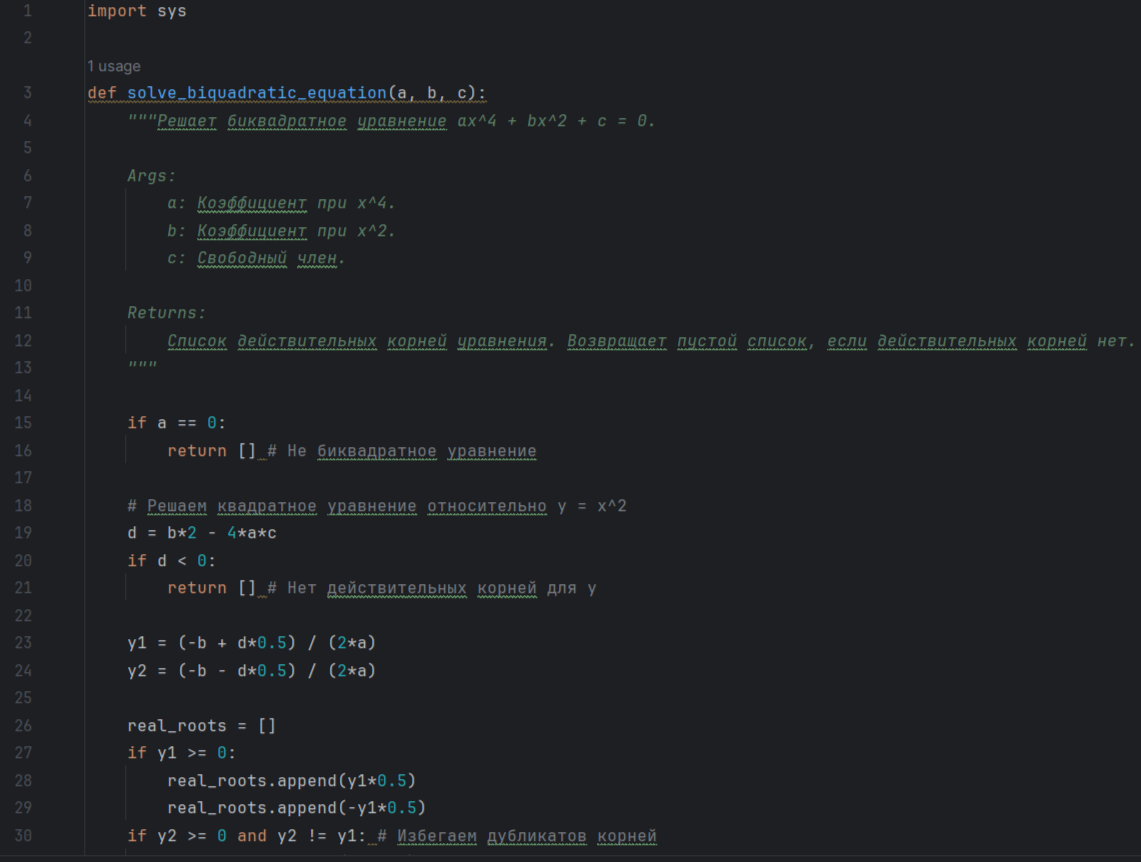
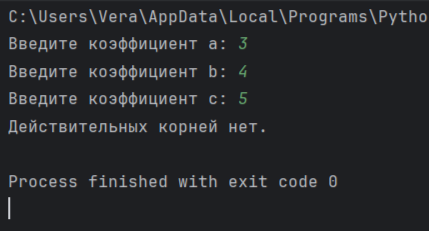
**ЛР1**

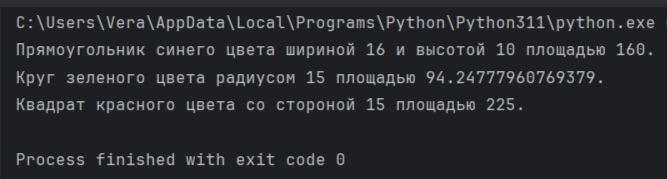






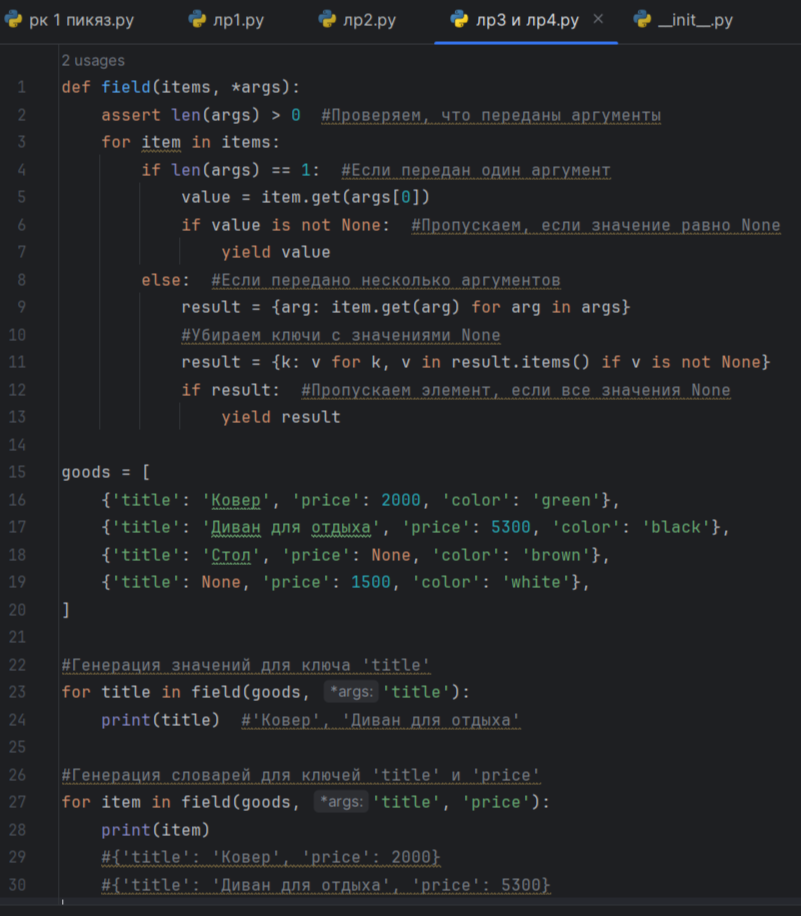
**ЛР2**

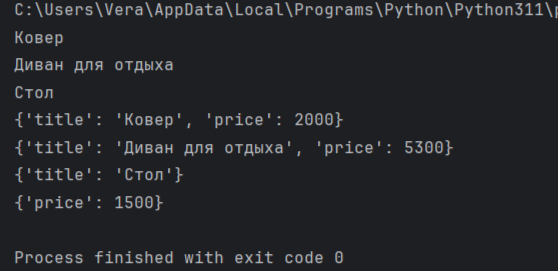
import math  
from abc import ABC, abstractmethod  
  
  
class Figure(ABC):  
 @abstractmethod  
 def square(self):  
 pass  
  
class FigureColor:  
 def \_\_init\_\_(self):  
 self.\_color = None  
  
 @property  
 def colorproperty(self):  
 return self.\_color  
  
 @colorproperty.setter  
 def colorproperty(self, value):  
 self.\_color = value  
  
class Rectangle(Figure):  
 FIGURE\_TYPE = "Прямоугольник"  
  
 @classmethod  
 def get\_figure\_type(cls):  
 return cls.FIGURE\_TYPE  
  
 def \_\_init\_\_(self, color\_param, width\_param, height\_param):  
 self.width = width\_param  
 self.height = height\_param  
 self.fc = FigureColor()  
 self.fc.colorproperty = color\_param  
  
 def square(self):  
 return self.width \* self.height  
  
 def \_\_repr\_\_(self):  
 return '{} {} цвета шириной {} и высотой {} площадью {}.'.format(  
 Rectangle.get\_figure\_type(),  
 self.fc.colorproperty,  
 self.width,  
 self.height,  
 self.square()  
 )  
  
class Circle(Figure):  
 FIGURE\_TYPE = "Круг"  
  
 @classmethod  
 def get\_figure\_type(cls):  
 return cls.FIGURE\_TYPE  
  
 def \_\_init\_\_(self, color\_param, r\_param):  
 self.r = r\_param  
 self.fc = FigureColor()  
 self.fc.colorproperty = color\_param  
  
 def square(self):  
 return math.pi \* (self.r\*2)  
  
 def \_\_repr\_\_(self):  
 return '{} {} цвета радиусом {} площадью {}.'.format(  
 Circle.get\_figure\_type(),  
 self.fc.colorproperty,  
 self.r,  
 self.square()  
 )  
  
class Square(Rectangle):  
 FIGURE\_TYPE = "Квадрат"  
  
 @classmethod  
 def get\_figure\_type(cls):  
 return cls.FIGURE\_TYPE  
  
 def \_\_init\_\_(self, color\_param, side\_param):  
 self.side = side\_param  
 super().\_\_init\_\_(color\_param, self.side, self.side)  
  
 def \_\_repr\_\_(self):  
 return '{} {} цвета со стороной {} площадью {}.'.format(  
 Square.get\_figure\_type(),  
 self.fc.colorproperty,  
 self.side,  
 self.square()  
 )  
  
def main():  
 r = Rectangle("синего", 16, 10)  
 c = Circle("зеленого", 15)  
 s = Square("красного", 15)  
 print(r)  
 print(c)  
 print(s)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

****

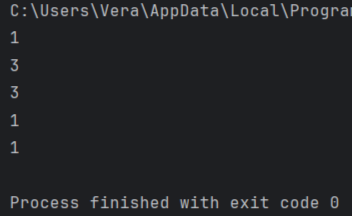
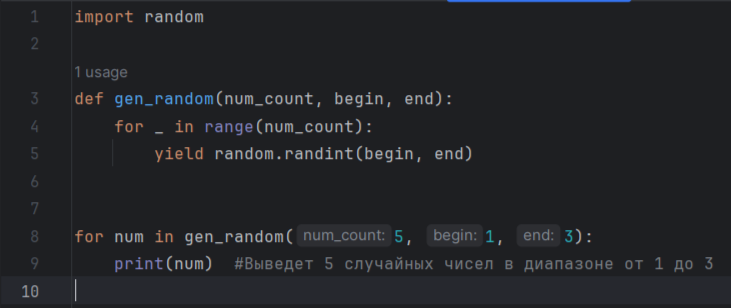
**ЛР3 и ЛР4**

**Номер1**

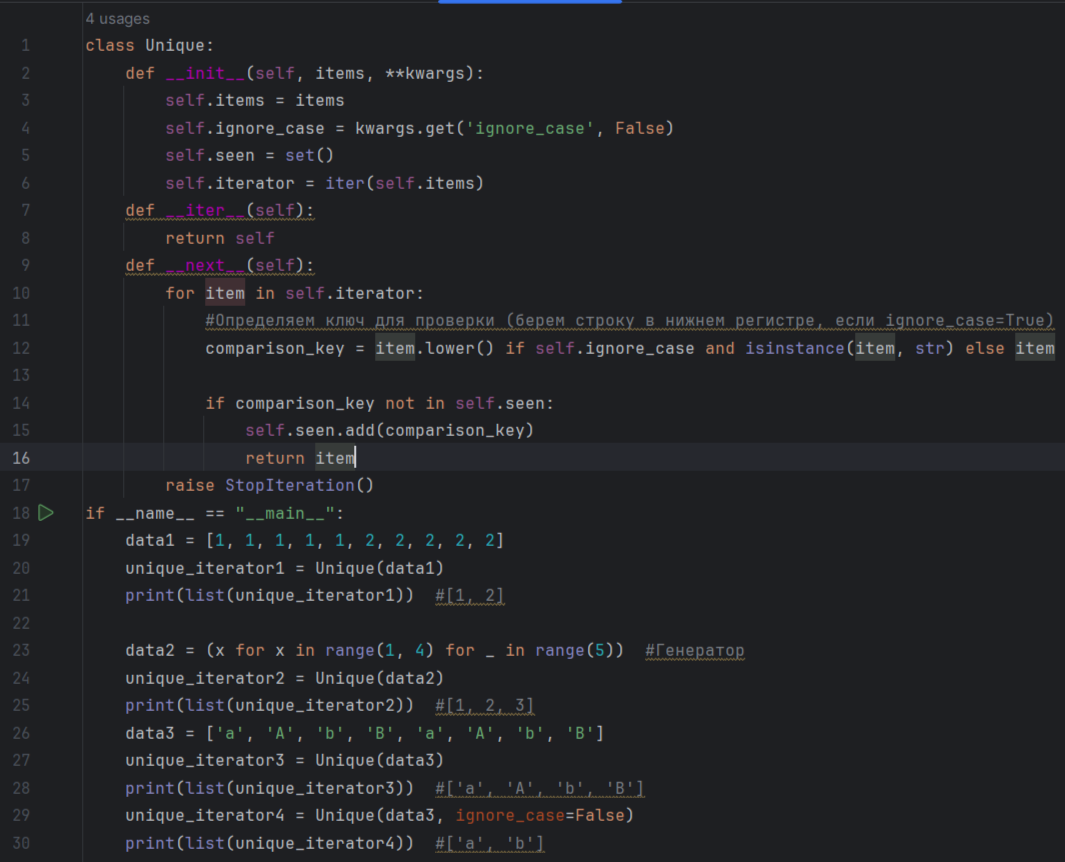
****

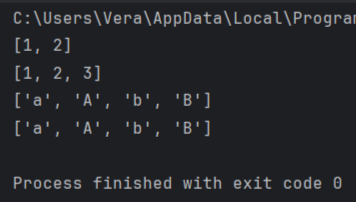
****

**Номер2**

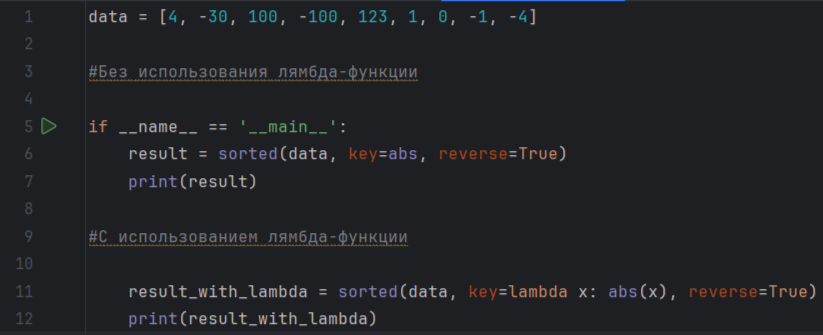
****

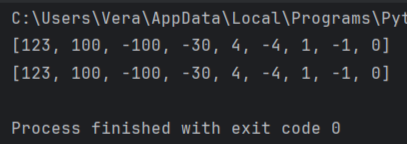
**Номер3**

****

****

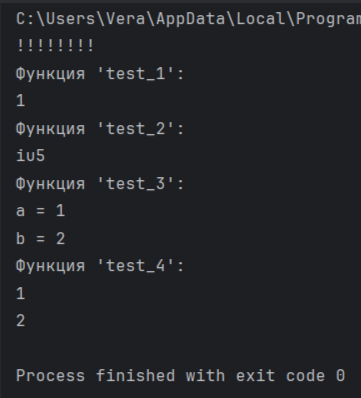
**Номер4**

****

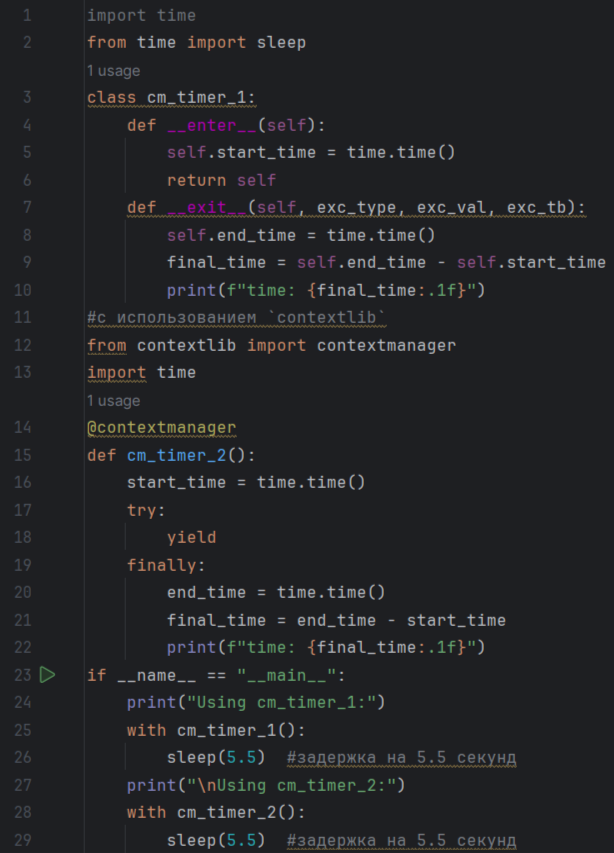
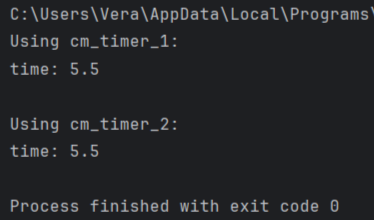
****

**Номер5**

def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 result = func(\*args, \*\*kwargs)  
 print(f"Функция '{func.\_\_name\_\_}':")  
 if isinstance(result, list):  
 for item in result:  
 print(item)  
 elif isinstance(result, dict):  
 for key, value in result.items():  
 print(f"{key} = {value}")  
 else:  
 print(result)  
 return result  
 return wrapper  
@print\_result  
def test\_1():  
 return 1  
@print\_result  
def test\_2():  
 return 'iu5'  
@print\_result  
def test\_3():  
 return {'a': 1, 'b': 2}  
@print\_result  
def test\_4():  
 return [1, 2]  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('!!!!!!!!')  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

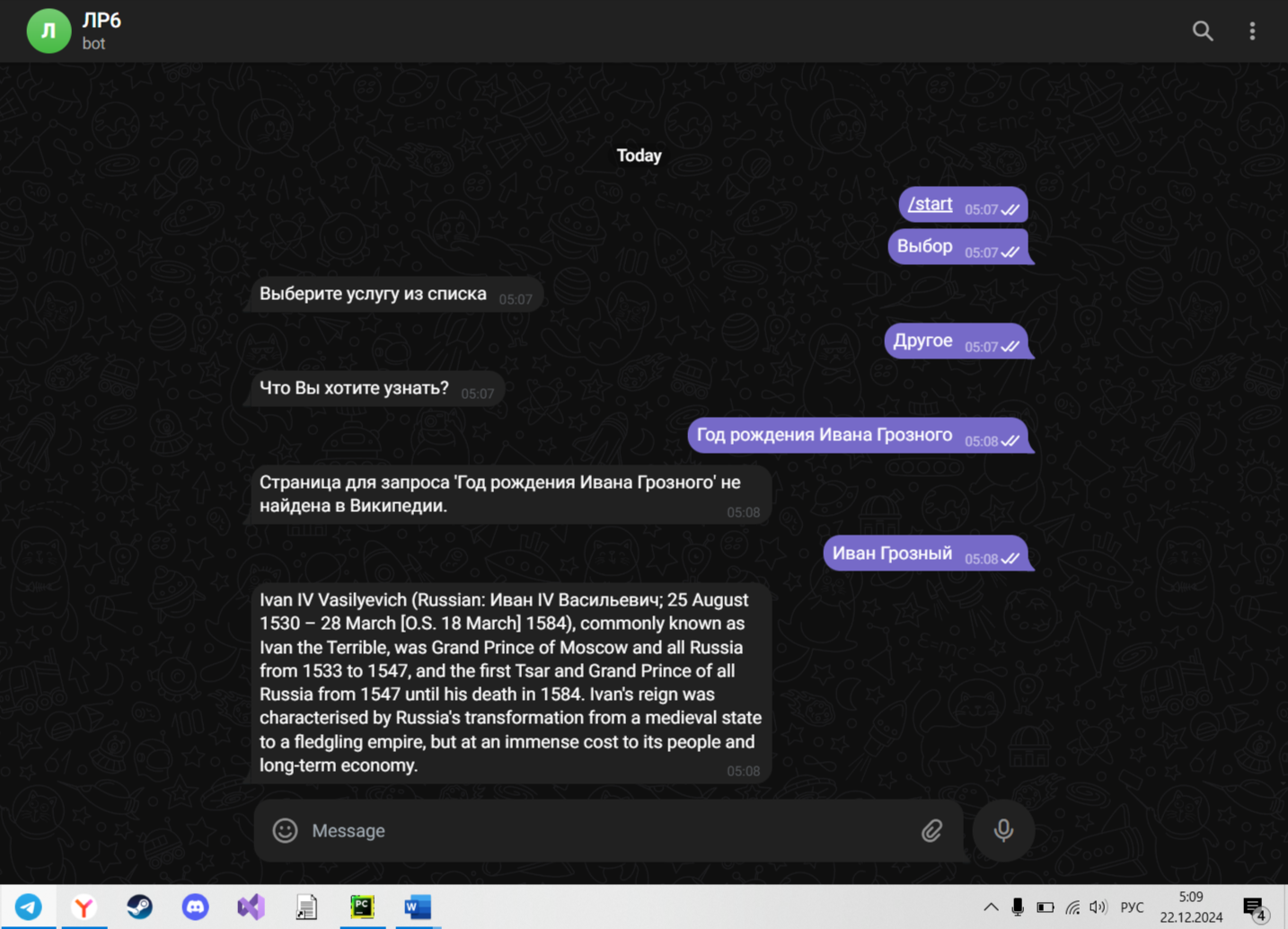


**Номер6**

**ЛР5-6**

**import asyncio  
from aiogram import Bot, Dispatcher, types  
from aiogram.filters import CommandStart  
from wikipedia import page, summary, exceptions  
import logging  
  
logging.basicConfig(level=logging.INFO) # для отладки  
logger = logging.getLogger(\_\_name\_\_)  
  
TOKEN = '7590644351:AAHxVq0Y5MEMg7mlQREWQWzHgKOHpNqD9KE'  
  
bot = Bot(token=TOKEN)  
dp = Dispatcher()  
  
  
def create\_keyboard():  
 keyboard = types.ReplyKeyboardMarkup(resize\_keyboard=True)  
 keyboard.add("Выбор")  
 keyboard.add("Очистить историю")  
 keyboard.add("Узнать о(б)")  
 keyboard.add("Другое")  
 return keyboard  
  
  
@dp.message(CommandStart())  
async def send\_welcome(message: types.Message):  
 await message.reply("Привет!", reply\_markup=create\_keyboard())  
  
  
@dp.message(lambda message: message.text in ["Выбор", "Очистить историю", "Другое"])  
async def handle\_service\_selection(message: types.Message):  
 if message.text == "Выбор":  
 await message.answer("Выберите услугу из списка")  
 elif message.text == "Очистить историю":  
 await message.answer("История очищена.")  
 elif message.text == "Другое":  
 await message.answer("Что Вы хотите узнать?")  
  
  
@dp.message(lambda message: message.text == "Узнать о(б)")  
async def handle\_wikipedia\_search(message: types.Message):  
 await message.answer("Введите запрос для поиска в Википедии:")  
  
  
@dp.message(  
 lambda message: message.text != "Выбор" and message.text != "Очистить историю" and message.text != "Узнать о(б)" and message.text != "Другое")  
async def handle\_wikipedia\_query(message: types.Message):  
 query = message.text  
 try:  
 result = summary(query, sentences=2) # 2 предложения из Википедии  
 await message.answer(result)  
 except exceptions.PageError:  
 await message.answer(f"Страница для запроса '{query}' не найдена в Википедии.")  
 except exceptions.DisambiguationError as e:  
 await message.answer(  
 f"Запрос '{query}' неоднозначен. Попробуйте уточнить запрос. Возможные варианты: {', '.join(e.options)}")  
 except Exception as e:  
 logger.exception(f"Ошибка при поиске в Википедии: {e}") # Логирование ошибок  
 await message.answer(f"Произошла ошибка при поиске: {e}")  
  
  
async def main():  
 await dp.start\_polling(bot)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 asyncio.run(main())**

****