План на занятие:

1. Словари
2. Функции
    - 2.1. Создание и вызов функции
    - 2.2. Аргументы
    - 2.3. return
    - 2.4. Инкапсуляция
    - 2.5. Декомпозиция
    - 2.6. Callback
    - 2.7. lambda функции

План на занятие:

1. Словари
2. Функции
    - 2.1. Создание и вызов функции
    - 2.2. Аргументы
    - 2.3. return
    - 2.4. Инкапсуляция
    - 2.5. Декомпозиция
    - 2.6. Callback
    - 2.7. lambda функции

Словари

Словари:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
{key1: value1, key2: value2}
```

Важно, что это не последовательность, т.е. не итерируемый объект, а именно набор, реализованный в виде хэш-таблицы

Ячейка	Данные
...	...
1064708	'Moscow'
...	...
4675752	'WashingtonDC'
...	...

Словари

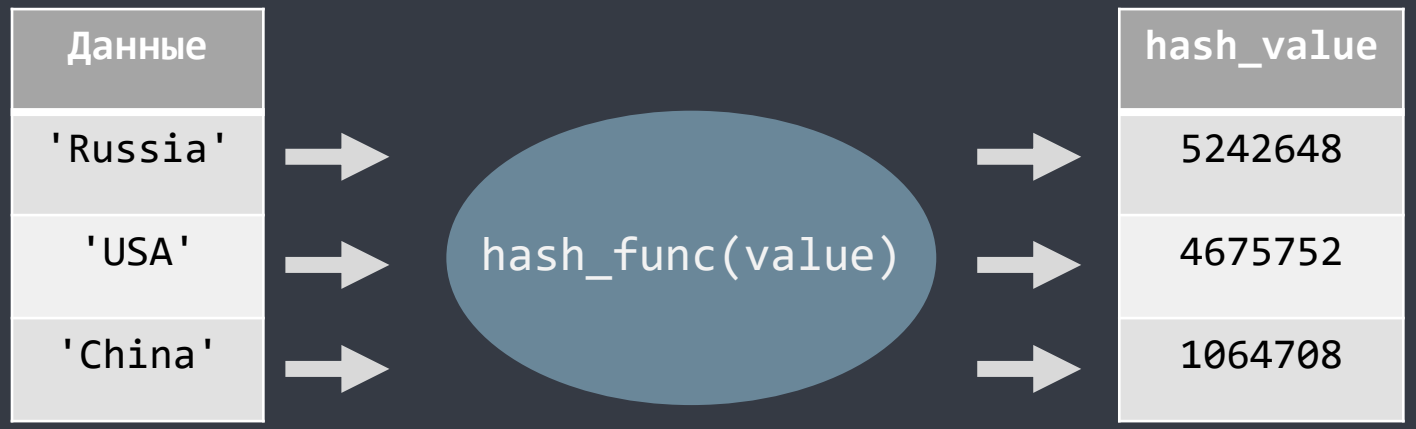
Словари:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
{key1: value1, key2: value2}
```

Важно, что это не последовательность, т.е. не итерируемый объект, а именно набор, реализованный в виде хэш-таблицы



Ячейка	Данные
...	...
1064708	'China'
...	...
4675752	'USA'
...	...

Словари

Словари:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
{key1: value1, key2: value2}
```

Словари

Словари:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
{key1: value1, key2: value2}
```

Пример:

```
1. cities = {'Russia': ['Moscow'], 'USA': ['Washington DC', 'New York']}
   len(cities['Russia'])  # 1
```

Словари

Словари:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
{key1: value1, key2: value2}
```

Пример:

```
2. prices = download_prices('YNDX.ME')
   prices.keys()  # ['close', 'open', 'high', 'low']
   prices['close'] = [60.85, 60.87, 60.76, ...]
```

Словари

Словари:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
{key1: value1, key2: value2}
```

Пример:

```
3. data = {'Russia':
        {'Capital': 'Moscow', 'Population': 140000000}}
   data['Russia']['Capital']  # Moscow
```

Словари

Словари:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
{key1: value1, key2: value2}
```

Пример:

```
3. data = {
        'Russia': {
                'Capital': 'Moscow',
                'Population': 140000000
        }
    }
    data['Russia']['Capital']  # Moscow
```

Словари

Словари:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
{key1: value1, key2: value2}
```

Пример:

```
4. profile = dict(name='Ivan', age=23, skills=['Python', 'Basketball'])
   profile['age'] = 23
```

Типы данных: список (list)

Создание	{k:v}, dict(k=v)
Индексация	l[key]
Проверка наличия ключа/ Итерирование по словарю	key in dictionary # True/False for key in dictionary: ...
Методы d.method(..)	
Список ключей	d.keys()
Список значений	d.values()
Список пар: k,v	d.items()
Извлечение по ключу	d.get(key, None), d.pop(key, None)
Объединение	d.update(new_d)
Изменяемость	d[key] = new_value

План на занятие:

1. Словари
2. Функции
   - 2.1. Создание и вызов функции
   - 2.2. Аргументы
   - 2.3. return
   - 2.4. Инкапсуляция
   - 2.5. Декомпозиция
   - 2.6. Callback
   - 2.7. lambda функции

Функции

y(x) = 2^x

Функции

Функции:

- объект, хранящий в себе участок кода, который можно выполнить, вызвав его по имени

Синтаксис:

```
def foo():
    ____...
```

Пример:

```
1. def do_nothing():
       pass
   do_nothing()
```

Функции

Функции:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
def foo():
    ____...
```

Пример:

```
2. def greeting():
       print('Hi!')
   greeting()  # Hi!
```

План на занятие:

1. Словари
2. Функции
   - 2.1. Создание и вызов функции
   - 2.2. Аргументы
   - 2.3. return
   - 2.4. Инкапсуляция
   - 2.5. Декомпозиция
   - 2.6. Callback
   - 2.7. lambda функции

Функции

Функции:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
def foo():
    ____...
```

Пример:

```
1. def say_hello(name):
       print('Hello ', name)
   say_hello('Ivan')  # Hello, Ivan
   say_hello('Gosha')  # Hello, Gosha
```

План на занятие:

1. Словари
2. Функции
   - 2.1. Создание и вызов функции
   - 2.2. Аргументы
   - 2.3. return
   - 2.4. Инкапсуляция
   - 2.5. Декомпозиция
   - 2.6. Callback
   - 2.7. lambda функции

Функции

Функции:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
def foo():
    ____...
```

Пример:

1. a = type(1)

```
# где-то в исходниках ...
def type(object):
    ...
    return info
```

Функции

Функции:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
def foo():
    ____...
```

Пример:

```
2. def increase(value, by=1):
        if type(by) == type(''):
                by = '_' + by
        return value + b
   sentence = increase('hello', by='world!')  # sentence = 'hello_world!'
```

Функции

Функции:

- тип данных, представляющий из себя набор пар: ключ, значение

Синтаксис:

```
def foo():
____...
```

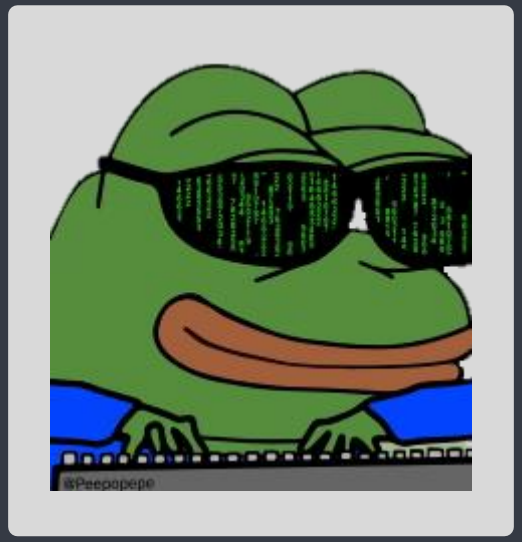
Пример:

```
3. def increase(value, by=1):
        if type(by) == type(''):
                by = '_' + by
        return value + b
   sentence = increase('hello', 'world!')  # sentence = 'hello_world!'
```

Логические выражения

Пример:

1. Функция, решающая квадратное уравнение



Функции

```
def solve_quadratic_equation(a, b, c):
    """Summary on description
    equasion: a*x^2 + b*x + c = 0

    Parameters:
    a (int): value for a coefficient
    b (int): value for b coefficient
    c (int): value for c coefficient

    Returns:
    bool: Has roots
    list: x roots
    """
```

Функции

```
def solve_quadratic_equation(a, b, c):
    d = b**2 - 4*a*c
    if d < 0:
        return False, None
    elif d == 0:
        return True, -b/2*a
    elif d > 0:
        return True, [(-b - d**0.5) / (2*a), (-b + d**0.5) / (2*a)]
    else:
        return False, 'wtf, dude?'
```

План на занятие:

1. Словари
2. Функции
   - 2.1. Создание и вызов функции
   - 2.2. Аргументы
   - 2.3. return
   - 2.4. Инкапсуляция
   - 2.5. Декомпозиция
   - 2.6. Callback
   - 2.7. lambda функции

Функции

Инкапсуляция:

- концепция "хранения кода в капсуле". Внешний код не зависит от происходящего в "капсуле", т.е. функции. Локальная область видимости функции.

Функции

Инкапсуляция:

- концепция "хранения кода в капсуле". Внешний код не зависит от происходящего в "капсуле", т.е. функции. Локальная область видимости функции.

Пример:

1. a = type(1)

```
# где-то в исходниках ...
def type(object):
    ...
    return info
```

Функции

Инкапсуляция:

- концепция "хранения кода в капсуле". Внешний код не зависит от происходящего в "капсуле", т.е. функции. Локальная область видимости функции.

Пример:

```
2. print('Hello')
   # где-то в исходниках ...
   def print(...):
       ...
       ...
```

План на занятие:

1. Словари
2. Функции
    - 2.1. Создание и вызов функции
    - 2.2. Аргументы
    - 2.3. return
    - 2.4. Инкапсуляция
    - 2.5. Декомпозиция
    - 2.6. Callback
    - 2.7. lambda функции

Функции

Декомпозиция:

- разделение целого на части. Заменяем решение одной большой задачи на более маленькие подзадачи, которые тоже могут ветвиться. Получаем дерево подзадач. Каждую из этих подзадач, к примеру, может выполнять какая-то функция

Функции

Декомпозиция, пример:

Задача – узнать, влияет ли этаж квартиры на стоимость жилья и как именно.

```
|- собрать данные
|    |- скачать данные
|    |- "почистить" данные
|
|- "познакомиться" с данными
|    |- посмотреть описательные статистики
|    |- визуализировать данные
|
|- "построить модель"
| ...
```

План на занятие:

1. Словари
2. Функции
   - 2.1. Создание и вызов функции
   - 2.2. Аргументы
   - 2.3. return
   - 2.4. Инкапсуляция
   - 2.5. Декомпозиция
   - 2.6. Callback
   - 2.7. lambda функции

Функции

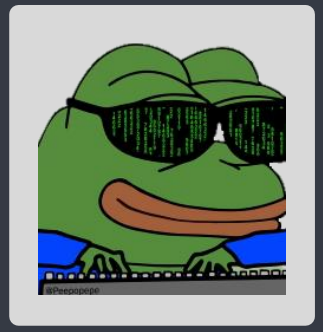
Callback:

- получение внутри функции в качестве аргумента другой функции

Синтаксис:

```
def foo(bar):
____bar()
```

Пример:



План на занятие:

1. Словари
2. Функции
    - 2.1. Создание и вызов функции
    - 2.2. Аргументы
    - 2.3. return
    - 2.4. Инкапсуляция
    - 2.5. Декомпозиция
    - 2.6. Callback
    - 2.7. lambda функции

Функции

`lambda` функции:

- компактный способ создания функций в одну строчку, которые могут не иметь имени

Синтаксис:

```
lambda x: print(x + 1)

foo = lambda x, y: print(x + y)
```

Пример: