Программирование 7 сем. Итоговый тест

I. Threads, AsyncIO, Concurrency

Напишите функцию, которая создает и запускает поток, выводящий числа от 1 до 10 в обратном порядке с интервалом в 1 секунду.

```
import threading
import time

def worker() -> None:
    for i in range(10, 0, -1):
        print(i)
        time.sleep(1)

def main() -> None:
    t = threading.Thread(target=worker)
    t.start()
    t.join()

if __name__ == "__main__":
    main()
```

Создайте два потока, один из которых выводит четные числа, а другой - нечетные числа от 1 до 20 с интервалом в 0.5 секунды.

```
import threading
import time

def print_numbers(even: bool) -> None:
    for i in range(1, 21):
        if (i % 2 == 0) == even:
            print(i)
            time.sleep(0.5)
```

```
def main() -> None:
    t_even = threading.Thread(
            target=print_numbers,
            args=(True,),
        )
    t_odd = threading.Thread(
        target=print_numbers,
        args=(False,),
    )
    t_even.start()
    t_odd.start()
    t_even.join()
    t_odd.join()

if __name__ == "__main__":
    main()
```

Реализуйте счетчик с использованием блокировки (Lock) и для обеспечения параллельного доступа к списку в нескольких читающих и пишущих потоках - минимум двух.

```
import threading
import time

counter = 0
data = []
lock = threading.Lock()

def writer(thread_id: int) -> None:
    global counter, data
    for _ in range(10):
        with lock:
            counter += 1
            data.append(counter)
            print(f"Writer {thread_id} added: {counter}")
        time.sleep(0.1)

def reader(thread_id: int) -> None:
```

```
    global data
    for _ in range(10):
        with lock:
            print(f"Reader {thread_id} read: {data}")
        time.sleep(0.15)

def main() -> None:
    writers = [threading.Thread(target=writer, args=(i,)) for i in range(2)]
    readers = [threading.Thread(target=reader, args=(i,)) for i in range(2)]
    for t in writers + readers:
        t.start()
    for t in writers + readers:
        t.join()

if __name__ == "__main__":
    main()
```

Реализуйте программу на Python, использующую Barrier для синхронизации выполнения двух потоков, печатающих четные и нечетные числа.

```
import threading

barrier = threading.Barrier(2)

def print_numbers(even: bool) -> None:
    for i in range(1, 21):
        if (i % 2 == 0) == even:
            print(i)
            barrier.wait()

def main() -> None:
    t_even = threading.Thread(target=print_numbers, args=(True,))
    t_odd = threading.Thread(target=print_numbers, args=(False,))
    t_even.start()
    t_odd.start()
    t_even.join()
    t_odd.join()
```

```
if __name__ == "__main__":
    main()
```

Напишите функцию, использующую asyncio, для асинхронного выполнения и получения результатов любых нескольких задач в виде списка.

```
import asyncio

async def sample_task(n: int) → int:
    await asyncio.sleep(0.1)
    return n * 2

async def main() → None:
    tasks = [sample_task(i) for i in range(1, 6)]
    results = await asyncio.gather(*tasks)
    print(results)

if __name__ == "__main__":
    asyncio.run(main())
```

Напишите программу, использующую asyncio и async/await, для параллельной обработки нескольких задач, выполняемых в разных потоках.

```
import asyncio
import time

def blocking_task(name: str, dur: float) → str:
    time.sleep(dur)
    return f"{name} completed after {dur}s"

async def main() → None:
    tasks = [
        asyncio.to_thread(blocking_task, "A", 2.0),
        asyncio.to_thread(blocking_task, "B", 1.0),
        asyncio.to_thread(blocking_task, "C", 3.0)
    ]
```

```
    results = await asyncio.gather(*tasks)
    print(results)

if __name__ == "__main__":
    asyncio.run(main())
```

II. Django Part

Что такое фреймворк Django и какие задачи он решает?

Django – это высокоуровневый MVC веб-фреймворк для Python, который упрощает разработку веб-приложений, обеспечивая быстрый старт, безопасность и масштабируемость.

Как организована файловая структура проекта в Django?

Файловая структура проекта включает файл manage.py, папку с настройками (например, settings.py, urls.py, wsgi.py) и отдельные папки для приложений, каждое из которых содержит в себе модели, представления, шаблоны и статические файлы.

Что такое модель в Django и какие операции можно выполнять с моделями? (простым ответом)

Модель – это класс, описывающий структуру данных (таблицы базы данных). С моделями можно выполнять CRUD операции .

```
from django.db import models

class Product(models.Model):
    name = models.CharField(max_length=100)
```

Что такое представления (views) в Django и как они связаны с URL-шаблонами? (простым ответом)

Представления (views) – это функции или классы, обрабатывающие HTTP-запросы и возвращающие ответы. Они связываются с URL через конфигурацию в файле urls.py.

Как работает система шаблонов (template system) в Django? (простым ответом)

Система шаблонов позволяет отделить логику от представления, используя HTML-шаблоны с динамическими переменными, которые заполняются данными из контекста представления.

Как использовать миграции (migrations) для изменения схемы базы данных в Django? (простым ответом и/или примером команд)

Миграции отслеживают изменения в моделях и позволяют применять их к базе данных. Пример команд:

```
python manage.py makemigrations
python manage.py migrate
```

Что такое Django REST Framework (DRF) и какие возможности он предоставляет для создания API? (простым ответом и/или примером кода)

Django REST Framework (DRF) – расширение для Django, облегчающее создание RESTful API. Предоставляет сериализаторы, viewset’ы и инструменты аутентификации.

```
from rest_framework import serializers

class ProductSerializer(serializers.ModelSerializer):
    class Meta:
        model = Product
        fields = '__all__'
```

Как можно ограничить доступ к определенным частям приложения с использованием разрешений (permissions)? (простым ответом и/или примером кода)

Доступ к частям приложения можно ограничить с помощью разрешений, используя декораторы или атрибут **permission_classes** в DRF.

```
from rest_framework.permissions import IsAuthenticated
from rest_framework import viewsets

class MyViewSet(viewsets.ModelViewSet):
    permission_classes = [IsAuthenticated]
```

Какие инструменты предоставляет Django для работы с URL-адресами и маршрутами (routes)? (простым ответом и/или примером кода)

Django работает с URL через файл **urls.py**, где с помощью функций **path()** или **re_path()** определяются маршруты к представлениям.

```
from django.urls import path
from .views import home

urlpatterns = [
    path('', home, name='home'),
]
```

Как настроить отправку электронной почты (email) из Django приложения? (простым ответом и/или примером кода)

Отправку электронной почты настраивают через параметры в settings.py (например, EMAIL_BACKEND, EMAIL_HOST, EMAIL_PORT).

```
# settings.py - обязательная конфигурация для отправки писем
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.example.com'
EMAIL_PORT = 587

# mail.py - пример отправки письма
from django.core.mail import send_mail

send_mail(
    "Тема письма",
    "Содержание письма",
    "alice@example.com",
    ["bob@example.com"],
)
```

Как обрабатывать (и валидировать) формы в Django и как обрабатывать данные, введенные пользователем? (простым ответом и/или примером кода)

Для обработки и валидации форм используются классы forms.Form или forms.ModelForm, которые позволяют обрабатывать данные, введённые

пользователем.

```
from django import forms

class ContactForm(forms.Form):
    email = forms.EmailField()
    message = forms.CharField(widget=forms.Textarea)
```

Какие методы аутентификации предоставляет Django и как их использовать? (простым ответом)

Django предоставляет встроенную систему аутентификации с функциями authenticate, login, logout и поддержкой смены пароля, а также расширяемость для интеграции со сторонними сервисами.

Что такое контейнеризация и для чего она нужна в приложениях, использующих Django, Django REST Framework, FastAPI, Flask, etc.

Контейнеризация – это упаковка приложения и его зависимостей в изолированные контейнеры (например, с использованием Docker), что обеспечивает переносимость, удобство развертывания и масштабируемость.

III. Java Part

Что такое Apache Maven и какие возможности он предоставляет для управления зависимостями и сборки Java-проектов?

Apache Maven — это инструмент автоматизации сборки, который упрощает управление зависимостями, компиляцию, тестирование и упаковку Java-проектов через конфигурационный файл pom.xml.

Какие основные компоненты входят в файл pom.xml в проекте Maven?

Основные компоненты: groupId, artifactId, version, dependencies, build (с плагинами) и репозитории.

Как добавить новую зависимость в проект с использованием Maven и какие параметры можно указать для зависимости?

Добавление зависимости происходит через элемент <dependency> в pom.xml с указанием groupId, artifactId, version, а также опционально scope и type.

```
<dependency>
  <groupId>org.example</groupId>
  <artifactId>example-library</artifactId>
  <version>1.0.0</version>
</dependency>
```

Как управлять и настраивать плагины (plugins) в Maven и для чего они используются?

Плагины настраиваются в разделе <build> файла pom.xml и используются для выполнения задач сборки (компиляция, тестирование, упаковка и т.д.).

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
    </plugin>
  </plugins>
</build>
```

Что такое Spring Boot и какие преимущества он предоставляет для разработки Java-приложений?

Spring Boot — фреймворк, облегчающий создание автономных Spring-приложений с минимальной конфигурацией, автонастройкой и встроенным сервером для быстрого старта разработки.

Как создать новый проект Spring Boot с использованием Maven?

Новый проект можно создать через Spring Initializr (https://start.spring.io), выбрав Maven и нужные зависимости, затем запустить команду:

```
mvn spring-boot:run
```

Что такое SDKMAN и для чего он используется в контексте разработки на Java?

SDKMAN — утилита для управления версиями SDK, таких как JDK, Maven, Gradle, позволяющая легко устанавливать, переключать и обновлять

инструменты разработки.

Как управлять установленными версиями JDK с помощью SDKMAN?

Используйте команды: sdk list java для просмотра доступных версий, sdk install java <version> для установки и sdk use java <version> для переключения между версиями.

Как установить конкретную версию Maven с использованием SDKMAN?

Запустите команду:

```
sdk install maven <version>
```

Какие дополнительные инструменты или фреймворки могут быть полезны при разработке Java-проектов с использованием Maven и Spring Boot? Приведите примеры таких инструментов, доступных через SDKMAN.

Полезны инструменты, такие как Spring Boot CLI, Gradle, Kotlin и Groovy, которые облегчают разработку и автоматизацию процессов.

IV. OAuth 2.0

1. **Каков процесс выдачи Authorization Code? Раскройте основные аспекты этого процесса, представьте примеры своими словами для лучшего понимания.**

Процесс начинается с того, что клиентское приложение перенаправляет пользователя на сервер авторизации. Там пользователь проходит аутентификацию и дает согласие на доступ. После успешного входа сервер отправляет временный код (Authorization Code) обратно на указанный клиентом URL. Клиент затем использует этот код, отправляя запрос на сервер, чтобы обменять его на токен доступа. Например, приложение для работы с календарем может таким образом получить разрешение на доступ к вашим событиям.

V. OpenID Connect + OAuth 2.0

1. **Что добавляет OpenID Connect для Authentication? Объясните, как он расширяет OAuth 2.0 и добавляет возможность аутентификации через ID Token. Что такое ID Token?**

OpenID Connect добавляет механизм аутентификации поверх OAuth 2.0, вводя понятие ID Token – структурированного JWT, который содержит зашифрованные сведения о пользователе (его идентификатор, время аутентификации и другие данные). Это расширение позволяет не только авторизовывать доступ к ресурсам, но и подтверждать личность пользователя, обеспечивая единый стандарт для аутентификации и обмена информацией.

VI. Kafka

2. **На каком порту по умолчанию запускается ZooKeeper? Расскажите о его роли в экосистеме Kafka и процессе запуска.**

ZooKeeper запускается на порту 2181 по умолчанию. Он обеспечивает координацию и хранение метаданных кластера Kafka (информация о брокерах, топиках и разделах) и участвует в выборе контроллера. Перед запуском Kafka-брокеров запускается ZooKeeper, чтобы они могли подключаться к нему для синхронизации и управления кластером.

VII. RabbitMQ

3. **Что позволяет fanout exchange? Объясните, как работает данный exchange (основная идея), в каких сценариях он подходит лучше всего.**

Fanout exchange направляет каждое входящее сообщение на все связанные с ним очереди, игнорируя routing key. Это обеспечивает широковещательную рассылку, что идеально подходит для систем уведомлений или других сценариев, где необходимо доставлять одинаковую информацию всем подписчикам.