

# Инфраструктура вычислений в биоинформатике

## Лекция 2. Скрэпинг. Кроулинг. API.

FastAPI. Aria и работа с большими  
данными.



# Парсинг данных из интернета

Самый простой случай автоматической обработки данных - умение парсить HTML-сайты.

Элементы разметки: теги; скрипты JS; скрытые посредством CSS теги

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Example</title>
5     <link rel="stylesheet" href="style.css">
6   </head>
7   <body>
8     <h1>
9       <a href="/">Header</a>
10    </h1>
11    <nav>
12      <a href="one/">One</a>
13      <a href="two/">Two</a>
14      <a href="three/">Three</a>
15    </nav>
```

```
.nav {
  background-color: #4F4D53;
  height: 48px;
  width: 100%
}

.logo {
  position: relative;
  left: 25%;
  padding-top: 10px
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <script>  </script>          JS code can be here

  <title>NetBurner JS Example</title>
</head>
<body>

  <script>  </script>          JS code can be here

</body>

</html>
```

# Web-Scraping

Автоматизация обработки и парсинга веб-страниц с помощью скрипта-бота.

Этичность: практически у каждого сайта есть **robots.txt**

Например, у [kodomo.fbb.msu.ru](http://kodomo.fbb.msu.ru):

```
User-agent: *
Disallow: /wiki/*?*
Disallow: /wiki/HelpOn
Disallow: /wiki/WikiCourse
Disallow: /wiki/SystemPages
Disallow: /wiki/OtherUser
Disallow: /mail

User-agent: SemrushBot
Disallow: /
```

Любому агенту не посещать части **/wiki** и **/mail** полностью

Агенту **SemrushBot** закрыть весь сайт

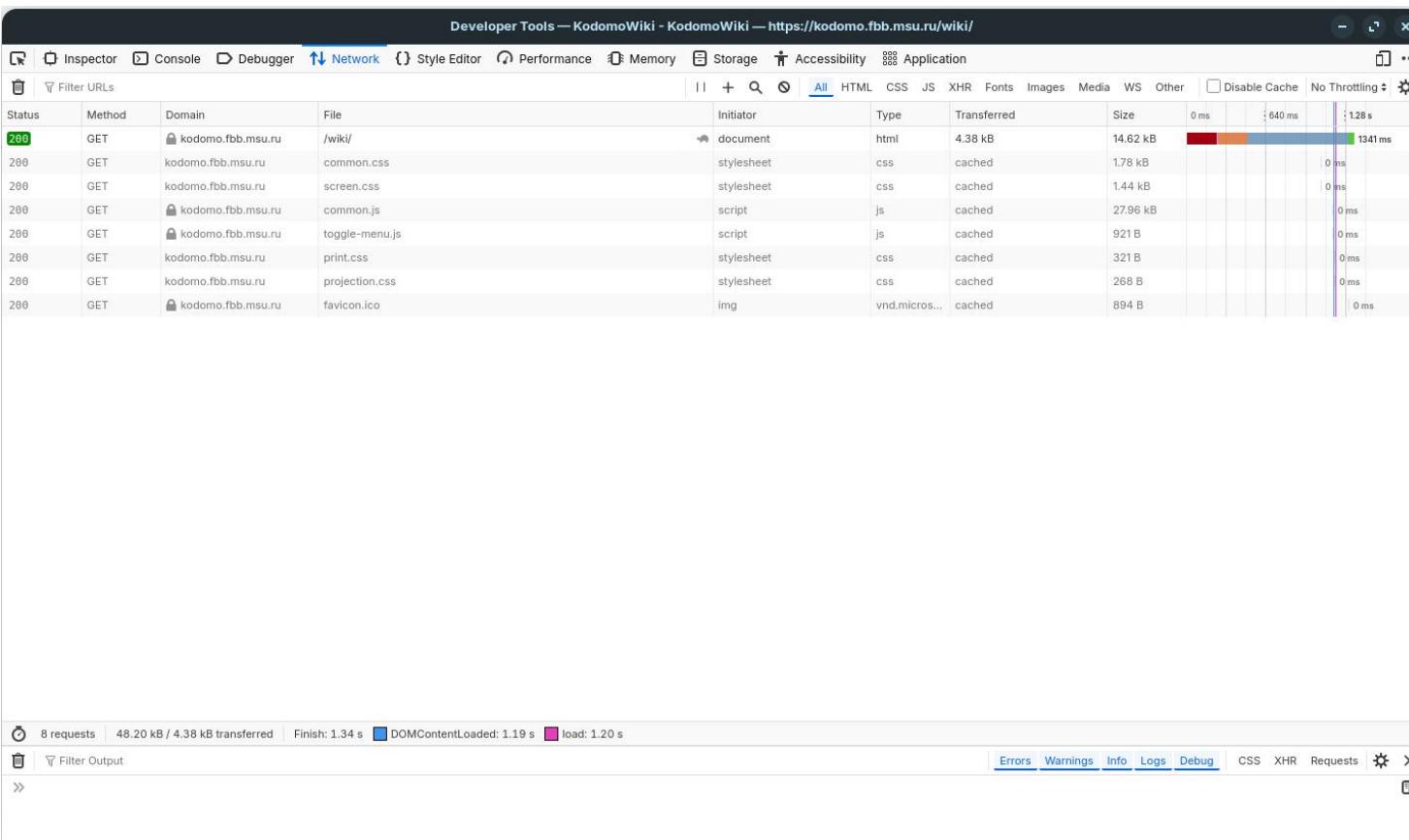
**Полностью запретить скрейпинг нельзя, но можно настоятельно просить.**



# Как написать свой скрейпер?

Вкладка “инструменты разработчика”

Пользуемся python: `requests + bs4`



Большая часть парсинга  
HTML-страницы -  
разглядывание кода  
элемента

Вкладка **Network**:  
куда идут запросы от  
веб-страницы для её  
корректной работы (в  
т.ч. для логирования  
вашей активности)

# requests

Поддержка HTTP-запросов к веб-серверам.

**Типы HTTP-запросов:**

## GET

Запрос на  
получение данных.  
**С payload**

## POST

Отправка данных,  
обычно для  
сохранения. **С payload**

## PUT

Отправка данных  
для обновления  
или создания  
новых. **С payload**

## DELETE

Удаление данных с  
сервера. **С payload**

## HEAD

Отладка запросов  
GET. **С payload**

# requests

## Работа с requests

```
import requests
from requests.auth import HTTPBasicAuth
r = requests.post("https://api.example.com/data",
                  data={"a":1})
r = requests.get("https://example.com",
                  params={"q": "test",
                           "page":2})
# response attributes:
# r.status_code      # Статус-код (200, 404, 500,...)
# r.ok               # True если < 400
# r.headers          # dict
# r.text              # декодированный текст
# r.content           # байты
# r.url
# r.json() - перевод в словарь
```

Можно указывать таймауты, хедеры в запросе - так он будет выглядеть менее роботизированным.

Можно запускать поток, постить файлы, хранить куки-файлы в объекте-сессии. Опций очень много, хорошо описаны в документации.

```
requests.get(url,
             auth=HTTPBasicAuth("user", "pass"))
```

# Пример

Многие сайты бросают автоматические перенаправления с главной страницы. Нужно позволять redirect-ы

```
1 import requests
2
3 url = "https://kodomo.fbb.msu.ru"
4
5 try:
6     with requests.Session() as s:
7         r = s.get(url, timeout=5, allow_redirects=True)
8         r.raise_for_status()
9         data = r.text
10        print(r.url, '\n-----')
11        print(data)
12 except requests.RequestException as e:
13     print("Request failed:", e)
```

<https://kodomo.fbb.msu.ru/wiki/>

```
-----
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta name="robots" content="index, follow">

<title>KodomoWiki - KodomoWiki</title>
<script type="text/javascript" src="/wiki-data/common/js/common.js"></script>

<script type="text/javascript">
<!--
var search_hint = "Искать";
//-->
</script>
```

# Что делать с содержимым?

HTML - иерархический язык разметки. Велосипед уже изобретён: **beautifulSoup**.

Можно использовать разные парсеры.  
Например, `lxml` - более прощающий, но работает дольше

html → <html><head>...</head><body>...</body></html>  
— head → <head><title>A Useful Page<title></head>  
— title → <title>A Useful Page</title>  
— body → <body><h1>An Int...</h1><div>Lorem ip...</div></body>  
— h1 → <h1>An Interesting Title</h1>  
— div → <div>Lorem Ipsum dolor...</div>

```
bs = BeautifulSoup(html.read(), 'html.parser')
title = bs.body.h1
```

Обращение к элементам иерархии через точку и атрибут-тег

# bs4: поиск тегов

**.find** для первого вхождения

**.find\_all** - список всех вхождений (позволяет поиск по атрибутам тегов)

**.select** - CSS-селектор, подбор тегов внутри тегов

```
1 from bs4 import BeautifulSoup
2 html = requests.get(url='https://kodomo.fbb.msu.ru/wiki')
3 bs = BeautifulSoup(html.text, 'html.parser')
4 bs.find('h1')
```

```
<h1 class="sidepanel">Kodomo</h1>
```

```
1 from bs4 import BeautifulSoup
2 html = requests.get(url='https://kodomo.fbb.msu.ru/wiki')
3 bs = BeautifulSoup(html.text, 'html.parser')
4 bs.find_all('h1')
```

```
[<h1 class="sidepanel">Kodomo</h1>,
 <h1 class="sidepanel">Пользователь</h1>,
 <h1
 id="KodomoWikiWelcome.A.2BBBQEPgQxBEAPg_.2BBD8EPgQ2BDAE0wQ.2B
 пожаловать на kodomo, сервер компьютерного класса ФББ</h1>]
```

```
1 from bs4 import BeautifulSoup
2 html = requests.get(url='https://kodomo.fbb.msu.ru/wiki')
3 bs = BeautifulSoup(html.text, 'html.parser')
4 bs.select('div > a')
```

```
[<a href="#" onclick="toggle_menu(1, 1)">Показать меню</a>,
 <a href="#" onclick="toggle_menu(0, 1)">Спрятать меню</a>,
 <a href="/wiki/KodomoWiki"></a>]
```

# bs4: иерархия тегов

- HTML

- body

- div.wrapper

- h1

- div.content

- table#giftList

- tr

- th

- th

- th

- th

- tr.gift#gift1

- td

- td

- span.excitingNote

- td

- td

- img

- ...table rows continue...

- div.footer



## Totally Normal Gifts

Here is a collection of totally normal, totally reasonable gifts that your friends are sure to love! Our collection is hand-curated by well-paid,

We haven't figured out how to make online shopping carts yet, but you can send us a check to:

123 Main St.

Abuja, Nigeria

We will then send your totally amazing gift, pronto! Please include an extra \$5.00 for gift wrapping.

Item Title	Description	Cost	Image
Vegetable Basket	This vegetable basket is the perfect gift for your health conscious (or overweight) friends! <b>Now with super-colorful bell peppers!</b>	\$15.00	
Russian Nesting Dolls	Hand-painted by trained monkeys, these exquisite dolls are priceless! And by "priceless," we mean "extremely expensive"! <b>8 entire dolls per set! Octuple the presents!</b>	\$10,000.52	
Fish Painting	If something seems fishy about this painting, it's because it's a fish! <b>Also hand-painted by trained monkeys!</b>	\$10,005.00	
Dead Parrot	This is an ex-parrot! <b>Or maybe he's only resting?</b>	\$0.50	
Mystery Box	If you love surprises, this mystery box is for you! Do not place on light-colored surfaces. May cause oil staining. <b>Keep your friends guessing!</b>	\$1.50	

© Totally Normal Gifts, Inc.

+234 (617) 863-0736

# bs4: иерархия тегов

- HTML
  - body

— div.wrapper

— h1

— div.content

— table#giftList

— tr

— th

— th

— th

— th

— tr.gift#gift1

— td

— td

— span.excitingNote

— td

— td

— img

— ...table rows continue...

— div.footer

```
for child in bs.find('table',{'id':'giftList'}).children:  
    print(child)
```

Here is a collection of totally normal, totally reasonable gifts that your friends are sure to love! Our collection is hand-curated by well-paid,

We haven't figured out how to make online shopping carts yet, but you can send us a check to:

123 Main St.

Abuja, Nigeria

We will then send your totally amazing gift, pronto! Please include an extra \$5.00 for gift wrapping.

Item Title	Description	Cost	Image
Vegetable Basket	This vegetable basket is the perfect gift for your health conscious (or overweight) friends! <b>Now with super-colorful bell peppers!</b>	\$15.00	
Russian Nesting Dolls	Hand-painted by trained monkeys, these exquisite dolls are priceless! And by "priceless," we mean "extremely expensive"! <b>8 entire dolls per set! Octuple the presents!</b>	\$10,000.52	
Fish Painting	If something seems fishy about this painting, it's because it's a fish! <b>Also hand-painted by trained monkeys!</b>	\$10,005.00	
Dead Parrot	This is an ex-parrot! <b>Or maybe he's only resting?</b>	\$0.50	
Mystery Box	If you love surprises, this mystery box is for you! Do not place on light-colored surfaces. May cause oil staining. <b>Keep your friends guessing!</b>	\$1.50	

© Totally Normal Gifts, Inc.  
+234 (617) 863-0736

# bs4: иерархия тегов

- HTML
  - body

```
— div.wrapper

- h1
- div.content
- table#giftList

```

```
— tr

- th
- th
- th
- th

  
— tr.gift#gift1

- td
- td
  - span.excitingNote
- td
- td
  - img

  
— ...table rows continue...
```

```
for sibling in bs.find('table', {'id':'giftList'}).tr.next_siblings:  
    print(sibling)
```

Here is a collection of totally normal, totally reasonable gifts that your friends are sure to love! Our collection is hand-curated by well-paid,

We haven't figured out how to make online shopping carts yet, but you can send us a check to:

123 Main St.

Abuja, Nigeria

We will then send your totally amazing gift, pronto! Please include an extra \$5.00 for gift wrapping.

Item Title	Description	Cost	Image
Vegetable Basket	This vegetable basket is the perfect gift for your health conscious (or overweight) friends! <b>Now with super-colorful bell peppers!</b>	\$15.00	
Russian Nesting Dolls	Hand-painted by trained monkeys, these exquisite dolls are priceless! And by "priceless," we mean "extremely expensive"! <b>8 entire dolls per set! Octuple the presents!</b>	\$10,000.52	
Fish Painting	If something seems fishy about this painting, it's because it's a fish! <b>Also hand-painted by trained monkeys!</b>	\$10,005.00	
Dead Parrot	This is an ex-parrot! <b>Or maybe he's only resting?</b>	\$0.50	
Mystery Box	If you love surprises, this mystery box is for you! Do not place on light-colored surfaces. May cause oil staining. <b>Keep your friends guessing!</b>	\$1.50	

© Totally Normal Gifts, Inc.  
+234 (617) 863-0736

# bs4: иерархия тегов

- HTML
  - body
    - div.wrapper
      - h1
      - div.content
      - table#giftList
        - tr
          - th
          - th
          - th
          - th
        - tr.gift#gift1
          - td
          - td
            - span.excitingNote
          - td
          - td
            - img
      - ...table rows continue...
    - div.footer

```
print(bs.find('img',  
             {'src': '../img/gifts/img1.jpg'}))  
.parent.previous_sibling.get_text())
```

Here is a collection of totally normal, totally reasonable gifts that your friends are sure to love! Our collection is hand-curated by well-paid, 123 Main St. Abuja, Nigeria	We haven't figured out how to make online shopping carts yet, but you can send us a check to: 123 Main St. Abuja, Nigeria	We will then send your totally amazing gift, pronto! Please include an extra \$5.00 for gift wrapping.
Item Title	Description	Cost
Vegetable Basket	This vegetable basket is the perfect gift for your health conscious (or overweight) friends! <b>Now with super-colorful bell peppers!</b>	\$15.00
Russian Nesting Dolls	Hand-painted by trained monkeys, these exquisite dolls are priceless! And by "priceless," we mean "extremely expensive"! <b>8 entire dolls per set! Octuple the presents!</b>	\$10,000.52
Fish Painting	If something seems fishy about this painting, it's because it's a fish! <b>Also hand-painted by trained monkeys!</b>	\$10,005.00
Dead Parrot	This is an ex-parrot! <b>Or maybe he's only resting?</b>	\$0.50
Mystery Box	If you love surprises, this mystery box is for you! Do not place on light-colored surfaces. May cause oil staining. <b>Keep your friends guessing!</b>	\$1.50

© Totally Normal Gifts, Inc.  
+234 (617) 863-0736

# Регулярные выражения

Стандартный инструмент, очень помогающий в поиске тегов с определёнными атрибутами или id.

```
bs = BeautifulSoup(html, 'html.parser')
images = bs.find_all('img',
    {'src':re.compile('.\\.\\./img/gifts/img.*\\.jpg')})
for image in images:
    print(image['src'])
```

./img/gifts/img1.jpg  
./img/gifts/img2.jpg  
./img/gifts/img3.jpg  
./img/gifts/img4.jpg  
./img/gifts/img6.jpg

Символ в реджексе	Значение
"A", "B", "c", "7"	любой не спецсимвол соответствует самому себе
"."	один любой символ (по умолчанию - кроме переноса строки)
любая цифра	
\s	любой whitespace character - пробел, знак табуляции (\t) и тд
\S	любой непробельный символ
"W"	Любая буква (то, что может быть частью слова), а также цифры и "_" (нижнее подчёркивание)
"W+"	Любая не-буква, не-цифра и не подчёркивание
[ABCDE]	Один из символов в скобках A, E, C
[0-9], [A-E]	также любой символ из диапазона
[^ABCDE], [^A-E]	любой символ, кроме перечисленных
A	B
A?	встреча символа возможна 1 раз или не происходит
A*	символ может встретиться любое количество раз (в т.ч. и не встретиться совсем)
A+	один или более раз
\b	граница слова
\B	Не граница слова: либо и слева, и справа буквы, либо и слева, и справа НЕ буквы
S{x}	(x - число) повторение символа S x раз
^, \$	символы начала и конца строки

# Веб-кроулер (crawlers)

# Пример: правило шести рукопожатий

```
1 headers = {"User-Agent": "EducationalScraper/1.0 (contact: Daniil Khlebnikov)"}
2 r = requests.get("https://en.wikipedia.org/wiki/Kevin_Bacon",
3 | | | | | | | | headers=headers)
4 bs = BeautifulSoup(r.text, "html.parser")
5 pattern = re.compile(r"^(?:(?://)|/wiki/)(?!:).)*$")
6 content = bs.find("div", id="bodyContent")
7 for link in content.find_all("a", href=pattern):
8 | | print(link["href"])
```

/wiki/David\_Boreanaz  
/wiki/David\_Boreanaz  
/wiki/Ben\_Browder  
/wiki/Matthew\_Fox  
/wiki/Michael\_C.\_Hall  
/wiki/Matthew\_Fox  
/wiki/Edward\_James\_Olmos  
/wiki/Josh\_Holloway  
/wiki/Stephen\_Moyer  
/wiki/Bryan\_Cranston  
/wiki/Bryan\_Cranston  
/wiki/Mads\_Mikkelsen  
/wiki/Hugh\_Dancy  
/wiki/Andrew\_Lincoln

Википедия очень строга к ботам:  
нужно устанавливать  
User-Agent'a

Отбор ссылок только на статьи: кроме них на странице есть ещё ссылки на технические, главные страницы и контакты.

# Простейший кроулер

```
3 random.seed(42)
4 BASE_URL = "https://en.wikipedia.org"
5 HEADERS = {"User-Agent": "EducationalScraper/1.0 (contact: Daniil Khlebnikov)"}
6
7 def get_links(article_path):
8     url = f"{BASE_URL}{article_path}"
9     r = requests.get(url, headers=HEADERS, timeout=5)
10    r.raise_for_status()
11
12    bs = BeautifulSoup(r.text, "html.parser")
13    content = bs.find("div", id="bodyContent")
14
15    if content is None:
16        return []
17    pattern = re.compile(r"^(?:(?!:).)*$")
18    return content.find_all("a", href=pattern)
19
20 links = get_links("/wiki/Nightcrawler_(film)")
21 print(f"\t0. Nightcrawler (film)")
22 for i in range(1, 11):
23     new_article = random.choice(links)["href"]
24     print(f"\t{i}. {new_article.split('/')[-1].replace('_', ' ')}")
25     links = get_links(new_article)
```

---

```
0. Nightcrawler (film)
1. Freejack
2. Morgan Creek Entertainment Group
3. Major League (film)
4. 1956 Cleveland Indians season
5. Jim Wilson (pitcher)
6. Henry Ford Hospital
7. Fisher Building
8. Crain Communications
9. Plastics News Global Group
10. Emap
```

В тегах `<div id='bodyContent'>` находим ссылки на статьи Википедии, поочерёдно ходим по ним и печатаем названия статей, куда попали.

Что получится, если сохранять пути между переходами и ссылками? Какой объект и что он представляет собой по смыслу?

# Получение внутренних и внешних ссылок

```
1 def getInternalLinks(bs, base_url):
2     internalLinks = []
3     for link in bs.find_all("a", href=True):
4         href = link["href"]
5         if re.match(r"^(/|#)", href):
6             full_url = requests.compat.urljoin(base_url, href)
7         elif href.startswith(base_url):
8             full_url = href
9         else:
10             continue
11         if full_url not in internalLinks:
12             internalLinks.append(full_url)
13     return internalLinks
14
15
16 def getExternalLinks(bs, base_url):
17     externalLinks = []
18     for link in bs.find_all("a", href=True):
19         href = link["href"]
20         full_url = requests.compat.urljoin(base_url, href)
21         if full_url.startswith("http") and not full_url.startswith(base_url):
22             if full_url not in externalLinks:
23                 externalLinks.append(full_url)
24
25     return externalLinks
```

Внутренние ссылки  
внутри сайта  
необязательно имеют  
полный URL в себе

В **requests** есть полезные  
инструменты для работы с  
гиперссылками,  
например **.compat.urljoin** как  
аналог **os.path.join**

напри

# scrapy

Питоновский пакет-утилита для веб-краулинга и скрейпинга.

Писать свой каждый раз неудобно,  
scrapy автоматически умеет многое из коробки.

Тут лежат шаблоны собираемых данных

Тут лежат различные прослойки между движком сайта, пауком и т.д.

Тут определения этапов пост-обработки данных после парсинга



# Scrapy

```
scrapy startproject wiki_spider
```

```
wiki_spider
└── scrapy.cfg
    └── wiki_spider
        ├── __init__.py
        ├── items.py
        ├── middlewares.py
        ├── pipelines.py
        ├── settings.py
        └── spiders
            └── __init__.py
```

Тут прописывать классы объектов-пауков, обходящих сайт

# Скрейпер для статей

wiki\_spiders/spiders/article.py

```
import scrapy
class ArticleSpider(scrapy.Spider):
    name='article'

    def start_requests(self):
        urls = [
            'http://en.wikipedia.org/wiki/Python_%28programming_language%29',
            'https://en.wikipedia.org/wiki/Functional_programming',
            'https://en.wikipedia.org/wiki/Monty_Python'
        ]
        return [scrapy.Request(url=url, callback=self.parse) for url in urls]

    def parse(self, response):
        url = response.url
        title = response.xpath('string(//h1[@id="firstHeading"])').get().strip()
        print(f'URL is: {url}')
        print(f'Title is: {title}'')
```

# Скрейпер для статей - new

```
import scrapy
class ArticleCLISpider(scrapy.Spider):
    name = 'article_CLI'
    def __init__(self, urls=None, *args, **kwargs):
        super(ArticleCLISpider, self).__init__(*args, **kwargs)
        if urls:
            self.start_urls = [url.strip() for url in urls.split(',')]

    def start_requests(self):
        for url in self.start_urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        url = response.url
        title = response.xpath('string(//h1[@id="firstHeading"])').get().strip()
        print(f'URL is: {url}')
        print(f'Title is: {title}')
```

```
scrapy crawl article_CLI -a urls="https://en.wikipedia.org/wiki/API"
```

# scrappy rules

Можно задавать правила перехода по ссылкам через regex внутри объекта Rule - LinkExtractor

Здесь в примере написано  
сложно, с парсингом командной  
строки. В ДЗ будет проще.

```
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule

class WikiSpider(CrawlSpider):
    name = 'wiki'
    allowed_domains = ['en.wikipedia.org']
    def __init__(self, start_url=None, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.start_urls = [start_url] if start_url else [
            'https://en.wikipedia.org/wiki/Benevolent_dictator_for_life'
        ]
        self.rules = (
            Rule(
                LinkExtractor(
                    allow=r'^https?://en\.wikipedia\.org/wiki/[A-Za-z0-9\-\'\.\,\,]+$', 
                    deny=r'/.(?:jpg|jpeg|png|gif|pdf)$',
                ),
                callback='parse_article',
                follow=True,
            ),
        )
        self._compile_rules()

    def parse_article(self, response):
        # Verify we're on an actual article page
        if not response.css('h1#firstHeading'):
            self.logger.warning(f"Not an article page: {response.url}")
            return

        title = response.xpath('string(//h1[@id="firstHeading"])').get()
        last_updated = response.css('#footer-info-lastmod::text').re_first(
            r'edited on ([^<]+)'
        )
        yield {
            'url': response.url,
            'title': title,
            'last_updated': last_updated,
        }
```

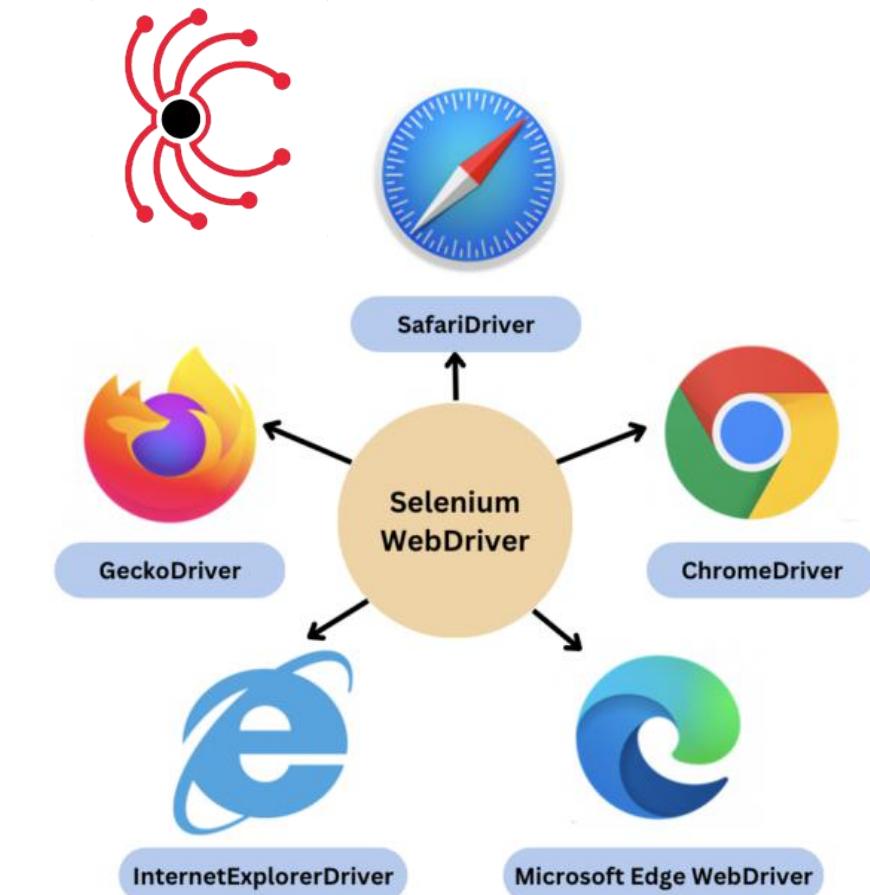
```
scrapy crawl wiki -a start_url="https://en.wikipedia.org/wiki/API" -o wiki.json
```

# Скрейпинг динамического сайта

Большинство сайтов не генерируют HTML целиком. Вместо этого, частично подгружают контент с помощью JS при определённых действиях пользователя.

Для скрейпинга такого рода придётся использовать веб-драйвер. В питоне: `splash, selenium`.

Обычно большинство всех интерактивных страниц генерируется JS :(



# Selenium

Пример: парсинг Amazon

```
driver = webdriver.Firefox(service=Service(), options=options) нужно создать объект-вебдрайвер

def scroll_to_bottom(driver, pause=1):
    last_height = driver.execute_script("return document.body.scrollHeight")
    while True:
        driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
        time.sleep(pause)
        new_height = driver.execute_script("return document.body.scrollHeight")
        if new_height == last_height:
            break
        last_height = new_height

    driver.get("https://www.amazon.com/s?k=book")
    WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.CSS_SELECTOR,
                                         "div[data-component-type='s-search-result']"))
    )
    scroll_to_bottom(driver, pause=1.5)
    product_cards = driver.find_elements(By.CSS_SELECTOR, "div[data-component-type='s-search-result']")
```

**к объекту можно применять методы JS**

**в объекте можно найти CSS элементов, фильтровать по ним**

# API

Строго говоря, в программировании, API - это вообще любой интерфейс программ, сервера, модуля, к которому есть доступ у внешнего пользователя. Например,

## API numpy:

### Python API

[NumPy's module structure](#)

[Array objects](#)

[The N-dimensional array \(`ndarray`\)](#)

[Scalars](#)

[Data type objects \(`dtype`\)](#)

[Data type promotion in NumPy](#)

[Iterating over arrays](#)

[Standard array subclasses](#)

[Masked arrays](#)

[The array interface protocol](#)

[Datetime and timedeltas](#)

[Universal functions \(`ufunc`\)](#)

**Закрытая часть кода numpy**  
(пользование ей не предполагается,  
но прочитать её можно)

```
def _align(self, axis):
    """A convenience function for operations that need to preserve axis
    orientation.
    """
    if axis is None:
        return self[0, 0]
    elif axis == 0:
        return self
    elif axis == 1:
        return self.transpose()
    else:
        raise ValueError("unsupported axis")
```

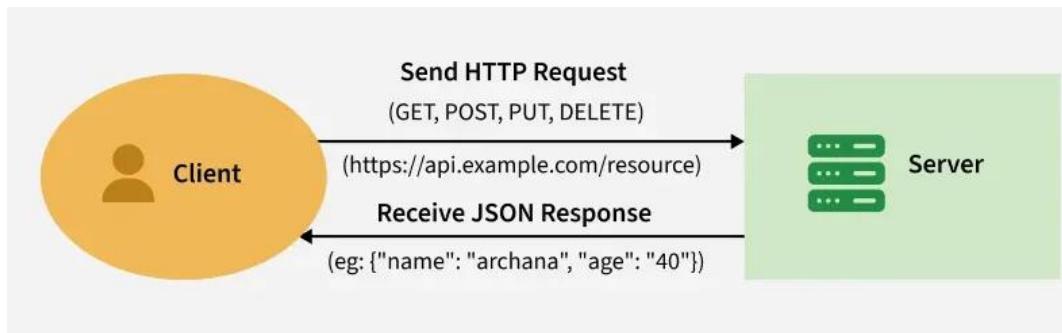
# API на сайтах

Как мы увидели, парсить HTML-страницы неудобно самому. Более того, значительная часть из них генерируется с помощью JS - selenium мы вообще не разбираем. Благо, некоторые сайты завели API для обращения и получения данных от них.

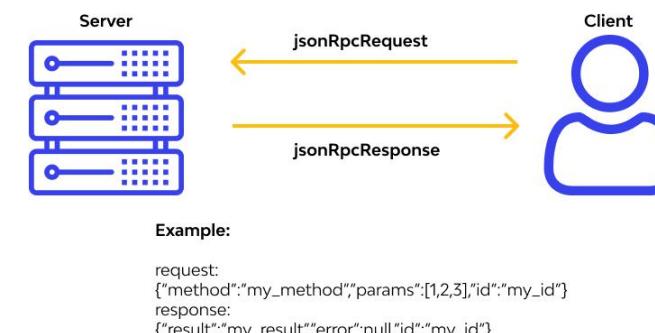
API - способ задать payload сайту.

## Виды стандартизованного API:

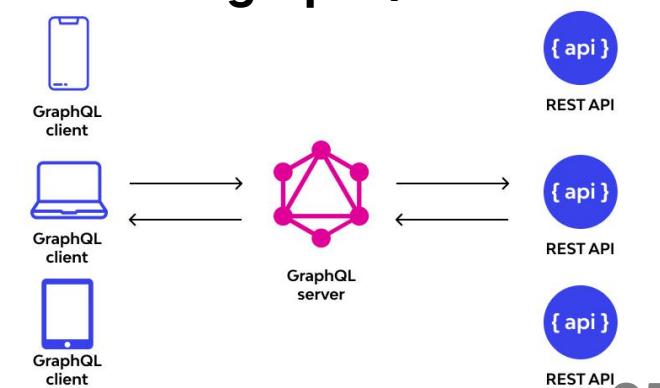
### RESTful API, SOAP API



### (g)RPC API



### \*GraphQL

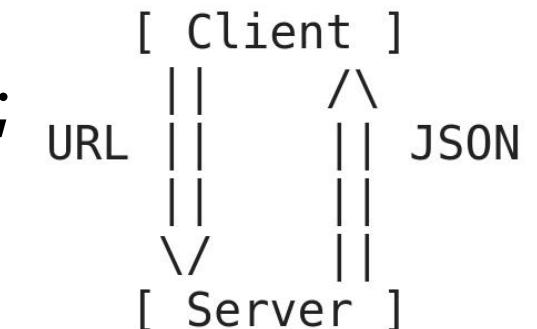


# REST API

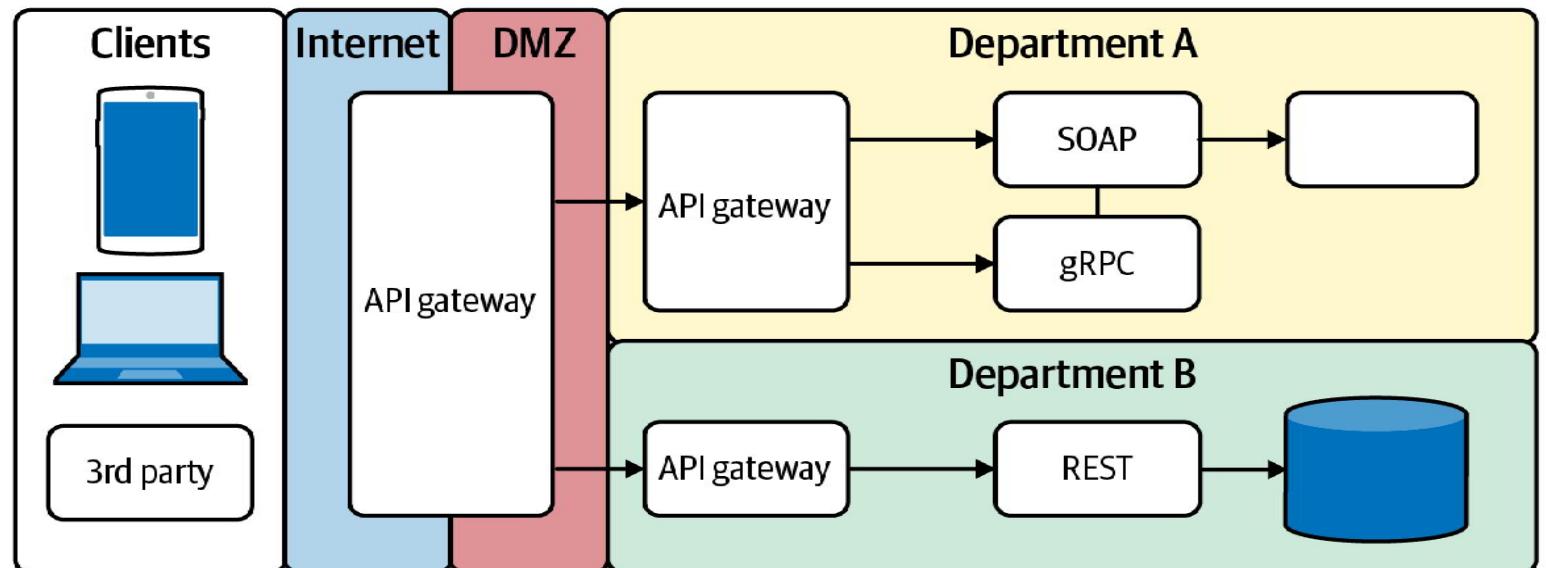
Ответ API - json.

## REpresentational State Transfer

- Клиент отделён от сервера, который хранит данные;
- **Stateless** - сервер не хранит состояние между запросами;
- Реализуется поверх HTTP; можно кэшировать;
- Используется в коммуникации сервисов при RPC



```
GET http://mastering-api.com/attendees
---
200 OK
{
  "value": [
    {
      "displayName": "Jim",
      "givenName": "James",
      "surname": "Gough",
      "email": "jim@mastering-api.com",
      "id": 1,
    }
  ],
  "@nextLink": "{opaqueUrl}"
}
```



# REST API Endpoint

Эндпоинт - ресурс, к которому отправляется запрос.

Пример InterPro: GET <https://www.ebi.ac.uk/interpro/api/>

```
HTTP 200 OK
Allow: GET, HEAD
Content-Type: application/json
InterPro-Version: 107.0
InterPro-Version-Minor: 0
Vary: Accept

{
  "endpoints": [
    "entry",
    "protein",
    "structure",
    "taxonomy",
    "proteome",
    "set",
    "utils"
  ],
  "databases": {
    "uniprot": {...},
    ...
  }
}
```

Эндпоинт доступа к разным БД и их записям  
(есть uniprot, cathgene3d, pfam, ...)

Эндпоинт доступа к записям белков (reviewed,  
unreviewed, uniprot)

GET <https://www.ebi.ac.uk/interpro/api/protein/uniprot/P123456>

Эндпоинт доступа к таксономии  
UniProt

Эндпоинт технического доступа:  
релизы, OpenAPI, тестирование

GET <https://www.ebi.ac.uk/interpro/api/utils/openapi>

# REST API через requests

```
import requests

BASE_URL = "https://www.ebi.ac.uk/interpro/api"
resp = requests.get(f"{BASE_URL}/entry/interpro", params={"page_size": 5})
entries = resp.json()["results"]

for entry in entries:
    accession = entry["metadata"]["accession"]
    details_resp = requests.get(f"{BASE_URL}/entry/interpro/{accession}")
    details = details_resp.json()
    name = details.get("metadata", {}).get("name", "N/A")
    entry_type = details.get("metadata", {}).get("type", "N/A")
    proteins_resp = requests.get(
        f"{BASE_URL}/protein/uniprot/entry/interpro/{accession}",
        params={"page_size": 1}
    )
    proteins_count = proteins_resp.json().get("count", 0)
    print(f"{accession}\t{name}\t{entry_type}\t{proteins_count}")
```

Берём первые 5 доменов InterPro и считаем, сколько белков с таким доменом

# Ещё пример: NCBI Genomes

По API референсу  
в БД Assembly получаем все  
метаданные сборок *E. coli*

```
import requests

taxon_name = "Escherichia coli"
params = {"db": "assembly", "term": f'"{taxon_name}"[Organism]',  
          "retmax": 20, "retmode": "json"}
url_search = "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi"
response = requests.get(url_search, params=params)
data = response.json()

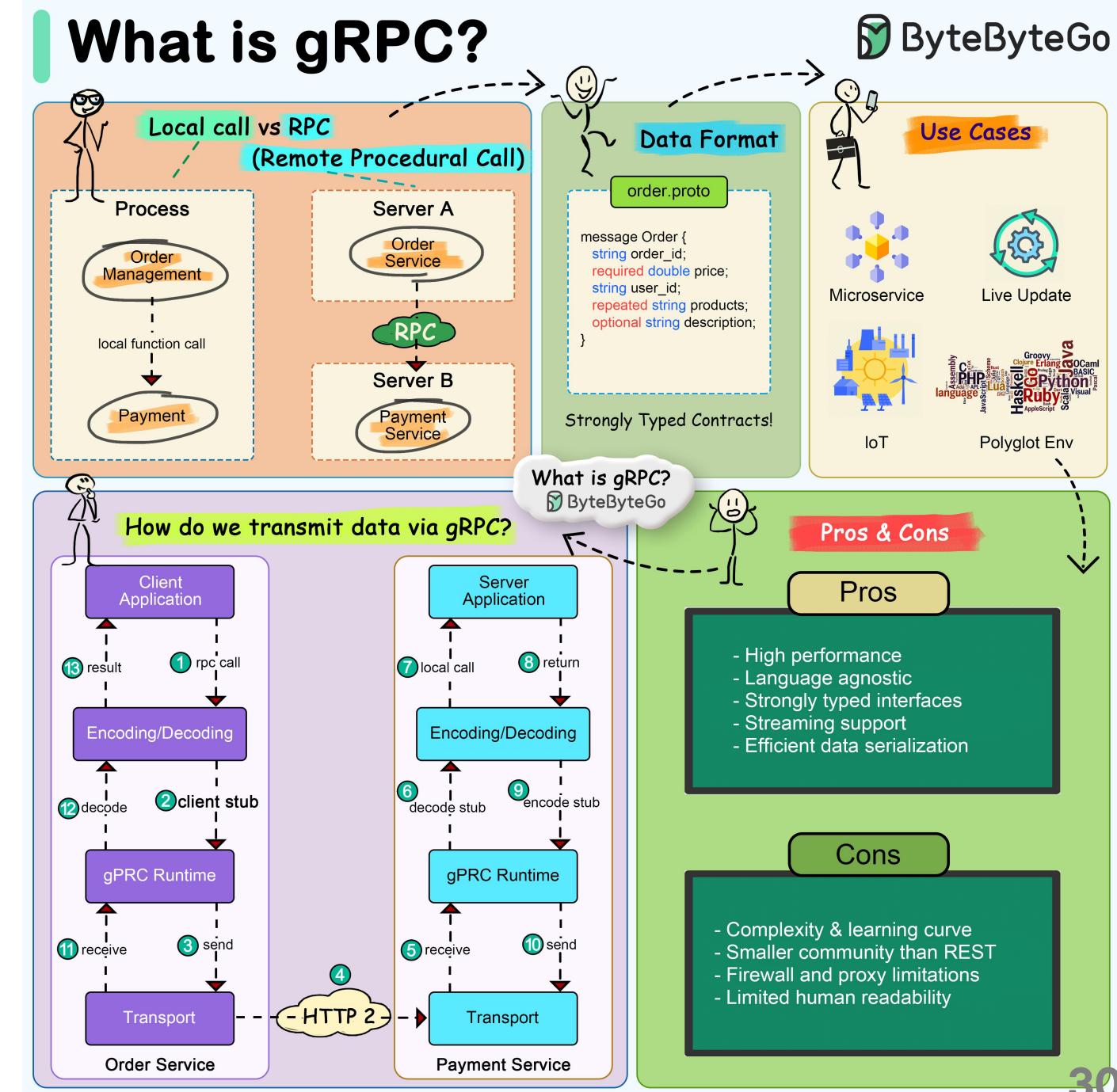
assembly_ids = data["esearchresult"].get("idlist", [])
print("Assembly IDs:", assembly_ids)
params_summary = {"db": "assembly",  
                  "id": ",".join(assembly_ids),  
                  "retmode": "json"}  
  
Внимание: батч-запрос!  
  
url_summary = "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi"
response_summary = requests.get(url_summary, params=params_summary)
summary = response_summary.json()["result"]
```

# gRPC

## (google) Remote Procedure Call.

Клиент вызывает экспонируемый API как функцию, а RPC внутри сам запускает методы и механизмы на сервере или микросервисе.  
Описывается на языке protobuf

Пример: биометрия Bioid



# gRPC protobuf

## Ha BiOID:

```
// Copyright 2025 BioID GmbH.  
// Definition of the BioID Web Service (BWS) 3 API.  
  
syntax = "proto3";  
  
package bioid.services.v1;  
  
option java_package = "com.bioid.services";  
option csharp_namespace = "BioID.Services";  
  
import "bwsmessages.proto";  
  
// BioID Web Service definition.  
service BioIDWebService {  
    // Liveness-detection API  
    // - 1 image: passive liveness detection only  
    // - 2 images: passive and active liveness detection  
    // - 2 images and tags: active liveness detection with challenge response  
    rpc LivenessDetection (LivenessDetectionRequest) returns (LivenessDetectionResponse);  
    // Video liveness-detection API  
    rpc VideoLivenessDetection (VideoLivenessDetectionRequest) returns (LivenessDetectionResponse);  
    // Photo-verification API  
    rpc PhotoVerify (PhotoVerifyRequest) returns (PhotoVerifyResponse);  
}  
  
// Liveness detection input images.  
message LivenessDetectionRequest {  
    // The input image samples.  
    repeated ImageData live_images = 1;  
}  
  
// Video liveness detection input video.  
message VideoLivenessDetectionRequest {  
    // The input video.  
    bytes video = 1;  
}
```

```
// Liveness detection output.  
message LivenessDetectionResponse {  
    // The return-status of the processing job.  
    JobStatus status = 1;  
    // Any error messages collected during processing.  
    repeated JobError errors = 2;  
    // Calculated image properties for each of the provided images in the given order.  
    repeated ImageProperties image_properties = 3;  
    // The liveness decision.  
    bool live = 4;  
    // The calculated liveness score that led to the live decision.  
    double liveness_score = 5;  
}  
  
// Photo-verification input images and flags.  
message PhotoVerifyRequest {  
    // One or more live images.  
    repeated ImageData live_images = 1;  
    // The ID-photo image.  
    bytes photo = 2;  
    // Flag to switch off LivenessDetection which is enabled by default.  
    bool disable_liveness_detection = 3;  
}  
  
// Photo-verification output.  
message PhotoVerifyResponse {  
    // The return-status of the processing job.  
    JobStatus status = 1;  
    // Any error messages collected during processing.  
    repeated JobError errors = 2;  
    // Calculated image properties for each of the provided live images if available.  
    repeated ImageProperties image_properties = 3;  
    // Calculated image properties for the provided photo.  
    ImageProperties photo_properties = 4;  
    // The determined verification level.  
    AccuracyLevel verification_level = 5;  
    // The calculated verification score that led to the decision for the verification level.  
    double verification_score = 6;  
    // In case a liveness detection was performed, here is the liveness decision.  
    bool live = 7;  
    // The calculated liveness score that led to the live decision.  
    double liveness_score = 8;  
  
    // Verification accuracy levels.  
    // We recommend not to accept verified persons with low verification levels.  
    enum AccuracyLevel {  
        // The person has not been recognized at all.  
        NOT_RECOGNIZED = 0;  
        // Worst accuracy level that correlates with a FAR of 0.5%  
        LEVEL_1 = 1;  
        // Bad accuracy level that correlates with a FAR of 0.25%  
        LEVEL_2 = 2;  
        // Not so good accuracy level that correlates with a FAR of 0.1%  
        LEVEL_3 = 3;  
        // Good accuracy level that correlates with a FAR of 0.01%  
        LEVEL_4 = 4;  
        // Best accuracy level that correlates with a FAR of 0.001%  
        LEVEL_5 = 5;  
    }  
}
```

# Игрушечный пример gRPC

protobuf:

```
syntax = "proto3";
package bio;

service SequenceAnalyzer {
    rpc AnalyzeSequence(SeqReq) returns (SeqResp);
}

message SeqReq {
    string seq_id = 1;
    string dna = 2;
}

message SeqResp {
    double gc_content = 1;
    repeated string motifs = 2;
}
```

```
python -m tools.protoc -I.
--python_out=.
--grpc_python_out=.
sequence.proto
```

С помощью протобуфера генерируется питоновский пакет для процедур. С ним можно взаимодействовать в коде программы.

```
import grpc
import sequence_pb2
import sequence_pb2_grpc

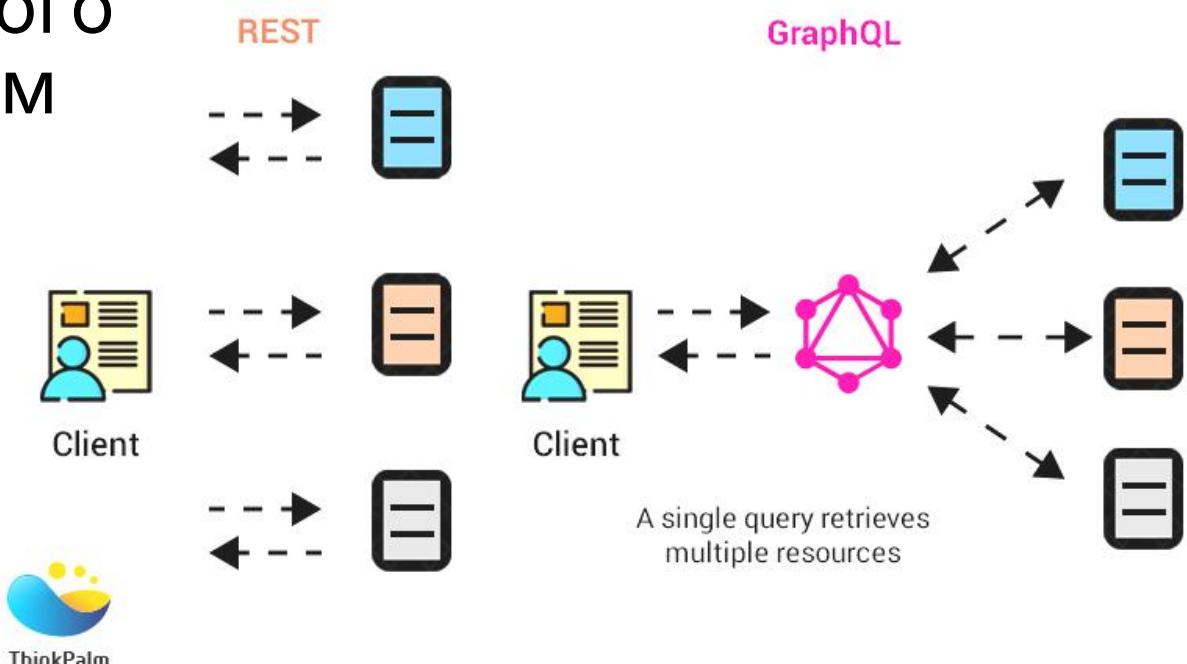
def run():
    chnl = grpc.insecure_channel('localhost:50051')
    stub = sequence_pb2_grpc.SequenceAnalyzerStub(chnl)
    request = sequence_pb2.SeqReq(
        seq_id="test1",
        dna="ATGCGCGATAACGATCG"
    )
    response = stub.AnalyzeSequence(request)
    print("GC content:", response.gc_content)
    print("Motifs found:", response.motifs)
```

# \*GraphQL

<https://beta.ensembl.org/data/graphql>

Язык запросов для API и среда выполнения, которая позволяет точно указывать, какие данные нужны.

В биоинформатике полезно: много связанных сущностей под одним GraphQL-эндпоинтом



# graphQL: пример с ensembl

```
import requests

url = "https://beta.ensembl.org/data/graphql"
response = requests.post(
    url,
    json={"query": query}
)

if response.status_code == 200:
    data = response.json()
    gene = data['data']['gene']
    print("Gene name:", gene['name'])
    print("Stable ID:", gene['stable_id'])
    print("SO term:", gene['so_term'])
    print("Alternative symbols:",
          gene['alternative_symbols'])
    print("Transcripts:")
    for t in gene['transcripts']:
        print(f"- {t['stable_id']} ({t['symbol']})")
```

```
query getGene {
  gene(
    by_id: {genome_id: ensembl-genomeID,
            stable_id: "ENSG00000NNNNN"}
  ) {
    alternative_symbols
    name
    so_term
    stable_id
    transcripts {
      stable_id
      symbol
    }
  }
}
```

Пример запроса к геному и гену.

Возвращаем несколько полей

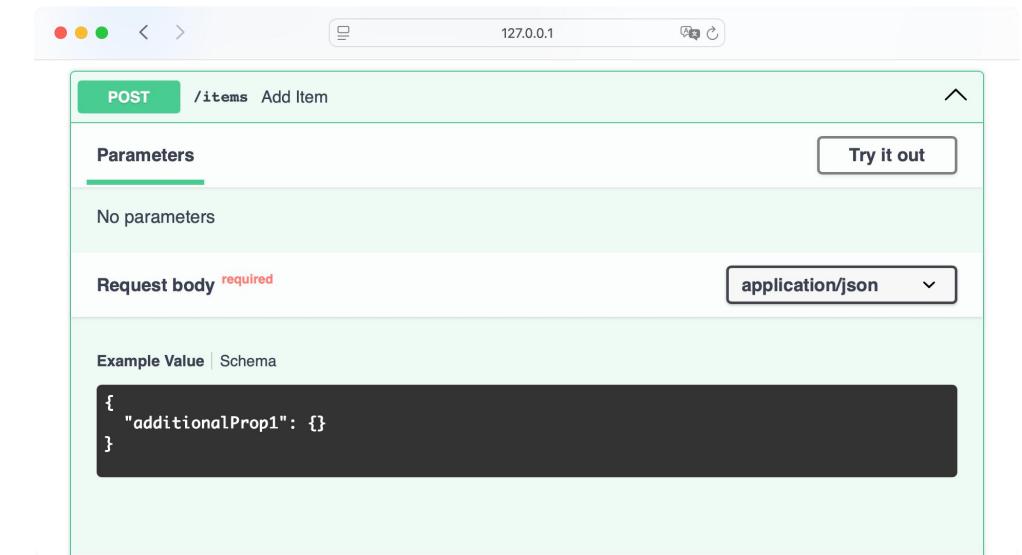
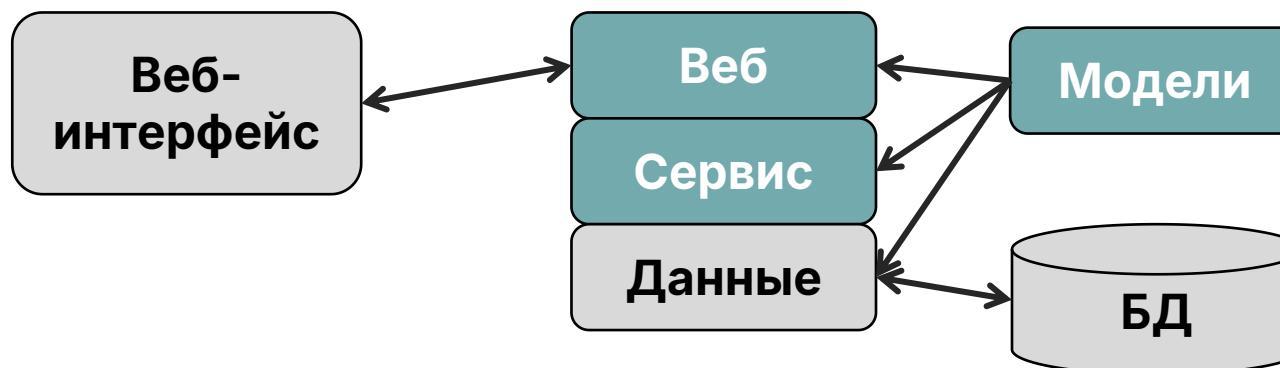
# Написание своего API. FastAPI

Иногда нужно, чтобы ваш сервис сам умел обрабатывать запросы без вмешательства кода. Для этого нужен фреймворк, который будет реализовывать API.

В случае с индустрией это Flask, Django.

Для нас подойдёт **FastAPI**.

Умеет работать с моделями ответов (pydantic), создаёт автоматическую документацию для API



# Модели данных

Обеспечиваются Pydantic. Например:

```
from pydantic import BaseModel, Field

class SequenceInput(BaseModel):
    sequence: str = Field(..., example="ATGGCCA")
    alphabet: str = Field("dna", example="dna")

class GCResponse(BaseModel):
    gc_content: float

class TranslationResponse(BaseModel):
    protein: str

class ORFResponse(BaseModel):
    start: int
    end: int
    protein: str
```

Pydantic позволяет контролировать, что вы отдаёте пользователю, а какие атрибуты и данные доступны сервисам внутри вашего сервера

# Приложение FastAPI

Приложение в отдельном питоновском скрипте:

```
from fastapi import FastAPI, HTTPException
from models import (SequenceInput, GCResponse,
                    TranslationResponse, ORFResponse)
import bio

app = FastAPI(    переменная, в которой хранится основной объект
    title="BioSeq API",
    description="REST API for training",
    version="1.0"
)

В декораторе указывается эндпоинт!

@app.post("/gc", response_model=GCResponse)
def compute_gc(data: SequenceInput):
    if data.alphabet.lower() != "dna":
        raise HTTPException(status_code=400)
    gc = bio.gc_content(data.sequence)
    return GCResponse(gc_content=gc)
```

# Запуск и тест

```
uvicorn main:app --reload
```

[127.0.0.1:8000/docs](http://127.0.0.1:8000/docs)

**BioSeq API** 1.0 OAS 3.1

[/openapi.json](#)

REST API for basic bioinformatics sequence analysis

**default**

**POST** /gc Compute Gc

**POST** /translate Translate

**POST** /orfs Find Orfs

Uvicorn - поддержка асинхронного веб-сервера.

```
import requests
```

```
url = "http://127.0.0.1:8000/gc"
payload = {
    "sequence": "ATGGCCATTGTAATGGGCCGC",
    "alphabet": "dna"
}
headers = {
    "Content-Type": "application/json"
}
response = requests.post(url, json=payload,
headers=headers)
print("Status code:", response.status_code)
print("Response JSON:", response.json())
```

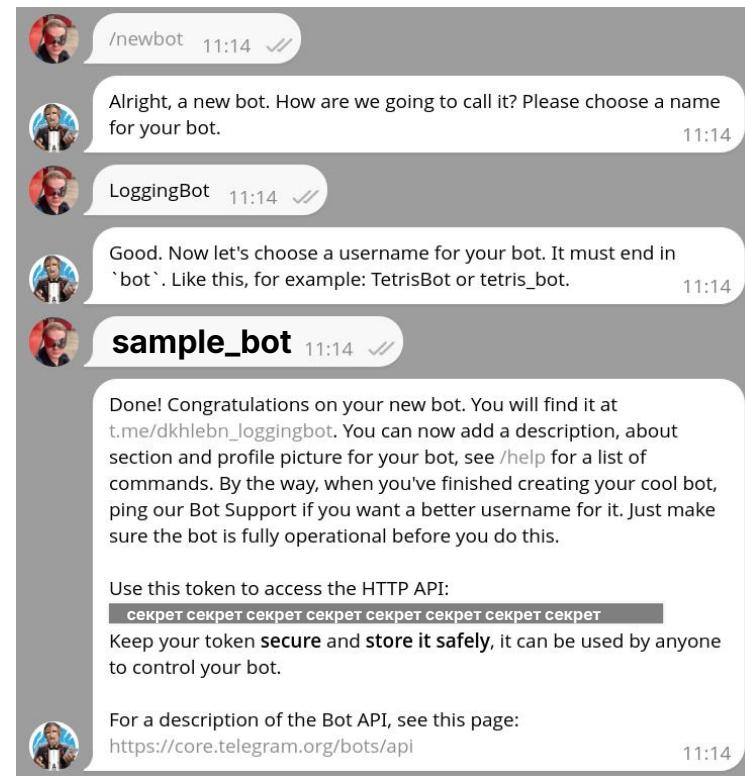
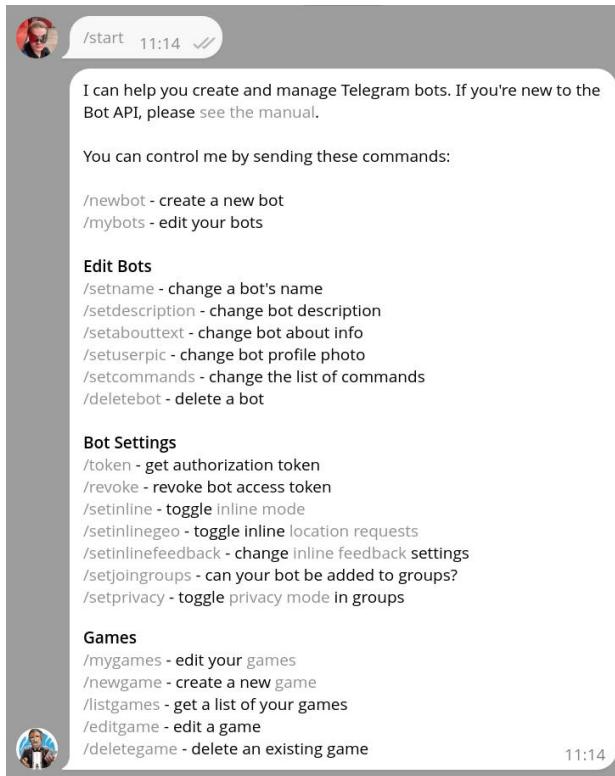
Status code: 200

Response JSON: {'gc\_content':57.14285714285714}

# Telegram API

Не связанный с сетью API. Пакет для настройки ботов.  
Простой пример: логирование информации вашего скрипта вам в  
личные сообщения ботом.

## [t.me/BotFather](https://t.me/BotFather)



Нужно получить API-токен для взаимодействия бота с телеграмом.

**Внимание:** этот токен лучше никому не передавать.  
Лучше всего в своей среде в **.bashrc** сделать:

```
export \
TGBOT_TOKEN="1234:b045ef1"
```

После этого на свой бот напишите **/start**

# Telegram API

```
pip install python-telegram-bot
```

Вся библиотека асинхронная: операции ввода-вывода не блокируют поток выполнения, а возвращают управление в event-loop. Для нас не так важно, всё изменение в определении функции:

```
async def get_chat_ids(username, token):
    bot = Bot(token=token)
    updates = await bot.get_updates()
    for update in updates:
        print(update)
        if update.message:
            chat = update.message.chat
            print(chat)
    return chat
```

Получаем, какие чаты есть у бота:

```
Update(message=Message(channel_chat_created=False,
chat=Chat(first_name='Daniil', id=421791509, last_name='Khlebnikov',
type=<ChatType.PRIVATE>, username='dkhlebn'),
date=datetime.datetime(2026, 1, 6, 8, 31, 57,
tzinfo=datetime.timezone.utc), delete_chat_photo=False,
entities=(MessageEntity(length=6, offset=0,
type=<MessageEntityType.BOT_COMMAND>)),
from_user=User(first_name='Daniil', id=421791509, is_bot=False,
is_premium=True, language_code='en', last_name='Khlebnikov',
username='dkhlebn'), group_chat_created=False, message_id=1,
supergroup_chat_created=False, text='/start'), update_id=274598183)
```

Важен атрибут `chat.id`. Получаем, сматчив `chat.username`

# Telegram API

```
pip install python-telegram-bot
```

Вся библиотека асинхронная: операции ввода-вывода не блокируют поток выполнения, а возвращают управление в event-loop. Для нас не так важно, всё изменение в определении функции:

```
async def get_chat_ids(username, token):
    bot = Bot(token=token)
    updates = await bot.get_updates()
    for update in updates:
        chat = update.effective_chat
        user = update.effective_user
        if chat is None or user is None:
            continue
        if user.username == username:
            return chat.id
```

Запуск асинхронной функции:

```
chat_id = asyncio.run(
    get_chat_ids(
        'dkhlebn',
        BOT_TOKEN
    )
)
```

# Отправка сообщения

```
import os
import traceback
import asyncio
from telegram import Bot

async def send_message(text, chat_id, token):
    bot = Bot(token=token)
    await bot.send_message(chat_id=chat_id,
                           text=text)

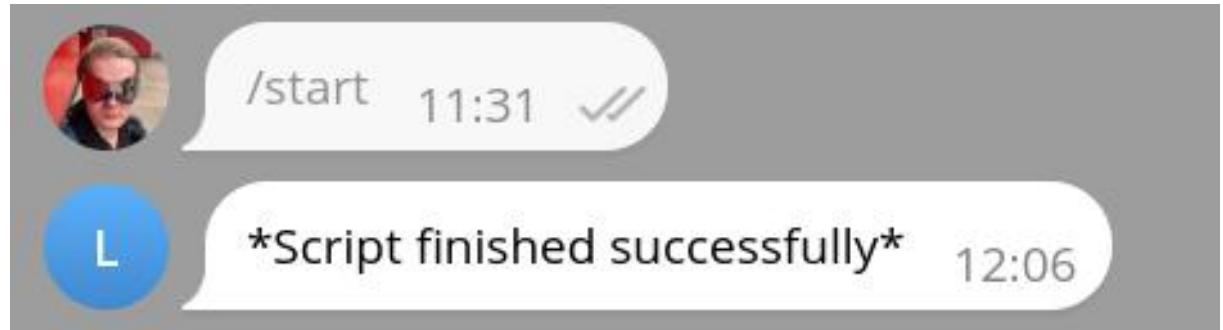
async def get_chat_ids(username, token):
    bot = Bot(token=token)
    updates = await bot.get_updates()
    for update in updates:
        chat = update.effective_chat
        user = update.effective_user
        if chat is None or user is None:
            continue
        if user.username == username:
            return chat.id

def notify(text, uname='dkhlebn'):
    BOT_TOKEN = os.environ["TGBOT_TOKEN"]
    chat_id = asyncio.run(get_chat_ids(uname, BOT_TOKEN))
    asyncio.run(send_message(text, chat_id, BOT_TOKEN))

def mainloop():
    print("Doing important work...")
    x = 1 / 2
```

```
if __name__ == "__main__":
    try:
        mainloop()
    except Exception:
        error = traceback.format_exc()
        msg = "*Script failed*\n\n" + error + "````"
        notify(msg)
    else:
        msg = notify("*Script finished successfully*")
```

## Без ошибок:



# Отправка сообщения

```
import os
import traceback
import asyncio
from telegram import Bot

async def send_message(text, chat_id, token):
    bot = Bot(token=token)
    await bot.send_message(chat_id=chat_id,
                           text=text)

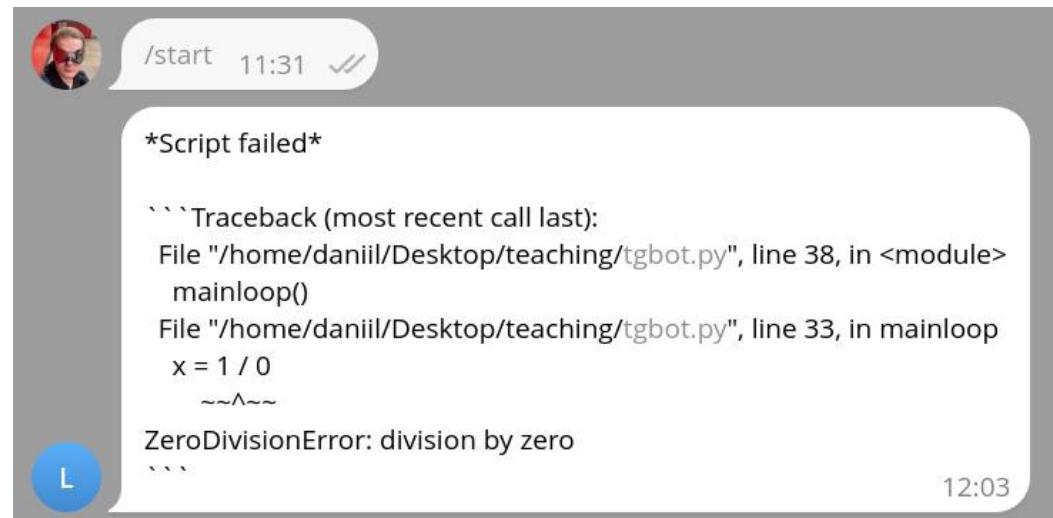
async def get_chat_ids(username, token):
    bot = Bot(token=token)
    updates = await bot.get_updates()
    for update in updates:
        chat = update.effective_chat
        user = update.effective_user
        if chat is None or user is None:
            continue
        if user.username == username:
            return chat.id

def notify(text, uname='dkhlebn'):
    BOT_TOKEN = os.environ["TGBOT_TOKEN"]
    chat_id = asyncio.run(get_chat_ids(uname, BOT_TOKEN))
    asyncio.run(send_message(text, chat_id, BOT_TOKEN))

def mainloop():
    print("Doing important work...")
    x = 1 / 0
```

```
if __name__ == "__main__":
    try:
        mainloop()
    except Exception:
        error = traceback.format_exc()
        msg = "*Script failed*\n\n```" + error + "```
        notify(msg)
    else:
        msg = notify("*Script finished successfully*")
```

## С ошибкой:



# Работа с большими файлами

Интернет предоставляет службу best-effort и может упасть.  
Большие файлы скачивать через rsync становится неудобно

Решение: aria2. Быстрое, параллелизуемое скачивание по любым протоколам (HTTP / FTP / BitTorrent). Кроме того, есть интерфейс RPC API.

```
aria2c -x 16 -s 8 -j 5 -k 100M <LINK>
  -d dwn_files -o bigfile.h5 -V true
  -c true --load-cookies cookies.txt
  --(http/ftp)-user=USER
  --(http/ftp)-passwd=passwd
  -log
```

**с** - флаг продолжения скачивания (если до этого скачивали не через aria)

**x** - количество соединений на сервер  
**s** - количество соединений на файл  
**j** - max количество паралельных скачиваний  
**k** - размер скачиваемого фрагмента

Проверка целостности файла через контрольные суммы и хеш-функции

# Пример: скачивание БД для BLAST

Для запуска BLAST на сервере / локальном ПК нужна БД.  
Скачаем, например, nt:

```
aria2c -x 8 -s 8 -k 1M -V \
https://ftp.ncbi.nlm.nih.gov/blast/db/nt.*.tar.gz;
```

```
aria2c https://ftp.ncbi.nlm.nih.gov/blast/db/nt.md5;
md5sum -c nt.md5
```

```
tar -xzf nt.*.tar.gz
export BLASTDB=/path/to/blast/db
```

Что такое  
checksum?

Здесь мы отдельно скачиваем MD5 checksum  
для проверки файлов из архива. Кроме того,  
с помощью aria мы ещё проверяем сам архив.  
Вероятность, что скачали что-то не то,  
снижена

# Checksum и хеширование

Идея та же: если хеши содержимого файлов не равны, значит скачанный файл скачан с ошибкой

Случай коллизии на больших файлах очень маловероятен.

**Checksum** (контрольная сумма) – собственно значение хеша. Пользователь проверяет равенство остатков от деления.

