Вопросы к экзамену по модулю ПМ.05

1) Что такое СКД? Как создать?

- СКД (система классификации данных) - это структурированный способ организации информации для удобства и эффективности ее использования. Для создания СКД необходимо определить категории данных, атрибуты и связи между ними. Структура создается с учетом конкретных потребностей и целей организации, которая будет использовать эту систему.

2) Что такое запрос? Как написать запрос?

- Запрос - это инструкция к базе данных для получения или изменения данных в соответствии с определенными критериями.

- Пример запроса на SQL для выбора данных из таблицы:

```sql

SELECT \* FROM table\_name WHERE condition;

```

3) Какие существуют циклы в 1С? Написать их синтаксис

- В 1С существуют циклы Для каждого, Пока, Цикл По, Для Каждого Символа в Строке.

- Примеры синтаксиса:

```1c

// Цикл "Для каждого"

Для каждого Элемент Из Коллекция Цикл

// Код

КонецЦикла;

// Цикл "Пока"

Пока Условие Цикл

// Код

КонецЦикла;

```

4) Что такое процедура и функция? Написать их синтаксис

- Процедура - это блок кода, выполняющий определенные действия, не возвращающий значение. Функция - это блок кода, выполняющий определенные действия и возвращающий значение.

- Примеры синтаксиса:

```1c

// Процедура

Процедура ИмяПроцедуры(Параметры)

// Код

КонецПроцедуры;

// Функция

Функция ИмяФункции(Параметры) Экспорт

// Код

Возврат Результат;

КонецФункции;

```

5) Что такое события? Где находятся все события формы? Что такое обработчик события?

- Событие - это действие, которое происходит в приложении или форме при определенных условиях (например, клик мыши).

- В 1С все события формы можно найти в реквизите объекта формы.

- Обработчик события - это блок кода, который выполняется при возникновении определенного события. Обработчики событий могут быть назначены на различные элементы управления формы.

6) Что такое печатная форма? С помощью чего выполняется вывод печатной формы? Что представляет макет табличного документа?

- Печатная форма - это представление данных для вывода на печать или в виде документа.

- Вывод печатной формы осуществляется с помощью механизма отчетов и обработок вывода.

- Макет табличного документа представляет собой шаблон расположения полей и элементов для вывода данных на печатную форму.

7) Что такое планы видов характеристик? Когда их лучше всего использовать?

- Планы видов характеристик - это инструмент для группировки и хранения наборов информации о характеристиках товаров или услуг.

- Их лучше всего использовать для управления и организации информации о характеристиках товаров в системе учета или торговли.

8) Общие сведения о формах различных объектов.

- Формы представляют собой графические элементы интерфейса для ввода и вывода данных в информационных системах. Например, формы объектов справочников, документов, справочники, отчеты.

9) Командный интерфейс формы. Виды команд.

- Командный интерфейс формы предоставляет доступ к основным командам и действиям при работе с формой.

- Виды команд: действия над записью, навигационные команды, системные команды, команды работы с отчетностью.

10) Из чего состоит редактор управляемой формы?

- Редактор управляемой формы состоит из шаблонов расположения элементов управления, настраиваемых свойств элементов, макетов формы, событий и обработчиков.

11) Что такое модуль? Какие бывают модули?

- Модуль - это единица кода, которая выполняет определенные функции или процессы.

- Бывают модули обработки, модули объектов метаданных, модули управляемых форм, модули внешних обработок.

12) Дайте определение понятию «Тестирование». Перечислите цели тестирования;

- Тестирование - это процесс проверки программного продукта на соответствие его спецификации и выявление ошибок.

- Цели тестирования: выявление ошибок, обеспечение качеств

а программного продукта, повышение надежности и стабильности приложения.

13) Дайте определение понятию «Тестировщик». Что должен знать и уметь тестировщик?

- Тестировщик - это специалист, отвечающий за проведение тестирования программного обеспечения.

- Тестировщик должен знать методы тестирования, инструменты тестирования, иметь навыки анализа, логического мышления, умение составлять тест-кейсы.

14) Виды тестирования. Перечислите функциональные виды тестирования;

Виды функциональных тестов:

* Функциональное тестирование (Functional testing);
* Тестирование безопасности (Security and Access Control Testing);
* Тестирование взаимодействия (Interoperability Testing).

15) Виды тестирования. Перечислите нефункциональные виды тестирования;

Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. **В целом, это тестирование того, "Как" система работает.** **Виды нефункциональных тестов:**

* **Все виды тестирования производительности:**
  + **нагрузочное тестирование (Performance and Load Testing);**
  + **стрессовое тестирование (Stress Testing);**
  + **тестирование стабильности или надежности (Stability / Reliability Testing);**
  + **объемное тестирование (Volume Testing).**
* **Тестирование установки (Installation testing);**
* **Тестирование удобства пользования (Usability Testing);**
* **Тестирование на отказ и восстановление (Failover and Recovery Testing);**
* **Конфигурационное тестирование (Configuration Testing).**

16) Виды тестирования. Виды тестов, связанные с изменениями;

- Тестирование изменений включает: регрессионное тестирование, тестирование воспроизведения, адаптационное тестирование.

17) Уровни тестирования. Модульное тестирование.

- **Компонентное (модульное) тестирование проверяет функциональность и ищет дефекты в частях приложения**, которые доступны и могут быть протестированы по-отдельности (**модули программ, объекты, классы, функции и т.д.**). Обычно компонентное (модульное) тестирование проводится вызывая код, который необходимо проверить и при поддержке сред разработки, таких как фреймворки для модульного тестирования или инструменты для отладки.

Один из наиболее эффективных подходов к компонентному (модульному) тестированию - это **подготовка автоматизированных тестов** до начала основного кодирования (разработки) программного обеспечения. Это называется разработка от тестирования (**test-driven development**) или подход тестирования вначале (**test first approach**). При этом подходе создаются и интегрируются небольшие куски кода, напротив которых запускаются тесты, написанные до начала кодирования. Разработка ведется до тех пор пока все тесты не будут успешно пройдены.

18) Уровни тестирования. Системное тестирование;

- Основной задачей системного тестирования является проверка как функциональных, так и нефункциональных требований в системе в целом. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д.

Для минимизации рисков, связанных с особенностями поведения системы в той или иной среде, во время тестирования рекомендуется использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи.

**Можно выделить два подхода к системному тестированию:**

* **на базе требований (requirements based)**
  + Для каждого требования пишутся тестовые случаи (test cases), проверяющие выполнение данного требования.
* **на базе случаев использования (use case based)**
  + На основе представления о способах использования продукта создаются случаи использования системы (Use Cases). По конкретному случаю использования можно определить один или более сценариев. На проверку каждого сценария пишутся тест кейсы (test cases), которые должны быть протестированы.

19) Уровни тестирования. Интеграционное тестирование;

* **Интеграционное тестирование** предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами). Компонентный интеграционный уровень (Component Integration testing)
  + Проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.
* Системный интеграционный уровень (System Integration Testing)
  + Проверяется взаимодействие между разными системами после проведения системного тестирования.
* Снизу вверх (Bottom Up Integration)
* Сверху вниз (Top Down Integration)
* Большой взрыв ("Big Bang" Integration)

20) Уровни тестирования. Приемочное тестирование;

- **Приемочное тестирование или Приемо-сдаточное испытание (Acceptance Testing)** - формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью:

* определения удовлетворяет ли система приемочным критериям;
* вынесения решения заказчиком или другим уполномоченным лицом принимается приложение или нет.

Приемочное **тестирование выполняется на основании набора типичных тестовых случаев и сценариев,** разработанных на основании требований к данному приложению.

Решение о проведении приемочного тестирования принимается, когда:

* продукт достиг необходимого уровня качества;
* заказчик ознакомлен с **Планом Приемочных Работ (Product Acceptance Plan)** или иным документом, где описан набор действий, связанных с проведением приемочного тестирования, дата проведения, ответственные и т.д.

Фаза приемочного тестирования длится до тех пор, пока заказчик не выносит решение об отправлении приложения на доработку или выдаче приложения.

21) Дайте определение понятиям «Дефект» и «Отказ». Почему появляются ошибки?

- Дефект - это недостаток или ошибка в программном продукте, который может привести к отказу в работе.

- Отказ - это ситуация, когда программный продукт не выполняет свои функции правильно.

- Ошибки появляются из-за неправильного проектирования, реализации или недостатка тестирования.

22) Что такое отчеты об ошибках? Структура отчета об ошибке;

- Отчет об ошибке - это документ, в котором описана найденная проблема в программном продукте.

- Структура отчета об ошибке: описание проблемы, шаги для воспроизведения, ожидаемое и фактическое поведение, приоритет ошибки.

23) Автоматизированное тестирование. Цель автоматизации. Преимущества и недостатки автоматизированного тестирования;

- Цель автоматизации - увеличение эффективности и скорости процесса тестирования.

- Преимущества: повышение производительности, повторяемость тестов, раннее обнаружение ошибок. Недостатки: высокие затраты на разработку и поддержку автотестов.

24) Три подхода к тестированию программного обеспечения. Тестирование белого ящика;

- Тестирование белого ящика - это вид тестирования, при котором тестируются структуры, архитектура и код программного продукта.

25) Три подхода к тестированию программного обеспечения. Тестирование черного ящика;

- Тестирование черного ящика - это вид тестирования, при котором проверяется функциональность программного продукта без знания его внутренней структуры.

26) Три подхода к тестированию программного обеспечения. Тестирование серого ящика;

- Тестирование серого ящика - это комбинация тестирования белого и черного ящиков, при котором тестируется функциональность переднего и внутреннего кода программы.

Практические задания к экзамену по модулю ПМ.05

django.VERSION

скачиваем Django с помощью команды – pip install Django

django-admin startproject name

Далее подключаемся к нашему проекту – cd flowershop

И запускаем наш проект командой – python manage.py runserver

1. Измените код Django таким образом, чтобы на странице отображалась информация: наименование группы, ФИО, электронную почту.
2. Напишите код Django, который осуществляет переход по страницам сайта.

1. \*\*views.py:\*\*

from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

from django.shortcuts import render

from .models import YourModel

def your\_view(request):

all\_objects = YourModel.objects.all()

# Number of items per page

items\_per\_page = 10

paginator = Paginator(all\_objects, items\_per\_page)

page\_number = request.GET.get('page')

try:

paginated\_objects = paginator.page(page\_number)

except PageNotAnInteger:

paginated\_objects = paginator.page(1)

except EmptyPage:

paginated\_objects = paginator.page(paginator.num\_pages)

return render(request, 'your\_template.html', {'paginated\_objects': paginated\_objects})

2. \*\*your\_template.html:\*\*

<!-- Displaying paginated objects -->

{% for obj in paginated\_objects %}

<!-- Display your object details here -->

<p>{{ obj.title }}</p>

{% endfor %}

<!-- Pagination links -->

<div class="pagination">

<span class="step-links">

<!-- Previous page link -->

{% if paginated\_objects.has\_previous %}

<a href="?page=1">&laquo; first</a>

<a href="?page={{ paginated\_objects.previous\_page\_number }}">previous</a>

{% endif %}

<!-- Page numbers -->

<span class="current">

Page {{ paginated\_objects.number }} of {{ paginated\_objects.paginator.num\_pages }}.

</span>

<!-- Next page link -->

{% if paginated\_objects.has\_next %}

<a href="?page={{ paginated\_objects.next\_page\_number }}">next</a>

<a href="?page={{ paginated\_objects.paginator.num\_pages }}">last &raquo;</a>

{% endif %}

</span>

</div>

В этом примере мы используем Paginator для разделения списка объектов на страницы и отображения ссылок для перехода между страницами на вашем сайте. При переходе между страницами параметр ?page= добавляется к URL, указывая номер страницы.

Не забудьте настроить URL-адреса и шаблоны для вашего представления. После внедрения этого кода у вас должна быть рабочая пагинация на вашем сайте Django.

1. Установите библиотеку django-bootstrap/аналог и продемонстрируйте его работу.

1. \*\*Установка библиотеки:\*\*

Выполните следующую команду для установки django-bootstrap4:

pip install django-bootstrap4

2. \*\*Интеграция в Django проект:\*\*

- Добавьте 'bootstrap4' в INSTALLED\_APPS вашего settings.py.

- В шаблонах используйте теги и фильтры из bootstrap4 для форматирования.

3. \*\*Пример работы с django-bootstrap4:\*\*

# views.py

from django.shortcuts import render

from django.http import HttpResponse

from django import forms

class ContactForm(forms.Form):

subject = forms.CharField()

email = forms.EmailField()

message = forms.CharField(widget=forms.Textarea)

def contact\_view(request):

form = ContactForm()

return render(request, 'contact.html', {'form': form})

# contact.html

{% extends 'base.html' %}

{% block content %}

<h2>Contact Us</h2>

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit" class="btn btn-primary">Submit</button>

</form>

{% endblock %}

4. \*\*base.html:\*\*

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>My Django Bootstrap App</title>

{% load bootstrap4 %}

{% bootstrap\_css %}

</head>

<body>

<div class="container">

{% block content %}

{% endblock %}

</div>

{% bootstrap\_javascript %}

</body>

</html>

1. Создайте меню навигации используя django-bootstrap/аналог и продемонстрируйте её работу.

<!-- base.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>My Django Bootstrap App</title>

{% load bootstrap4 %}

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-light bg-light">

<a class="navbar-brand" href="#">My Site</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<li class="nav-item active">

<a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">About</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Services</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Contact</a>

</li>

</ul>

</div>

</nav>

<div class="container">

{% block content %}

{% endblock %}

</div>

{% bootstrap\_javascript %}

</body>

</html>

2. \*\*Пример использования базового шаблона:\*\*

{% extends 'base.html' %}

{% block content %}

<h1>Welcome to My Site</h1>

{% endblock %}

1. Создайте простую модель и сделайте вывод данных из этой модели

1. \*\*Создание модели\*\*:

Откройте файл models.py в вашем Django приложении и определите простую модель. Например, создадим модель Product с двумя полями name и price:

# models.py

from django.db import models

class Product(models.Model):

name = models.CharField(max\_length=100)

price = models.FloatField()

2. \*\*Создание и применение миграций\*\*:

После создания модели, сгенерируйте миграции и примените их, чтобы создать таблицу в базе данных:

python manage.py makemigrations

python manage.py migrate

3. \*\*Добавление данных\*\*:

Вы можете добавить данные в вашу модель либо через административный интерфейс Django, либо через код. Например, добавим данные через оболочку Django:

python manage.py shell

from yourapp.models import Product

# Создание объектов Product

Product.objects.create(name='Product1', price=10.99)

Product.objects.create(name='Product2', price=20.49)

4. \*\*Вывод данных из модели\*\*:

Теперь вы можете вывести данные из вашей модели Product. Для этого создайте вид представления, который будет передавать объекты Product в шаблон:

# views.py

from django.shortcuts import render

from yourapp.models import Product

def product\_list(request):

products = Product.objects.all()

return render(request, 'product\_list.html', {'products': products})

5. \*\*Шаблон для отображения данных\*\*:

Создайте шаблон product\_list.html, где вы будете выводить данные Product:

<!-- product\_list.html -->

<!DOCTYPE html>

<html>

<head>

<title>Product List</title>

</head>

<body>

<h1>Product List</h1>

<ul>

{% for product in products %}

<li>{{ product.name }} - ${{ product.price }}</li>

{% endfor %}

</ul>

</body>

</html>

6. \*\*Настройка URL маршрута\*\*:

Не забудьте добавить URL маршрут для вашего представления в файле urls.py вашего приложения:

# urls.py

from django.urls import path

from . import views

urlpatterns = [

path('product-list/', views.product\_list, name='product\_list'),

]

Теперь, при переходе на маршрут /product-list/, вы увидите список ваших продуктов, отобранных из модели Product. Если у вас возникнут вопросы, не стесняйтесь задавать!

1. Создайте простую модель и запишите туда данные. Не используйте для выполнения работы панель-администратора.

Давайте создадим простую модель в Django, запишем данные в нее из кода, обходя панель администратора. Для этого выполним следующие шаги:

1. \*\*Создание модели и запись данных из кода\*\*:

2. \*\*Создайте модель.\*\*

Откройте файл models.py в вашем Django приложении и создайте простую модель. Давайте создадим модель Book с полями title и author:

# models.py

from django.db import models

class Book(models.Model):

title = models.CharField(max\_length=100)

author = models.CharField(max\_length=50)

3. \*\*Используйте оболочку Django для создания и записи данных\*\*:

Воспользуйтесь интерактивной оболочкой Django для создания объектов Book и сохранения их в базе данных:

python manage.py shell

from yourapp.models import Book

# Создаем и сохраняем объекты Book

book1 = Book(title='Book Title 1', author='Author 1')

book1.save()

book2 = Book(title='Book Title 2', author='Author 2')

book2.save()

4. \*\*Проверка записанных данных:\*\*

Вы можете проверить данные, вызвав их из оболочки Django:

# Проверка данных

books = Book.objects.all()

for book in books:

print(book.title, "-", book.author)

Таким образом, вы создали модель Book, добавили в нее две записи и вывели их с помощью оболочки Django, не используя панель администратора.

1. Создайте форму регистрации, соблюдая валидацию данных. Добавьте запись данных в модель.

1. \*\*Создание формы регистрации\*\*:

Создайте файл forms.py в вашем Django приложении и определите форму регистрации с использованием forms.ModelForm. Давайте создадим простую форму для регистрации пользователей с полями username, email и password:

# forms.py

from django import forms

from yourapp.models import User

class RegistrationForm(forms.ModelForm):

password = forms.CharField(widget=forms.PasswordInput)

class Meta:

model = User

fields = ['username', 'email', 'password']

def clean\_email(self):

email = self.cleaned\_data['email']

if User.objects.filter(email=email).exists():

raise forms.ValidationError("Email already exists. Please use another email.")

return email

2. \*\*Обработка данных формы и добавление записи в модель\*\*:

Создайте вид представления, который будет отображать и обрабатывать форму регистрации, а затем добавлять запись в модель User:

# views.py

from django.shortcuts import render, redirect

from yourapp.forms import RegistrationForm

def register(request):

if request.method == 'POST':

form = RegistrationForm(request.POST)

if form.is\_valid():

user = form.save()

return redirect('registration\_success') # Redirect to a success page

else:

form = RegistrationForm()

return render(request, 'register.html', {'form': form})

def registration\_success(request):

return render(request, 'registration\_success.html')

3. \*\*Создание шаблонов\*\*:

Создайте шаблоны register.html и registration\_success.html для отображения формы регистрации и страницы успешной регистрации соответственно.

4. \*\*Настройка маршрутов URL\*\*:

Настройте маршруты URL для ваших представлений в файле urls.py вашего приложения:

# urls.py

from django.urls import path

from yourapp import views

urlpatterns = [

path('register/', views.register, name='register'),

path('registration-success/', views.registration\_success, name='registration\_success'),

]

1. Создайте супер-пользователя, использовав консоль, а также обычного пользователя, использовав панель администратора.

1. \*\*Создание суперпользователя через консоль Django\*\*:

Для создания суперпользователя откройте терминал и выполните следующую команду:

python manage.py createsuperuser

Затем введите запрошенную информацию, такую как имя пользователя, электронную почту и пароль.

2. \*\*Создание обычного пользователя через панель администратора\*\*:

- Зайдите в панель администратора Django, обычно по адресу http://yourdomain.com/admin/.

- В разделе Authentication and Authorization выберите Users, затем нажмите на кнопку Add user.

- Укажите необходимые данные для создания нового пользователя: имя, электронную почту, пароль и другие данные при необходимости.

- Нажмите кнопку Save для создания нового пользователя.

После выполнения этих шагов у вас будет создан суперпользователь и обычный пользователь в вашем проекте Django. Суперпользователь имеет права администратора и может управлять всеми аспектами приложения, в то время как обычный пользователь будет иметь ограниченный доступ.

1. Продемонстрируйте работу ограничения прав для администраторов и обычных пользователей, используйте панель администратора.

1. \*\*Работа с правами доступа\*\*:

Для ограничения доступа к модели Book, отредактируем файл admin.py вашего приложения. Мы можем добавить дополнительную логику для определения прав доступа в зависимости от роли пользователя.

# admin.py

from django.contrib import admin

from .models import Book

class BookAdmin(admin.ModelAdmin):

def has\_add\_permission(self, request):

if request.user.is\_superuser:

return True

return False

def has\_change\_permission(self, request, obj=None):

if request.user.is\_superuser:

return True

if obj is not None and obj.author == request.user: # Пример: разрешить изменение только автору книги

return True

return False

def has\_delete\_permission(self, request, obj=None):

if request.user.is\_superuser:

return True

return False

admin.site.register(Book, BookAdmin)

2. \*\*Демонстрация работы ограничений\*\*:

- Зайдите в панель администратора Django.

- Попробуйте добавить, изменить и удалить объекты модели Book как администратор и как обычный пользователь.

- Вы увидите, что в соответствии с определенными правами доступа, у администратора будет доступ ко всем функциям, в то время как у обычного пользователя будут ограничены определенные действия.

Это лишь пример того, как можно настроить ограничения прав доступа для разных типов пользователей в Django через панель администратора. Вы можете создать более сложные ограничения и логику в зависимости от ваших потребностей.

1. Создайте форму для подсчёта периметра и площади треугольника, используя модель Django.

1. \*\*Создание модели треугольника\*\*:

Сначала создадим модель Triangle в файле models.py вашего Django приложения для хранения длин сторон треугольника:

# models.py

from django.db import models

class Triangle(models.Model):

side\_a = models.IntegerField()

side\_b = models.IntegerField()

side\_c = models.IntegerField()

2. \*\*Создание формы Django\*\*:

Создадим форму TriangleForm для ввода длин сторон треугольника и расчета его параметров в файле forms.py вашего приложения:

# forms.py

from django import forms

class TriangleForm(forms.Form):

side\_a = forms.IntegerField(label='Side A')

side\_b = forms.IntegerField(label='Side B')

side\_c = forms.IntegerField(label='Side C')

3. \*\*Создание представления для обработки формы\*\*:

Создадим представление calculate\_triangle в файле views.py:

# views.py

from django.shortcuts import render

from .forms import TriangleForm

def calculate\_triangle(request):

if request.method == 'POST':

form = TriangleForm(request.POST)

if form.is\_valid():

side\_a = form.cleaned\_data['side\_a']

side\_b = form.cleaned\_data['side\_b']

side\_c = form.cleaned\_data['side\_c']

perimeter = side\_a + side\_b + side\_c

s = perimeter / 2

area = (s \* (s - side\_a) \* (s - side\_b) \* (s - side\_c)) \*\* 0.5

return render(request, 'triangle\_result.html', {'perimeter': perimeter, 'area': area})

else:

form = TriangleForm()

return render(request, 'triangle\_form.html', {'form': form})

4. \*\*Создание шаблонов для формы и результатов\*\*:

Создадим шаблоны triangle\_form.html и triangle\_result.html для отображения формы и результатов расчета соответственно.

5. \*\*Настройка маршрутов URL\*\*:

Настройте маршруты URL для представления calculate\_triangle в файле urls.py вашего приложения.

После выполнения этих шагов у вас будет форма, через которую пользователи смогут вводить длины сторон треугольника, а затем вычислять его периметр и площадь. Результаты будут отображаться на отдельной странице.

1. Создайте форму для подсчета среднего заработка за квартал, используя модель Django

1. \*\*Создание модели\*\*:

Создадим модель Salary в файле models.py вашего Django приложения для хранения ежемесячных заработков:

# models.py

from django.db import models

class Salary(models.Model):

month = models.CharField(max\_length=20)

amount = models.DecimalField(max\_digits=10, decimal\_places=2)

2. \*\*Создание формы Django\*\*:

Создадим форму SalaryForm для ввода заработков по месяцам в файле forms.py вашего приложения:

# forms.py

from django import forms

class SalaryForm(forms.Form):

january = forms.DecimalField(label='January')

february = forms.DecimalField(label='February')

march = forms.DecimalField(label='March')

3. \*\*Создание представления для обработки формы\*\*:

Создадим представление calculate\_average\_salary в файле views.py:

# views.py

from django.shortcuts import render

from .forms import SalaryForm

def calculate\_average\_salary(request):

if request.method == 'POST':

form = SalaryForm(request.POST)

if form.is\_valid():

january\_salary = form.cleaned\_data['january']

february\_salary = form.cleaned\_data['february']

march\_salary = form.cleaned\_data['march']

average\_salary = (january\_salary + february\_salary + march\_salary) / 3

return render(request, 'average\_salary\_result.html', {'average\_salary': average\_salary})

else:

form = SalaryForm()

return render(request, 'average\_salary\_form.html', {'form': form})

4. \*\*Создание шаблонов для формы и результатов\*\*:

Создадим шаблоны average\_salary\_form.html и average\_salary\_result.html для отображения формы и результатов расчета соответственно.

5. \*\*Настройка маршрутов URL\*\*:

Настройте маршруты URL для представления calculate\_average\_salary в файле urls.py вашего приложения.

После выполнения этих шагов у вас будет форма, через которую пользователи смогут вводить заработки по месяцам, а затем вычислять и отображать средний заработок за квартал.

1. 1. Прочитать из текстового файла строку "Hellow word!". Удалить каждый второй символ и записать в этот же файл. Выполнить рекурсивно. Составить блок-схему

def process\_file(file\_path, level=1):

with open(file\_path, 'r') as file:

text = file.read()

modified\_text = ''

for index, char in enumerate(text):

if (index + 1) % 2 != 0: # Удаляем каждый второй символ

modified\_text += char

with open(file\_path, 'w') as file:

file.write(modified\_text)

print(f'Level {level}: Processed {file\_path}')

# Делаем рекурсивный вызов для следующего уровня

if level > 1:

process\_file(file\_path, level - 1)

file\_path = 'example.txt'

process\_file(file\_path, 3) # Пример рекурсии на 3 уровня

Этот код читает строку из файла, удаляет каждый второй символ и записывает измененную строку в тот же файл. Затем он делает рекурсивные вызовы для обработки файла на каждом уровне. Уровень рекурсии указывается аргументом level.

Что касается блок-схемы, у меня, к сожалению, нет возможности предоставить графическое представление здесь, но вот общая идея:

1. Прочитать содержимое файла.

2. Удалить каждый второй символ.

3. Записать измененное содержимое обратно в файл.

4. Повторить шаги 1-3 на следующем уровне рекурсии (если level > 1).

1. 2. Дан текстовый файл, в котором хранится список группы. Информация о каждом студенте сохранена в отдельной строке. Составить блок-схему. Информация хранится в следующем формате: Фамилия Имя

1. Откройте файл и считайте каждую строку.

2. Разделите строку на фамилию и имя.

3. Обработайте и храните информацию о студентах (например, сохраните в список или словарь).

Теперь давайте создадим пример блок-схемы алгоритма для чтения данных из файла со списком студентов в формате "Фамилия Имя":

graph TB

A[Открыть файл] --> B{Есть ли еще строки?}

B -- Да --> C{Разделить на Фамилию и Имя}

C --> D[Сохранить информацию о студенте]

D --> B

B -- Нет --> E[Закрыть файл]

Вот код на Python, который реализует этот алгоритм:

students = []

with open('students.txt', 'r') as file:

for line in file:

data = line.strip().split()

if len(data) == 2:

surname, name = data

students.append({'surname': surname, 'name': name})

# Вывод информации о студентах

for student in students:

print(f"Фамилия: {student['surname']}, Имя: {student['name']}")

Этот код читает список студентов из файла students.txt, разделяет строки на фамилию и имя, и сохраняет информацию о каждом студенте в виде словаря в списке students. После успешного чтения информации из файла, он выводит информацию о студентах.

1. Составить программу, которая:

а) подсчитывает сколько студентов в группе имеют одинаковое с вами имя.

б) определяет есть ли в вашей группе однофамильцы.

Все найденные сведения вывести в файл. В формате:

Отчет составлен Ивановым Иваном.

Количество студентов в группе с именем - Иван - 3 студента.

В группе есть однофамилец - Иванов Петр.

1. Прочитать информацию о студентах из файла.

2. Провести анализ данных для подсчета студентов с одинаковыми именами и поиска однофамильцев.

3. Записать результаты в файл в заданном формате.

Вот пример кода на Python для выполнения этих задач:

def analyze\_students(students):

name\_counts = {}

surname\_counts = {}

has\_lastname\_match = False

for student in students:

name\_counts[student['name']] = name\_counts.get(student['name'], 0) + 1

surname\_counts[student['surname']] = surname\_counts.get(student['surname'], 0) + 1

with open('report.txt', 'w') as file:

file.write(f"Отчет составлен {students[0]['surname']} {students[0]['name']}.\n")

for name, count in name\_counts.items():

file.write(f"Количество студентов в группе с именем - {name} - {count} студента.\n")

for surname, count in surname\_counts.items():

if count > 1:

has\_lastname\_match = True

file.write(f"В группе есть однофамилец - {surname} {students[0]['name']}.\n")

if has\_lastname\_match:

print("Результаты анализа сохранены в файл 'report.txt'. В группе есть однофамилец.")

else:

print("Результаты анализа сохранены в файл 'report.txt'. В группе нет однофамильцев.")

# Пример списка студентов (можете изменить на чтение из файла)

students = [

{'name': 'Иван', 'surname': 'Иванов'},

{'name': 'Петр', 'surname': 'Иванов'},

{'name': 'Иван', 'surname': 'Петров'},

{'name': 'Анна', 'surname': 'Сидорова'},

{'name': 'Иван', 'surname': 'Смирнов'}

]

analyze\_students(students)

Этот код анализирует данные о студентах и подсчитывает количество студентов с одинаковыми именами, а также наличие однофамильцев в группе. Результаты записываются в файл 'report.txt'.

1. Создать файл, содержащий содержащим следующие сведения о торговом предприятии: код товара, наименование товара, фирма-поставщик, цена, количество, проданное за день. Составить блок-схему. Поиск осуществлять по следующим параметрам:

а) фирма-поставщик;

б) наименование товара и цена (диапазон);

в) фирма-поставщик, при этом определить общую стоимость проданных товаров данной фирмы и самый дорогой товар.

Для начала давай создадим файл, содержащий информацию о товарах. Мы можем использовать формат CSV (Comma Separated Values) для хранения этих данных. Вот как можно создать начальный файл products.csv:

код товара,наименование товара,фирма-поставщик,цена,количество,проданное за день

1,Шоколадка,Конфетки ООО,2.5,100,20

2,Молоко,Молочное господарство,1.8,200,50

3,Хлеб,Хлебопекарня "Хлебушек",1.0,150,30

4,Сок,Фруктовый рай,3.0,80,15

5,Яблоки,Фермерская гавань,2.0,120,25

Теперь давай напишем программу на Python для поиска информации о товаре по различным параметрам:

import csv

def search\_by\_supplier(supplier, filename):

with open(filename, 'r') as file:

reader = csv.DictReader(file)

for row in reader:

if row['фирма-поставщик'] == supplier:

print(row)

def search\_by\_name\_and\_price\_range(name, min\_price, max\_price, filename):

with open(filename, 'r') as file:

reader = csv.DictReader(file)

for row in reader:

if row['наименование товара'] == name and min\_price <= float(row['цена']) <= max\_price:

print(row)

def search\_by\_supplier\_details(supplier, filename):

total\_price = 0

max\_price = 0

with open(filename, 'r') as file:

reader = csv.DictReader(file)

for row in reader:

if row['фирма-поставщик'] == supplier:

total\_price += float(row['цена']) \* int(row['проданное за день'])

if float(row['цена']) > max\_price:

max\_price = float(row['цена'])

print(f"Общая стоимость проданных товаров от {supplier}: {total\_price}")

print(f"Самый дорогой товар: {max\_price}")

# Использование функций для поиска

search\_by\_supplier("Фруктовый рай", "products.csv")

search\_by\_name\_and\_price\_range("Яблоки", 1.5, 2.5, "products.csv")

search\_by\_supplier\_details("Фруктовый рай", "products.csv")

В этой программе мы создали три функции:

1. search\_by\_supplier(supplier, filename) - для поиска поставщика товаров.

2. search\_by\_name\_and\_price\_range(name, min\_price, max\_price, filename) - для поиска по наименованию товара и ценовому диапазону.

3. search\_by\_supplier\_details(supplier, filename) - для определения общей стоимости проданных товаров и самого дорогого товара для данного поставщика.

Теперь у нас есть удобные функции для поиска и анализа данных о товарах по заданным параметрам. Не забудь создать файл products.csv с описанием товаров для корректной работы программы.

А вот и блок-схема нашей программы:

Начало

|

| --- Функция search\_by\_supplier(supplier, filename)

| --- Функция search\_by\_name\_and\_price\_range(name, min\_price, max\_price, filename)

| --- Функция search\_by\_supplier\_details(supplier, filename)

|

--- Вызываем функции поиска для различных параметров

Конец

1. Реализовать метод DFS. Используя возможности графических библиотек языка Python написать метод рисующий граф. Ребра, образующие лес обхода в глубину закрасить в красный цвет. Составить блок-схему.

import networkx as nx

import matplotlib.pyplot as plt

def dfs(graph, node, visited):

if node not in visited:

visited.add(node)

print(node, end=' ')

for neighbor in graph[node]:

dfs(graph, neighbor, visited)

# Создание графа

G = {

'A': ['B', 'C'],

'B': ['A', 'D', 'E'],

'C': ['A', 'F'],

'D': ['B'],

'E': ['B', 'F'],

'F': ['C', 'E']

}

visited = set()

print("DFS обход:")

dfs(G, 'A', visited)

# Визуализация графа

G\_nx = nx.from\_dict\_of\_lists(G)

pos = nx.spring\_layout(G\_nx)

nx.draw(G\_nx, pos, with\_labels=True, node\_size=1000, node\_color='skyblue', font\_size=10, font\_color='black')

# Отмечаем рёбра обхода в глубину красным цветом

red\_edges = [('A', 'B'), ('B', 'D'), ('D', 'A'), ('B', 'E'), ('E', 'F'), ('F', 'C')]

for edge in G\_nx.edges():

if edge in red\_edges:

G\_nx[edge[0]][edge[1]]['color'] = 'red'

edge\_colors = [G\_nx[u][v]['color'] if (u, v) in red\_edges else 'black' for u,v in G\_nx.edges()]

nx.draw\_networkx\_edges(G\_nx, pos, edgelist=G\_nx.edges(), edge\_color=edge\_colors)

plt.show()

Этот код содержит реализацию метода DFS для обхода графа и визуализацию графа, где рёбра обхода в глубину окрашены в красный цвет.

Блок-схема программы:

Начало

|

| --- Создание функции DFS(graph, node, visited)

| |

| | --- Если узел node не посещен

| | | --- Добавляем узел в посещенные

| | | --- Печатаем узел

| | | --- Рекурсивно вызываем DFS для соседних узлов

|

|

--- Создание исходного графа G

--- Вызов DFS для обхода графа с начальным узлом 'A' и набором посещенных узлов visited

--- Визуализация графа с помощью библиотеки networkx и matplotlib

--- Окраска рёбер обхода в глубину красным цветом

--- Отображение графа

Конец

1. Решить задачу топологической сортировки для неориентированного невзвешенного графа, используя метод обхода в глубину DFS. Составить блок-схему.

Для решения задачи топологической сортировки для неориентированного невзвешенного графа с помощью метода обхода в глубину (DFS), мы можем использовать алгоритм, который будет фиксировать узлы в порядке, обратном порядку закрытия их во время DFS.

Вот пример программы на Python, которая реализует топологическую сортировку для такого графа:

def dfs\_topsort(graph, node, visited, stack):

visited.add(node)

for neighbor in graph[node]:

if neighbor not in visited:

dfs\_topsort(graph, neighbor, visited, stack)

stack.append(node)

def toposort(graph):

visited = set()

stack = []

for node in graph:

if node not in visited:

dfs\_topsort(graph, node, visited, stack)

return stack[::-1]

# Пример неориентированного графа

graph = {

'A': ['B', 'C'],

'B': ['D'],

'C': [],

'D': ['C']

}

topological\_order = toposort(graph)

print("Результат топологической сортировки:", topological\_order)

Этот код содержит реализацию функции для топологической сортировки графа с использованием метода обхода в глубину (DFS).

Блок-схема программы:

Начало

|

| --- Создание функции dfs\_topsort(graph, node, visited, stack)

| |

| | --- Добавляем узел node в посещенные

| | --- Для каждого соседа neighbor узла node

| | | --- Если сосед не посещен, вызываем dfs\_topsort рекурсивно

| --- Создание функции toposort(graph)

| |

| | --- Инициализация множества visited и стека stack

| | --- Для каждого узла node в графе

| | | --- Если узел не посещен, вызываем dfs\_topsort

|

--- Создание примера неориентированного графа graph

--- Вызов функции toposort для графа

--- Вывод отсортированного порядка узлов

Конец

1. Реализовать волновой алгоритм, используя метод обхода в ширину BFS. Составить блок-схему.

Для реализации волнового алгоритма с использованием метода обхода в ширину (BFS) в Python, мы можем использовать очередь для хранения узлов, которые нужно посетить. Вот пример программы:

from collections import deque

def wave\_algorithm(graph, start\_node):

visited = set()

queue = deque([(start\_node, 0)]) # Очередь с кортежами (узел, расстояние)

visited.add(start\_node)

while queue:

node, distance = queue.popleft()

print(f"Узел: {node}, Расстояние: {distance}")

for neighbor in graph[node]:

if neighbor not in visited:

queue.append((neighbor, distance + 1))

visited.add(neighbor)

# Пример графа для волнового алгоритма

graph = {

'A': ['B', 'C'],

'B': ['A', 'D', 'E'],

'C': ['A', 'F'],

'D': ['B'],

'E': ['B', 'F'],

'F': ['C', 'E']

}

start\_node = 'A'

print("Результат работы волнового алгоритма (BFS):")

wave\_algorithm(graph, start\_node)

Этот код реализует волновой алгоритм с помощью метода обхода в ширину (BFS) для заданного графа.

Блок-схема программы:

Начало

|

| --- Создание функции wave\_algorithm(graph, start\_node)

| |

| | --- Инициализация множества visited и очереди queue

| | --- Добавление начального узла и расстояния в очередь

| | --- Добавление начального узла в посещенные

|

| | --- Пока очередь не пуста

| | | --- Извлечь узел и расстояние из очереди

| | | --- Вывести узел и расстояние

| | | --- Для каждого соседа узла

| | | | --- Если сосед не посещен, добавить его в очередь и посещенные

|

|

--- Создание примера графа graph и начального узла start\_node

--- Вызов функции wave\_algorithm для графа с начальным узлом

--- Вывод результата работы волнового алгоритма

Конец

1. Создать словарь, состоящий из N элементов по следующему правилу: ключи – числа от 1 до N, значения – первые N простых чисел. Величина N вводиться с клавиатуры. Например, при N = 7 программа формирует словарь из 7-ми простых чисел – {1:2, 2:3, 3:5, 4:7, 5:11, 6:13, 7:17}. Составить блок-схему

Чтобы создать словарь, содержащий первые N простых чисел, давай сначала разберемся, что такое простые числа. Простые числа - это числа, которые больше 1 и делятся только на 1 и на само себя.

Сначала нам нужно написать функцию для определения простого числа. В Python, мы можем сделать это следующим образом:

def is\_prime(num):

if num < 2:

return False

for i in range(2, int(num \*\* 0.5) + 1):

if num % i == 0:

return False

return True

Эта функция is\_prime принимает число num и возвращает True, если число является простым, иначе False.

Теперь, когда у нас есть функция для проверки простого числа, мы можем создать словарь, содержащий первые N простых чисел:

def create\_prime\_dict(N):

prime\_dict = {}

num = 2

count = 1

while count <= N:

if is\_prime(num):

prime\_dict[count] = num

count += 1

num += 1

return prime\_dict

В данной функции create\_prime\_dict(N), мы начинаем с числа 2 и проверяем, является ли каждое число простым. Если число простое, мы добавляем его в словарь с ключом, равным текущему счетчику.

Теперь у нас есть функции для определения простого числа и создания словаря с первыми N простыми числами. Теперь давайте создадим словарь для N, вводимого с клавиатуры:

N = int(input("Введите число N: "))

prime\_numbers\_dict = create\_prime\_dict(N)

print(prime\_numbers\_dict)

Этот код позволит пользователю ввести значение N, а затем создаст словарь с первыми N простыми числами и выведет его на экран.

Теперь ты можешь попробовать запустить этот код в своей среде разработки и посмотреть, как он создает словарь с первыми N простыми числами!

1. Даны два списка одинаковой длины. Создать словарь, так чтобы элементы первого списка были ключами словаря, а элементы второго списка значениями. Составить блок-схему.

def create\_dict\_from\_lists(keys, values):

result\_dict = {} # Инициализируем пустой словарь

for key, value in zip(keys, values):

result\_dict[key] = value

return result\_dict

Эта функция create\_dict\_from\_lists(keys, values) принимает два списка keys и values одинаковой длины и создает словарь, где элементы первого списка будут ключами, а элементы второго списка - значениями.

Теперь давай использовать эту функцию, чтобы создать словарь из двух списков:

keys = ['a', 'b', 'c', 'd']

values = [1, 2, 3, 4]

result\_dict = create\_dict\_from\_lists(keys, values)

print(result\_dict)

При запуске этого кода мы создадим словарь, где элементы первого списка keys ('a', 'b', 'c', 'd') станут ключами, а элементы второго списка values (1, 2, 3, 4) - значениями.

Таким образом, мы можем легко и быстро создать словарь из двух списков одинаковой длины, превратив их в пары ключ-значение.

А вот и блок-схема для нашего кода:

Начало

|

| --- Создать функцию create\_dict\_from\_lists(keys, values)

| |

| | --- Инициализировать пустой словарь result\_dict

| | --- Для каждого элемента key и value из keys и values

| | --- Добавить в словарь result\_dict пару ключ-значение (key, value)

| --- Вернуть result\_dict

|

|

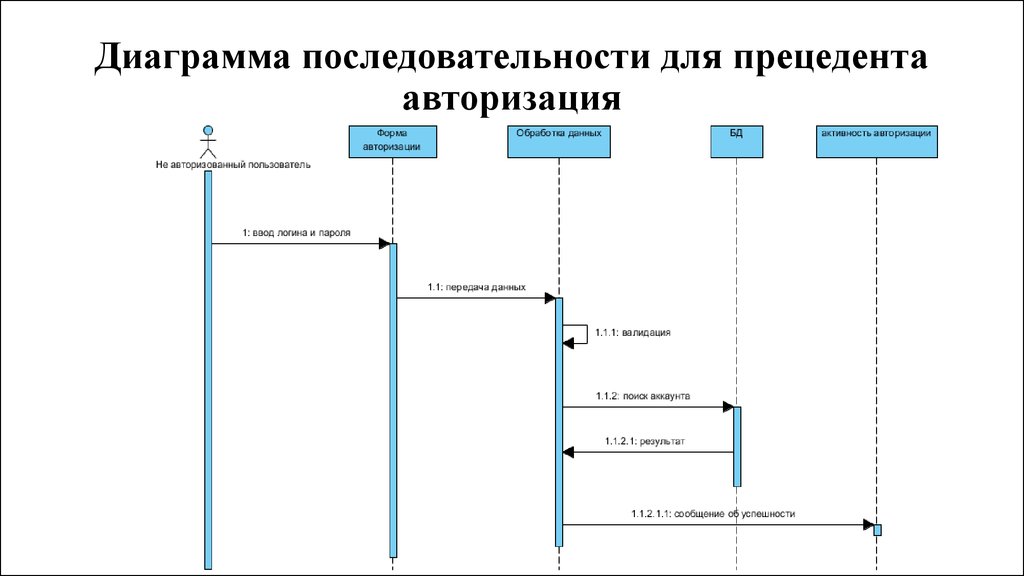
--- Создать списки keys и values

| --- Вызвать функцию create\_dict\_from\_lists(keys, values) и сохранить результат в result\_dict

--- Вывести result\_dict на экран

Конец

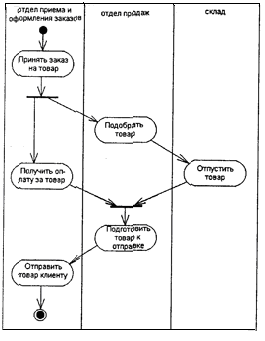
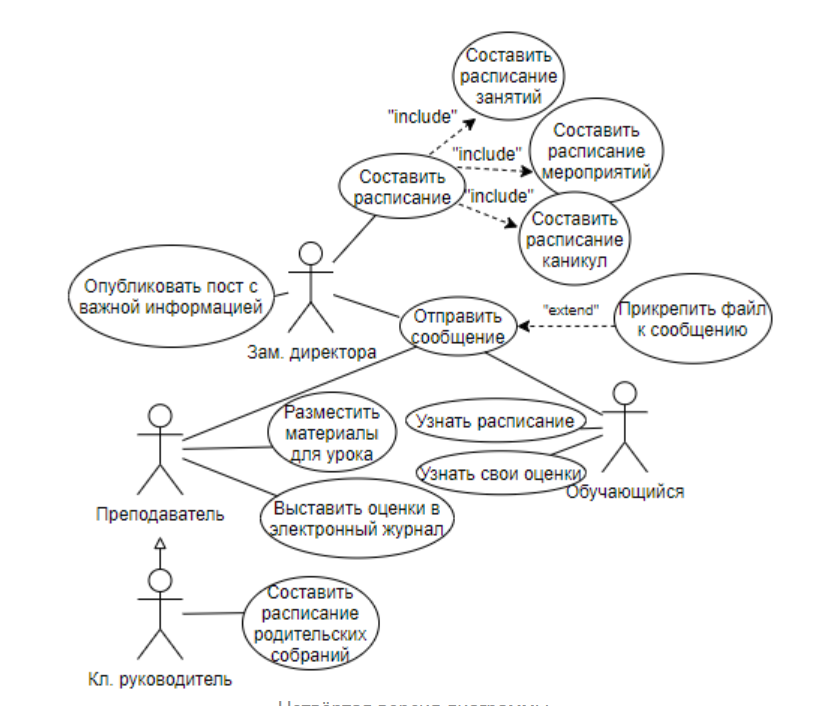
1. Построить диаграмму последовательности для любого процесса по теме ... показаны взаимодействия объектов, упорядоченные по времени их проявления.

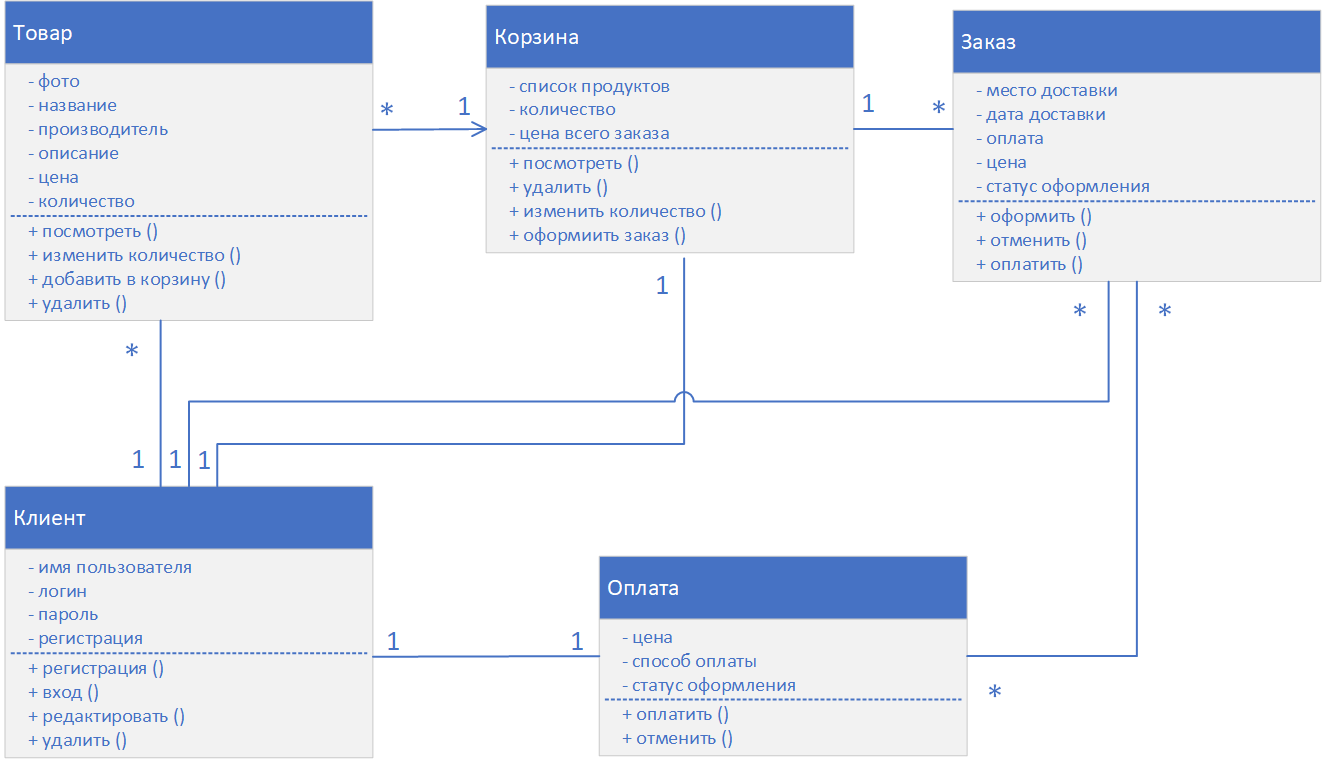


1. Построить диаграмму DFD (контекстный уровень и декомпозицию первого уровня) по теме … диаграмма потоков данных Отображает пути, по которым циркулируют данные внутри системы, а также между системой и внешним миром.





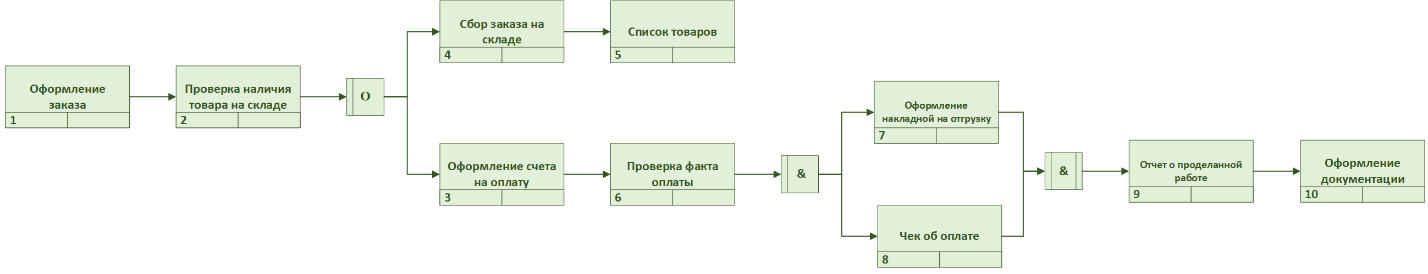
1. Построить диаграмму деятельности для любого процесса по теме Она **показывает**, как каждое действие ведет к следующему, и как все они связаны. …
2. Построить диаграмму вариантов использования по теме … то делает система и как ее используют действующие лица, но не **показывают**, как системаработает внутри.  
3. Построить диаграмму классов по теме ... **модель статической структуры программной системы в терминах классов объектно-ориентированного программирования**.

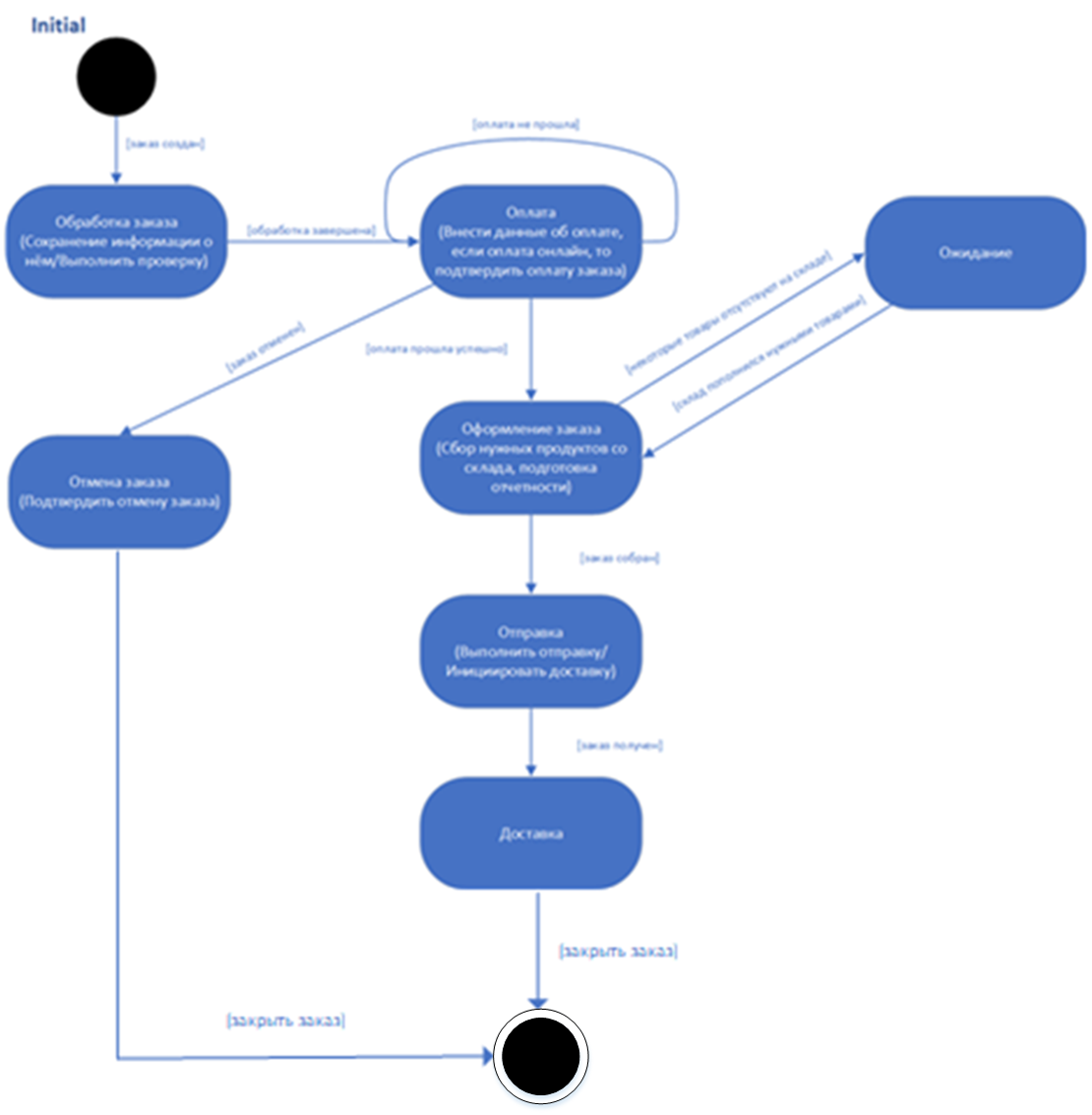


1. Построить диаграмму объектов по теме ...

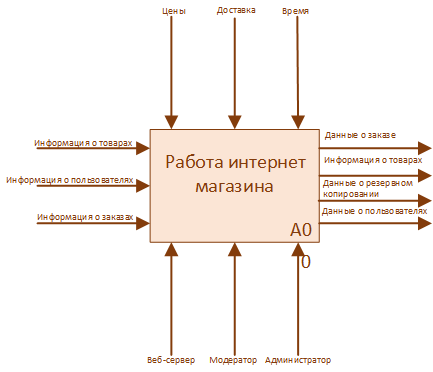


1. Построить диаграмму IDEF3 для любого процесса из темы ...

**как организована работа и какие пользователи работают с моделируемой системой**. 

1. Построить диаграмму состояний любого объекта, который меняется в течение работы системы, по теме ... **Диаграмма** **состояний** **показывает**, как объект переходит из одного **состояния** в другое.  
2. Построить диаграмму IDEF0 по теме ...

**Диаграмма IDEF0** показывает процессы, процедуры и действия внутри организации. Интегрированное автоматизированное производство



Декомпозиция первого уровня контекстной диаграммы IDEF0

