

**МУНИЦИПАЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ
ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
«ДВОРЕЦ ДЕТСКОГО И ЮНОШЕСКОГО ТВОРЧЕСТВА»
ГОРОДСКОГО ОКРУГА ГОРОД ОКТЯБРЬСКИЙ
РЕСПУБЛИКА БАШКОРТОСТАН**

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К
ДОПОЛНИТЕЛЬНОЙ ОБЩЕОБРАЗОВАТЕЛЬНОЙ
ОБЩЕРАЗВИВАЮЩЕЙ ПРОГРАММЕ
ТЕХНИЧЕСКОЙ НАПРАВЛЕННОСТИ
«ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ.
ИНДИВИДУАЛЬНАЯ»**

Составитель: педагог ДО

Зинатуллин К.Р

тел.: +7(996) 256 34 19

Qukeq@mail.ru

г. Октябрьский - 2025г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Данные методические рекомендации содержат общие указания по выполнению практических работ по дополнительной общеобразовательной программе «Прикладное программирование. Индивидуальная» (далее - Программа), реализуемой в МБУ ДО «ДДиЮТ» ГО г. Октябрьский Республики Башкортостан.

Целью изучения является формирование навыков практического программирования с использованием Python и сопутствующих технологий для разработки приложений, работы с базами данных и создания пользовательских интерфейсов.

В результате освоения программы учащийся должен **знать:**

- принципы техники безопасности при работе с программным обеспечением;
- основы работы с базами данных и SQL;
- методы создания графических интерфейсов на Python с использованием tkinter и PyQt6;
- базовые подходы веб-разработки с использованием фреймворков Python.

Уметь:

- разрабатывать приложения с использованием структур и элементов языка Python;
- создавать и поддерживать базы данных с использованием SQL и SQLite3;
- проектировать и реализовывать графические интерфейсы;
- разрабатывать веб-приложения.

Программа опирается на базовые знания программирования и является основой для дальнейшего углубленного изучения прикладного программирования.

ВВЕДЕНИЕ

Учащийся должен иметь представление:

о роли и месте знаний по программированию в современном мире

о перспективах развития IT-сферы и программирования

о возможностях языка Python и его применении в различных областях

Цели и задачи: Общее ознакомление с разделами программы и методами их изучения. Краткие исторические сведения о развитии программирования и языка Python. Общие сведения об основных парадигмах программирования. Роль стандартов кодирования в обеспечении качества программного обеспечения.

Ознакомление обучающихся с необходимыми для занятий:

- программным обеспечением (Python, PyCharm, Wing)
- учебными материалами и документацией
- инструментами разработки
- системами контроля версий
- базами данных и ORM

Методические рекомендации

В современной IT-сфере программирование является ключевым навыком для создания программных продуктов различной сложности. Программа направлена на формирование практических навыков разработки через последовательное освоение следующих разделов:

Раздел 1. Основы программирования - знакомит с базовыми понятиями программирования, синтаксисом Python и основами работы со структурами данных.

Раздел 2. Работа с базами данных - включает изучение SQL, работу с SQLite3 и ORM SQLAlchemy для взаимодействия с базами данных.

Раздел 3. Создание графических интерфейсов - обучает использованию библиотеки tkinter и PyQt6 для разработки пользовательских интерфейсов.

Раздел 4. Веб-разработка - охватывает основы создания веб-приложений с использованием фреймворков Python.

PEP8 - набор рекомендаций по написанию кода Python, обеспечивающий:

- единообразие кода
- лучшую читаемость
- простоту поддержки

Основные правила PEP8 включают:

- отступы в 4 пробела
- ограничение длины строки 79 символами
- пустые строки для разделения логических блоков
- именование переменных в нижнем регистре с подчеркиванием

Вопросы для самоконтроля:

- Какие правила устанавливает PEP8?
- Что входит в основные разделы программы обучения?
- На сколько основных разделов структурирована программа?

Раздел 1. ОСНОВЫ ПРОГРАММИРОВАНИЯ

Тема 1.1 Подготовка рабочей среды и оформление кода

Обучающийся должен:

***знать:** Правила оформления кода согласно PEP8; Структуру проектной документации; Принципы организации рабочего пространства в IDE; Базовые элементы синтаксиса Python.*

***уметь:** настраивать рабочую среду в PyCharm/Wing; Применять стандарты PEP8 при написании кода; Создавать структуру проекта; Документировать код.*

Методические рекомендации

Именованние:

- переменных - snake_case
- констант - UPPER_CASE
- классов - CamelCase

Организация проекта:

Этапы проектной работы:

- Планирование структуры
- Создание базовой архитектуры
- Разработка функционала
- Тестирование
- Документирование
- Защита проекта

```
1 project/
2 |
3 |— main.py      # Главный файл программы
4 |— modules/     # Директория для модулей
5 |   |— __init__.py
6 |   |— module1.py
7 |— data/        # Данные проекта
8 |— docs/        # Документация
```

Рисунок 1 –Базовая архитектура проекта.

Документация кода:

Правила документирования:

- Описание назначения функции
- Аннотации типов данных
- Описание параметров
- Описание возвращаемого значения
- Использование однострочных и многострочных комментариев

```
1 v def calculate_sum(a, b):
2     """
3     Складывает два числа
4
5     :param a: Первое число
6     :param b: Второе число
7     :return: Сумма чисел
8     """
9     return a + b
```

Рисунок 2 – Пример документации функции.

Настройка IDE:

- Установка интерпретатора Python
- Настройка PEP8 линтера
- Создание виртуального окружения
- Конфигурация автосохранения

Управление проектом:

Инструменты:

- Git для контроля версий
- PyCharm/Wing как IDE
- pip для управления зависимостями

Процессы:

- Еженедельные проверки прогресса
- Промежуточные презентации
- Итоговая защита проекта

Вопросы для самоконтроля:

- Какие основные правила содержит PEP8?
- Как организовать структуру проекта?
- Какие есть варианты документирования кода?
- Как настроить рабочую среду в IDE?
- Какие элементы включает pythonic code style?

1.2 Работа с API и сторонними библиотеками

Обучающийся должен:

знать:

- *принципы работы с API (Application Programming Interface)*
- *основные популярные библиотеки Python (requests, pandas, numpy)*
- *правила интеграции сторонних API*

уметь:

- *создавать запросы к API*
- *обработывать полученные данные*
- *интегрировать сторонние библиотеки*
- *применять библиотеки для решения практических задач*

Методические рекомендации

Структура обучения:

- Начинать с простых примеров использования API
- Постепенно усложнять задачи интеграции
- Разбирать реальные кейсы применения библиотек
- Уделять внимание обработке ошибок
- Практиковать работу с документацией

Основные термины:

API (Application Programming Interface) - Интерфейс программирования приложений, набор готовых классов, функций и методов для взаимодействия с внешними системами или компонентами.

GET - HTTP-метод для получения данных с сервера без их изменения. Используется для чтения информации.

POST - HTTP-метод для отправки данных на сервер для создания нового ресурса или обновления существующего.

PUT - HTTP-метод для полной замены существующего ресурса на сервере новыми данными.

DELETE - HTTP-метод для удаления указанного ресурса на сервере.

PATCH - HTTP-метод для частичного обновления существующего ресурса, изменяет только определенные поля.

CRUD (Create, Read, Update, Delete) - Базовые операции с данными: создание, чтение, обновление и удаление.

Endpoint - Конечная точка API, конкретный URL, который обрабатывает определенный запрос.

HTTP Status Code - Трехзначный код ответа сервера, указывающий на результат обработки запроса (200 - успех, 404 - не найдено и т.д.)

Authentication - Механизмы проверки подлинности пользователя при доступе к API (ключи, токены, OAuth).

Rate Limiting - Ограничение количества запросов к API за определенный период времени для предотвращения злоупотреблений.

Pagination - Разделение больших объемов данных на отдельные страницы для эффективной обработки.

Payload - Данные, отправляемые в теле HTTP-запроса или получаемые в ответе.

Header - Метаданные HTTP-запроса/ответа, содержащие служебную информацию о передаваемых данных.

Serialization - Преобразование сложных структур данных в формат, подходящий для передачи по сети (JSON, XML).

Error Handling - Система обработки ошибок API, включающая коды состояния и сообщения об ошибках.

SDK (Software Development Kit) - Набор инструментов для упрощения работы с API через готовые библиотеки.

Webhook - Механизм обратного вызова API, когда сервер отправляет данные клиенту при определенных событиях.

Вопросы для самоконтроля:

- Что означает аббревиатура API и какую роль она играет в программировании?
- Какие основные типы HTTP-запросов используются при работе с API?
- В чем заключается процесс аутентификации при взаимодействии с API?
- Какие популярные библиотеки Python применяются для работы с API?
- Что такое endpoint в контексте API и для чего он используется?

Раздел 2. РАБОТА С БАЗАМИ ДАННЫХ

2.1 Язык запросов SQL

Обучающийся должен:

Знать:

- Основные принципы реляционных баз данных
- Синтаксис и семантику языка SQL
- Типы данных и структуры в реляционных БД
- Методы нормализации баз данных

Уметь:

- Создавать и модифицировать таблицы
- Формулировать SQL-запросы различной сложности
- Выполнять операции CRUD (Create, Read, Update, Delete)
- Оптимизировать запросы и индексацию

Методические рекомендации

Основные понятия реляционных БД:

Реляционная модель - способ организации данных в виде связанных между собой таблиц.

Первичный ключ (Primary Key) - уникальный идентификатор записи в таблице.

Внешний ключ (Foreign Key) - поле, ссылающееся на первичный ключ другой таблицы.

Базовые команды SQL:

CREATE TABLE - создание новой таблицы

Пример:

CREATE TABLE users (

id INT PRIMARY KEY,
name VARCHAR(50),
age INT);

INSERT - добавление данных

Пример:

```
INSERT INTO users (id, name, age)  
VALUES (1, 'John', 25);
```

SELECT - получение данных

Пример:

```
SELECT * FROM users;
```

WHERE - условие фильтрации

Пример:

```
SELECT * FROM users WHERE age > 20;
```

UPDATE - изменение существующих записей

Пример:

```
UPDATE users SET age = 26 WHERE id = 1;
```

DELETE - удаление записей

Пример:

```
DELETE FROM users WHERE id = 1;
```

Продвинутые концепции:

- JOIN - объединение таблиц
- INNER JOIN - только совпадающие записи
- LEFT/RIGHT JOIN - все записи из одной таблицы
- Агрегатные функции:

- COUNT() - количество записей
- SUM() - сумма значений

Практические рекомендации:

- Начинать с простых запросов
- Постепенно усложнять структуру
- Использовать псевдонимы для читаемости
- Документировать сложные запросы
- Применять форматирование кода

Оптимизация:

- Использовать индексы для ускорения запросов
- Избегать избыточных данных
- Оптимизировать JOIN операции
- Применять пагинацию для больших выборок

Безопасность:

- Защищать от SQL-инъекций
- Использовать параметризованные запросы
- Контролировать права доступа
- Шифровать чувствительные данные

Практические задания:

- Создание учебной базы данных
- Реализация CRUD операций
- Построение отчетов
- Оптимизация производительности
- Работа с транзакциями

Частые ошибки:

- Неправильное использование кавычек

- Отсутствие обработки исключений
- Избыточные данные в запросах
- Нечеткие условия JOIN
- Отсутствие индексации

Вопросы для самоконтроля:

- Что такое реляционная база данных и как она организована?
- Какие основные команды SQL необходимо знать для работы с базами данных?
- Что такое первичный ключ (Primary Key) и для чего он используется?
- Как работает команда SELECT и какие основные параметры она может включать?
- Что такое внешний ключ (Foreign Key) и как он используется в реляционных базах данных?
- Как выполняется фильтрация данных в SQL-запросах?

2.2 Работа с SQLite3

Обучающийся должен:

Знать:

- Основные принципы работы с локальными базами данных
- Структуру и особенности SQLite3
- Методы создания и управления базами данных
- Правила выполнения операций CRUD

Уметь:

- Создавать и подключаться к локальным базам данных SQLite3
- Формировать таблицы и управлять их структурой
- Выполнять операции с данными (создание, чтение, обновление, удаление)
- Использовать SQLite3 в Python через модуль sqlite3

Методические рекомендации

Основные понятия SQLite3:

SQLite3 - Встраиваемая система управления базами данных (SQLite) третьей версии. Представляет собой библиотеку на языке C, реализующую автономную, безсерверную, нулевую конфигурацию SQL-базу данных.

Преимущества SQLite3:

- Легковесность и высокая производительность
- Отсутствие необходимости в отдельном сервере
- Поддержка большинства функций SQL92
- Кроссплатформенность

Создание и подключение к БД:

Connection - Объект соединения с базой данных:

```
import sqlite3
```

```
connection = sqlite3.connect('example.db') # Создание/подключение к БД
```

Cursor - Объект курсора для выполнения SQL-запросов:

```
cursor = connection.cursor()
```

CREATE TABLE - Создание новой таблицы

```
CREATE TABLE users (  
    id INTEGER PRIMARY KEY,  
    name TEXT NOT NULL,  
    age INTEGER  
);
```

ALTER TABLE - Изменение структуры таблицы

DROP TABLE - Удаление таблицы

Create (Создание):

```
cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ('John', 25))
```

```
connection.commit()
```

Read (Чтение):

```
cursor.execute("SELECT * FROM users")
```

```
rows = cursor.fetchall()
```

Update (Обновление):

```
cursor.execute("UPDATE users SET age = ? WHERE name = ?", (26, 'John'))
```

Delete (Удаление):

```
cursor.execute("DELETE FROM users WHERE name = ?", ('John',))
```

Практические рекомендации:

- Использовать контекстный менеджер для управления соединением
- Применять параметризованные запросы для безопасности
- Регулярно делать commit() после изменений
- Закрывать соединение после работы

Распространенные ошибки:

- OperationalError - Ошибки работы с БД
- IntegrityError - Нарушение целостности данных
- ProgrammingError - Ошибки программирования

Производительность:

- Использовать индексы для ускорения запросов
- Оптимизировать сложные запросы
- Использовать транзакции для массовых операций

Безопасность:

- Защищать от SQL-инъекций
- Управлять правами доступа
- Шифровать чувствительные данные

Вопросы для самоконтроля:

- Что такое SQLite3 и для чего он используется в программировании?
- Как выполняется подключение к базе данных SQLite3 в Python?
- Какие основные операции можно выполнять с данными в SQLite3?
- В чем заключаются преимущества использования SQLite3 по сравнению с другими системами управления базами данных?

2.3 ORM системы. SQLAlchemy.

Обучающийся должен:

Знать:

- *Основные принципы работы ORM (Object-Relational Mapping)*
- *Структуру и особенности SQLAlchemy*
- *Методы создания и управления базами данных через ORM*
- *Правила выполнения операций CRUD с использованием SQLAlchemy*

Уметь:

- *Создавать и подключаться к базам данных через SQLAlchemy*
- *Формировать модели данных и управлять их структурой*
- *Выполнять операции с данными (создание, чтение, обновление, удаление)*
- *Использовать SQLAlchemy в Python для работы с базами данных*

Методические рекомендации

Основные понятия ORM и SQLAlchemy:

ORM (Object-Relational Mapping) - Технология программирования, которая связывает объектно-ориентированные модели с реляционными базами данных.

SQLAlchemy - Популярная ORM-библиотека для Python, предоставляющая полный набор инструментов для работы с базами данных.

Преимущества использования ORM:

- Упрощение работы с базами данных
- Абстракция от SQL-запросов
- Кроссплатформенность
- Защита от SQL-инъекций

Создание подключения к БД:

Engine - Объект соединения с базой данных:

```
from sqlalchemy import create_engine  
  
engine = create_engine('sqlite:///example.db')
```

Session - Объект сессии для выполнения операций.

```
from sqlalchemy.orm import sessionmaker  
  
Session = sessionmaker(bind=engine)  
  
session = Session()
```

Declarative Base - Базовый класс для определения моделей

```
from sqlalchemy.ext.declarative import declarative_base  
  
Base = declarative_base()
```

Model - Класс, представляющий таблицу в базе данных.

```
from sqlalchemy import Column, Integer, String  
  
class User(Base):
```

```
__tablename__ = 'users'
```

```
id = Column(Integer, primary_key=True)
```

```
name = Column(String)
```

```
age = Column(Integer)
```

Create (Создание):

```
new_user = User(name='John', age=25)
```

```
session.add(new_user)
```

```
session.commit()
```

Read (Чтение):

```
user = session.query(User).filter_by(name='John').first()
```

Update (Обновление):

```
user.age = 26
```

```
session.commit()
```

Delete (Удаление):

```
session.delete(user)
```

```
session.commit()
```

Практические рекомендации:

- Начинать с простых моделей
- Постепенно усложнять структуру
- Использовать relationship для связей между таблицами
- Применять query API для сложных запросов
- Документировать модели и связи

Распространенные ошибки:

- `OperationalError` - Ошибки работы с БД
- `IntegrityError` - Нарушение целостности данных
- `NoResultFound` - Запись не найдена
- `MultipleResultsFound` - Найдено несколько записей

Производительность:

- Использовать `lazy loading` для оптимизации
- Оптимизировать запросы через `join()`
- Использовать пагинацию для больших выборок
- Применять индексы через `Index()`

Безопасность:

- Защищать от SQL-инъекций
- Управлять правами доступа
- Шифровать чувствительные данные
- Использовать безопасные методы аутентификации

Вопросы для самоконтроля:

- Что такое ORM (Object-Relational Mapping) и какую роль он играет в работе с базами данных?
- Какие основные компоненты SQLAlchemy необходимо знать для работы с базами данных?
- Как выполняются CRUD-операции (Create, Read, Update, Delete) в SQLAlchemy?
- В чем заключаются преимущества использования SQLAlchemy по сравнению с чистым SQL?
- Как происходит сопоставление Python-классов с таблицами базы данных в SQLAlchemy?

Раздел 3. СОЗДАНИЕ ГРАФИЧЕСКИХ ИНТЕРФЕЙСОВ

3.1 Библиотека Tkinter.

Обучающийся должен:

Знать:

- Основные принципы создания графических интерфейсов пользователя (GUI).
- Структуру и особенности библиотеки Tkinter в Python.
- Методы создания окон, виджетов и их настройки.
- Принципы обработки событий в графических интерфейсах.

Уметь:

- Создавать базовые оконные приложения с использованием Tkinter.
- Добавлять и настраивать виджеты (кнопки, текстовые поля, метки и т.д.).
- Реализовывать обработку событий (например, нажатие кнопки).
- Проектировать простые пользовательские интерфейсы для приложений.

Методические рекомендации

Основные понятия Tkinter:

Tkinter - Встроенная библиотека Python для создания графических интерфейсов пользователя. Она предоставляет инструменты для создания окон, виджетов и управления их поведением.

Widget (виджет) - Основной элемент графического интерфейса, такой как кнопка, текстовое поле или метка.

```
from tkinter import *
```

```
root = Tk() # Создание основного окна
```

```
label = Label(root, text="Привет, Tkinter!") # Создание виджета метки  
label.pack() # Размещение виджета в окне  
root.mainloop() # Запуск главного цикла обработки событий
```

Создание окон и их настройка:

Window (окно) - Основной контейнер для размещения виджетов.

```
root = Tk() # Создание окна  
root.title("Мое первое приложение") # Установка заголовка окна  
root.geometry("400x300") # Установка размеров окна  
root.resizable(False, False) # Запрет изменения размеров окна
```

Работа с виджетами:

Button (кнопка) - Виджет для выполнения действий.

Entry (текстовое поле) - Виджет для ввода текста.

Label (метка) - Виджет для отображения текста или изображения.

```
button = Button(root, text="Нажми меня", command=lambda: print("Кнопка  
нажата"))
```

```
button.pack()
```

```
entry = Entry(root)
```

```
entry.pack
```

```
label = Label(root, text="Введите текст:")
```

```
label.pack()
```

Обработка событий:

Event Handling (обработка событий) - Механизм реагирования на действия пользователя, такие как нажатие кнопки или ввод текста.

```
def on_button_click():  
    user_input = entry.get()  
    label.config(text=f"Вы ввели: {user_input}")  
  
button = Button(root, text="Отправить", command=on_button_click)  
  
button.pack()
```

Расположение виджетов:

Layout Managers (менеджеры компоновки) - Инструменты для управления расположением виджетов в окне.

pack() - Простой способ размещения виджетов по порядку.

grid() - Размещение виджетов в таблице (строки и столбцы).

place() - Точное позиционирование виджетов с указанием координат.

Пример использования grid()

```
label1 = Label(root, text="Имя:")
```

```
label1.grid(row=0, column=0)
```

```
entry1 = Entry(root)
```

```
entry1.grid(row=0, column=1)
```

Практические рекомендации:

- Начинать с простых интерфейсов (одно окно, несколько виджетов).
- Постепенно добавлять функциональность и сложные элементы.
- Использовать комментарии для описания назначения виджетов и функций.
- Тестировать интерфейс на разных разрешениях экрана.

Распространенные ошибки:

- `AttributeError` - Вызов несуществующего метода или свойства виджета.

- `TypeError` - Передача неверных типов данных в параметры виджета.
- `RuntimeError` - Ошибки, связанные с неправильным использованием главного цикла.

Производительность:

- Минимизировать количество глобальных переменных.
- Использовать локальные переменные для хранения состояния виджетов.
- Избегать сложных вычислений внутри обработчиков событий.

Безопасность:

- Валидация входных данных от пользователя.
- Защита от некорректного ввода через проверку типов и значений.
- Использование безопасных методов для работы с файлами и сетью.

Вопросы для самоконтроля:

- Какие основные этапы разработки программного обеспечения необходимо учитывать при создании приложений с использованием Python?
- В чем заключаются преимущества использования ORM (Object-Relational Mapping) по сравнению с написанием сырых SQL-запросов?
- Какие ключевые особенности библиотеки Tkinter делают ее популярным выбором для создания графических интерфейсов в Python?
- Какие основные принципы работы с API и какие популярные сторонние библиотеки Python используются для взаимодействия с внешними сервисами?
- Какие базовые структуры данных Python наиболее часто применяются в прикладном программировании и почему?

3.2. Работа с PyQt6 и QtDesigner.

Обучающийся должен:

Знать:

- Основные принципы работы с библиотекой PyQt6 для создания графических интерфейсов.
- Структуру и особенности использования Qt Designer для визуального проектирования интерфейсов.
- Методы интеграции файлов .ui (созданных в Qt Designer) с Python-кодом.
- Принципы обработки событий и сигналов в PyQt6.
- Базовые виджеты PyQt6 и их функциональность.

Уметь:

- Создавать графические интерфейсы с использованием Qt Designer.
- Конвертировать файлы .ui в Python-код с помощью pyuic6.
- Программировать логику приложения на основе сигналов и слотов.
- Настроить взаимодействие между различными элементами интерфейса.
- Реализовывать базовые функции управления окнами и диалогами.

Методические рекомендации

Основные понятия PyQt6 и Qt Designer:

PyQt6 - Это набор Python-библиотек, предоставляющих инструменты для создания кроссплатформенных графических интерфейсов.

Qt Designer - Визуальный редактор, позволяющий создавать интерфейсы путем перетаскивания виджетов.

```
from PyQt6.QtWidgets import QApplication, QMainWindow
```

```
app = QApplication([]) # Создание экземпляра приложения
```

```
window = QMainWindow() # Создание главного окна
```

```
window.setWindowTitle("Пример PyQt6") # Установка заголовка окна
```

```
window.show() # Отображение окна
```

```
app.exec() # Запуск главного цикла приложения
```

Создание интерфейса в Qt Designer:

- Добавление виджетов (QPushButton, QLabel, QLineEdit и др.) на форму.
- Настройка свойств виджетов через панель свойств.
- Сохранение интерфейса в файл формата .ui.

Конвертация .ui в Python-код:

```
pyuic6 design.ui -o design.py
```

После конвертации можно импортировать и использовать класс в основном коде:

```
from PyQt6.QtWidgets import QApplication, QMainWindow
```

```
from design import Ui_MainWindow # Импорт сконвертированного  
класса
```

```
class MainWindow(QMainWindow, Ui_MainWindow):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
        self.setupUi(self) # Инициализация интерфейса
```

```
app = QApplication([])
```

```
window = MainWindow()
```

```
window.show()
```

```
app.exec()
```

Работа с сигналами и слотами:

Signal (сигнал) - Событие, которое происходит в интерфейсе (например, нажатие кнопки).

Slot (слот) - Функция, которая вызывается в ответ на сигнал.

```
from PyQt6.QtWidgets import QPushButton  
  
button = QPushButton("Нажми меня")  
  
button.clicked.connect(lambda: print("Кнопка нажата")) # Подключение  
сигнала к слоту
```

Основные виджеты PyQt6:

QLabel - Отображение текста или изображения.

QLineEdit - Поле для ввода текста.

QPushButton - Кнопка для выполнения действий.

QComboBox - Выпадающий список.

QTableWidget - Таблица для отображения данных.

Обработка событий:

Event Handling (обработка событий) - Механизм реагирования на действия пользователя.

```
def on_button_click():  
  
    user_input = line_edit.text()  
  
    label.setText(f"Вы ввели: {user_input}")  
  
button.clicked.connect(on_button_click)
```

Практические рекомендации:

- Начинать с простых интерфейсов (одно окно, несколько виджетов).
- Постепенно добавлять функциональность и сложные элементы.

- Использовать комментарии для описания назначения виджетов и функций.
- Тестировать интерфейс на разных разрешениях экрана.

Распространенные ошибки:

- `AttributeError` - Вызов несуществующего метода или свойства виджета.
- `TypeError` - Передача неверных типов данных в параметры виджета.
- `RuntimeError` - Ошибки, связанные с неправильным использованием главного цикла.

Производительность:

- Минимизировать количество глобальных переменных.
- Использовать локальные переменные для хранения состояния виджетов.
- Избегать сложных вычислений внутри обработчиков событий.

Безопасность:

- Валидация входных данных от пользователя.
- Защита от некорректного ввода через проверку типов и значений.
- Использование безопасных методов для работы с файлами и сетью.

Вопросы для самоконтроля:

- Что такое PyQt6 и какие основные возможности он предоставляет для создания графических интерфейсов?
- Как выполняется конвертация файлов .ui, созданных в Qt Designer, в Python-код?
- Какие основные виджеты PyQt6 используются для создания пользовательских интерфейсов?
- В чем заключается механизм сигналов и слотов в PyQt6?
- Какие распространенные ошибки могут возникать при работе с PyQt6 и как их избежать?

Раздел 4. ВЕБ-РАЗРАБОТКА

4.1 Базовый HTML, CSS

Обучающийся должен:

Знать:

- *Основные теги и структуру HTML-документа.*
- *Правила написания и использования CSS для стилизации веб-страниц.*
- *Принципы работы с селекторами, свойствами и значениями в CSS.*
- *Методы создания адаптивных и кроссбраузерных веб-страниц.*
- *Основные принципы семантической верстки.*

Уметь:

- *Создавать базовые HTML-документы с использованием семантических тегов.*
- *Применять CSS для оформления и стилизации элементов страницы.*
- *Использовать селекторы, псевдоклассы и псевдоэлементы для точного управления стилями.*
- *Реализовывать адаптивный дизайн с помощью медиа-запросов.*
- *Разрабатывать простые веб-страницы с учетом современных стандартов веб-разработки.*

Методические рекомендации

Основные понятия HTML:

HTML (HyperText Markup Language) - Язык разметки, используемый для создания структуры веб-страниц.

`<!DOCTYPE html>`

`<html lang="ru">`

`<head>`

```
<meta charset="UTF-8">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>Пример HTML</title>
```

```
</head>
```

```
<body>
```

```
<h1>Заголовок</h1>
```

```
<p>Абзац текста.</p>
```

```
</body>
```

```
</html>
```

Семантическая верстка:

Семантические теги (например, <header>, <main>, <footer>) помогают улучшить читаемость кода и SEO-оптимизацию.

```
<header>
```

```
<h1>Заголовок сайта</h1>
```

```
<nav>
```

```
<ul>
```

```
<li><a href="#">Главная</a></li>
```

```
<li><a href="#">О нас</a></li>
```

```
</ul>
```

```
</nav>
```

```
</header>
```

```
<main>
```

```
<article>
```

```
<h2>Статья</h2>
```

```
<p>Текст статьи.</p>
```

```
</article>
```

```
</main>
```

```
<footer>
```

```
<p>© 2024 Все права защищены.</p>
```

```
</footer>
```

Основные понятия CSS:

CSS (Cascading Style Sheets) - Язык описания внешнего вида документа, написанного на HTML.

```
body {  
    font-family: Arial, sans-serif;  
    background-color: #f0f0f0;  
}
```

```
h1 {  
    color: #333;  
    text-align: center;  
}
```

Селекторы и их использование:

Селекторы позволяют выбирать элементы для применения стилей.

Типовой селектор: `p { color: red; }`

Классовый селектор: `.example { font-size: 20px; }`

ID-селектор: `#unique { background: yellow; }`

Псевдоклассы: `a:hover { color: blue; }`

Работа с макетами и позиционированием:

Box Model - Модель, описывающая структуру элемента (content, padding, border, margin).

Flexbox - Инструмент для создания гибких макетов.

Grid - Система для построения сложных сеток.

```
.container {  
    display: flex;  
    justify-content: space-between;  
}  
  
.item {  
    flex: 1;  
    margin: 10px;  
}
```

Адаптивный дизайн:

Медиа-запросы позволяют изменять стили в зависимости от размера экрана.

```
@media (max-width: 768px) {  
    body {  
        font-size: 14px;  
    }  
}
```

Практические рекомендации:

— Начинать с простых HTML-документов и постепенно добавлять стили.

- Использовать комментарии для объяснения сложных частей кода.
- Проверять совместимость с различными браузерами.
- Тестировать страницы на разных устройствах (компьютерах, планшетах, телефонах).

Распространенные ошибки:

- Отсутствие закрывающих тегов в HTML.
- Неправильное использование селекторов в CSS.
- Игнорирование семантической верстки.
- Отсутствие адаптивности для мобильных устройств.

Производительность:

- Минимизировать количество глобальных стилей.
- Использовать сжатие CSS-файлов.
- Оптимизировать изображения для быстрой загрузки.

Безопасность:

- Защищать данные пользователей при работе с формами.
- Использовать HTTPS для безопасной передачи данных.
- Проверять входные данные на стороне сервера.

Вопросы для самоконтроля:

- Что такое HTML и какие основные теги используются для создания структуры веб-страниц?
- Какие преимущества предоставляет семантическая верстка в HTML?
- Как работают селекторы в CSS и какие виды селекторов существуют?
- В чем заключается принцип работы Flexbox и Grid в CSS?
- Как реализуется адаптивный дизайн с помощью медиа-запросов?

4.2 Основы Fast API и веб-хуки.

Обучающийся должен:

Знать:

- Основные принципы работы с FastAPI для создания веб-приложений.
- Структуру и особенности асинхронного программирования в Python.
- Методы обработки HTTP-запросов и ответов в FastAPI.
- Принципы работы с веб-хуками (webhooks) для взаимодействия между сервисами.
- Базовые концепции RESTful API и их реализацию в FastAPI.

Уметь:

- *Создавать базовые веб-приложения с использованием FastAPI.*
- *Обрабатывать различные типы HTTP-запросов (GET, POST, PUT, DELETE).*
- *Реализовывать асинхронные операции в FastAPI.*
- *Настроить и использовать веб-хуки для интеграции с внешними сервисами.*
- *Тестировать и отлаживать RESTful API с помощью инструментов, таких как Swagger UI.*

Методические рекомендации

Основные понятия FastAPI:

FastAPI - Современный фреймворк для создания веб-приложений на Python, который поддерживает асинхронное программирование и автоматически генерирует документацию API.

```
from fastapi import FastAPI
```

```
app = FastAPI()
```

```
@app.get("/")
```

```
def read_root():  
  
    return {"message": "Hello, FastAPI!"}
```

Асинхронное программирование:

Async/Await - Ключевые слова Python для асинхронного выполнения кода, которые позволяют эффективно обрабатывать задачи без блокировки основного потока.

```
@app.get("/async")  
  
async def async_example():  
  
    await some_async_function()  
  
    return {"message": "Async operation completed"}
```

Обработка HTTP-запросов:

HTTP Methods (методы HTTP) - **GET, POST, PUT, DELETE** используются для выполнения различных операций над ресурсами.

```
@app.post("/items/")  
  
def create_item(item: dict):  
  
    return {"item": item}  
  
@app.put("/items/{item_id}")  
  
def update_item(item_id: int, item: dict):  
  
    return {"item_id": item_id, "updated_item": item}
```

Работа с веб-хуками:

Webhooks (веб-хуки) - Механизм, позволяющий внешним сервисам отправлять данные в ваше приложение через HTTP-запросы.

```
@app.post("/webhook")  
  
async def handle_webhook(data: dict):
```

```
# Обработка данных, полученных через веб-хук
```

```
return {"status": "Webhook received", "data": data}
```

RESTful API:

REST (Representational State Transfer) - Архитектурный стиль для создания веб-сервисов, основанный на стандартных HTTP-методах.

Использование маршрутов для доступа к ресурсам.

Возврат данных в формате JSON.

Автоматическая документация:

Swagger UI - Инструмент, автоматически генерируемый FastAPI для тестирования и документирования API. Доступен по адресу `/docs`.

Практические рекомендации:

- Начинать с простых эндпоинтов (например, GET-запросов).
- Постепенно добавлять сложные операции (POST, PUT, DELETE).
- Использовать модели данных Pydantic для валидации входящих данных.
- Тестируйте API с помощью инструментов, таких как Postman или Swagger UI.

Распространенные ошибки:

- `ValidationError` - Ошибки валидации данных, вызванные неверным форматом входных данных.
- `HTTPException` - Исключения, связанные с HTTP-статусами (например, 404 Not Found).
- `RuntimeError` - Ошибки, возникающие при неправильной настройке асинхронных операций.

Производительность:

- Использовать асинхронные функции для повышения производительности.
- Минимизировать количество синхронных операций в асинхронном коде.
- Оптимизировать запросы к базам данных и внешним сервисам.

Безопасность:

- Защищать API с помощью аутентификации и авторизации (например, JWT).
- Валидировать все входящие данные.
- Использовать HTTPS для безопасной передачи данных.

Вопросы для самоконтроля:

- Что такое FastAPI и какие преимущества он предоставляет для создания веб-приложений?
- Как выполняется асинхронная обработка запросов в FastAPI?
- Какие основные HTTP-методы используются в RESTful API и для чего они нужны?
- Что такое веб-хуки и как они используются для взаимодействия между сервисами?
- Как обеспечивается безопасность в FastAPI и какие методы защиты данных существуют?

СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ:

Основная литература:

1. Изотова Е.Д.

Основы научного программирования на примере языка Python. Учебно-методическое пособие.

Казань: Казан. ун-т, 2015. – 42 с.

2. Лутц М.

Изучаем Python

СПб.: Символ-Плюс, 2009.

3. Хахаев И.А.

Практикум по алгоритмизации и программированию на Python

М.: Альт Линукс, 2010. – 126 с.

Интернет-ресурсы:

ALT Linux. Свободные книги о свободном ПО

URL: <http://books.altlinux.ru/PythonSchool/>

Документация Python

URL: <https://www.python.org/doc/>

Python ru. Курс Python без воды для новичков и продолжающих

URL: <https://python.ru/>

Онлайн-обучение детей программированию

URL: <http://progkids.ru>

CheckiO (Кодирование игр и задачи программирования для начинающих и продвинутых).

URL: <https://checkio.org/>