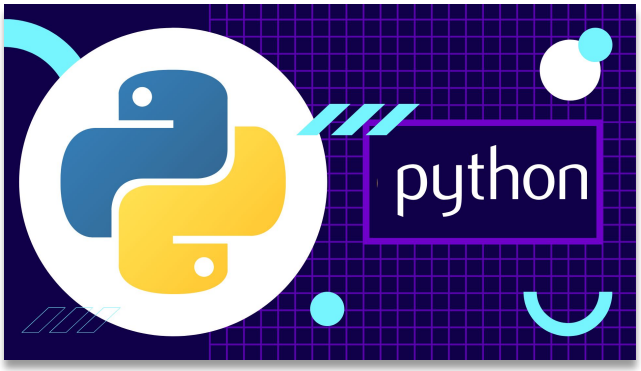
Python-10
Модули и библиотеки.
OS, MATH, PILLOW…

Что такое модуль/библиотека?

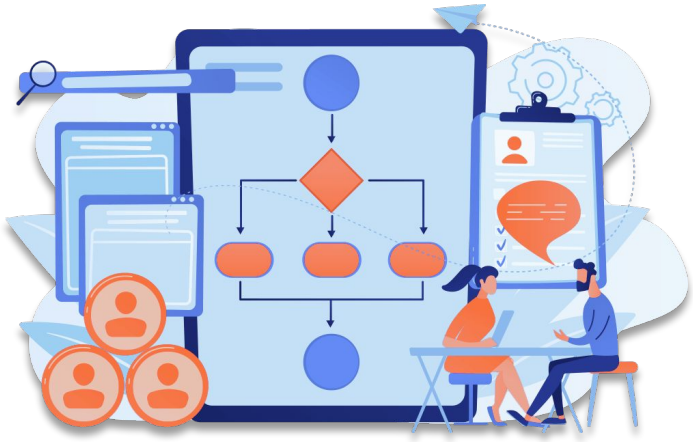
Модуль в Python - это файл (.py), содержащий Python код. Этот код может включать в себя функции, переменные и классы, а также другие инструкции. Модули используются для организации кода в более логические и управляемые блоки. Они позволяют группировать связанный функционал в отдельные файлы, обеспечивая более чистую и структурированную архитектуру программы. (ключевое слово IMPORT)

Библиотека в Python - это коллекция модулей. Она представляет собой совокупность кода, предназначенного для решения определенных задач. Библиотеки включают в себя готовые модули, которые можно использовать в ваших программах. Python имеет обширное количество библиотек для различных целей, таких как обработка данных, веб-разработка, научные вычисления и т. д. (ключевое слово FROM)

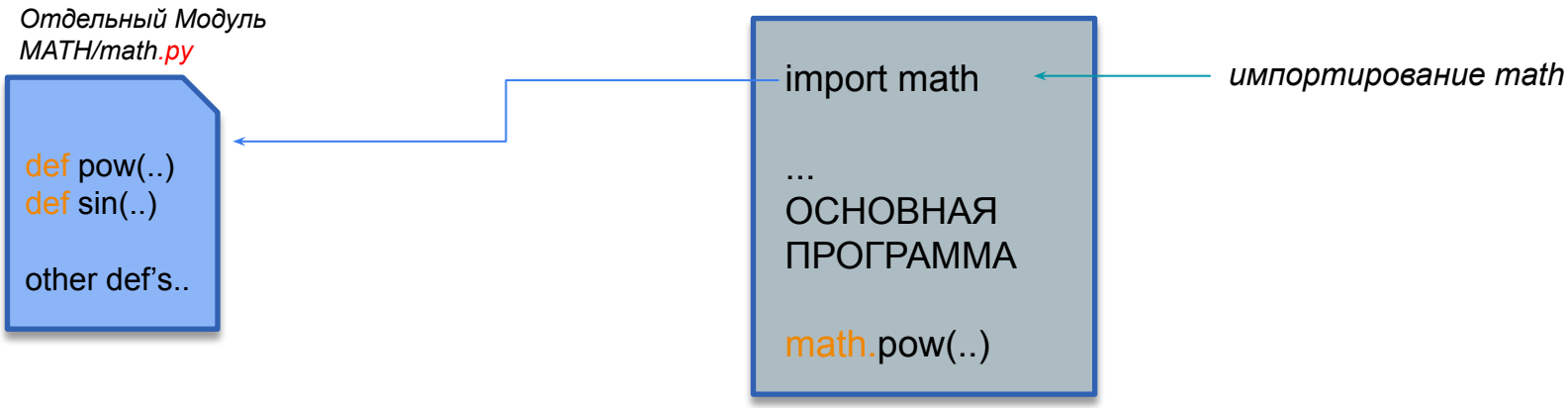


Плюсы использования модулей/библиотек в Python

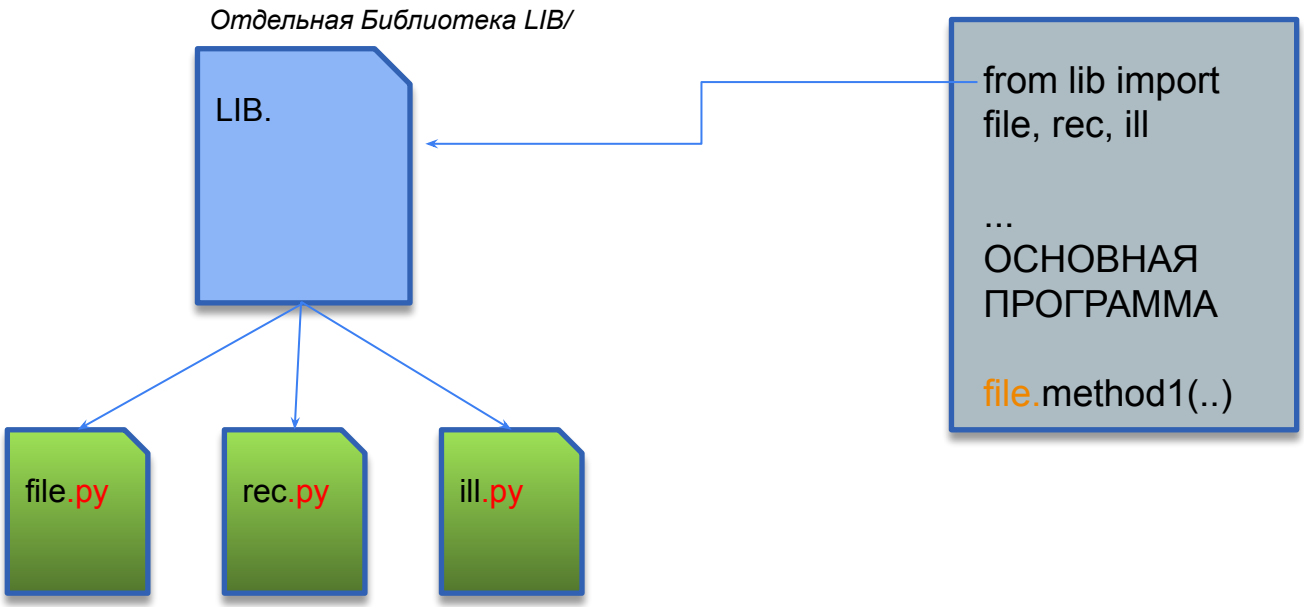
- **Модульность и структурирование кода:** Модули и библиотеки позволяют разбивать код на логические блоки, что упрощает его понимание, сопровождение и повторное использование.
- **Готовый функционал:** Библиотеки предоставляют готовые решения для широкого спектра задач, что позволяет разработчикам использовать проверенный код, экономя время и ресурсы.
- **Сокрытие деталей реализации:** Модули позволяют скрывать детали реализации и предоставлять только необходимый интерфейс, что способствует инкапсуляции и облегчает сопровождение.
- **Поддержка повторного использования:** Модули и библиотеки создают возможность повторного использования кода в различных проектах, что снижает объем работы и ускоряет разработку новых приложений.
- **Совместная работа:** Использование стандартных библиотек и модулей позволяет разработчикам легко обмениваться кодом и совместно работать над проектами.
- **Улучшение поддержки и безопасности:** Многие библиотеки проходят тщательное тестирование и поддерживаются сообществом разработчиков, что повышает качество кода и обеспечивает поддержку в случае



Структура вызова модуля.



Структура вызова библиотеки.



Типы модулей/библиотек.

Модули и библиотеки в Python можно разделить на **стандартные** и **сторонние (внешние).**

Стандартные модули входят в состав стандартной библиотеки Python. Они доступны "из коробки" после установки Python. *Некоторые примеры* стандартных модулей: *os, math, datetime, random, json..*

Сторонние модули и библиотеки разрабатываются сторонними разработчиками и обычно не входят в стандартную библиотеку Python. Их нужно устанавливать дополнительно, например, с помощью инструмента управления пакетами PIP.

Стандартные модули/библиотеки.

Модуль OS.

Модуль os предоставляет функции для взаимодействия с операционной системой, такие как доступ к файловой системе, выполнение команд в командной строке, изменение переменных окружения и другие операции. *Импорт модуля:* import os.

Методы:

- os.getcwd() - возвращает текущую рабочую директорию.
- os.chdir(path) - изменяет текущую рабочую директорию на path.
- os.listdir(path) - возвращает список файлов и директорий в директории, указанной в path.
- os.mkdir(path) - создает директорию с именем path.
- os.makedirs(path) - создает директории в пути path.
- os.remove(file) - удаляет файл с именем file.
- os.rmdir(path) - удаляет директорию с именем path.
- os.removedirs(path) - удаляет директории в пути path.
- os.rename(src, dst) - переименовывает файл или директорию с именем src на имя dst.
- os.path.abspath(path) - возвращает абсолютный путь к файлу или директории.

Модуль MATH.

Модуль math предоставляет математические функции для работы с числами. *Импорт модуля:* import math.

Методы:

- math.ceil(x) - округляет значение x до ближайшего большего целого числа.
- math.floor(x) - округляет значение x до ближайшего меньшего целого числа.
- math.sqrt(x) - возвращает квадратный корень из x.
- math.exp(x) - возвращает экспоненту x (e в степени x), где e - число Эйлера (приблизительно 2,71828).
- math.log(x) - возвращает натуральный логарифм x (с основанием e).
- math.log10(x) - возвращает десятичный логарифм x (с основанием 10).
- math.pow(x, y) - возвращает x в степени y.
- math.pi - константа, которая представляет собой число π (приблизительно 3,14159).
- math.sin(x) - возвращает синус x (x в радианах).
- math.cos(x) - возвращает косинус x (x в радианах).
- math.tan(x) - возвращает тангенс x (x в радианах).

Модуль RANDOM.

Модуль random предоставляет функции для генерации случайных чисел. *Импорт модуля:* import random.

Методы:

```
import random

val = random.randint(1, 10)
print(val)
```

- randint(a, b) - возвращает случайное целое число из диапазона [a, b], включая границы.
- choice(seq) - случайно выбирает элемент из последовательности seq.
- shuffle(seq) - перемешивает элементы последовательности seq в случайном порядке.
- random() - возвращает случайное число с плавающей запятой в диапазоне [0.0, 1.0).
- randrange([start], stop[, step]) - возвращает случайное число из диапазона range([start], stop, [step]). Если указан только один аргумент, то начальное значение принимается за 0 и шаг за 1.
- и др методы.

Модуль JSON.

Модуль json предоставляет функции для работы с форматом данных JSON (JavaScript Object Notation). *Импорт модуля:* import json.

Методы:

- json.dumps(obj) - Преобразование объекта в строку JSON. (Возвращает строку JSON, представляющую объект)
- json.loads(s) - Преобразование строки JSON в объект - Возвращает объект Python, созданный из строки JSON.shuffle(seq) - перемешивает элементы последовательности seq в случайном порядке.
- json.dump(obj, fp) - Запись объекта в файл в формате JSON - Записывает объект в файл в формате JSON.randrange([start], stop[, ...ное число из диапазона range([start], stop, [step]). Если указан только один аргумент, то начальное ...0 и шаг за 1.

```
import json

data = {"ключ": "значение"}
with open("файл.json", "w") as f:
    json.dump(data, f)
```

- ...екта из файла в формате JSON. (Читает объект из файла в формате JSON.)

```
import json

with open("файл.json", "r") as f:
    data = json.load(f)
```

- и др методы

JSON файл пример;

JSON online creator .json files:

https://jsoneditoronline.org/

```
[
  {
    user: "John",
    age: 39,
    is_medic: false,
    email: "john@mail.com"
    children:
    [
      {
        name: "Jessika",
      },
      {
        name: "Robert",
      },
    ]
  },
  {
    city: "London",
    time: "08.22",
    price: null
  }
]
```

Модуль RE.

Модуль re предоставляет функции для работы с регулярными выражениями, что позволяет осуществлять более сложные операции с текстовыми данными. *Импорт модуля:* import re.

Методы:

- re.search(pattern, string) - Поиск первого совпадения (Ищет первое совпадение заданного шаблона в строке.)
- re.match(pattern, string) - Поиск совпадения в начале строки (Проверяет, соответствует ли начало строки заданному шаблону.)
- re.findall(pattern, string) - Поиск всех совпадений (Ищет все совпадения заданного шаблона в строке и возвращает их в виде списка.)
- re.sub(pattern, replacement, string) - Замена совпадений (Заменяет все совпадения заданного шаблона в строке на указанную подстроку.)
-

```
import re

text = "Привет, мир!"
res = re.search(r"\bмир\b", text)
print(res.group())
```

```
мир
```

Модуль DATETIME.

Модуль datetime предоставляет классы для работы с датами и временем. Вот несколько основных методов и классов модуля datetime.
Импорт модуля: from datetime import datetime.

Методы:

- now() - возвращает текущую дату и время в объекте типа datetime.
- date()- возвращает дату в объекте типа date.
- time() - возвращает время в объекте типа time.
- strftime(format) - преобразует объект datetime в строку в соответствии с форматом, указанным в аргументе format.

```
from datetime import datetime

текущее_время = datetime.now()
print(текущее_время)
```

```
from datetime import datetime

дата = datetime(2022, 1, 1)
print(дата)
```

```
from datetime import datetime

текущее_время = datetime.now()
год = текущее_время.year
месяц = текущее_время.month
день = текущее_время.day
час = текущее_время.hour
минута = текущее_время.minute
секунда = текущее_время.second
```

```
from datetime import datetime, timedelta

# разница дат
дата_1 = datetime(2022, 1, 1)
дата_2 = datetime(2022, 1, 10)
разница = дата_2 - дата_1
print(разница.days)
```

Внешние модули/библиотеки.

Установка библиотек.

В Python существует огромное количество сторонних библиотек, которые расширяют возможности языка. Чтобы использовать эти библиотеки, их нужно сначала установить.

Для установки библиотек используется менеджер пакетов pip, который поставляется вместе с Python. Для установки библиотеки нужно выполнить команду:

```
pip install название_библиотеки
```

Например, чтобы установить библиотеку requests для работы с HTTP-запросами, нужно выполнить команду:

```
pip install requests
```

После этого можно использовать функции и классы из библиотеки. Например, чтобы отправить HTTP-запрос на сервер и получить ответ, можно использовать функцию requests.get():

```
import requests

resp = requests.get('https://www.python.org/')
print(resp.status_code)
```

Эта программа отправляет GET-запрос на сайт python.org и выводит на экран код ответа сервера. В данном случае он должен быть равен 200, что означает успешное выполнение запроса.

Модуль REQUESTs.

Модуль requests предоставляет простые и удобные средства для отправки HTTP-запросов и работы с ответами. Она часто используется для взаимодействия с веб-ресурсами. *Импорт модуля:* import requests.

Методы:

- requests.get(url, params=None, args) - Выполнение GET-запроса (Выполняет GET-запрос по указанному URL.)
- requests.post(url, data=None, json=None, args) - Выполнение POST-запроса (Выполняет POST-запрос по указанному URL с передачей данных.)
-
```python
import requests

response = requests.get("https://www.google.com")
if response.status_code == 200:
    print("Успешный запрос!")
    print(response.text)
else:
    print(f"Ошибка запроса: {response.status_code}")
```

- и др методы.

Библиотека Pillow.

Библиотека pillow предоставляет средства для работы с изображениями. Она является форком библиотеки Python Imaging Library (PIL).
Импорт LIB: from PIL import Image.

Методы:

- Image.open() - открыть изображение. <img = Image.open("img.jpg")>
- show() - показать изображение. <img.show()>
- save() - сохранение изображения. <img.save("img_new.jpg")>
- rotate() - поворот изображения. <img2 = img.rotate(90)>
- crop() - обрезка изображения. <img = img.crop((0, 0, img.width/2, img.height/2))>
- resize() - изменение размера изображения. <img = img.resize((img.width//2, img.height//2))>
- reduce(n) - уменьшение в n-раз. <img = img.reduce(2)>
- size() - размер изображения. <img.size()>
- paste() - наложить одно изображение на другое. <img.paste(img2)>
- transpose() - зеркальное отражение согласно параметрам. <img.transpose(Image.Transpose.FLIP_LEFT_RIGHT)>

Библиотека Pillow. Значимые поля.

- **filename:** имя файла или путь к файлу в виде строки
- **format:** формат файла. Если изображение создано самой библиотекой, то имеет значение None.
- **mode:** режим изображения, например, "1", "L", "RGB" или "CMYK". (Полный список форматов доступен в документации)
- **size:** размер в виде кортежа (width, height)
- **width:** ширина
- **height:** высота
- **info:** словарь dict, который хранит дополнительную ассоциированную с файлом информацию
- **is_animated:** представляет булевое значение и равно True, если изображение содержит более одного фрейма. Применяется к анимированным изображениям
- **n_frames:** количество фреймов в изображении. Применяется к анимированным изображениям

Pillow пример #1

```
from PIL import Image, ImageDraw

# Создаем изображение размером 300x200 пикселей
image = Image.new("RGB", (300, 200), "white")

# Создаем объект для рисования на изображении
draw = ImageDraw.Draw(image)

# Рисуем простой прямоугольник
draw.rectangle([50, 50, 250, 150], outline="black", fill="blue")

# Сохраняем изображение в файл
image.save("img.png")
```

Pillow пример #2

```
from PIL import Image

# Открываем изображение
image = Image.open("img.png")

# Изменяем размер изображения
resized_image = image.resize((400, 300))

# Отображаем изображение
resized_image.show()
```

Pillow пример #3

```
from PIL import Image, ImageOps

# Открываем изображение
image = Image.open("image.png")

# Применяем черно-белый фильтр
bw_image = ImageOps.grayscale(image)

# Сохраняем черно-белое изображение в файл
bw_image.save("bw_image.png")
```

Pillow пример #4

```
from PIL import Image, ImageDraw, ImageFont

# Открываем изображение
image = Image.open("image.png")

# Преобразуем изображение в черно-белое
bw_image = image.convert("L")

# Создаем объект для рисования на изображении
draw = ImageDraw.Draw(bw_image)

# Добавляем текст
font = ImageFont.load_default()
text = "Черно-белое изображение с текстом"
draw.text((10, 10), text, font=font, fill=255)  # fill=255 означает белый цвет текста

# Сохраняем изображение с текстом
bw_image.save("bw_image_with_text.png")
```

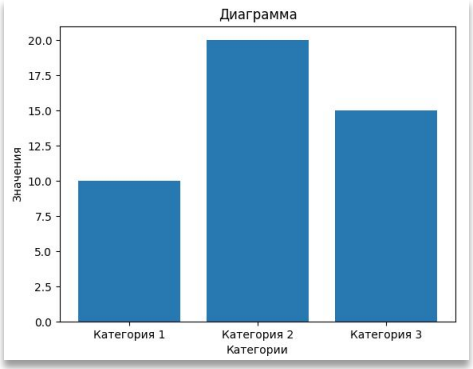
Библиотека MatPlotLib.

Библиотека matplotlib предоставляет средства для визуализации данных в виде графиков и диаграмм. *Импорт LIB:* import matplotlib.pyplot as plt

Методы:

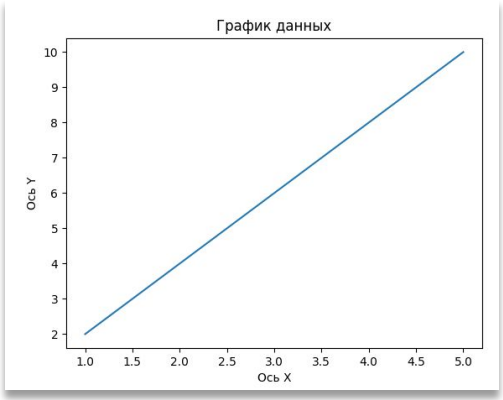
- matplotlib.pyplot.plot: Рисует линейный график.
- matplotlib.pyplot.scatter: Рисует точечный график.
- matplotlib.pyplot.xlabel: Задает подпись оси X.
- matplotlib.pyplot.ylabel: Задает подпись оси Y.
- matplotlib.pyplot.title: Задает заголовок графика.
- matplotlib.pyplot.legend: Добавляет легенду к графику.
- matplotlib.pyplot.grid: Включает сетку на графике.
- matplotlib.pyplot.xlim: Задает пределы по оси X.
- matplotlib.pyplot.ylim: Задает пределы по оси Y.
- matplotlib.pyplot.xticks: Задает метки на оси X.

```python
# диаграмма
import matplotlib.pyplot as plt

категории = ["Категория 1", "Категория 2", "Категория 3"]
значения = [10, 20, 15]

plt.bar(категории, значения)
plt.xlabel("Категории")
plt.ylabel("Значения")
plt.title("Диаграмма")
plt.show()
```

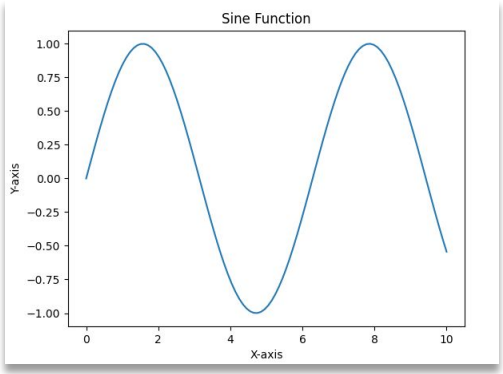


plt пример.

```python
# построение графика
import matplotlib.pyplot as plt

данные_x = [1, 2, 3, 4, 5]
данные_y = [2, 4, 6, 8, 10]

plt.plot(данные_x, данные_y)
plt.xlabel("Ось X")
plt.ylabel("Ось Y")
plt.title("График данных")
plt.show()
```




```python
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace( start: 0,  stop: 10,  num: 100)
y = np.sin(x)

plt.plot( *args: x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Sine Function')
plt.show()
```



Библиотека NumPY.

Библиотека numpy предоставляет поддержку для работы с многомерными массивами и выполнения математических операций над ними. *Импорт LIB:* import numpy as np.

Методы:

- np.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0) - Создает массив.
- np.zeros(shape, dtype=float, order='C') - Создает массив из нулей.
- np.ones(shape, dtype=None, order='C') - Создает массив из единиц.
- np.arange([start,]stop, [step,]dtype=None) - Возвращает массив с равномерно разнесенными значениями в указанном диапазоне.
- np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0) - Возвращает массив с равномерно разнесенными значениями в указанном интервале.
- np.reshape(a, newshape, order='C') - Изменяет форму массива.
- np.transpose(a, axes=None) - Транспонирует массив.
- np.sum(a, axis=None, dtype=None, keepdims=<no value>, initial=<no value>, where=<no value>) - Суммирует значения массива по указанной оси.
- np.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>) - Вычисляет среднее значение массива по указанной оси.
- np.std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>, where=<no value>) - стандартное отклонение массива по указанной оси.

NumPy пример #1

Создание одномерного массива и выполнение математических операций

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
result = arr ** 2 + 10
print(result)
```

```
[11 14 19 26 35]
```

NumPy пример #2

Использование функций для создания массивов

```
import numpy as np

zeros_array = np.zeros(5)
ones_array = np.ones(5)
random_array = np.random.rand(5)
print(zeros_array, ones_array, random_array)
# [0. 0. 0. 0. 0.] [1. 1. 1. 1. 1.] [0.28031105 0.92326794 0.11974611 0.22937439 0.10629041]
```

NumPy пример #3

Выполнение операций с матрицами

```python
import numpy as np

matrix_a = np.array([[1, 2], [3, 4]])
matrix_b = np.array([[5, 6], [7, 8]])
result_matrix = np.dot(matrix_a, matrix_b)
print(result_matrix)
# [[19 22]
#  [43 50]]
```

NumPy пример #4

Использование функций для статистических вычислений

```
import numpy as np

data = np.array([1, 2, 3, 4, 5])
mean_value = np.mean(data)
std_deviation = np.std(data)
print("Mean:", mean_value, "Standard Deviation:", std_deviation)
# Mean: 3.0 Standard Deviation: 1.4142135623730951
```

NumPy пример #5

Работа с индексами и срезами

```
import numpy as np

arr = np.array([0, 1, 2, 3, 4, 5])
subset = arr[2:5]
print(subset)  # [2 3 4]
```

Модули для АИСД.

АИСД модули;

- **collections:** Этот модуль предоставляет специализированные контейнеры данных в дополнение к встроенным контейнерам, таким как списки, словари и кортежи. Например, collections.Counter предоставляет удобный способ подсчета элементов в контейнере, а collections.defaultdict позволяет создавать словари с значениями по умолчанию для отсутствующих ключей.
- **heapq:** Этот модуль реализует кучи (heap), которые являются полезной структурой данных для реализации приоритетных очередей и сортировки кучей. Он предоставляет функции для добавления элементов в кучу, извлечения минимального (или максимального) элемента, а также для преобразования неупорядоченного списка в кучу.
- **queue:** Модуль queue предоставляет различные классы для реализации различных типов очередей, таких как FIFO (First-In-First-Out) и LIFO (Last-In-First-Out). Например, queue.Queue представляет общую FIFO-очередь, а queue.LifoQueue - LIFO-очередь.
- **itertools:** Модуль itertools предоставляет набор функций для эффективной работы с итерируемыми объектами. Он включает функции для комбинаторики, перестановок, комбинаций, фильтрации и многого другого, что делает его полезным инструментом для работы с последовательностями данных.

<collections-defaultdict>

defaultdict - это подкласс словаря, который позволяет указать значение по умолчанию для новых ключей. Это удобно, когда вы хотите избежать проверок наличия ключа перед его использованием.

```
from collections import defaultdict

# Создание defaultdict с значением по умолчанию int
d = defaultdict(int)
d['a'] += 1
d['b'] += 2
print(d)  # defaultdict(<class 'int'>, {'a': 1, 'b': 2})
# Создание defaultdict с значением по умолчанию list
d_list = defaultdict(list)
d_list['a'].append(1)
d_list['b'].append(2)
print(d_list)  # defaultdict(<class 'list'>, {'a': [1], 'b': [2]})
```

<collections-counter>

Counter - это словарь, который позволяет быстро подсчитывать количество элементов в последовательности.

```
from collections import Counter

# Создание Counter из строки
c = Counter('hello')
print(c)  # Counter({'l': 2, 'h': 1, 'e': 1, 'o': 1})

# Подсчет элементов в списке
c = Counter([1, 2, 3, 1, 2, 1, 1])
print(c)  # Counter({1: 4, 2: 2, 3: 1})
```

<collections-deque>

deque - это двусвязанный список, который поддерживает эффективные добавления и удаления как с начала, так и с конца списка.

```
from collections import deque


# Создание deque
d = deque([1, 2, 3])
print(d)  # deque([1, 2, 3])
# Добавление элемента в начало и конец
d.appendleft(0)
d.append(4)
print(d)  # deque([0, 1, 2, 3, 4])
# Извлечение элемента из начала и конца
print(d.popleft())  # 0
print(d.pop())  # 4
print(d)  # deque([1, 2, 3])
```

<collections-namedtuple>

namedtuple - это фабричная функция для создания именованных кортежей, которые могут быть удобны для представления простых структур данных без явного создания класса.

```
from collections import deque


# Создание deque
d = deque([1, 2, 3])
print(d)  # deque([1, 2, 3])
# Добавление элемента в начало и конец
d.appendleft(0)
d.append(4)
print(d)  # deque([0, 1, 2, 3, 4])
# Извлечение элемента из начала и конца
print(d.popleft())  # 0
print(d.pop())  # 4
print(d)  # deque([1, 2, 3])
```

<collections-ordereddict>

OrderedDict - это словарь, который запоминает порядок добавления элементов. Это полезно, когда важен порядок итерации по элементам.

```
from collections import OrderedDict


# Создание OrderedDict
d = OrderedDict()
d['b'] = 2
d['a'] = 1
d['c'] = 3
print(d)  # OrderedDict([('b', 2), ('a', 1), ('c', 3)])
```

<collections-chainmap>

ChainMap - это структура данных, которая позволяет объединять несколько словарей в единое представление, что удобно при работе с настройками или контекстами.

```
from collections import ChainMap

# Создание ChainMap
defaults = {'theme': 'default', 'language': 'english'}
custom = {'language': 'russian'}
settings = ChainMap(custom, defaults)
print(settings['theme'])  # default
print(settings['language'])  # russian
```

<collections-UserDict, UserList, UserString>

- **UserDict** представляет собой обертку вокруг стандартного словаря Python. Он предоставляет интерфейс, похожий на словарь, что позволяет легко создавать подклассы словаря с собственной логикой.
- **UserList** аналогичен UserDict, но для списков. Это обертка вокруг стандартного списка, предоставляющая возможность создавать подклассы списка с собственной логикой.
- **UserString** также аналогичен UserDict и UserList, но для строк. Это обертка вокруг стандартной строки, позволяющая создавать подклассы строки с собственной логикой.

Эти обертки предоставляют удобный способ создания подклассов со своей собственной логикой, расширяя функциональность стандартных типов данных в Python. Они могут быть полезны в различных ситуациях, когда вам нужно добавить дополнительную функциональность к стандартным типам данных.

Схема работы <collections-UserDict>

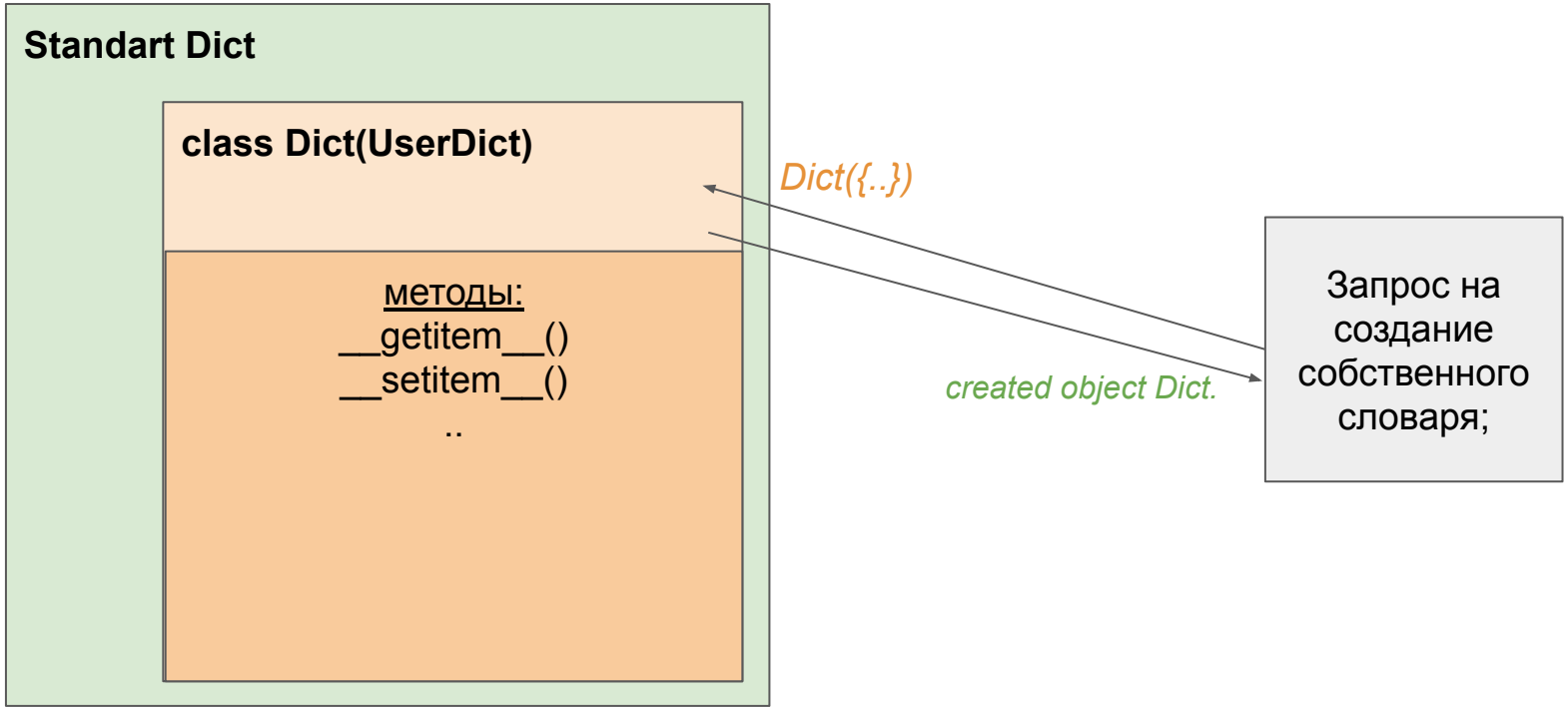
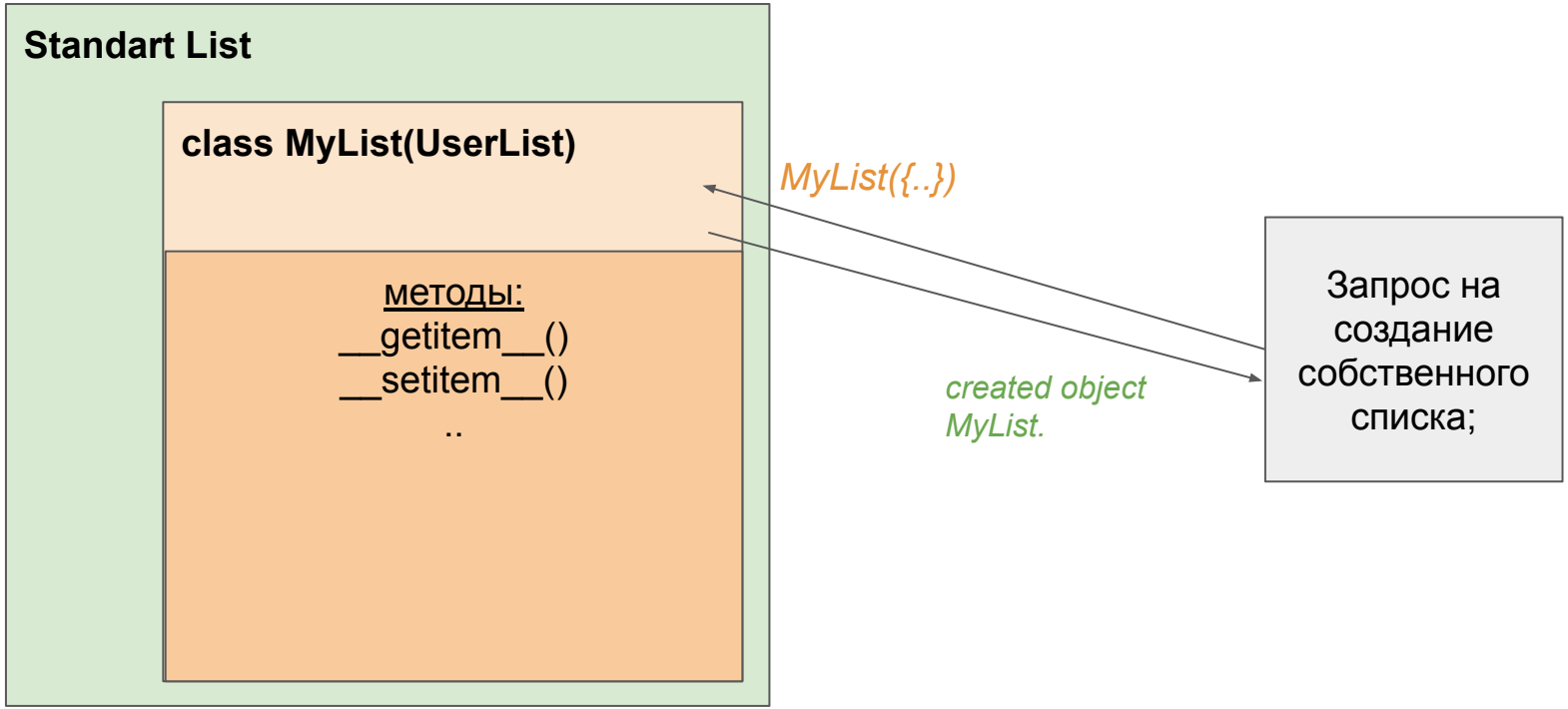


Схема работы <collections-UserList>



пример кода <collections-UserDict>

```
from collections import UserDict


class Dict(UserDict):
    def __init__(self, initial_data=None, **kwargs):
        super().__init__(**kwargs)
        if initial_data:
            self.update(initial_data)

    def __setitem__(self, key, value):
        # Доп.логика при установке элемента
        print('Setting value:', key, value)
        super().__setitem__(key, value)

# Использование подкласса Dict
d = Dict({'a': 1, 'b': 2})
d['c'] = 3  # Setting value: c 3
print(d)  # {'a': 1, 'b': 2, 'c': 3}
```

пример кода <collections-UserList>

```
from collections import UserList


class MyList(UserList):
    def append(self, item):
        # Доп логика при добавлении элемента
        print('Appending item:', item)
        super().append(item)

# Использование подкласса MyList
l = MyList([1, 2, 3])
l.append(4)  # Appending item: 4
print(l)  # [1, 2, 3, 4]
```

пример кода <collections-UserString>

```
from collections import UserString

class MyString(UserString):
   def capitalize(self):
       # Доп логика при вызове метода capitalize
       return '***' + self.data.capitalize() + '***'

# Использование подкласса MyString
s = MyString('hello')
print(s.capitalize())  # ***Hello***
```

<heapq-1>

```
import heapq

# Начальный список
nums = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]

# Преобразование списка в кучу
heapq.heapify(nums)

# Извлечение элементов из кучи (они будут в порядке возрастания)
sorted_nums = []
while nums:
    sorted_nums.append(heapq.heappop(nums))

print(sorted_nums)  # Вывод: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

<heapq-2 search max, min>

```python
import heapq

nums = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]

# Три наименьших элемента
print(heapq.nsmallest(3, nums))  # Вывод: [0, 1, 2]

# Три наибольших элемента
print(heapq.nlargest(3, nums))   # Вывод: [9, 8, 7]
```

<heapq-3>

```python
import heapq

# создание отсортированных последовательностей
# Отсортированные последовательности
nums1 = [1, 3, 5, 7]
nums2 = [2, 4, 6, 8]

# Слияние
merged = list(heapq.merge(nums1, nums2))
print(merged)  # Вывод: [1, 2, 3, 4, 5, 6, 7, 8]
```

<heapq-4>

```
import heapq

# Куча максимального размера 3
h = []
heapq.heappush(h, 1)
heapq.heappush(h, 2)
heapq.heappush(h, 3)
heapq.heappush(h, 4)  # Превышение размера, 1 выбрасывается
print(h)  # Вывод: [3, 4, 2]
```

<heapq-5>

```python
import heapq

class Task:
    def __init__(self, priority, description):
        self.priority = priority
        self.description = description

    def __lt__(self, other):
        return self.priority < other.priority

# Создание списка задач
tasks = [Task(1, 'task1'), Task(3, 'task2'), Task(2, 'task3')]

# Преобразование списка в кучу
heapq.heapify(tasks)

# Обработка задач по приоритету
while tasks:
    task = heapq.heappop(tasks)
    print(f'Обработка задачи: {task.description}')  # Вывод задач в порядке приоритета
```

<queue>

Виды/Разновидности очередей по модулю:

- **Queue:** Этот класс реализует обычную FIFO-очередь. Он поддерживает операции добавления элементов в конец очереди (put()), извлечения элементов из начала очереди (get()), проверки пуста ли очередь (empty()), получения размера очереди (qsize()), блокирования при извлечении элементов из пустой очереди (get(block=True)), установки максимального размера очереди (maxsize) и другие.

- **LifoQueue:** Этот класс реализует стековую LIFO-очередь. Он имеет те же методы, что и класс Queue, но извлекает элементы в порядке Last In, First Out (последним пришел, первым вышел).

- **PriorityQueue:** Этот класс реализует приоритетную очередь, где каждый элемент имеет свой приоритет. При извлечении элементы с наивысшим приоритетом извлекаются первыми. Он также имеет те же методы, что и класс Queue.

<queue-Queue>

```
import queue

# Создание FIFO-очереди
fifo_queue = queue.Queue()
# Добавление элементов
fifo_queue.put(1)
fifo_queue.put(2)
fifo_queue.put(3)
# Извлечение элементов
while not fifo_queue.empty():
   print(fifo_queue.get())  # Вывод: 1, 2, 3
```

<queue-LifoQueue>

```
import queue

# Создание LIFO-очереди
lifo_queue = queue.LifoQueue()

# Добавление элементов
lifo_queue.put(1)
lifo_queue.put(2)
lifo_queue.put(3)

# Извлечение элементов
while not lifo_queue.empty():
    print(lifo_queue.get())  # Вывод: 3, 2, 1
```

<queue-PriorityQueue>

```
import queue

# Создание приоритетной очереди
priority_queue = queue.PriorityQueue()

# Добавление элементов с приоритетом
priority_queue.put((2, 'second'))
priority_queue.put((1, 'first'))
priority_queue.put((3, 'third'))

# Извлечение элементов с наивысшим приоритетом
while not priority_queue.empty():
   print(priority_queue.get())  # Вывод: (1, 'first'), (2,
'second'), (3, 'third')
```

<itertools>

несколько из наиболее часто используемых функций из itertools:

1. **count(start=0, step=1):** Создает бесконечную арифметическую прогрессию, начиная с указанного start и с шагом step.
2. **cycle(iterable):** Бесконечно повторяет элементы из iterable.
3. **repeat(elem, n=None):** Возвращает элемент elem n раз или бесконечно, если n=None.
4. **chain(*iterables):** Объединяет несколько итерируемых объектов в один последовательный итератор.
5. **zip_longest(*iterables, fillvalue=None):** Объединяет элементы из нескольких итерируемых объектов в кортежи. Если длины объектов не совпадают, то используется fillvalue.
6. **product(*iterables, repeat=1):** Возвращает декартово произведение итерируемых объектов.
7. **permutations(iterable, r=None):** Возвращает все возможные перестановки элементов итерируемого объекта.
8. **combinations(iterable, r):** Возвращает все комбинации из r элементов итерируемого объекта без повторений.
9. **combinations_with_replacement(iterable, r):** Возвращает все комбинации из r элементов итерируемого объекта с повторениями.

Пример1 <itertools>

```
import itertools

# Создание бесконечной арифметической прогрессии
for i in itertools.count(1, 2):
    print(i)
    if i >= 10:
        break

# Бесконечное повторение элементов
for i in itertools.cycle('ABC'):
    print(i)
    if i == 'C':
        break

# Повторение элемента 3 раза
for i in itertools.repeat(2, 3):
    print(i)
```

Пример2 <itertools>

```
import itertools

# Объединение нескольких итерируемых объектов
for i in itertools.chain('ABC', 'DEF'):
   print(i)

# Создание всех возможных перестановок
for i in itertools.permutations('ABC', 2):
   print(i)

# Создание всех возможных комбинаций без повторений
for i in itertools.combinations('ABC', 2):
   print(i)

# Создание всех возможных комбинаций с повторениями
for i in itertools.combinations_with_replacement('ABC', 2):
   print(i)
```

Модули bisect, functools, fractions, decimal, operator

существует ряд модулей, которые предоставляют инструменты для работы с алгоритмами:

- **bisect:** Этот модуль реализует алгоритмы для работы с отсортированными последовательностями, такие как вставка элемента в отсортированный список (bisect.insort()), нахождение позиции вставки элемента (bisect.bisect_left() и bisect.bisect_right()).
- **functools:** Модуль functools содержит полезные функции для функционального программирования, включая reduce(), partial(), wraps() и другие.
- **fractions:** Модуль fractions предоставляет поддержку рациональных чисел в Python, что может быть полезным при работе с точными вычислениями.
- **decimal:** Этот модуль предоставляет поддержку десятичных чисел с фиксированной точностью, что может быть полезно для точных финансовых или научных вычислений.
- **operator:** Модуль operator предоставляет функции, эквивалентные встроенным операторам Python, что может быть полезно для работы с коллекциями и функциями высшего порядка.

Пример1 <bisect>

```
#Вставка элемента в отсортированный список:
import bisect

# Отсортированный список
nums = [1, 3, 5, 7, 9]

# Вставка элемента 6
bisect.insort(nums, 6)
print(nums)  # Вывод: [1, 3, 5, 6, 7, 9]
```

Пример2 <bisect>

```
#Нахождение позиции для вставки элемента
import bisect

# Отсортированный список
nums = [1, 3, 5, 7, 9]

# Нахождение позиции для вставки элемента 6
pos = bisect.bisect_left(nums, 6)
print(pos)  # Вывод: 3
```

Пример3 <bisect>

```
#Нахождение позиции для вставки элемента с учетом дубликатов

import bisect

# Отсортированный список с дубликатами
nums = [1, 3, 5, 5, 7, 9]

# Нахождение позиции для вставки элемента 5
pos = bisect.bisect_left(nums, 5)
print(pos)  # Вывод: 2
```

Пример4 <bisect>

```
#Вставка элемента с сохранением порядка сортировки
import bisect

# Отсортированный список строк по длине
words = ['apple', 'banana', 'cherry', 'date']

# Вставка строки 'egg' с сохранением порядка сортировки по длине
bisect.insort(words, 'egg', key=len)
print(words)  # Вывод: ['egg', 'apple', 'date', 'banana', 'cherry']
```

Пример1 <functools>

```
#для создания функции с частично заданными аргументами
from functools import partial

# Обычная функция с двумя аргументами
def power(base, exponent):
    return base ** exponent

# Создание новой функции, которая всегда возводит число в квадрат
square = partial(power, exponent=2)

# Вызов новой функции
result = square(5)
print(result)  # Вывод: 25
```

Пример2 <functools>

```python
#копирования метаданных функции
from functools import wraps

# Декоратор для измерения времени выполнения функции
def measure_time(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        import time
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"Execution time of {func.__name__}: {end_time - start_time} seconds")
        return result
    return wrapper

# Использование декоратора
@measure_time
def some_function(n):
    return sum(range(n))
# Вызов функции
result = some_function(1000000)
print(result)  # Вывод: 499999500000
```

Пример <fractions>

```
from fractions import Fraction

# Создание двух рациональных чисел
frac1 = Fraction(1, 2)
frac2 = Fraction(3, 4)
# Сложение рациональных чисел
result = frac1 + frac2
print(result)  # Вывод: 5/4 (1 целая и 1/4)


# Список рациональных чисел
nums = [Fraction(1, 2), Fraction(2, 3), Fraction(3, 4)]
# Вычисление среднего арифметического
average = sum(nums, Fraction(0, 1)) / len(nums)
print(average)  # Вывод: 11/6 (1 целая и 5/6)
```

Пример1 <decimal>

```
#Выполнение точных арифметических операций с десятичными числами
from decimal import Decimal

# Создание десятичных чисел
num1 = Decimal('0.1')
num2 = Decimal('0.2')

# Сложение десятичных чисел
result = num1 + num2
print(result)  # Вывод: 0.3
```

Пример2 <decimal>

```
# Выполнение действий с точным округлением
from decimal import Decimal, ROUND_HALF_UP

# Создание десятичного числа
num = Decimal('1.225')

# Округление до двух знаков после запятой
rounded_num = num.quantize(Decimal('0.01'), rounding=ROUND_HALF_UP)
print(rounded_num)  # Вывод: 1.23
```

Пример3 <decimal>

```
#Решение финансовых задач с точным представлением денежных величин
from decimal import Decimal

# Создание десятичных чисел для суммы платежей
payment1 = Decimal('45.67')
payment2 = Decimal('89.95')
payment3 = Decimal('12.34')

# Вычисление общей суммы платежей
total_payment = payment1 + payment2 + payment3
print(total_payment)  # Вывод: 148.96
```

Пример1 <operator>

```
#Сортировка списка словарей по ключу
import operator

# Список словарей
data = [
    {'name': 'John', 'age': 30},
    {'name': 'Alice', 'age': 25},
    {'name': 'Bob', 'age': 35}
]

# Сортировка по ключу 'age'
sorted_data = sorted(data, key=operator.itemgetter('age'))
print(sorted_data)  # Вывод:
# [{'name': 'Alice', 'age': 25}, {'name': 'John', 'age': 30}, {'name': 'Bob', 'age': 35}]
```

Пример2 <operator>

```
#Вычисление суммы элементов списка
import operator

# Список чисел
nums = [1, 2, 3, 4, 5]

# Вычисление суммы с использованием функции operator.add
total = reduce(operator.add, nums)
print(total)  # Вывод: 15
```

Пример3 <operator>

```
#Получение атрибута объекта
import operator

# Простой класс с атрибутами
class Person:
   def __init__(self, name, age):
       self.name = name
       self.age = age

# Создание объектов
person1 = Person('Alice', 30)
person2 = Person('Bob', 25)

# Получение значения атрибута 'age'
get_age = operator.attrgetter('age')
# Получение и печать значений атрибутов 'age' для объектов
print(get_age(person1))  # Вывод: 30
print(get_age(person2))  # Вывод: 25
```

Telebot. Создание телеграм бота.

Telebot. Установка.

Для установки стороннего модуля Telebot вам нужно выполнить следующие шаги:

- Проверить что установлен Python актуальной версии (не Python0.1, 0.2..).
- Открыть командную строку (cmd/terminal). На Windows можно открыть ее, нажав Win+R и набрав cmd.
- Установить Telebot при помощи специальной команды **pip install telebot; pyTelegramBotAPI**

Что такое ТОКЕН и как его создать?

Telegram Bot API использует токен для аутентификации вашего бота и взаимодействия с API. **Токен** - это уникальная строка символов, которая выдается при создании бота через официального бота Telegram, известного как BotFather.

Вот шаги по созданию токена:

- Откройте Telegram и найдите BotFather (*https://t.me/BotFather*).
- Начните чат с BotFather, нажав кнопку **"Start"**.
- Используйте команду **/newbot**, чтобы создать нового бота.
- Следуйте инструкциям BotFather, предоставляя ему информацию о боте, такую как его имя и уникальное имя пользователя.
- В конце процесса BotFather предоставит вам уникальный токен для вашего бота.

Токен выглядит примерно так: *1234567890:ABCdefGHIjKlMnOpQrStUvWxYz*. Этот токен необходим для идентификации вашего бота при отправке запросов к Telegram Bot API.

Telebot simple code

```python
import telebot

# 'YOUR_BOT_TOKEN' ваш токен
bot = telebot.TeleBot('YOUR_BOT_TOKEN')

# bot.message_handler - это декоратор, который привязывает функцию
# (в данном случае, echo_all) к обработке входящих сообщений
# в вашем боте с использованием библиотеки Telebot.
@bot.message_handler(func=lambda message: True)
def echo_all(message):
    #
    bot.reply_to(message, message.text)


# Запуск бота
bot.polling(none_stop=True)
```

Telebot. Декораторы.

message_handler

@bot.message_handler(func=None, content_types=None, regexp=None, commands=None)

func: Функция-обработчик сообщения.
content_types: Типы контента, которые бот будет обрабатывать (например, 'text', 'audio', 'document' и т.д.).
regexp: Регулярное выражение для фильтрации сообщений.
commands: Список команд, которые бот будет обрабатывать.

```
@bot.message_handler(func=lambda message: True)
def echo_all(message):
    bot.reply_to(message, message.text)
```

callback_query_handler

@bot.callback_query_handler(func=None)

func: Функция-обработчик для callback-запросов.

```python
@bot.callback_query_handler(func=lambda call: True)
def handle_callback(call):
    bot.send_message(call.message.chat.id, "Callback received!")
```

inline_handler

@bot.inline_handler(func=None)

func: Функция-обработчик для inline-запросов.

```python
@bot.inline_handler(func=lambda query: True)
def handle_inline(query):
    results = []
    # Добавление результатов inline-запроса в список results
    bot.answer_inline_query(query.id, results)
```

edited_message_handler

@bot.edited_message_handler(func=None, content_types=None)

func: Функция-обработчик для отредактированных сообщений.
content_types: Типы контента, которые бот будет обрабатывать.

```python
@bot.edited_message_handler(func=lambda message: True)
def handle_edited_message(message):
    bot.send_message(message.chat.id, "Edited message received!")
```

обработка изображения

Нет прямого декоратора обработки изображения, но можно использовать типы контента сообщений, с дальнейшей их обработкой и отправкой текстового ответа например.

```python
import telebot

bot = telebot.TeleBot('YOUR_BOT_TOKEN')


@bot.message_handler(content_types=['photo'])
def handle_images(message):
    # Обработка входящего изображения
    file_id = message.photo[-1].file_id
    file_info = bot.get_file(file_id)
    file_path = file_info.file_path

    # код обработки изображения

    # Отправка ответа
    bot.reply_to(message, "Изображение получено и обработано!")


# Запуск бота
bot.polling(none_stop=True)
```

Telebot. Методы.

send_message

bot.send_message(chat_id, text, parse_mode=None, disable_web_page_preview=None, disable_notification=None, reply_to_message_id=None, reply_markup=None)

chat_id: Идентификатор чата, куда отправляется сообщение.
text: Текст сообщения.
parse_mode: Режим разбора текста (например, "Markdown").
disable_web_page_preview: Отключает предварительный просмотр веб-страниц в сообщении.
disable_notification: Отправляет сообщение без звуковых уведомлений.
reply_to_message_id: Идентификатор сообщения, на которое следует отвечать.
reply_markup: Дополнительные параметры для клавиатуры или меню.

```
chat_id = 123456789  # id
bot.send_message(chat_id, "Привет, мир!")
```

edit_message_text

bot.edit_message_text(text, chat_id=None, message_id=None, inline_message_id=None, parse_mode=None, disable_web_page_preview=None, reply_markup=None)

text: Новый текст сообщения.
chat_id: Идентификатор чата.
message_id: Идентификатор редактируемого сообщения.
inline_message_id: Идентификатор сообщения в inline-режиме.
parse_mode: Режим разбора текста.
disable_web_page_preview: Отключает предварительный просмотр веб-страниц в сообщении.
reply_markup: Дополнительные параметры для клавиатуры или меню.

```
chat_id = 123456789  # идентификатор чата
message_id = 42  # идентификатор сообщения
bot.edit_message_text("Новый текст сообщения", chat_id=chat_id, message_id=message_id)
```

send_photo

bot.send_photo(chat_id, photo, caption=None, parse_mode=None, disable_notification=None, reply_to_message_id=None, reply_markup=None)

chat_id: Идентификатор чата, куда отправляется фото.
photo: Файл фотографии (может быть объектом файла или строкой с идентификатором файла).
caption: Описание фотографии.
parse_mode: Режим разбора текста.
disable_notification: Отправляет фото без звуковых уведомлений.
reply_to_message_id: Идентификатор сообщения, на которое следует отвечать.
reply_markup: Дополнительные параметры для клавиатуры или меню.

```
chat_id = 123456789  # идентификатор чата
photo = open('path/to/photo.jpg', 'rb')
bot.send_photo(chat_id, photo, caption="Описание фотографии")
```

send_document

bot.send_document(chat_id, document, caption=None, parse_mode=None, disable_notification=None, reply_to_message_id=None, reply_markup=None)

chat_id: Идентификатор чата, куда отправляется документ.
document: Файл документа (может быть объектом файла или строкой с идентификатором файла).
caption: Описание документа.
parse_mode: Режим разбора текста.
disable_notification: Отправляет документ без звуковых уведомлений.
reply_to_message_id: Идентификатор сообщения, на которое следует отвечать.
reply_markup: Дополнительные параметры для клавиатуры или меню.

```
chat_id = 123456789  # идентификатор чата
document = open('path/to/document.txt', 'rb')  # путь к документу
bot.send_document(chat_id, document, caption="Описание документа")
```

reply_to

bot.reply_to(message, text, parse_mode=None, disable_web_page_preview=None, disable_notification=None, reply_markup=None)

message: Объект сообщения, на которое следует отвечать.
text: Текст ответного сообщения.
parse_mode: Режим разбора текста.
disable_web_page_preview: Отключает предварительный просмотр веб-страниц в сообщении.
disable_notification: Отправляет ответ без звуковых уведомлений.
reply_markup: Дополнительные параметры для клавиатуры или меню.

```
bot.reply_to(message, "Ваш ответ: " + message.text)
```

get_me

bot.get_me()

Параметров нет.

Данный метод - возвращает информацию о вашем боте, включая его имя пользователя, идентификатор и другие детали.

```
bot_info = bot.get_me()
print(bot_info.username)
```

get_chat

bot.get_chat(chat_id)

chat_id: Идентификатор чата.

Данный метод - возвращает объект чата по его идентификатору. (*title - заголовок)

```
chat_id = 123456789  # идентификатор чата
chat_info = bot.get_chat(chat_id)
print(chat_info.title)
```

get_chat_member

bot.get_chat_member(chat_id, user_id)

chat_id: Идентификатор чата.
user_id: Идентификатор пользователя в чате.

Данный метод - возвращает информацию о пользователе в чате.

```
chat_id = 123456789  # идентификатор чата
user_id = 987654321  # идентификатор пользователя
chat_member_info = bot.get_chat_member(chat_id, user_id)
print(chat_member_info.status)
```

get_updates

bot.get_updates(offset=None, limit=None, timeout=20, allowed_updates=None)

offset: Идентификатор обновления, начиная с которого нужно получить обновления.
limit: Количество обновлений, которое нужно получить (по умолчанию 100).
timeout: Тайм-аут для ожидания обновлений в секундах.
allowed_updates: Список типов обновлений, которые разрешены.

Данный метод - возвращает список объектов обновлений (сообщений, inline-запросов и др.).

```
updates = bot.get_updates()
for update in updates:
    print(update.message.text)
```

set_webhook

bot.set_webhook(url=None, certificate=None)

url: URL, который будет использоваться для вебхука.
certificate: Путь к сертификату для использования HTTPS.

Устанавливает вебхук для бота. Вебхук - это механизм, при котором Telegram отправляет обновления боту, когда они доступны, вместо того чтобы опрашивать сервера Telegram.

```
webhook_url = "https://your_domain.com/your_webhook_endpoint"
bot.set_webhook(url=webhook_url)
```

delete_webhook

bot.delete_webhook()
Нет параметров.

Удаляет вебхук для бота.

```
bot.delete_webhook()
```

Пример onClick-{hello}

```
import telebot
from telebot import types

TOKEN = '...'
bot = telebot.TeleBot(TOKEN)

# Обработчик события нажатия на кнопку
@bot.message_handler(commands=['start'])
def send_welcome(message):
    # Создание клавиатуры с кнопкой
    keyboard = types.ReplyKeyboardMarkup(row_width=1, resize_keyboard=True)
    button = types.KeyboardButton("Нажми меня!")
    keyboard.add(button)
    # Отправка сообщения с клавиатурой
    bot.send_message(message.chat.id,"Привет! Нажми кнопку!", reply_markup=keyboard)

# Обработчик события нажатия на кнопку
@bot.message_handler(func=lambda message: True)
def echo_message(message):
    if message.text == "Нажми меня!":
        # Отправка текстового сообщения при нажатии на кнопку
        bot.send_message(message.chat.id,"Вы нажали кнопку!")
# Запуск бота
bot.polling()
```

Пример onClick-{load file} part1

```
import telebot
from telebot import types

# Токен вашего бота
TOKEN = '...'

# Создание экземпляра бота
bot = telebot.TeleBot(TOKEN)

# Обработчик команды /start
@bot.message_handler(commands=['start'])
def send_welcome(message):
    bot.send_message(message.chat.id,"Привет! Отправь мне текстовый файл .txt")
```

Пример onClick-{load file} part2

```
...
@bot.message_handler(content_types=['document'])
def handle_text_document(message):
    # Проверка расширения файла
    if message.document.file_name.endswith('.txt'):
        try:
            # Загрузка файла
            file_info = bot.get_file(message.document.file_id)
            downloaded_file = bot.download_file(file_info.file_path)
            # Обработка содержимого файла
            file_content = downloaded_file.decode('utf-8')  # Декодирование байтов в строку
            # Отправка содержимого файла в качестве ответа
            bot.reply_to(message, f"Содержимое файла: \n{file_content}")
        except Exception as e:
            print(f"Ошибка обработки файла: {e}")
            bot.reply_to(message, "Произошла ошибка при обработке файла. ")
    else:
        bot.reply_to(message, "Пожалуйста, отправьте текстовый файл с расширением .txt. ")

# Запуск бота
bot.polling()
```

Пример onClick-{load image}

```
# Обработчик команды /start
@bot.message_handler(commands=['start'])
def send_welcome(message):
    bot.send_message(message.chat.id, "Привет! Отправь мне изображение.")

# Обработчик изображений
@bot.message_handler(content_types=['photo'])
def handle_photo(message):
    try:
        # Получение информации о файле
        file_id = message.photo[-1].file_id
        file_info = bot.get_file(file_id)
        # Загрузка файла
        downloaded_file = bot.download_file(file_info.file_path)
        # Сохранение изображения
        file_name = f"photo_{file_id}.jpg"
        with open(file_name, 'wb') as new_file:
            new_file.write(downloaded_file)
        # Отправка подтверждения
        bot.reply_to(message, "Изображение получено и сохранено.")
    except Exception as e:
        print(f"Ошибка обработки изображения:{e}")
        bot.reply_to(message, "Произошла ошибка при обработке изображения.")
# Запуск бота
bot.polling()
```

Пример tests-questions part1

```
import telebot
from telebot import types

# Токен вашего бота
TOKEN = 'YOUR_BOT_TOKEN'

# Создание экземпляра бота
bot = telebot.TeleBot(TOKEN)

# Словарь с вопросами и правильными ответами
questions = {
    "Какая столица Франции?": "Париж",
    "Какой год появления Python?": "1991",
    "Сколько планет в Солнечной системе?": "8"
}
```

Пример tests-questions part2

```
# Обработчик команды /start
@bot.message_handler(commands=['start'])
def send_question(message):
    for question_text in questions.keys():
        # Создание клавиатуры с вариантами ответов
        keyboard = types.InlineKeyboardMarkup()
        keyboard.add(types.InlineKeyboardButton("Париж", callback_data=f"{question_text}:Париж"),
                     types.InlineKeyboardButton("Москва", callback_data=f"{question_text}:Москва"))
        # Отправка вопроса с вариантами ответов
        bot.send_message(message.chat.id, question_text, reply_markup=keyboard)

# Обработчик нажатия на кнопку
@bot.callback_query_handler(func=lambda call: True)
def handle_answer(call):
    # Разбор данных из callback_data
    question, answer = call.data.split(':')
    # Проверка правильности ответа и отправка сообщения
    if answer == questions[question]:
        bot.send_message(call.message.chat.id, f"Правильно! {question}")
    else:
        bot.send_message(call.message.chat.id, f"Неправильно! {question}")
# Запуск бота
bot.polling()
```

Пример Простой калькулятор

```
import telebot

TOKEN = '...'
bot = telebot.TeleBot(TOKEN)

# Обработчик команды /start
@bot.message_handler(commands=['start'])
def send_welcome(message):
    bot.reply_to(message, "Привет! Я калькулятор-бот. Отправь мне математическое выражение.  ")

# Обработчик текстовых сообщений с математическими выражениями
@bot.message_handler(func=lambda message: True)
def calculate_expression(message):
    try:
        # Вычисление выражения
        result = eval(message.text)
        # Отправка результата
        bot.reply_to(message, f"Результат: {result}")
    except Exception as e:
        bot.reply_to(message, f"Ошибка: {e}")
# Запуск бота
bot.polling()
```

Пример Самый простой ToDo Manager part1

```
import telebot
import json

TOKEN = '...'
bot = telebot.TeleBot(TOKEN)
# Файл для хранения задач
TASKS_FILE = 'tasks.json'

# Загрузка задач из файла (если файл существует)
try:
    with open(TASKS_FILE, 'r') as file:
        tasks = json.load(file)
except FileNotFoundError:
    tasks = {}

# Обработчик команды /start
@bot.message_handler(commands=['start'])
def send_welcome(message):
    bot.send_message(message.chat.id, "Привет! Это бот-менеджер задач. Используйте команды /add и /list.")
```

Пример Самый простой ToDo Manager part2

```
# Обработчик команды /add
@bot.message_handler(commands=['add'])
def add_task(message):
    try:
        # Получение текста задачи из сообщения
        task_text = message.text.split(maxsplit=1)[1]
        # Добавление задачи в словарь
        tasks[message.chat.id] = tasks.get(message.chat.id, []) + [task_text]
        # Сохранение задач в файл
        with open(TASKS_FILE, 'w') as file:
            json.dump(tasks,file)
        bot.reply_to(message, "Задача добавлена!")
    except IndexError:
        bot.reply_to(message, "Используйте команду в формате: /add Новая задача")
# Обработчик команды /list
@bot.message_handler(commands=['list'])
def list_tasks(message):
    user_tasks = tasks.get(message.chat.id, [])
    if user_tasks:
        task_list = '\n'.join([f"{index + 1}. {task}" for index, task in enumerate(user_tasks)])
        bot.send_message(message.chat.id, "Список ваших задач:\n" + task_list)
    else:
        bot.send_message(message.chat.id, "У вас пока нет задач!")
# Запуск бота
bot.polling()
```