12 мощных Python однострочников от Диджитализируй!

По видео: https://youtu.be/LkHCy5JZtsA

1. Распаковка значений

Например, мы пишем программу, которая считает объём комнаты, на вход она принимает три числа — размеры комнаты. Как мы можем написать такой код?

Было:

```
sizes = input()
sizes = sizes.strip()
sizes = sizes.split()
x = sizes[0]
y = sizes[1]
z = sizes[2]
```

Стало:

```
x, y, z = input().strip().split()
```

2. map

Но затем, чтобы посчитать объём, нам надо привести типы к int. Как это можно сделать?

Было:

```
x, y, z = input().strip().split()
x = int(x)
y = int(y)
z = int(z)
print(f"Volume is {x * y * z}")
```

Стало:

```
x, y, z = map(int, input().strip().split())
print(f"Volume is {x * y * z}")
```

Аналогично можно, например, быстро обрезать пробельные символы в наборе данных:

```
names = ["  Христофор", "Адемар  ", " Тэя ", "  Стефания ", "  Архип"]
names = map(str.strip, names)
```

3. reduce

Используется, когда надо свести последовательность к одному значению. Например, у нас есть объект `top_level_object` и произвольная цепочка атрибутов, которые передаются откуда-то снаружи и мы не можем её задать явно в нашем коде, при этом нам нужно получить значение конечного атрибута:

```
class Inner3:
    attr4 = 444
class Inner2:
    attr3 = Inner3()
class Inner1:
    attr2 = Inner2()
class TopLevel:
    attr1 = Inner1()
top_level_object = TopLevel()
# value = top_level_object.attr1.attr2.attr2.attr4
```

Как это сделать без `reduce`?

```
inner = top_level_object
for attr in ("attr1", "attr2","attr3", "attr4"):
    inner = getattr(inner, attr)
print(inner)
```

Можно воспользоваться `reduce` и сделать короче:

```
from functools import reduce
value = reduce(getattr, ("attr1", "attr2", "attr3", "attr4"), self)
```

4. List comprehensions

Это удобная конструкция для сборки новых списков из какой-то другой последовательности.
Например, у нас есть список имён и нам нужно получить имена, начинающиеся на А.

Было:

```
names = ["Христофор", "Адемар",
         "Тэя", "Стефания", "Архип"]
names_starts_a = []
for name in names:
    if name.startswith("A"):
        names_starts_a.append(name)
print(names_starts_a)
```

Стало:

```
names = ["Христофор", "Адемар", "Тэя", "Стефания", "Архип"]
names_starts_a = [name for name in names if name.startswith("A")]
print(names_starts_a)
```

5. Фильтрация

Как можно еще написать фильтрацию данных? Можно через цикл, через comprehension, а можно через `filter`:

```
names = ["Христофор", "Адемар", "Тэя", "Стефания", "Архип"]
names_starts_with_a = filter(lambda name: name.startswith("A"), names)
print(tuple(names_starts_with_a))
```

6. Быстрое копирование списка

Иногда нам нужно скопировать список, то есть не создать ещё одну ссылку на него, а именно создать новый список со значениями из старого списка.

```
>>> indexes = [1, 2, 3]
>>> another_indexes = indexes
>>> indexes.append(4)
>>> another_indexes
[1, 2, 3, 4]
```

Как же нам скопировать список, а не создать еще одну ссылку на старый список?

```
indexes = [1, 2, 3]
another_indexes = [x for x in indexes]  # слишком длинно
another_indexes = indexes[:]  # просто берём полный слайс из начального списка
```

7. Обратить список

Нужно нечасто, но иногда задача возникает, проще всего сделать через слайс, указав шаг в минус 1 элемент:

```
numbers = [100, 200, 300]
numbers[::-1]
```

8. Много проверок == одной переменной

Было:

```
if name == "Христофор" or name == "Адемар" or name == "Тэя":
    print(name)
```

Стало:

```
if name in ("Христофор", "Адемар", "Тэя"):
    print(name)
```

9. Проверить, что все значения True

Было:

```
if a and b and c and d and e:
    print("ok")
```

Стало:

```
if all(a, b, c, d, e):
    print("ok")
```

10. Проверить, что хотя бы 1 значение True

Было:

```
if a or b or c or d or e:
    print("ok")
```

Стало:

```
if any(a, b, c, d, e):
    print("ok")
```

`any` переводится как любой, то есть если любой из переданных в функцию аргументов приводится к `True`, то функция возвращает `True`.

11. Тернарный оператор

Было:

```
if user.active:
    color = "green"
else:
    color = "red"
```

Стало:

```
color = "green" if user.active else "red"
```

12. Конфигурируем, что вызвать

Было:

```
if user.group == "admin":
    process_admin_request(user, request)
elif user.group == "manager":
    process_manager_request(user, request)
elif user.group == "client":
    process_client_request(user, request)
```

Стало:

```
group_to_method = {
    "admin": process_admin_request,
    "manager": process_manager_request,
    "client": process_client_request
}
group_to_method[user.group](user, request)
```