DIRTY PYTHON 1.0

STONE 17TH PRESENTS

ЛЕКЦИЯ 3

D1RTY PYTHON

ЧТО У НАС СЕГОДНЯ?

- Коллекции: с чем едят и где их применять
- Списки
- Кортежи
- Множества
- Словари

D1RTY PYTHON

Списки: как назвать и что покласть

- Тип данных list – поэтому в нейминге именно list не используем
- Запихиваем что угодно и сколько угодно (тем и отличается от массива)
- Мы всегда можем увеличить и уменьшить: вполне себе резиновый в отличие от Москвы)))
- Что можно положить? Все что угодно, да еще и одновременно. Но не грешите этим, ибо такой многотиповой список может стать причиной багов и исключений (если, конечно, у вас нет специального требования.

```python
my_list = []     # создали пустой список
new_list = list()    # и еще один способ создания пустого списка

list_1 = [1, 'Smith', [1, 2, 3], 4.5, True]     # заполнили список чем пришлось
```

СПИСКИ: ИНДЕКС И ПОРЯДОК

- **Индексируемый** тип данных.
- Помним, что индексирование начинается с нуля
- Если вы попробуете обратиться к элементу по индексу, которого в вашем списке нет - получите ошибку out of range
- Индексирование означает, кстати, что элементы **упорядочены** – куда вы их разместили, там они и лежат.

```
7   # индексы 0 1 2 3 4
8   list_2 = [1, 2, 3, 4, 5]
9   print(list_2[3])  # напечатает нам вовсе не 3, а 4, которая хранится под индексом 3
10  print(list_2[5])  # получим ошибку из-за того, что элемента с таким индексом нет
```

```
Run: lection3
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyPython1/Lection/lection3.py
4
Traceback (most recent call last):
  File "/Users/di/Desktop/DirtyPython/dirtyPython1/Lection/lection3.py", line 10, in <module>
    print(list_2[5])  # получим ошибку из-за того, что элемента с таким индексом нет
IndexError: list index out of range
```

СПИСКИ: ИЗМЕНА!

- **Изменяемы:** вы можете добавить элемент, удалить, изменить расположение, поменять местами (в отличие от строк, помните эксперименты с a и b из первой лекции?), поскольку мы при изменении не создаем новый объект.

```python
a = [1, 2, 3]
b = a
print(a)
print(b)    # получим одинаковые списки a и b
a.append(4)
print(a)
print(b)    # получим одинаковые списки a и b (в отличие от строк)
b.append(5)
print(a)
print(b)    # опять получим одинаковые списки a и b (в отличие от строк)
```

Run: lection3

```
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyPython1/Lection/lection3.py
[1, 2, 3]
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

СПИСКИ: КОПИЯ

- Если мы хотим создать список, который был таким же как уже имеющийся, но не копировать ссылку на объект, а создать совершенно новый (хоть и с таким же содержимым) - используем метод copy()

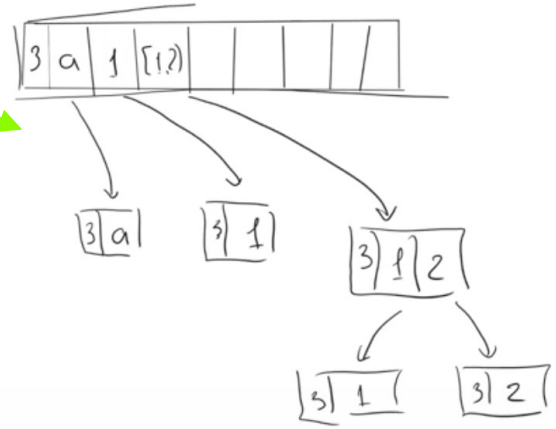
```python
a = [1, 2, 3]
b = a.copy()   # создаем совсем новый, хоть и такой же список
print(a)
print(b)    # получим одинаковые списки a и b
a.append(4)
print(a)
print(b)    # получим разные списки a и b
b.append(5)
print(a)
print(b)    # опять получим одинаковые списки a и b (в отличие от строк)
print(id(a))
print(id(b))    # у них и место в памяти разное
```

lection3

```
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3, 5]
4478598080
4478564928
```

СПИСКИ: ОСОБЕННОСТИ КОПИРОВАНИЯ

- Функция .copy() создает только поверхностную копию. То есть, если у вас в списках есть какие-то вложенные объекты (скажем еще одни списки), то в новом списке вы получите только ссылку на них. Что это значит? Что изменяя такой объект для первого списка, вы изменяет его и для второго… В коде это выглядит понятнее.
- И если мы хотим прям все скопировать глубоко, то нам надо скопировать вот это вот все




```python
a = [1, 2, 3, [1, 2, 3]]
b = a.copy()   # создаем совсем новый, хоть и такой же список
print(a)
print(b)    # получим одинаковые списки a и b
a.append(4)
print(a)
print(b)    # получим разные списки a и b
a[3][1] = 9     # поменяли во вложенном списке второй элемент
print(a)
print(b)    # и смотрите-ка, в списке b он тоже поменялся
print(id(a[3]))
print(id(b[3]))    # у них и место в памяти одно
```

lection3

```
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyPython1/Lection/lection3.py
[1, 2, 3, [1, 2, 3]]
[1, 2, 3, [1, 2, 3]]
[1, 2, 3, [1, 2, 3], 4]
[1, 2, 3, [1, 2, 3]]
[1, 2, 3, [1, 9, 3], 4]
[1, 2, 3, [1, 9, 3]]
4474960832
```

ГЛУБОКАЯ КОПИЯ

- А если мы хотим все-все скопировать, то нам нужно кое-что специальное: библиотеке deepcopy
- Объявляем ее
- Используем deepcopy()

```python
from copy import deepcopy
a = [1, 2, 3, [1, 2, 3]]
b = deepcopy(a)   # вот теперь мы все-все скопировали
print(a)
print(b)    # получим одинаковые списки a и b
a.append(4)
print(a)
print(b)    # получим разные списки a и b
a[3][1] = 9     # поменяли во вложенном списке второй элемент
print(a)
print(b)    # и смотрите-ка, списки в списках разные!
print(id(a[3]))
print(id(b[3]))    # как и место в памяти
```

lection3

```
[1, 2, 3, [1, 2, 3]]
[1, 2, 3, [1, 2, 3]]
[1, 2, 3, [1, 2, 3], 4]
[1, 2, 3, [1, 2, 3]]
[1, 2, 3, [1, 9, 3], 4]
[1, 2, 3, [1, 2, 3]]
4491372928
4491374912
```

МЕТОДЫ СПИСКА

- Их много! Как и у строк
- Ведь у списков, как у строк есть индексы)))
- Делаем список из строки: list(some_string) (но значения будут строчными, даже если у вас были цифры)
- Делаем списки из строк some_string.split()
- Используем списки в циклах for и даже меняем

ВСТАВЛЯЕМ

- my_list.append(’то что добавляем’) добавляем в конец
- Лирическое отступление про способы добавления:
- FIFO: First in, first out – первый пришел, первый ушел, классическая очередь
- LIFO: Last in, first out – первый пришел, последний ушел, как патроны в рожке (если верить Стоуну – лично я понятия не имею)))
- my_list.insert(0, ‘элемент’) – вставляем по индексу(любой): сначала куда, а потом что.

УДАЛЯЕМ ИЗ СПИСКА

- my_list.pop(индекс) – по умолчанию вытаскивает(аналог cntlZ – удаляет из списка по индексу, но возвращает сам элемент: его можно распечатать, присвоить переменной и т.д.) последний элемент, но можно прописать индекс
- my_list.pop(значение) – удаляет первое вхождение элемента из списка, оставляя все последующие. Но если такого элемента нет – то будет ошибка

```
my_list = [1, 2, 3]
print(my_list)
num = my_list.pop(1) # удаляем из списка и возвращаем в переменную
print(my_list)
print(num)
```

```
Run: lection3
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyPython1/Lection/lec
[1, 2, 3]
[1, 3]
2
```

```
my_list = [1, 2, 3, 2]
print(my_list)
my_list.remove(2) # удаляем из списка первую 2
print(my_list)
my_list.remove(2) # удаляем из списка оставшуюся 2
print(my_list)
my_list.remove(2) # выдаст ошибку
print(my_list)
```

```
Run: lection3
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyPyth
[1, 2, 3, 2]
[1, 3, 2]
[1, 3]
Traceback (most recent call last):
  File "/Users/di/Desktop/DirtyPython/dirtyPython1/Lection/lectio
    my_list.remove(2) # выдаст ошибку
ValueError: list.remove(x): x not in list
```

КАК В СТРОКАХ

- my_list.count(‘элемент’) – посчитает сколько элемент встречается в списке
- my_list.index(‘элемент’) – вернет индекс первого вхождения элемента, дополнительно можно указать с какого элемента начинать искать - my_list.index(‘элемент’, 2). Если элемента нет – получится ошибка. Можно проверить count
- my_list.reverse() – разворачиваем список
- my_list.sort() – сортирует по юникоду, можно в скобках указать reverse=True – отсортирует по убыванию
- Срезы: все как в строках

```
my_list = [1, 2, 3, 2]
if my_list.count(5) > 0:        # проверяем наличие 5 в списке
    print(my_list.index(5))     # печатаем, если есть

if my_list.count(2) > 0:         # проверяем наличие 2 в списке
    print(my_list.index(2))      # печатаем, если есть

print(my_list.index(6))         # получаем ошибку, ведь 6 нет
```

```
lection3
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyPython1/Lection
1
Traceback (most recent call last):
  File "/Users/di/Desktop/DirtyPython/dirtyPython1/Lection/lection3.py", lin
    print(my_list.index(6))        # получаем ошибку, ведь 6 нет
ValueError: 6 is not in list

Process finished with exit code 1
```

НЕМНОГО ПРО JOIN

- Если мы хотим получить строку из списка – то есть join
- Используем только списки со строковыми значениями
- Указываем, через что будем клеить
- ’-’.join(my_list)
- Поверьте, когда дойдем до парсинга, вам этот джойн ой как понадобиться

```python
my_list = ['1', '2', '3', '2']
my_string = '-'.join(my_list)   # склеиваем в строку через дефис
print(my_string)
my_string = ' '.join(my_list)   # склеиваем в строку через пробел
print(my_string)
my_string = ''.join(my_list)   # просто склеиваем
print(my_string)
```

lection3

```
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyPython1/Lection/l
1-2-3-2
1 2 3 2
1232
```

КОРТЕЖИ TUPLE

DIRTY PYTHON

- my_tuple = (1, 2, 3)
- Как список он индексируемый, как список он упорядоченный, и запихать в него можно, что угодно, но НЕ ИЗМЕНЯМЫЙ! Каким вы его создали, таким он и помрет…
- Удалить, изменить, добавить – это все нельзя!
- А что можно? Можно поменять изменяемый объект – например, если в кортеже есть список, то внутри списка все можно, что можно со списками.
- Можно использовать count и index

```python
my_tuple = (1, 2, 3, [1, 2, 3])
print(my_tuple)
my_tuple[-1][0] = 'AAAA'        # меняем первый элемент последнего элемента
print(my_tuple)
my_tuple[-1] = ['AAAA', 2, 3]   # а так ничего не получится
print(my_tuple)
```

lection3

```
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyPython1/Lection/lection3.py
Traceback (most recent call last):
  File "/Users/di/Desktop/DirtyPython/dirtyPython1/Lection/lection3.py", line 55, in <module>
    my_tuple[-1] = ['AAAA', 2, 3]   # а так ничего не получится
TypeError: 'tuple' object does not support item assignment
(1, 2, 3, [1, 2, 3])
(1, 2, 3, ['AAAA', 2, 3])
```

И ЗАЧЕМ ОНИ ТАКИЕ?

- Использовать как константы, чтобы случайно не изменить что-то важное
- Для работы с базами данных в SQL- запрос можно передать именно кортеж
- Конечно, обойти неизменяемость кортежа можно: превратите его сначала в список, измените, а затем снова превратите в кортеж

```python
my_tuple = (1, 2, 3, [1, 2, 3])
print(my_tuple)
print(type(my_tuple))
my_tuple = list(my_tuple)   # превратили в список
print(my_tuple)
print(type(my_tuple))
my_tuple[0] = 0             # поменяли что хотели
print(my_tuple)
my_tuple = tuple(my_tuple)   # превратили в кортеж
print(my_tuple)
print(type(my_tuple))
```

lection3

```
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyP
(1, 2, 3, [1, 2, 3])
<class 'tuple'>
[1, 2, 3, [1, 2, 3]]
<class 'list'>
[0, 2, 3, [1, 2, 3]]
(0, 2, 3, [1, 2, 3])
<class 'tuple'>
```

РАСПАКОВКА

- Чтобы достать данные из коллекции, можно просто присвоить их переменным
- Но если будет не совпадение количества переменных и элементов – то нужно использовать *, чтобы в одну переменную запихнуть оставшуюся коллекцию
- Эту же звездочку можно использовать и для print – все будет красиво.
- Кстати, использовав sep='-' можно вывести через дефис (или что вам нравится, по умолчанию это пробел)
- Распаковка аналогично работает и со списками и другими коллекциями

```python
my_tuple = (1, 2, 3, 4, 5, 6, 7)
dig1, dig2, *dig3, last = my_tuple
print(dig1)     # первый элемент
print(dig2)     # второй элемент
print(dig3)     # кортеж с третьего по предпоследний
print(last)     # последний элемент
```

lection3

```
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyP
1
2
[3, 4, 5, 6]
7

Process f
```

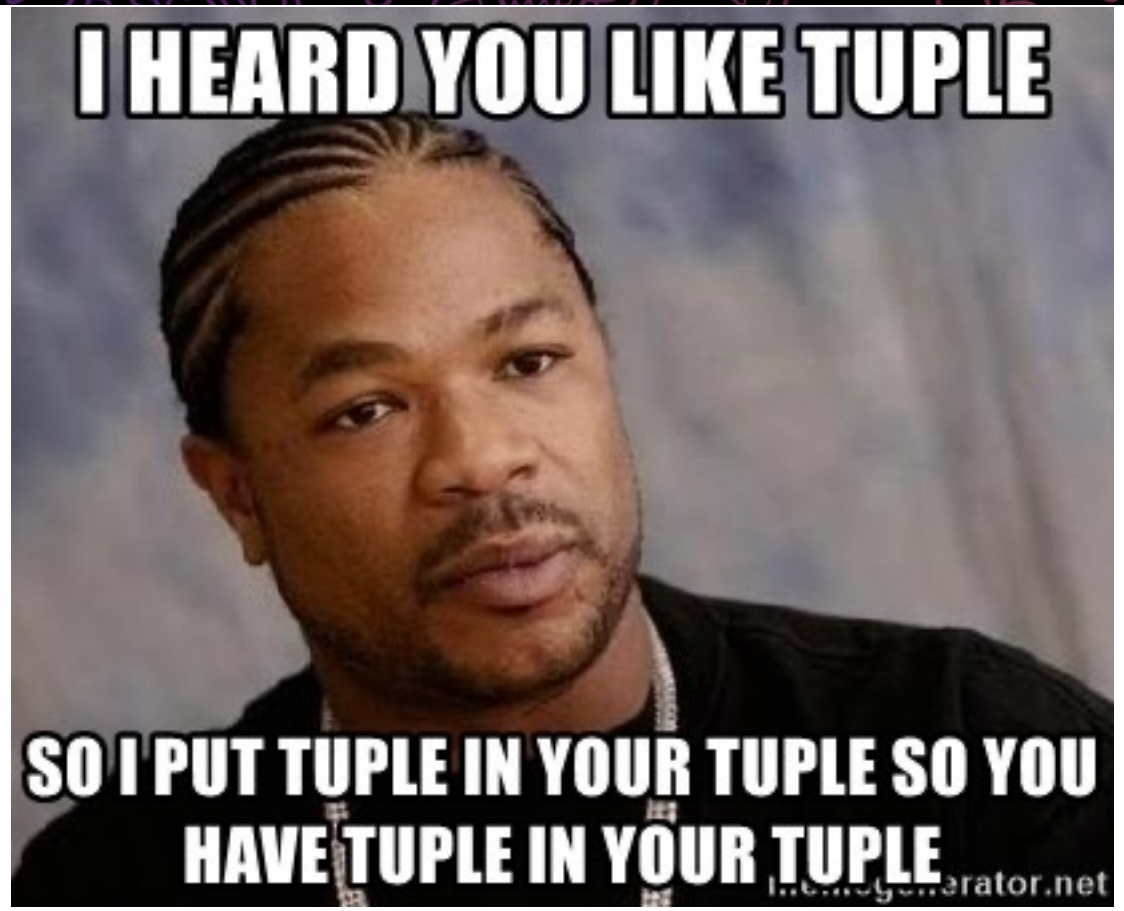
```python
my_tuple = (1, 2, 3, 4, 5, 6, 7)
print(my_tuple)  # напечатает в скобках
print(*my_tuple)  # напечатает через пробел
print(*my_tuple, sep='-')  # напечатает через -
```

lection3

```
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/
(1, 2, 3, 4, 5, 6, 7)
1 2 3 4 5 6 7
1-2-3-4-5-6-7
```

Кортеж из одного элемента

- Если вы вдруг захотите создать кортеж из одного элемента (и да, это действительно бывает нужно), то сделать это можно так:
- my_tuple = (1,) – используйте круглые скобки и после элемента поставьте запятую
- Вот такая конструкция my_tuple = (1) будет воспринята как просто интеджервая единица



МНОЖЕСТВО SET

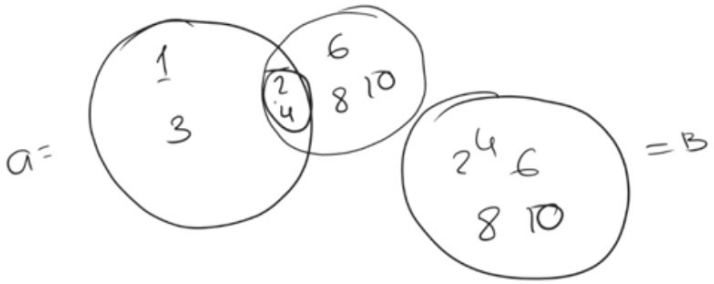
- Вспоминаем математику и кружочки… Почитайте сами, это не сложно)))
- Множества НЕУПОРЯДОЧЕНЫ и НЕИДЕКСИРУЕМЫ
- Множества ИЗМЕНЯМЫ
- Множества состоят из УНИКАЛЬНЫХ элементов
- В множествах можно хранить только неизменяемые типы данных

```python
my_set = {1, 2, 3, 4, 5, 5, 5, 5}   # пишем в фигурных скобках, а повторы сам уберет
print(my_set)
```

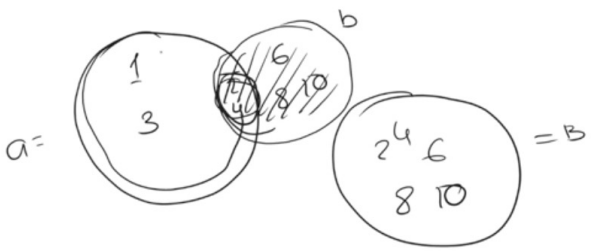
lection3

```
/usr/local/bin/python3.10 /Users/di/Desktop/DirtyPython/dirtyPython1/Lection/lection3.py
{1, 2, 3, 4, 5}
```

Объединение множеств



Вычитание из множества a - b



Методы множеств

- remove – удаляем по значению
- copy – делаем копию
- add – добавление, и даже если вам красивенько добавило в конец – не обольщайтесь: через пару операций все пойдет по…
- clear – очищает множество
- pop() – удаляет и возвращает случайный элемент
- И куча методов чисто для множеств:
- issubset проверяет является ли множество подмножеством другого
- difference – разница между множествами
- intersection – пересечение и еще много чего…

ГДЕ ЖЕ ПРИГОДИТЬСЯ МНОЖЕСТВО?

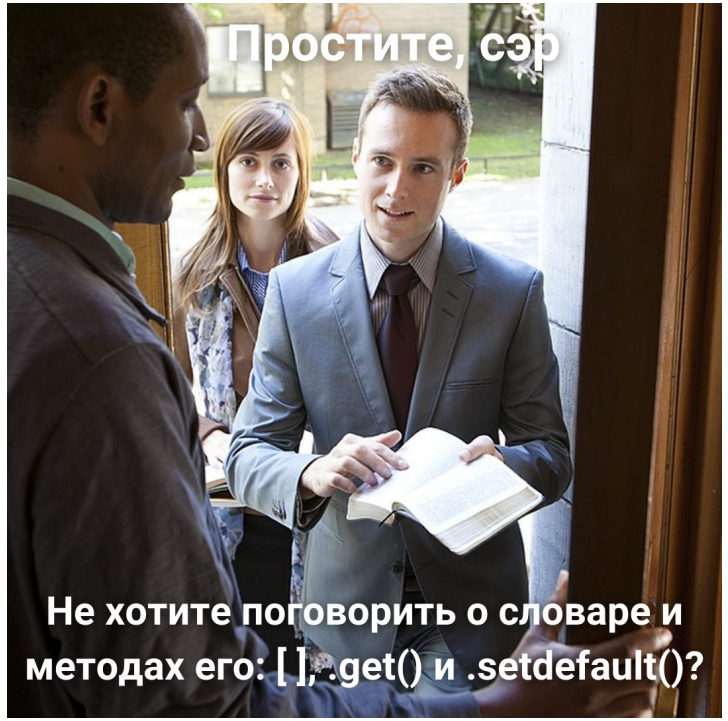
- Для работы с множествами)))) серьезно, кстати, люди с ними тоже работают
- Для того, чтобы отобрать уникальные значения
- И потом использовать как ключи для словаря, например
- Псевдорандомайзер (при помощи pop, лучше брать строковые значения, с цифрами не очень на рандом похоже)

```
my_set = {str(i) for i in range(100)}

for _ in range(6):
    print(my_set.pop())
    input()
```

Кстати, если переменная используется один раз, то ее можно обозначить нижним подчеркиванием (это хороший стиль

СЛОВАРЬ

- тип данных чем-то похожий на json, и поэтому прям очень широко используется
- Представляет собой набор уникальных ключей и их значений (значения могут быть уникальны)
- Сложнее и интереснее чем списки, но его надо понять. Как только поймете – вы его полюбите
- ИЗМЕНЯЕМЫЙ
- НЕУПОРЯДОЧЕННЫЙ (индексов нет!)
- ПАРНЫЙ: ключ (уникальный, неизменяемый тип данный) и значение (а тут вообще, что угодно)
- Могут быть ВЛОЖЕННЫМИ




```
dict

my_dict = {'Key': {1234: {'key': 900}}}
```

РАБОТАЕМ СО СЛОВАРЕМ

- Чтобы извлечь что-то из словаря нужно знать ключ. И обратиться по нему
- pop() удаляет пару
- copy() делает копию
- keys() возвращает все ключи
- get() возвращает по ключу
- values() возвращает все значения
- items() возвращает и ключ, и значение
- update() дополняет один словарь другим
- И др.

```
my_dict = {'Key': {1234: {'key': 900}}}

print(my_dict.get('Key').get(1234).get('key'))
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonP
900
```

Я К ТЕБЕ ОБРАЩАЮСЬ

- my_dict[‘ключ’] – вернет значение по ключу, но если такого ключа нет – будет ошибка
- my_dict.get(‘ключ’) – вернет значение, а если его нет – то вернет None или то, что вы задаете по умолчанию
- Как добавить новый? Очень легко
- my_dict[‘новый ключ’]= ’значение’
- Если ключ уже был – то будет перезаписано новое значение.
- Безопасное добавление my_dict.setdefault(key, value) – добавит только в том случае, если такого ключа еще нет в вашем словаре.

```
my_dict = {'ONE': 1, 2: 'Two', 3: 'ТРИ'}

my_dict[2] = 'FOUR'
print(my_dict.get(4, 'Нет такого ключа'))
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pyth
Нет такого ключа
```

```
my_dict = {'ONE': 1, 2: 'Two', 3: 'ТРИ'}

my_dict[2] = 'FOUR'
print(my_dict)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\
{'ONE': 1, 2: 'FOUR', 3: 'ТРИ'}
```

А ТЕПЕРЬ У НАС ДВА СЛОВАРЯ

- Метод dict1.update(dict2) обновляет словарь dict1, добавляя пары (ключ, значение) из dict2. Существующие ключи перезаписываются. Возвращает None (не новый словарь!) - сами словари остаются, только dict1 вы обновили.

```python
my_dict = {'ONE': 1, 2: 'Two', 3: 'ТРИ'}
new_dict = {'ONE': 10, 2: 'Two!', 4: 'Четыре'}

my_dict.update(new_dict)
print(my_dict)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\Wor
{'ONE': 10, 2: 'Two!', 3: 'ТРИ', 4: 'Четыре'}
```

- Чтобы сделать из списка или множества ключи – используйте функцию fromkeys(): по умолчанию всем ключам будут присвоены значения None, но вы можете прописать что-то свое

```python
my_dict = {'ONE': 1, 2: 'Two', 3: 'ТРИ'}
new_dict = {'ONE': 10, 2: 'Two!', 4: 'Четыре'}
keys = {'ONE', 2, 3}
print(my_dict.fromkeys(keys))
print(my_dict)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\Work
{2: None, 3: None, 'ONE': None}
```

FOR СО СЛОВАРЕМ

- По умолчанию проходится только по ключам, но можно и уточнить при помощи keys
- А можем пробежаться по значениям
- А можем по items – получим кортежи из пары ключ, значение
- А можно при помощи items разобрать на ключи и значения

```
my_dict = {'ONE': 1, 2: 'Two', 3: 'ТРИ'}
new_dict = {'ONE': 10, 2: 'Two!', 4: 'Четыре'}

for i in my_dict:
    print(i)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pytho
ONE
2
```

```
my_dict = {'ONE': 1, 2: 'Two', 3: 'ТРИ'}
new_dict = {'ONE': 10, 2: 'Two!', 4: 'Четыре'}

for i in my_dict.items():
    print(i)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\Work
('ONE', 1)
(2, 'Two')
(3, 'ТРИ')
```

```
my_dict = {'ONE': 1, 2: 'Two', 3: 'ТРИ'}
new_dict = {'ONE': 10, 2: 'Two!', 4: 'Четыре'}

for key, value in my_dict.items():
    print(key)
    print(value)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D
ONE
1
2
Two
3
ТРИ
```

```
my_dict = {'ONE': 1, 2: 'Two', 3: 'ТРИ'}
new_dict = {'ONE': 10, 2: 'Two!', 4: 'Четыре'}

for i in my_dict.values():
    print(i)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonProjecti
1
Two
ТРИ
```

FOR ДЛЯ МНОЖЕСТВА..

- При помощи цикла for можно пробежаться по множеству и достать из него все элементы (напомним, что порядка в этом не будет)

```
6   my_set = {'31231', 'asdfds', '1231', 1231, 'd'}
7
8   for item in my_set:
9       print(item)
10
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonI
1231
d
31231
asdfds
1231

Process finished with exit code 0
```

МАССИВЫ

- Они как бы есть, но с этим сложнее
- Импортируем библиотеку
- Создаем массив с указанием типа данных
- Но если добавить туда float
- Получим ошибку
- Вот вам и массив!

```python
from array import *

my_array = array('i', (1,2,3,4,5,6,77,8,9,0))

print(my_array.tolist())
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\Wo
[1, 2, 3, 4, 5, 6, 77, 8, 9, 0]

Process finished with exit code 0
```

```python
from array import *

my_array = array('i', (1,2,3,4.8,5,6,77,8,9,0))

print(my_array)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonProje
Traceback (most recent call last):
  File "D:\WorkShopPy\pythonProject1\main.py", line 3, in <module>
    my_array = array('i', (1,2,3,4.8,5,6,77,8,9,0))
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: 'float' object cannot be interpreted as an integer

Process finished with exit code 1
```

COMPREHENSION

```python
my_list = []

for i in range(10):
    my_list.append(i)

print(my_list)
```

```python
my_list = [i for i in range(10)]


print(my_list)
```

- Если нам надо заполнить список, а не хочется делать это в ручном режиме, можно использовать цикл for
- Или еще более короткую запись List comprehension, которая позволяет делать и более сложные, даже вычурные вещи: кортеж из i и i в кубе для i от 0 до 10 (не вкл), если i четное

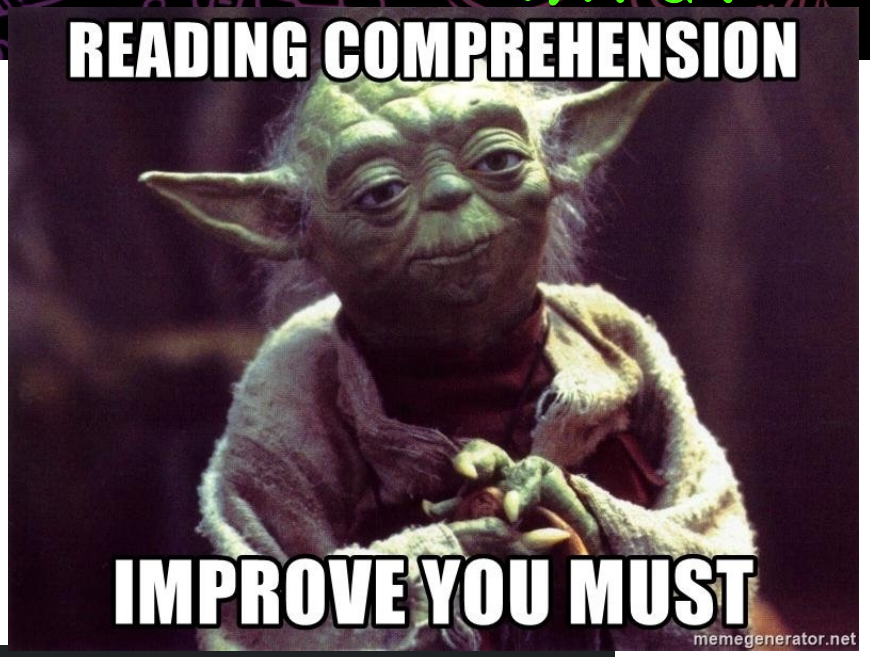
```python
my_list = [(i, i**3) for i in range(10) if not i%2 ]


print(my_list)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkS
[(0, 0), (2, 8), (4, 64), (6, 216), (8, 512)]
```

COMPREHENSION ДЛЯ ВСЕГО

DIRTY PYTHON

- Так же мы можем генерировать множества
- Кортеж так не создать, получите генератор (а про них позже)
- Можем сделать список из случайных значений
- Словари то же можно так создавать



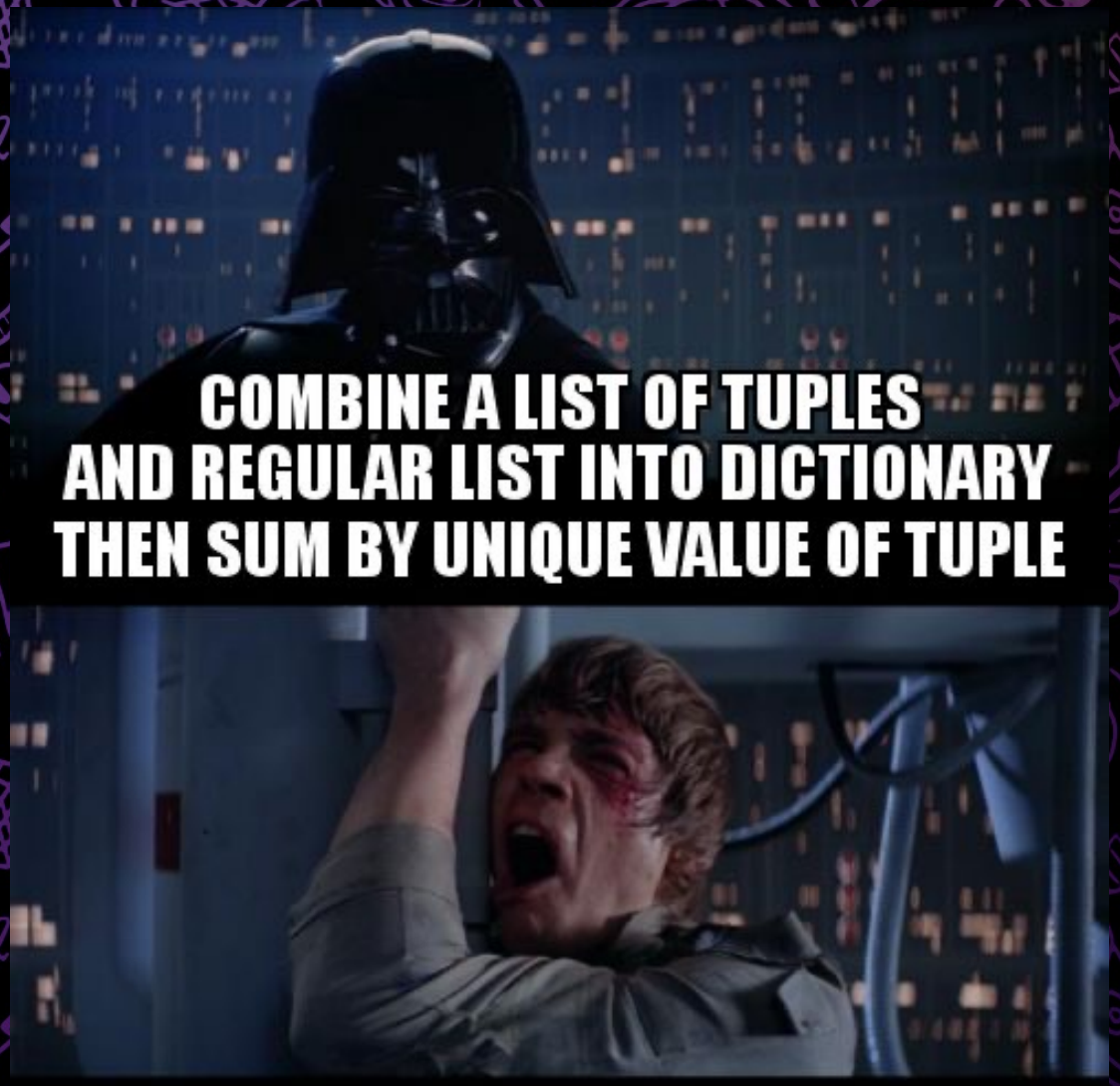

```python
my_set = {random.randint(0,10) for i in range(20)}


print(my_set)
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkSho
{0, 1, 2, 3, 4, 6, 7, 8, 9, 10}
```

```python

my_set = set()
[my_set.add(random.randint(0,100)) for i in range(10000) if len(my_set) < 20]

print(my_set)
print(len(my_set))
```

```
D:\WorkShopPy\.my_venv_3.11.3\Scripts\python.exe D:\WorkShopPy\pythonProject1\main.py
{0, 8, 9, 11, 23, 24, 25, 28, 42, 46, 53, 57, 61, 64, 70, 73, 74, 80, 98, 100}
20
```

И ЧТО ДАЛЬШЕ?



DIRTY PYTHON

И ЧТО ДАЛЬШЕ?

- НЕ СТЕСНЯЕМСЯ ОШИБОК! ОБУЧЕНИЕ БЕЗ ОШИБОК НЕ ВОЗМОЖНО!
- ПОЭТОМУ ДЕЛАЕМ ДОМАШКУ! ЕСЛИ ВАМ ВСЕ ПОНЯТНО, НО ВЫ НЕ УСПЕЛИ ПОКОДИТЬ ЛУЧШЕ РАЗБОР ДОМАШКИ ПОСМОТРИТЕ В ЗАПИСИ, ПОСЛЕ ТОГО КАК РЕШИТЕ САМИ. А ЕСЛИ ВЫ УСПЕЛИ ПОКОДИТЬ, НО ЧЕГО-ТО НЕ ПОНИМАЕТЕ ПРИХОДИТЕ!
- РАЗБЕРЕМ ДОМАШКУ ИЗ ГБ И ПЕРВУЮ ДОМАШКУ ИЗ DP

D1RTY PYTHON