140312184155336
```

Словари Python

Что это?

Словарь (хэш, ассоциативный массив) — изменяемая неупорядоченная последовательность данных разного типа в формате «ключ-значение».

`D = {}` #пустой словарь

`D = dict()` #преобразовать в словарь (или создать пустой)

`D = dict(Ivan="manager", Mark="worker")`

`D = {"A1": "123", "A2": "456"}` #словари со значениями

Генератор словаря

`D = {ключ: значение for параметр in диапазон}`

`D = {i: i**2 for i in range(5)}`

`#{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}`

Обработка словарей

```
D = {"Russia": "Moscow", "USA": "Washington"}
```

Доступ к значению происходит по ключу. Ключом могут быть только данные неизменяемых типов, а значениями – любых типов. Ключи должны быть уникальными, значения могут повторяться.

```
print(D['Russia']) #Moscow
```

Добавление элементов происходит при обращении к новому ключу. Если ключ существовал, его значение заменяется.

```
D["China"] = "Beijing"
```

```
# {"Russia": "Moscow", "China": "Beijing", "USA": "Washington"}
```

```
D["USA"] = "Moscow"
```

```
# {"Russia": "Moscow", "China": "Beijing", "USA": "Moscow"}
```

Обработка словарей

Удаление элементов производится оператором `del`
`del D["USA"] # {"Russia": "Moscow", "China": "Beijing"}`

Оператор `in` проверяет наличие ключа (но не значения)

```
"Russia" in D # True
```

```
"France" in D # False
```

При переборе словаря циклом `for` счетчик цикла также принимает значения ключа. Чтобы получить значение нужно обратиться к нему используя ключ.

```
for i in D:
```

```
    print(i, D[i])
```

```
# Russia Moscow
```

```
# China Beijing
```

Методы словарей

Данные методы позволяют получать отдельно ключи и отдельно значения. Они возвращают значения в виде особого типа данных, но могут быть преобразованы в список, множество и т.п.

Метод	Назначение	Пример
<code>D.values()</code>	Возвращает значения	<code>print(D.values())</code>
<code>D.keys()</code>	Возвращает ключи	<code>print(D.keys())</code>
<code>D.items()</code>	Возвращает пары ключ-значение	<code>x=D.items()</code>

Метод `items()` возвращает пару ключ-значение в виде кортежа.

```
print(D.items())
```

```
# [("Russia", "Moscow"), ("China", "Beijing")]
```


Методы словарей

Метод	Назначение	Пример
<code>D.get(key)</code>	Возвращает значение ключа <code>key</code> . Аналогично обращению <code>D[key]</code>	<code>print(D.get("Russia"))</code> <code>)</code>
<code>D.update(d2)</code>	Добавляет в словарь <code>D</code> словарь <code>d2</code> (существующие ключи перезаписываются)	<code>D.update({"Belarus": "Minsk"})</code>
<code>D.pop()</code>	Удаляет ключ и возвращает его значение	<code>x=D.pop()</code>
<code>D.popitem()</code>	Удаляет и возвращает пару ключ-значение	<code>x=D.popitem()</code>
<code>D.clear()</code>	Очищает словарь	<code>D.clear()</code>
<code>D.copy()</code>	Копирует словарь	<code>D2=D.copy()</code>

```
my_dict={'a':42 , 'b':42}
```

```
my_dict.update({'a': 77, 'c': 77}) #{'a': 77, 'b ': 42, 'c': 77}
```

Методы словарей

Метод	Назначение
<code>d.setdefault(key[,default])</code>	Возвращает значение для ключа <code>key</code> , если ключа нет, создает его со значением <code>default</code>
<code>dict.fromkeys(seq[, value])</code>	Создает словарь с ключами из списка <code>seq</code> и значением <code>value</code> (по умолчанию <code>None</code>)

```
my_dict.setdefault('c') #{'a': 123, 'b': 42, 'c': None}
my_dict.setdefault('c ', 43) #{'a': 123, 'b': 42, 'c': 43}
x= my_dict.setdefault('c') #43
```

```
dict.fromkeys(['a', 'b']) #{'a': None, 'b': None}
dict.fromkeys(['a', 'b'], 42) # {'a': 42, 'b': 42}
```

Словари

Можно хранить структурно связанные данные

```
man = {'name': "Serg",  
      'jobs': ['programmer', 'writer'],  
      'web': 'www.bestsite.org',  
      'home': {'city': 'Moscow', 'zip': 129000}}
```

Перебрать все ключи

```
for key in man:  
    print(key, man[key])
```

Изменить значения

```
man['name']='Mels'
```

Словари

```
def get_female_local():  
    print('Женщина')
```

```
def get_male_local():  
    print('Мужчина')
```

так как функции - тоже объекты, то создаем словарь привязки ключа к имени функции

```
gender_handlers = {  
    'male': get_male_local,  
    'female': get_female_local  
}
```

```
gender = 'male'
```

получаем по ключу нужную функцию и вызываем ее

```
gender_handlers[gender]()
```

Кортежи Python

Что это?

Кортеж — **неизменяемая** упорядоченная последовательность данных разного типа.

T=() #пустой кортеж

T=tuple() #преобразовать кортеж

T= (1,2,3) #кортежи со значениями

Важно!

T = ('s') #создается строка

T= ('s',) #создается кортеж

Также можно создавать кортеж генератором.

T=(i**2 for i in range(5)) #(0,1,4,9,16)

Зачем кортежи, когда есть списки?

Назначение:

- 1) Меньший размер объектов по сравнению со списками. Экономия ресурсов и времени.
- 2) Безопасность данных от случайных (что хорошо) и намеренных (что не так хорошо) изменений.
- 3) Возможность использовать кортежи, как ключи словарей.

```
rectangle_coords = {  
    (0, 0): 'левый нижний угол',  
    (1, 1): 'правый верхний угол',  
}  
point = (1, 1)  
print('Координаты ' + str(point) + ' это ' + str(rectangle_coords[point]))
```

Действия с кортежами

Все операции над списками, не изменяющие список (сложение, умножение на число, методы `index()` и `count()` и некоторые другие операции).

Нельзя изменять элементы - нельзя удалить элемент, но можно удалить кортеж целиком.

```
# index(x, [start [, end]]) вернуть положение первого элемента
# со значением x (при этом поиск ведется от индекса start до индекса
# end)
my_tuple = (1, 2, 3, 1, 2, 42, 4, 3, 2, 1)
index = my_tuple.index(42) # index = 5
index = my_tuple.index(42, 0, 4) # ValueError: 42 is not in list
index = my_tuple.index(42, 3, 7) # index = 5

# count(x) вернуть количество элементов со значением x
count = my_tuple.count(2) # count = 3
```