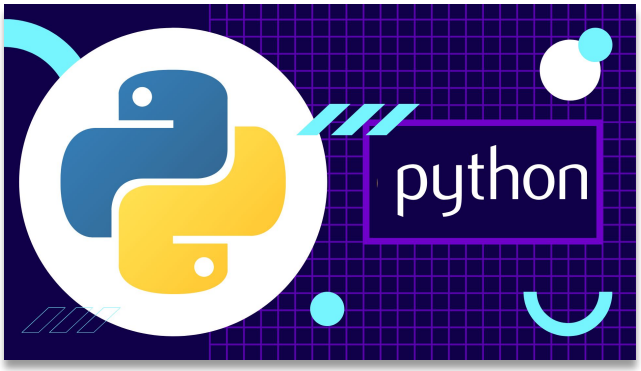
Python-10
Модули и библиотеки.
OS, MATH, PILLOW…

Что такое модуль/библиотека?

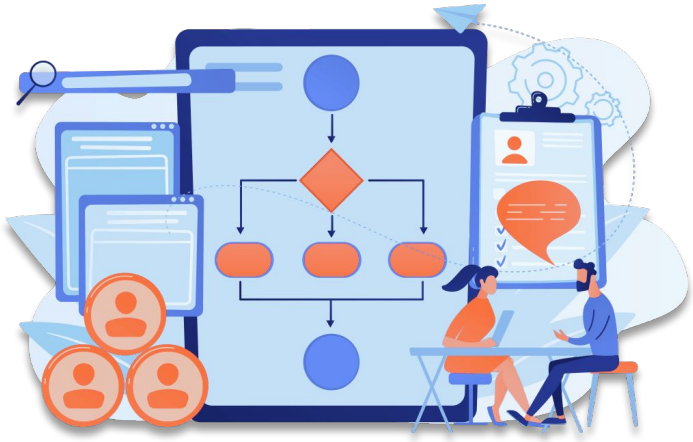
Модуль в Python - это файл (.py), содержащий Python код. Этот код может включать в себя функции, переменные и классы, а также другие инструкции. Модули используются для организации кода в более логические и управляемые блоки. Они позволяют группировать связанный функционал в отдельные файлы, обеспечивая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

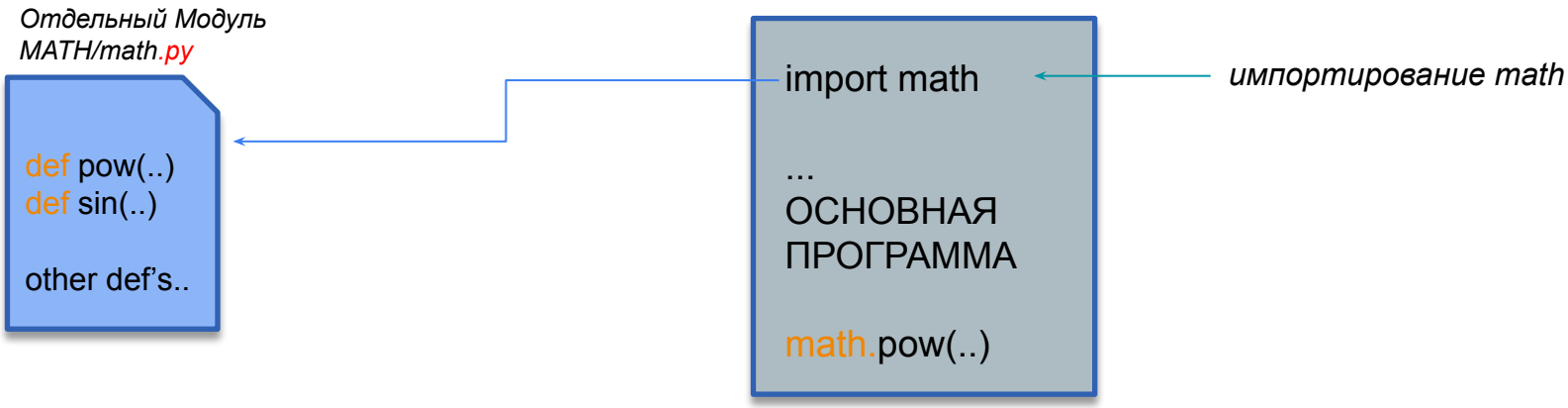


Плюсы использования модулей/библиотек в Python

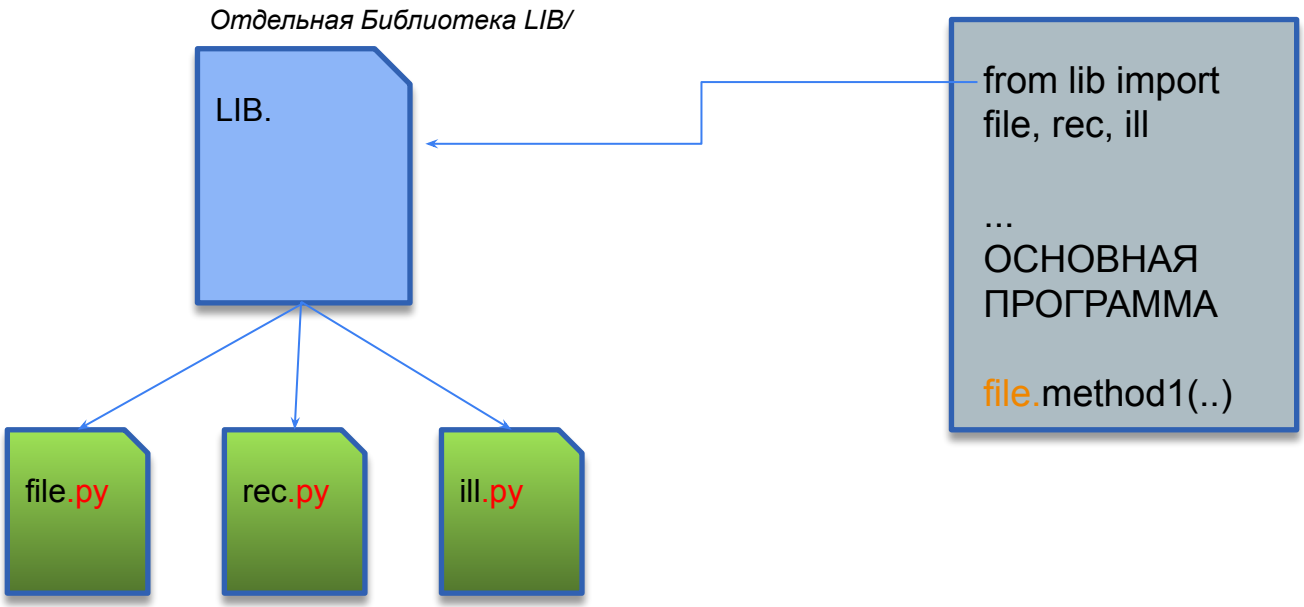
- **Модульность и структурирование кода:** Модули и библиотеки позволяют разбивать код на логические блоки, что упрощает его понимание, сопровождение и повторное использование.
- **Готовый функционал:** Библиотеки предоставляют готовые решения для широкого спектра задач, что позволяет разработчикам использовать 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

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")
```

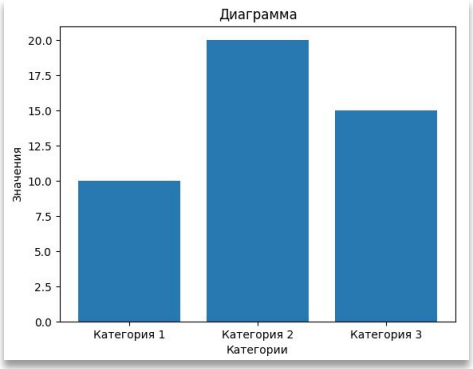
Библиотека MatPlotLib.

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

Методы:

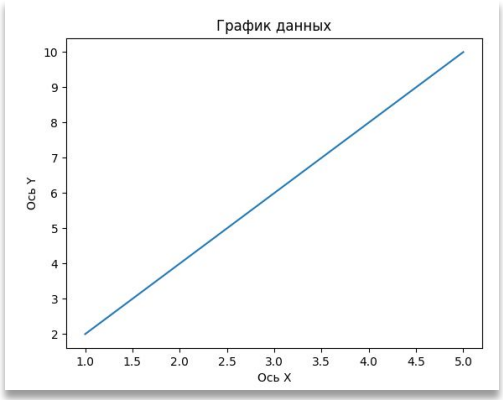
- matplotlib.pyplot.plot: Рисует линейный график.
- matplotlib.pyplot.scatter: Рисует точечный график.
- matplotlib.pyplot.xlabel: Задает подпись оси X.
- matplotlib.pyplot.ylabel: Задает подпись оси Y.
- matplotlib.pyplot.title: Задает заголовок графика.
- matplotlib.pyplot.legend: Добавляет легенду 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()
```

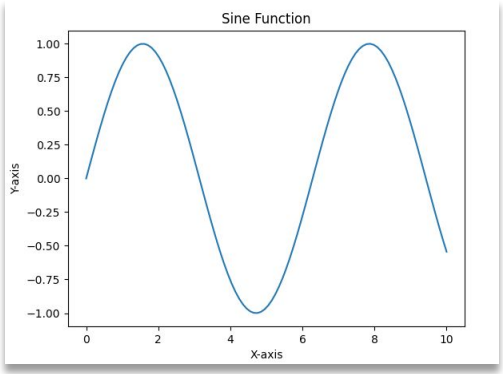


plt пример.

```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]
```

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()
```