1. **Расскажите, какие действия выполняет метод __init__ класса SQLProvider**

```
def __init__(self, path):
    self._scripts = {}
    for file in os.listdir(path):
        if file.endswith('.sql'):
            self._scripts[file] = Template(
                open(f'{path}/{file}').read()
            )
```

2. **Расскажите, какие действия выполняет метод get класса SQLProvider.**

```
def get(self, name, **kwargs):
    if name not in self._scripts:
        raise ValueError("No such file in sql_provider")
    return self._scripts[name].substitute(**kwargs)
```

3. **Поясните, для чего в классе SQLProvider используется класс Template**
   Template in Python — это класс из модуля строк, который позволяет изменять данные без необходимости редактировать приложение. Он обеспечивает простую замену строк, когда поля шаблона заменяются на подходящие замены, предоставленные пользователем.
   Работа с шаблоном:
   1. Класс принимает строку в качестве шаблона. Внутри строки используют имена placeholder-переменных, предшествуя им знак доллара ($) для обозначения места за placeholder.
   2. Затем подставляют значение в шаблон с помощью метода substitute() с использованием словаря, где ключ в словаре соответствует имени placeholder-переменной.
   3. Возвращённая строка — это шаблон со всеми значениями, а не с placeholder-переменными.
4. **Поясните, какие действия выполняет следующий оператор.**
   provider = SQLProvider(os.path.join(os.path.dirname(__file__), ‘sql’)))
   создает объект класса SQLProvider, в конструктор которого передается путь к папке sql, который формируется автоматически, в зависимости от текущей директории, из которой запускатеся файл

5. **Поясните, почему мы стараемся не включать задание конкретных значений для любых данных в текст скриптов Python.**
   Тогда любой код становится статичным и неуниверсальным.

6. **Поясните, как работает конструкция with в функции select_list**

```
####### порядок работы конструкции with #######
# инициируются переменные (cursor) в методе __init__
# управление передаётся методу __enter__
# создаётся курсор или ничего
# возвращение управления вызвавшей функции
# если курсор не был создан, то создаётся ошибка ValueError
# выполняются все действия в функции,
# если возникает ошибка, то вызывается метод __error__
```

7. **Расскажите, в какой момент запускается метод __init__ класса DBContextManager (DBConnection)**
   app.py -> model_route() -> select() - > DBConnection(db_config) -> во время создания экземпляра класса DBConnection

8. **Расскажите, в какой момент запускается метод __enter__ класса DBContextManager**
   app.py -> model_route() -> select() - > при выполнении sql запроса (with DBConnection(db_config) *as cursor*)

9. **Расскажите, в какой момент запускается метод __exit__ класса DBContextManager. Какие параметры передаются этому методу**
   В момент, когда SQL сервер получил ошибку. В параметрах метода лежат ошибки, которые передаёт SQL сервер при ошибке

   Какую роль играет оператор Raise Valuerror()
   Если курсор не был создан, то создаётся ошибка ValueError

10. **Что представляет собой app.config во flask**
    app.config во Flask — это **атрибут объекта Flask, который содержит загруженные значения конфигурации**. 1
    Это место, где Flask сама определяет определённые значения конфигурации, а также где расширения могут определять свои значения. Кроме того, здесь можно создать собственную конфигурацию.

11. **Почему http протокол называют протоколом без памяти.**
    HTTP-протокол называют протоколом без памяти, потому что он не хранит информацию о предыдущих запросах клиентов и ответах сервера.

12. **Какова структура HTTP запроса**
    Структура запроса в протоколе HTTP включает три основных части: 1
1. Строка запроса. Указывает метод передачи, версию протокола HTTP и URL, к которому должен обратиться сервер. 1
2. Заголовок. Содержит тело сообщения, передаваемые параметры и другие сведения. 1
3. Тело сообщения. В нём могут находиться передаваемые в запросе данные. 1
   Между заголовком и телом есть пустая разделительная строка. 1
   Не у каждого HTTP-метода предполагается наличие тела. Например, методам GET, HEAD, DELETE, OPTIONS обычно не требуется тело. Некоторые виды запросов могут отправлять данные на сервер в теле запроса.

13. **Какова структура HTTP ответа**
    Структура ответа HTTP идентична структуре запроса и включает в себя:
1. Строку статуса (Status line). Содержит следующую информацию:
   - Версию протокола, обычно HTTP/1.1.
   - Код состояния (status code), показывающий, был ли запрос успешным. Примеры: 200, 404 или 302.
   - Пояснение (status text). Краткое текстовое описание кода состояния, помогающее пользователю понять сообщение HTTP.
2. Заголовок. Имеет ту же структуру, что и все остальные заголовки: не зависящая от регистра строка, завершаемая двоеточием (':') и значение, структура которого определяется типом заголовка.

3. Тело. Присутствует не у всех ответов. Как правило, тело ответа используется в том случае, когда нужно вернуть вызывающей стороне информацию о ресурсе.

14. Что представляет собой WSGI интерфейс.

WSGI (Web Server Gateway Interface) — стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером, например Apache.

Он предоставляет простой и универсальный интерфейс между большинством веб-серверов и веб-приложениями или фреймворками.

Он представляет собой протокол для коммуникации, который позволяет различным серверам и приложениям взаимодействовать друг с другом согласованным образом.

15. Какие действия выполняет WSGI сервер.

По стандарту, WSGI-приложение должно удовлетворять следующим требованиям:

- должно быть вызываемым (callable) объектом (обычно это функция или метод)
- принимать два параметра:
    1. словарь переменных окружения (environ)
    2. обработчик запроса (start_response)
- вызывать обработчик запроса с кодом HTTP-ответа и HTTP-заголовками
- возвращать итерируемый объект с телом ответа

Чтобы запустить наше WSGI приложение, нужен WSGI сервер. Он запускает WSGI приложение один раз при каждом HTTP запросе от клиента.

Задачи WSGI сервера:

- Сформировать переменные окружения (environment)
- Описать функцию обработчик запроса (start_response)
- Передать их в WSGI приложение
- Результат WSGI сервер отправляет по HTTP клиенту
- а WSGI шлюз приводит к формату клиент-серверного протокола (CGI, FastCGI, SCGI, uWSGI, …) и передает их на Веб-сервер (например выводит в stdout, stderr).

16. Какие действия умеет выполнять web server

Некоторые действия, которые умеет выполнять веб-сервер:

- Приём и обработка запросов по протоколу HTTP. Сервер принимает запросы от браузера, проверяет их корректность и определяет запрашиваемый ресурс на основе указанного URL.
- Обработка программных скриптов. В процессе обработки запроса сервер может выполнять программные скрипты, запрашивать данные из базы данных, считывать данные из файловой системы.
- Формирование и отправка HTTP-ответов в браузерах, установленных на компьютерах пользователей. Ответ включает статус, заголовки и тело, содержащее запрошенный ресурс.
- Создание журналов ошибок и обращений к документам (логов). 2 Информация записывается в файлы журналов для обеспечения безопасности и статистических целей.

- Аутентификация и авторизация пользователей. Веб-сервер может разрешать, запрещать или авторизировать доступ к частям веб-сайтов (веб-ресурсам)
- Использование различных настроек для обработки файлов и данных. Например, сервер может использовать механизмы кэширования, сжатия данных и другие механизмы, позволяющие улучшить скорость работы сайта.

17. **Как параметры HTTP запроса становятся доступными в скриптах на Python.**
    Параметры HTTP-запроса становятся доступными в скриптах на Python благодаря библиотеке Requests.
    Метод GET в Requests принимает параметр с именем params, в котором можно указать параметры запроса в формате словаря Python. Метод самостоятельно включит эти параметры в запрос.

```
get_params = {'page': 11, 'product': 'car'}
response = requests.get('https://httpbin.org/', params=get_params)
print(response.url)  # Выведет ссылку с параметрами запроса:
https://httpbin.org/?page=11&product=car [1](https://smartiqa.ru/blog/pyth
on-requests)
```

    В этом примере get_params — это переменная, которая содержит параметры запроса, а response — результат выполнения метода requests.get() с указанными параметрами.

18. **Какие вы знаете поля ввода в формах ввода.**
    текстовые поля, поля пароля, раскрывающиеся списки, флажки, радиокнопки, сборщики информации и др.
    TEXT, PASSWORD, CHECKBOX, RADIO, SUBMIT, RESET, FILE, HIDDEN, IMAGE
19. **Какие обязательные параметры надо задавать в теге form. Какую роль играют эти параметры.**
    Некоторые обязательные параметры, которые нужно задавать в теге <form>, и их роль:
    - action. Адрес, на который будут отправлены данные формы. Когда форма отправляется на сервер, управление данными передаётся программе, заданной этим атрибутом.
    - method. Метод отправки данных на сервер (GET или POST). Если использовать метод GET, параметры формы будут отображаться в адресной строке браузера.
    - target. Окно или фрейм, в котором будет открыт результат отправки данных формы.
    - name. Имя формы, которое используется для её идентификации при отправке данных на сервер.
    - autocomplete. Указывает, может ли браузер запоминать введённые пользователем данные для автозаполнения.
    - enctype. Способ кодирования данных формы при отправке на сервер.
    - novalidate. Указывает, что данные формы не нужно проверять на корректность перед отправкой на сервер.
20. **Какую роль в динамических шаблонах играют фигурные скобки. Какая программа обрабатывает выражения с фигурными скобками.**

Фигурные скобки в динамических шаблонах Python играют роль переменных, выражений и вызовов функций. Они позволяют получить результат выражения, переменной или вызвать функцию и вывести значение в шаблоне.

Во Flask выражения с фигурными скобками обрабатывает встроенный движок шаблонов Jinja. Он читает данные из файла-шаблона, находит в нём специальные конструкции и, согласно определённым правилам, заменяет их на данные из другого источника. На выходе получается готовый документ, где уже нет никаких «переменных» и «неизвестных», все данные определены и «жёстко вшиты».

21. **Паттерн MVC**
MVC (Model-View-Controller) — это **архитектурный паттерн, который разделяет приложение на три логических компонента**: модель, представление и контроллер. 2
Модель (Model) содержит данные приложения и связанную с ними логику, а также закрепляет структуру программы. Например, если программист создаёт приложение для списка дел, код модели будет определять основные компоненты приложения: что такое «задача» и что такое «список». 2
Представление (View) состоит из функций, которые отвечают за интерфейс и способы взаимодействия пользователя с ним. Представления создают на основе данных, собранных из модели. В случае приложения со списком дел View определяет, как выглядят задачи. 2
Контроллер (Controller) связывает модель и представление. Он получает на вход пользовательский ввод, интерпретирует его и информирует о необходимых изменениях. Например, отправляет команды для обновления состояния, такие как сохранение документа. В приложении со списком дел контроллер определяет, как пользователь добавляет задачу или помечает её как выполненную. 2
Паттерн MVC позволяет писать независимые друг от друга блоки кода, которые можно менять отдельно, не затрагивая остальные. 2 Это помогает эффективно работать разным программистам — каждый занимается своим компонентом.

22. **Для чего нужны системные диаграммы последовательности.**
Системные диаграммы последовательности нужны для наглядного представления взаимодействия между объектами в системе. 34 Они отображают порядок выполнения действий и обмена информацией во времени. 4
Вот некоторые цели использования таких диаграмм:

- Понимание функционирования системы. Диаграммы помогают уточнить логику взаимодействия объектов и участников системы. 4
- Выявление потенциальных ошибок и проблем. 4 Моделируя последовательность взаимодействий, диаграммы помогают определить узкие места, неэффективность или ошибки в системных процессах. 1
- Проектирование системы. Диаграммы помогают определить, как общаются различные компоненты или сервисы в системе, что важно при разработке сложных, распределённых систем или сервисно-ориентированных архитектур. 1
- Документация поведения системы. Диаграммы дают возможность задокументировать, как работают вместе разные части системы, что полезно для разработчиков и команд по обслуживанию.