ИНСТИТУТ
МАТЕМАТИКИ
МЕХАНИКИ
КОМПЬЮТЕРНЫХ
НАУК

# Основы работы с данными для ИИ

# Лекция 5
# Этические аспекты ИИ. Парсинг HTML и скрапинг

Доц
Екатерина Александровна
*Преподаватель кафедры информатики и вычислительного эксперимента*

# Зачем говорить об этике ИИ

ИИ — не просто инструмент, а система, влияющая на людей, бизнес и общество.

Ошибки ИИ масштабируются, а последствия часто «трудно объяснить» (black box).

Этика нужна, чтобы:

- минимизировать вред (дискриминация, утечки, небезопасный код);
- повысить доверие (прозрачность, проверяемость);
- соответствовать законам и лицензиям;
- сделать выгоды от ИИ устойчивыми и справедливыми.

# Ключевые принципы ответственного ИИ (RAI)

- **Прозрачность и объяснимость**: фиксировать источники, версию модели, ограничения; уметь объяснить вывод.
- **Справедливость и недискриминация**: оценивать и снижать смещения в данных и результатах.
- **Приватность и минимизация данных**: собирать ровно столько, сколько нужно; не хранить секреты в промптах.
- **Безопасность и надёжность**: защищаться от атак (prompt injection, supply chain), тестировать на отказоустойчивость.
- **Ответственность человека**: человек принимает финальные решения (human-in-the-loop), фиксирует авторство и рецензирует.
- **Устойчивость и экологичность**: учитывать вычислительные затраты и энергоэффективность.

# Что может пойти не так (общие риски ИИ)

- Смещения и дискриминация: модель повторяет перекосы исходных данных.
- Галлюцинации: уверенные, но неверные ответы; опасно в медицине, праве, безопасности.
- Приватность и секреты: случайная утечка PII/секретов в промптах или логах.
- Интеллектуальная собственность: нарушение авторских прав и лицензий, «заражение» кода не совместимыми лицензиями.
- Безопасность: атакующие подсказки, уязвимые шаблоны кода, вредоносные зависимости.
- Зависимость и «разучивание»: снижение квалификации, если слепо полагаться на ИИ.
- Непрозрачность: трудно воспроизвести и объяснить результат.

# Плюсы ИИ в программировании

- Скорость разработки: генерация чернового кода, шаблонов, документации, тестов.
- Качество: напоминание про edge-cases, рефакторинг, статический анализ, автогенерация тестов.
- Доступность: помогает новичкам быстрее входить в стек, снижает языковой барьер.
- Поддержка безопасности: подсказки по исправлению уязвимостей, чек-листы best practices.
- Рутины: миграции, конфиги CI/CD, boilerplate — экономия времени на низкоуровневых задачах.

# Минусы/риски ИИ в программировании

- Уязвимый/устаревший код: ассистент может сгенерировать небезопасные конструкции или старые API.
- Лицензии и авторские права: фрагменты могут походить на код с GPL/AGPL и др.; риск несоответствия корпоративной политике.
- Ложная уверенность: «код компилируется» ≠ «код корректен, безопасен и эффективен».
- Зависимости и supply chain: ассистент предложит библиотеку с низкой репутацией.
- Секреты: случайная вставка ключей/токенов, отправка их в промпт или логи.
- Потеря навыков: без рецензии и обучения разработчик перестаёт понимать дизайн-решения.
- Непредсказуемость: одинаковые запросы → разные ответы; сложно вести диффы и RCA без протоколов.

# Этические правила для разработчика

## Перед использованием ассистента:

- Не вставлять в промпт секреты/PII (ключи, пароли, приватный код клиента).
- Ставить чёткую цель: «что именно генерировать/исправлять», какие ограничения (лицензии, стиль, версия языка).

## После получения кода:

- Считать это черновиком: проверить логику, граничные случаи, производительность.
- Запустить тесты (если есть).
- Сверить код с официальной документацией API/фреймворка (мог быть устаревший синтаксис).
- Дополнить комментарии и документацию: зачем принято решение, какие альтернативы.

# Баланс: как извлечь максимум пользы

- Используйте ИИ там, где **много рутины** и есть чёткие спецификации.
- Всегда оставляйте **человеческое рецензирование** для критичных частей.
- Документируйте **ограничения** ассистента и **границы ответственности**.
- Включайте автоматические **сканеры лицензий, секретов, уязвимостей** в CI.
- Развивайте **навыки разработчика**: ИИ — усилитель, а не замена мышления.

# Советы по промптингу

- Сформулировать цель одной фразой. Что именно нужно на выходе? («Сгенерируй функцию валидации e-mail с тестами».)
- Дать контекст. Среда, версии, ограничения (Python 3.11, pandas 2.x, без внешних API).
- Определить формат вывода. «Верни только JSON со схемой …» / «Код в одном блоке, без комментариев».
- Задать критерии качества. Скорость O(n), совместимость лицензий.
- Пример(ы) ввода/вывода. 1–2 эталона сильно повышают точность
- Границы и запреты. «Не использовать глобальные переменные», «не ставить новые зависимости».
- Попросить самопроверку. «Проверь граничные случаи: пустой ввод, дубликаты, Unicode».
- Попросить краткое резюме решений. 2–3 пункта: что выбрано и почему (без лишней «воды»).
- Ограничить длину. «Не более 150 строк кода/500 слов пояснений».
- Итерировать. Дальше давать точечные правки: «Перепиши только часть X»

# Советы по промптингу

**Скелет «идеального» промпта (Prompt Canvas)**

**Роль:** «Ты —разработчик Python»

**Задача:** (1 фраза, глаголом)

**Контекст/среда:** версии, ОС, политика лицензий

**Входные данные:** как устроен input

**Выход/формат:** код/JSON/таблица, чёткие поля

**Критерии качества:** метрики, стиль, совместимость

**Примеры:** 1–2 пары input→output

**Ограничения/запреты:** что нельзя использовать

**Самопроверка:** чек-лист из 3–5 пунктов

**Длина/тон:** кратко/подробно; язык ответа

# Парсинг HTML страниц

**Парсинг** – это **автоматизированный процесс сбора и преобразования данных** из неструктурированных или полуструктурированных источников, таких как веб-страницы, текстовые файлы или таблицы, в удобный для анализа и обработки формат. Этот процесс выполняется с помощью специальных программ – **парсеров**, которые извлекают нужную информацию согласно заданным правилам и сохраняют ее в структурированном виде, например, в таблицах или базах данных.

Для чего используется парсинг:

- **Автоматизация** сбора больших объемов данных без ручного ввода
- **Анализ** информации с конкурирующих или тематических сайтов для принятия решений
- **Подготовка данных** для машинного обучения и бизнес-аналитики
- **Обновление и пополнение** баз данных с актуальными сведениями
- **Оптимизация** маркетинга, SEO и исследований рынка через быстрый сбор релевантной информации.

# Парсинг HTML страниц

Основные библиотеки и инструменты для парсинга на Python

**requests** – библиотека для отправки HTTP-запросов и получения HTML страниц.

**BeautifulSoup (bs4)** – популярный парсер HTML/XML, удобен для навигации по дереву документа и извлечения информации.

**lxml** – быстрая библиотека для парсинга HTML и XML, используется как альтернатива или дополнение к BeautifulSoup.

**Scrapy** – мощный и масштабируемый фреймворк для веб-скрапинга, подходящий для проектов с большим объёмом данных и сложной логикой.

**Selenium** – инструмент для автоматизации браузера, нужен, если сайт динамически генерирует контент с использованием JavaScript (например, SPA или загрузка данных через AJAX).

**pandas** – для обработки и сохранения собранных данных в удобном формате (DataFrame, CSV, Excel).

# Парсинг HTML страниц

## Сайт для скрапинга – Books to Scrape

Books to Scrape We love being scraped!

Home / All products

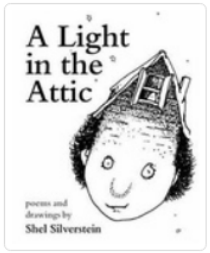
Books

- Travel
- Mystery
- Historical Fiction
- Sequential Art
- Classics
- Philosophy
- Romance
- Womens Fiction
- Fiction
- Childrens
- Religion
- Nonfiction
- Music
- Default
- Science Fiction
- Sports and Games
- Add a comment
- Fantasy
- New Adult

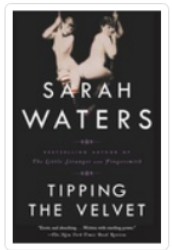
### All products

**1000** results - showing **1** to **20**.

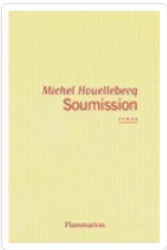
**Warning!** This is a demo website for web scraping purposes. Prices and ratings here were randomly assigned and have no real meaning.



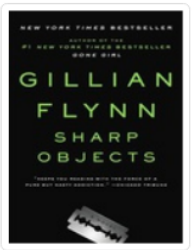

A Light in the ...




Tipping the Velvet




Soumission




Sharp Objects

# Парсинг HTML страниц

```python
import requests
from bs4 import BeautifulSoup
import pandas as pd

all_books = []
```

Импорт библиотек

Объявление пустого массива

Сохраняем URL страниц каждой книги

```python
for page_num in range(1, 5):
    url = f'https://books.toscrape.com/catalogue/page-{page_num}.html'
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')

    books = soup.find_all('li', class_='col-xs-6')
```

# Парсинг HTML страниц

Проходим по страницам каждой книги и сохраняем в массив all_books

```
    for book in books:
        book_url = 'https://books.toscrape.com/catalogue/' + book.find('a')['href']
        book_resp = requests.get(book_url)
        book_soup = BeautifulSoup(book_resp.content, 'html.parser')

        all_books.append({
            'Name': book_soup.find('h1').text,
            'Description': book_soup.select('p')[3].text,
            'Price': book_soup.find('p', class_='price_color').text,
            'Category': book_soup.find('ul', class_='breadcrumb').select('li')[2].text.strip(),
            'Availability': book_soup.select_one('p.availability').text.strip(),
            'UPC': book_soup.find('table').find_all('tr')[0].find('td').text
        })
```

```
df = pd.DataFrame(all_books)
```

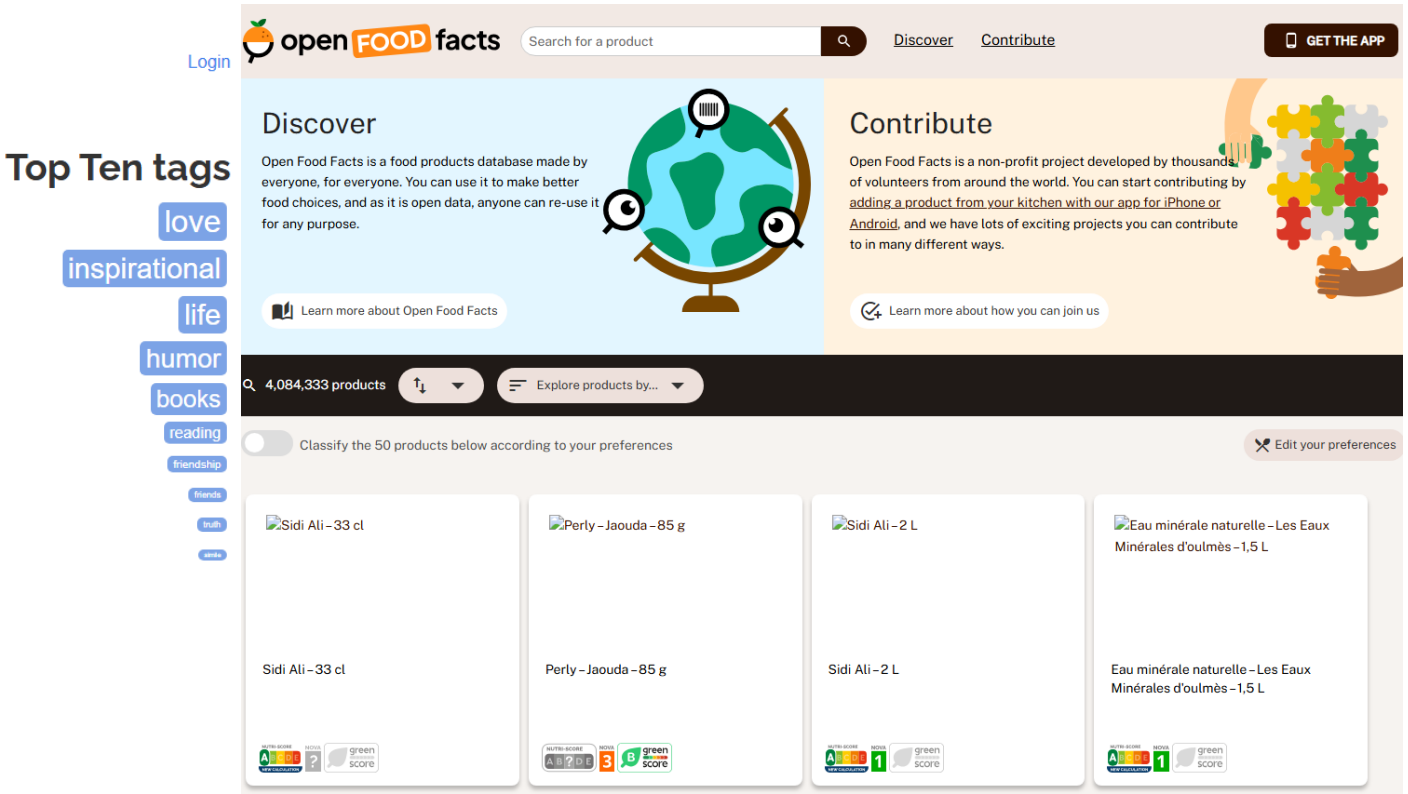
Создаем датафрейм на основе полученного массива

# Парсинг HTML страниц

## Больше сайт для скрапинга

- Quotes to Scrape


- Open Food Facts



# Парсинг HTML страниц

**urllib** – это стандартная библиотека Python **для работы с URL и веб-запросами.** Он состоит из нескольких подтем: **urllib.request** для отправки запросов и получения ответов, **urllib.parse** для разбора и построения URL, **urllib.error** для обработки ошибок и **urllib.robotparser** для анализа файла robots.txt.

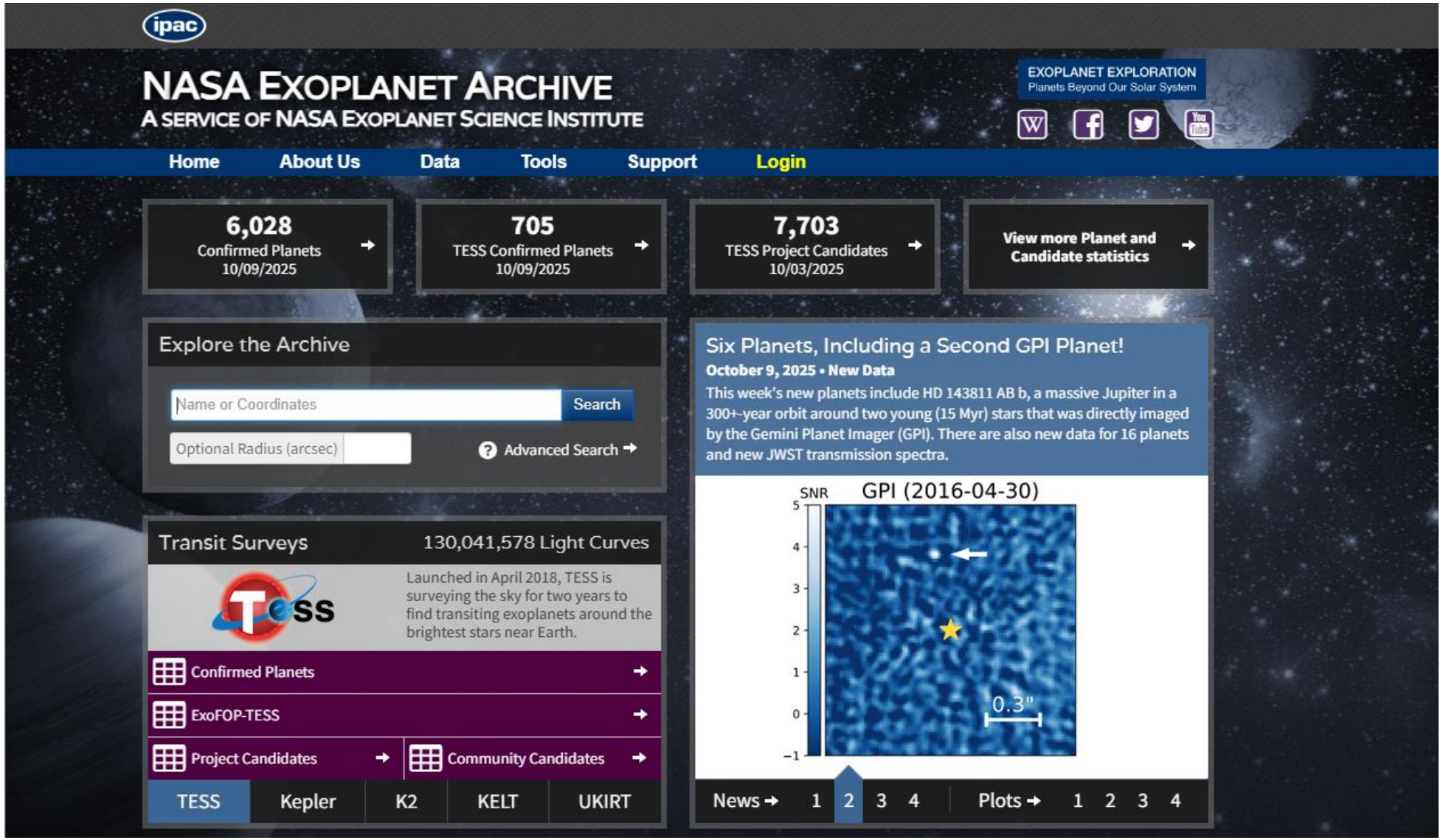
```python
from urllib.parse import urlparse, urlunparse, urlencode

url = 'https://edu.mmcs.sfedu.ru/course/view.php?id=806'
parsed = urlparse(url)
print(parsed.scheme)  # https
print(parsed.netloc)  # edu.mmcs.sfedu.ru
print(parsed.path)    # /course/view.php

# Изменение и сборка URL
new_url = urlunparse(parsed._replace(scheme='http'))
print(new_url)  # http://edu.mmcs.sfedu.ru/course/view.php?id=806

# Кодирование параметров для URL
params = {'q': 'python urllib', 'page': 2}
query_string = urlencode(params)
print(query_string)  # q=python+urllib&page=2
```

# Получение данных через API

**API (Application Programming Interface)** – это **программный интерфейс приложения**, набор правил и инструкций, по которым разные программы взаимодействуют между собой и обмениваются данными. Проще говоря, API – это «интерфейс» или «переводчик», благодаря которому одна программа может «общаться» с другой, независимо от их внутренней реализации.

API широко используются для получения данных из внешних источников, интеграции разных систем, автоматизации процессов и расширения возможностей приложений. Например, с помощью API можно подключить погодный сервис к вашему приложению, получить данные о курсах валют, работать с базами данных, управлять удалёнными устройствами и многое другое. API работают через **стандартизованные запросы** (например, **HTTP**), обычно возвращая данные в удобных форматах, таких как **JSON** или **XML**. Это упрощает разработку, делает программы более гибкими и масштабируемыми.

# Получение данных через API



# Получение данных через API

Отправка GET-запроса к NASA Exoplanet Archive через TAP API. Метод raise_for_status() проверяет, что запрос успешен (код 200), иначе вызовет исключение.

```
url = "https://exoplanetarchive.ipac.caltech.edu/TAP/sync?query=select+*+from+pscomppars&format=json"
response = requests.get(url)
response.raise_for_status()
data = response.json()
df = pd.DataFrame(data)
```