

< Teach
Me
Skills />

Занятие 2. Основы python

План занятия

- Проверка домашнего задания
- Повторение теории
- Комментарии
- Функции
- Математические операции над числами
- Типы данных
 - Списки
 - Кортежи (tuple)
 - Словари (dict)
 - Сеты (set, frozenset)
- Изменяемые и неизменяемые типы данных
- Преобразование типов данных

1

Комментарии

Комментарии

Зачем?

- пояснить сложные или неочевидные части кода
- документирование кода

комментарий может быть односрочным

"""

Либо комментарий может быть
многострочным

"""

2

ФУНКЦИИ

Функции

Зачем?

Функция - кусок кода, который мы можем использовать МНОГОКРАТНО и нам не нужно будет его снова писать

Пример:

```
# функция выводит сообщение на экран
def func():
    print("Hello world!")
```

Функции - синтаксис

```
: # функция может принимать аргументы - переменные,  
# которые она будет использовать для вычислений  
def func(name):  
    string = f"Hello, {name}"  
    return string # при помощи return функция возвращает значение
```

```
: message = func("Lena")  
print(message)
```

Hello, Lena

3

Математические операции над числами

Математические операции над числами

- **min(1, 2, 3)** - поиск минимального значения
- **max(1, 2, 3)** - поиск максимального значения
- **abs(-1000)** - модуль числа
- **round(10.20232, 2)** - округлить число до 2 символов после запятой
- **pow(2, 3)** - возвести число 2 в степень 3

```
round(10.989921898129389, 2)
```

10.99

4

Коллекции

Список

Список - набор элементов, желательно однотипных, например:

- список продуктов - ["молоко", "сыр", "картошка"]
- список принадлежностей - ["тетрадь", "ручка", "пенал", "карандаш"]

Объявление пустого списка:

- `products = [] # предпочтительно`
- `products = list()`

Основные операции:

- `len(products)` - получить длину списка
- `products.append("хлеб")` - добавить 1 элемент в список
- `products.extend(["яблоки", "огурцы", "бананы"])` - добавить несколько эл-тов

Работа со списком

- `products[0]` - получить элемент списка по индексу
- `products[1:5:2]` - получить с первого по пятый элемент с шагом 2
- `products[:]` - сделать копию списка (или `products.copy()`)

Важно! Список изменяется на месте, то есть функция `products.append()` ничего не возвращает

Поэтому если мы хотим сохранить предыдущее состояние списка, то нам нужно сделать его копию:

```
products_1 = ["молоко", "сыр", "картошка"]
```

```
products_2 = products_1[:]
products_2.append("огурцы")
```

```
print(f"products_1: {products_1}")
print(f"products_2: {products_2}")
```

```
products_1: ['молоко', 'сыр', 'картошка']
products_2: ['молоко', 'сыр', 'картошка', 'огурцы']
```

Работа со списком - основные функции

<code>list.append(x)</code>	Добавляет элемент в конец списка
<code>list.extend(L)</code>	Расширяет список <code>list</code> , добавляя в конец все элементы списка <code>L</code>
<code>list.insert(i, x)</code>	Вставляет на <code>i</code> -ый элемент значение <code>x</code>
<code>list.remove(x)</code>	Удаляет первый элемент в списке, имеющий значение <code>x</code> . <code>ValueError</code> , если такого элемента не существует
<code>list.pop([i])</code>	Удаляет <code>i</code> -ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
<code>list.index(x, [start [, end]])</code>	Возвращает положение первого элемента со значением <code>x</code> (при этом поиск ведется от <code>start</code> до <code>end</code>)
<code>list.count(x)</code>	Возвращает количество элементов со значением <code>x</code>
<code>list.sort([key=функция])</code>	Сортирует список на основе функции
<code>list.reverse()</code>	Разворачивает список
<code>list.copy()</code>	Поверхностная копия списка
<code>list.clear()</code>	Очищает список

Кортежи

Кортеж - набор элементов, который нельзя изменить

Объявление:

- tuple_1 = (1, 2, 3)
- tuple_1 = tuple(1, 2, 3)

Зачем (чем отличается от списка):

- меньше весят
- нельзя изменять (защищаем от изменений таким образом)

Операции:

- tuple[1] - получение элемента по индексу
- len(tuple) - получение длины
- tuple_1, tuple_2 = tuple_2, tuple_1

Словарь - dict

Словарь - структура для хранения пар ключ-значение

Объявление:

```
: contacts = {  
    "Katyа": "+375-29-111-11-11",  
    "Dima": "+7 000 222-4444"  
}
```

```
: # этот способ объявления словаря используется реже  
contacts = dict(  
    Katyа="+375-29-111-11-11",  
    Dima="+7 000 222-4444"  
)
```

Работа со словарем

- `dict_["key"]` - получение значения по ключу
- `dict_.get("key")` - получение значения по ключу без ошибок, если такого ключа нет
- `dict_.items()` - получение списка пар ключ-значение
- `dict_.keys()` - получить ключи
- `dict_.values()` - получить значения
- `dict_["key"] = "value"` - добавить ключ в словарь
- `dict_.pop("key")` - удалить ключ из словаря и вернуть значение

Set - множество

Set- набор НЕПОВТОРЯЮЩИХСЯ элементов

Объявление:

```
# объявление
my_set = set(["йогурт", "банан", "апельсин"])
my_set

{'апельсин', 'банан', 'йогурт'}
```

```
my_set_1 = set("азбука")
my_set_1

{'а', 'б', 'з', 'к', 'у'}
```

```
# другой вариант объявления
my_set = {"йогурт", "банан", "апельсин"}
my_set

{'апельсин', 'банан', 'йогурт'}
```

Работа со множествами

- `my_set.add("сгущенка")` - добавить элемент в сет
- `"сгущенка" in my_set` - проверить, есть ли элемент в set
- `my_set.remove("сгущенка")` - удалить элемент из set
- `my_set.discard("молоко")` - удалить элемент, при этом не бросает ошибку как remove в случае если элемента в сете нет
- `my_set.pop()` - удалить первый элемент из множества
- `my_set.clear()` - удалить все элементы из множества

```
# метод remove возвращает ошибку если хотим удалить то чего нет
my_set.remove("молоко")
```

```
-----  
KeyError                                         Traceback (most recent call last)  
Cell In[104], line 2  
      1 # метод remove возвращает ошибку если хотим удалить то чего нет  
----> 2 my_set.remove("молоко")  
  
KeyError: 'молоко'
```

frozense

Frozense - то же самое что и set, только в него **НЕЛЬЗЯ ДОБАВЛЯТЬ ЭЛЕМЕНТЫ**, так как он неизменяемый

```
my_set = frozense(["йогурт", "банан", "апельсин"])
my_set
```

```
frozense({'апельсин', 'банан', 'йогурт'})
```

```
my_set.add("апельсин")
```

```
-----  
AttributeError
```

```
Cell In[112], line 1
```

```
----> 1 my_set.add("апельсин")
```

```
Traceback (most recent call last)
```

```
AttributeError: 'frozense' object has no attribute 'add'
```

5

Изменяемые и неизменяемые типы данных

Изменяемые и неизменяемые типы данных

Неизменяемые:

- числа (int, float)
- bool
- кортежи
- строки
- frozenset

Мы не можем обновить их значения, мы можем только переопределить, то есть присвоить переменной новое значение. Новое значение - новая ячейка в памяти

Изменяемые:

- списки
- словари
- множества

Мы можем их изменять, при этом адрес в памяти не меняется. Это опасно тем, что мы передать список в функцию, его там изменят и мы потеряем оригинальный список

Работа с неизменяемыми типами данных

Если мы хотим повлиять на значение неизменяющего типа данных вне функции, то нам нужно **ЯВНО ПРИСВОИТЬ РЕЗУЛЬТАТ**

```
def func(str_):
    print(f"{id(str_)}: адрес до изменения str_")
    str_ = "new_value"
    print(f"{id(str_)}: адрес после изменения str_")
    return str_

my_str = "test"
print(f"{id(my_str)}: адрес оригинальной строки до выполнения функции")
my_str = func(my_str)
print(f"{id(my_str)}: адрес оригинальной строки после выполнения функции")
print(f"{my_str}: строка после выполнения функции")
```

1750832164144: адрес оригинальной строки до выполнения функции

1750832164144: адрес до изменения str_

1750837214704: адрес после изменения str_

1750837214704: адрес оригинальной строки после выполнения функции

new_value: строка после выполнения функции

Работа с изменямыми типами данных

В чем опасность:

Если мы передаем в функцию изменяемый тип данных и меняем его внутри, то это **ИЗМЕНИТ НАШУ ИЗНАЧАЛЬНУЮ ПЕРЕМЕННУЮ ВНЕ ФУНКЦИИ**

```
def func(list_):
    print(f"{id(list_)}: адрес до изменения list_")
    list_.append(4)
    print(f"{id(list_)}: адрес после изменения list_")
    return list_

my_list = [1, 2, 3]
print(f"{id(my_list)}: адрес оригинального списка до выполнения функции")
func(my_list)
print(f"{id(my_list)}: адрес оригинального списка после выполнения функции")
print(f"{my_list}: список после выполнения функции")
```

1750909623040: адрес оригинального списка до выполнения функции

1750909623040: адрес до изменения list_

1750909623040: адрес после изменения list_

1750909623040: адрес оригинального списка после выполнения функции

[1, 2, 3, 4]: список после выполнения функции

Работа с изменямыми типами данных

Как работать:

Чтобы это исправить, нужно передавать в функцию **КОПИЮ ИЗМЕНЯЕМОГО ТИПА ДАННЫХ**

```
def func(list_):
    print(f"{id(list_)}: адрес до изменения list_")
    list_.append(4)
    print(f"{id(list_)}: адрес после изменения list_")
    return list_

my_list = [1, 2, 3]
print(f"{id(my_list)}: адрес оригинального списка до выполнения функции")
func(my_list[:])
print(f"{id(my_list)}: адрес оригинального списка после выполнения функции")
print(f"{my_list}: список после выполнения функции")
```

1750909623168: адрес оригинального списка до выполнения функции

1750910340800: адрес до изменения list_

1750910340800: адрес после изменения list_

1750909623168: адрес оригинального списка после выполнения функции

[1, 2, 3]: список после выполнения функции

Как сделать копию

[:] либо var.copy()

```
list_1 = [1, 2, 3]
print(f"{id(list_1)}: адрес оригинального списка")
list_2 = list_1[:]
print(f"{id(list_2)}: адрес копии 1")
list_3 = list_1.copy()
print(f"{id(list_3)}: адрес копии 2")
```

1750910705216: адрес оригинального списка

1750909618560: адрес копии 1

1750909617856: адрес копии 2

```
set_1 = {1, 2, 3}
print(f"{id(set_1)}: адрес оригинального сета")
set_2 = set_1.copy()
print(f"{id(set_2)}: адрес копии")
```

1750910553440: адрес оригинального сета

1750908856224: адрес копии

6

Как выделяется память

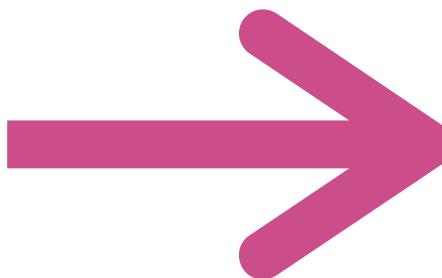
Как выделяется память

number = 1



id = 1

number += 1



id = 2

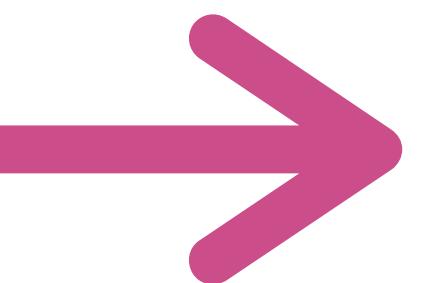
Как выделяется память

string = 'Hello'



id = 1

string = string.capitalize()



id = 2

Как выделяется память для изменяемых типов

animals = ['cat', 'dog'] →

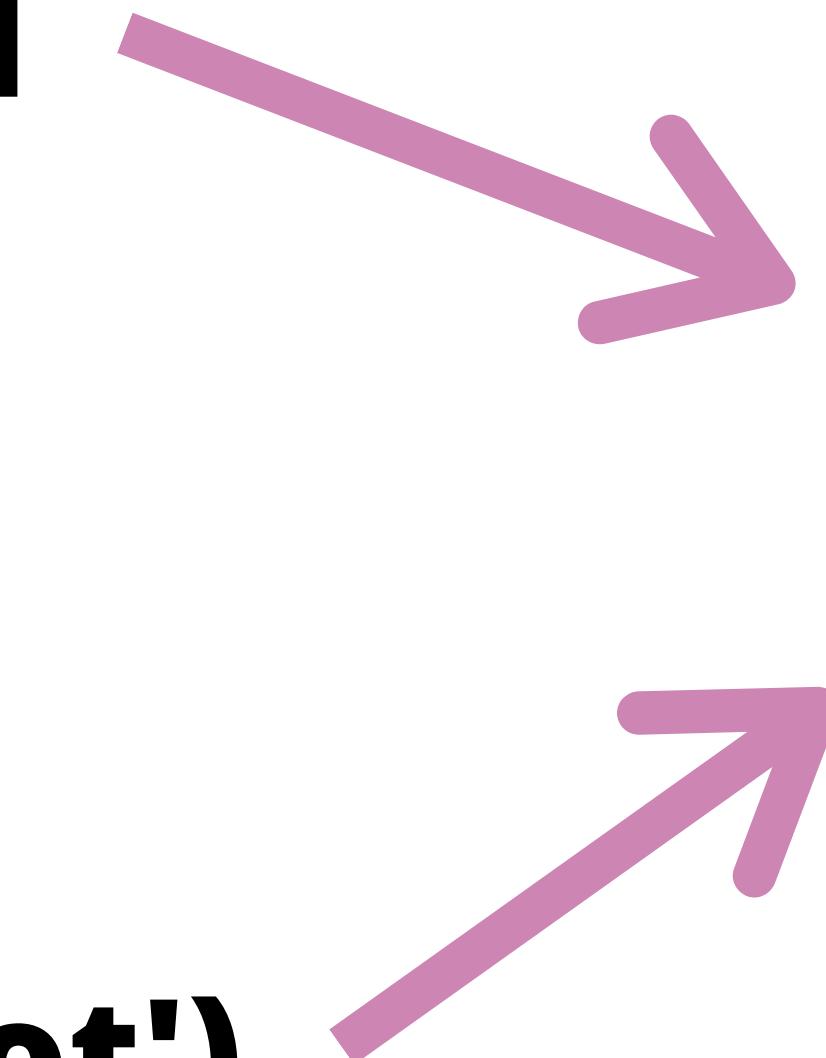
id = 3

Как выделяется память для изменяемых типов

```
animals = ['cat', 'dog']
```

```
animals.append('parrot')
```

```
=> ['cat', 'dog', 'parrot']
```



`id = 3`

Как сравнивать значения

Из этого следует то что если мы хотим сравнить **ЗНАЧЕНИЯ**, то используем оператор `==`, а если хотим сравнить **АДРЕСА В ПАМЯТИ**, то `is`

```
num_1 = 10  
num_2 = 10  
num_1 == num_2
```

True

```
list_1 = [1, 2, 3]  
list_2 = [1, 2, 3]  
list_3 = list_1
```

```
list_1 == list_2
```

True

```
list_1 is list_2
```

False

7

Преобразование типов данных

Преобразование типов

- **str(x)** - привести к строке
- **int(x)** - привести к целому числу
- **float(x)** - привести к числу с точкой
- **list(x)** - привести к списку
- **bool(x)** - привести к boolean (истина или ложь)

`bool(0)` - всегда `False`, отличное от `0` - `True`

`bool = False` когда значения пустые, а именно:

- 0
- пустой список []
- пустая строка ""
- None
- пустой кортеж ()
- пустой set()
- пустой словарь {}

Все что не пустое будет `True!`

8

домашнее задание

Домашнее задание (часть 1)

Часть 1. Работа со списком: Есть список $a = [7, 10, 23, -19, 80]$

- Вывести отсортированный список
- Добавить в список числа 20 и 21
- Снова отсортировать список
- Вывести каждый второй элемент в списке
- Узнать, сколько в списке элементов со значением 20
- Вывести позицию, на которой находится число 20
- Вставить число 19 на позицию, на которой находится число 20

Часть 2. Работа с кортежами: Создать кортеж из трех элементов, получить его длину, значение по индексу 0 и умножить на 2

Домашнее задание (часть 2)

Часть 3. Работа со словарем. Создать словарь, в котором есть ключи имя, возраст, список членов семьи и профессия. Нужно:

- получить имя человека
- обновить профессию
- добавить члена семьи
- вывести список ключей
- добавить в словарь пол человека
- удалить пол

Спасибо за внимание!