1. **Напишите программу, которая запрашивает у пользователя два**

**числа, затем предлагает пользователю выбрать операцию (сложение,**

**вычитание, умножение или деление) и выводит результат**

**выбранной операции**

|  |
| --- |
| # Функция для выполнения операции сложения  def add(x, y):  return x + y  # Функция для выполнения операции вычитания  def subtract(x, y):  return x - y  # Функция для выполнения операции умножения  def multiply(x, y):  return x \* y  # Функция для выполнения операции деления  def divide(x, y):  # Добавим проверку деления на ноль  if y != 0:  return x / y  else:  return "Ошибка: деление на ноль"  # Запрос у пользователя двух чисел  num1 = float(input("Введите первое число: "))  num2 = float(input("Введите второе число: "))  # Вывод меню операций  print("Выберите операцию:")  print("1. Сложение")  print("2. Вычитание")  print("3. Умножение")  print("4. Деление")  # Запрос у пользователя выбора операции  choice = input("Введите номер операции (1/2/3/4): ")  # Выполнение выбранной операции и вывод результата  if choice == '1':  result = add(num1, num2)  print(f"{num1} + {num2} = {result}")  elif choice == '2':  result = subtract(num1, num2)  print(f"{num1} - {num2} = {result}")  elif choice == '3':  result = multiply(num1, num2)  print(f"{num1} \* {num2} = {result}")  elif choice == '4':  result = divide(num1, num2)  print(f"{num1} / {num2} = {result}")  else:  print("Неверный ввод. Пожалуйста, выберите операцию снова.") |

Эта программа сначала запрашивает у пользователя два числа, затем предоставляет меню операций (сложение, вычитание, умножение, деление) и выполняет выбранную операцию, выводя результат.

1. **Напишите программу, которая создает список, заполняет его случайными элементами, и сохраняет этот список в текстовом файле.**

|  |
| --- |
| import random  # Генерация случайного списка  random\_list = [random.randint(1, 100) for \_ in range(10)]  # Имя файла для сохранения списка  file\_name = "random\_list.txt"  # Запись списка в текстовый файл  with open(file\_name, "w") as file:  for number in random\_list:  file.write(str(number) + "\n")  print(f"Список сохранен в файле {file\_name}") |

В этом примере создается список из 10 случайных целых чисел от 1 до 100. Затем список сохраняется в текстовом файле "random\_list.txt". Каждый элемент списка записывается в отдельной строке файла.

Вы можете изменить размер списка, диапазон случайных чисел и имя файла, в соответствии с вашими требованиями.

1. **Создайте базовый класс &quot;Транспортное средство&quot; с методами для вычисления максимальной скорости и вместимости. Затем создайте производные классы, представляющие разные виды транспорта (например, &quot;Автомобиль&quot; и &quot;Самолет&quot;), и реализуйте соответствующие методы для каждого виде транспорта**

|  |
| --- |
| class TransportVehicle:  def \_\_init\_\_(self, brand, model):  self.brand = brand  self.model = model  def calculate\_max\_speed(self):  raise NotImplementedError("Метод должен быть переопределен в производном классе")  def calculate\_capacity(self):  raise NotImplementedError("Метод должен быть переопределен в производном классе")  class Car(TransportVehicle):  def \_\_init\_\_(self, brand, model, num\_doors):  super().\_\_init\_\_(brand, model)  self.num\_doors = num\_doors  def calculate\_max\_speed(self):  # Предположим, что для автомобиля максимальная скорость зависит от количества дверей  return 200 + self.num\_doors \* 10  def calculate\_capacity(self):  # Предположим, что вместимость автомобиля зависит от количества дверей  return 4 + self.num\_doors  class Airplane(TransportVehicle):  def \_\_init\_\_(self, brand, model, wingspan):  super().\_\_init\_\_(brand, model)  self.wingspan = wingspan  def calculate\_max\_speed(self):  # Предположим, что для самолета максимальная скорость зависит от размаха крыльев  return 800 + self.wingspan \* 20  def calculate\_capacity(self):  # Предположим, что вместимость самолета зависит от размаха крыльев  return 100 + self.wingspan // 2  # Пример использования классов  car\_instance = Car("Toyota", "Camry", 4)  print(f"Автомобиль {car\_instance.brand} {car\_instance.model}:")  print(f"Максимальная скорость: {car\_instance.calculate\_max\_speed()} км/ч")  print(f"Вместимость: {car\_instance.calculate\_capacity()} человек\n")  airplane\_instance = Airplane("Boeing", "747", 60)  print(f"Самолет {airplane\_instance.brand} {airplane\_instance.model}:")  print(f"Максимальная скорость: {airplane\_instance.calculate\_max\_speed()} км/ч")  print(f"Вместимость: {airplane\_instance.calculate\_capacity()} человек") |

В этом примере класс **TransportVehicle** является базовым классом с методами **calculate\_max\_speed** и **calculate\_capacity**, которые предоставляют интерфейс для вычисления максимальной скорости и вместимости транспортного средства. Классы **Car** и **Airplane** являются производными классами, которые реализуют эти методы с учетом своих особенностей.

1. **Создайте модель «Product». Модель должна содержать следующие поля: «name», «price», «description» и «quantity». Определите соответствующие типы полей и их параметры. Затем создайте миграции для этой модели и примените их к базе данных.**

Для создания модели "Product" в Django и выполнения миграций, вам нужно создать новое Django-приложение (если у вас его еще нет) и определить модель внутри этого приложения. Вот пример того, как вы можете сделать это:

1. Создайте новое Django-приложение:

|  |
| --- |
| python manage.py startapp myapp |

Где **myapp** - это имя вашего нового приложения.

1. Внутри вашего приложения определите модель "Product" в файле **models.py**. Пример может выглядеть так:

|  |
| --- |
| # myapp/models.py  from django.db import models  class Product(models.Model):  name = models.CharField(max\_length=255)  price = models.DecimalField(max\_digits=10, decimal\_places=2)  description = models.TextField()  quantity = models.IntegerField()  def \_\_str\_\_(self):  return self.name |

Здесь **CharField** используется для строки (названия), **DecimalField** для цены (десятичное число), **TextField** для описания и **IntegerField** для количества. **max\_length** указывает максимальную длину строки в случае **CharField**.

1. Теперь выполните миграции и примените их к базе данных:

|  |
| --- |
| python manage.py makemigrations  python manage.py migrate |

Эти команды создадут файл миграции, который содержит изменения в вашей модели, а затем применят эти изменения к базе данных.

Теперь ваша модель "Product" определена и готова к использованию в Django-приложении. Вы можете добавлять, извлекать, изменять и удалять объекты "Product" через административный интерфейс Django или через свой собственный код в вашем приложении.

1. **На основе модели Product создайте форму для добавления данных**

**о продуктах в базу данных. Форма должна включать поля для ввода**

**&quot;name&quot;, &quot;price&quot;, &quot;description&quot; и &quot;quantity&quot;. Реализуйте представление**

**для обработки данных из этой формы и сохранения их в базе**

**данных.**

Для создания формы и представления в Django, вы можете использовать встроенные средства Django, такие как **forms** и **views**. Вот пример того, как вы можете создать форму и представление на основе модели "Product":

1. В вашем приложении (в том же, где у вас модель "Product"), создайте файл **forms.py** и определите в нем форму:

|  |
| --- |
| # myapp/forms.py  from django import forms  from .models import Product  class ProductForm(forms.ModelForm):  class Meta:  model = Product  fields = ['name', 'price', 'description', 'quantity'] |

Теперь создайте представление в файле **views.py**, которое будет обрабатывать данные из формы и сохранять их в базе данных:

|  |
| --- |
| # myapp/views.py  from django.shortcuts import render, redirect  from .forms import ProductForm  def add\_product(request):  if request.method == 'POST':  form = ProductForm(request.POST)  if form.is\_valid():  form.save()  return redirect('success') # Перенаправление на страницу успешного добавления продукта  else:  form = ProductForm()  return render(request, 'add\_product.html', {'form': form}) |

Создайте шаблон для формы в папке **templates** вашего приложения. Создайте файл **add\_product.html** и определите в нем форму:

|  |
| --- |
| <!-- myapp/templates/add\_product.html -->  {% extends 'base.html' %}  {% block content %}  <h2>Add Product</h2>  <form method="post" action="{% url 'add\_product' %}">  {% csrf\_token %}  {{ form.as\_p }}  <button type="submit">Add Product</button>  </form>  {% endblock %} |

Не забудьте создать URL-маршрут для вашего представления в файле **urls.py** вашего приложения:

|  |
| --- |
| # myapp/urls.py  from django.urls import path  from .views import add\_product  urlpatterns = [  path('add\_product/', add\_product, name='add\_product'),  # Другие URL-маршруты вашего приложения...  ] |

Наконец, убедитесь, что ваше приложение подключено в **INSTALLED\_APPS** в файле **settings.py**:

|  |
| --- |
| # settings.py  INSTALLED\_APPS = [  # ...  'myapp',  # ...  ] |

Теперь, при переходе по URL '/add\_product/', вы увидите форму для добавления продукта, и данные будут сохраняться в базе данных при ее отправке.