Иванов Иван Андреевич

ИВТ 4 курс

**I. Threads, AsyncIO, Concurrency**

**1.**

import threading  
import time  
  
def print\_numbers():  
 for i in range(10, 0, -1):  
 print(i)  
 time.sleep(1)  
  
number\_thread = threading.Thread(target=print\_numbers)  
  
number\_thread.start()  
  
number\_thread.join()

**2.**

import threading  
import time  
  
def print\_even\_numbers():  
 for i in range(2, 21, 2):  
 print(f"Четное: {i}")  
 time.sleep(0.5)  
  
def print\_odd\_numbers():  
 for i in range(1, 21, 2):  
 print(f"Нечетное: {i}")  
 time.sleep(0.5)  
  
even\_thread = threading.Thread(target=print\_even\_numbers)  
odd\_thread = threading.Thread(target=print\_odd\_numbers)  
  
even\_thread.start()  
odd\_thread.start()  
  
even\_thread.join()  
odd\_thread.join()

**3.**

import threading  
import time  
  
class Counter:  
 def \_\_init\_\_(self):  
 self.value = 0  
 self.lock = threading.Lock()  
  
 def increment(self):  
 with self.lock:  
 self.value += 1  
  
 def decrement(self):  
 with self.lock:  
 self.value -= 1  
  
 def get\_value(self):  
 with self.lock:  
 return self.value  
  
counter = Counter()  
  
  
def increment\_counter():  
 for \_ in range(500):  
 counter.increment()  
  
  
def decrement\_counter():  
 for \_ in range(500):  
 counter.decrement()  
  
  
increment\_thread = threading.Thread(target=increment\_counter)  
decrement\_thread = threading.Thread(target=decrement\_counter)  
  
  
increment\_thread.start()  
decrement\_thread.start()  
  
increment\_thread.join()  
decrement\_thread.join()

**4.**

import threading  
  
  
barrier = threading.Barrier(2)  
  
def print\_even\_numbers():  
 for i in range(2, 51, 2):  
 print(f"Четное: {i}")  
 barrier.wait()   
  
  
def print\_odd\_numbers():  
 for i in range(1, 51, 2):  
 print(f"Нечетное: {i}")  
 barrier.wait()   
  
even\_thread = threading.Thread(target=print\_even\_numbers)  
odd\_thread = threading.Thread(target=print\_odd\_numbers)  
  
  
even\_thread.start()  
odd\_thread.start()  
  
  
even\_thread.join()  
odd\_thread.join()

**5.**

import asyncio  
  
async def perform\_tasks():  
 tasks = [  
 asyncio.create\_task(task1()),  
 asyncio.create\_task(task2()),  
 asyncio.create\_task(task3())  
 ]  
  
   
 await asyncio.gather(\*tasks)  
  
  
 results = [task.result() for task in tasks]  
 print("Результаты задач:", results)  
  
  
async def task1():  
 await asyncio.sleep(2)  
 return "Результат из задачи 1"  
  
async def task2():  
 await asyncio.sleep(1)  
 return "Результат из задачи 2"  
  
async def task3():  
 await asyncio.sleep(3)  
 return "Результат из задачи 3"  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 asyncio.run(perform\_tasks())

**6.**

import asyncio  
  
async def process\_task1():  
 print("Задача 1: Начало выполнения")  
 await asyncio.sleep(2)   
 print("Задача 1: Завершение выполнения")  
 return "Результат задачи 1"  
  
async def process\_task2():  
 print("Задача 2: Начало выполнения")  
 await asyncio.sleep(1)   
 print("Задача 2: Завершение выполнения")  
 return "Результат задачи 2"  
  
async def process\_task3():  
 print("Задача 3: Начало выполнения")  
 await asyncio.sleep(3)   
 print("Задача 3: Завершение выполнения")  
 return "Результат задачи 3"  
  
  
async def main():  
  
 task1 = asyncio.create\_task(process\_task1())  
 task2 = asyncio.create\_task(process\_task2())  
 task3 = asyncio.create\_task(process\_task3())  
  
   
 await asyncio.gather(task1, task2, task3)  
  
  
 results = [task1.result(), task2.result(), task3.result()]  
 print("Результаты задач:", results)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 asyncio.run(main())

**II. Django Part**

**1)** Django — это бесплатный фреймворк для веб-приложений, написанный на Python. Django решает такие задачи как: создание веб-приложений, создание API, создание сервисов обработки данных, создание CMS и CRM систем и реализация email-рассылок.

**2)** Файловая структура в Django:

Корневая директория проекта (настройки проекта, URL-маршруты, статические файлы и другие компоненты)

Директория приложения (apps) (файлы, связанные с этим приложением, такие как модели, представления, URL-маршруты и шаблоны)

Директория static (статические файлы, такие как CSS, JavaScript, изображения и другие ресурсы)

Директория media (загруженные пользователем медиа файлы, такие как изображения или видео)

Директория templates (шаблоны (HTML-файлы), используемые для отображения данных на веб-страницах)

**3)** Модель в Django — это способ описать, какие данные нужно хранить и как с ними работать. Операции, которые можно выполнять с моделями, включают:

Создание записей: можно добавлять новые записи в базу данных, используя модель.

Чтение данных: можно получать данные из базы данных, фильтровать и сортировать их.

Обновление данных: можно изменять значения полей в существующих записях.

Удаление записей: можно удалять записи из базы данных.

**4)** Представления (views) в Django — это Python-функции, которые обрабатывают входящие HTTP-запросы и возвращают HTTP-ответы. Они связаны с URL-шаблонами, которые определяют, какой код должен выполняться при обращении к определенному URL.

**5)** В Django шаблоны написаны на языке, называемом Django Template Language (DTL). Они определяют, какие данные отображать на веб-страницах, а URL-шаблоны указывают, какие представления использовать для конкретных URL.

**6)** Создание миграции: python manage.py makemigrations

Применение миграции: python manage.py migrate

Создание суперпользователя: python manage.py createsuperuser

Запуск сервера разработки: python manage.py runserver

**7)** Django REST Framework (DRF) — это расширение для Django, которое упрощает создание REST API.

Основные возможности DRF:

Сериализация: преобразование данных между форматами (например, Python-объекты в JSON).

Аутентификация: поддержка различных методов аутентификации, таких как OAuth1 и OAuth2.

Разрешение доступа: управление правами доступа к данным.

Поддержка ORM и не-ORM источников данных.

**8)** from django.db import models  
  
class Order(models.Model):  
  
permissions = [  
("can\_add\_order", "Can add order"),  
 ]  
  
from django.contrib.auth.decorators import permission\_required  
  
@permission\_required("myapp.can\_add\_order")  
def add\_order(request):

**9)** URLconf (URL Configuration): это Python-модуль, который определяет, какие представления (views) обрабатывают запросы для конкретных URL.

Шаблонные теги для URL: в шаблонах можно использовать тег {% url %} для генерации URL на основе именованных URL-шаблонов.

Функция reverse(): в Python-коде можно использовать функцию reverse() для получения URL на основе именованных URL-шаблонов.

**10)** В файле settings.py укажите настройки для отправки почты, такие как хост, порт, протокол, имя пользователя и пароль.

**11)** Создание формы: определите форму, используя классы форм Django.

Отправка формы на сервер: в HTML-шаблоне добавьте форму на страницу:

Обработка данных на сервере: в представлении (view) обработайте данные, отправленные пользователем:

**12)** Аутентификация по паролю (Password-based authentication): пользователи могут войти, используя имя пользователя и пароль.

Аутентификация по токену (Token-based authentication): пользователи получают уникальный токен, который используется для аутентификации.

Социальная аутентификация (Social authentication): пользователи могут войти через сторонние сервисы, такие как Google, Facebook или Twitter.

Аутентификация через сторонние системы (Third-party authentication): пользователи могут войти через LDAP, SAML или другие сторонние системы.

**13)** Контейнеризация — это методология упаковки приложений и их зависимостей в изолированные контейнеры. Каждый контейнер содержит все необходимое для запуска приложения, включая код, библиотеки, среду выполнения и настройки. Вот простой ответ:

Цель контейнеризации:

Обеспечить портабельность приложений, позволяя запускать их в разных окружениях (локально, на сервере, в облаке) без изменений.

Упростить развертывание приложений, так как контейнеры могут быть быстро созданы, запущены и масштабированы.

**III. Java Part**

**1)** Apache Maven — это мощный инструмент управления проектами для Java и других языков на платформе JVM. Вот его основные возможности:

Управление зависимостями: Maven автоматически загружает и управляет зависимостями (библиотеками) для вашего проекта.

Он использует центральный репозиторий для загрузки JAR-файлов.

**2)** groupId: идентификатор группы проекта.

artifactId: идентификатор артефакта (проекта).

version: версия проекта.

packaging: тип упаковки (например, JAR, WAR).

dependencies: зависимости проекта.

build: настройки сборки, включая плагины и цели.

profiles: профили с различными настройками.

description: описание проекта.

developers: информация о разработчиках.

scm: настройки системы контроля версий.

repositories: определение репозиториев для зависимостей.

**3)** Присвойте вашему JAR-файлу groupId, artifactId и version.

Добавьте его в ваш репозиторий. Если у вас нет внутреннего репозитория, и вы хотите добавить JAR-файл в локальный репозиторий, выполните следующую команду:

mvn install:install-file -DgroupId=com.example -DartifactId=myartifact -Dversion=1.0 -Dpackaging=jar -Dfile=/путь/к/файлу.jar

Если у вас есть внутренний репозиторий, вы можете разместить JAR-файл там с помощью команды mvn deploy:deploy-file.

**4)** Плагины в Maven выполняют две основные функции:

**Плагины сборки (Build Plugins):**

Эти плагины выполняются во время процесса сборки. Примеры включают плагины Clean, Install и Surefire.

Они должны быть настроены в разделе <build> файла pom.xml.

**Плагины отчетности (Reporting Plugins):**

Эти плагины выполняются при генерации сайта для создания различных отчетов о проекте.

Они настраиваются в разделе <reporting> файла pom.xml.

Управление плагинами в Maven можно осуществлять через pluginManagement. В этом разделе мы объявляем плагины так же, как и в разделе <plugins>, но они становятся доступными для данного pom.xml и всех дочерних pom.xml. Это позволяет наследовать настройки плагинов без дублирования конфигурации или управления версиями.

**5)** Spring Boot предоставляет набор функций, которые делают процесс разработки более быстрым, эффективным и удобным для разработчиков. Вот несколько ключевых преимуществ Spring Boot:

Автоматическая конфигурация: аннотация @SpringBootApplication включает автоматическую конфигурацию, которая настраивает ваше приложение на основе библиотек, присутствующих в classpath. Это позволяет избежать ручного определения бинов в файле конфигурации.

Самостоятельность: приложения на Spring Boot могут работать независимо, без внешних серверов. Это удобно для микросервисной архитектуры, где каждый сервис может быть запущен отдельно без внешних серверов (например, Tomcat).

Готовность к продакшену: встроенные инструменты для мониторинга и управления приложениями. Модуль Actuator предоставляет эффективные конечные точки для мониторинга, проверки состояния, метрик и другой информации.

Отсутствие генерации кода: разработчику нужно сосредоточиться только на бизнес-логике. Spring Boot не генерирует лишний код и не требует XML-конфигурации.

**6)** **Настройка Java-проекта:**

Создайте структуру каталогов для вашего проекта.

**Создание проекта с помощью Maven:**

Откройте Spring Tool Suite (STS) или другую среду разработки.

Создайте новый Maven Project:

Выберите File → New → Maven Project.

В мастере создания проекта выберите архетип maven-archetype-quickstart.

Укажите Group Id и Artifact Id для вашего проекта.

**7)** SDKMAN — это инструмент для управления параллельными версиями различных SDK

Вот какие задачи можно выполнять с помощью SDKMAN:

Установка и управление версиями

Управление локальными версиями

Удаление версий

Удалите установленную версию

Просмотр доступных кандидатов и версий

**8)** Установка и управление версиями: установите последнюю стабильную версию выбранного SDK (например, Java JDK) с помощью следующей команды:

$ sdk install java

Вы можете указать конкретную версию, например:

$ sdk install maven 3.3.3

Управление локальными версиями: если у вас уже есть локальная установка, вы можете настроить ее как кандидата:

$ sdk install groovy 3.0.0-SNAPSHOT /path/to/groovy-3.0.0-SNAPSHOT

Удаление версий: удалите установленную версию:

$ sdk uninstall maven 3.3.3

Просмотр доступных кандидатов и версий: получите список доступных кандидатов:

$ sdk list

**9)** Если у вас еще нет SDKMAN, установите его. После установки, откройте новое окно терминала.

Установка конкретной версии Maven:

Чтобы установить определенную версию Maven, выполните команду:

$ sdk install maven <версия>