Изменяемые и Неизменяемые объекты, хранение объектов в памяти

1. Простые типы данных, такие как **int, float, str, bool** являются НЕИЗМЕНЯЕМЫМИ: они создаются в памяти и хранятся там в своей ячейке.  
  
**Пример:**  
***>>>* var1 = 1** # *создаем в памяти объект типа* ***int,*** *значение которого* ***1****, переменная* ***var1*** *является ссылкой на этот объект*

***>>>* id(var1)**

**10735336**  
***>>>* var2 = var1**  # *создаем* ***новую ссылку*** *на этот же объект – в переменной* ***var2***

***>>>* id(var2) #** *получаем адрес в памяти объекта*

**10735336 *#*** *как видите, адрес в памяти у* ***var1*** *и* ***var2*** *одинаков, так как это 2 ссылки на один и тот же объект*  
  
**>>> var1 = 2** # *Cоздаем в памяти* ***новый объект*** *типа* ***int*** *со значением* ***2,*** *переменная* ***var1*** *теперь уже ссылается на* ***новый объект***  
**>>> id(var1) #** *получаем адрес в памяти объекта*

**10735368 *#*** *теперь у* ***var1*** *уже другой адрес в памяти.*  
  
(Объекты “живут” в памяти до тех пор, пока на них есть хотя бы одна ссылка)  
 **2. ИЗМЕНЯЕМЫМИ** могут быть только Составные типы (**list, dict, set)**Составной тип можно представить как некий контейнер, хранящий какое-то количество объектов разных типов (“контейнер” может быть пуст)

**3. ИЗМЕНЯЕМЫЙ тип данных:** Объект такого тип можно Изменить, через его **методы**:  
Базовые операции изменения объекта:  
  
- ***Добавить элемент***  
- ***Заменить элемент***  
- ***Удалить элемент***  
  
**Пример:  
  
>>>** **my\_list = [1, 2, 3] #** *создаем объект списка, переменная* ***my\_list*** *– ссылка на этот объект*

**>>> my\_list**

**[1, 2, 3]**

**>>> id(my\_list) #** *получаем адрес в памяти созданного объекта списка*

**140161983860096**

**>>>** **my\_list.append(100) #** *добавляем новый элемент 100 в список*

**>>> my\_list**

**[1, 2, 3, 100]**

**>>> id(my\_list) #** *получаем адрес в памяти созданного объекта списка*

**140161983860096 #** *адрес в памяти не поменялся*

**>>> my\_list.remove(1) #** *удаляем элемент со значением 1 из списка*

**>>> my\_list**

**[2, 3, 100]**

**>>> id(my\_list) #** *получаем адрес в памяти созданного объекта списка*

**140161983860096 #** *адрес в памяти не поменялся* **4. Кортеж (tuple) – составной объект,** представляющий собой контейнер, содержащий другие объекты **(как и список).** Однако, **Кортеж** являтся НЕИЗМЕНЯЕМЫМ типом.

Базовые структуры данных

**List (Список)**  
  
1. **ИЗМЕНЯЕМЫЙ** тип данных  
2. Элементы добавляются в конец списка  
3. Элементы списка хранятся упорядочено – в порядке их добавления  
4. Может хранить разные типы данных, а также список может хранить другие списки  
5. Элементы в списке хранятся по индексу: поэтому списки поддерживат обращение по Индексу и Срезы (как и строки)  
6. Элементы в списке могут дублироваться  
  
Наиболее распространенная структура данных, когда нужно просто хранить упорядоченную последовательность объектов.

**Tuple (кортеж)**  
  
1. **НЕИЗМЕНЯЕМЫЙ** тип данных  
2. Кортежи похожи на Списки, за исключением того, что Кортежи НЕ поддерживают операции **Добавления/Вставки/Замены/Удаления** элементов.  
  
Когда нужно хранить упорядоченную последовательность объектов, без необходимости явно ее изменять – лучше использовать **Кортеж** вместо **Списка**, так как:  
  
- **Кортеж** занимает меньше места в памяти (даже при одинаковом количестве элементов)

**>>>** ***import sys*>>> *my\_list = list(range(0, 100))  
>>> my\_tuple =*** ***tuple(range(0, 100))***

***>>> sys.getsizeof(my\_list)  
856***

***>>> sys.getsizeof(my\_tuple)  
840***- Список можно случайно (по ошибке) изменить, и по итогу получим неожидаемое поведение.  
  
  
**Dict (Словарь)**  
  
1. **ИЗМЕНЯЕМЫЙ** тип данных  
2. Хранит пары “ключ-значение”, где каждое **Значение** связано с конкретным **Ключом**  
3. Элементы словаря хранятся в хэш-таблице, однако, начиная с версиии **Python 3.7** словари упорядочены по добавлению элементов.   
4. При добавлении элемента в словарь вычисляется **хэш** **ключа** (не Значения!). Объекты, которые нельзя хэшировать НЕ могут быть **ключами** словарей! **\***  
5. **Ключи** в словаре должны быть Уникальными, **Значения** – могут повторяться  
6. Словари не поддерживают обращение по Индексу и Срезы. Но поддерживают обращение по **ключу** (***получить/изменить/удалить*** элемент словаря)  
  
- Словари в Python используются повсеместно на уровне реализации самого языка Python:  
Глобальные и локальные переменные, поля объектов классов – для хранения этих данных используются словари.  
- Удобная структура данных, когда нужно хранить значения, ассоциированные с ключами (ключом)

**Set (Множество)**  
  
1. **ИЗМЕНЯЕМЫЙ** тип данных  
2. Элементы множества хранятся в хэш-таблице, поэтому они неупорядочены по добавлению   
3. При добавлении элемента в множество вычисляется **хэш** этого элемента – Объекты, которые нельзя хэшировать НЕ могут быть элементами множества! **\***  
4. Множества НЕ содержат дубликаты (повторяющиеся элементы)  
5. Множества НЕ поддерживают обращение по Индексу и Срезы  
6. Множества поддерживают математические операции такие как:  
  
***- объединение  
- пересечение  
- разность  
- симметричная разность***

Множества применяются тогда, когда:  
  
***- нужно хранить только уникальные элементы  
- быстрый доступ к любому элементу множества (за константное время, независимо от размера множества)  
- нужны математические операции над множествами***

**Поведение множеств при добавлении элементов:**  
  
# Создание множества  
**>>> my\_set =** **{0, 1, 2, 3, 3, 5, 6, 7, 7}  
>>> my\_set{0, 1, 2, 3, 5, 6, 7}**  *# как видите, повторяющиеся* ***3*** *и* ***7*** *автоматически удалились*  
  
# Добавление элемента  
**>>> my\_set.add(1)** *# не будет добавлен, так как такой элемент уже есть* **>>> my\_set.add(‘test’)** *# будет добавлен, так как так такого элемента не было* **>>> my\_set.add(True) #** *не будет добавлен, так как есть элемент 1* **>>> my\_set.add(False) #** *не будет добавлен, так как есть элемент 0*

**>>> my\_set  
{0, 1, 2, 3, 5, 6, 7, ‘test’}**

***ВАЖНО!***  
**\*** - В Python объект является “**хэшируемым**”, если он реализует встроенный метод ***\_\_hash\_\_***  
Объекты cоставных типов Python, которые являются ИЗМЕНЯЕМЫМИ, хэшировать НЕЛЬЗЯ! Например, объекты таких типов как **list, dict и set.   
- Кортеж** является НЕИЗМЕНЯЕМЫМ составным типом, однако, если **Кортеж** содержит внутри хотя бы 1 элемент ИЗМЕНЯЕМОГО типа, то такой Кортеж нельзя использовать в качестве ключа словаря или элемента множества.(Подробнее о том, что такое хэш-функции и хэш-таблицы, можно почитать здесь:  
[*http://aliev.me/runestone/SortSearch/Hashing.html*](http://aliev.me/runestone/SortSearch/Hashing.html)*)*

Подробнее о **list, set, dict и tuple** в Python:

- [*https://pythonworld.ru/tipy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html*](https://pythonworld.ru/tipy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html)**-** [*https://pythonworld.ru/tipy-dannyx-v-python/mnozhestva-set-i-frozenset.html*](https://pythonworld.ru/tipy-dannyx-v-python/mnozhestva-set-i-frozenset.html) *- https://pythonworld.ru/tipy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html*

*- https://pythonworld.ru/tipy-dannyx-v-python/kortezhi-tuple.html*

**Range  
  
Генератор** последовательности целых чисел в диапазоне ОТ и ДО.  
Фактически, НЕ является Структурой данных, так как не хранит всю последовательность чисел в памяти

(Что такое **Генератор** в Python мы будем проходить позже. На данном этапе обучения достаточно запомнить только то, что **range генерирует последовательность целых чисел в указанном диапазоне)**Форма записи: **range(START, STOP, STEP)**

**Примеры:  
  
#** последовательность целых чисел **от 0 до 99** (верхняя граница не входит в результат, нижнюю границу можно не указывать, если она 0)

**>>> my\_range1 = range(100)**

**#** последовательность целых чисел **от 5 до 50**  **>>> my\_range2 = range(5, 51)**# последовательность целых чисел от 0 до 100 с шагом 2   
(в случае с указанием шага (по умолчанию равен 1), нужно указывать нижнюю границу **0** явно) **>>>** **my\_range3 = range(0, 101, 2)**Используется чаще всего прямо в **циклах**, когда нужно “пробежаться” циклом какое-то количество раз (реализация некого “счетчика”):

***>>> counter = 0  
>>> for x in range(1, 101):  
… counter += 1  
>>> counter  
100*  
  
(**Запись **counter += 1** эквивалентна записи **counter = counter + 1)**