



# Коллекции данных. Профилирование и отладка.

Курс



# Оглавление

Термины, используемые в лекции	2
Списки	2
List Comprehension	10
Профилирование и отладка	12

## Термины, используемые в лекции

**Кортеж** — это неизменяемый список.

**Словари** — неупорядоченные коллекции произвольных объектов с доступом по ключу.

## Коллекции данных, что это такое и для чего они нужны?

Коллекция — структура данных, которая создана, чтобы содержать в себе некоторое количество объектов (они могут быть одного типа или могут быть разных типов).

Рано или поздно переменную становятся не такими удобными, как могло показаться на первый взгляд. Например: **Необходимо записать имя каждого студента на потоке 1111**. И так, как мы будем решать эту задачу? На потоке около 5000 студентов и заводить для каждого человека отдельную переменную не очень разумное решение, в этом случае нам помогает такой тип данных, как List(список)

# Списки

Список - это упорядоченный конечный набор элементов. Давайте разбираться, по сути список - это тот же самый массив, в котором можно хранить элементы любых типов данных.

## Как работать со списками?

```
list_1 = [] # Создание пустого списка
list_2 = list() # Создание пустого списка
list_1 = [7, 9, 11, 13, 15, 17]
```

В списках существует нумерация, которая начинается с 0, чтобы вывести первый элемент списка воспользуемся следующей конструкцией:

```
list_1 = [7, 9, 11, 13, 15, 17]
print(list_1[0]) # 7
```

Чтобы узнать количество элементов в списке необходимо использовать функцию **len(имя\_списка)**:

```
list_1 = [7, 9, 11, 13, 15, 17]
print(len(list_1)) # 6
```

Можно список заполнять не только при его создание, но и во время работы программы:

```
list_1 = list() # создание пустого списка
for i in range(5): # цикл выполнится 5 раз
    n = int(input()) # пользователь вводит целое число
    list_1.append(n) # сохранение элемента в конец списка
# 1-я итерация цикла(повторение 1): n = 12, list_1 = [12]
# 2-я итерация цикла(повторение 2): n = 7, list_1 = [12, 7]
# 3-я итерация цикла(повторение 3): n = -1, list_1 = [12, 7, -1]
# 4-я итерация цикла(повторение 4): n = 21, list_1 = [12, 7, -1, 21]
```

```
# 5-я итерация цикла (повторение 5): n = 0, list_1 = [12, 7, -1, 21, 0]
print(list_1) # [12, 7, -1, 21, 0]
```

Мы обговорили с Вами создание списка и поняли, что мы можем пользоваться нумерацией, для того чтобы узнать какой элемент стоит на той или иной позиции. Но это не всегда удобно, особенно, когда список будет состоять из 1000, 1000000... элементов. В этом случае необходимо использовать цикл **for**.

### Взаимодействие цикла for со списком:

```
list_1 = [12, 7, -1, 21, 0]
for i in list_1:
    print(i) # вывод каждого элемента списка

# 1-я итерация цикла (повторение 1): i = 12
# 2-я итерация цикла (повторение 2): i = 7
# 3-я итерация цикла (повторение 3): i = -1
# 4-я итерация цикла (повторение 4): i = 21
# 5-я итерация цикла (повторение 5): i = 0
```

### Не забываем, что у списка есть нумерация:

```
list_1 = [12, 7, -1, 21, 0]
for i in range(len(list_1)):
    print(list_1[i]) # вывод каждого элемента списка

# 1-я итерация цикла (повторение 1): list_1[0] = 12
# 2-я итерация цикла (повторение 2): list_1[1] = 7
# 3-я итерация цикла (повторение 3): list_1[2] = -1
# 4-я итерация цикла (повторение 4): list_1[3] = 21
# 5-я итерация цикла (повторение 5): list_1[4] = 0
```

## Основные действия со списками:

1. Удаление последнего элемента списка.

**Метод pop удаляет последний элемент из списка:**

```
list_1 = [12, 7, -1, 21, 0]

print(list_1.pop()) # 0

print(list_1) # [12, 7, -1, 21]

print(list_1.pop()) # 21

print(list_1) # [12, 7, -1]

print(list_1.pop()) # -1

print(list_1) # [12, 7]
```

2. Удаление конкретного элемента из списка.

**Надо указать значение индекса в качестве аргумента функции pop:**

```
list_1 = [12, 7, -1, 21, 0]

print(list_1.pop(0)) # 12

print(list_1) # [7, -1, 21, 0]
```

3. Добавление элемента на нужную позицию.

**Функция insert — указание индекса (позиции) и значения.**

```
list_1 = [12, 7, -1, 21, 0]

print(list_1.insert(2, 11))

print(list_1) # [12, 7, 11, -1, 21, 0]
```

## Срез списка

Помните в конце первой лекции Вы прошли срезы строк? Также существует срез списка, давайте научимся изменять наш список

- Отрицательное число в индексе — счёт с конца списка

```
list_1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(list_1[0])           # 1
print(list_1[1])           # 2
print(list_1[len(list_1)-1]) # 10
print(list_1[-5])          # 6
print(list_1[:])           # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(list_1[:2])          # [1, 2]
print(list_1[len(list_1)-2:]) # [9, 10]
print(list_1[2:9])         # [3, 4, 5, 6, 7, 8, 9]
print(list_1[6:-18])       # []
print(list_1[0:len(list_1):6]) # [1, 7]
print(list_1[::6])         # [1, 7]
```

## Кортежи



Кортеж — это неизменяемый список.

Тогда для чего нужны кортежи, если их нельзя изменить? В случае защиты каких-либо данных от изменений (намеренных или случайных). Кортеж занимает **меньше места в памяти и работают быстрее**, по сравнению со списками

```
t = () # создание пустого кортежа

print(type(t)) # class <'tuple'>

t = (1,)

print(type(t))

t = (1)

print(type(t))

t = (28, 9, 1990)

print(type(t))

colors = ['red', 'green', 'blue']

print(colors)          # ['red', 'green', 'blue']

t = tuple(colors)

print(t)                # ('red', 'green', 'blue')

t = tuple(['red', 'green', 'blue'])

print(t[0]) # red

print(t[2]) # blue

for e in t:

    print(e) # red green blue

t[0] = 'black' # TypeError: 'tuple' object does not support (нельзя изменять кортеж)
```


## Можно распаковать кортеж в независимые переменные:

```
t = tuple(['red', 'green', 'blue'])

red, green, blue = t

print('r:{} g:{} b:{}'.format(red, green, blue)) # r:red g:green b:blue
```

## Словари

 Словари — неупорядоченные коллекции произвольных объектов с доступом по ключу.

В списках в качестве ключа используется индекс элемента. В словаре для определения элемента используется значение ключа (строка, число).

```
dictionary = {}

dictionary = {'up': '↑', 'left': '←', 'down': '↓', 'right': '→'}

print(dictionary) # {'up':'↑', 'left':'←', 'down':'↓', 'right':'→'}

print(dictionary['left']) # ←

# типы ключей могут отличаться

print(dictionary['up']) # ↑

# типы ключей могут отличаться

dictionary['left'] = '↵'

print(dictionary['left']) # ↵

print(dictionary['type']) # KeyError: 'type'

del dictionary['left'] # удаление элемента

for item in dictionary: # for (k,v) in dictionary.items():

    print('{}: {}'.format(item, dictionary[item]))

# up: ↑

# down: ↓

# right: →
```



## Множества



Множества содержат в себе уникальные элементы, не обязательно упорядоченные.

Одно множество может содержать значения любых типов. Если у Вас есть два множества, Вы можете совершать над ними любые стандартные операции, например, объединение, пересечение и разность. Давайте разберем их.

```
colors = {'red', 'green', 'blue'}

print(colors) # {'red', 'green', 'blue'}

colors.add('red')

print(colors) # {'red', 'green', 'blue'}

colors.add('gray')

print(colors) # {'red', 'green', 'blue', 'gray'}

colors.remove('red')

print(colors) # {'green', 'blue', 'gray'}

colors.remove('red') # KeyError: 'red'

colors.discard('red') # ok

print(colors) # {'green', 'blue', 'gray'}

colors.clear() # { }

print(colors) # set()
```

## Операции со множествами в Python

```
a = {1, 2, 3, 5, 8}

b = {2, 5, 8, 13, 21}

c = a.copy() # c = {1, 2, 3, 5, 8}

u = a.union(b) # u = {1, 2, 3, 5, 8, 13, 21}

i = a.intersection(b) # i = {8, 2, 5}

dl = a.difference(b) # dl = {1, 3}

dr = b.difference(a) # dr = {13, 21}

q=a.union(b).difference(a.intersection(b)) # {1, 21, 3, 13}
```

**Неизменяемое или замороженное множество(frozenset)** — множество, с которым не будут работать методы удаления и добавления.

```
a = {1, 2, 3, 5, 8}
b = frozenset(a)
print(b) # frozenset({1, 2, 3, 5, 8})
```

### Обобщение свойств встроенных коллекций в сводной таблице:

Тип коллекции	Изменяемость	Индексированность	Уникальность	Как создаём
<b>Список</b> (list)	+	+	-	<code>[]</code> <code>list()</code>
<b>Кортеж</b> (tuple)	-	+	-	<code>()</code> , <code>tuple()</code>
<b>Строка</b> (string)	-	+	-	<code>" "</code> <code>" "</code>
<b>Множество</b> (set)	+	-	+	<code>{elm1, elm2}</code> <code>set()</code>
<b>Неизменяемое множество</b> (frozenset)	-	-	+	<code>frozenset()</code>
<b>Словарь</b> (dict)	+ элементы - ключи + значения	-	+ элементы + ключи - значения	<code>{}</code> <code>{key: value,}</code> <code>dict()</code>

## List Comprehension

У каждого языка программирования есть свои особенности и преимущества. Одна из культовых фишек Python — list comprehension (редко переводится на русский, но можно использовать определение «генератора списка»). Comprehension легко читать, и их используют как начинающие, так и опытные разработчики.

**List comprehension** — это упрощенный подход к созданию списка, который задействует цикл for, а также инструкции **if-else** для определения того, что в итоге окажется в финальном списке.

### 1. Простая ситуация — список:

```
list_1 = [exp for item in iterable]
```

### 2. Выборка по заданному условию:

```
list_1 = [exp for item in iterable (if conditional)]
```

**Задача:** Создать список, состоящий из четных чисел в диапазоне от 1 до 100.

**Решение:**

#### 1. Создать список чисел от 1 до 100

```
list_1 = []  
for i in range(1, 101):  
    list_1.append(i)  
print(list_1) # [1, 2, 3, ..., 100]
```

**Эту же функцию можно записать так:**

```
list_1 = [i for i in range(1, 101)] # [1, 2, 3, ..., 100]
```

## 2. Добавить условие (только чётные числа)

```
list_1 = [i for i in range(1, 101) if i % 2 == 0] # [2, 4, 6, ..., 100]
```

Допустим, вы решили создать пары каждому из чисел (кортежи)

```
list_1 = [(i, i) for i in range(1, 101) if i % 2 == 0] # [(2, 2), (4, 4), ..., (100, 100)]
```

Также можно умножать, делить, прибавлять, вычитать. Например, умножить значение на 2.

```
list_1 = [i * 2 for i in range(10) if i % 2 == 0]
print(list_1) # [0, 4, 8, 12, 16]
```

# Профилирование и отладка

Мы с вами люди, а людям суждено ошибаться, даже при написании программного кода 😊

Давайте разберем с Вами самые частые ошибки в написании кода на Python.



Самые распространенные ошибки:

- `SyntaxError`(Синтаксическая ошибка)

```
number_first = 5
number_second = 7
if number_first > number_second # !!!!!
    print(number_first)

# Отсутствие :
```

- IndentationError(Ошибка отступов)

```
number_first = 5
number_second = 7
if number_first > number_second:
print(number_first) # !!!!!

# Отсутствие отступов
```

- TypeError(Типовая ошибка)

```
text = 'Python'
number = 5
print(text + number)

# Нельзя складывать строки и числа
```

- ZeroDivisionError(Деление на 0)

```
number_first = 5
number_second = 0
print(number_first // number_second)

# Делить на 0 нельзя
```

- KeyError(Ошибка ключа)

```
dictionary = {1: 'Monday', 2: 'Tuesday'}
print(dictionary[3])

# Такого ключа не существует
```

- `NameError`(Ошибка имени переменной)

```
name = 'Ivan'

print(names)

# Переменной names не существует
```

- `ValueError`(Ошибка значения)

```
text = 'Python'

print(int(text))

# Нельзя символы перевести в целые значения
```

## Итоги:

### *1. Мы изучили коллекции данных:*

- *списки*
- *кортежи*
- *словари*
- *множества*

### *2. List Comprehension*

### *3. Профилирование и отладка*