# М1.

## Урок №1: Введение в Python. Установка и среда разработки.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомить учащихся с языком программирования Python и его возможностями.
* Объяснить ключевые отличия текстового программирования от визуального (Scratch).
* Помочь учащимся установить Python и выбрать подходящую среду разработки (IDE).
* Написать первую программу "Hello, World!" и понять принципы её выполнения.

**Основные понятия:**

* Язык программирования
* Интерпретатор
* Интегрированная среда разработки (IDE)
* Консоль/терминал
* Синтаксис
* Отладка (простейшее понимание)
* Программа "Hello, World!"

**Необходимое программное обеспечение:**

* Python (скачанный с python.org)
* Thonny IDE (рекомендуется для первых уроков) или PyCharm Community Edition.

**План занятия:**

1. **Введение (10 минут)**
   * Приветствие, краткое напоминание о пройденных курсах ("Уверенный пользователь ПК", "Попробуй IT!").
   * "Поздравляю! Вы выбрали одно из самых востребованных и интересных направлений в IT – программирование на Python!"
   * Обсуждение: Что такое программирование? (вспомнить Scratch).
   * В чем отличие от Scratch? (Текстовые команды вместо блоков). "Представьте, что вы пишете книгу, а не собираете конструктор".
2. **Что такое Python? (15 минут)**
   * **Теоретический материал:**
     + **Язык программирования:** Специальный язык для общения с компьютером, чтобы давать ему инструкции. Python - один из таких языков.
     + **История:** Создан Гвидо ван Россумом в конце 80-х - начале 90-х. Назван в честь комедийного шоу "Летающий цирк Монти Пайтона".
     + **Популярность и особенности:**
       - **Простота синтаксиса:** Очень похож на английский язык, легко читается. Меньше специальных символов, чем в других языках.
       - **Универсальность:** Можно использовать почти для всего!
       - **Интерпретируемый:** Программы на Python выполняются интерпретатором построчно, без предварительной компиляции всего кода (как Scratch выполнял блоки по очереди).
     + **Где используется Python:**
       - **Веб-разработка:** Создание сайтов и веб-приложений (например, Instagram, Spotify).
       - **Наука о данных и машинное обучение (ИИ):** Анализ больших объемов информации, создание "умных" систем (распознавание лиц, рекомендательные системы).
       - **Автоматизация:** Написание скриптов для автоматизации рутинных задач (например, сортировка файлов, отправка отчетов).
       - **Игры:** Разработка логики для игр (например, Battlefield 2, EVE Online).
       - **Встроенные системы:** Управление устройствами (например, на Raspberry Pi).
     + **Аналогия:** "Представьте, что Python - это универсальный набор инструментов, которым можно построить что угодно, от маленького домика до небоскреба, а также чинить бытовую технику."
3. **Установка Python и IDE (25 минут) – Практическая часть под руководством преподавателя.**
   * **Теоретический материал:**
     + **Установка Python:** Чтобы компьютер "понимал" команды Python, нужно установить сам язык (интерпретатор). Это как установить новую программу на компьютер.
     + **Интегрированная среда разработки (IDE):** Специальная программа, которая помогает программировать. Она предоставляет удобный редактор для написания кода, кнопки для запуска, окно для вывода результата (консоль), и часто инструменты для поиска ошибок. Это как продвинутый текстовый редактор, специально для кода.
     + **Рекомендуемые IDE для начала:**
       - **Thonny:** Простой и интуитивно понятный, идеально подходит для начинающих. Имеет встроенный отладчик, который помогает пошагово следить за выполнением программы.
       - **PyCharm Community Edition:** Более мощный и профессиональный инструмент, используемый разработчиками по всему миру. Имеет множество функций, но для новичков может быть сложнее.
     + **Консоль/Терминал:** Это текстовое окно, где программа выводит результаты своей работы или где мы можем вводить команды.
   * **Практика:**
     + Пошаговая установка Python с официального сайта ([python.org/downloads](https://www.python.org/downloads/)). Обязательно отметить галочку "Add Python to PATH" во время установки (это позволит запускать Python из любой папки).
     + Установка выбранной IDE (предложить Thonny).
     + Запуск IDE, знакомство с интерфейсом:
       - **Окно редактора:** Где пишем код.
       - **Окно Shell/Run:** Где видим результат работы программы и сообщения об ошибках.
       - **Кнопка "Run" (Запустить):** Для выполнения написанного кода.
4. **Первая программа "Hello, World!" (25 минут) – Практическая часть.**
   * **Теоретический материал:**
     + **Функция print():** Это одна из базовых "команд" в Python. Она используется для вывода информации на экран (в консоль). Все, что находится внутри круглых скобок () и является текстом (строкой), будет показано.
     + **Синтаксис print():**
       - print("Текст, который нужно вывести")
       - Текст (строка) всегда должен быть заключен в кавычки (" или ').
       - Пример: print("Hello, World!") выведет Hello, World!
       - Пример: print('Я учу Python!') выведет Я учу Python!
     + **Запуск программы:** Нажимаем кнопку "Run" или используем соответствующую комбинацию клавиш (обычно F5 в Thonny).
     + **Ошибки:** Если программа не работает, скорее всего, есть ошибка в синтаксисе (например, пропущенная кавычка, опечатка в названии функции). Компьютер выведет сообщение об ошибке, которое поможет найти проблему. Покажем, как выглядит простая ошибка.
   * **Практика:**
     + Написание программы: print("Hello, World!") в редакторе кода.
     + Запуск программы.
     + **Легкий уровень:** Напишите программу, которая выводит на экран ваше имя.
       - Пример: print("Меня зовут [Ваше имя]")
     + **Средний уровень:** Напишите программу, которая выводит на экран 3-5 разных приветствий (например, "Привет!", "Добрый день!", "Как дела?"). Каждое приветствие на новой строке.
       - Пример:

Python

print("Привет!")

print("Добрый день!")

print("Как дела?")

* + - **Сложный уровень:** Напишите программу, которая:
      1. Создает "секретное" сообщение, в котором используется ваше имя.
      2. Выводит персонализированное приветствие, например: "Привет, [Имя пользователя]! Добро пожаловать в мир Python!". *(Пока не используем input(), просто попросить вставить имя в строку прямо в коде, например, print("Привет, Алиса! Добро пожаловать в мир Python!"))*.
    - **Игровой элемент: "Шпионская шифровка":**
      * Задание: "Представьте, что вы шпион и получили секретное сообщение. Ваша задача – написать программу, которая "произносит" это сообщение. Сообщение: 'Мир. Труд. Python.' Каждое слово на отдельной строке."
      * Решение: Использование нескольких print() или специального символа \n для новой строки: print("Мир.\nТруд.\nPython.")

1. **Обсуждение и вопросы (15 минут)**
   * Проверить, у всех ли получилось запустить программу.
   * Ответить на вопросы.
   * Краткий обзор пройденного: Что такое Python, зачем нужен print(), как запускать программу.
   * Превью следующего урока: "На следующем занятии мы узнаем, как компьютер запоминает информацию и как мы можем хранить данные в наших программах!"

**Материалы/Ресурсы:**

* Презентация с ключевыми понятиями.
* Примеры кода для демонстрации.
* Ссылка на официальный сайт Python для загрузки: [python.org/downloads](https://www.python.org/downloads/)
* Ссылка на сайт Thonny: [thonny.org](https://thonny.org/)

**Домашнее задание:**

* Повторить установку Python и IDE, если были сложности.
* Написать программу, которая выводит на экран небольшое стихотворение или цитату (4-5 строк).
* Попробовать изменить текст внутри скобок print() и посмотреть, что получится.

## Урок №2: Переменные и типы данных (числа и строки).

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять концепцию переменных для хранения данных в программе.
* Изучить основные типы данных в Python: целые числа (int), числа с плавающей точкой (float), строки (str).
* Научиться объявлять переменные, присваивать им значения и использовать их в программе.
* Понять правила именования переменных и их важность для читаемости кода.

**Основные понятия:**

* Переменная
* Присваивание (оператор =)
* Тип данных
* Целое число (int)
* Число с плавающей точкой (float)
* Строка (str)
* Функция type()

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое Python? Для чего нужна функция print()? Как написать "Hello, World!"?
   * Проверка домашнего задания (выборочно или по поднятым рукам), обсуждение возникших вопросов.
2. **Что такое переменная? (20 минут)**
   * **Теоретический материал:**
     + **Аналогия:** "Представьте, что компьютер – это очень большая библиотека с множеством ячеек хранения. Каждая ячейка может хранить какую-то информацию. Переменная – это как одна из таких ячеек, у которой есть свое уникальное имя (название), чтобы мы могли к ней обращаться и класть/доставать информацию."
     + **Зачем нужны переменные:**
       - Хранить изменяющиеся данные: счет в игре, возраст пользователя, температура.
       - Делать код более гибким: если нужно изменить значение, меняем его только в одном месте (в переменной), а не по всей программе.
       - Делать код более читаемым: price = 100 понятнее, чем просто число 100.
     + **Присваивание значений:** Для того чтобы положить данные в "коробку" (переменную), мы используем **оператор присваивания =**.
       - Синтаксис: имя\_переменной = значение
       - Пример: score = 0 (в переменную score положили число 0)
       - Пример: user\_name = "Алиса" (в переменную user\_name положили текст "Алиса")
     + **Правила именования переменных:**
       - Только латинские буквы (a-z, A-Z), цифры (0-9) и символ нижнего подчеркивания (\_).
       - Не может начинаться с цифры.
       - Нельзя использовать зарезервированные слова Python (например, print, if, else).
       - **Рекомендация:** Имена переменных должны быть понятными и отражать их содержимое (например, user\_age вместо a).
       - Пример хороших имен: total\_score, player\_name, level1.
       - Пример плохих имен: 1name, if, \_sum (если не знаете, что это).
3. **Типы данных (25 минут)**
   * **Теоретический материал:**
     + **Что такое тип данных:** Компьютеру нужно знать, с каким видом информации он работает, чтобы правильно её обрабатывать. Например, числа складываются, а текст – нет.
     + **Основные типы данных в Python:**
       - **int (integer - целое число):** Используется для целых чисел, без дробной части.
         * Пример: age = 10, count = 100, temperature = -5
       - **float (floating-point number - число с плавающей точкой):** Используется для чисел с дробной частью (с точкой).
         * Пример: pi = 3.14, price = 49.99, distance = 15.5
       - **str (string - строка):** Используется для хранения текстовой информации. Строки всегда заключаются в кавычки (одинарные ' или двойные ").
         * Пример: name = "Боб", message = 'Привет, как дела?', city = "Нью-Йорк"
     + **Функция type():** Позволяет узнать тип данных любой переменной.
       - Синтаксис: type(имя\_переменной)
       - Пример:

Python

my\_number = 10

print(type(my\_number)) # Выведет: <class 'int'>

my\_text = "Python"

print(type(my\_text)) # Выведет: <class 'str'>

* + **Демонстрация:** Создание переменных разных типов, присвоение значений, вывод их значений и типов на экран с помощью print() и type().

1. **Практические задания (30 минут)**
   * **Легкий уровень:**
     + Создайте переменную city и присвойте ей название вашего города (например, "Бишкек"). Выведите значение этой переменной на экран.
     + Создайте переменную my\_age и присвойте ей ваш возраст (например, 14). Выведите значение my\_age на экран.
     + Проверьте тип данных обеих созданных переменных с помощью type().
   * **Средний уровень:**
     + **Программа для расчета общей стоимости:**
       1. Создайте переменную item\_price (цена товара, например, 250.75).
       2. Создайте переменную item\_quantity (количество, например, 3).
       3. Вычислите общую стоимость (total\_cost = item\_price \* item\_quantity) и выведите её на экран, например: "Общая стоимость: [значение]".
     + **Программа для расчета возраста (автоматически):**
       1. Создайте переменную current\_year и присвойте ей текущий год (например, 2025).
       2. Создайте переменную birth\_year и присвойте ей год своего рождения (например, 2010).
       3. Вычислите свой возраст (age = current\_year - birth\_year).
       4. Выведите на экран сообщение: "Ваш возраст: [вычисленное\_значение] лет."
   * **Сложный уровень:**
     + **"Игра в шифровку":** Напишите программу, которая принимает (пока просто создайте переменную) слово, например, word = "PYTHON". А затем выводит его же, но после каждой буквы добавляет число, хранящееся в переменной secret\_number = 7.
       - Пример вывода: P7Y7T7H7O7N7
       - *(Подсказка: пока без циклов. Вам нужно будет "склеить" каждую букву слова с числом, которое предварительно нужно превратить в строку с помощью str(). Например, print(word[0] + str(secret\_number) + word[1] + str(secret\_number) + ...)*
     + **Игровой элемент: "Математический квест":**
       - Создайте переменные для персонажа игры: player\_hp = 100 (здоровье), monster\_damage = 25 (урон, который наносит монстр).
       - Вычислите, сколько раз монстр может нанести урон, прежде чем "победить" игрока (используйте целочисленное деление //).
       - Выведите на экран: "У игрока было [player\_hp] здоровья. Монстр наносит [monster\_damage] урона. Игрок сможет выдержать [количество\_ударов] ударов."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы учащихся.
   * Краткий обзор: что такое переменные, зачем нужны типы данных (int, float, str), как использовать оператор присваивания =.
   * Превью следующего урока: "На следующем занятии мы научимся не только хранить данные, но и выполнять с ними разные операции – складывать, вычитать, умножать, делить. И даже делать так, чтобы компьютер "спрашивал" у нас информацию!"

**Материалы/Ресурсы:**

* Презентация с примерами переменных и типов данных.
* Схема "Коробки для данных" для объяснения переменных.
* Примеры кода для демонстрации type().

**Домашнее задание:**

* Создать переменные для информации о себе (имя, фамилия, любимый цвет, любимое число) и вывести их на экран в разных print() командах.
* Написать программу, которая считает, сколько яблок останется, если из 10 яблок съесть 3. Используйте переменные для количества яблок и вычитания.

## Урок №3: Базовые операции и ввод/вывод данных.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Изучить основные арифметические операции в Python и порядок их выполнения.
* Научиться выполнять операции со строками: "склеивание" (конкатенация) и повторение.
* Освоить получение текстовых и числовых данных от пользователя с помощью функции input().
* Понять и применять преобразование типов данных (int(), float(), str()) при работе с введенными данными.

**Основные понятия:**

* Арифметические операторы (+, -, \*, /, %, //, \*\*)
* Приоритет операций
* Конкатенация строк
* Умножение строк
* Функция input()
* Преобразование типов (int(), float(), str())

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое переменная? Какие типы данных мы знаем и для чего они нужны? Как мы присваиваем значение переменной?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Арифметические операции (25 минут)**
   * **Теоретический материал:**
     + Python умеет выполнять все стандартные математические операции.
     + **Операторы:**
       - + : Сложение (Например: 5 + 3 будет 8)
       - - : Вычитание (Например: 10 - 4 будет 6)
       - \* : Умножение (Например: 2 \* 6 будет 12)
       - / : Деление (**Всегда возвращает число с плавающей точкой (float)!** Например: 10 / 3будет 3.333...)
       - % : Остаток от деления (Модуль. Находит остаток после деления. Полезно для проверки четности/нечетности. Например: 10 % 3 будет 1 (10 разделить на 3 = 3 и 1 в остатке), 4 % 2 будет 0)
       - // : Целочисленное деление (Делит и отбрасывает дробную часть. Например: 10 // 3будет 3, 9 // 2 будет 4)
       - \*\* : Возведение в степень (Например: 2 \*\* 3 будет 8 (2 в степени 3, т.е. 2\*2\*2))
     + **Приоритет операций (порядок выполнения):** Как в математике: сначала скобки (), затем степени \*\*, потом умножение/деление/модуль/целочисленное деление (\*, /, %, //), и в конце сложение/вычитание (+, -).
       - Пример: 2 + 3 \* 4 будет 14, а не 20. (2 + 3) \* 4 будет 20.
   * **Демонстрация:** Использование всех операторов с числами и переменными. Показать примеры с приоритетом операций.
3. **Операции со строками (15 минут)**
   * **Теоретический материал:**
     + Строки – это тоже данные, и с ними тоже можно выполнять операции!
     + **Конкатенация (+):** "Склеивание" или объединение двух или более строк в одну.
       - Пример:

Python

first\_name = "Анна"

last\_name = "Иванова"

full\_name = first\_name + " " + last\_name

print(full\_name) # Выведет: Анна Иванова

* + - **Умножение (\*):** Повторение строки заданное количество раз.
      * Пример:

Python

line = "=" \* 20

print(line) # Выведет: ====================

* + - * Важно: умножать строку можно только на целое число!
  + **Демонстрация:** Примеры конкатенации и умножения строк.

1. **Получение данных от пользователя (input()) (25 минут)**
   * **Теоретический материал:**
     + **Функция input():** Позволяет программе "спросить" что-то у пользователя и получить ответ. То, что введет пользователь, становится значением, которое мы можем сохранить в переменную.
     + **Синтаксис:** переменная = input("Сообщение для пользователя: ")
     + **Пример:**

Python

user\_name = input("Введите ваше имя: ")

print("Привет, " + user\_name + "!")

* + - **Очень ВАЖНО: input() всегда возвращает СТРОКУ!** Даже если пользователь ввел число, Python будет считать его текстом.
      * Пример: Если пользователь ввел 5 в input(), для Python это будет "5", а не число 5.
    - **Преобразование типов:** Если нам нужно работать с введенным числом как с числом (например, для сложения), мы должны его "преобразовать" в числовой тип.
      * int(значение): Преобразует в целое число.
        + Пример: age = int(input("Введите ваш возраст: "))
      * float(значение): Преобразует в число с плавающей точкой.
        + Пример: price = float(input("Введите цену: "))
      * str(значение): Преобразует в строку (полезно, если нужно "склеить" число с текстом).
        + Пример: message = "У вас " + str(count) + " яблок"
  + **Демонстрация:** Показать работу input(), затем продемонстрировать проблему сложения строк ("5" + "3" будет "53", а не 8) и показать, как int() решает эту проблему.

1. **Практические задания (10 минут)**
   * **Легкий уровень:**
     + Создайте программу, которая спрашивает "Какое ваше любимое животное?" и затем выводит "Мое любимое животное - [ответ пользователя]!". Используйте input().
   * **Средний уровень:**
     + **Простой калькулятор сложения:**
       1. Спросите у пользователя первое число.
       2. Спросите у пользователя второе число.
       3. Преобразуйте введенные данные в числа (используйте int() или float()).
       4. Сложите их и выведите результат, например: "Сумма ваших чисел: [результат]".
   * **Сложный уровень:**
     + **"Генератор истории":**
       1. Программа запрашивает у пользователя (с помощью input()):
          - Имя главного героя
          - Название любимого животного
          - Любимый цвет
       2. Затем составляет короткий рассказ на основе этих данных (используя конкатенацию строк и переменные).
       - Пример рассказа: "Однажды [Имя главного героя] отправился в путешествие со своим [любимое животное]. У них был [любимый цвет] рюкзак, и они искали приключения!"
     + **Игровой элемент: "Калькулятор монстра":**
       - Задание: "В магическом лесу живут монстры. У каждого монстра по 3 лапы. Спросите у пользователя, сколько монстров он встретил. А затем вычислите и выведите, сколько всего лап у всех монстров вместе взятых."
       - Пример:

Python

num\_monsters = int(input("Сколько монстров вы встретили? "))

total\_legs = num\_monsters \* 3

print("У всех монстров вместе " + str(total\_legs) + " лап.")

1. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: основные операции, input(), int(), float(), str().
   * Превью следующего урока: "На следующем занятии мы узнаем, как заставить компьютер принимать решения и выбирать разные пути выполнения программы – это очень важный шаг к созданию "умных" программ!"

**Материалы/Ресурсы:**

* Презентация с символами операторов и их объяснением.
* Схема работы input() и преобразования типов.
* Примеры кода для демонстрации.

**Домашнее задание:**

* Написать программу, которая запрашивает у пользователя его имя и возраст, а затем выводит сообщение: "Привет, [Имя]! Тебе [Возраст] лет.". (Используйте input() и преобразование типов при необходимости).
* Написать программу для расчета площади круга: запросить радиус у пользователя, использовать 3.14 как значение Pi (можно создать для этого переменную), вывести площадь.

## Урок №4: Условный оператор if/else.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять концепцию условных операторов и их необходимость для создания "умных" программ.
* Изучить синтаксис и работу операторов if и else.
* Научиться использовать операторы сравнения (равно, не равно, больше, меньше и т.д.).
* Понять важность и правильное использование отступов в Python для обозначения блоков кода.

**Основные понятия:**

* Условный оператор
* Условие
* if (если)
* else (иначе)
* Операторы сравнения: == (равно), != (не равно), > (больше), < (меньше), >= (больше или равно), <= (меньше или равно)
* Булевы значения (True, False)
* Блок кода
* Отступы (indentation)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Как получить данные от пользователя? Зачем нужно преобразовывать типы (int(), float())? Какие арифметические и строковые операции мы знаем?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Зачем нужны условия? (15 минут)**
   * **Теоретический материал:**
     + **Программы принимают решения:** До сих пор наши программы выполняли команды строго по порядку. Но реальные задачи требуют, чтобы программа могла "выбирать" разные действия в зависимости от ситуации.
     + **Аналогия:** "Представьте, что вы программируете робота. Вы говорите ему: 'Если (if) на улице дождь, тогда (:) возьми зонт. Иначе (else) надень кепку.'"
     + **Примеры из повседневной жизни:**
       - Светофор: "Если свет красный, стоп. Иначе, иди."
       - Вход по паролю: "Если пароль верный, пропустить. Иначе, отказать."
       - Игра: "Если игрок нажал кнопку 'прыжок', то персонаж прыгает."
     + **Задача условных операторов:** Позволить программе выполнять разный код в зависимости от того, истинно (True) или ложно (False) какое-либо условие.
3. **Операторы сравнения (15 минут)**
   * **Теоретический материал:**
     + Чтобы компьютер мог "принимать решения", ему нужно уметь сравнивать значения. Результатом любого сравнения будет **булево значение**: True (Истина) или False (Ложь).
     + **Основные операторы сравнения:**
       - == : Равно (Проверяет, равны ли два значения. Пример: 5 == 5 -> True, 5 == 7 -> False)
       - != : Не равно (Проверяет, не равны ли два значения. Пример: 5 != 7 -> True, 5 != 5 -> False)
       - > : Больше (Пример: 10 > 5 -> True)
       - < : Меньше (Пример: 3 < 8 -> True)
       - >= : Больше или равно (Пример: 5 >= 5 -> True, 5 >= 7 -> False)
       - <= : Меньше или равно (Пример: 5 <= 5 -> True, 5 <= 3 -> False)
     + **Примеры использования:**

Python

age = 15

is\_adult = (age >= 18) # is\_adult будет False

print(is\_adult)

* + **Демонстрация:** Проверка каждого оператора сравнения в консоли с различными значениями и вывод True/False.

1. **Условный оператор if/else и отступы (25 минут)**
   * **Теоретический материал:**
     + **Синтаксис if:**

Python

if условие:

# Этот блок кода выполняется, если условие ИСТИННО (True)

# Важно: здесь должен быть ОТСТУП!

* + - * **if:** Ключевое слово, начинает условный оператор.
      * **условие:** Выражение, которое возвращает True или False (обычно сравнение).
      * **: (двоеточие):** Обязательный символ после условия.
      * **Блок кода:** Строки, которые выполняются при истинном условии. **Они должны быть сдвинуты вправо (иметь отступ)!**
    - **Что такое "отступы" (indentation):** В Python отступы (обычно 4 пробела или 1 Tab) используются для обозначения принадлежности строк к определенному блоку кода (например, к if, else, функциям, циклам). Это критически важно! Если отступы неверные, программа выдаст ошибку.
      * Пример правильного отступа:

Python

if 10 > 5:

print("10 больше 5") # Этот print с отступом

print("Конец программы") # Этот print без отступа, выполняется всегда

* + - **Синтаксис else:**

Python

if условие:

# Блок A: выполняется, если условие ИСТИННО

else:

# Блок B: выполняется, если условие ЛОЖНО (False)

# Тоже должен быть ОТСТУП!

* + - * **else:** Ключевое слово, указывает на альтернативное действие. Выполняется, если условие if было ложным.
      * **: (двоеточие):** Обязательный символ после else.
    - **Примеры:**
      * Проверка, является ли число положительным:

Python

number = 7

if number > 0:

print("Число положительное")

else:

print("Число неположительное (отрицательное или ноль)")

* + - * Простой вход по "паролю":

Python

secret\_password = "python"

user\_input = input("Введите пароль: ")

if user\_input == secret\_password:

print("Доступ разрешен!")

else:

print("Неверный пароль!")

* + **Демонстрация:** Показать примеры if и else. Намеренно сделать ошибку с отступами, чтобы показать, как Python реагирует на это.

1. **Практические задания (20 минут)**
   * **Легкий уровень:**
     + Напишите программу, которая спрашивает у пользователя, сколько будет 2+2 (пусть ответ будет 4). Если пользователь ввел 4, выведите "Правильно!". Иначе - "Неправильно!".
     + *Подсказка: не забудьте преобразовать ввод пользователя в число!*
   * **Средний уровень:**
     + Программа, которая определяет, четное или нечетное число ввел пользователь.
       - *Подсказка: используйте оператор % (остаток от деления) с числом 2. Если остаток 0, число четное.*
       - Пример:

Python

num = int(input("Введите число: "))

if num % 2 == 0:

print("Это четное число.")

else:

print("Это нечетное число.")

* + **Сложный уровень:**
    - **"Волшебная шляпа":** Напишите программу, которая:
      1. Спрашивает у пользователя возраст (используйте input() и int()).
      2. Если возраст пользователя **больше или равен 12**, программа выводит "Добро пожаловать в Школу Магии!".
      3. Иначе (если возраст **меньше 12**), выводит "Вы еще слишком юны для магии. Приходите, когда подрастете!".
    - **Игровой элемент: "Игра 'Больше-меньше'":**
      * Программа "загадывает" фиксированное число (например, secret\_number = 7).
      * Пользователь вводит свое предположение (user\_guess = int(input("Угадайте число от 1 до 10: "))).
      * Используя if и else, программа должна сказать:
        + "Больше!" (если user\_guess < secret\_number)
        + "Меньше!" (если user\_guess > secret\_number)
        + "Угадал!" (если user\_guess == secret\_number) *(Пока без циклов, только одно предположение и ответ)*

1. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: if/else для принятия решений, операторы сравнения, важность отступов.
   * Превью следующего урока: "На следующем занятии мы узнаем, как делать еще более сложные выборы в программе, когда есть не только два варианта, но и множество разных путей – для этого нам понадобится оператор elif!"

**Материалы/Ресурсы:**

* Презентация с блок-схемами для if/else.
* Примеры кода с правильными и неправильными отступами для демонстрации.

**Домашнее задание:**

* Написать программу, которая просит пользователя ввести два числа и выводит сообщение, какое из них больше, или они равны.
* Написать программу "Возрастной ценз для фильма": если возраст пользователя меньше 16, выведите "Извините, этот фильм для вас недоступен!". Иначе - "Приятного просмотра!".

## Урок №5: Условный оператор elif и логические операторы.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Расширить понимание условных операторов с помощью elif.
* Изучить, как использовать логические операторы (and, or, not) для создания сложных условий.
* Научиться комбинировать операторы сравнения и логические операторы.
* Закрепить понимание отступов и структуры условных конструкций.

**Основные понятия:**

* elif (else if - иначе если)
* Логические операторы: and (И), or (ИЛИ), not (НЕ)
* Сложные условия
* Вложенные if (краткое знакомство)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Для чего нужен if и else? Что такое операторы сравнения? Почему важны отступы?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Условный оператор elif (25 минут)**
   * **Теоретический материал:**
     + До сих пор у нас было только два варианта: "если" (if) и "иначе" (else). Но часто в жизни бывает больше двух вариантов выбора. Например, оценки в школе: A, B, C, D, F.
     + elif позволяет проверять несколько условий по порядку. Если первое if условие ложно, Python переходит к проверке следующего elif. Если и оно ложно, то к следующему elif и так далее, пока не найдет истинное условие или не дойдет до else (который выполняется, если все предыдущие if и elif были ложными).
     + **Синтаксис if-elif-else:**

Python

if условие1:

# Код, если условие1 истинно

elif условие2:

# Код, если условие1 ложно, но условие2 истинно

elif условие3:

# Код, если условие1 и условие2 ложны, но условие3 истинно

else:

# Код, если все предыдущие условия ложны

* + - **Порядок важен:** Условия проверяются сверху вниз. Как только одно условие оказывается True, соответствующий блок кода выполняется, и вся конструкция if-elif-else завершается. Остальные elif и else пропускаются.
    - **Пример:**

Python

temperature = 25

if temperature > 30:

print("Очень жарко!")

elif temperature > 20: # Проверяется, если temperature не > 30

print("Тепло.")

elif temperature > 10: # Проверяется, если temperature не > 30 и не > 20

print("Прохладно.")

else:

print("Холодно.")

* + **Демонстрация:** Программа для определения времени суток по введенному часу (ночь, утро, день, вечер).

1. **Логические операторы (and, or, not) (25 минут)**
   * **Теоретический материал:**
     + Иногда нам нужно проверять сразу несколько условий одновременно. Например, "Если и возраст подходит, И есть билет".
     + **and (И):** Условие истинно только тогда, когда **ОБА (или все) условия** истинны.
       - Пример: age >= 18 and has\_ticket == True (Оба должны быть True)
     + **or (ИЛИ):** Условие истинно, когда **ХОТЯ БЫ ОДНО** из условий истинно.
       - Пример: is\_weekend or is\_holiday (Достаточно, чтобы is\_weekend был True ИЛИ is\_holiday был True)
     + **not (НЕ):** Инвертирует (меняет на противоположное) логическое значение условия. not Trueстановится False, not False становится True.
       - Пример: not is\_raining (Если is\_raining было True, то not is\_raining будет False, и наоборот).
     + **Примеры использования:**

Python

age = 15

has\_permission = True

if age >= 12 and has\_permission:

print("Можно смотреть фильм.")

else:

print("Нельзя смотреть фильм.")

weather = "солнечно"

mood = "хорошее"

if weather == "солнечно" or mood == "хорошее":

print("Отличный день!")

* + **Демонстрация:** Простые примеры с and, or, not и комбинациями.

1. **Практические задания (25 минут)**
   * **Легкий уровень:**
     + Программа, которая спрашивает у пользователя число от 1 до 10.
       - Если число меньше 5, выведите "Маленькое число".
       - Если число от 5 до 7 (включительно), выведите "Среднее число".
       - Если число больше 7, выведите "Большое число". (Используйте if-elif-else).
   * **Средний уровень:**
     + **Оценка по баллам:** Спросите у пользователя количество баллов (от 0 до 100).
       - Если баллов 90-100: "Оценка A"
       - 80-89: "Оценка B"
       - 70-79: "Оценка C"
       - 60-69: "Оценка D"
       - Менее 60: "Оценка F"
       - *Подсказка: используйте and для диапазонов.*
   * **Сложный уровень:**
     + **"Проверка на совершеннолетие и гражданство":**
       1. Спросите у пользователя возраст.
       2. Спросите, является ли он гражданином (ответ "да" или "нет").
       3. Если возраст >= 18 **И** он гражданин: "Вы имеете право голосовать."
       4. Если возраст < 18 **ИЛИ** он не гражданин: "Вы не имеете право голосовать."
       - *Подсказка: для "да" / "нет" сравнивайте со строками, для возраста - с числом.*
     + **Игровой элемент: "Детективная логика":**
       - Задание: "Вы детектив. У вас есть подозреваемые. Только один из них убийца.
         * Убийца **высокий (рост > 180 см) ИЛИ носит шляпу**.
         * Убийца **НЕ блондин**.
         * Создайте переменные для характеристик подозреваемого (рост, носит\_шляпу (True/False), цвет\_волос) и с помощью if и логических операторов определите, может ли он быть убийцей."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: elif, and, or, not.
   * Превью следующего урока: "На следующем занятии мы научимся повторять действия много-много раз, не переписывая один и тот же код снова и снова – это называется циклами!"

**Материалы/Ресурсы:**

* Презентация с блок-схемами для if-elif-else.
* Таблицы истинности для and, or, not.
* Примеры кода для демонстрации сложных условий.

**Домашнее задание:**

* Написать программу, которая запрашивает у пользователя день недели (число от 1 до 7) и выводит, является ли этот день выходным (6 и 7) или рабочим.
* Создать программу "Проверка возраста для аттракциона":
  + Если возраст меньше 6 лет, вывести "Слишком молод".
  + Если возраст от 6 до 12 лет, вывести "Можно только со взрослыми".
  + Если возраст больше или равен 13, вывести "Добро пожаловать!".

## Урок №6: Цикл while.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять концепцию циклов и их необходимость для автоматизации повторяющихся задач.
* Изучить синтаксис и работу цикла while.
* Научиться создавать условия выхода из цикла, чтобы избежать бесконечных циклов.
* Применять цикл while для решения задач, требующих повторения действий.

**Основные понятия:**

* Цикл (loop)
* while (пока)
* Условие цикла
* Итерация
* Бесконечный цикл
* Изменение счетчика/условия

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Для чего нужны elif? Когда использовать and, а когда or? Приведите пример сложного условия.
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Зачем нужны циклы? (15 минут)**
   * **Теоретический материал:**
     + Представьте, что вам нужно вывести "Привет" 100 раз. Или попросить пользователя ввести пароль, пока он не будет верным. Писать 100 раз print("Привет") – неэффективно.
     + **Цикл (loop):** Это способ многократно повторять один и тот же блок кода. Это как "повтори, пока..." в Scratch, но теперь с текстовым кодом.
     + Циклы делают программы короче, эффективнее и мощнее, позволяя выполнять повторяющиеся задачи автоматически.
   * **Аналогия:** "Робот-повар: 'Повторяй, ПОКА тесто не станет гладким: мешай тесто'. Или 'Повторяй, ПОКА все тарелки не будут чистыми: мой тарелку'."
3. **Цикл while (25 минут)**
   * **Теоретический материал:**
     + **while (пока):** Цикл while выполняет блок кода до тех пор, пока заданное условие является истинным (True). Как только условие становится ложным (False), цикл завершается, и программа переходит к следующему коду после цикла.
     + **Синтаксис:**

Python

while условие:

# Этот блок кода выполняется, пока условие истинно

# Важно: здесь должен быть ОТСТУП!

# Обязательно нужно что-то изменить в условии,

# чтобы цикл когда-нибудь закончился!

* + - **Пример: Счетчик от 1 до 5:**

Python

count = 1

while count <= 5: # Пока count меньше или равен 5

print(count)

count = count + 1 # Увеличиваем count на 1

print("Цикл завершен.")

* + - **Бесконечный цикл:** Это ситуация, когда условие цикла всегда остается True, и цикл никогда не заканчивается. Программа "зависает". **Очень важно** всегда продумывать, как условие цикла изменится и в какой момент станет False.
      * Пример бесконечного цикла: while True: print("Привет") (или если не менять count в примере выше).
  + **Демонстрация:** Продемонстрировать простой счетчик while. Показать, что происходит, если забыть изменить переменную условия (бесконечный цикл), и как его остановить (обычно Ctrl+C в консоли).

1. **Практические задания (25 минут)**
   * **Легкий уровень:**
     + Напишите программу, которая выводит числа от 1 до 10 с помощью цикла while.
   * **Средний уровень:**
     + **"Обратный отсчет":** Напишите программу, которая запрашивает у пользователя начальное число, а затем считает от этого числа до 1, выводя каждое число на экран. После 1 выводит "Пуск!".
       - Пример: Ввод 5 -> Вывод: 5, 4, 3, 2, 1, Пуск!
   * **Сложный уровень:**
     + **"Игра 'Угадай число' (простая версия)":**
       1. Программа "загадывает" фиксированное число (например, secret\_number = 42).
       2. В цикле while программа постоянно спрашивает у пользователя "Угадайте число: ".
       3. Если пользователь угадал, программа выводит "Поздравляю, вы угадали!" и цикл завершается.
       4. Если не угадал, программа говорит "Неверно, попробуйте еще раз." и продолжает спрашивать.
       - *Подсказка: используйте input() внутри цикла и условие user\_guess != secret\_number для while.*
     + **Игровой элемент: "Магическая фраза":**
       - Задание: "Вы волшебник, и чтобы открыть дверь, нужно произнести 'секретное заклинание'. Используйте цикл while, чтобы спрашивать у пользователя 'Каково секретное заклинание?' до тех пор, пока он не введет слово 'Абракадабра'."
       - После правильного ввода вывести "Дверь открыта!"
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: зачем нужны циклы, как работает while, как избежать бесконечных циклов.
   * Превью следующего урока: "На следующем занятии мы познакомимся с другим очень мощным циклом, который идеально подходит для работы с последовательностями, например, со строками и списками – циклом for!"

**Материалы/Ресурсы:**

* Презентация с блок-схемами для цикла while.
* Примеры кода для демонстрации различных вариантов использования while.

**Домашнее задание:**

* Написать программу, которая выводит на экран все нечетные числа от 1 до 15 с помощью цикла while.
* Создать программу "Копилка":
  1. Начальный баланс money = 0.
  2. Программа спрашивает: "Сколько денег вы хотите добавить в копилку? (0 для выхода)".
  3. Цикл продолжается, пока пользователь не введет 0.
  4. После выхода вывести "Ваш баланс: [итоговый\_баланс]".

## Урок №7: Цикл for и итерируемые объекты.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять принцип работы цикла for для перебора элементов.
* Изучить понятие "итерируемый объект" (строки, range()).
* Научиться использовать цикл for с функцией range() для выполнения действий заданное количество раз.
* Применять цикл for для обработки символов в строках.

**Основные понятия:**

* for (для каждого)
* Итерируемый объект
* Итерация (повторение)
* range() (диапазон)
* Элемент последовательности

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Для чего нужен цикл while? Как избежать бесконечного цикла? Приведите пример использования while.
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Зачем нужен цикл for? (15 минут)**
   * **Теоретический материал:**
     + Цикл while хорош, когда мы не знаем заранее, сколько раз нужно что-то повторять (например, "пока пароль неверный").
     + Но часто мы знаем, сколько раз нужно повторить, или хотим пройтись по каждому элементу в какой-то последовательности (например, по каждой букве в слове, или по каждому числу от 1 до 10).
     + **Цикл for:** Идеально подходит для перебора элементов в **итерируемых объектах** (то, что можно "перебрать" по очереди: строки, списки, диапазоны чисел).
   * **Аналогия:** "Робот-уборщик: 'Для каждой комнаты в доме: пропылесось комнату'. Или 'Для каждого фрукта в корзине: проверь, не испорчен ли он'."
3. **Цикл for с range() (25 минут)**
   * **Теоретический материал:**
     + **Синтаксис for:**

Python

for переменная\_элемента in итерируемый\_объект:

# Код, который выполняется для каждого элемента

# Важно: здесь тоже должен быть ОТСТУП!

* + - * переменная\_элемента: На каждом шаге цикла в эту переменную будет поочередно записываться текущий элемент из итерируемый\_объект.
    - **Функция range():** Очень полезная функция, которая создает последовательность чисел. Идеально подходит для повторения действий заданное количество раз.
      * range(stop): Генерирует числа от 0 до stop-1.
        + Пример: range(5) даст 0, 1, 2, 3, 4.
      * range(start, stop): Генерирует числа от start до stop-1.
        + Пример: range(1, 6) даст 1, 2, 3, 4, 5.
      * range(start, stop, step): Генерирует числа с заданным шагом.
        + Пример: range(0, 10, 2) даст 0, 2, 4, 6, 8.
    - **Пример использования for с range():**

Python

for i in range(3): # i будет 0, потом 1, потом 2

print("Привет!") # Выведет "Привет!" 3 раза

for num in range(1, 6): # num будет 1, 2, 3, 4, 5

print(num)

* + **Демонстрация:** Примеры for с разными вариантами range().

1. **Цикл for со строками (15 минут)**
   * **Теоретический материал:**
     + Строка - это последовательность символов, и мы можем перебирать каждый символ строки с помощью for.
     + Пример:

Python

word = "Python"

for char in word: # char будет 'P', потом 'y', потом 't' и т.д.

print(char)

* + - Это очень удобно для обработки текста, анализа букв и т.д.
  + **Демонстрация:** Программа, которая выводит каждую букву слова на новой строке.

1. **Практические задания (20 минут)**
   * **Легкий уровень:**
     + Напишите программу, которая с помощью цикла for выводит все числа от 1 до 5.
     + Напишите программу, которая выводит каждую букву вашего имени на новой строке.
   * **Средний уровень:**
     + **"Поздравление 5 раз":** Спросите у пользователя его имя. Затем выведите "С днем рождения, [Имя]!" 5 раз, используя цикл for и range().
     + **"Гласные буквы":** Запросите у пользователя слово. С помощью цикла for пройдитесь по каждой букве слова. Если буква является гласной (а, е, и, о, у, ы, э, ю, я), выведите её на экран.
       - *Подсказка: используйте if и or для проверки гласных.*
   * **Сложный уровень:**
     + **"Таблица умножения":** Запросите у пользователя число. Выведите таблицу умножения для этого числа от 1 до 10.
       - Пример для числа 7:
       - 7 \* 1 = 7
       - 7 \* 2 = 14
       - ...
       - 7 \* 10 = 70
     + **Игровой элемент: "Генератор пароля":**
       - Задание: "Создайте программу, которая генерирует простой "пароль" из повторяющихся символов. Запросите у пользователя символ (например, '\*') и количество повторений (например, 10). С помощью цикла for и range() выведите этот символ заданное количество раз в одной строке."
       - *Подсказка: используйте print(char, end='') чтобы не переходить на новую строку после каждого символа.*
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: for для перебора, range() для повторений, перебор строк.
   * Превью следующего урока: "На следующем занятии мы углубимся в работу со строками, узнаем, как изменять их, искать символы, делать их большими или маленькими. Строки – это очень важный тип данных для работы с текстом!"

**Материалы/Ресурсы:**

* Презентация с блок-схемами для цикла for.
* Объяснение работы range() с разными параметрами.
* Примеры кода для перебора строк.

**Домашнее задание:**

* Написать программу, которая выводит все числа от 20 до 0 с шагом -2, используя for и range().
* Запросить у пользователя любое слово. С помощью цикла for вывести это слово по одной букве, но каждую вторую букву сделать заглавной (это задание для очень продвинутых, можно дать как "звездочку").
* Создать программу "Пирамида из символов": запросить символ у пользователя, затем вывести его в виде пирамиды (например, 5 строк):
* \*
* \*\*
* \*\*\*
* \*\*\*\*
* \*\*\*\*\*

## Урок №8: Методы строк.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомиться с понятием "методы" как действий, которые можно выполнять с объектами (в данном случае, со строками).
* Изучить основные методы строк: upper(), lower(), capitalize(), find(), replace(), strip().
* Научиться применять методы строк для манипулирования текстовыми данными в программах.
* Понять, что строки в Python неизменяемы.

**Основные понятия:**

* Метод строки
* upper()
* lower()
* capitalize()
* find()
* replace()
* strip()
* Неизменяемые объекты (immutable)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Для чего нужен цикл for? Что такое range()? Как перебрать символы в строке с помощью for?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Что такое методы строк? (15 минут)**
   * **Теоретический материал:**
     + Мы уже знаем, что строки – это текст. Но что, если мы хотим изменить этот текст? Сделать все буквы большими? Найти определенное слово?
     + **Метод:** Это как "функция", которая "принадлежит" определенному типу данных (в нашем случае, строкам). Методы позволяют выполнять специфические действия с данными этого типа.
     + **Синтаксис вызова метода:** переменная\_строки.имя\_метода() или переменная\_строки.имя\_метода(параметры)
     + Аналогия: "У смартфона есть разные функции (методы): позвонить, отправить СМС, сделать фото. Вы вызываете их так: телефон.позвонить(номер), телефон.сделать\_фото()."
     + **Важно:** Строки в Python **неизменяемы (immutable)**. Это значит, что методы строк не изменяют саму исходную строку, а **возвращают новую строку** с изменениями.
   * **Демонстрация:** Показать, что my\_string.upper() не меняет my\_string, а нужно присвоить результат новой переменной: new\_string = my\_string.upper().
3. **Основные методы строк (40 минут)**
   * **Теоретический материал:**
     + **upper():** Возвращает новую строку, где все буквы переведены в верхний регистр (заглавные).
       - Пример: text = "hello"; print(text.upper()) -> HELLO
     + **lower():** Возвращает новую строку, где все буквы переведены в нижний регистр (строчные).
       - Пример: text = "HELLO"; print(text.lower()) -> hello
     + **capitalize():** Возвращает новую строку, где первый символ переведен в верхний регистр, а остальные в нижний.
       - Пример: text = "привет мир"; print(text.capitalize()) -> Привет мир
     + **find(sub):** Ищет первое вхождение подстроки sub в строке и возвращает индекс (позицию) ее начала. Если подстрока не найдена, возвращает -1. (Индексы начинаются с 0).
       - Пример: text = "Python"; print(text.find("o")) -> 4; print(text.find("z")) -> -1
     + **replace(old, new):** Возвращает новую строку, в которой все вхождения подстроки oldзаменены на new.
       - Пример: text = "Я люблю кошек"; print(text.replace("кошек", "собак")) -> Я люблю собак
     + **strip():** Возвращает новую строку, в которой удалены пробелы (или другие символы по умолчанию) с начала и конца строки. Очень полезно для очистки ввода пользователя.
       - Пример: text = " Привет, мир! "; print(text.strip()) -> Привет, мир!
     + **len() (не метод, а функция):** Возвращает длину строки (количество символов).
       - Пример: word = "hello"; print(len(word)) -> 5
   * **Демонстрация:** Показать работу каждого метода с примерами кода. Обратить внимание, что методы вызываются с помощью точки после имени переменной.
4. **Практические задания (20 минут)**
   * **Легкий уровень:**
     + Запросите у пользователя слово. Выведите это слово в верхнем регистре, а затем в нижнем регистре.
     + Запросите предложение. Найдите позицию первой буквы "а" (или любой другой) в этом предложении.
   * **Средний уровень:**
     + **"Цензор слов":** Запросите у пользователя предложение. Если в предложении есть слово "плохо" (или "ужасно", "некрасиво"), замените его на "хорошо" (или "отлично", "красиво") и выведите измененное предложение. Используйте replace().
     + **"Форматирование имени":** Запросите у пользователя его имя. Убедитесь, что оно выведено с заглавной первой буквой и остальными строчными, даже если пользователь ввел его неправильно (например, "иван" -> "Иван", "МАРиЯ" -> "Мария"). Используйте capitalize() и strip().
   * **Сложный уровень:**
     + **"Проверка на спам":** Запросите у пользователя сообщение. Если сообщение содержит слова "купить" ИЛИ "деньги" (в любом регистре), выведите "Это похоже на спам!". Иначе - "Сообщение принято.".
       - *Подсказка: используйте lower() для сообщения, чтобы упростить проверку, и find()или оператор in.*
     + **Игровой элемент: "Шифровальщик сообщений":**
       - Задание: "Создайте программу, которая "шифрует" короткое сообщение. Запросите у пользователя сообщение. Замените все буквы 'е' на '3', а все буквы 'о' на '0'. Выведите зашифрованное сообщение."
       - *Например: "Привет, Олег!" -> "Прив3т, 0л3г!"*
5. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: что такое методы, основные методы строк, концепция неизменяемости.
   * Превью следующего урока: "На следующем занятии мы познакомимся с новой и очень мощной структурой данных – списками! Списки позволяют хранить множество значений в одной переменной, что очень удобно для организации данных."

**Материалы/Ресурсы:**

* Презентация с таблицей основных методов строк и их описанием.
* Примеры кода для демонстрации каждого метода.

**Домашнее задание:**

* Написать программу, которая запрашивает у пользователя email-адрес. Проверьте, есть ли в нем символ @. Если нет, выведите "Некорректный email-адрес!".
* Создать программу "Повторитель слова": запросить слово у пользователя, затем запросить число. Вывести слово заданное число раз, используя умножение строк.
* Написать программу, которая убирает лишние пробелы в начале и конце введенной пользователем фразы, а затем выводит ее в верхнем регистре.

## Урок №9: Списки (List). Основы работы со списками.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомить с новой структурой данных – списками (list) и их преимуществами перед отдельными переменными.
* Научиться создавать списки и получать доступ к элементам по индексу.
* Изучить операции добавления, изменения и удаления элементов списка.
* Понять, что списки в Python являются изменяемыми (mutable) объектами.

**Основные понятия:**

* Список (list)
* Элемент списка
* Индекс
* Изменяемый объект (mutable)
* Методы списков: append(), insert(), remove(), pop(), del (оператор)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое методы строк? Приведите примеры методов upper(), replace(). Почему строки неизменяемы?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Зачем нужны списки? (15 минут)**
   * **Теоретический материал:**
     + Представьте, что у нас есть 100 имен игроков или 500 цен на товары. Создавать для каждого отдельную переменную (player1 = "Иван", player2 = "Мария", ...) – это неудобно и неэффективно.
     + **Список (List):** Это упорядоченная коллекция элементов, которая позволяет хранить множество значений в одной переменной. Элементами списка могут быть данные любых типов (числа, строки, даже другие списки).
     + **Аналогия:** "Список покупок", "Список класса", "Коробочки, собранные в одну большую коробку с ячейками".
     + **Создание списка:** Списки обозначаются квадратными скобками [], а элементы внутри разделяются запятыми.
       - Пример: my\_list = [1, 2, 3, 4, 5]
       - Пример: names = ["Алиса", "Боб", "Чарли"]
       - Пример: mixed\_list = ["Привет", 123, True]
3. **Доступ к элементам списка (20 минут)**
   * **Теоретический материал:**
     + **Индекс (Index):** Каждый элемент в списке имеет свой номер (позицию), который называется индексом. **Нумерация в Python всегда начинается с 0!**
       - Пример: В списке ["яблоко", "банан", "апельсин"]
         * "яблоко" имеет индекс 0
         * "банан" имеет индекс 1
         * "апельсин" имеет индекс 2
     + **Доступ по индексу:** Чтобы получить элемент, используйте квадратные скобки после имени списка: имя\_списка[индекс].
       - Пример: fruits = ["яблоко", "банан", "апельсин"]
       - print(fruits[0]) # Выведет: яблоко
       - print(fruits[2]) # Выведет: апельсин
     + **Отрицательные индексы:** Позволяют получать элементы с конца списка. -1 - последний элемент, -2 - предпоследний и т.д.
       - Пример: print(fruits[-1]) # Выведет: апельсин
     + **len() для списков:** Функция len() также работает со списками, возвращая количество элементов в списке.
       - Пример: print(len(fruits)) # Выведет: 3
   * **Демонстрация:** Создание списка, доступ к элементам по положительным и отрицательным индексам. Использование len().
4. **Изменение, добавление и удаление элементов (30 минут)**
   * **Теоретический материал:**
     + **Изменяемые объекты (Mutable):** В отличие от строк, списки являются **изменяемыми**. Это означает, что мы можем добавлять, удалять или изменять элементы списка напрямую, без создания нового списка.
     + **Изменение элемента:** имя\_списка[индекс] = новое\_значение
       - Пример: fruits[1] = "вишня" # "банан" изменится на "вишню"
     + **Добавление элементов:**
       - список.append(элемент): Добавляет элемент в **конец** списка.
         * Пример: fruits.append("манго")
       - список.insert(индекс, элемент): Вставляет элемент по указанному индексу. Остальные элементы сдвигаются.
         * Пример: fruits.insert(1, "груша") # Груша будет на месте банана, банан сдвинется
     + **Удаление элементов:**
       - список.remove(значение): Удаляет **первое** найденное вхождение указанного значения. Если значения нет, будет ошибка.
         * Пример: fruits.remove("яблоко")
       - список.pop(индекс): Удаляет элемент по указанному индексу и **возвращает его значение**. Если индекс не указан, удаляет и возвращает последний элемент.
         * Пример: removed\_fruit = fruits.pop(0) # Удалит "яблоко" и сохранит его в removed\_fruit
       - del список[индекс]: Оператор del удаляет элемент по указанному индексу.
         * Пример: del fruits[1]
       - список.clear(): Удаляет все элементы из списка, делая его пустым.
   * **Демонстрация:** Продемонстрировать все операции на живых примерах, показывая состояние списка после каждой операции.
5. **Практические задания (10 минут)**
   * **Легкий уровень:**
     + Создайте список из 5 ваших любимых цветов. Выведите первый и последний цвет.
     + Добавьте в этот список еще один цвет.
     + Измените второй цвет в списке на новый.
   * **Средний уровень:**
     + Создайте список numbers = [10, 20, 30, 40, 50]. Удалите число 30 из списка. Затем добавьте число 25 на вторую позицию. Выведите итоговый список.
     + Создайте список tasks = ["Купить хлеб", "Позвонить маме", "Сделать уроки"]. Попросите пользователя ввести номер выполненной задачи (начиная с 1). Удалите выполненную задачу из списка. Выведите оставшиеся задачи.
       - *Подсказка: Не забудьте, что индексы начинаются с 0!*
   * **Сложный уровень:**
     + **"Список дел":** Напишите программу, которая позволяет пользователю:
       1. Добавить новую задачу в список.
       2. Удалить задачу по номеру.
       3. Показать весь список задач.
       - Используйте цикл while для меню программы и input() для выбора действий.
     + **Игровой элемент: "Инвентарь героя":**
       - Задание: "Создайте список inventory = ['меч', 'щит', 'зелье'].
         * Добавьте 'лук' в инвентарь.
         * Удалите 'щит'.
         * Поменяйте 'меч' на 'кинжал'.
         * Выведите итоговый инвентарь."
6. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: списки – удобный способ хранения данных, индексы, изменяемость, основные операции.
   * Превью следующего урока: "На следующем занятии мы научимся использовать циклы со списками, чтобы обрабатывать каждый элемент списка, и познакомимся с тем, как создавать свои собственные 'инструменты' – функции!"

**Материалы/Ресурсы:**

* Презентация с визуализацияцией списков и индексов.
* Таблица методов списков с примерами.

**Домашнее задание:**

* Создайте список shopping\_list = ["Молоко", "Хлеб", "Яйца"].
  + Добавьте "Сыр" в конец списка.
  + Вставьте "Масло" на первую позицию.
  + Удалите "Хлеб".
  + Выведите итоговый список.
* Написать программу, которая запрашивает у пользователя 5 названий книг, сохраняет их в список, а затем выводит каждую книгу из списка на новой строке.

## Урок №10: Циклы со списками. Введение в функции.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Научиться эффективно использовать циклы for для перебора элементов списков.
* Понять концепцию функций: зачем они нужны и как помогают организовать код.
* Изучить синтаксис определения функций (def).
* Научиться вызывать созданные функции и передавать им аргументы.

**Основные понятия:**

* Перебор списка (list iteration)
* Функция (function)
* def (define - определить)
* Вызов функции
* Аргумент функции
* Параметр функции
* Повторное использование кода (reusability)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое список? Как получить элемент по индексу? Как добавить/удалить элемент в списке?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Циклы for со списками (25 минут)**
   * **Теоретический материал:**
     + Цикл for – самый естественный и удобный способ пройтись по каждому элементу списка. На каждой итерации цикла, переменная, указанная после for, будет принимать значение текущего элемента списка.
     + **Синтаксис:**

Python

for элемент in мой\_список:

# Код, который будет выполнен для каждого элемента

* + - **Пример:**

Python

fruits = ["яблоко", "банан", "вишня"]

for fruit in fruits:

print("Я люблю " + fruit)

* + - **Использование range(len(list)) для доступа к индексам:** Иногда нужно знать не только сам элемент, но и его индекс. Для этого можно комбинировать for с range() и len().

Python

fruits = ["яблоко", "банан", "вишня"]

for i in range(len(fruits)): # i будет 0, 1, 2

print(f"Фрукт под индексом {i}: {fruits[i]}") # Используем f-строки для удобства

* + - Это позволяет удобно обрабатывать каждый элемент списка, выполнять над ним операции или выводить его.
  + **Демонстрация:** Продемонстрировать перебор списка с помощью for и получение элементов, а также пример с range(len()).

1. **Введение в функции (25 минут)**
   * **Теоретический материал:**
     + Представьте, что в вашей программе есть кусок кода, который вы используете много раз (например, вывод приветствия, или сложный расчет). Каждый раз копировать и вставлять его – неэффективно, это делает код длинным и сложным для изменения.
     + **Функция (Function):** Это именованный блок кода, который выполняет определенную задачу. Мы можем "определить" (создать) функцию один раз, а затем "вызывать" (использовать) её многократно в разных частях программы.
     + **Зачем нужны функции:**
       - **Повторное использование кода (Reusability):** Написал один раз, используй много раз.
       - **Организация кода:** Разделяет программу на логические части, делает её понятнее.
       - **Упрощение отладки:** Если ошибка в функции, исправляем её в одном месте.
     + **Определение функции (def):**
       - Функции определяются с помощью ключевого слова def, за которым следует имя функции, круглые скобки () и двоеточие :.
       - Блок кода внутри функции должен быть с отступом.
       - **Синтаксис:**

Python

def имя\_функции():

# Блок кода функции

print("Я внутри функции!")

* + - **Вызов функции:** Чтобы выполнить код внутри функции, нужно просто написать её имя с круглыми скобками.
      * Пример: имя\_функции()
  + **Демонстрация:** Создать простую функцию say\_hello() и показать, как её можно вызывать несколько раз.

1. **Функции с аргументами (15 минут)**
   * **Теоретический материал:**
     + Часто функции нужно давать какую-то информацию для работы. Например, функция print()принимает аргументы (текст, который нужно вывести).
     + **Аргументы (Arguments):** Это значения, которые мы передаем функции, когда её вызываем.
     + **Параметры (Parameters):** Это переменные, которые функция "ожидает" получить в скобках при своем определении. Аргументы, переданные при вызове, записываются в эти параметры.
     + **Синтаксис функции с аргументами:**

Python

def имя\_функции(параметр1, параметр2):

# Код функции, использующий параметры

print(f"Привет, {параметр1}! Тебе {параметр2} лет.")

# Вызов функции с аргументами

имя\_функции("Алиса", 10)

имя\_функции("Боб", 12)

* + - Количество и порядок аргументов при вызове должны совпадать с количеством и порядком параметров при определении функции.
  + **Демонстрация:** Создать функцию greet(name) и add\_numbers(a, b), показать их вызов с разными аргументами.

1. **Практические задания (10 минут)**
   * **Легкий уровень:**
     + Создайте список movies = ["Аватар", "Терминатор", "Матрица"]. С помощью цикла forвыведите каждый фильм из списка.
     + Создайте функцию show\_message(), которая выводит "Привет из функции!". Вызовите её 3 раза.
   * **Средний уровень:**
     + **"Список покупок 2.0":** Создайте список shopping\_list. С помощью цикла for попросите пользователя 5 раз ввести товар и добавить его в список. Затем выведите весь итоговый список.
     + **"Функция приветствия":** Создайте функцию greet\_user(name), которая принимает имя пользователя и выводит "Добро пожаловать, [Имя]!". Вызовите эту функцию для 3 разных имен.
   * **Сложный уровень:**
     + **"Калькулятор площади":** Создайте функцию calculate\_area(length, width), которая принимает длину и ширину прямоугольника и выводит его площадь. Затем вызовите эту функцию 3 раза с разными значениями.
     + **Игровой элемент: "Навыки героя":**
       - Задание: "Создайте список skills = ['Удар мечом', 'Лечение', 'Огненный шар'].
       - Создайте функцию use\_skill(skill\_name), которая принимает название навыка.
       - Внутри функции:
         * Если skill\_name находится в списке skills, вывести "Вы использовали [skill\_name]!".
         * Иначе, вывести "У вас нет такого навыка."
       - Вызовите эту функцию для нескольких разных навыков (как из списка, так и несуществующих).
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: циклы со списками, зачем нужны функции, def, аргументы/параметры.
   * Превью следующего урока: "На следующем занятии мы узнаем, как функции могут не только что-то делать, но и 'возвращать' результаты своей работы, а также познакомимся с такой важной структурой данных как словари!"

**Материалы/Ресурсы:**

* Презентация с визуализацией перебора списков.
* Схемы работы функций с аргументами.

**Домашнее задание:**

* Написать программу, которая создает список из 5 случайных чисел (можно просто ввести их). Затем с помощью цикла for выводит только четные числа из этого списка.
* Создать функцию print\_stars(count), которая принимает число count и выводит строку из countзвездочек. Вызовите эту функцию несколько раз с разными значениями.

## Урок №11: Возвращаемые значения функций. Словари (Dictionary).

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять концепцию возвращаемых значений (return) в функциях.
* Научиться использовать return для получения результатов работы функции.
* Познакомиться со словарями (dictionary) как неупорядоченной коллекцией пар ключ-значение.
* Научиться создавать словари, получать доступ к элементам, добавлять/изменять/удалять пары ключ-значение.

**Основные понятия:**

* return (вернуть)
* Возвращаемое значение функции
* Словарь (dictionary)
* Пара ключ-значение (key-value pair)
* Ключ (key)
* Значение (value)
* Неупорядоченная коллекция
* Методы словарей: get(), keys(), values(), items()

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Как использовать цикл for со списками? Для чего нужны функции? Как передать аргументы в функцию?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Возвращаемые значения функций (return) (25 минут)**
   * **Теоретический материал:**
     + До сих пор наши функции просто что-то печатали на экран или выполняли действия. Но часто нам нужно, чтобы функция "вычислила" что-то и "вернула" этот результат, чтобы мы могли использовать его дальше в программе.
     + **return:** Ключевое слово return используется внутри функции для возврата значения. Как только Python встречает return, функция завершает свою работу, и возвращаемое значение передается туда, откуда функция была вызвана.
     + **Синтаксис:**

Python

def имя\_функции(параметры):

# Код функции

результат = ... # Вычисления

return результат # Возвращаем результат

* + - **Использование возвращаемого значения:** Мы можем сохранить возвращаемое значение в переменную или использовать его напрямую.
      * Пример:

Python

def add(a, b):

sum\_result = a + b

return sum\_result # Функция возвращает сумму

result = add(5, 3) # result получит значение 8

print(result \* 2) # Выведет 16

print(add(10, 20)) # Можно использовать напрямую в print

* + - Если функция не имеет return (или имеет return без значения), она неявно возвращает None(специальное значение, означающее "ничего").
  + **Демонстрация:** Функция calculate\_area из прошлого урока, но теперь она должна return площадь, а не печатать ее. Показать, как можно использовать возвращенное значение.

1. **Словари (Dictionary). Основы работы (30 минут)**
   * **Теоретический материал:**
     + Списки хранят данные по числовым индексам (0, 1, 2...). Но что, если мы хотим хранить информацию в виде "имя-значение", например, "имя": "Алиса", "возраст": 10?
     + **Словарь (Dictionary):** Это неупорядоченная коллекция данных, которая хранит элементы в виде **пар "ключ-значение"**. Каждый ключ должен быть уникальным, а значение может быть любым типом данных.
     + **Аналогия:** "Словарь: у каждого слова (ключа) есть свое определение (значение)." "Записная книжка: имя человека (ключ) и его телефон (значение)."
     + **Создание словаря:** Словари обозначаются фигурными скобками {}, а пары ключ-значение разделяются двоеточием :, пары друг от друга – запятыми ,.
       - Пример: person = {"name": "Алиса", "age": 10, "city": "Москва"}
       - Ключи обычно строки, но могут быть и числами.
     + **Доступ к значениям:** Используется ключ в квадратных скобках имя\_словаря[ключ].
       - Пример: print(person["name"]) # Выведет: Алиса
       - Если ключа нет, будет ошибка!
     + **Метод get():** Более безопасный способ доступа. словарь.get(ключ, значение\_по\_умолчанию). Если ключа нет, возвращает None или указанное значение по умолчанию.
       - Пример: print(person.get("age")) # Выведет: 10
       - print(person.get("email", "Нет данных")) # Выведет: Нет данных
     + **Добавление/Изменение:** Присваивание значения по ключу. Если ключ уже есть, значение обновится; если нет – добавится новая пара.
       - Пример: person["email"] = "alice@example.com" # Добавление
       - Пример: person["age"] = 11 # Изменение
     + **Удаление:**
       - del словарь[ключ]: Удаляет пару по ключу.
       - словарь.pop(ключ): Удаляет пару по ключу и возвращает её значение.
   * **Демонстрация:** Создание словаря, доступ, добавление, изменение, удаление. Показать разницу между прямым доступом и get().
2. **Практические задания (10 минут)**
   * **Легкий уровень:**
     + Создайте функцию multiply(a, b), которая возвращает произведение двух чисел. Вызовите её и выведите результат.
     + Создайте словарь book = {"title": "Мастер и Маргарита", "author": "Булгаков", "year": 1967}. Выведите название книги и её автора.
   * **Средний уровень:**
     + **"Калькулятор НДС":** Создайте функцию calculate\_vat(price, vat\_rate), которая принимает цену и ставку НДС (например, 0.2 для 20%), и возвращает сумму НДС.
       - vat = calculate\_vat(1000, 0.2)
     + **"Информация о пользователе":** Создайте словарь user\_profile = {"name": "Олег", "age": 25, "city": "Киев"}. Попросите пользователя ввести ключ (например, "age") и выведите соответствующее значение из словаря. Обработайте случай, если ключа нет (например, выведите "Таких данных нет").
   * **Сложный уровень:**
     + **"Статистика слов":** Запросите у пользователя короткое предложение. Создайте функцию count\_words(sentence), которая принимает предложение и возвращает словарь, где ключами являются слова из предложения, а значениями - количество их вхождений.
       - *Подсказка: используйте sentence.split() для разделения предложения на слова, и цикл for для подсчета.*
       - Пример: count\_words("я люблю питон я") -> {"я": 2, "люблю": 1, "питон": 1}
     + **Игровой элемент: "Профиль персонажа RPG":**
       - Задание: "Создайте словарь player = {'name': 'Герой', 'health': 100, 'mana': 50, 'items': ['меч', 'зелье']}.
       - Создайте функцию deal\_damage(player\_dict, damage\_amount), которая принимает словарь игрока и количество урона. Функция должна уменьшить health игрока на damage\_amount и вернуть новое значение health.
       - Вызовите функцию несколько раз, наблюдая за изменением здоровья. Если здоровье <= 0, выведите "Игра окончена!".
3. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: return для получения результатов, словари для хранения данных ключ-значение, их создание и базовые операции.
   * Превью следующего урока: "На последнем занятии нашего первого модуля мы подытожим все полученные знания, создадим небольшую интерактивную программу, которая объединит в себе циклы, условия, списки и функции. Также у вас будет возможность задать все накопившиеся вопросы!"

**Материалы/Ресурсы:**

* Презентация с демонстрацией return и блок-схем.
* Визуализация словарей (ключ-значение).
* Примеры кода для работы со словарями.

**Домашнее задание:**

* Написать функцию is\_even(number), которая принимает число и возвращает True, если оно четное, и False, если нечетное.
* Создать словарь contacts = {"Иван": "123-456", "Мария": "789-012"}.
  + Добавить новый контакт.
  + Изменить номер телефона для существующего контакта.
  + Попросить пользователя ввести имя и вывести его номер (используйте get()).

## Урок №12: Итоговое занятие. Повторение и мини-проект.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Повторить и систематизировать основные понятия и конструкции, изученные в первом модуле: переменные, типы данных, операции, условия, циклы, строки, списки, функции, словари.
* Применить полученные знания для создания небольшого интерактивного мини-проекта.
* Закрепить навыки самостоятельного решения задач и отладки кода.
* Ответить на вопросы учащихся и подвести итоги первого модуля.

**Основные понятия:**

* Все изученные понятия модуля 1.

**План занятия:**

1. **Разминка и повторение (20 минут)**
   * **Теоретический материал (в формате вопросов-ответов):**
     + Что такое переменная? Какие типы данных вы знаете?
     + Как получить данные от пользователя? Зачем нужно преобразование типов?
     + Для чего нужны if, elif, else? Какие логические операторы вы знаете?
     + Когда использовать while, а когда for?
     + Чем списки отличаются от строк? Чем отличаются от словарей?
     + Зачем нужны функции? Что такое return?
   * Краткий разбор основных ошибок, которые часто встречаются у новичков (например, неверные отступы, забытые кавычки/скобки, преобразование типов).
2. **Работа над мини-проектом (45 минут)**
   * **Теоретический материал:**
     + Сегодня мы объединим все наши знания для создания небольшой, но полноценной программы.
     + **Этапы создания проекта:**
       1. **Понимание задачи:** Что должна делать программа?
       2. **Проектирование:** Какие переменные нужны? Какие функции? Какие условия и циклы?
       3. **Написание кода:** Постепенно, шаг за шагом.
       4. **Тестирование:** Проверка работы программы в разных ситуациях.
       5. **Отладка:** Исправление ошибок.
   * **Предлагаемый мини-проект: "Викторина по программированию".**
     + **Задача:** Создать простую викторину, которая задает несколько вопросов по пройденному материалу, проверяет ответы пользователя и считает очки.
     + **Требования к проекту:**
       1. Должны использоваться: input(), print().
       2. Условные операторы (if/else или if/elif/else).
       3. Цикл (for или while) для прохождения по вопросам.
       4. Списки (для хранения вопросов и ответов).
       5. Переменные для счета очков.
       6. Функции (минимум 1-2, например, ask\_question() или check\_answer()).
   * **Структура проекта (руководство для учащихся):**
     + Создайте список словарей, где каждый словарь – это вопрос:

Python

questions = [

{"question": "Какая функция используется для вывода текста?", "answer": "print"},

{"question": "Какой оператор используется для присваивания?", "answer": "="},

# Добавьте еще 2-3 вопроса

]

* + - Инициализируйте счетчик очков: score = 0.
    - Используйте цикл for для перебора вопросов: for q\_data in questions:.
    - Внутри цикла:
      1. Задайте вопрос (input()).
      2. Преобразуйте ответ пользователя к нижнему регистру для сравнения (например, user\_answer.lower()).
      3. Сравните ответ пользователя с правильным ответом (if/else).
      4. Если ответ правильный, увеличьте score и выведите "Правильно!". Иначе - "Неправильно. Правильный ответ: [правильный\_ответ]".
    - После цикла выведите итоговый счет: print(f"Ваш итоговый счет: {score} из {len(questions)}").
  + **Помощь преподавателя:** Пошаговая помощь, демонстрация, ответы на вопросы, помощь в отладке.

1. **Защита мини-проектов (опционально, 10 минут)**
   * Несколько учащихся демонстрируют свои викторины.
2. **Подведение итогов модуля и вопросы (15 минут)**
   * Обсуждение трудностей, с которыми столкнулись учащиеся.
   * Краткий обзор всего пройденного материала. "Вы изучили основы программирования на Python и теперь можете создавать интерактивные программы!"
   * Мотивация к дальнейшему обучению: что можно делать с этими знаниями, какие направления развивать дальше.
   * Ответы на оставшиеся вопросы.
   * Объявление о завершении модуля 1 и возможности перехода ко второму модулю.

**Материалы/Ресурсы:**

* Скелет кода для викторины или примеры кода для функций/списков, которые можно использовать.
* Дополнительные вопросы для викторины.

**Домашнее задание:**

* Доработать свою викторину: добавить больше вопросов, сделать более интересные сообщения, добавить приветствие и прощание.
* Подумать над следующими шагами: что бы хотелось изучить дальше в Python?

# М2.

**Урок №13: Кортежи (Tuple) и множества (Set).**

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомить с новыми структурами данных: кортежами (tuple) и множествами (set).
* Понять ключевые отличия кортежей от списков (неизменяемость).
* Изучить основные операции с кортежами и их применение.
* Понять концепцию множеств как неупорядоченных коллекций уникальных элементов.
* Освоить базовые операции с множествами (добавление, удаление, проверка наличия, объединение, пересечение).

**Основные понятия:**

* Кортеж (tuple)
* Неизменяемый объект (immutable)
* Множество (set)
* Уникальные элементы
* Методы множеств: add(), remove(), union(), intersection(), difference()

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Для чего нужен return в функциях? Что такое словарь? Как получить значение из словаря по ключу? Чем списки отличаются от словарей?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Кортежи (Tuple) (30 минут)**
   * **Теоретический материал:**
     + Мы уже знаем списки – изменяемые коллекции. Но иногда нам нужна коллекция, которая не будет меняться после создания.
     + **Кортеж (Tuple):** Это упорядоченная, **неизменяемая** коллекция элементов. Похож на список, но после создания его нельзя изменить (добавить, удалить или изменить элементы).
     + **Аналогия:** "Список дел, который уже нельзя переписать (только создать новый)." "Адрес, который всегда остается одним и тем же."
     + **Создание кортежа:** Кортежи обозначаются круглыми скобками (), элементы разделяются запятыми.
       - Пример: my\_tuple = (1, 2, 3, 4, 5)
       - Пример: coordinates = (10, 20)
       - Для кортежа из одного элемента обязательно нужна запятая: (1,)
     + **Доступ к элементам:** Как и в списках, по индексу: кортеж[индекс]. Также поддерживаются отрицательные индексы.
       - Пример: print(coordinates[0]) # Выведет: 10
     + **Почему кортежи?**
       - **Безопасность данных:** Гарантируют, что данные не изменятся случайно.
       - **Производительность:** Могут быть немного быстрее списков, так как они фиксированы.
       - **Использование в словарях:** Кортежи могут быть ключами в словарях (в отличие от списков).
       - **Возврат нескольких значений из функции:** Функции могут возвращать несколько значений в виде кортежа.
   * **Демонстрация:** Создание кортежа. Попытка изменить элемент кортежа (что вызовет ошибку TypeError). Доступ к элементам, использование len().
3. **Множества (Set) (30 минут)**
   * **Теоретический материал:**
     + Иногда нам нужна коллекция, где важен только сам факт наличия элемента, а порядок и повторения не имеют значения.
     + **Множество (Set):** Это неупорядоченная коллекция **уникальных** элементов. Это значит, что каждый элемент может встречаться в множестве только один раз. Дубликаты автоматически удаляются при создании или добавлении.
     + **Аналогия:** "Список участников, где каждый человек может быть только один раз." "Корзина с разными видами фруктов, но если положить два яблока, все равно будет считаться, что есть одно 'яблоко'."
     + **Создание множества:** Множества обозначаются фигурными скобками {} (как словари, но без пар ключ-значение). Из списка или кортежа можно создать множество с помощью set().
       - Пример: my\_set = {1, 2, 3, 4, 5}
       - Пример с дубликатами: unique\_numbers = {1, 2, 2, 3} -> unique\_numbers будет {1, 2, 3}
       - Создание из списка: my\_list = [1, 2, 2, 3]; unique\_list = set(my\_list)
       - **Важно:** Пустое множество создается как set(), а не {} ({} создает пустой словарь).
     + **Основные операции и методы множеств:**
       - add(элемент): Добавляет элемент в множество. Если элемент уже есть, ничего не происходит.
       - remove(элемент): Удаляет элемент. Если элемента нет, будет ошибка KeyError.
       - discard(элемент): Удаляет элемент. Если элемента нет, ошибки не будет.
       - in (проверка наличия): элемент in множество возвращает True или False.
       - **Операции с множествами (как в математике):**
         * union() (|): Объединение (все уникальные элементы из обоих множеств).
         * intersection() (&): Пересечение (только общие элементы).
         * difference() (-): Разность (элементы первого множества, которых нет во втором).
   * **Демонстрация:** Создание множества с дубликатами. Добавление, удаление элементов. Проверка наличия. Примеры операций объединения, пересечения, разности.
4. **Практические задания (10 минут)**
   * **Легкий уровень:**
     + Создайте кортеж colors = ("red", "green", "blue"). Выведите первый элемент. Попробуйте изменить второй элемент (убедитесь в ошибке).
     + Создайте список numbers = [1, 2, 2, 3, 4, 4, 5]. Преобразуйте его в множество и выведите, чтобы увидеть уникальные элементы.
   * **Средний уровень:**
     + **"Координаты точки":** Напишите функцию create\_point(x, y), которая возвращает кортеж с координатами (x, y). Вызовите её, сохраните результат в переменную и выведите координаты.
     + **"Уникальные слова":** Запросите у пользователя предложение. Преобразуйте его в множество слов (используя split()) и выведите количество уникальных слов.
       - *Подсказка: после split(), преобразуйте список слов в set().*
   * **Сложный уровень:**
     + **"Пользователи онлайн":** У вас есть два списка пользователей, которые были онлайн сегодня: online\_morning = {"user1", "user3", "user5"} и online\_evening = {"user2", "user3", "user4"}.
       - Найдите всех уникальных пользователей, которые были онлайн хотя бы один раз.
       - Найдите пользователей, которые были онлайн и утром, и вечером.
       - Найдите пользователей, которые были онлайн утром, но не вечером.
     + **Игровой элемент: "Проверка инвентаря героя":**
       - Задание: "Создайте кортеж rare\_items = ('Магический меч', 'Древний свиток') и список player\_inventory = ['Магический меч', 'Зелье', 'Щит'].
       - Создайте функцию has\_rare\_item(inventory, rare\_item\_tuple), которая проверяет, есть ли у игрока хоть один предмет из rare\_item\_tuple. Функция должна вернуть Trueили False.
       - Проверьте, есть ли у игрока редкие предметы."
       - *Подсказка: можете преобразовать player\_inventory в множество для быстрой проверки наличия.*
5. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: кортежи (неизменяемые, ()), множества (уникальные, {}), их основные операции и когда их использовать.
   * Превью следующего урока: "На следующем занятии мы погрузимся в очень важную тему – работу с файлами! Мы научимся читать информацию из файлов и записывать туда данные, что открывает огромные возможности для хранения и обмена информацией."

**Материалы/Ресурсы:**

* Презентация с визуализацией кортежей и множеств.
* Сравнение списков, кортежей и множеств.

**Домашнее задание:**

* Создать кортеж с данными о книге: (Название, Автор, Год). Вывести данные, используя распаковку кортежа (например, title, author, year = book\_data).
* Создать два множества с любимыми фруктами двух друзей. Найти фрукты, которые нравятся обоим, и фрукты, которые нравятся только одному из них.

**Урок №14: Работа с файлами (чтение и запись).**

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять необходимость работы с файлами для долговременного хранения данных.
* Изучить основные режимы открытия файлов ('r', 'w', 'a').
* Научиться читать данные из текстовых файлов (read(), readline(), readlines()).
* Научиться записывать данные в текстовые файлы (write(), writelines()).
* Освоить использование конструкции with open() для безопасной работы с файлами.

**Основные понятия:**

* Файл (file)
* Режим открытия файла (mode)
* Чтение ('r')
* Запись ('w')
* Дозапись ('a')
* open()
* read()
* readline()
* readlines()
* write()
* writelines()
* close()
* Конструкция with open()

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Чем кортежи отличаются от списков? Зачем нужны множества? Какие операции с множествами вы знаете?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Зачем работать с файлами? (15 минут)**
   * **Теоретический материал:**
     + Все данные, с которыми мы работали до сих пор, хранятся в оперативной памяти и исчезают, когда программа завершается.
     + **Файлы:** Позволяют хранить данные долговременно на диске (жестком диске, SSD, флешке). Это необходимо для:
       - Сохранения настроек программы.
       - Хранения пользовательских данных (списки, словари, тексты).
       - Обработки больших объемов информации, которые не помещаются в память.
       - Обмена данными между программами или пользователями.
     + **Путь к файлу:** Для доступа к файлу нужно указать его путь (полный или относительный). Для простоты, на первых порах, будем работать с файлами в той же папке, где находится Python-скрипт.
3. **Открытие и закрытие файлов. Режимы работы (30 минут)**
   * **Теоретический материал:**
     + **Функция open():** Используется для открытия файла. Возвращает объект файла, с которым мы будем работать.
       - Синтаксис: файл\_объект = open('имя\_файла.txt', 'режим')
       - **Режимы открытия:**
         * 'r' (read - чтение): Открывает файл для чтения. Если файла нет, будет ошибка.
         * 'w' (write - запись): Открывает файл для записи. Если файла нет, создаст его. **ВНИМАНИЕ: Если файл существует, его содержимое будет ПЕРЕЗАПИСАНО!**
         * 'a' (append - дозапись): Открывает файл для дозаписи. Если файла нет, создаст его. Данные добавляются в конец файла, существующее содержимое сохраняется.
       - Другие режимы (кратко упомянуть): 'x' (создание, если не существует), 'b' (бинарный режим).
     + **close():** После работы с файлом его ОБЯЗАТЕЛЬНО нужно закрыть с помощью метода файл\_объект.close(). Это освобождает ресурсы и гарантирует, что все изменения записаны.
     + **Конструкция with open() (рекомендуется!):** Это самый безопасный и удобный способ работы с файлами. with open() автоматически закрывает файл, даже если произошла ошибка.
       - Синтаксис:

Python

with open('файл.txt', 'режим') as file:

# Весь код для работы с файлом находится здесь (с отступом)

# Файл автоматически закроется после выхода из блока with

* + **Демонстрация:** Открытие файла в разных режимах. Показать ошибку, если файла нет в режиме 'r'. Показать перезапись в режиме 'w'. Демонстрация with open().

1. **Чтение данных из файла (15 минут)**
   * **Теоретический материал:**
     + После открытия файла в режиме 'r', можно использовать методы для чтения:
       - file.read(): Читает **все содержимое** файла в одну строку.
         * Пример: content = file.read()
       - file.readline(): Читает **одну строку** из файла. При каждом вызове читает следующую строку.
         * Пример: line1 = file.readline(); line2 = file.readline()
       - file.readlines(): Читает **все строки** файла и возвращает их в виде **списка строк**, где каждая строка списка включает символ переноса строки \n.
         * Пример: lines = file.readlines()
       - **Перебор файла в цикле for:** Самый простой способ читать файл построчно.
         * Пример:

Python

with open('my\_file.txt', 'r') as file:

for line in file:

print(line.strip()) # strip() убирает лишние пробелы и \n

* + **Демонстрация:** Создать простой текстовый файл вручную (например, data.txt). Прочитать его разными методами.

1. **Запись данных в файл (10 минут)**
   * **Теоретический материал:**
     + После открытия файла в режиме 'w' или 'a', можно использовать методы для записи:
       - file.write(строка): Записывает указанную строку в файл. **Не добавляет символ новой строки \n автоматически!** Его нужно добавлять вручную.
         * Пример: file.write("Привет, мир!\n")
       - file.writelines(список\_строк): Записывает список строк в файл. Каждая строка из списка записывается, но **без автоматического добавления \n**. Нужно, чтобы каждая строка в списке уже содержала \n в конце, если нужно писать на новой строке.
         * Пример: lines\_to\_write = ["Первая строка\n", "Вторая строка\n"]
         * file.writelines(lines\_to\_write)
   * **Демонстрация:** Записать текст в новый файл. Дописать текст в существующий файл.
2. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Создайте файл hello.txt и запишите в него строку "Привет, Python!".
     + Откройте этот файл и прочитайте его содержимое, выведя на экран.
   * **Средний уровень:**
     + **"Записная книжка":** Напишите программу, которая запрашивает у пользователя имя и номер телефона, а затем дописывает их в файл contacts.txt (каждый контакт на новой строке).
     + **"Чтение списка задач":** Пусть у вас есть файл tasks.txt, где каждая задача записана на новой строке. Прочитайте все задачи из файла и выведите их в виде нумерованного списка.
       - *Предварительно создайте файл tasks.txt с несколькими задачами.*
   * **Сложный уровень:**
     + **"Логирование действий":** Создайте функцию log\_action(action\_text), которая принимает текст действия (например, "Пользователь вошел") и записывает его в файл log.txt вместе с текущей датой и временем (пока дату/время можно просто ввести вручную или использовать f-строки для красивого вывода). Каждая запись должна быть на новой строке.
     + **Игровой элемент: "Дневник приключений":**
       - Задание: "Создайте функцию write\_to\_diary(entry), которая принимает запись для дневника и дописывает ее в файл diary.txt.
       - Пусть пользователь сможет вводить новые записи, пока не введет 'конец'.
       - После завершения, прочитайте и выведите все записи из дневника."
3. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: зачем файлы, режимы 'r', 'w', 'a', with open(), методы чтения/записи.
   * Превью следующего урока: "На следующем занятии мы познакомимся с обработкой ошибок – что делать, если что-то пошло не так в нашей программе, чтобы она не "падала", а могла продолжать работать или gracefully завершиться!"

**Материалы/Ресурсы:**

* Презентация с режимами открытия файлов.
* Схемы работы read(), readline(), readlines().
* Примеры кода для чтения и записи.

**Домашнее задание:**

* Написать программу, которая запрашивает у пользователя несколько строк текста (пока пользователь не введет пустую строку), и записывает каждую введенную строку в файл user\_text.txt.
* Написать программу, которая читает числа из файла numbers.txt (по одному числу на строке), складывает их и выводит сумму.
* Создать программу "Счетчик слов в файле": запросить имя файла у пользователя, прочитать его содержимое и вывести общее количество слов в файле (используйте split() на всей строке, а затем len()для списка слов).

## Урок №15: Обработка ошибок (try-except).

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять концепцию ошибок и исключений (exceptions) в программировании.
* Изучить, как Python сообщает об ошибках (трассировка стека).
* Научиться использовать конструкцию try-except для перехвата и обработки исключений.
* Применять try-except для создания более надежных и "дружелюбных" к пользователю программ.

**Основные понятия:**

* Ошибка (error)
* Исключение (exception)
* Трассировка стека (traceback)
* try
* except
* Типы исключений (например, ValueError, FileNotFoundError, ZeroDivisionError)
* Обработка ошибок

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Зачем нам нужны файлы? Какие режимы открытия файлов вы знаете? Для чего нужна конструкция with open()?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Что такое ошибки и исключения? (20 минут)**
   * **Теоретический материал:**
     + До сих пор, когда что-то шло не так в нашей программе (например, деление на ноль, или попытка преобразовать "привет" в число), программа просто "падала" и выводила длинное сообщение об ошибке (трассировку стека).
     + **Ошибка (Error):** Это то, что приводит к некорректной работе программы. В Python это часто называется **исключением (Exception)**.
     + **Исключение (Exception):** Это событие, которое нарушает нормальный ход выполнения программы. Python "выбрасывает" (raises) исключение, когда сталкивается с ситуацией, которую не может обработать обычным способом.
     + **Трассировка стека (Traceback):** Это подробное сообщение об ошибке, которое Python выводит, когда происходит исключение. Оно показывает, в каком файле, на какой строке и какая именно ошибка произошла. Важно уметь читать трассировку, чтобы понять, что пошло не так.
     + **Примеры распространенных исключений:**
       - ValueError: Происходит при некорректном значении (например, int("abc")).
       - TypeError: Происходит при операции с неподходящими типами (например, 5 + "abc").
       - ZeroDivisionError: Происходит при делении на ноль.
       - FileNotFoundError: Происходит при попытке открыть несуществующий файл.
       - IndexError: Происходит при попытке получить элемент списка/кортежа по несуществующему индексу.
       - KeyError: Происходит при попытке получить значение из словаря по несуществующему ключу.
     + **Цель обработки ошибок:** Сделать программу более "выносливой". Вместо того, чтобы просто "падать", программа может "поймать" ошибку, "обработать" ее (например, вывести понятное сообщение пользователю) и продолжить работу или завершиться контролируемо.
   * **Демонстрация:** Продемонстрировать разные типы ошибок, чтобы учащиеся видели трассировку стека. Например: деление на ноль, преобразование нечисловой строки в int.
3. **Конструкция try-except (40 минут)**
   * **Теоретический материал:**
     + **try:** Блок кода, который мы хотим "попробовать" выполнить. Мы ожидаем, что в этом блоке может возникнуть исключение.
     + **except:** Блок кода, который выполняется, если в блоке try произошло исключение. Здесь мы "обрабатываем" ошибку.
     + **Синтаксис:**

Python

try:

# Код, который может вызвать ошибку

result = 10 / 0

print(result)

except ZeroDivisionError: # Ловим конкретный тип ошибки

print("Ошибка: Деление на ноль невозможно!")

except ValueError: # Можно ловить несколько типов ошибок

print("Ошибка значения!")

except Exception as e: # Ловим любую ошибку (хорошо для отладки, но конкретнее лучше)

print(f"Произошла неизвестная ошибка: {e}")

else:

# Код, который выполняется, если в блоке try ошибок НЕ БЫЛО

print("Операция выполнена успешно.")

finally:

# Код, который выполняется ВСЕГДА, независимо от того,

# была ошибка или нет. Полезно для закрытия файлов и т.п.

print("Блок try-except завершен.")

* + - **Как это работает:**
      1. Python пытается выполнить код в блоке try.
      2. Если возникает исключение, Python ищет подходящий блок except.
      3. Если находит, выполняет код в этом except блоке.
      4. Если исключение не соответствует ни одному except (или нет except для этого типа), оно выбрасывается дальше (программа "падает").
      5. Блок else выполняется, если try прошел без ошибок.
      6. Блок finally выполняется всегда, в любом случае.
  + **Демонстрация:** Реализовать пример с делением на ноль, затем с ValueError (ввод нечисловой строки). Показать, как программа не "падает", а выводит сообщение об ошибке. Продемонстрировать else и finally.

1. **Практические задания (15 минут)**
   * **Легкий уровень:**
     + Напишите программу, которая запрашивает у пользователя число. Попробуйте преобразовать его в int(). Если пользователь ввел не число, перехватите ValueError и выведите "Ошибка: Введите, пожалуйста, число!".
   * **Средний уровень:**
     + **"Безопасное деление":** Запросите у пользователя два числа (делимое и делитель). Выполните деление. Используйте try-except для перехвата ZeroDivisionError (если делитель равен 0) и ValueError (если введены не числа).
   * **Сложный уровень:**
     + **"Безопасное чтение файла":**
       1. Запросите у пользователя имя файла.
       2. Попробуйте открыть и прочитать этот файл.
       3. Используйте try-except для перехвата FileNotFoundError, если файл не существует. Выведите понятное сообщение: "Ошибка: Файл [имя\_файла] не найден. Проверьте имя."
       4. Также используйте except Exception as e для перехвата любых других непредвиденных ошибок при работе с файлом.
     + **Игровой элемент: "Обработчик ввода в игре":**
       - Задание: "В игре пользователь должен ввести количество золота для покупки предмета. Это должно быть целое число.
       - Создайте цикл while True, который будет запрашивать 'Сколько золота вы хотите потратить?'.
       - Внутри try-except блок:
         * Попробуйте преобразовать ввод в int.
         * Если успешно, вывести 'Вы потратили X золота.' и выйти из цикла (break).
         * Если ValueError, вывести 'Некорректный ввод. Введите число.' и продолжить цикл."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: зачем обрабатывать ошибки, try-except, основные типы ошибок.
   * Превью следующего урока: "На следующем занятии мы начнем изучать основы объектно-ориентированного программирования (ООП) – мощного подхода, который позволяет создавать сложные программы, организованные как взаимодействующие объекты. Это большой шаг в освоении Python!"

**Материалы/Ресурсы:**

* Презентация с примерами трассировки стека.
* Блок-схемы для try-except-else-finally.
* Примеры кода для обработки различных ошибок.

**Домашнее задание:**

* Написать программу, которая запрашивает у пользователя индекс элемента списка my\_list = [10, 20, 30]. Попробуйте вывести элемент по этому индексу. Перехватите IndexError, если индекс вне диапазона, и выведите "Ошибка: такого элемента нет".
* Создать программу "Безопасный доступ к словарю":
  + Создайте словарь student = {"name": "Петр", "age": 15}.
  + Попросите пользователя ввести ключ, по которому он хочет получить информацию.
  + Используйте try-except KeyError для безопасного получения значения. Если ключа нет, выведите "Ошибка: такой информации нет".

## Урок №16: Основы объектно-ориентированного программирования (ООП). Классы и объекты.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомить с базовыми принципами объектно-ориентированного программирования (ООП).
* Объяснить понятия "класс" и "объект".
* Научиться определять классы и создавать их экземпляры (объекты).
* Изучить, что такое "атрибуты" (свойства) и "методы" (поведения) объектов.
* Освоить использование self для доступа к атрибутам и методам внутри класса.

**Основные понятия:**

* Объектно-ориентированное программирование (ООП)
* Класс (class)
* Объект (object) / Экземпляр класса (instance)
* Атрибут (attribute) / Свойство (property)
* Метод (method) / Поведение
* self
* \_\_init\_\_ (конструктор)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое исключение? Для чего нужна конструкция try-except? Приведите примеры ошибок, которые она может обрабатывать.
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Введение в ООП: Зачем это нужно? (20 минут)**
   * **Теоретический материал:**
     + До сих пор мы писали программы, которые представляли собой последовательность инструкций (процедурное программирование). Это хорошо для простых задач.
     + Но для больших, сложных программ (игры, большие веб-приложения) такой подход становится неудобным. Код становится запутанным, тяжело вносить изменения.
     + **Объектно-ориентированное программирование (ООП):** Это парадигма программирования, которая позволяет создавать программы, моделируя реальный мир. Вместо простого набора инструкций, мы создаем "объекты", которые имеют свои "свойства" и могут выполнять "действия".
     + **Аналогия:** "Мы живем в мире объектов: человек, собака, машина, дом. У каждого из них есть свои характеристики (цвет волос, количество колес, адрес) и свои действия (говорить, ехать, открывать дверь). ООП позволяет нам создавать такие 'цифровые' объекты в программе."
     + **Преимущества ООП:**
       - **Модульность:** Разделение программы на независимые, многократно используемые части.
       - **Гибкость:** Легче добавлять новые функции или изменять существующие.
       - **Понятность:** Код лучше отражает реальный мир, его легче читать и понимать.
3. **Классы и Объекты (30 минут)**
   * **Теоретический материал:**
     + **Класс (Class):** Это как "чертеж" или "шаблон" для создания объектов. Он определяет, какие свойства (атрибуты) будут у объектов этого типа и какие действия (методы) они смогут выполнять. Класс не является самим объектом, это только его описание.
       - Пример: Класс Человек
     + **Объект (Object) / Экземпляр класса (Instance):** Это конкретная "копия" класса. Каждый объект создается на основе класса и имеет свои собственные значения для свойств.
       - Пример: Объекты Алиса (экземпляр класса Человек), Боб (экземпляр класса Человек).
     + **Определение класса:** Используется ключевое слово class.
       - Синтаксис:

Python

class ИмяКласса:

# Тело класса: атрибуты и методы

pass # 'pass' используется, когда тело класса пока пустое

* + - * Имена классов обычно пишутся с заглавной буквы (CamelCase), например MyClass.
    - **Создание объекта (экземпляра класса):**
      * Синтаксис: имя\_объекта = ИмяКласса()
      * Пример: my\_person = Person()
    - **Атрибуты (Attributes) / Свойства:** Это переменные, которые хранят данные, связанные с конкретным объектом. Они определяют "что" объект собой представляет.
      * Доступ к атрибутам: объект.имя\_атрибута
      * Пример: my\_person.name = "Алиса"; print(my\_person.name)
    - **Методы (Methods) / Поведения:** Это функции, которые "принадлежат" классу и определяют "что" объект может делать. Методы всегда имеют первый параметр self.
      * Доступ к методам: объект.имя\_метода()
      * Пример: my\_person.greet()
    - **self:** Это обязательный первый параметр в любом методе класса. Он автоматически ссылается на текущий объект, для которого вызывается метод. Через self методы могут получать доступ к атрибутам и другим методам этого же объекта.
    - **Метод \_\_init\_\_ (Конструктор):** Это специальный метод, который вызывается автоматически каждый раз, когда создается новый объект класса. Он используется для инициализации (установки начальных значений) атрибутов объекта.
      * Пример:

Python

class Person:

def \_\_init\_\_(self, name, age): # self, а потом параметры для создания

self.name = name # Создаем атрибут 'name' для объекта

self.age = age # Создаем атрибут 'age' для объекта

def greet(self): # Метод объекта

print(f"Привет, меня зовут {self.name} и мне {self.age} лет.")

# Создаем объекты, передавая аргументы в \_\_init\_\_

person1 = Person("Алиса", 10)

person2 = Person("Боб", 12)

person1.greet() # Выведет: Привет, меня зовут Алиса и мне 10 лет.

* + **Демонстрация:** Создать простой класс Dog с атрибутами name, breed и методом bark(). Создать несколько объектов Dog с разными именами и породами, вызвать их методы. Показать работу \_\_init\_\_.

1. **Практические задания (20 минут)**
   * **Легкий уровень:**
     + Создайте класс Car (Автомобиль). Добавьте ему атрибуты brand (марка) и model (модель) в методе \_\_init\_\_. Создайте 2-3 объекта Car с разными марками/моделями и выведите их атрибуты.
   * **Средний уровень:**
     + **"Класс Книга":** Создайте класс Book.
       - В \_\_init\_\_ он должен принимать title (название) и author (автор).
       - Добавьте метод display\_info(self), который выводит "Книга: [название], Автор: [автор]".
       - Создайте несколько объектов Book и вызовите их метод display\_info().
     + **"Управление игроком":** Создайте класс Player.
       - В \_\_init\_\_ добавьте name и health (например, 100).
       - Добавьте метод take\_damage(self, damage\_amount), который уменьшает health игрока на damage\_amount и выводит текущее здоровье.
       - Создайте игрока, нанесите ему урон несколько раз.
   * **Сложный уровень:**
     + **"Виртуальный питомец":** Создайте класс Pet.
       - В \_\_init\_\_ добавьте name, species (вид, например "кошка", "собака"), happiness(счастье, изначально 100).
       - Добавьте методы:
         * feed(self): Увеличивает happiness на 10, выводит сообщение.
         * play(self): Увеличивает happiness на 15, выводит сообщение.
         * status(self): Выводит текущее happiness питомца.
       - Создайте объект питомца, покормите его, поиграйте с ним, проверяйте статус.
     + **Игровой элемент: "Класс Враг":**
       - Задание: "В вашей игре есть враги. Создайте класс Enemy.
       - В \_\_init\_\_ он должен принимать name, health и damage (урон).
       - Добавьте метод attack(self, target\_player\_health), который симулирует атаку: выводит "[Имя врага] атакует!", уменьшает target\_player\_health на damage врага и возвращает новое значение здоровья игрока.
       - Создайте объект врага, объект игрока (можно просто переменную player\_hp = 100), и пусть враг атакует игрока."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: ООП, класс как шаблон, объект как экземпляр, атрибуты (что?) и методы (что делает?), self, \_\_init\_\_.
   * Превью следующего урока: "На следующем занятии мы продолжим погружение в ООП, узнаем о наследовании – как классы могут 'унаследовать' свойства и поведение от других классов, что очень удобно для создания иерархий и повторного использования кода!"

**Материалы/Ресурсы:**

* Презентация с визуализацией классов и объектов.
* Схемы отношений класс-объект, атрибут-метод.
* Примеры кода для определения классов и создания объектов.

**Домашнее задание:**

* Создайте класс Student с атрибутами name и grade (оценка, например, 5). Добавьте метод study(self), который выводит "Студент [имя] изучает Python!". Создайте объект студента и вызовите его метод.
* Создайте класс Circle (Круг). В \_\_init\_\_ он должен принимать radius (радиус). Добавьте метод calculate\_area(self), который возвращает площадь круга (Pi \* r^2, используйте 3.14 для Pi). Создайте объект круга и выведите его площадь.

**Урок №17: Нас**ледова**ние в ООП.**

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять концепцию наследования как одного из ключевых принципов ООП.
* Изучить, как создавать дочерние классы (классы-наследники) от родительских классов.
* Освоить использование ключевого слова super() для вызова методов родительского класса.
* Понять преимущества наследования: повторное использование кода, создание иерархий классов.

**Основные понятия:**

* Наследование (Inheritance)
* Родительский класс (Parent Class) / Базовый класс (Base Class)
* Дочерний класс (Child Class) / Класс-наследник (Derived Class)
* super()
* Переопределение методов (Method Overriding)
* Иерархия классов

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое класс и объект? Что такое атрибуты и методы? Зачем нужен self и \_\_init\_\_?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Зачем нужно наследование? (20 минут)**
   * **Теоретический материал:**
     + Представьте, что у нас есть класс Animal (Животное) со свойствами name, age и методом make\_sound(). А потом нам нужны классы Dog (Собака) и Cat (Кошка). У них тоже будут name, age и make\_sound().
     + Без наследования нам пришлось бы повторять один и тот же код в каждом новом классе. Это неэффективно и чревато ошибками.
     + **Наследование (Inheritance):** Позволяет новому классу (дочернему) "унаследовать" все атрибуты и методы от существующего класса (родительского). Это как ребенок наследует черты от родителей.
     + **Аналогия:** "Родительский класс Автомобиль имеет общие свойства (колеса, двигатель) и действия (ехать). Дочерние классы Грузовик и ЛегковойАвтомобиль наследуют эти общие черты, но добавляют свои уникальные (грузоподъемность у грузовика, тип кузова у легкового)."
     + **Преимущества наследования:**
       - **Повторное использование кода (Code Reusability):** Не нужно писать одно и то же много раз.
       - **Создание иерархий:** Позволяет моделировать отношения "является" (is-a relationship), например, "Собака ЯВЛЯЕТСЯ Животным".
       - **Упрощение поддержки:** Изменения в родительском классе автоматически распространяются на все дочерние.
3. **Создание дочерних классов. super() (35 минут)**
   * **Теоретический материал:**
     + **Синтаксис наследования:** Дочерний класс указывается в скобках после имени родительского класса при его определении.

Python

class ParentClass:

def \_\_init\_\_(self, name):

self.name = name

def greet(self):

print(f"Я родительский класс, меня зовут {self.name}")

class ChildClass(ParentClass): # ChildClass наследует от ParentClass

def \_\_init\_\_(self, name, age):

# Вызываем конструктор родительского класса

super().\_\_init\_\_(name) # Передаем name в ParentClass.\_\_init\_\_

self.age = age # Добавляем новый атрибут для дочернего класса

def say\_age(self):

print(f"Мне {self.age} лет.")

# Создаем объект дочернего класса

child\_obj = ChildClass("Максим", 5)

child\_obj.greet() # Метод унаследован от ParentClass

child\_obj.say\_age() # Метод специфичен для ChildClass

* + - **super():** Это функция, которая позволяет дочернему классу вызывать методы родительского класса, в частности его конструктор \_\_init\_\_. Это очень важно, чтобы инициализировать все атрибуты, которые определены в родительском классе.
      * super().\_\_init\_\_(аргументы): Вызывает \_\_init\_\_ родителя.
    - **Переопределение методов (Method Overriding):** Дочерний класс может переопределить (изменить) метод, который он унаследовал от родительского класса, дав ему новую реализацию. Это позволяет дочерним классам вести себя по-своему, даже если у них есть одноименный метод от родителя.
      * Пример:

Python

class Animal:

def make\_sound(self):

print("Животное издает звук")

class Dog(Animal):

def make\_sound(self): # Переопределяем метод

print("Гав-гав!")

animal = Animal()

dog = Dog()

animal.make\_sound() # Выведет: Животное издает звук

dog.make\_sound() # Выведет: Гав-гав!

* + **Демонстрация:** Создать класс Vehicle (Транспортное средство) с атрибутами speed и методом accelerate(). Затем создать класс Car (Автомобиль), который наследует от Vehicle и добавляет атрибут num\_doors (количество дверей) и, возможно, переопределяет accelerate() или добавляет свой honk().

1. **Практические задания (15 минут)**
   * **Легкий уровень:**
     + Создайте класс Shape (Фигура) с методом display(), который выводит "Это фигура.".
     + Создайте класс Circle (Круг), который наследует от Shape. Создайте объект Circle и вызовите его метод display().
   * **Средний уровень:**
     + **"Классы Работник и Менеджер":**
       - Создайте класс Employee (Работник) с \_\_init\_\_, принимающим name и salary (зарплата). Добавьте метод get\_info(), который выводит информацию о работнике.
       - Создайте класс Manager (Менеджер), который наследует от Employee. В его \_\_init\_\_ он должен также принимать department (отдел). Убедитесь, что вы правильно используете super().\_\_init\_\_().
       - Создайте объект Manager и вызовите его метод get\_info().
   * **Сложный уровень:**
     + **"Герои и Маги":**
       - Создайте класс Hero с атрибутами name, health и методом attack(), который выводит "[Имя] атакует!".
       - Создайте класс Mage (Маг), который наследует от Hero.
       - У Mage должен быть дополнительный атрибут mana.
       - Переопределите метод attack() для Mage, чтобы он выводил "[Имя] кастует заклинание!" и уменьшал mana.
       - Создайте героя и мага, пусть они оба атакуют.
     + **Игровой элемент: "Виды транспортных средств":**
       - Задание: "Создайте класс Transport с атрибутом max\_speed и методом move(), который выводит 'Транспорт движется со скоростью [max\_speed] км/ч'.
       - Создайте класс Bicycle (Велосипед), наследующий от Transport. У него должен быть дополнительный атрибут num\_gears (количество передач).
       - Переопределите метод move() для Bicycle, чтобы он выводил 'Велосипед с [num\_gears] передачами движется со скоростью [max\_speed] км/ч'.
       - Создайте объекты Transport и Bicycle с разными параметрами и вызовите их методы move()."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: наследование, родительский/дочерний класс, super(), переопределение.
   * Превью следующего урока: "На следующем занятии мы продолжим разбираться с ООП и познакомимся с концепцией полиморфизма – как объекты разных классов могут реагировать на одни и те же команды по-разному!"

**Материалы/Ресурсы:**

* Презентация с диаграммами наследования (стрелки от дочернего к родительскому).
* Примеры кода для наследования и super().

**Домашнее задание:**

* Создать класс Animal с атрибутом name и методом speak(), который выводит "Животное издает звук".
* Создать класс Cat, который наследует от Animal. Переопределить метод speak() так, чтобы он выводил "Мяу!".
* Создать класс Dog, который наследует от Animal. Переопределить метод speak() так, чтобы он выводил "Гав!".
* Создать объекты Animal, Cat, Dog и вызвать их методы speak().
* Создать класс Person с name и age. Создать класс Student (наследует от Person) с дополнительным атрибутом student\_id. Инициализировать Student с помощью super().

Урок №18: Полиморфизм и Абстракция в ООП**.**

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомить с концепцией полиморфизма и его применением в Python.
* Понять, как объекты разных классов могут отвечать на одинаковые вызовы методов по-разному.
* Объяснить, что такое абстракция и инкапсуляция как принципы ООП (кратко, для общего понимания).
* Закрепить понимание, что ООП помогает создавать гибкий и расширяемый код.

**Основные понятия:**

* Полиморфизм (Polymorphism)
* Абстракция (Abstraction)
* Инкапсуляция (Encapsulation)
* isinstance()
* Принципы ООП (SOLID - краткое упоминание)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое наследование? Зачем нужен super()? Как дочерний класс может изменить поведение родительского?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Полиморфизм (40 минут)**
   * **Теоретический материал:**
     + Мы уже видели, что разные объекты (например, Cat и Dog), наследуя от Animal, могут иметь один и тот же метод speak(), но реализовывать его по-разному. Это и есть основа **полиморфизма**.
     + **Полиморфизм (Polymorphism):** "Много форм". В ООП это означает способность объектов разных классов отвечать на одно и то же сообщение (вызов метода) по-разному, в зависимости от их типа.
     + **Аналогия:** "Водитель знает, что у любой машины есть педаль газа. Нажимает на нее - машина едет. Неважно, какая марка машины, действие 'ехать' выполняется. Полиморфизм позволяет нам работать с разными объектами одинаковым способом."
     + **Примеры:**
       - Функции, работающие с разными типами данных (например, len() работает и со строками, и со списками, и с кортежами).
       - Один и тот же метод (make\_sound()) в разных классах (Dog, Cat), унаследованных от общего родителя (Animal).
       - **Пример кода:**

Python

class Animal:

def make\_sound(self):

pass # Базовая реализация или ничего

class Dog(Animal):

def make\_sound(self):

print("Гав-гав!")

class Cat(Animal):

def make\_sound(self):

print("Мяу!")

# Список животных

animals = [Dog(), Cat(), Dog()]

for animal in animals:

animal.make\_sound() # Вызываем один и тот же метод, но результат разный

* + - **isinstance(объект, класс):** Функция, которая проверяет, является ли объект экземпляром указанного класса (или его дочернего класса). Может быть полезна для проверки типов в полиморфных сценариях, хотя обычно стремятся к тому, чтобы код работал без явных проверок типа.
      * Пример: isinstance(dog\_obj, Animal) -> True
  + **Демонстрация:** Реализовать пример с Animal, Dog, Cat и списком животных. Показать, как один и тот же вызов animal.make\_sound() приводит к разным результатам.

1. **Абстракция и Инкапсуляция (краткое введение) (25 минут)**
   * **Теоретический материал:**
     + **Абстракция (Abstraction):**
       - Это процесс сокрытия сложной внутренней реализации объекта и представления только тех деталей, которые важны для пользователя.
       - **Аналогия:** "Когда вы нажимаете кнопку на пульте телевизора, вы видите только результат (канал переключился), но не знаете, как именно это происходит внутри телевизора (сложные электронные схемы). Пульт предоставляет вам абстракцию."
       - В Python мы достигаем абстракции, определяя методы, которые пользователи вызывают, не вдаваясь в их внутреннюю логику.
     + **Инкапсуляция (Encapsulation):**
       - Это связывание данных (атрибутов) и методов (функций), работающих с этими данными, в единый блок – класс.
       - Также это относится к сокрытию внутренней информации объекта от внешнего мира. Некоторые атрибуты/методы могут быть "приватными" (недоступными напрямую извне) и изменяться только через специальные "публичные" методы.
       - **Аналогия:** "В машине двигатель, коробка передач и другие сложные механизмы инкапсулированы внутри корпуса. Вы не можете просто так поменять шестеренку. Вы взаимодействуете с машиной через руль, педали, рычаг переключения передач."
       - В Python нет строгих "приватных" полей, но есть соглашения (например, атрибуты с одним подчеркиванием \_private\_attribute или двумя \_\_really\_private\_attribute). Цель – показать, что эти детали предназначены для внутреннего использования класса.
     + **Общее понимание:** Эти принципы помогают создавать хорошо структурированный, легко изменяемый и безопасный код, снижая сложность взаимодействия между разными частями программы.
   * **Демонстрация:** Кратко показать абстракцию на примере использования методов len() или append(): мы знаем, что они делают, но не знаем их внутренней реализации. Краткое объяснение инкапсуляции через атрибуты класса.
2. **Практические задания (10 минут)**
   * **Легкий уровень:**
     + Измените класс Shape и Circle из прошлого урока. Добавьте в Shape метод draw() (который пока ничего не делает). В Circle переопределите draw(), чтобы он выводил "Рисуем круг!". Создайте список из объектов Shape и Circle (один Shape, один Circle). В цикле вызовите draw() для каждого объекта.
   * **Средний уровень:**
     + **"Платежные системы":**
       - Создайте базовый класс PaymentMethod с методом process\_payment(amount), который просто выводит "Обработка платежа...".
       - Создайте дочерние классы CreditCard и PayPal, которые наследуют от PaymentMethod.
       - Переопределите process\_payment(amount) в каждом дочернем классе, чтобы он выводил специфическое сообщение, например, "Оплата кредитной картой на сумму [amount]" или "Оплата через PayPal на сумму [amount]".
       - Создайте список из различных платежных методов и в цикле обработайте платеж для каждого из них.
   * **Сложный уровень:**
     + **"Деревья и плоды":**
       - Создайте класс Tree (Дерево) с методом grow(), который выводит "Дерево растет.".
       - Создайте класс FruitTree (Фруктовое дерево), наследующий от Tree. Переопределите grow() так, чтобы оно выводило "Фруктовое дерево растет и дает плоды!".
       - Добавьте в FruitTree метод harvest\_fruit(), который выводит "Собраны фрукты.".
       - Создайте объекты Tree и FruitTree. Поместите их в список.
       - В цикле пройдитесь по списку. Вызовите grow() для каждого объекта.
       - Используйте isinstance() внутри цикла: если объект является FruitTree, вызовите также harvest\_fruit().
     + **Игровой элемент: "Звуки животных в зоопарке":**
       - Задание: "Используя классы Animal, Dog, Cat из домашнего задания или прошлого урока.
       - Создайте класс Zoo. В его \_\_init\_\_ он должен принимать список animals.
       - Добавьте в Zoo метод make\_all\_sounds(), который перебирает всех животных в списке animals зоопарка и вызывает их метод speak().
       - Создайте несколько объектов Dog и Cat, поместите их в список и создайте объект Zoo с этим списком. Вызовите zoo.make\_all\_sounds()."
3. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: полиморфизм (одинаковый метод, разное поведение), абстракция (скрываем детали), инкапсуляция (объединяем данные и методы).
   * Превью следующего урока: "На следующем занятии мы начнем изучать управление пакетами и модулями в Python. Это позволит нам использовать готовый код, написанный другими программистами, и эффективно организовывать наш собственный большой код!"

**Материалы/Ресурсы:**

* Презентация с примерами полиморфизма.
* Схемы, иллюстрирующие абстракцию и инкапсуляцию.

**Домашнее задание:**

* Создайте базовый класс Employee с методом work(), который выводит "Работник выполняет общую работу".
* Создайте два дочерних класса: Developer (Разработчик) и Designer (Дизайнер), наследующие от Employee.
* Переопределите метод work() в каждом из них, чтобы Developer выводил "Разработчик пишет код", а Designer выводил "Дизайнер создает макеты".
* Создайте список из нескольких Developer и Designer объектов. Пройдитесь по списку и вызовите work()для каждого.
* Создайте класс Bank\_Account с атрибутом balance и методами deposit(amount) (пополнить) и withdraw(amount) (снять). Реализуйте базовую проверку на недостаточность средств при снятии.

## Урок №19: Модули и пакеты.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять концепцию модулей как способа организации кода.
* Изучить, как импортировать модули (import).
* Научиться использовать функции и переменные из импортированных модулей.
* Понять, что такое пакеты и как они помогают в организации больших проектов.
* Освоить использование некоторых встроенных модулей Python (например, math, random).

**Основные понятия:**

* Модуль (module)
* Пакет (package)
* import
* from ... import ...
* Встроенные модули
* Стандартная библиотека Python
* Виртуальное окружение (кратко, для будущего)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое полиморфизм? Как он проявляется в ООП? Что такое абстракция и инкапсуляция (своими словами)?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Зачем нужны модули? (20 минут)**
   * **Теоретический материал:**
     + Пока наши программы были небольшими, весь код помещался в один файл. Но что, если программа становится большой? (например, 1000 строк кода, или 10 000?)
     + Становится очень тяжело ориентироваться. Также, если мы написали полезную функцию, которую хотим использовать в другой программе, нам придется ее копировать.
     + **Модуль (Module):** Это обычный файл Python (.py), который содержит функции, классы, переменные и другой код. Модули позволяют:
       - **Организовать код:** Разбить большую программу на более мелкие, логические части.
       - **Повторно использовать код:** Написать код один раз и использовать его во множестве разных программ, просто импортируя модуль.
       - **Избегать конфликтов имен:** Разные модули могут иметь функции с одинаковыми именами, но они не будут конфликтовать, так как они находятся в разных "пространствах имен" (namespaces).
     + **Аналогия:** "Книга по рецептам: каждый рецепт - это отдельный модуль. Вы берете нужный рецепт из книги и используете его для приготовления блюда, не переписывая его каждый раз."
3. **Импорт модулей (import, from...import) (30 минут)**
   * **Теоретический материал:**
     + Чтобы использовать код из другого модуля, его нужно **импортировать**.
     + **import имя\_модуля:** Импортирует весь модуль. Для доступа к содержимому модуля нужно использовать префикс имя\_модуля..

Python

# Файл my\_module.py

def greet(name):

print(f"Привет, {name}!")

PI = 3.14159

# В другом файле (например, main.py)

import my\_module

my\_module.greet("Алиса") # Вызов функции из модуля

print(my\_module.PI) # Доступ к переменной из модуля

* + - **import имя\_модуля as новое\_имя:** Импортирует модуль и дает ему короткий псевдоним.

Python

import my\_module as mm

mm.greet("Боб")

* + - **from имя\_модуля import имя\_функции/переменной/класс:** Импортирует только конкретные части модуля. Эти части можно использовать напрямую, без префикса.

Python

from my\_module import greet, PI

greet("Чарли") # Используем напрямую

print(PI)

* + - **from имя\_модуля import \* (не рекомендуется!):** Импортирует **все** из модуля. Может привести к конфликтам имен, если у вас или у других модулей есть функции/переменные с такими же названиями. Избегайте этого в больших проектах.
  + **Демонстрация:** Создать два файла: my\_utils.py с парой функций (например, add, subtract) и main.py. В main.py показать разные способы импорта и использования функций из my\_utils.py.

1. **Встроенные модули и пакеты (20 минут)**
   * **Теоретический материал:**
     + Python поставляется с огромной **стандартной библиотекой** – набором полезных модулей, которые уже установлены и готовы к использованию.
     + **Примеры встроенных модулей:**
       - math: Математические функции (квадратный корень sqrt(), ceil(), floor(), pi).
         * Пример: import math; print(math.sqrt(16))
       - random: Генерация случайных чисел.
         * random.randint(a, b): Случайное целое число в диапазоне от a до b(включительно).
         * random.choice(sequence): Случайный элемент из списка/строки.
         * Пример: import random; print(random.randint(1, 10))
       - os: Взаимодействие с операционной системой (работа с файлами и папками).
       - datetime: Работа с датами и временем.
     + **Пакет (Package):** Это способ организации модулей. Пакет – это папка, которая содержит несколько модулей и специальный файл \_\_init\_\_.py (который может быть пустым, но указывает Python, что это пакет). Пакеты позволяют создавать иерархии модулей.
       - import package.module
       - from package.module import function
       - Аналогия: "Книга рецептов (пакет) может быть разделена на главы (подпакеты), и каждая глава содержит рецепты (модули)."
     + **Виртуальное окружение (краткое упоминание):** Для больших проектов часто используются виртуальные окружения, чтобы изолировать зависимости пакетов для каждого проекта и избежать конфликтов версий. Это тема для более продвинутого уровня, но стоит упомянуть о существовании.
   * **Демонстрация:** Показать использование math.sqrt() и random.randint(). Кратко объяснить структуру папок для пакета.
2. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Импортируйте модуль math и используйте math.pi для вычисления площади круга с радиусом 5.
     + Импортируйте модуль random и выведите случайное число от 1 до 100.
   * **Средний уровень:**
     + **"Игра 'Орел или Решка'":** Используйте random.choice() для симуляции подбрасывания монеты. Выведите "Орел" или "Решка".
     + **"Модуль для расчетов":** Создайте файл calculator.py с функциями add(a, b), subtract(a, b), multiply(a, b), divide(a, b). В другом файле main\_app.py импортируйте эти функции и используйте их для выполнения простых расчетов.
   * **Сложный уровень:**
     + **"Генератор паролей":** Используйте модуль random и строки (для букв, цифр, символов) для создания функции, которая генерирует случайный пароль заданной длины.
     + **Игровой элемент: "Бросок кубика":**
       - Задание: "Используйте модуль random для симуляции броска двух шестигранных кубиков.
       - Выведите результат каждого кубика и их сумму.
       - Создайте функцию roll\_dice(), которая возвращает сумму бросков.
       - Попросите пользователя бросить кубики, используя эту функцию."
3. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: модули для организации, import, стандартная библиотека, пакеты.
   * Превью следующего урока: "На следующем занятии мы познакомимся с более продвинутыми концепциями работы со строками, включая форматирование, работу с регулярными выражениями и основы парсинга текста, что очень полезно для обработки текстовых данных!"

**Материалы/Ресурсы:**

* Презентация со схемами импорта модулей.
* Примеры использования math и random.

**Домашнее задание:**

* Создайте модуль my\_greetings.py с функцией hello(name). В основном файле импортируйте эту функцию и используйте.
* Создайте список из 10 случайных чисел от 1 до 50, используя цикл и random.randint(). Выведите этот список.
* Напишите программу, которая использует модуль datetime для вывода текущей даты и времени. (Найти в документации datetime.datetime.now()).

## Урок №20: Работа с датой и временем.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомить с модулем datetime для работы с датами и временем.
* Научиться получать текущую дату и время.
* Изучить, как создавать объекты даты и времени.
* Освоить форматирование даты и времени для вывода в нужном формате.
* Понять основы арифметики дат (разница между датами).

**Основные понятия:**

* Модуль datetime
* datetime.datetime.now()
* datetime.date
* datetime.time
* datetime.timedelta
* Форматирование даты (strftime(), strptime())
* UNIX-время (кратко)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое модуль? Зачем его импортировать? В чем разница между import и from...import? Приведите примеры встроенных модулей.
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Введение в datetime. Текущая дата и время (25 минут)**
   * **Теоретический материал:**
     + Во многих программах (логирование, расписание, игры, веб-приложения) необходимо работать с датами и временем. Python предоставляет мощный встроенный модуль datetime для этих целей.
     + **import datetime:** Модуль, содержащий классы для работы с датами и временем.
     + **datetime.datetime:** Класс, который представляет дату и время (год, месяц, день, час, минута, секунда, микросекунда).
     + **Получение текущих даты и времени:**
       - datetime.datetime.now(): Возвращает текущую локальную дату и время.
       - datetime.datetime.utcnow(): Возвращает текущее время по UTC (универсальное скоординированное время, без учета часового пояса).
       - Пример:

Python

import datetime

now = datetime.datetime.now()

print(f"Текущая дата и время: {now}")

* + - **Создание объектов даты и времени вручную:**
      * datetime.date(год, месяц, день): Создает объект даты.
      * datetime.time(час, минута, секунда): Создает объект времени.
      * datetime.datetime(год, месяц, день, час, минута, секунда): Создает полный объект даты и времени.
      * Пример: my\_birthday = datetime.date(2000, 5, 15)
  + **Демонстрация:** Получение и вывод текущей даты/времени. Создание даты рождения или конкретного события.

1. **Форматирование даты и времени (strftime()) (30 минут)**
   * **Теоретический материал:**
     + Когда мы выводим объект datetime, он отображается в стандартном формате. Но часто нам нужен специфический формат (например, "15 мая 2000 года" или "10:30 PM").
     + **strftime(format\_string):** Метод объекта datetime, который форматирует дату и время в строку согласно указанному format\_string.
     + **Коды форматирования (основные):**
       - %Y: Год (четырехзначный, например, 2023)
       - %m: Месяц как число (01-12)
       - %d: День месяца (01-31)
       - %H: Час (24-часовой формат, 00-23)
       - %I: Час (12-часовой формат, 01-12)
       - %M: Минута (00-59)
       - %S: Секунда (00-59)
       - %p: AM/PM
       - %A: Полное название дня недели (Monday, Tuesday...)
       - %B: Полное название месяца (January, February...)
       - %c: Локальное представление даты и времени
       - %x: Локальное представление даты
       - %X: Локальное представление времени
     + Пример:

Python

import datetime

now = datetime.datetime.now()

formatted\_date = now.strftime("%d-%m-%Y %H:%M:%S")

print(formatted\_date) # Например: 07-07-2025 15:30:00

formatted\_verbose = now.strftime("Сегодня %A, %d %B %Y года.")

print(formatted\_verbose) # Например: Сегодня Понедельник, 07 Июль 2025 года.

* + - **strptime(string, format\_string):** (Обратная операция) Функция модуля datetime, которая парсит строку с датой/временем в объект datetime по заданному формату.
      * Пример: date\_str = "15-05-2000"; parsed\_date = datetime.datetime.strptime(date\_str, "%d-%m-%Y")
  + **Демонстрация:** Форматирование текущей даты/времени в различные удобные форматы. Парсинг строки даты.

1. **Арифметика дат (timedelta) (15 минут)**
   * **Теоретический материал:**
     + Мы можем складывать и вычитать даты, чтобы найти разницу во времени или получить дату, которая будет через N дней.
     + **datetime.timedelta:** Класс, который представляет продолжительность (разницу) во времени. Используется для выполнения арифметических операций с датами и временем.
       - Создание: datetime.timedelta(days=N, hours=M, minutes=K, ...)
     + **Операции:**
       - дата\_время + timedelta = новая дата/время
       - дата\_время - timedelta = новая дата/время
       - дата\_время1 - дата\_время2 = timedelta (разница)
     + Пример:

Python

import datetime

today = datetime.date.today()

tomorrow = today + datetime.timedelta(days=1)

print(f"Завтра будет: {tomorrow}")

in\_week = today + datetime.timedelta(weeks=1)

print(f"Через неделю: {in\_week}")

diff = tomorrow - today

print(f"Разница: {diff}")

* + **Демонстрация:** Вычисление даты через N дней. Вычисление разницы между двумя датами.

1. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Выведите текущее время в формате ЧЧ:ММ:СС (например, 15:30:45).
     + Создайте объект даты вашего дня рождения и выведите его.
   * **Средний уровень:**
     + **"Сколько дней до Нового года?":** Вычислите и выведите, сколько дней осталось до ближайшего Нового года (01.01 следующего года).
     + **"Время работы программы":**
       - В начале программы сохраните текущее время: start\_time = datetime.datetime.now().
       - Выполните какую-нибудь простую операцию (например, цикл на 1000 итераций).
       - В конце программы снова получите текущее время: end\_time = datetime.datetime.now().
       - Вычислите и выведите разницу между end\_time и start\_time.
   * **Сложный уровень:**
     + **"Таймер обратного отсчета":** Запросите у пользователя дату и время какого-либо события (например, "31.12.2025 23:59:00"). Создайте функцию countdown(target\_datetime), которая будет вычислять и выводить, сколько времени осталось до этого события (дни, часы, минуты, секунды).
       - *Пока можно просто вывести разницу, а не интерактивный таймер.*
     + **Игровой элемент: "Дневник игровых сессий":**
       - Задание: "Создайте функцию log\_session(player\_name, session\_duration\_minutes), которая записывает в файл sessions.txt строку вида: [Дата и время начала сессии] - Игрок: [player\_name], Продолжительность: [session\_duration\_minutes] мин.
       - Вызовите эту функцию несколько раз, симулируя разные игровые сессии."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: модуль datetime, текущая дата/время, форматирование, timedelta.
   * Превью следующего урока: "На следующем занятии мы начнем изучать работу с базами данных, в частности, с SQLite – простой, но мощной реляционной базой данных, которая очень удобна для небольших проектов и позволит нам сохранять сложные структурированные данные!"

**Материалы/Ресурсы:**

* Презентация с таблицей кодов форматирования strftime().
* Примеры работы с timedelta.

**Домашнее задание:**

* Написать программу, которая запрашивает у пользователя его день, месяц и год рождения. Вычислите и выведите точный возраст пользователя в годах, месяцах и днях.
* Создать программу "Календарь событий":
  + Создайте список из нескольких словарей, где каждый словарь - это событие с полями: name, date (в формате "ГГГГ-ММ-ДД").
  + Напишите функцию show\_upcoming\_events(events\_list), которая принимает этот список и выводит только те события, которые произойдут в будущем.
  + *Подсказка: используйте strptime для преобразования строки даты в объект datetime.date для сравнения.*

Урок №21: Работа с JSON данными**.**

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомить с форматом JSON (JavaScript Object Notation) как распространенным способом обмена данными.
* Изучить, как Python работает с JSON с помощью встроенного модуля json.
* Научиться преобразовывать Python-объекты (словари, списки) в JSON-строки (json.dumps()).
* Научиться парсить JSON-строки обратно в Python-объекты (json.loads()).
* Освоить чтение и запись JSON-данных в файлы.

**Основные понятия:**

* JSON (JavaScript Object Notation)
* Модуль json
* json.dumps() (Python -> JSON-строка)
* json.loads() (JSON-строка -> Python)
* json.dump() (Python -> JSON-файл)
* json.load() (JSON-файл -> Python)
* Сериализация / Десериализация

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Для чего нужен модуль datetime? Как получить текущую дату? Как отформатировать дату? Как вычислить разницу между датами?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Что такое JSON и зачем он нужен? (20 минут)**
   * **Теоретический материал:**
     + Когда мы работаем с веб-сервисами, API, или просто хотим сохранить структурированные данные в файле таким образом, чтобы их легко могли прочитать и другие программы (не только на Python), нам нужен универсальный формат.
     + **JSON (JavaScript Object Notation):** Легковесный формат обмена данными. Он удобен для человека и легко парсится машинами. JSON широко используется в веб-разработке для обмена данными между сервером и браузером/мобильным приложением.
     + **Аналогия:** "JSON — это как универсальный язык, на котором 'говорят' разные программы, чтобы обмениваться информацией. Он похож на словари и списки Python."
     + **Основные типы данных в JSON (соответствия с Python):**
       - Объекты (словари Python): { "ключ": "значение" }
       - Массивы (списки Python): [ "элемент1", "элемент2" ]
       - Строки: "текст"
       - Числа: 123, 3.14
       - Логические значения: true, false (в Python True, False)
       - null (в Python None)
     + **Ключевая идея:** JSON позволяет нам представлять сложные структуры данных (такие как список пользователей, у каждого из которых есть имя, возраст, роли и т.д.) в виде текста, который легко передавать и хранить.
3. **Работа с JSON в Python: json.dumps() и json.loads() (30 минут)**
   * **Теоретический материал:**
     + Python имеет встроенный модуль json для работы с JSON-данными.
     + **json.dumps(python\_object):** (dump string - сбросить в строку)
       - Преобразует (сериализует) Python-объект (словарь, список и т.д.) в строку в формате JSON.
       - Пример:

Python

import json

data = {

"name": "Алиса",

"age": 30,

"is\_student": False,

"courses": ["Python", "JavaScript"]

}

json\_string = json.dumps(data, indent=4, ensure\_ascii=False)

# indent=4 делает вывод красивым (с отступами)

# ensure\_ascii=False позволяет выводить кириллицу без экранирования

print(json\_string)

# Вывод:

# {

# "name": "Алиса",

# "age": 30,

# "is\_student": false,

# "courses": [

# "Python",

# "JavaScript"

# ]

# }

* + - **json.loads(json\_string):** (load string - загрузить из строки)
      * Парсит (десериализует) строку в формате JSON и преобразует ее обратно в Python-объект (словарь, список и т.д.).
      * Пример:

Python

json\_data\_str = '{"product": "Laptop", "price": 1200, "in\_stock": true}'

product\_info = json.loads(json\_data\_str)

print(product\_info["product"]) # Выведет: Laptop

print(type(product\_info)) # Выведет: <class 'dict'>

* + **Демонстрация:** Преобразование словаря Python в JSON-строку и обратно. Показать, как indentулучшает читаемость.

1. **Чтение и запись JSON в файлы (json.dump(), json.load()) (20 минут)**
   * **Теоретический материал:**
     + Для работы с JSON-файлами используются методы json.dump() и json.load(). Они автоматически обрабатывают чтение/запись в файл, нам не нужно вручную читать/писать строки.
     + **json.dump(python\_object, file\_object):**
       - Записывает Python-объект в файл в формате JSON.
       - Пример:

Python

import json

data = {"city": "Бишкек", "population": 1200000}

with open("city.json", "w", encoding="utf-8") as f:

json.dump(data, f, indent=4, ensure\_ascii=False)

# В файле city.json будет:

# {

# "city": "Бишкек",

# "population": 1200000

# }

* + - **json.load(file\_object):**
      * Считывает JSON-данные из файла и преобразует их в Python-объект.
      * Пример:

Python

import json

with open("city.json", "r", encoding="utf-8") as f:

loaded\_data = json.load(f)

print(loaded\_data["city"]) # Выведет: Бишкек

* + **Демонстрация:** Запись словаря в JSON-файл. Чтение словаря из JSON-файла.

1. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Создайте Python-словарь person = {"name": "Иван", "age": 25}. Преобразуйте его в JSON-строку и выведите.
     + Создайте JSON-строку "{\"item\": \"Apple\", \"price\": 0.5}". Преобразуйте ее в Python-словарь и выведите цену.
   * **Средний уровень:**
     + **"Сохранение настроек":** Создайте словарь settings = {"theme": "dark", "language": "ru", "notifications": True}. Запишите этот словарь в файл config.json в читаемом формате. Затем прочитайте его обратно и выведите на экран.
     + **"Список дел в JSON":** Создайте список словарей, где каждый словарь представляет задачу: {"title": "Купить продукты", "completed": False}. Сохраните этот список в файл tasks.json.
   * **Сложный уровень:**
     + **"База данных учеников":** Создайте функцию add\_student(name, age, grade) которая:
       1. Считывает текущий список учеников из файла students.json (если файла нет, начинает с пустого списка).
       2. Добавляет нового ученика в виде словаря в список.
       3. Записывает обновленный список обратно в students.json.
       - *Подсказка: используйте try-except FileNotFoundError для проверки наличия файла.*
     + **Игровой элемент: "Сохранение игрового прогресса":**
       - Задание: "У вас есть словарь game\_state = {'player\_name': 'Герой', 'level': 5, 'score': 1200, 'inventory': ['меч', 'щит']}.
       - Создайте функцию save\_game(state\_data, filename), которая сохраняет state\_data в JSON-файл с именем filename.
       - Создайте функцию load\_game(filename), которая загружает игровой прогресс из JSON-файла.
       - Продемонстрируйте сохранение и загрузку состояния игры."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: JSON для обмена данными, json.dumps/loads для строк, json.dump/load для файлов.
   * Превью следующего урока: "На следующем занятии мы начнем изучать работу с базами данных, в частности, с SQLite – простой, но мощной реляционной базой данных, которая очень удобна для небольших проектов и позволит нам сохранять сложные структурированные данные!"

**Материалы/Ресурсы:**

* Презентация с примерами JSON-структур.
* Сравнение Python-типов и JSON-типов.
* Примеры кода для сериализации/десериализации.

**Домашнее задание:**

* Создать программу для управления контактами:
  + Запрашивать у пользователя имя и телефон.
  + Сохранять контакты в список словарей, а затем этот список сохранять в файл contacts.json.
  + Программа должна уметь дописывать новые контакты, а не перезаписывать их каждый раз.
* Написать функцию read\_user\_profile(username), которая считывает информацию о пользователе из файла users.json. Если пользователя нет или файл не найден, функция должна обрабатывать ошибку и возвращать None.

Урок №22: Введение в базы данных. SQLite**.**

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять, что такое база данных и зачем она нужна.
* Познакомить с реляционными базами данных и SQL.
* Изучить основы работы с SQLite как простой и удобной БД для начинающих.
* Научиться подключаться к базе данных SQLite и создавать таблицы.
* Освоить выполнение базовых SQL-запросов на вставку, выборку данных.

**Основные понятия:**

* База данных (Database)
* Реляционная база данных (RDBMS)
* SQL (Structured Query Language)
* SQLite
* Таблица (Table)
* Столбец (Column) / Поле (Field)
* Запись (Record) / Строка (Row)
* Подключение к БД (Connection)
* Курсор (Cursor)
* CREATE TABLE
* INSERT INTO
* SELECT
* COMMIT
* CLOSE

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое JSON? Как преобразовать Python-словарь в JSON-строку? Как считать JSON-данные из файла?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Что такое база данных и SQL? (20 минут)**
   * **Теоретический материал:**
     + Мы уже умеем сохранять данные в файлы (текстовые, JSON). Но для сложных, больших и взаимосвязанных данных, а также для быстрого поиска и модификации, файлы не очень подходят.
     + **База данных (Database - БД):** Упорядоченный набор данных, который хранится и управляется специальной программой (СУБД - Система Управления Базами Данных). БД позволяют эффективно хранить, извлекать, изменять и удалять большие объемы структурированной информации.
     + **Реляционная база данных:** Самый распространенный тип БД, где данные хранятся в таблицах. Каждая таблица состоит из строк (записей) и столбцов (полей). Таблицы могут быть связаны друг с другом.
       - **Аналогия:** "База данных — это как очень умная и большая электронная таблица (Excel), где данные разных таблиц могут быть связаны, и у вас есть мощные инструменты для быстрого поиска и фильтрации."
     + **SQL (Structured Query Language - Язык структурированных запросов):** Стандартный язык для взаимодействия с реляционными базами данных. С помощью SQL мы "говорим" базе данных, что хотим сделать (создать таблицу, добавить данные, выбрать данные, обновить, удалить).
       - **Основные типы команд SQL (кратко):**
         * **DDL (Data Definition Language):** CREATE, ALTER, DROP (для работы со структурой БД).
         * **DML (Data Manipulation Language):** INSERT, SELECT, UPDATE, DELETE (для работы с данными).
3. **SQLite: Подключение и создание таблицы (35 минут)**
   * **Теоретический материал:**
     + **SQLite:** Очень популярная, легкая, самодостаточная, бессерверная, транзакционная реляционная база данных.
       - **"Бессерверная":** Не требует отдельного "сервера" БД, как MySQL или PostgreSQL. Вся база данных хранится в одном файле на диске. Это удобно для небольших проектов, мобильных приложений.
       - **"Встроена в Python":** Python имеет встроенный модуль sqlite3 для работы с SQLite, поэтому ничего дополнительно устанавливать не нужно.
     + **Шаги для работы с SQLite в Python:**
       1. **Импорт модуля:** import sqlite3
       2. **Подключение к БД:** conn = sqlite3.connect('my\_database.db')
          - Если файла my\_database.db нет, он будет создан.
       3. **Создание курсора:** cursor = conn.cursor()
          - Курсор — это объект, который позволяет нам выполнять SQL-запросы и получать результаты.
       4. **Выполнение SQL-запроса:** cursor.execute("SQL-запрос")
       5. **Сохранение изменений (Commit):** conn.commit()
          - Очень важный шаг! После операций INSERT, UPDATE, DELETE изменения нужно зафиксировать, иначе они не сохранятся в БД.
       6. **Закрытие соединения:** conn.close()
          - Закрывает соединение с базой данных, освобождая ресурсы.
       - **Рекомендуется with для соединения:**

Python

import sqlite3

with sqlite3.connect('my\_database.db') as conn:

cursor = conn.cursor()

# Здесь все операции с курсором

# conn.close() вызывается автоматически

* + - **Создание таблицы (CREATE TABLE):**
      * Определяем имя таблицы и имена/типы столбцов.
      * Типы данных в SQLite (основные): INTEGER (целые числа), TEXT (строки), REAL (числа с плавающей точкой), BLOB (двоичные данные).
      * Пример:

SQL

CREATE TABLE users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

age INTEGER

);

* + - * + PRIMARY KEY AUTOINCREMENT: id будет уникальным и будет автоматически увеличиваться.
        + NOT NULL: Это поле не может быть пустым.
  + **Демонстрация:** Подключение к БД. Создание таблицы products с полями id, name, price. Проверка наличия файла .db.

1. **Вставка данных (INSERT INTO) и выборка данных (SELECT) (15 минут)**
   * **Теоретический материал:**
     + **Вставка данных (INSERT INTO):** Добавление новых строк в таблицу.
       - Синтаксис: INSERT INTO имя\_таблицы (столбец1, столбец2) VALUES (значение1, значение2);
       - **Важно: Используйте плейсхолдеры (?) для данных!** Это предотвращает SQL-инъекции (угроза безопасности) и правильно обрабатывает специальные символы.
       - Пример:

Python

product\_name = "Ноутбук"

product\_price = 1200.50

cursor.execute("INSERT INTO products (name, price) VALUES (?, ?)", (product\_name, product\_price))

conn.commit()

* + - **Выборка данных (SELECT):** Получение данных из таблицы.
      * Синтаксис: SELECT столбец1, столбец2 FROM имя\_таблицы WHERE условие;
      * \* (звездочка) выбирает все столбцы.
      * **Получение результатов:**
        + cursor.fetchone(): Возвращает одну следующую строку (как кортеж) или None.
        + cursor.fetchall(): Возвращает все оставшиеся строки (как список кортежей).
        + Итерирование по курсору: for row in cursor:
      * Пример:

Python

cursor.execute("SELECT name, price FROM products")

all\_products = cursor.fetchall()

for product in all\_products:

print(f"Название: {product[0]}, Цена: {product[1]}")

cursor.execute("SELECT \* FROM products WHERE price > ?", (1000,))

expensive\_products = cursor.fetchall()

* + **Демонстрация:** Добавление нескольких записей в таблицу products. Выборка всех продуктов. Выборка продуктов по условию (например, дороже определенной цены).

1. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Подключитесь к БД library.db. Создайте таблицу books с полями id (INTEGER PRIMARY KEY AUTOINCREMENT), title (TEXT), author (TEXT).
     + Добавьте в таблицу 2-3 книги. Выведите все книги из таблицы.
   * **Средний уровень:**
     + **"Список пользователей":** Создайте БД users.db и таблицу users с полями id, username(TEXT UNIQUE NOT NULL), email (TEXT).
     + Напишите функцию add\_user(username, email), которая добавляет нового пользователя. Обработайте ошибку, если пользователь с таким username уже существует (используйте try-except sqlite3.IntegrityError).
     + Напишите функцию get\_all\_users(), которая возвращает список всех пользователей.
   * **Сложный уровень:**
     + **"Простой менеджер задач":**
       1. Создайте БД tasks.db и таблицу tasks с полями id, description (TEXT NOT NULL), is\_completed (INTEGER, 0 или 1).
       2. Напишите функцию add\_task(description).
       3. Напишите функцию view\_tasks(completed\_only=False): если completed\_only True, показывает только выполненные; иначе - все.
       4. Напишите функцию mark\_completed(task\_id).
       5. Создайте простое меню для пользователя (добавить, посмотреть, отметить выполненной).
     + **Игровой элемент: "Хранение рекордов":**
       - Задание: "Создайте БД game\_scores.db и таблицу scores с полями player\_name (TEXT), score (INTEGER), date (TEXT - для хранения даты в текстовом формате).
       - Создайте функцию add\_high\_score(player\_name, score). Дата должна быть текущей (используйте datetime).
       - Создайте функцию get\_top\_scores(limit), которая возвращает limit лучших результатов, отсортированных по убыванию счета."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: БД для структурированных данных, SQL, SQLite, подключение, курсор, CREATE, INSERT, SELECT, commit.
   * Превью следующего урока: "На следующем занятии мы продолжим работать с базами данных, научимся обновлять и удалять данные, а также изучим более сложные запросы SQL для фильтрации и сортировки!"

**Материалы/Ресурсы:**

* Презентация с табличной структурой данных.
* Схемы работы connect, cursor, execute, commit.
* Основные команды SQL для DDL и DML.

**Домашнее задание:**

* Создать программу, которая имитирует хранение контактов:
  + Таблица contacts с id, name, phone.
  + Функции для: добавления нового контакта, просмотра всех контактов, поиска контакта по имени.
* Написать программу, которая сохраняет историю поисковых запросов пользователя.
  + Таблица search\_history с id, query (текст запроса), timestamp (время запроса).
  + Функции для добавления запроса и просмотра всей истории.

## Урок №23: Обновление и удаление данных в SQLite. Расширенные SELECT запросы.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Научиться изменять существующие записи в базе данных (UPDATE).
* Научиться удалять записи из базы данных (DELETE).
* Освоить более сложные SQL-запросы SELECT с использованием WHERE, ORDER BY, LIMIT.
* Понять важность безопасного выполнения запросов с параметрами.

**Основные понятия:**

* UPDATE
* SET
* WHERE (для UPDATE, DELETE, SELECT)
* DELETE FROM
* ORDER BY
* ASC (возрастание)
* DESC (убывание)
* LIMIT
* Фильтрация данных
* Сортировка данных

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое база данных? Зачем нам SQL? Как подключиться к SQLite? Как создать таблицу? Как добавить и выбрать данные?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Изменение данных (UPDATE) (25 минут)**
   * **Теоретический материал:**
     + После того, как данные добавлены в базу, часто возникает необходимость их изменить.
     + **UPDATE:** SQL-команда для модификации существующих записей в таблице.
       - Синтаксис: UPDATE имя\_таблицы SET столбец1 = значение1, столбец2 = значение2 WHERE условие;
       - **Важно:** Предложение WHERE очень критично! Если его не указать, будут изменены ВСЕ записи в таблице!
       - Используйте плейсхолдеры (?) для значений.
       - Пример:

Python

import sqlite3

conn = sqlite3.connect('my\_database.db')

cursor = conn.cursor()

# Обновить цену продукта с именем "Ноутбук"

new\_price = 1300.00

product\_name = "Ноутбук"

cursor.execute("UPDATE products SET price = ? WHERE name = ?", (new\_price, product\_name))

conn.commit()

# Обновить имя пользователя с ID 1

new\_user\_name = "Алиса Петрова"

user\_id = 1

cursor.execute("UPDATE users SET name = ? WHERE id = ?", (new\_user\_name, user\_id))

conn.commit()

* + **Демонстрация:** Создать таблицу students с полями id, name, grade. Добавить несколько студентов. Обновить оценку у одного из студентов.

1. **Удаление данных (DELETE) (25 минут)**
   * **Теоретический материал:**
     + Иногда данные становятся ненужными, и их нужно удалить из базы.
     + **DELETE FROM:** SQL-команда для удаления записей из таблицы.
       - Синтаксис: DELETE FROM имя\_таблицы WHERE условие;
       - **Важно:** Как и для UPDATE, WHERE условие абсолютно необходимо! Если его не указать, будут удалены ВСЕ записи из таблицы!
       - Пример:

Python

# Удалить продукт с именем "Старый телефон"

old\_product = "Старый телефон"

cursor.execute("DELETE FROM products WHERE name = ?", (old\_product,))

conn.commit()

# Удалить всех пользователей старше 60 лет

cursor.execute("DELETE FROM users WHERE age > ?", (60,))

conn.commit()

* + **Демонстрация:** Удалить одного студента по имени или ID. Показать ошибку (или предупредить), если WHERE не указать и удалить все.

1. **Расширенные SELECT запросы (WHERE, ORDER BY, LIMIT) (20 минут)**
   * **Теоретический материал:**
     + Мы уже умеем выбирать все данные (SELECT \* FROM table). Теперь научимся их фильтровать и сортировать.
     + **WHERE:** Используется с SELECT для фильтрации строк на основе заданного условия.
       - Операторы: =, !=, <, >, <=, >=, AND, OR, LIKE (для поиска по шаблону), IN (для списка значений).
       - Пример: SELECT \* FROM products WHERE price > 100 AND in\_stock = 1;
     + **ORDER BY:** Сортирует результаты запроса.
       - Синтаксис: SELECT ... FROM ... ORDER BY столбец [ASC|DESC];
       - ASC: По возрастанию (по умолчанию).
       - DESC: По убыванию.
       - Можно сортировать по нескольким столбцам: ORDER BY price DESC, name ASC;
       - Пример: SELECT \* FROM users ORDER BY age DESC;
     + **LIMIT:** Ограничивает количество возвращаемых строк. Очень полезно для пагинации или получения "топ-N" записей.
       - Синтаксис: SELECT ... FROM ... LIMIT количество;
       - Пример: SELECT \* FROM products ORDER BY price DESC LIMIT 5; (Топ-5 самых дорогих продуктов)
   * **Демонстрация:** Выбрать студентов с оценкой выше 4. Отсортировать студентов по имени. Получить 3 самых молодых студента.
2. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Используя таблицу books из прошлого урока:
       - Обновите автора одной из книг.
       - Удалите одну книгу.
       - Выберите все книги и отсортируйте их по названию в алфавитном порядке.
   * **Средний уровень:**
     + **"Управление товарами":**
       - В таблице products (из прошлого урока):
         * Обновите цену товара по его ID.
         * Удалите все товары, цена которых ниже 100.
         * Выберите 5 самых дорогих товаров.
   * **Сложный уровень:**
     + **"Система управления курсами":**
       1. Создайте БД courses.db и таблицу courses с полями: id, title (TEXT), instructor(TEXT), duration\_hours (INTEGER).
       2. Реализуйте функции для:
          - Добавления нового курса.
          - Обновления преподавателя для курса по его названию.
          - Удаления курса по его ID.
          - Получения всех курсов, отсортированных по продолжительности (от коротких к длинным).
          - Получения курсов, которые ведут определенный преподаватель.
       3. Создайте простое текстовое меню для взаимодействия с этой системой.
     + **Игровой элемент: "Обновление игровых персонажей":**
       - Задание: "У вас есть таблица characters с полями name, level, xp (опыт).
       - Создайте функцию gain\_xp(character\_name, amount), которая добавляет amount к xpперсонажа.
       - Если xp достигает определенного порога (например, 100), уровень (level) персонажа увеличивается, а xp обнуляется.
       - Продемонстрируйте "прокачку" персонажа."
3. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: UPDATE, DELETE, расширенные SELECT с WHERE, ORDER BY, LIMIT.
   * Превью следующего урока: "На следующем занятии мы рассмотрим более продвинутые концепции SQL, такие как агрегатные функции (SUM, AVG, COUNT), группировка данных (GROUP BY) и объединение таблиц (JOIN), а также поговорим об индексах для ускорения запросов!"

**Материалы/Ресурсы:**

* Презентация с синтаксисом UPDATE, DELETE, SELECT с условиями.
* Примеры операторов WHERE.
* Примеры сортировки и ограничения.

**Домашнее задание:**

* Создать программу для управления библиотекой (продолжение из прошлых уроков):
  + Добавить функцию update\_book\_title(old\_title, new\_title).
  + Добавить функцию delete\_books\_by\_author(author\_name).
  + Добавить функцию get\_books\_by\_author(author\_name), которая возвращает все книги данного автора, отсортированные по названию.
  + Добавить функцию get\_top\_N\_books(N), которая возвращает N первых книг по названию (или по ID).

## Урок №24: Агрегатные функции, GROUP BY и JOIN в SQLite.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомить с агрегатными функциями SQL (COUNT, SUM, AVG, MIN, MAX).
* Изучить, как группировать данные с помощью GROUP BY.
* Освоить объединение данных из нескольких таблиц с помощью JOIN.
* Кратко упомянуть об индексах для оптимизации производительности БД.

**Основные понятия:**

* Агрегатные функции: COUNT, SUM, AVG, MIN, MAX
* GROUP BY
* JOIN (INNER JOIN)
* Внешний ключ (Foreign Key)
* Индекс (Index)
* Оптимизация БД

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Как изменить данные в БД? Как удалить данные? Как выбрать данные по условию? Как отсортировать и ограничить выборку?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Агрегатные функции (25 минут)**
   * **Теоретический материал:**
     + Часто нам нужно не просто получить отдельные записи, а вычислить что-то по всем записям или по группе записей (например, общее количество, сумму, среднее значение).
     + **Агрегатные функции:** SQL-функции, которые выполняют вычисления над набором строк и возвращают одно результирующее значение.
       - COUNT(столбец): Считает количество строк (или непустых значений в столбце). COUNT(\*) считает все строки.
       - SUM(столбец): Суммирует значения в числовом столбце.
       - AVG(столбец): Вычисляет среднее значение в числовом столбце.
       - MIN(столбец): Находит минимальное значение в столбце.
       - MAX(столбец): Находит максимальное значение в столбце.
       - Пример:

SQL

SELECT COUNT(\*) FROM users; -- Общее количество пользователей

SELECT SUM(price) FROM products; -- Сумма цен всех продуктов

SELECT AVG(grade) FROM students; -- Средняя оценка студентов

SELECT MAX(age) FROM users; -- Максимальный возраст пользователя

* + **Демонстрация:** Использовать существующую таблицу products или students. Посчитать общее количество записей, найти среднюю цену/оценку, минимальное/максимальное значение.

1. **Группировка данных (GROUP BY) (25 минут)**
   * **Теоретический материал:**
     + Агрегатные функции полезны, но что, если мы хотим применить их не ко всей таблице, а к группам записей? Например, посчитать среднюю оценку для студентов по каждому классу, или количество товаров по каждой категории.
     + **GROUP BY:** Группирует строки, имеющие одинаковые значения в указанных столбцах, в одну сводную строку. После группировки, агрегатные функции применяются к каждой группе отдельно.
       - Синтаксис: SELECT столбец\_группировки, АГРЕГАТНАЯ\_ФУНКЦИЯ(столбец) FROM таблица GROUP BY столбец\_группировки;
       - Пример:

SQL

-- Таблица orders: id, product\_name, category, price

SELECT category, SUM(price) FROM orders GROUP BY category;

-- Выведет: категория, сумма\_цен\_в\_этой\_категории

* + - * **HAVING:** Похож на WHERE, но используется для фильтрации ГРУПП после GROUP BY.
        + Пример: SELECT category, SUM(price) FROM orders GROUP BY category HAVING SUM(price) > 1000;
  + **Демонстрация:** Создать таблицу employees с department и salary. Посчитать среднюю зарплату по каждому отделу. Посчитать количество сотрудников в каждом отделе.

1. **Объединение таблиц (JOIN) (20 минут)**
   * **Теоретический материал:**
     + В реляционных базах данных информация часто разбивается на несколько связанных таблиц. Например, users (информация о пользователях) и orders (заказы, сделанные пользователями).
     + **Внешний ключ (Foreign Key):** Столбец в одной таблице, который ссылается на PRIMARY KEY в другой таблице. Он устанавливает связь между таблицами.
       - Пример: В таблице orders может быть поле user\_id, которое ссылается на id в таблице users.
     + **JOIN:** SQL-команда для объединения строк из двух или более таблиц на основе связанного столбца между ними.
       - **INNER JOIN (или просто JOIN):** Возвращает только те строки, для которых есть совпадения в обеих таблицах.
       - Синтаксис:

SQL

SELECT \*

FROM table1

INNER JOIN table2 ON table1.common\_column = table2.common\_column;

* + - * Пример:

SQL

-- Получить все заказы с именем пользователя, который их сделал

SELECT users.name, orders.product\_name, orders.price

FROM users

JOIN orders ON users.id = orders.user\_id;

* + **Демонстрация:** Создать две таблицы: authors (id, name) и books (id, title, author\_id). Связать их по author\_id (в books) и id (в authors). Вывести названия книг вместе с именами их авторов.

1. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + В таблице books (из прошлых уроков):
       - Посчитайте общее количество книг.
       - Найдите самую дорогую книгу (если добавить поле price).
     + Используя employees таблицу: найдите среднюю зарплату по всем сотрудникам.
   * **Средний уровень:**
     + **"Отчет по продажам":** Создайте таблицы products (id, name, category, price) и sales (id, product\_id, quantity, sale\_date).
       - Свяжите их.
       - Сделайте запрос, который покажет общую выручку по каждой категории товаров (используйте GROUP BY).
       - Найдите товар, который был продан наибольшее количество раз.
   * **Сложный уровень:**
     + **"Анализ активности пользователей":**
       - Создайте таблицы users (id, name) и posts (id, user\_id, title, content).
       - Сделайте запрос, который выведет имена пользователей и количество постов, которые они написали.
       - Выведите только тех пользователей, у кого больше 3 постов.
       - Выведите список всех постов, включая имя автора для каждого поста.
     + **Игровой элемент: "Отчет по инвентарю игроков":**
       - Задание: "У вас есть таблица players (id, name) и inventory (id, player\_id, item\_name, quantity).
       - Создайте запросы:
         * Показать, сколько уникальных предметов у каждого игрока.
         * Вывести всех игроков и список их предметов (используя JOIN).
         * Найти игрока, у которого больше всего предметов."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: агрегатные функции, GROUP BY, JOIN.
   * **Завершение Модуля 2.** Обсуждение того, что было изучено.
   * Превью следующего этапа: "Мы закончили Модуль 2, который глубоко погрузил нас в работу с более сложными структурами данных, обработку ошибок, основы ООП и взаимодействие с базами данных. Теперь у нас есть прочная основа для создания более функциональных и надежных программ. Какие шаги мы предпримем дальше? Возможно, перейдем к графическим интерфейсам, веб-разработке, или углубимся в другие аспекты Python?"

**Материалы/Ресурсы:**

* Презентация с примерами агрегатных функций.
* Схемы работы GROUP BY и JOIN.
* Визуализация отношений между таблицами.

**Домашнее задание:**

* Используя таблицы authors и books из примеров:
  + Посчитать количество книг у каждого автора.
  + Вывести имя автора и количество его книг, но только для тех авторов, у которых больше 2 книг.
  + Вывести все книги и их авторов, даже если у автора нет книг (нужно будет кратко упомянуть о LEFT JOIN или просто показать, что INNER JOIN не выведет автора без книг).
* Создать две таблицы: categories (id, name) и products (id, name, category\_id, price). Заполнить их данными. Написать запрос, который выведет название продукта, его цену и название категории, к которой он относится.

# Модуль 3: Веб-разработка, графические интерфейсы и основы автоматизации.

Урок №25: Введение в веб-разработку: HTTP, HTML**.**

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять базовые принципы работы Интернета и протокола HTTP.
* Познакомить с HTML как основным языком для создания веб-страниц.
* Изучить структуру HTML-документа: <!DOCTYPE html>, <html>, <head>, <body>.
* Освоить использование основных HTML-тегов для текста: заголовки (<h1>-<h6>), параграфы (<p>), списки (<ul>, <ol>, <li>).
* Научиться создавать простые веб-страницы.

**Основные понятия:**

* Интернет, Всемирная паутина (WWW)
* Клиент-серверная архитектура
* HTTP (HyperText Transfer Protocol)
* Веб-браузер, Веб-сервер
* URL (Uniform Resource Locator)
* HTML (HyperText Markup Language)
* Тег (Tag), Элемент (Element), Атрибут (Attribute)
* <!DOCTYPE html>, <html>, <head>, <body>
* Заголовки <h1>-<h6>
* Параграфы <p>
* Списки <ul>, <ol>, <li>
* Строчные и блочные элементы (кратко)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Для чего нужны агрегатные функции в SQL? Как использовать GROUP BY? Зачем нужен JOIN?
   * Проверка домашнего задания, обсуждение возникших вопросов по базам данных.
2. **Как работает Интернет? Основы HTTP (25 минут)**
   * **Теоретический материал:**
     + Когда мы открываем сайт в браузере, что происходит?
     + **Клиент-серверная архитектура:**
       - **Клиент (Client):** Обычно это наш веб-браузер (Chrome, Firefox и т.д.), который запрашивает информацию.
       - **Сервер (Server):** Компьютер, который хранит веб-сайты и "подает" их клиентам по запросу.
     + **HTTP (HyperText Transfer Protocol):** Протокол, по которому браузер (клиент) и веб-сервер обмениваются информацией. Это как "язык", на котором они "общаются".
       - **Запрос (Request):** Браузер отправляет запрос серверу (например, "дай мне страницу https://www.google.com/search?q=google.com").
       - **Ответ (Response):** Сервер отправляет ответ браузеру (например, HTML-код страницы Google).
     + **URL (Uniform Resource Locator):** Адрес веб-страницы (например, https://www.google.com).
     + **Всемирная паутина (World Wide Web - WWW):** Система взаимосвязанных гипертекстовых документов, доступных через Интернет.
     + **Ключевая идея:** Веб-разработка — это создание содержимого (веб-страниц) и логики для веб-серверов и клиентов.
   * **Демонстрация:** Открыть "Инструменты разработчика" (Developer Tools) в браузере (F12 или Ctrl+Shift+I), показать вкладку "Network" (Сеть) и продемонстрировать запросы/ответы при загрузке страницы.
3. **HTML: Структура и основные теги (35 минут)**
   * **Теоретический материал:**
     + **HTML (HyperText Markup Language):** Не язык программирования, а язык разметки. Он используется для создания структуры и содержимого веб-страниц. Браузеры "читают" HTML и отображают страницу.
     + **Теги (Tags):** Основные строительные блоки HTML. Обычно они парные (<p>Содержимое</p>) и окружают содержимое. Есть и одиночные теги (например, <br> для переноса строки).
     + **Элементы (Elements):** Тег открытия, содержимое и тег закрытия вместе образуют элемент.
     + **Атрибуты (Attributes):** Добавляют дополнительную информацию к тегу, например, id, class, href, src. Они указываются в теге открытия: <a href="link.html">Ссылка</a>.
     + **Базовая структура HTML-документа:**

HTML

<!DOCTYPE html> <html lang="ru"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Моя первая веб-страница</title> </head>

<body> <h1>Привет, мир!</h1>

<p>Это мой первый параграф на HTML.</p>

</body>

</html>

* + - **Основные теги для текста:**
      * **Заголовки:** <h1>, <h2>, <h3>, <h4>, <h5>, <h6> (от самого большого до самого маленького).
      * **Параграфы:** <p> - для основного текста.
      * **Списки:**
        + <ul> (unordered list - неупорядоченный список, с маркерами)
        + <ol> (ordered list - упорядоченный список, с номерами)
        + <li> (list item - элемент списка)
      * **Пример:**

HTML

<h2>Мои хобби:</h2>

<ul>

<li>Чтение</li>

<li>Программирование</li>

<li>Путешествия</li>

</ul>

<ol>

<li>Первый шаг</li>

<li>Второй шаг</li>

</ol>

* + **Демонстрация:**
    - Создать простой HTML-файл в текстовом редакторе (например, VS Code).
    - Показать базовую структуру.
    - Добавить заголовки, параграфы, разные типы списков.
    - Открыть файл в браузере.

1. **Практические задания (15 минут)**
   * **Легкий уровень:**
     + Создайте HTML-файл my\_page.html.
     + В <head> укажите заголовок вкладки "Моя страница".
     + В <body> создайте заголовок <h1> "Добро пожаловать!".
     + Добавьте параграф <p> с текстом "Это моя первая веб-страница, написанная на HTML!".
     + Откройте файл в браузере.
   * **Средний уровень:**
     + **"Мое резюме (начало)":**
       - Создайте HTML-файл resume.html.
       - Добавьте заголовок <h1> с вашим именем.
       - Заголовок <h2> "Контактная информация".
       - Неупорядоченный список <ul> с вашим Email и телефоном.
       - Заголовок <h2> "Образование".
       - Упорядоченный список <ol> с вашим образованием (например, "Школа", "Университет").
   * **Сложный уровень:**
     + **"Рецепт блюда":**
       - Создайте HTML-файл recipe.html.
       - В заголовке страницы укажите название рецепта.
       - В <body>:
         * Заголовок <h1> с названием блюда.
         * Параграф с кратким описанием блюда.
         * Заголовок <h2> "Ингредиенты".
         * Неупорядоченный список ингредиентов.
         * Заголовок <h2> "Инструкции по приготовлению".
         * Упорядоченный список шагов приготовления.
     + **Проект: "Блог-пост (структура)":**
       - Задание: "Создайте HTML-файл blog\_post.html.
       - В нем должна быть структура блог-поста:
         * Заголовок <h1> поста.
         * Параграф с информацией об авторе и дате.
         * Несколько параграфов <p> с текстом самого поста.
         * Заголовок <h2> "Похожие статьи".
         * Неупорядоченный список <ul> с названиями похожих статей (пока без ссылок)."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: HTTP, HTML, структура документа, основные теги.
   * Превью следующего урока: "На следующем занятии мы продолжим изучение HTML, добавим изображения, ссылки и научимся создавать таблицы. А также начнем знакомство с CSS, чтобы сделать наши страницы красивыми!"

**Материалы/Ресурсы:**

* Презентация с основами HTML.
* Схемы клиент-серверной архитектуры.
* Шпаргалка по основным HTML-тегам.

**Домашнее задание:**

* Создайте HTML-файл animals.html:
  + Заголовок <h1> "Мои любимые животные".
  + Для каждого животного:
    - Заголовок <h2> с его названием.
    - Параграф <p> с кратким описанием.
    - Неупорядоченный список <ul> из 2-3 интересных фактов о животном.
* Поэкспериментируйте с разными тегами заголовков (h1-h6) и посмотрите, как меняется их размер.

## Урок №26: HTML: Ссылки, изображения, таблицы. Введение в CSS.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Научиться вставлять ссылки (<a>) на другие страницы или ресурсы.
* Освоить вставку изображений (<img>).
* Изучить создание таблиц (<table>, <tr>, <th>, <td>).
* Познакомить с CSS (Cascading Style Sheets) и его ролью в стилизации веб-страниц.
* Изучить 3 способа подключения CSS: инлайн, внутренний, внешний.
* Освоить базовые CSS-свойства: color, font-size, background-color.

**Основные понятия:**

* Ссылки <a>
* Атрибут href
* Изображения <img>
* Атрибуты src, alt
* Таблицы <table>, <tr>, <th>, <td>
* CSS (Cascading Style Sheets)
* Стилизация, оформление
* Селектор, Свойство, Значение
* Инлайн-стили
* Внутренние стили (<style>)
* Внешние стили (<link rel="stylesheet">)
* Базовые CSS-свойства: color, font-size, background-color, text-align

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое HTML? Зачем нужны теги? Какие основные теги для текста вы знаете?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **HTML: Ссылки, Изображения, Таблицы (35 минут)**
   * **Теоретический материал:**
     + HTML позволяет не только структурировать текст, но и добавлять интерактивные элементы и медиа.
     + **Ссылки (<a> - anchor):** Позволяют переходить между страницами или на внешние ресурсы.
       - Атрибут href (Hypertext REFerence) указывает URL назначения.
       - Пример: <a href="https://www.google.com">Перейти в Google</a>
       - Ссылка на другой HTML-файл в том же проекте: <a href="about.html">О нас</a>
       - Открытие в новой вкладке: <a href="url" target="\_blank">Ссылка</a>
     + **Изображения (<img>):** Вставляют изображения на страницу. Это одиночный тег.
       - Атрибут src (source) указывает путь к файлу изображения.
       - Атрибут alt (alternative text) предоставляет текстовое описание изображения (важно для SEO и доступности).
       - Пример: <img src="my\_image.jpg" alt="Моя фотография">
       - Размеры: width="100" или style="width: 100px;" (пока кратко про стили).
     + **Таблицы (<table>):** Для отображения табличных данных.
       - <table>: Контейнер для всей таблицы.
       - <tr> (table row): Строка таблицы.
       - <th> (table header): Заголовочная ячейка (по умолчанию жирная, центрированная).
       - <td> (table data): Обычная ячейка данных.
       - Пример:

HTML

<table border="1"> <tr>

<th>Имя</th>

<th>Возраст</th>

</tr>

<tr>

<td>Анна</td>

<td>28</td>

</tr>

<tr>

<td>Петр</td>

<td>35</td>

</tr>

</table>

* + **Демонстрация:** Добавить ссылки на другие страницы (https://www.google.com/search?q=google.com, yandex.ru), на внутренний файл (который еще не существует, просто для демонстрации). Вставить изображение. Создать простую таблицу с данными.

1. **Введение в CSS: Стилизация веб-страниц (35 минут)**
   * **Теоретический материал:**
     + HTML дает структуру, но выглядит скучно. Чтобы сделать страницу красивой, использовать цвета, шрифты, отступы, мы используем **CSS (Cascading Style Sheets - Каскадные таблицы стилей)**.
     + CSS описывает, как HTML-элементы должны отображаться на экране.
     + **Аналогия:** "HTML — это скелет дома, а CSS — это его дизайн, цвета стен, мебель, отделка."
     + **Основные понятия в CSS:**
       - **Селектор:** Выбирает HTML-элемент, к которому применяются стили (например, p для всех параграфов, h1 для заголовков).
       - **Свойство:** Что мы хотим изменить (например, color, font-size, background-color).
       - **Значение:** Как мы хотим изменить свойство (например, red, 20px, lightblue).
       - Правило CSS: селектор { свойство: значение; свойство: значение; }
     + **3 способа подключения CSS:**
       1. **Инлайн-стили (Inline Styles):** Стили применяются прямо внутри HTML-тега с помощью атрибута style.
          - Плохо для больших проектов, так как смешивает структуру и стиль.
          - Пример: <p style="color: blue; font-size: 18px;">Синий текст.</p>
       2. **Внутренние стили (Internal Styles):** Стили пишутся внутри тега <style> в секции <head> HTML-документа.
          - Хорошо для одной страницы, но если страниц много, то стили будут повторяться.
          - Пример:

HTML

<head>

<style>

h1 { color: green; text-align: center; }

p { font-family: Arial; }

</style>

</head>

* + - 1. **Внешние стили (External Styles):** Самый рекомендуемый способ! Стили пишутся в отдельном файле .css, который затем подключается к HTML-документу.
         * Позволяет применять одни и те же стили ко многим страницам.
         * HTML: <link rel="stylesheet" href="style.css"> (в <head>)
         * CSS (файл style.css):

CSS

body {

background-color: lightgray;

}

h1 {

color: purple;

}

* + - **Базовые CSS-свойства:**
      * color: Цвет текста.
      * background-color: Цвет фона.
      * font-size: Размер шрифта.
      * text-align: Выравнивание текста (left, center, right).
  + **Демонстрация:**
    - Показать инлайн-стили.
    - Перенести стили во внутренний <style>.
    - Создать style.css и перенести стили туда, подключить.
    - Изменить цвета, размеры шрифтов, выравнивание.

1. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Используя my\_page.html из прошлого урока:
       - Добавьте ссылку на ваш любимый сайт.
       - Вставьте любое изображение.
       - Сделайте заголовок <h1> красным, используя инлайн-стили.
   * **Средний уровень:**
     + **"Профиль пользователя":**
       - Создайте profile.html. Добавьте:
         * Заголовок с именем пользователя.
         * Фотографию пользователя.
         * Пару параграфов с описанием.
         * Небольшую таблицу с "Навыками" (например, "Python", "HTML", "CSS").
       - Используйте внутренние стили (<style>) для:
         * Изменения цвета текста заголовков.
         * Изменения размера шрифта параграфов.
         * Установки фонового цвета для всей страницы.
   * **Сложный уровень:**
     + **"Галерея изображений":**
       - Создайте gallery.html.
       - Вставьте несколько изображений. Каждое изображение должно быть ссылкой на более крупную версию (или на внешний ресурс, например, Википедию по названию объекта на фото).
       - Создайте таблицу, которая будет описывать эти изображения (Название, Автор, Год).
       - Используйте внешний CSS-файл gallery\_styles.css для:
         * Центрирования заголовка страницы.
         * Установки рамки вокруг каждого изображения.
         * Стиль таблицы (цвет фона заголовков, цвет текста).
     + **Проект: "Портфолио (начало)":**
       - Задание: "Создайте два HTML-файла: index.html (главная страница) и projects.html(страница с проектами).
       - index.html должен содержать: заголовок <h1> с вашим именем, краткое описание, ссылку на projects.html.
       - projects.html должен содержать: заголовок <h1> 'Мои проекты', список проектов (пока просто названия).
       - Создайте внешний CSS-файл portfolio.css, который будет применяться к обоим HTML-файлам.
       - В portfolio.css задайте общий цвет фона для body, цвет текста для h1."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: ссылки, изображения, таблицы, CSS (3 способа), базовые свойства.
   * Превью следующего урока: "На следующем занятии мы глубже погрузимся в CSS: изучим селекторы, работу с блочной моделью (отступы, границы, поля) и базовое позиционирование элементов!"

**Материалы/Ресурсы:**

* Презентация с примерами тегов <a>, <img>, <table>.
* Схемы 3-х способов подключения CSS.
* Шпаргалка по базовым CSS-свойствам.

**Домашнее задание:**

* Продолжите работу над animals.html из прошлого урока:
  + Для каждого животного добавьте небольшое изображение.
  + Сделайте название животного ссылкой на его страницу в Википедии (найдите в поиске).
  + Используйте внутренние стили (<style>) для:
    - Установки размера шрифта для параграфов.
    - Изменения цвета заголовков <h2>.
    - Выравнивания текста заголовка <h1> по центру.

## Урок №27: CSS: Селекторы, блочная модель, позиционирование.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Глубже изучить CSS-селекторы: по тегу, классу, ID.
* Понять концепцию блочной модели CSS (margin, border, padding, content).
* Освоить работу с размерами элементов (width, height).
* Изучить основы позиционирования элементов (display: block/inline/inline-block, float, position).

**Основные понятия:**

* CSS-селекторы: по тегу, классу (.), ID (#)
* Блочная модель (Box Model)
* content (содержимое)
* padding (внутренние отступы)
* border (граница)
* margin (внешние отступы)
* width, height
* Свойства display: block, inline, inline-block
* Позиционирование: float, clear
* position: static, relative, absolute (кратко)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Как вставить ссылку и изображение в HTML? Какие элементы используются для таблиц? Какие 3 способа подключения CSS вы знаете и какой лучший?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **CSS Селекторы (30 минут)**
   * **Теоретический материал:**
     + Селекторы — это мощный инструмент для точного указания, к каким HTML-элементам применять стили.
     + **Селектор по тегу (Element Selector):** Применяет стили ко всем элементам определенного типа.
       - Пример: p { color: gray; } (все параграфы будут серыми)
     + **Селектор по классу (Class Selector):** Применяет стили к элементам, у которых есть определенный класс. Классы используются для группировки элементов со схожими стилями. Один элемент может иметь несколько классов.
       - В HTML: <p class="highlight">Важный текст</p>
       - В CSS: .highlight { background-color: yellow; } (точка перед именем класса в CSS)
     + **Селектор по ID (ID Selector):** Применяет стили к элементу с уникальным ID. ID должен быть уникальным на всей странице! Используется для очень специфических элементов.
       - В HTML: <div id="main-header">Заголовок сайта</div>
       - В CSS: #main-header { font-size: 36px; } (решетка перед именем ID в CSS)
     + **Комбинированные селекторы (кратко):**
       - div p: Выбирает параграфы, которые находятся внутри div.
       - h1.title: Выбирает заголовок h1 с классом title.
   * **Демонстрация:**
     + Создать HTML-файл.
     + Создать несколько параграфов. Стилизовать их по тегу.
     + Добавить классы к некоторым параграфам и стилизовать их по классу.
     + Добавить ID к одному элементу и стилизовать его по ID.
     + Показать приоритет: ID > Класс > Тег.
3. **Блочная модель CSS (30 минут)**
   * **Теоретический материал:**
     + Каждый HTML-элемент на веб-странице представляется в виде прямоугольного блока. Это называется **блочной моделью CSS**.
     + Понимание блочной модели критично для правильного расположения и форматирования элементов.
     + **Компоненты блочной модели (снаружи внутрь):**
       1. **margin (внешние отступы):** Пространство ВОКРУГ элемента, отделяющее его от соседних элементов. Прозрачное.
       2. **border (граница):** Линия вокруг содержимого и внутренних отступов.
       3. **padding (внутренние отступы):** Пространство МЕЖДУ содержимым элемента и его границей. Цвет padding совпадает с фоном элемента.
       4. **content (содержимое):** Фактическое содержимое элемента (текст, изображения и т.д.).
     + **Свойства для управления блочной моделью:**
       - margin: 10px; (со всех сторон)
       - margin-top: 5px;, margin-right: 10px; и т.д.
       - padding: 20px 10px; (сверху/снизу, слева/справа)
       - border: 1px solid black; (ширина, тип, цвет)
       - width, height: Задают ширину и высоту содержимого элемента.
     + **box-sizing: border-box; (важное свойство!):** По умолчанию, width и height применяются только к content. padding и border добавляются поверх этого, увеличивая общий размер блока. border-box включает padding и border в заданные width/height, что упрощает расчеты размеров. Рекомендуется использовать для всех элементов.

CSS

\* {

box-sizing: border-box; /\* Применяем ко всем элементам \*/

}

* + **Демонстрация:**
    - Создать div (просто контейнер) с текстом внутри.
    - Применить к нему background-color, border, padding, margin.
    - Показать, как меняются размеры элемента при добавлении padding и border без box-sizing: border-box; и с ним.
    - Использовать "Инструменты разработчика" -> вкладка "Elements" (Элементы) -> "Box Model" (Блочная модель) для визуализации.

1. **Позиционирование элементов: display, float, position (20 минут)**
   * **Теоретический материал:**
     + Как элементы располагаются на странице?
     + **Свойство display:** Определяет, как элемент отображается и взаимодействует с другими элементами.
       - block: Занимает всю доступную ширину, начинает с новой строки. Примеры: div, p, h1.
       - inline: Занимает только необходимую ширину, не начинает с новой строки. Примеры: span, a, img. Нельзя задать width, height, margin-top/bottom.
       - inline-block: Комбинация inline и block. Элементы располагаются в строку, но можно задать width, height, margin, padding. Хорош для меню, кнопок, галерей.
     + **float (обтекание):** Выталкивает элемент из нормального потока документа и прижимает его к одной из сторон родительского контейнера. Текст и другие элементы будут обтекать его. Используется для колонок или размещения изображений с текстом.
       - float: left;, float: right;
       - clear: both; (для сброса обтекания, чтобы следующий элемент не обтекал)
     + **position (кратко):** Более мощный способ позиционирования, позволяющий точно располагать элементы.
       - static (по умолчанию): Элемент находится в нормальном потоке.
       - relative: Элемент остается в нормальном потоке, но можно сдвигать его относительно своей исходной позиции (top, bottom, left, right).
       - absolute: Элемент полностью вырывается из нормального потока и позиционируется относительно ближайшего **позиционированного** родителя (не static). Если такого нет, то относительно body.
   * **Демонстрация:**
     + Показать разницу между block, inline, inline-block на примерах div, span, a.
     + Использовать float для создания двух колонок или размещения изображения слева от текста.
     + Кратко показать position: relative и position: absolute.
2. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Создайте HTML-файл с 3 параграфами.
     + Стилизуйте первый параграф по тегу (например, font-size: 20px;).
     + Добавьте класс error ко второму параграфу и сделайте его текст красным.
     + Добавьте ID info-block к третьему параграфу и дайте ему зеленый фон и белый текст.
     + Для всех параграфов добавьте padding: 10px; и border: 1px solid black;.
   * **Средний уровень:**
     + **"Карточки товаров":**
       - Создайте несколько div-элементов, каждый из которых представляет "карточку товара".
       - Каждая карточка должна содержать заголовок <h2> (название товара), изображение <img> и параграф <p> (цена).
       - Используйте внешний CSS-файл.
       - Стилизуйте карточки так, чтобы у них был фон, рамка, внутренние и внешние отступы.
       - Используйте display: inline-block; для размещения карточек в ряд.
   * **Сложный уровень:**
     + **"Двухколоночный макет":**
       - Создайте HTML-страницу с двумя div-блоками: left-column и right-column.
       - В каждом блоке добавьте какой-либо контент (заголовки, параграфы).
       - Используйте float: left; и float: right; (или просто два float: left;) для создания двухколоночного макета.
       - Убедитесь, что после колонок есть элемент с clear: both; для сброса обтекания.
       - Добавьте padding, border, margin для каждой колонки.
     + **Проект: "Страница статьи":**
       - Задание: "Создайте HTML-файл article.html с заголовком, изображением, несколькими параграфами текста.
       - Используйте различные CSS-селекторы (тег, класс) для стилизации.
       - Примените блочную модель: задайте width для всего блока статьи, margin для центрирования, padding внутри.
       - Позиционируйте изображение float: left; так, чтобы текст обтекал его справа. Убедитесь, что текст не "залезает" на изображение."
3. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: селекторы (тег, класс, ID), блочная модель (содержимое, padding, border, margin), display, float.
   * Превью следующего урока: "На следующем занятии мы начнем изучение основ JavaScript – языка программирования, который делает веб-страницы интерактивными, позволяя им реагировать на действия пользователя!"

**Материалы/Ресурсы:**

* Презентация с примерами селекторов.
* Интерактивная демонстрация блочной модели.
* Примеры display, float, position.

**Домашнее задание:**

* Продолжите работу над profile.html из прошлого урока:
  + Добавьте к тегам p класс description. Стилизуйте его с помощью font-style: italic;.
  + Добавьте ко всей информации о контактах (списку <ul>) класс contact-info. Добавьте этому блоку рамку (border).
  + Добавьте margin и padding к различным элементам, чтобы улучшить их внешний вид и отступы.
  + Используйте box-sizing: border-box; для всех элементов (\* { box-sizing: border-box; }).

## Урок №28: Введение в JavaScript: Основы и интерактивность.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомить с JavaScript как языком для создания интерактивных веб-страниц.
* Изучить 3 способа подключения JavaScript к HTML.
* Освоить основы синтаксиса JavaScript: переменные, типы данных, операторы.
* Научиться работать с консолью разработчика для отладки.
* Понять, как JavaScript взаимодействует с HTML-элементами (базовое DOM-манипулирование).
* Создать простую интерактивную страницу с реакцией на действия пользователя.

**Основные понятия:**

* JavaScript (JS)
* Интерактивность веб-страниц
* Клиентский язык программирования
* Подключение JS: инлайн, внутренний (<script>), внешний (<script src="...">)
* Комментарии в JS
* Переменные (let, const, var - кратко)
* Типы данных: string, number, boolean, null, undefined
* Операторы: арифметические, сравнения, логические
* Консоль разработчика (console.log())
* DOM (Document Object Model) - кратко
* document.getElementById()
* innerHTML, textContent
* Обработчики событий (onclick)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Какие бывают CSS-селекторы? Опишите блочную модель CSS. В чем разница между display: block и display: inline?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Зачем нужен JavaScript? (20 минут)**
   * **Теоретический материал:**
     + HTML дает структуру, CSS — внешний вид. Но веб-страницы до сих пор статичны. Они не реагируют на клики, не обновляют информацию в реальном времени.
     + **JavaScript (JS):** Язык программирования, который делает веб-страницы интерактивными и динамичными.
       - Позволяет реагировать на действия пользователя (клики, ввод текста).
       - Изменять содержимое и стиль HTML-элементов "на лету".
       - Загружать данные без перезагрузки страницы.
       - Выполнять сложные вычисления прямо в браузере пользователя.
     + **Клиентский язык программирования:** JavaScript выполняется в браузере пользователя (на стороне клиента), а не на сервере.
     + **Аналогия:** "Если HTML - это скелет, CSS - одежда, то JavaScript - это нервная система и мозг, которые оживляют страницу, позволяют ей двигаться и реагировать."
   * **Демонстрация:** Открыть любую интерактивную веб-страницу (например, Google Maps, или простой калькулятор онлайн) и показать, как она реагирует без перезагрузки. Объяснить, что за этим стоит JavaScript.
3. **Подключение JavaScript и основы синтаксиса (30 минут)**
   * **Теоретический материал:**
     + Как добавить JavaScript на веб-страницу?
     + **3 способа подключения JS:**
       1. **Инлайн-JS (Inline JavaScript):** JS-код пишется прямо в атрибуте HTML-тега (не рекомендуется для сложной логики).
          - Пример: <button onclick="alert('Привет!')">Нажми меня</button>
       2. **Внутренний JS (Internal JavaScript):** JS-код пишется внутри тега <script> в HTML-документе.
          - Может быть в <head> (но лучше в конце <body>).
          - Пример:

HTML

<body>

<button id="myButton">Нажми меня</button>

<script>

// JS код здесь

document.getElementById('myButton').onclick = function() {

alert('Привет!');

};

</script>

</body>

* + - 1. **Внешний JS (External JavaScript):** Самый рекомендуемый способ! JS-код пишется в отдельном файле .js, который затем подключается к HTML-документу.
         * HTML: <script src="script.js"></script> (в конце <body> перед </body> для лучшей производительности).
         * JS (файл script.js): alert('Привет из внешнего файла!');
    - **Комментарии в JS:**
      * Однострочные: // Это комментарий
      * Многострочные: /\* Это многострочный комментарий \*/
    - **Переменные:** Для хранения данных.
      * let: Современный способ. Переменная может быть переназначена, но не переобъявлена в той же области видимости.
      * const: Для констант. Значение не может быть изменено после присвоения.
      * var: Старый способ (старайтесь избегать в новом коде из-за особенностей области видимости).
      * Пример: let name = "Джон"; const PI = 3.14;
    - **Типы данных (основные):**
      * string (строки): "Hello", 'World'
      * number (числа): 10, 3.14
      * boolean (логические): true, false
      * null: Отсутствие какого-либо значения (специальное значение).
      * undefined: Переменная объявлена, но ей не присвоено значение.
    - **Операторы:** Похожи на Python.
      * Арифметические: +, -, \*, /, %
      * Сравнения: == (нестрогое равно), === (строгое равно), !=, !==, <, >, <=, >=
      * Логические: && (И), || (ИЛИ), ! (НЕ)
    - **Консоль разработчика (console.log()):** Инструмент для отладки. Выводит информацию в консоль браузера (F12 -> вкладка "Console"). Очень полезно!
      * Пример: console.log("Значение переменной:", name);
  + **Демонстрация:**
    - Показать 3 способа подключения JS.
    - Написать простые переменные, выполнить арифметические операции, сравнения.
    - Вывести результаты в console.log().

1. **Взаимодействие с HTML (DOM-манипулирование) (20 минут)**
   * **Теоретический материал:**
     + **DOM (Document Object Model):** Представление HTML-документа в виде древовидной структуры, где каждый HTML-элемент является "узлом". JavaScript может получать доступ к этим узлам, изменять их, добавлять, удалять.
     + **document.getElementById('id\_элемента'):** Самый простой способ получить ссылку на HTML-элемент по его уникальному ID.
     + **Изменение содержимого:**
       - элемент.innerHTML: Позволяет получить или установить HTML-содержимое элемента.
       - элемент.textContent: Позволяет получить или установить только текстовое содержимое элемента (без HTML-тегов).
       - Пример:

HTML

<p id="myParagraph">Исходный текст.</p>

<button onclick="changeText()">Изменить текст</button>

<script>

function changeText() {

const paragraph = document.getElementById('myParagraph');

paragraph.textContent = 'Текст изменен JavaScript!';

// paragraph.innerHTML = '<i>Текст</i> изменен <b>JS</b>!';

}

</script>

* + - **Обработчики событий (onclick):** Функции, которые выполняются при наступлении определенного события (например, клик мышью).
      * Пример: onclick="имя\_функции()" прямо в HTML-теге.
      * Лучший способ (в JS):

JavaScript

const button = document.getElementById('myButton');

button.onclick = function() {

console.log('Кнопка нажата!');

};

// Или: button.addEventListener('click', function() { ... });

* + **Демонстрация:**
    - Создать кнопку и параграф.
    - По нажатию на кнопку менять текст в параграфе.
    - Показать, как изменить цвет элемента при клике.

1. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Создайте HTML-страницу с заголовком <h1> и кнопкой.
     + При клике на кнопку, используйте alert() для вывода сообщения "Кнопка нажата!".
     + Внешний JS-файл app.js.
   * **Средний уровень:**
     + **"Счетчик кликов":**
       - Создайте HTML-страницу с параграфом <span> с ID "count" (начальное значение 0) и кнопкой "Увеличить".
       - При каждом клике на кнопку, JavaScript должен увеличивать число в <span> на 1.
       - Используйте внешние JS-файлы.
     + **"Приветствие по имени":**
       - Создайте HTML-страницу с полем ввода (<input type="text" id="nameInput">) и кнопкой "Поприветствовать".
       - При клике на кнопку, возьмите текст из поля ввода и выведите его в параграфе <p id="greetingMessage"> как "Привет, [Имя]!".
   * **Сложный уровень:**
     + **"Изменение стилей по клику":**
       - Создайте div с каким-либо содержимым.
       - Создайте 3 кнопки: "Красный фон", "Синий фон", "Зеленый фон".
       - При клике на каждую кнопку, меняйте background-color div-а на соответствующий цвет.
       - Дополнительно: добавьте кнопку "Скрыть/Показать", которая будет переключать видимость div-а (используйте element.style.display = 'none'; и element.style.display = 'block';).
     + **Проект: "Простой калькулятор":**
       - Задание: "Создайте HTML-файл с двумя полями ввода чисел и кнопкой 'Сложить'.
       - Создайте параграф для вывода результата.
       - Используйте JavaScript:
         * При клике на кнопку, получите значения из полей ввода.
         * Преобразуйте их в числа (parseInt() или parseFloat()).
         * Сложите числа и выведите результат в параграф."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: JS для интерактивности, подключение, переменные, типы, операторы, консоль, DOM-манипулирование, события.
   * Превью следующего урока: "На следующем занятии мы продолжим изучение JavaScript: освоим условные операторы, циклы, функции и массивы, что позволит нам писать более сложную логику для веб-страниц!"

**Материалы/Ресурсы:**

* Презентация с основами JavaScript.
* Схемы подключения JS.
* Шпаргалка по основным типам данных и операторам JS.
* Примеры DOM-манипулирования.

**Домашнее задание:**

* Продолжите работу над проектом "Портфолио" из прошлого урока. Добавьте на страницу index.htmlкнопку "Сменить тему". При клике на нее, меняйте цвет фона всей страницы (используйте document.body.style.backgroundColor = '...').
* Создайте HTML-страницу с текстом "Добро пожаловать!". Добавьте кнопку "Скрыть". При клике на нее, текст должен исчезать. При следующем клике - появляться снова.

## Урок №29: JavaScript: Условные операторы, циклы, функции, массивы.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Освоить условные операторы (if, else if, else) для выполнения кода по условию.
* Изучить циклические конструкции (for, while) для повторения действий.
* Научиться объявлять и использовать функции для организации кода.
* Познакомиться с массивами как способом хранения коллекций данных.
* Применять полученные знания для создания более сложной интерактивности.

**Основные понятия:**

* Условные операторы: if, else if, else
* Циклы: for, while
* Функции: объявление, вызов, параметры, возвращаемое значение (return)
* Массивы (Arrays): создание, доступ к элементам, длина
* Методы массивов (кратко): push, pop
* Логика программы, Управление потоком

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое JavaScript и зачем он нужен? Какие способы подключения JS вы знаете? Как получить доступ к HTML-элементу по ID? Как изменить его текст?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Условные операторы (if, else if, else) (25 минут)**
   * **Теоретический материал:**
     + Часто нам нужно, чтобы код выполнялся только при определенных условиях (например, "если пользователь авторизован, показать ему личный кабинет", "если возраст больше 18, разрешить доступ").
     + **if:** Выполняет блок кода, если условие истинно.
     + **else:** Выполняет блок кода, если условие в if ложно.
     + **else if:** Проверяет дополнительное условие, если предыдущее if (или else if) было ложно.
     + **Синтаксис:**

JavaScript

let age = 20;

if (age >= 18) {

console.log("Доступ разрешен.");

} else {

console.log("Доступ запрещен.");

}

let score = 75;

if (score >= 90) {

console.log("Отлично!");

} else if (score >= 70) {

console.log("Хорошо.");

} else {

console.log("Удовлетворительно.");

}

* + **Демонстрация:** Создать поле ввода для возраста. По нажатию кнопки проверять возраст и выводить соответствующее сообщение.

1. **Циклы (for, while) (25 минут)**
   * **Теоретический материал:**
     + Циклы позволяют повторять блок кода заданное количество раз или пока выполняется условие.
     + **for цикл:** Используется, когда заранее известно количество итераций.
       - Синтаксис: for (инициализация; условие; обновление) { // код }
       - Пример:

JavaScript

for (let i = 0; i < 5; i++) {

console.log("Итерация:", i); // 0, 1, 2, 3, 4

}

* + - **while цикл:** Используется, когда количество итераций неизвестно, и цикл должен продолжаться, пока условие истинно.
      * Синтаксис: while (условие) { // код }
      * Пример:

JavaScript

let count = 0;

while (count < 3) {

console.log("Счетчик:", count); // 0, 1, 2

count++;

}

* + **Демонстрация:** С помощью цикла for вывести в консоль числа от 1 до 10. С помощью whileсоздать простой "обратный отсчет".

1. **Функции и Массивы (20 минут)**
   * **Теоретический материал:**
     + **Функции:** Блоки кода, которые можно многократно вызывать по имени. Они помогают организовать код, сделать его переиспользуемым и читаемым.
       - Объявление: function имяФункции(параметр1, параметр2) { // код }
       - Вызов: имяФункции(значение1, значение2);
       - return: Возвращает значение из функции.
       - Пример:

JavaScript

function greet(name) {

return "Привет, " + name + "!";

}

let message = greet("Алиса");

console.log(message); // Привет, Алиса!

* + - **Массивы (Arrays):** Упорядоченные коллекции элементов.
      * Создание: let fruits = ["Яблоко", "Банан", "Апельсин"];
      * Доступ к элементам по индексу (начинаются с 0): fruits[0] (Яблоко)
      * Длина массива: fruits.length (3)
      * Добавление/удаление элементов (кратко): push() (добавить в конец), pop() (удалить с конца).
      * Пример:

JavaScript

let numbers = [10, 20, 30, 40];

for (let i = 0; i < numbers.length; i++) {

console.log(numbers[i]);

}

numbers.push(50); // [10, 20, 30, 40, 50]

* + **Демонстрация:** Создать функцию, которая принимает два числа и возвращает их сумму. Вызвать ее и вывести результат. Создать массив строк, вывести все элементы массива с помощью цикла.

1. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Напишите функцию checkNumber(num), которая принимает число. Если число больше 10, выведите "Число больше 10", иначе "Число меньше или равно 10".
     + Создайте массив colors = ["red", "green", "blue"]. Используя цикл for, выведите каждый цвет в консоль.
   * **Средний уровень:**
     + **"Генератор списка":**
       - Создайте HTML-страницу с полем ввода для количества элементов и кнопкой "Создать список".
       - При клике, используя цикл, сгенерируйте и добавьте в <ul> на странице столько <li>элементов, сколько указано в поле ввода (например, "Элемент 1", "Элемент 2" и т.д.).
     + **"Фильтрация массива":**
       - Создайте массив чисел grades = [85, 92, 60, 78, 95, 45].
       - Напишите функцию getPassingGrades(arr), которая принимает массив чисел и возвращает новый массив только тех чисел, которые больше или равны 70. Выведите результат в консоль.
   * **Сложный уровень:**
     + **"Простой To-Do лист":**
       - Создайте HTML-страницу с полем ввода (<input type="text" id="taskInput">), кнопкой "Добавить задачу" и <ul> с ID "taskList".
       - В JavaScript:
         * Создайте пустой массив tasks = [].
         * При клике на кнопку:

Получите текст из поля ввода.

Добавьте его в массив tasks.

Создайте новый <li> элемент, добавьте в него текст задачи и кнопку "Выполнено" (с onclick для удаления задачи из списка).

Добавьте <li> в <ul>.

Очистите поле ввода.

* + - * + *Подсказка: Для удаления используйте parentNode.removeChild() на <li>элементе.*
    - **Проект: "Викторина (логика)":**
      * Задание: "Создайте массив questions, где каждый элемент - это объект (словарь) с полями: questionText (текст вопроса), answer (правильный ответ).
      * Создайте функцию askQuestion(questionIndex), которая:
        + Выводит questionText в HTML-элемент.
        + Запрашивает ответ у пользователя (пока через prompt() или поле ввода).
        + Проверяет ответ и выводит "Правильно!" или "Неправильно!"."

1. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: if/else, for/while, функции, массивы.
   * Превью следующего урока: "На следующем занятии мы погрузимся в более продвинутые концепции JavaScript: объекты, события DOM (click, input), а также научимся менять стили элементов с помощью JS!"

**Материалы/Ресурсы:**

* Презентация с примерами условных операторов и циклов.
* Схемы работы функций.
* Шпаргалка по методам массивов.

**Домашнее задание:**

* Создайте HTML-страницу с кнопкой "Показать счастливое число".
  + При клике на кнопку:
    - Сгенерируйте случайное число от 1 до 10 (используйте Math.random() и Math.floor()).
    - Если число больше 7, выведите в параграф "Ваше счастливое число: [число]! Удача на вашей стороне!".
    - Иначе выведите "Ваше число: [число]. Удачи в следующий раз!".
* Создайте HTML-страницу с полем ввода и кнопкой "Добавить в список".
  + При клике на кнопку, текст из поля ввода должен добавляться в неупорядоченный список <ul> на странице.

## Урок №30: JavaScript: Объекты, события DOM и изменение стилей.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомить с объектами как способом хранения структурированных данных.
* Изучить основные события DOM: click, input, mouseover, mouseout.
* Научиться работать с обработчиками событий с помощью addEventListener().
* Освоить изменение CSS-стилей элементов напрямую через JavaScript.
* Применять знания для создания более динамичных и отзывчивых веб-страниц.

**Основные понятия:**

* Объекты (Objects): свойства, методы
* Доступ к свойствам: точечная нотация, скобочная нотация
* События DOM (DOM Events)
* addEventListener()
* Типы событий: click, input, mouseover, mouseout, keydown
* Объект event
* element.style.propertyName
* Изменение классов: classList.add(), classList.remove(), classList.toggle()

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Как использовать if/else? В чем разница между for и while? Как объявить функцию и передать в нее параметры? Как создать массив и получить доступ к его элементам?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Объекты в JavaScript (25 минут)**
   * **Теоретический материал:**
     + Мы уже знакомы с массивами для хранения списков. Но что, если нам нужно хранить более сложные, структурированные данные, где каждый элемент имеет имя (ключ)?
     + **Объекты:** Коллекции пар "ключ-значение". Похожи на словари в Python. Ключи (свойства) - это строки (или символы), значения - это любые типы данных (даже другие объекты или функции).
     + Создание:

JavaScript

let user = {

name: "Алиса",

age: 30,

isStudent: false,

hobbies: ["чтение", "кодинг"],

address: {

city: "Бишкек",

street: "Пушкина"

},

greet: function() { // Метод объекта (функция внутри объекта)

console.log("Привет, меня зовут " + this.name);

}

};

* + - **Доступ к свойствам:**
      * **Точечная нотация:** user.name (Алиса)
      * **Скобочная нотация:** user["age"] (30) – удобно, если ключ хранится в переменной.
    - **Изменение/добавление свойств:** user.age = 31; user.email = "test@example.com";
    - **Вызов методов:** user.greet();
  + **Демонстрация:** Создать объект product (название, цена, вНаличии). Вывести его свойства. Добавить новое свойство (например, категория). Создать метод, который выводит информацию о продукте.

1. **События DOM и addEventListener() (30 минут)**
   * **Теоретический материал:**
     + Веб-страницы должны реагировать на действия пользователя: клики, ввод текста, наведение мыши и т.д. Эти действия называются **событиями (Events)**.
     + **Обработчик событий (Event Handler):** Функция, которая "слушает" событие и выполняется, когда оно происходит.
     + **addEventListener(eventType, handlerFunction):** Современный и рекомендуемый способ назначать обработчики событий.
       - eventType: Тип события (строка, например 'click', 'input', 'mouseover').
       - handlerFunction: Функция, которая будет вызвана при событии.
       - Пример:

HTML

<button id="myButton">Нажми меня</button>

<script>

const button = document.getElementById('myButton');

button.addEventListener('click', function() {

console.log('Кнопка кликнута!');

});

</script>

* + - **Объект event:** Автоматически передается в функцию обработчика. Содержит информацию о произошедшем событии (например, event.target - элемент, на котором произошло событие, event.key - для событий клавиатуры).
    - **Популярные события:**
      * click: Клик мышью.
      * input: Изменение значения в поле ввода (срабатывает при каждом вводе символа).
      * change: Изменение значения в поле ввода (срабатывает, когда элемент теряет фокус).
      * mouseover: Курсор наведен на элемент.
      * mouseout: Курсор покинул элемент.
      * keydown, keyup: Нажатие/отпускание клавиши.
  + **Демонстрация:**
    - Применить click к кнопке.
    - Применить mouseover и mouseout к div, чтобы менять его фон при наведении.
    - Показать input событие на поле ввода, выводя текст в консоль при каждом нажатии.

1. **Изменение CSS-стилей с помощью JavaScript (20 минут)**
   * **Теоретический материал:**
     + JavaScript может не только менять содержимое HTML, но и его внешний вид.
     + **Прямое изменение стиля:** Доступ к свойству style элемента.
       - элемент.style.propertyName = 'значение';
       - Свойства CSS, содержащие дефисы (например, background-color), в JS пишутся в camelCase (backgroundColor).
       - Пример: myDiv.style.backgroundColor = 'red';
     + **Изменение классов CSS (рекомендуется!):** Более чистый и мощный способ. CSS-правила определяются для классов, а JS просто добавляет/удаляет эти классы у элемента.
       - элемент.classList.add('имя\_класса'): Добавляет класс.
       - элемент.classList.remove('имя\_класса'): Удаляет класс.
       - элемент.classList.toggle('имя\_класса'): Переключает класс (добавляет, если нет; удаляет, если есть).
       - Пример:

HTML

<style>

.red-bg { background-color: red; }

.blue-text { color: blue; }

</style>

<div id="myBox">Привет</div>

<button onclick="changeStyle()">Изменить</button>

<script>

function changeStyle() {

const box = document.getElementById('myBox');

box.classList.add('red-bg');

box.classList.toggle('blue-text');

}

</script>

* + **Демонстрация:**
    - Создать div и кнопку. По нажатию на кнопку менять цвет фона div через element.style.backgroundColor.
    - Создать CSS-класс .highlight. По нажатию кнопки добавлять/удалять этот класс у элемента через classList.toggle().

1. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Создайте HTML-страницу с параграфом и кнопкой.
     + При клике на кнопку, измените цвет текста параграфа на синий, используя element.style.color.
     + Создайте объект book = { title: "Война и мир", author: "Л. Толстой", year: 1869 }. Выведите название книги в консоль.
   * **Средний уровень:**
     + **"Интерактивная кнопка":**
       - Создайте HTML-страницу с кнопкой.
       - При наведении мыши на кнопку (mouseover), ее фон должен становиться желтым.
       - При убирании мыши (mouseout), фон должен возвращаться к исходному (или любому другому) цвету.
       - Используйте addEventListener().
     + **"Карточка пользователя (динамическая)":**
       - Создайте объект userProfile = { name: "Иван", age: 25, city: "Москва" }.
       - Создайте HTML-элементы (<p> или <span>) для отображения имени, возраста и города.
       - Используйте JavaScript для того, чтобы вывести данные из объекта userProfile в соответствующие HTML-элементы при загрузке страницы.
   * **Сложный уровень:**
     + **"Генератор случайных цветов":**
       - Создайте div и кнопку "Сменить цвет".
       - При клике на кнопку, генерируйте случайный шестнадцатеричный цвет (например, #FF00FF) и применяйте его к background-color div-а.
       - *Подсказка: Math.random().toString(16).substring(2, 8) может помочь.*
     + **Проект: "Галерея изображений (интерактивность)":**
       - Задание: "Используйте HTML и CSS из урока 26/27 (галерея изображений).
       - При клике на любое изображение, оно должно увеличиваться (например, его width и height увеличиваются, и появляется рамка другого цвета). При повторном клике - возвращаться к исходному размеру.
       - Используйте classList.toggle() и CSS-классы для изменения размеров и стилей."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: объекты, события DOM, addEventListener(), изменение стилей (прямое и через классы).
   * Превью следующего урока: "На следующем занятии мы изучим более продвинутые аспекты работы с формами в JavaScript, валидацию данных, а также начнем знакомство с AJAX для подгрузки данных без перезагрузки страницы!"

**Материалы/Ресурсы:**

* Презентация с примерами объектов и синтаксиса.
* Список популярных событий DOM.
* Примеры изменения стилей через JS и classList.

**Домашнее задание:**

* Продолжите работу над "Простым To-Do листом" из прошлого урока:
  + При добавлении задачи, она должна быть объектом: { id: ..., text: "...", completed: false }.
  + Добавьте кнопку "Отметить как выполненную". При клике на нее, текст задачи должен перечеркиваться (используйте element.style.textDecoration = 'line-through'; или добавьте/удалите CSS-класс).
  + Добавьте кнопку "Удалить" для каждой задачи.

## Урок №31: JavaScript: Работа с формами и валидация данных. AJAX.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Научиться работать с HTML-формами в JavaScript: получение значений, отправка.
* Освоить основы валидации данных формы на стороне клиента.
* Понять концепцию AJAX (Asynchronous JavaScript and XML) и ее применение.
* Изучить использование XMLHttpRequest (кратко) и fetch API для асинхронных запросов.
* Краткое введение в API (Application Programming Interface).

**Основные понятия:**

* HTML-формы: <form>, <input>, <textarea>, <select>, <button>
* Событие submit
* event.preventDefault()
* Валидация данных (клиентская)
* Регулярные выражения (кратко)
* AJAX (Asynchronous JavaScript and XML)
* Асинхронные запросы
* API (Application Programming Interface)
* XMLHttpRequest (кратко)
* fetch API
* Promise (кратко)
* JSON (для обмена данными с API)

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое объекты в JS? Как к ним получить доступ? Как добавить обработчик события с помощью addEventListener()? Как изменить стиль элемента через JS?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Работа с HTML-формами в JavaScript (30 минут)**
   * **Теоретический материал:**
     + Формы - это основной способ взаимодействия пользователя с веб-сайтом (регистрация, вход, поиск, отправка комментариев).
     + **Основные HTML-элементы формы:**
       - <form>: Контейнер для всех элементов формы. Атрибуты action (куда отправлять) и method (как отправлять: GET или POST).
       - <input type="text">, <input type="password">, <input type="email">, <input type="number">: Различные поля ввода.
       - <textarea>: Многострочное текстовое поле.
       - <select>, <option>: Выпадающий список.
       - <button type="submit"> или <input type="submit">: Кнопка отправки формы.
     + **Получение значений из полей ввода:**
       - После получения ссылки на элемент по ID: element.value
       - Пример: const username = document.getElementById('usernameInput').value;
     + **Событие submit:** Срабатывает при отправке формы.
       - Назначается на сам <form>-элемент.
       - **event.preventDefault():** Очень важно! Предотвращает стандартное поведение браузера (перезагрузку страницы при отправке формы). Мы хотим обрабатывать отправку формы с помощью JavaScript.
       - Пример:

HTML

<form id="myForm">

<input type="text" id="name" placeholder="Имя">

<button type="submit">Отправить</button>

</form>

<script>

const form = document.getElementById('myForm');

form.addEventListener('submit', function(event) {

event.preventDefault(); // Остановить перезагрузку страницы

const name = document.getElementById('name').value;

console.log('Имя:', name);

// Здесь будет логика отправки данных на сервер или дальнейшей обработки

});

</script>

* + **Демонстрация:** Создать форму регистрации (имя, email, пароль). При отправке формы вывести значения полей в консоль, предотвращая перезагрузку.

1. **Валидация данных формы (клиентская) (25 минут)**
   * **Теоретический материал:**
     + Валидация — это проверка введенных пользователем данных на корректность.
     + **Клиентская валидация:** Происходит в браузере ДО отправки данных на сервер. Это улучшает пользовательский опыт, так как дает мгновенную обратную связь.
     + **Простые проверки:**
       - Пустые поля: if (value === '')
       - Длина строки: if (value.length < 5)
       - Числовые значения: isNaN(value) (is Not a Number)
     + **Регулярные выражения (кратко):** Мощный инструмент для проверки сложных шаблонов (например, формат email, телефонного номера).
       - Пример: const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
       - emailRegex.test(email): Возвращает true или false.
     + **Отображение ошибок:** Создавать <span> или <p> элементы рядом с полями ввода для вывода сообщений об ошибках.
   * **Демонстрация:** Добавить к форме регистрации валидацию:
     + Проверка на пустое имя.
     + Проверка, что пароль не короче 6 символов.
     + (По желанию) Базовая проверка формата email.
     + Выводить сообщения об ошибках рядом с полями.
2. **AJAX и fetch API. Введение в API (25 минут)**
   * **Теоретический материал:**
     + **Проблема:** До этого момента, чтобы получить новые данные или отправить их на сервер, страница полностью перезагружалась. Это неэффективно и плохо для пользователя.
     + **AJAX (Asynchronous JavaScript and XML):** Технология, которая позволяет веб-страницам асинхронно (то есть, без перезагрузки страницы) обмениваться данными с сервером.
       - **"Асинхронно":** Запрос отправляется, а JS-код продолжает выполняться. Когда ответ приходит, срабатывает специальная функция (callback).
       - **"XML":** Исторически использовался XML, но сейчас чаще всего передают данные в формате JSON.
     + **API (Application Programming Interface):** Набор правил и протоколов, по которым одно программное обеспечение взаимодействует с другим. Когда мы делаем AJAX-запрос, мы обращаемся к определенному API на сервере.
     + **XMLHttpRequest (XHR):** Старый, но все еще используемый объект для AJAX-запросов.
       - Пример (кратко):

JavaScript

const xhr = new XMLHttpRequest();

xhr.open('GET', 'https://api.example.com/data');

xhr.onload = function() { // когда придет ответ

if (xhr.status === 200) {

console.log(JSON.parse(xhr.responseText));

}

};

xhr.send();

* + - **fetch API (современный способ!):** Более новый, простой и мощный API для сетевых запросов. Использует **Promise**-ы для работы с асинхронностью.
      * Promise: Объект, который представляет собой результат асинхронной операции, который будет доступен в будущем (либо успех, либо ошибка).
      * Пример:

JavaScript

fetch('https://api.example.com/users') // Отправляем запрос

.then(response => response.json()) // Парсим ответ как JSON

.then(data => { // Работаем с данными

console.log(data);

})

.catch(error => { // Обработка ошибок

console.error('Ошибка:', error);

});

* + - * fetch возвращает Promise. .then() выполняется при успехе, .catch() при ошибке.
  + **Демонстрация:** Использовать fetch для получения данных с тестового API (например, JSONPlaceholder: https://jsonplaceholder.typicode.com/todos/1). Вывести полученные данные в консоль.

1. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Создайте форму с одним полем ввода (например, "Цвет") и кнопкой "Отправить".
     + При отправке формы, предотвратите перезагрузку страницы и выведите введенный цвет в консоль.
     + Выполните fetch запрос к https://jsonplaceholder.typicode.com/posts/1 и выведите полученные данные в консоль.
   * **Средний уровень:**
     + **"Форма обратной связи":**
       - Создайте HTML-форму с полями: "Имя", "Email", "Сообщение".
       - При отправке формы:
         * Выполните базовую валидацию: все поля должны быть не пустыми. Email должен содержать @.
         * Если валидация прошла успешно, выведите все данные формы в консоль и покажите сообщение "Сообщение отправлено!" на странице.
         * Если есть ошибки, покажите их пользователю.
     + **"Загрузка пользователей":**
       - Создайте кнопку "Загрузить пользователей" и пустой <ul>.
       - При клике на кнопку, используйте fetch для получения списка пользователей с https://jsonplaceholder.typicode.com/users.
       - Затем, используя цикл, для каждого пользователя создайте <li> элемент с его именем и email, и добавьте его в <ul>.
   * **Сложный уровень:**
     + **"Мини-поиск по API":**
       - Создайте поле ввода (<input type="text" id="searchInput">) и кнопку "Найти пост".
       - При клике на кнопку, возьмите текст из поля ввода (это будет ID поста).
       - Сделайте fetch запрос к https://jsonplaceholder.typicode.com/posts/ + ID поста.
       - Выведите заголовок (title) и тело (body) поста на страницу.
       - Обработайте ошибку, если пост с таким ID не найден (например, выведите сообщение "Пост не найден").
     + **Проект: "Динамическая галерея изображений из API":**
       - Задание: "Используйте fetch для загрузки данных об изображениях из публичного API (например, https://jsonplaceholder.typicode.com/photos - используйте только первые 10-20 для скорости).
       - Для каждого полученного объекта создайте div (карточку) с изображением (<img>) и заголовком (<h2> - title из данных API).
       - Разместите эти карточки на странице, используя CSS (например, display: inline-block;)."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: формы, валидация, AJAX, fetch, API.
   * Превью следующего урока: "На следующем занятии мы начнем знакомиться с более мощными инструментами для веб-разработки: основами сборки проектов, введением в работу с пакетами и библиотеками, и рассмотрим одну из популярных JavaScript-библиотек (например, React или Vue.js - для общего понимания, без глубокого погружения)."

**Материалы/Ресурсы:**

* Презентация с примерами HTML-форм.
* Примеры валидации данных.
* Схемы работы AJAX и fetch.
* Краткий обзор публичных API.

**Домашнее задание:**

* Создайте страницу с формой входа: поля "Логин" и "Пароль", кнопка "Войти".
  + При отправке формы:
    - Валидация: оба поля не должны быть пустыми. Пароль должен быть не менее 8 символов.
    - Если валидация прошла, имитируйте успешный вход, выводя сообщение "Добро пожаловать, [Логин]!" и скрывая форму.
    - Если валидация не прошла, выведите сообщения об ошибках.
* Добавьте на страницу кнопку "Получить случайный факт". При клике на нее, используйте fetch для получения случайного факта с API (например, https://uselessfacts.jsph.pl/random.json?language=en) и выведите его в параграф на странице.

## Урок №32: Введение в сборку проектов, менеджеры пакетов и фреймворки.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять, что такое сборка проектов в веб-разработке и зачем она нужна.
* Познакомить с концепцией менеджеров пакетов (npm/Yarn) и их ролью.
* Изучить, как устанавливать и использовать сторонние библиотеки/пакеты.
* Кратко ознакомиться с концепцией JavaScript-фреймворков (React, Vue.js, Angular) и их назначением.
* Обсудить, как Python может взаимодействовать с веб-технологиями (Flask/Django).

**Основные понятия:**

* Сборка проекта (Build process)
* Менеджер пакетов (Package Manager): npm (Node Package Manager), Yarn
* Пакет (Package), Библиотека (Library), Зависимость (Dependency)
* package.json
* npm install, npm start
* JavaScript-фреймворки (Frameworks): React, Vue.js, Angular
* SPA (Single Page Application) - кратко
* Фронтенд (Frontend), Бэкенд (Backend)
* Python-фреймворки для веба: Flask, Django

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Как получить данные из формы? Зачем нужна клиентская валидация? Что такое AJAX и fetch API? Что такое API?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Зачем нужна сборка проектов и менеджеры пакетов? (30 минут)**
   * **Теоретический материал:**
     + До сих пор мы писали весь HTML, CSS, JS в одном или нескольких простых файлах. Но для больших, современных веб-приложений этого недостаточно.
     + **Проблемы в больших проектах:**
       - Множество JS-файлов, CSS-файлов.
       - Код нужно оптимизировать (сжимать, минифицировать) для быстрой загрузки.
       - Использование сторонних библиотек.
       - Транспиляция (перевод нового JS-кода в старый для совместимости с браузерами).
     + **Сборка проекта (Build Process):** Процесс автоматического преобразования исходного кода проекта (обычно написанного в удобном для разработчика виде) в оптимизированный, готовый к развертыванию код, который понимает браузер. Для этого используются специальные инструменты (бандлеры, таск-раннеры).
     + **Менеджер пакетов (Package Manager):** Инструмент для установки, обновления и управления сторонними библиотеками (пакетами/модулями), которые используются в проекте.
       - **npm (Node Package Manager):** Самый популярный менеджер пакетов для JavaScript. Идет в комплекте с Node.js (среда выполнения JS вне браузера).
       - **Yarn:** Альтернатива npm, часто быстрее и надежнее.
       - **package.json:** Файл в корне проекта, который описывает проект: его название, версию, зависимости (список используемых библиотек и их версий).
       - **Команды (кратко):**
         * npm init: Создает package.json.
         * npm install [имя\_пакета]: Устанавливает пакет в папку node\_modules.
         * npm install: Устанавливает все зависимости, перечисленные в package.json.
         * npm start: Запускает скрипт, определенный в package.json (например, для запуска сервера разработки).
     + **Ключевая идея:** Менеджеры пакетов позволяют использовать готовые решения от других разработчиков, не "изобретая велосипед", и эффективно управлять всеми зависимостями проекта.
   * **Демонстрация:** (Если есть возможность установить Node.js)
     + Показать команду npm init и созданный package.json.
     + Показать npm install для какой-нибудь простой библиотеки (например, lodash). Показать папку node\_modules.
     + Объяснить, что это базовые инструменты для современной веб-разработки.
3. **JavaScript-фреймворки: React, Vue.js, Angular (30 минут)**
   * **Теоретический материал:**
     + По мере усложнения веб-приложений, стало трудно управлять HTML, CSS и JS "вручную".
     + **Фреймворк (Framework):** Это готовый "каркас" или набор инструментов, который предоставляет структуру и правила для разработки приложений. Он диктует, как нужно писать код, чтобы решать общие задачи.
     + **Библиотека (Library):** Это набор готовых функций, которые можно использовать в своем коде. Вы управляете потоком, библиотека - это просто набор инструментов.
     + **Почему фреймворки?**
       - Ускоряют разработку, предоставляя готовые решения для типовых задач (управление состоянием, рендеринг компонентов).
       - Улучшают читаемость и поддерживаемость кода.
       - Позволяют создавать **SPA (Single Page Application)** - одностраничные приложения, которые загружаются один раз, а затем динамически обновляют контент без перезагрузки всей страницы (как Gmail, Facebook).
     + **Популярные JS-фреймворки для Frontend:**
       - **React (by Facebook):** Библиотека для создания пользовательских интерфейсов. Очень популярен. Основан на компонентном подходе.
       - **Vue.js:** Прогрессивный, инкрементально адаптируемый фреймворк. Легко начать изучать.
       - **Angular (by Google):** Комплексный фреймворк для больших корпоративных приложений.
     + **Ключевая идея:** Эти фреймворки позволяют строить сложные, современные веб-приложения, которые ощущаются как "настольные" программы, работающие в браузере.
   * **Демонстрация:** (Без написания кода, просто обзор)
     + Открыть сайты React, Vue, Angular. Показать примеры кода (очень кратко), чтобы студенты увидели, как сильно отличается синтаксис от "чистого" HTML/JS.
     + Объяснить, что это следующий большой шаг в Frontend-разработке.
4. **Взаимодействие Python с веб-технологиями (Backend) (20 минут)**
   * **Теоретический материал:**
     + Мы изучили Python, HTML, CSS, JS. Как это все работает вместе?
     + **Frontend (Фронтенд):** Та часть приложения, которую видит и с которой взаимодействует пользователь (HTML, CSS, JavaScript). Работает в браузере.
     + **Backend (Бэкенд):** Серверная часть приложения, которая обрабатывает логику, работает с базами данных, аутентификацией и т.д. Пользователь ее не видит.
     + **Python для Backend:** Python очень популярен для создания бэкенда веб-приложений. Он служит "мозгом" приложения, который обрабатывает запросы от фронтенда (от браузера), взаимодействует с БД, выполняет вычисления и возвращает данные (чаще всего JSON) фронтенду.
     + **Python-фреймворки для веба:**
       - **Flask:** Легковесный микрофреймворк. Отлично подходит для небольших проектов и API.
       - **Django:** Мощный, "батарейки в комплекте" фреймворк. Подходит для больших, сложных проектов.
     + **Как это работает вместе:**
       1. Пользователь взаимодействует с Frontend (HTML/CSS/JS в браузере).
       2. Frontend (JS с помощью fetch/AJAX) отправляет запросы к Backend (Python-сервер).
       3. Backend (Python) обрабатывает запрос, работает с БД (которую мы изучили), выполняет логику.
       4. Backend возвращает ответ (JSON-данные) Frontend.
       5. Frontend (JS) получает эти данные и динамически обновляет страницу.
   * **Демонстрация:** Кратко показать пример Flask или Django "Hello World" (если есть готовый). Объяснить концепцию API-endpoint-ов (URL, на которые Frontend отправляет запросы).
5. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + (Теоретически) Объясните своими словами, зачем нужен менеджер пакетов npm.
     + (Теоретически) Приведите 2-3 отличия между фреймворком и библиотекой.
   * **Средний уровень:**
     + **"Исследование фреймворков":**
       - Найдите официальные сайты React, Vue.js, Angular.
       - Почитайте их "Getting Started" (Начало работы) и напишите, что вам кажется самым сложным и самым интересным на первый взгляд.
   * **Сложный уровень:**
     + **"Разработка API на Python (Mock)":**
       - Напишите простой Python-скрипт (без Flask/Django, используя только http.server), который по запросу на определенный URL (/api/data) возвращает JSON-объект: {"message": "Привет от Python!"}.
       - В HTML/JS-странице сделайте fetch запрос к этому локальному серверу Python и выведите сообщение на страницу.
       - *Это сложное задание, которое требует знания работы с сокетами или HTTP-серверами на Python, но оно отлично демонстрирует связку.*
     + **Проект: "Концепция клиент-серверного взаимодействия":**
       - Задание: "Опишите, как бы вы реализовали веб-приложение для списка задач (To-Do List), используя разделение на Frontend и Backend.
       - Какие технологии вы бы использовали для Frontend? (HTML, CSS, JS, какой-то фреймворк?)
       - Какие технологии вы бы использовали для Backend? (Python, какой-то фреймворк, какая база данных?)
       - Как Frontend и Backend будут обмениваться данными (какие запросы и ответы)?
       - Нарисуйте простую схему взаимодействия."
6. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: сборка, менеджеры пакетов, фреймворки JS, Frontend/Backend, Python для веба.
   * **Превью следующего этапа (Итоги Модуля 3):** "Мы завершили Модуль 3, где познакомились с основами веб-разработки (HTML, CSS, JS), интерактивностью, работой с формами и даже концепциями AJAX и современных фреймворков. Это дает вам понимание, как работают современные веб-сайты и куда двигаться дальше в веб-разработке. На следующем занятии мы будем двигаться дальше!"

**Материалы/Ресурсы:**

* Презентация с концепциями сборки, npm.
* Схемы клиент-серверного взаимодействия.
* Слайды с логотипами и кратким описанием React, Vue, Angular, Flask, Django.

**Домашнее задание:**

* Найдите и установите Node.js на свой компьютер (если еще не установлен).
* Создайте новый пустой каталог для проекта. Откройте его в терминале/командной строке и выполните npm init -y. Посмотрите на созданный файл package.json.
* Попробуйте установить какую-нибудь очень простую библиотеку через npm (например, npm install moment для работы с датами, если хотите посмотреть, как она работает, но можно просто установить без использования).

## Урок №33: Введение в базы данных: SQL (продолжение). ORM.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Углубить понимание SQL: подзапросы, агрегатные функции с HAVING, оконные функции (введение).
* Познакомить с концепцией ORM (Object-Relational Mapping) и ее преимуществами.
* Рассмотреть примеры работы с БД через ORM на Python (без глубокого погружения в конкретный ORM-фреймворк).
* Понять, как Python-приложения взаимодействуют с базами данных.

**Основные понятия:**

* SQL (продвинутые темы):
  + Подзапросы (Subqueries)
  + HAVING
  + Оконные функции (ROW\_NUMBER(), RANK(), SUM() OVER()) - кратко
* ORM (Object-Relational Mapping)
* Объекты и таблицы
* Преимущества ORM: абстракция, безопасность, читаемость
* SQLAlchemy (пример ORM для Python)
* CRUD-операции через ORM

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое JOIN? Для чего используется GROUP BY? Какие существуют типы данных в SQL?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **SQL: Продвинутые запросы (35 минут)**
   * **Теоретический материал:**
     + Мы уже умеем выбирать, вставлять, обновлять и удалять данные, а также объединять таблицы и группировать результаты. Теперь посмотрим на более сложные запросы.
     + **Подзапросы (Subqueries):** Запрос внутри другого запроса. Результат внутреннего запроса используется как часть внешнего.
       - Используются в SELECT, FROM, WHERE, HAVING.
       - Пример в WHERE: Найти сотрудников, чья зарплата выше средней зарплаты по компании.

SQL

SELECT name, salary

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

* + - * Пример в SELECT: Добавить среднюю зарплату отдела к каждому сотруднику.

SQL

SELECT name, salary,

(SELECT AVG(salary) FROM employees WHERE department\_id = e.department\_id) AS avg\_dept\_salary

FROM employees e;

* + - **HAVING:** Фильтрация групп после GROUP BY. WHERE фильтрует строки ДО группировки, HAVINGфильтрует группы ПОСЛЕ группировки.
      * Пример: Найти отделы, где средняя зарплата больше 50000.

SQL

SELECT department\_id, AVG(salary)

FROM employees

GROUP BY department\_id

HAVING AVG(salary) > 50000;

* + - **Оконные функции (Window Functions) - кратко:** Позволяют выполнять вычисления над набором строк (окном), связанных с текущей строкой, без схлопывания групп, как это делает GROUP BY.
      * ROW\_NUMBER(): Присваивает уникальный номер каждой строке в рамках окна.
      * RANK(): Присваивает ранг каждой строке в рамках окна (могут быть пропуски).
      * SUM() OVER(PARTITION BY column ORDER BY column): Сумма в рамках группы.
      * Пример: Пронумеровать сотрудников в каждом отделе по зарплате.

SQL

SELECT name, department\_id, salary,

ROW\_NUMBER() OVER(PARTITION BY department\_id ORDER BY salary DESC) as rn

FROM employees;

* + **Демонстрация:**
    - Создать таблицу employees (id, name, salary, department\_id).
    - Продемонстрировать подзапросы для поиска сотрудников с зарплатой выше средней.
    - Продемонстрировать HAVING для фильтрации отделов по средней зарплате.
    - (Очень кратко) Показать ROW\_NUMBER() для ранжирования.

1. **Введение в ORM (Object-Relational Mapping) (35 минут)**
   * **Теоретический материал:**
     + **Проблема:** Работать с SQL-запросами из кода на Python (или другом языке) бывает громоздко. Нужно строить строки запросов, парсить результаты.
     + **ORM (Object-Relational Mapping):** Технология, которая позволяет взаимодействовать с базой данных, используя объекты языка программирования, а не напрямую SQL-запросы.
       - **Сопоставление:** Каждая строка таблицы базы данных сопоставляется с объектом в коде, а столбцы таблицы — со свойствами (атрибутами) этого объекта.
       - **Абстракция:** ORM абстрагирует вас от специфики SQL-диалекта конкретной БД (MySQL, PostgreSQL, SQLite).
       - **Преимущества:**
         * **Ускоряет разработку:** Не нужно писать много SQL-кода.
         * **Повышает читаемость:** Код выглядит более "объектно-ориентированным".
         * **Снижает количество ошибок:** ORM берет на себя экранирование параметров, предотвращая SQL-инъекции.
         * **Позволяет легко менять БД:** В теории, можно поменять базу данных с минимальными изменениями в коде.
       - **Недостатки:**
         * Может быть медленнее для очень сложных запросов (иногда все равно приходится писать "сырой" SQL).
         * Добавляет уровень абстракции, который нужно изучить.
     + **Пример ORM для Python: SQLAlchemy (базовый пример):**
       - Модели (классы Python) описывают таблицы.
       - Создание объекта = создание строки.
       - Сохранение объекта = вставка строки.
       - Запросы:

Python

from sqlalchemy import create\_engine, Column, Integer, String

from sqlalchemy.orm import sessionmaker, declarative\_base

# 1. Определение базы данных

DATABASE\_URL = "sqlite:///./test.db"

engine = create\_engine(DATABASE\_URL)

Base = declarative\_base()

# 2. Определение модели (соответствует таблице 'users')

class User(Base):

\_\_tablename\_\_ = "users"

id = Column(Integer, primary\_key=True, index=True)

name = Column(String, index=True)

email = Column(String, unique=True, index=True)

# Создание таблиц в БД

Base.metadata.create\_all(bind=engine)

# 3. Работа с сессией

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

db = SessionLocal()

# Создание (Create)

new\_user = User(name="Алиса", email="alice@example.com")

db.add(new\_user)

db.commit()

db.refresh(new\_user)

print(f"Создан пользователь: {new\_user.name}, ID: {new\_user.id}")

# Чтение (Read)

user\_found = db.query(User).filter(User.name == "Алиса").first()

if user\_found:

print(f"Найден пользователь: {user\_found.email}")

# Обновление (Update)

if user\_found:

user\_found.email = "alice\_new@example.com"

db.commit()

db.refresh(user\_found)

print(f"Обновлен email: {user\_found.email}")

# Удаление (Delete)

# db.delete(user\_found)

# db.commit()

# print("Пользователь удален.")

db.close()

* + **Демонстрация:**
    - Объяснить концепцию ORM на примере User-объекта и таблицы users.
    - (Если есть среда) Показать простейший пример работы с SQLAlchemy (создание модели, добавление записи).

1. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Напишите SQL-запрос, который найдет все товары, цена которых выше средней цены по всем товарам.
     + Объясните своими словами, что такое ORM и зачем оно нужно.
   * **Средний уровень:**
     + **"Запрос с HAVING":**
       - Дана таблица orders (order\_id, customer\_id, amount).
       - Напишите SQL-запрос, который найдет customer\_id тех покупателей, у которых общая сумма заказов (суммы amount) больше 1000.
     + **"Моделирование с ORM":**
       - Представьте, что у вас есть таблица products со столбцами id, name, price, stock.
       - Как бы вы определили Python-класс Product для этой таблицы, используя концепции ORM? (Просто класс с атрибутами, без импортов)
   * **Сложный уровень:**
     + **"SQL с оконными функциями":**
       - Дана таблица sales (sale\_id, product\_name, amount, sale\_date).
       - Напишите SQL-запрос, который для каждой продажи показывает ее сумму и среднюю сумму продаж за тот же месяц (используйте оконную функцию).
     + **Проект: "Базовый CRUD с ORM (имитация)":**
       - Задание: "Напишите Python-скрипт (без реального подключения к БД, просто с классами и списками), который имитирует работу ORM:
         * Создайте класс Book с атрибутами id, title, author.
         * Создайте глобальный список all\_books = [] для хранения 'записей'.
         * Реализуйте функции:

create\_book(title, author): Создает новый объект Book, присваивает ему уникальный id и добавляет в all\_books.

get\_book\_by\_id(book\_id): Возвращает объект Book по id.

update\_book(book\_id, new\_title, new\_author): Обновляет Book по id.

delete\_book(book\_id): Удаляет Book из списка.

* + - * + Протестируйте функции, создавая, читая, обновляя и удаляя книги."

1. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: продвинутый SQL, ORM, взаимодействие Python с БД.
   * Превью следующего урока: "На следующем занятии мы перейдем к тестированию программного обеспечения – ключевому этапу разработки, который позволяет убедиться в корректности и надежности нашего кода!"

**Материалы/Ресурсы:**

* Презентация с примерами подзапросов, HAVING, оконных функций.
* Схемы, объясняющие концепцию ORM.
* Примеры кода на SQLAlchemy (или другом ORM).

**Домашнее задание:**

* Представьте, что вы разрабатываете приложение для управления библиотекой. Вам нужна таблица books(id, title, author, year).
  + Напишите SQL-запрос, который выберет все книги, выпущенные после 2000 года.
  + Напишите SQL-запрос, который посчитает количество книг у каждого автора.
  + Опишите, как вы будете работать с этой таблицей через ORM на Python (не пишите код, просто шаги: определить модель, создать сессию, добавить/прочитать/обновить/удалить запись).

## Урок №34: Основы тестирования программного обеспечения.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять важность тестирования программного обеспечения.
* Изучить основные виды тестирования (модульное, интеграционное, системное, приемочное) и их цели.
* Познакомить с концепцией тестовых случаев и сценариев.
* Освоить написание простых модульных тестов на Python с использованием unittest или pytest (базово).
* Рассмотреть понятие "баг", "отчет о баге".

**Основные понятия:**

* Тестирование программного обеспечения (Software Testing)
* Цели тестирования: поиск ошибок, обеспечение качества, соответствие требованиям
* Баг (Bug), Дефект (Defect)
* Тестовый случай (Test Case), Тестовый сценарий (Test Scenario)
* Виды тестирования:
  + Модульное (Unit Testing)
  + Интеграционное (Integration Testing)
  + Системное (System Testing)
  + Приемочное (Acceptance Testing)
* Автоматизированное тестирование, Ручное тестирование
* Отчет о баге (Bug Report)
* Фреймворки для тестирования на Python: unittest, pytest
* assert утверждения

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Для чего нужен HAVING? Когда используются подзапросы? Что такое ORM и в чем его преимущества?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Что такое тестирование и почему оно важно? (25 минут)**
   * **Теоретический материал:**
     + Мы научились писать код. Но как убедиться, что он работает правильно и делает то, что от него ожидается, без ошибок? Для этого существует **тестирование программного обеспечения**.
     + **Определение:** Процесс проверки программного обеспечения с целью выявления дефектов (багов), оценки его качества и соответствия заданным требованиям.
     + **Почему тестирование важно?**
       - **Поиск ошибок:** Баги приводят к неправильной работе, сбоям, потере данных.
       - **Обеспечение качества:** Уверенность в стабильности и надежности продукта.
       - **Соответствие требованиям:** Продукт делает то, что должен делать.
       - **Снижение рисков:** Избежание финансовых потерь, репутационных рисков.
       - **Улучшение кода:** Тестирование стимулирует написание более чистого и модульного кода.
     + **Баг (Bug) / Дефект (Defect):** Ошибка в коде, которая приводит к неправильному или неожиданному поведению программы.
     + **Тестовый случай (Test Case):** Набор условий и шагов, описывающих, как протестировать определенную функциональность, включая ожидаемый результат.
     + **Тестовый сценарий (Test Scenario):** Набор тестовых случаев, описывающих последовательность действий для тестирования более сложной функциональности или пользовательского пути.
   * **Демонстрация:**
     + Привести примеры известных багов (например, в играх, ПО), которые привели к проблемам.
     + Объяснить на примере: "Если мы делаем калькулятор, тестовый случай будет: ввод '2' + '2', ожидаемый результат '4'. А сценарий: 'проверка всех базовых операций'."
3. **Виды тестирования (30 минут)**
   * **Теоретический материал:**
     + Тестирование делится на разные уровни в зависимости от того, что мы тестируем.
     + **1. Модульное тестирование (Unit Testing):**
       - Тестирование самых маленьких, изолированных частей кода (функций, методов, классов).
       - Цель: Убедиться, что каждый "кирпичик" работает правильно сам по себе.
       - Обычно пишется самими разработчиками.
       - Пример: Тестирование функции add(a, b), чтобы она корректно складывала числа.
     + **2. Интеграционное тестирование (Integration Testing):**
       - Тестирование взаимодействия между несколькими модулями или компонентами.
       - Цель: Убедиться, что модули работают корректно вместе.
       - Пример: Тестирование взаимодействия функции сохранения данных с базой данных.
     + **3. Системное тестирование (System Testing):**
       - Тестирование всей системы как единого целого.
       - Цель: Убедиться, что система соответствует всем функциональным и нефункциональным требованиям.
       - Пример: Тестирование всего веб-сайта от начала до конца (регистрация -> вход -> добавление товара -> оформление заказа).
     + **4. Приемочное тестирование (Acceptance Testing):**
       - Тестирование, проводимое заказчиком или конечными пользователями, чтобы убедиться, что система соответствует их бизнес-требованиям и готова к выпуску.
       - Цель: Подтвердить, что продукт готов к использованию.
     + **Ручное тестирование vs. Автоматизированное тестирование:**
       - **Ручное:** Человек выполняет тестовые случаи вручную. Хорошо для исследования, UI/UX.
       - **Автоматизированное:** Тестовые сценарии пишутся в виде кода и выполняются автоматически. Быстро, повторяемо, хорошо для регрессионного тестирования.
   * **Демонстрация:** На примере простого веб-приложения объяснить, где какой вид тестирования применим.
4. **Модульное тестирование на Python (unittest/pytest) (25 минут)**
   * **Теоретический материал:**
     + Для автоматизированного тестирования на Python есть встроенный модуль unittest и более популярный сторонний фреймворк pytest.
     + **unittest (встроенный):**
       - Классы-тесты наследуются от unittest.TestCase.
       - Методы тестов начинаются с test\_.
       - Утверждения (assert): self.assertEqual(a, b), self.assertTrue(condition).
       - Пример:

Python

# my\_functions.py

def add(a, b):

return a + b

def subtract(a, b):

return a - b

# test\_my\_functions.py

import unittest

from my\_functions import add, subtract

class TestMyFunctions(unittest.TestCase): # Наследуемся от TestCase

def test\_add(self):

self.assertEqual(add(2, 3), 5) # Проверяем, что 2 + 3 = 5

self.assertEqual(add(-1, 1), 0)

self.assertEqual(add(0, 0), 0)

def test\_subtract(self):

self.assertEqual(subtract(5, 2), 3)

self.assertEqual(subtract(10, 10), 0)

self.assertEqual(subtract(2, 5), -3)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main() # Запускаем тесты

* + - **pytest (более простой и популярный):**
      * Устанавливается: pip install pytest.
      * Файлы тестов начинаются с test\_ или заканчиваются на \_test.py.
      * Функции тестов начинаются с test\_.
      * Использует стандартный assert.
      * Пример:

Python

# test\_my\_functions.py

from my\_functions import add, subtract

def test\_add():

assert add(2, 3) == 5

assert add(-1, 1) == 0

def test\_subtract():

assert subtract(5, 2) == 3

assert subtract(10, 10) == 0

* + - * Запуск: pytest в терминале.
    - **Отчет о баге (Bug Report):** Документ, описывающий найденный баг, чтобы разработчики могли его воспроизвести и исправить. Содержит: название, описание, шаги воспроизведения, ожидаемый результат, фактический результат, версию ПО, окружение.
  + **Демонстрация:**
    - Создать простой файл с функцией (например, is\_even(number)).
    - Создать тестовый файл и написать несколько тестов для этой функции, используя pytest.
    - Показать, как тесты проходят, и как они падают, если есть ошибка.

1. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Что такое "баг"?
     + Для функции multiply(a, b), которая умножает два числа, напишите 3 тестовых случая (например, умножение положительных, отрицательных, на ноль).
   * **Средний уровень:**
     + **"Модульный тест для строки":**
       - Создайте Python-функцию reverse\_string(s), которая переворачивает строку (например, "hello" -> "olleh").
       - Напишите модульный тест для этой функции, используя unittest или pytest. Проверьте несколько разных строк (пустая строка, одно слово, фраза).
     + **"Отчет о баге":**
       - Представьте, что вы тестируете веб-сайт и обнаружили, что кнопка "Отправить форму" неактивна, хотя все поля заполнены. Напишите краткий отчет о баге.
   * **Сложный уровень:**
     + **"Тестирование класса":**
       - Создайте Python-класс BankAccount с методами deposit(amount) и withdraw(amount).
       - Напишите модульный тест для этого класса. Проверьте:
         * Корректность пополнения.
         * Корректность снятия.
         * Попытку снять больше, чем есть на счету (должно быть исключение или сообщение об ошибке).
     + **Проект: "Тестирование To-Do листа (логика)":**
       - Задание: "Возьмите логику вашего To-Do листа из предыдущих уроков (функции добавления, удаления, отметки о выполнении задач в массиве).
       - Создайте Python-тесты для этих функций. Например, тест для add\_task должен проверить, что задача добавилась, remove\_task - что удалилась, complete\_task - что статус изменился."
2. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: важность тестирования, виды тестирования, модульные тесты на Python, баг-репорты.
   * Превью следующего урока: "На следующем занятии мы погрузимся в вопросы кибербезопасности – как защитить наши данные и системы от угроз, а также узнаем о ключевых принципах безопасного программирования!"

**Материалы/Ресурсы:**

* Презентация с концепциями тестирования.
* Примеры кода для unittest и pytest.
* Шаблон для баг-репорта.

**Домашнее задание:**

* Создайте Python-файл с функцией is\_palindrome(word), которая возвращает True, если слово является палиндромом ("казак"), и False в противном случае.
* Напишите модульные тесты для этой функции, используя pytest, проверяя как палиндромы, так и не палиндромы, а также пустые строки и строки с пробелами.

## Урок №35: Основы кибербезопасности и безопасное программирование.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Понять базовые угрозы в киберпространстве и их последствия.
* Изучить основные принципы обеспечения кибербезопасности (конфиденциальность, целостность, доступность).
* Познакомиться с распространенными уязвимостями веб-приложений (SQL-инъекции, XSS, CSRF).
* Освоить базовые методы безопасного программирования для предотвращения уязвимостей.
* Понять важность регулярных обновлений и защиты данных.

**Основные понятия:**

* Кибербезопасность (Cybersecurity)
* Угроза, Уязвимость, Риск
* Принципы безопасности: Конфиденциальность, Целостность, Доступность (CIA Triad)
* Распространенные атаки/уязвимости:
  + Фишинг (Phishing)
  + Вредоносное ПО (Malware): Вирусы, Трояны, Вымогатели
  + DDoS-атаки
  + **SQL-инъекция (SQL Injection)**
  + **Межсайтовый скриптинг (XSS - Cross-Site Scripting)**
  + **Межсайтовая подделка запросов (CSRF - Cross-Site Request Forgery)**
* Безопасное программирование (Secure Coding Practices)
  + Валидация ввода (Input Validation)
  + Параметризация запросов (Prepared Statements/Parameterized Queries)
  + Экранирование вывода (Output Encoding)
  + Хеширование паролей
* Двухфакторная аутентификация (2FA)
* Политики безопасности, Обновления ПО

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Для чего нужно тестирование? Какие бывают виды тестирования? Как написать простой модульный тест на Python?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Введение в кибербезопасность: Основные концепции (25 минут)**
   * **Теоретический материал:**
     + С ростом цифровизации, защита информации становится критически важной.
     + **Кибербезопасность:** Практика защиты компьютерных систем и сетей от цифровых атак, несанкционированного доступа, повреждения или кражи данных.
     + **Угроза:** Потенциальная причина инцидента, который может нанести ущерб системе или организации.
     + **Уязвимость:** Слабость в системе, которую можно использовать для реализации угрозы.
     + **Риск:** Вероятность реализации угрозы, умноженная на потенциальный ущерб.
     + **Три столпа безопасности (CIA Triad):**
       - **Конфиденциальность (Confidentiality):** Защита информации от несанкционированного доступа. Только авторизованные пользователи могут видеть данные.
       - **Целостность (Integrity):** Защита информации от несанкционированного изменения или уничтожения. Данные должны быть точными и полными.
       - **Доступность (Availability):** Обеспечение доступа к информации и системам для авторизованных пользователей, когда это необходимо.
     + **Распространенные угрозы:**
       - **Фишинг:** Попытки обманным путем получить конфиденциальную информацию (пароли, номера карт) под видом надежного источника.
       - **Вредоносное ПО (Malware):** Вирусы, трояны, программы-вымогатели (ransomware) – программы, разработанные для нанесения ущерба или получения несанкционированного доступа.
       - **DDoS-атаки (Distributed Denial of Service):** Затопление системы трафиком, чтобы сделать ее недоступной для легитимных пользователей.
   * **Демонстрация:**
     + Примеры фишинговых писем или сайтов.
     + Обсуждение последствий утечек данных (на примере известных компаний).
3. **Распространенные уязвимости веб-приложений (35 минут)**
   * **Теоретический материал:**
     + Наши веб-приложения могут быть уязвимы, если мы не знаем о типовых атаках.
     + **1. SQL-инъекция (SQL Injection):**
       - Атака, при которой злоумышленник вводит вредоносный SQL-код в поля ввода веб-формы (например, логин/пароль), чтобы манипулировать запросами к базе данных.
       - Пример уязвимого кода: SELECT \* FROM users WHERE username = ' + input\_username + ' AND password = ' + input\_password + '
       - Если input\_username = ' OR '1'='1 и input\_password = ' OR '1'='1, то запрос станет: SELECT \* FROM users WHERE username = '' OR '1'='1' AND password = '' OR '1'='1' – и вход будет выполнен без правильного пароля.
       - Последствия: Кража, изменение, удаление данных, обход аутентификации.
     + **2. Межсайтовый скриптинг (XSS - Cross-Site Scripting):**
       - Атака, при которой злоумышленник внедряет вредоносный JavaScript-код на веб-страницу. Этот код выполняется в браузере других пользователей, посещающих эту страницу.
       - Часто происходит через поля ввода (комментарии, поисковые запросы), если они не очищаются перед отображением.
       - Пример: Пользователь вводит в поле комментария <script>alert('Вы взломаны!');</script>. Если сайт просто отобразит этот текст, скрипт выполнится.
       - Последствия: Кража сессионных куки (что может привести к угону сессии), перенаправление на вредоносные сайты, фишинг.
     + **3. Межсайтовая подделка запросов (CSRF - Cross-Site Request Forgery):**
       - Атака, при которой злоумышленник заставляет пользователя выполнить нежелательное действие на доверенном веб-сайте, на котором пользователь уже авторизован.
       - Пример: Злоумышленник отправляет пользователю ссылку с изображением (<img src="https://bank.com/transfer?to=hacker&amount=1000">). Если пользователь авторизован в банке, его браузер отправит запрос на перевод, даже если пользователь ничего не нажимал.
       - Последствия: Несанкционированные финансовые операции, изменение паролей, публикация контента.
   * **Демонстрация:** (Без реального взлома)
     + Показать пример SQL-инъекции на доске или в псевдокоде.
     + Показать пример XSS в HTML-коде.
     + Кратко объяснить принцип CSRF.
4. **Безопасное программирование (20 минут)**
   * **Теоретический материал:**
     + Как защититься от вышеописанных атак?
     + **1. Валидация ввода (Input Validation):**
       - ВСЕГДА проверяйте и очищайте (санитизируйте) любые данные, которые приходят от пользователя.
       - Проверяйте тип, формат, длину, диапазон значений. Удаляйте или экранируйте специальные символы.
     + **2. Параметризация запросов (Prepared Statements / Parameterized Queries):**
       - **Защита от SQL-инъекций.** Используйте специальные функции БД или ORM, которые отделяют данные от SQL-кода.
       - Данные передаются как параметры, а не как часть строки запроса, и БД интерпретирует их как значения, а не как код.
       - Пример (Python/SQLAlchemy): session.query(User).filter(User.name == username).first() - здесь username передается как параметр.
     + **3. Экранирование вывода (Output Encoding):**
       - **Защита от XSS.** Перед тем как отобразить любые пользовательские данные на веб-странице, убедитесь, что все специальные символы (например, <, >, &) преобразованы в безопасные сущности (&lt;, &gt;, &amp;).
       - Фреймворки (Django, Flask, React) обычно делают это автоматически в шаблонах.
     + **4. Хеширование паролей:**
       - **НИКОГДА не храните пароли в открытом виде!** Храните только хеши паролей (одностороннее преобразование).
       - Используйте сильные алгоритмы хеширования с "солью" (например, bcrypt, scrypt, Argon2).
       - Пример: hash\_password('my\_password')
     + **5. Использование HTTPS:** Шифрование трафика между браузером и сервером.
     + **6. Двухфакторная аутентификация (2FA):** Дополнительный уровень безопасности (пароль + код из СМС/приложения).
     + **7. Регулярные обновления ПО:** Обновляйте операционные системы, библиотеки, фреймворки, чтобы получить исправления безопасности.
   * **Демонстрация:**
     + Показать пример параметризованного запроса в Python (псевдокод).
     + Показать, как innerHTML может быть опасен, а textContent безопасен для XSS.
     + Объяснить принцип хеширования паролей.
5. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Объясните, чем отличается фишинг от вредоносного ПО.
     + Какие три столпа кибербезопасности вы запомнили?
   * **Средний уровень:**
     + **"Защита от SQL-инъекции":**
       - У вас есть функция на Python, которая строит SQL-запрос для поиска пользователя: sql = f"SELECT \* FROM users WHERE name = '{username}'"
       - Как вы можете сделать этот запрос безопасным от SQL-инъекций, используя концепцию параметризации?
     + **"Защита от XSS":**
       - Пользователь вводит текст в поле commentInput. Вы получаете его через document.getElementById('commentInput').value. Вы хотите отобразить этот текст в <div id="commentDisplay">.
       - Какой способ (innerHTML или textContent) безопаснее использовать для commentDisplay.innerHTML = userInput и почему?
   * **Сложный уровень:**
     + **"Парольная политика":**
       - Напишите JavaScript-функцию validatePassword(password), которая возвращает True, если пароль соответствует следующим требованиям:
         * Минимум 8 символов.
         * Содержит хотя бы одну заглавную букву.
         * Содержит хотя бы одну цифру.
       - Используйте регулярные выражения.
     + **Проект: "Анализ уязвимостей в собственном коде":**
       - Задание: "Возьмите любой из ваших предыдущих проектов (например, To-Do лист с сохранением в JS-массиве).
       - Проанализируйте, где в этом проекте могли бы возникнуть уязвимости (SQL-инъекции, XSS, CSRF, если бы там был бэкенд или работа с БД).
       - Предложите, какие меры безопасности можно было бы применить для защиты от этих уязвимостей."
6. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: угрозы, принципы безопасности, SQL-инъекции, XSS, CSRF, методы защиты (валидация, параметризация, хеширование, HTTPS, 2FA).
   * Превью следующего урока: "На следующем занятии мы рассмотрим основы управления проектами и командной работы в IT, а также инструменты для совместной разработки, такие как Git и GitHub. Это последний урок Модуля 3!"

**Материалы/Ресурсы:**

* Презентация с примерами атак и методов защиты.
* Список OWASP Top 10 (краткий обзор).
* Примеры кода для безопасного программирования.

**Домашнее задание:**

* Найдите в Интернете информацию о том, что такое "соль" (salt) при хешировании паролей, и объясните, зачем она нужна.
* Подумайте, какие меры безопасности вы применяете в своей повседневной жизни для защиты своих данных в Интернете (например, для почты, соцсетей).

## Урок №36: Управление проектами и командная работа в IT. Git и GitHub.

**Продолжительность:** 1 час 30 минут (90 минут)

**Цели занятия:**

* Познакомить с основными методологиями управления IT-проектами (Agile, Scrum - кратко).
* Изучить роли в IT-команде (разработчик, тестировщик, менеджер проекта).
* Освоить базовые принципы системы контроля версий Git.
* Научиться работать с GitHub для совместной разработки и хранения кода.
* Провести практическое занятие по Git: инициализация репозитория, коммиты, ветки, слияние.

**Основные понятия:**

* Управление проектами в IT
* Методологии разработки: Agile, Scrum (кратко)
* Роли в IT-команде: разработчик, QA (тестировщик), Project Manager (PM), Business Analyst (BA), DevOps
* Система контроля версий (Version Control System - VCS)
* Git (распределенная VCS)
* Репозиторий (Repository): локальный, удаленный
* GitHub, GitLab, Bitbucket (хостинги Git)
* Базовые команды Git: git init, git add, git commit, git status, git log
* Ветки (Branches), Слияние (Merge)
* git clone, git push, git pull

**План занятия:**

1. **Повторение (10 минут)**
   * Краткий опрос: Что такое SQL-инъекция и XSS? Как от них защититься? Зачем хешировать пароли?
   * Проверка домашнего задания, обсуждение возникших вопросов.
2. **Управление проектами и роли в IT-команде (30 минут)**
   * **Теоретический материал:**
     + Программирование – это часто командный вид спорта. Нужны инструменты и процессы для эффективной совместной работы.
     + **Управление проектами в IT:** Организация и руководство процессом разработки программного обеспечения от начала до конца.
     + **Методологии разработки (кратко):**
       - **Водопадная (Waterfall):** Линейная, последовательная модель. Каждый этап завершается до начала следующего. Менее гибкая.
       - **Гибкие (Agile):** Итеративный подход. Короткие циклы (спринты), частые поставки, адаптация к изменениям.
       - **Scrum:** Самая популярная Agile-методология. Ежедневные стендапы, спринты (2-4 недели), роли (Scrum Master, Product Owner, Development Team).
     + **Основные роли в IT-команде:**
       - **Разработчик (Developer):** Пишет код, реализует функциональность.
       - **Тестировщик (QA Engineer / Quality Assurance):** Проверяет качество продукта, ищет баги, пишет тесты.
       - **Менеджер проекта (Project Manager - PM):** Управляет проектом, следит за сроками, бюджетом, коммуникацией, рисками.
       - **Бизнес-аналитик (Business Analyst - BA):** Собирает требования от заказчика и переводит их в технические задачи для команды.
       - **UX/UI Дизайнер:** Отвечает за пользовательский опыт (UX) и интерфейс (UI).
       - **DevOps Инженер:** Автоматизирует процессы разработки, тестирования, развертывания и эксплуатации.
   * **Демонстрация:**
     + Кратко объяснить Scrum-доску (канбан-доску) с задачами "To Do", "In Progress", "Done".
     + Обсудить, как различные роли взаимодействуют в процессе разработки.
3. **Система контроля версий Git: Основы (35 минут)**
   * **Теоретический материал:**
     + Когда над одним кодом работают несколько человек, нужна система, которая отслеживает изменения, позволяет их объединять и возвращаться к предыдущим версиям.
     + **Система контроля версий (Version Control System - VCS):** Инструмент для отслеживания изменений в файлах (обычно в коде) с течением времени. Позволяет:
       - Возвращаться к любой предыдущей версии.
       - Видеть, кто, когда и что изменил.
       - Объединять изменения от разных разработчиков.
     + **Git:** Самая популярная **распределенная VCS**. Каждому разработчику доступна полная история проекта локально.
     + **Репозиторий (Repository):** Место, где хранится весь код проекта и вся история изменений.
       - **Локальный репозиторий:** На вашем компьютере.
       - **Удаленный репозиторий:** На сервере (например, GitHub).
     + **GitHub:** Самый популярный веб-сервис для хостинга Git-репозиториев. Предоставляет удобный веб-интерфейс, инструменты для совместной работы (Pull Requests), отслеживания задач.
     + **Базовый рабочий процесс Git:**
       1. **Рабочий каталог (Working Directory):** Ваши файлы.
       2. **Индексация (Staging Area / Index):** Место, куда вы добавляете изменения, которые хотите включить в следующий коммит (git add).
       3. **Локальный репозиторий:** История коммитов на вашем компьютере (git commit).
       4. **Удаленный репозиторий:** Копия репозитория на сервере (git push, git pull).
     + **Основные команды Git:**
       - git init: Инициализирует новый Git-репозиторий в текущей папке.
       - git status: Показывает состояние рабочего каталога (измененные/новые файлы).
       - git add <file> / git add .: Добавляет файлы в индексацию.
       - git commit -m "Сообщение": Сохраняет индексированные изменения в локальный репозиторий с сообщением.
       - git log: Показывает историю коммитов.
       - git clone <url>: Копирует удаленный репозиторий на ваш компьютер.
       - git push: Отправляет ваши локальные коммиты в удаленный репозиторий.
       - git pull: Получает изменения из удаленного репозитория и объединяет их с вашими локальными изменениями.
     + **Ветки (Branches):** Позволяют разработчикам работать над разными функциями параллельно, не мешая друг другу. Каждая ветка — это отдельная линия разработки.
       - git branch <name>: Создать ветку.
       - git checkout <name>: Переключиться на ветку.
       - git merge <name>: Объединить изменения из другой ветки в текущую.
   * **Демонстрация (Практическое занятие):**
     + Создать пустой каталог.
     + git init.
     + Создать файл index.html, git add ., git commit.
     + Изменить файл, git status, git add ., git commit.
     + git log.
     + Создать ветку feature-x, переключиться на нее.
     + Внести изменения, закомитить.
     + Переключиться на main, внести изменения, закомитить.
     + git merge feature-x (показать простой merge).
4. **GitHub для совместной работы (20 минут)**
   * **Теоретический материал:**
     + GitHub — это больше, чем просто хостинг Git. Это платформа для совместной работы.
     + **Основные возможности GitHub:**
       - **Хостинг репозиториев:** Хранение кода.
       - **Issues (Задачи):** Система отслеживания задач, багов, предложений.
       - **Pull Requests (PRs):** Механизм для предложения изменений в коде. Разработчик создает ветку, делает изменения, отправляет PR, и другие могут его рецензировать, обсуждать и утверждать перед слиянием в основную ветку.
       - **Code Review:** Просмотр кода другими разработчиками.
       - **Wiki, Projects:** Инструменты для документации и планирования.
     + **Процесс совместной работы (типичный):**
       1. git clone репозитория.
       2. Создать новую ветку для своей задачи (git checkout -b feature-task-name).
       3. Работать, делать коммиты.
       4. git push ветку в GitHub.
       5. Создать Pull Request.
       6. Получить Code Review, внести исправления.
       7. PR объединяется (Merge) в main (или develop) ветку.
   * **Демонстрация:**
     + Создать репозиторий на GitHub.
     + git remote add origin <url>, git push -u origin main.
     + Показать веб-интерфейс GitHub: вкладки Code, Issues, Pull Requests.
     + Кратко объяснить идею Pull Request.
5. **Практические задания (5 минут)**
   * **Легкий уровень:**
     + Объясните, зачем нужен Git.
     + Какие три основные команды Git вы запомнили?
   * **Средний уровень:**
     + **"Локальный Git-проект":**
       - Создайте пустой каталог. Инициализируйте его как Git-репозиторий.
       - Создайте два файла: index.html и style.css.
       - Добавьте их в индексацию и сделайте первый коммит с сообщением "Initial commit".
       - Измените index.html, сделайте еще один коммит с сообщением "Updated index.html".
       - Посмотрите историю коммитов.
     + **"Ветка и слияние":**
       - В вашем локальном репозитории создайте новую ветку add-footer.
       - Переключитесь на эту ветку.
       - В index.html добавьте простой футер (<footer>).
       - Закоммитьте изменения в ветке add-footer.
       - Переключитесь обратно на main и объедините ветку add-footer в main.
   * **Сложный уровень:**
     + **"GitHub Collaboration Simulation":**
       - Зарегистрируйтесь на GitHub (если еще нет).
       - Создайте новый публичный репозиторий на GitHub.
       - Клонируйте его на свой компьютер.
       - Добавьте в него несколько файлов (например, HTML, CSS, JS). Закоммитьте и отправьте (push) их в GitHub.
       - Создайте новую ветку, сделайте в ней изменения, отправьте ее на GitHub.
       - Создайте Pull Request на GitHub и попробуйте "самостоятельно" его принять и объединить.
     + **Проект: "Создание простого веб-сайта с использованием Git и GitHub":**
       - Задание: "Создайте простой одностраничный сайт-визитку (HTML, CSS, JS), используя все знания из Модуля 3.
       - Используйте Git для управления версиями проекта:
         * Инициализируйте репозиторий.
         * Делайте коммиты по мере добавления функциональности (например, 'Add basic HTML structure', 'Add styling', 'Add JS interactivity').
         * Создайте ветку для новой фичи (например, 'add-contact-form'), реализуйте ее, а затем объедините в основную ветку.
       - Загрузите весь проект на GitHub."
6. **Обсуждение и вопросы (5 минут)**
   * Ответы на вопросы.
   * Краткий обзор: методологии, роли, Git, GitHub, базовые команды.
   * **Итоги Модуля 3:** "Поздравляю! Вы завершили Модуль 3, где мы глубоко погрузились в мир веб-разработки и командной работы. Вы получили фундаментальные знания по Frontend-разработке (HTML, CSS, JavaScript), поняли концепции Backend, баз данных, тестирования и кибербезопасности, а также освоили ключевой инструмент для разработчиков – Git и GitHub. Это огромный шаг вперед в вашем IT-путешествии!"