# **Основы Программирования**

**Компилятор -** выполняется весь код целиком за один раз. Более быстрый

**Интепритатор -** выполняет код построчно. Более медленный

**PEP 8**  - документ в котором написаны рекомендации для написания кода на python

**setting.json** - файл в котором хранятся все настройки VS Code

**Статическая Типизация** **(с++)** - тип переменой указывается при ее создании. Изменить етот тип или присвоить значение другого типа нельзя

**Динамическая Типизация (python)** - тип переменой указывается во время присвоения значения. Можно изменять етот тип и присваивать значение другого типа.

**Явная Конвертация Типов** - преобразования одного типа данных в другой самим програмистом

**Неявная Конвертация Типов** - преобразование одного типа данных в другой самим языком програмирования

**Python НЕ** выполняет неявную конвертация типов

**Зборщик Мусора Python** - когда завершается работа с обьектом, удаляет етот обьект из памяти. Если ссылки на обьект нету, тогда считаеться ненужным и удаляется

**Big-O Notation** – меод оценки алгоритма по времены выполения его и занятому месту в программе

**DRY (Don’t Repeat Yourself)** – принцип разработки програмы нацеленый на снижение повторения одинаковых блоков кода. За ето отвечают функции в нашем коде

# **SOLID**

**Single Responsibility** - принцип единственной ответственности. Один Класс дожен делать что-то одно и не больше. Выполнять одну функцию

**Open Closed** - принцип открытости / закрытости. Классы открыти для разширения, но закрити для изменения. Изменение функционала класса не через изменение существующего кода, а через добавление новый свойств и методов

**Принцип Подстановки Барбари Лисков** - Классы наследники только добавляют функционал к базовому классу и не переопределяют существующие свойства и методы

**Принцип Разделения Интерфейсов** - клиенты не зависят от методов, которые не используют. Создаеться много наследования для одного класса и етот класс наследвник вызываеть в себе только необходимый метод в текуйщи момент

**Принцип Инверсий Зависимостей** - классы верхних уровней не могут зависить от класов нижних уровень. Абастракные методы и классы не зависят от классов которые их наследуют

# **Дополнения**

--- **CI / CD ->**

**CI Неприрывная Интеграция -** интеграция нового кода в уже существующий без участия человека з помощью процесов автоматизации

**CD**  **Неприрывная Доставка** - процес автоматического развертывания приложения на реальном сервера после неприрывной интеграции без участия человека

# **Структура Кода**

**Модуля -** файл с розишрением **.py** , который вы добавляем в другой файл

**Пакет -** папка, которая содержит файлы с розширением **.py** (модули) и также файл \_\_init\_\_.py

**Файл \_\_init\_\_.py** - код етого файла выполянеться каждый раз, когда импортируем пакет или модуль

**Импортирование Модуля** - добавление к файлу опередельоного модуля с функционалом

**Експортирование Модуля** - виносим текущий файл для пользования функционалом в других файлах

**Встроенные Модули** - модули, которые уже входят в стандартный язык python, даються при скачивании. Сайт с документацией про модули docs.python.org -> General Module Index

**Сторонние Модули** - модули, которые написаны другими (сторонними) разработчиками

**Переменная -** именованая ссылка на обьект, который хранится в памяти компьютера. Переменные могут ссылатся на один и тотже обьект в памяти. Позволяют сохранят, изменять и использоваться указанные значения в наших программах.

**Область Видимости** - границы действия переменной

обращение к переменной внутри функции **-> Внутренняя Область -> Локальная -> Глобальная**

**Глобальные Переменные** - переменные, которые доступны в любом участке кода

**Локальные Переменные (Только Внутри Функции)** - переменные, которые доступны только в определеном участке кода. После выполнения функции, переменные внутри ее удаляются

**Список [ ]** - упорядоченый (пронумерованый) набор елементов разных типов, каждый елемент имеет свой индекс. Индекси начинаються с **0**

**Словарь** **{ }** - неупорядоченый (непровнумерованый) набор елементов по типу ключ : значение. Доступ к елементам производиться по вызову ключа. Нету индексов

**Кортеж ( )** - упорядоченый (пронумерованый) набор елементов разных типов. Кортеж нельзя изменять. Можно возвращать значения про индексу. Индекси начинаються с **0**

**Набор (Множество) { } -** неупорядоченый (непронумерованый) изменяемый набор елементов разных типов. Содержит только уникальные елементы. При вызове каждый раз изменяет разположение елементов в наборе. True переводиться в число 1, False остается таким же. Нету индексов. Внутрь набора нельзя добавлять изменяемы елементы **( list, dict ,set )**. Нельзя использовать **del**. Можна поместить внутрь кортеж **( )**

**Диапазон** - упорядоченый (пронумерованый) неизменяемый набор уникальных елементов типа **int**. Значение елементов не повторяется.

**Пересичение Множеств -** **my\_(set set)\_my -** одинаковые елементы, которые только находятса и в первом и во втором множестве

**Обьединение Множеств - (my\_set set\_my) -** все елементы, которые находятся в обеих множествах

**Разница Множеств - (my\_) set set\_my** - елементы, которые находятся в одном из множеств без елементов пересичения

**Подмножество** - множество, которое имеет елементы второго множества и не имеет других

**Розпаковка -** извлечение значений из последовательностей и присвоенние их переменным

**Операторы** - специальные символы, которые выполняют арифметические или логические вичисления

**List , Set , Dict Comprehentions** – ето название для инструкций for a in range( ) : my\_var.append( a )

**Операнды** - значения на которые действует оператор

**LEGB (Local, Enclosing, Global and Built-in)** – последовательность поиска переменных в Python. Сначала ищет значение переменной в внутренней функции, потом по внешней и потов в глобальной области нашей програмы

**Унарные Операторы** - состоят из одного оператора и одного операнда ( **+ a - a not a** )

**Бинарные Операторы** - состоят из одного оператора и двох операндов **( a = 5 , a != b , a and b )**

**Теранарный Оператор -** условное выражение, которое состоит из трьох операндов. Может возвращать значение. Выражение1 **if** Условие **else** Выражение2. Выполняется одно из выражений True - Выражение 1, False - Выражение 2

**Приоритетность Операторов -** последовательность выполнения операторов, нужно указывать скобки **a + ( ( b \* c ) / ( d - e ) )**. Скобки указывають последовательность выполнения выражения.

**Выражение** - код результат которого всегда является значение. **2 + 2 a > b “one” + ”two” my\_fn( 5 )**

**Инструкция -** код который выполняет действие. **a = 5 if True: print() import datetime return**

**Строка** - упорядоченая (пронумерованя) последовательность символов

**Комплексное Число 2 + 3j** - число, которое состоит из действительной и мнимой части.

**Ложное Значение** - значение, которое при переводе к логическому типу bool возвращает False **( 0 - 0.0 - 0J - False - None - [ ] - { } - ( ) - set( ) - range( ) - str “ ”)**

**Правдивое Значение** - значение, которое при переводе к логическому типу bool возвращает True **(1, - 1, True, 5.3 - [1 , 2 ] - { “key1”:1 } - (1 , 2 ) - set( [ 1 , 2 ] ) - str”text” )**

**Цикли -** повторение блока кода, пока условие истенное. Используються для перебора елементов последовательностей.

**Итератор** - интерфейс для перебора любого итерируемого обьекта (последовательности)

**Коментарии #** - текст написаный в коде, который не будет исполняться. Бывают однострочные и многострочные

Как давать имена в Python =>

**snake\_case** -> переменные , фукнции, методы и модули

**PascalCаse** -> классы

**my-package** -> пакеты

**DB\_PASSWORD** -> константы

# **Функции**

**Функция** - блок кода, который можна выполнять многократно в разных местах нашей програмы. Состоит из названия, параметров та тела. Внутри функции не рекомендуется изменять внешние обьекты.

**Имя Функции** - название функции по которому ее будут повторно вызывать

**Параметры Функции** - переменные доступные только внутри функции, которые будут означать реальные значения которые передаються внутрь ее

**Аргументы Функции** - реальное значение, которое уже передается в функцию при ее вызове

**Параметры по Умолчанию** - значения, которые будут переданные в функцию при отсутствии реальных аргументов. Если реальные аргументы есть, тогда параметры по умолчанию не используються.

**Позиционные Аргументы** - передача аргументов просто перечесляя их значения в функцию

**Ключевые Аргументы** - при передачи аргументов в функцию необходимо указывать значения для параметров. После ключевых аргументов нельзя использовать позиционные.

**Анотации Типов** - подсказка для програмиста, какой тип данных переменной принимает функция. Пример **def my\_fn( my\_var : int )**

**Тело Функции** - набор строк внутрие ее, который выполняется при вызове функции

**Возвращаемое Значение Функции** - значение, которое вернет функция после успешного выполнения ее тела. После return функция прекращает работу

**Встроенная Функция ZIP** - обьеденяет групу последовательностей в один список кортежей **->** **[ (1 , True ) , ( “text” , -3 ) ]**. Можна увидеть значения после конвертации в другой тип (например list - tuple - set). Если в одном списке больше елементов чем в другом, тогда они не берутся

**Callback Функция** - функция, которая передаеться как аргумент в другую функцию и там вызывается

**Lambda Функция** - функция, которая содержит в себе только одно выражение. Созданая для возвращения значения типа <class ‘function’> из внешней функции. Хранятся внутри переменных. Помещается в одну строчку кода. Не имеют названия.

**Декораторы Функций** - возможность для разширения функционала работы функции. Ето обертки вокруг етих функций

**Замыкания** - функция, которая находиться внутри другой функции и ссылаеться на переменные во внешней функции. Ета внешняя функция записываеться в переменую. Зборщик мусора не может удалить значения и области видимости итих двох переменных

**Распространение Исключений (Propagation Exceptions)** - исключение может разпространяться на внешние области видимости(стеки вызовов), пока не дойдет до области верхнего уровня main. Исключение можно ловить на разных уровнях

# **ООП - Обьектно Ориентированое Программирование**

**ООП -** вид програмирования, где все состоит из обьектов

**Обьекты** - екзепляр опеределеного класса. При создании обьекта унаследуються все атрибуты класса. Все сущности в python обьекты (число, строка)

**Класс** - шаблон (прототип) для создания обьекта

**Атрибут Обьекта** - характеристика обьекта, которая лежит внутри него

**Метод** - вид атрибута, значение которого функция

**\_\_Магические Методы\_\_** - внутренные методы классов, которые обычно не вызываются явно. Методы, которые вызывает python от имени програмиста в оперделеных обстоятельствах.

**Свойство (Поле)** - вид атрибута, значение которого ето переменная

**Наследование -** когда беруться свойства и методы одного класса и унаследуються другим классом. В классе наследнике сначала выполняется конструктор родительского класа, а потом текущего

class MyClass ( MyClass1 ): простое наследование

**Множественное Наследование** - когда одни дочерный класс может наследоваться из двох и более классов

**class MyClass (MyClass1 , MyClass2)** множественное наследование **==**

**Поиск Метода -> MyClass -> MyClass1 -> MyClass2 -> object**

**Method Resolution Order (MRO) -** указыват хронологию наследования свойст и методов для текущего класса от других класса. Методы и свойства с одинаковыми названия перереопределяються в классе который стоит правее при наследовании.

**Инкапсуляция -** когда свойства и методы одного обьекта скрыты от других обьектов

**Полиморфизм -** возможность работать одним методом(функцией) с обьектами разных классов. У всех классов есть метод с одним названием, но разной реализацией

**Базовый Класс (Родительский класс)** - класс от которого наследуються методы и свойства

**Дочерний Класс (Подкласс)** - класс, который сам унаследует методы и свойства

**Конструкторы** - часть кода, которая выполняется при создании обьекта

**Переопредиление Методов** - изменение работы методов с одним названием из базового класа в класах наследниках

**Делегирование Методов** - когда в дочерних класах вызываем методы из базового класса через функцию super()

**Абстрактные Класы** - обьект етого класса нельзя создать. Етот класс служит только для наследования другим классом

**Абстрактные Методы** - методы, которые необходимо переопределять в классах наседниках. В базовом классе он не реализован

**Примеси (Mixins - Миксины)** - специально созданый класс для добавления функционала к классам наследниками от базового класса. Предоставляет функционал, но не может быть использован самостоятельно. Реализован отдельно от базового и дочерних классов

**Дескриптор** - отдельный класс в котором реализуються геттери и сеттеры для остальных классов. Уже не нужно писать гетеры и сеттери в каждом классе. Позволяет сохранять, изменять и удалять атрибути екзепляра класса. Дескриптор имеет више приоритет выполнения чем просто обращение к свойствам обьекта.

**Функторы** - классы или обьекты которые можна вызывать по типу функции ( ) . Нужно переопределить метод def\_\_call\_\_(self)

**Колекции \_\_slots\_\_** - ограничивають количество и названия создаваемых свойств для обьекта. Уменьшают размер обьекта в памяти. Ускоряют работу с локальными методами обьекта. После указания колекции \_\_slots\_\_ уже колекция \_\_dict\_\_ не создаеться

**Механизмы Инкапсуляции -> Режимы Доступа -> Только сигнализирует програмисту про доступ**

**attribute - > доступ public** может обращается текущий класс, наследники класса и обьекты

**\_attribute - > доступ protected** может обращатся текущий класс и классы наследники

**\_\_attribute - > доступ private** может обращаться только текущий класс

**Метаклассы ( только type) -** класс, екзепляри которое тоже являються классами для создания обьеков. Отправная точка (вершина) для создания других классов или обьектов. Нужно их использовать чтоб в екзпелярах классах наследниках при создании уже находились локальные свойства с значениями етого екзепляра , а не свойства класса по которому создан. При создании екзпеляров через обычный класс етих свойст нету.

**Метакласс (только type) -> str , bool , int ,set , dict , list -> 5 , True , 3.6 , [1,2,3] , {5,7,1} , {“key”:”value”}**

**Неизменяемый Обьекты** - обьекты, которые нельзя изменять. Их адрес в памяти не изменяется. -> **int str bool float tuple NoneType range**

**Изменяемые Обьекты -** обьекты, которые можна изменять. Их адрес в памяти изменятеся -> **list dict set UserClass**

**Копирование a = b -** при таком копировании сохраняется только ссылка на один и тотже елемент в памяти. При изминении одного обьекта -> создается новый обьект с новой ссылкой в памяти

**Копирование a = b.copy( ) -** копируються только елементы из обьектов первого уровня. При таком копировани ссылки на обьекты, которые лежат внутри сохраняются и копируется только ссылка. Ето копирование первого уровня.

**Копирование a = deepcopy( b )** - копируються все елементи из обьекта и даже внутрение обьекты, а не ссылки на них. Ето копирование второго уровня

# **Синхроность - Асинхроность - Многопроцесорность - Многопоточность**

**Сихронность -** когда учаcтки кода выполняються последовательно. Пока предидущий не завершил работу следующий не будет запускаться

**Асинхроность** - когда участки кода могут выполняться паралельно друг другу. При выполнении одного кода, не создает новые потоки для следующего участка кода. Очень быстрое выполнение. Внутри асинхронным блоков кода можна использовать только асинхронные библиотеки и модули. Асинхроние модули начинаються c aio в названии -> aiohttp , aiogram , aiosqlite

**Коротины** - функции, которые имеют ключевое слово async и можно использовать в асинхронном выполнении кода

**Процесс** - запущеня независамая наша програма. Внутри процесса по умолчанию один поток

**Поток** - запущеня зависимая некая часть нашей програмы, которая работает внутри процесса нашей всей програмы

**Многопроцесорность** - когда для работы нашего кода создаеться новый процесс на нашем устройстве. Только в етоп процессе будет выполняться наш код

**Многопоточность** - когда внутри процесса нашей програмы начинает работать новый поток, в котором внутри выполняеться часть нашей програмы. В Python может быть много потоков, но только один будет выполняться. Переключение между потоками сьедает много времени

# **Углубленный Python**

**Хеш-Таблица** - стурктура данных (таблици) повзволяющая реализовать данные по типу ключ:значение. Позволяет быстро искать нужное значение ключу , но ключ имеет вид ID

**Хеш-Функция** - преобразовует наш ключи из хеш-таблици в ID ячейки етой хеш-таблици, в етой ячейке ID будут лежать наши значения. Преобразование, которое делает ета функция называеться хешированием

**Работа Хеш-Функции ->** Каждая буква ключа кодируеться в числа **->** Ети числа добалвляються **->** От етой суми береться остаток от деления по модулю **->** Число которое получем в результате являеться ID для текущей ячейки, внутри ячейки лежит значение

# **Патерны ООП**

**Sigleton** - можно создать только один обьект класса. При создании нового обьекта передаеться ссылка на старый обьект.

**Моносостояние** - все свойства обьектов класса имеют одни значения. При измении свойств одного обьекта, изменяеться во всех обьектах одновремено.

# **--------------- Back End --------------**

# **Сети**

**API (Програмный Интерфейс Приложения)** – Способ взаемодействия BACKEND и FRONTEND чаксти сайта через данные в формате JSON. Cпособ взаемодействия програм между собой или взамойдествие пользователей с пролижениями, как они могут общаться и передавать данные. Клиент сможет только выполнять только те действия которые заложил разработчик, залезть под капот приложения не сможет

**BackEND -> API -> JSON -> FrontEND**

**Клиент -> API -> Приложение(На Сервере)**

**Чужой Сервер -> API -> Приложение(На Севере)**

**RESTful API (Передача Репризентативного Состояния) –** тип интерфейса с помощью которого компьютеры передавют инормацию друг другу. Отправка данных на сервер(в наше приложение) или принятие данных от сервера(от нашего приложения) происходить по URL адресу который записывает в адресс клиент. GET , POST , PUT , DELETE запросы от клиента к серверу. Поддерживате обмен данных в таких форматах как JSON , HTML , XML , YAML . Сервер не запоминает ничего про перидущий запрос , каждий новый URL запрос нужно приписывать полностью с нуля

**JSON -** передача данных от клиента к серверу(или наоборот) или от сервера к серверу в формате ключ значение { “name”: “max” , “age”: 35 }. Такие файлы имеют разширение my\_file.json

**Кеширование(Caching)** - когда данные, которые принял клиент от нашего сервера(приложения) сохраняються в браузере клиента на несколько минут для повторного использования. Созданно чтоб облегчить работу нашего сервера и не оптравлять одну и туже информацию несколько раз

**↓** **Составные Части Запроса ↓**

**REQUEST LINE** - ето URL адресс в котором пользователь списиавет нужный ему метод взамодействия с серверов, ето GET , POST , PUT , DELETE. Например POST/my\_example/1.5

**HEADER** - информация клиенте, данные авторизации , информация об пк клиента. Например

Host: developer.mozilla.org

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate, br

**BODY MESSAGE** - информация, которую передает клиент на сервер. Ета информация создержит действия, которые необходимо выполнить серверу. Ето всегда файл в формате .json. Например

{“name” : ”mas” , “age”: 20 , “price” : 120, }

**↓** **Статус Коды Запроса ↓**

**Статус Код -** код который присылаеться в ответе пользователю от сервера, показывает успешность или не успешность запроса от пользователя к серверу

**1XX** - просто информация о запросе

**2XX -** показывает что запрос прошел успешно

**3XX -** показывает перенапралвение на другую страницу (redirect)

**4XX -** показывает что на стороне клиента произошла ошибка

**5XX -** показывает что на стороне сервера произошла ошибка

**↓ Типы Запросов ↓**

**GET Запрос -** [www.kondrich](http://www.kondrich) / category / top **?** name = max **&** age = 17  **-** служит для получение данных от сервера к клиенту

**POST Запрос** - www.kondrich / category / name\_category - служит для отправки данных от клиента на сервер (сохранение етих данных на сервер)

**PUT Запрос** - [www.kondrich](http://www.kondrich) / category / 1234 - служит для обновление запроса для отправки данных от клиента на сервер (изменение етих данных на сервере)

**DELETE Запрос -** [www.kondrich](http://www.kondrich) / category / 1234 - служит для удаления какой-то информации, которую отправил клиент на сервер (удаленные етих данных из серверв)

**Маршрутизация в Сети Интернет** - процесс опеределения найлучшего маршрута передачи данных от пользователя к серверу(и наоборот). Способ взаемодействия пользователся с сервером

**Протоколы** - набор правил(соглашений) с помощью которых сервера соединяються и обменються данными между собой

**Web Сервер (Apache , Nginx )** – програма запущеная на сервера, которая принимает постояно запросы от пользователся к сайту и перенапрявляет на обработку в фреймворки (Django) или языки програмирования (PHP)

**WSGI (Web Server Getaway Enterface**) - способ соединения(взаемодействия) web-сервера с python кодом в виде фреймворка Django. Передача запроса , который приняли от пользователя из web-сервера к фреймворку Django

**Local Host** **(127.0.0.1)** – доменное имя для компьютера пользователя. Позволяет использовать компьютер , как сервер где можна запускать программы или сайты

**Доменое Имя** - название сайта связаное с IP - адресом в интернете

**HTTP (Hyper Text Transfet Protocol)** - протокол по которым передаються страници написанные на HTML ( гипертекстовые документы) в интернете

**HTTPS (Hyper Text Transfer Protocol Secure)** - защищеный протокол по которым передаються страници написанные на HTML ( гипертекстовые документы) в интернете, который использует шифрование TLS

**Протокол Передачи Данных -** набор правил, по которым определяеться передача данные в сети интернет

**0.0.0.0 : 8000 --> IP адресс - 0.0.0.0 , Порт - 8000**

**IP (Inthernet Protocol)** - число, которое идентифицирует устройства в сети интернет. Ето устройство может быть пк , роутер , телефон , веб-сервер. Протокол сетевого уровня, его задача отправка пакетов от отправителя к получателю

**Порт** - число которое идентифицирует приложения , которые работают на устройствах в сети интернет

**Слушает Порт** - порт, который принимает на себя данные

**Сокет** - обьединение IP адреса и Порта, тоесть полный адресс

**Firewall (Брандмауер)** - програма для фильтрации пакетов нашего пк с удалеными сервера. Делит порты на свои и вражеские. Разрешеные порты свои , они подключены и от них можна принимать пакеты(данные) , а с вражеские порты нельзя подключать и следсвие нельзя принимать пакеты

**TCP / IP** - набор протоколов для обмена информацией в интернете. Связывание устройств происходит про IP адрессу компьютера(сервера) и Порту приложение, которые работают на нем. Внутри себя использует TCP , IP , UDP , HTTP , HTTPS , DNS протоколы

**TCP -** транспортный протокол с установлением соединение между устройствами и с гарантированой доставкой пакета. Если пакет не дошел, тогда посилаеться повторно

**UDP -** протокол без установление соединения между устройствмаи и без гарантированой доставкой пакетов

|  |  |
| --- | --- |
| Уровни Набора TCP / IP | Набор Протоколов TCP / IP |
| Прикладной | HTTP , HTTPS , FTP , DNS |
| Транспортый | TCP , UDP ↑ |
| Сетевой (Межсетевой) | IP (DHCP) ↑ |
| Уровень Сетевого Доступа (Канальный) | Ethernet , WLAN , WIFI ↑ |

Каждый уровень набора TCP / IP имеет свои протоколы передачи пакетов между устройствами. Каждый протокол зделан под свою задачу. Например передача пакетов между устройстами в одной сети или передача пакетов между устройствами в одном приложении

# **Django**

**Framework Django** – заготовка (каркас) для создания сайта, где внутри разработчик сам добавляет нужным функционал на сайте. Инструмент для back end разработки, взаемодействие пользователся с сервером на котором лежит сайт

**Node.js** - позволяет запускать код Java Script вне браузера

**Model , Temlate , View ( MTV ) -** три основные части Django приложений

**Model + Template => View => Ответ Пользователю**

**URL запроса пользователя -> URL Маршрутизатор -> View1(**предстваление1**)** или **Viev2(**представление2**) в зависимости от URL пользователя -> Формирование Ответа Пользователю, который состоит из Моделей + Шаблонов**

**Если URL запрос пользователя не найден тогда выводиться ошибка ERROR 404 - страница не найдена**

**Представление(View) -** формирует ответы для польователей основываясь на ихних запросах. Можно реализовать только или в виде функции или в виде класса

**Классы Представления (Class Based Views = CBV) -** позвоялет реализовывать больше функционала в представлениях. Можно прописывать разуную риализацию представление в записимости от типа запроса пользователся GET или POST

**Модели(Model) -** класс, который создает таблицу базы данных в проекте. Обьекты Класса - ето записи в етой таблице базе данных

**Django ORM (Object Relation Mapping)** - возможность взаемодейтствовать с моделями базы данных проекта при помощи языка Python, а не SQL

**Шаблоны(Templates) -** служат для генерации html кода, который будет показываться пользователю. Генерация внешнего вида ответа пользователю

**Приложения** - составные части сайта, которые не связаны между собой. Приложения реализуються как пакеты языка python

**Миграции** – действия, которые проделываем на базой данных. Все действия над базой храняться в папке migrations. Позволяет откатиться к предидущим версиям базы данных

**Пагинация Страниц** - постраничная навигация, разделение одной сплошной страници сайта на несколько и переключение между ними кнопками снизу

**Статические Файлы** - файлы CSS , JS , которые подключаються к проекту Django

**Медиа** – картинки или видео на нашем сайте

**CRUD (Create / Read / Update / Delete)** - создание / чтение / изменение / удаление записей в базе данных

**Индексы Базы Данных** - механизмы поиска в базе данных. Замедляет удаление, изменение и добавление значений, но ускоряет чтение етих значений.

**Менеджеры Обратного Связывания Записей Базы Данных** - механизм работы с базами данных при их вызове. По умолчанию ClassBase.objects

**Список Выборов(choices) -** когда вписываем в пользователю будут показываться одно значения, а в базу данных будут записываться другие

**Перечисления (Переопредиление Значений) -** когда опеределяем значения названием. И можем обращаться к етим значениям уже по указаным названиям. Можно переопределять в классе Meta (models.py)

**Поиски(Lookups)** - возможность прописывать характеристики для запроса из базы данных. Сопоставление по pk(id) или my\_property

**Формы Django** - пользовательский класс, который создан для автоматизации создания форм в html документах

**HTML Страница** – результатом работы фрейморка django являеться страница html, которая показываеться пользователю

Процесс Запроса  **==> Запрос Пользователя -> Web Сервер -> Django -> html страница**

Процесс Ответа **==>**  **html страница -> Django - > Web Сервер -> Ответ для пользователя**

**Виртуальнео Окружение** - изолирования среда для каждого проекта для использование конкретных версий модулей та интсрументов Django чтоб после обновления етих продуктов не возникали проблемы на сайте

**Аутентификация (Authentication)** - проверка подлиности (наличия ) пользователя по введеным данным с данными, которые находяться в базе данных. Существует ли такой пользователь в нашой базе данных

**Авторизация (Authorization)** - разрешение для пользователя на доступ к закрытой части сайта. Выполняеться после аутентификации

# **Django REST**

**Django Rest Framework (DRF)** - метод реализации RESTful API через framework Django. DRF позволяет создать API для нашего приложения

**Клиент -> Запрос -> Django REST -> file.json -> Возврат Клиенту file.json**

**API Запрос -> Маршрутизатор URL -> Представления -> Серилизатор -> База Данных**

**Сериализатор -** преобразовует файл .json формата в ячейки памяти моделей Django или наоборот. Выполянет также функцию создания , чтения , изменения или удаления записей в базе данных. Нужен для легкой передачи данных через API

**Сериализация Обьекта** - преобразования обьекта(ячейки в моделе django) в строку json

**Десериализация Обьектов** - преобразования строки json обратно в обьект (ячейку в моделе django)

**APIView(Представления)** - только обрабатывают все типи запросов от клинента на сайт

**ViewSet (Наборы Представлений)** - выполняют действия, которые можна указать для каждого методы. Делают комплексную работу. Наборы представлений хранят внутри себя все представления. Созданы для работы с роутерами, накже могут работать со статическими url адресами

**Mapping (Мапинг)** – связивание указаного типа запроса от пользователя с серверу с методом, которые определен в ViewSet

**Endpoint** - URL адресса с помощью которых frontend нашего сайта или другие сайты получаються данные в JSON формате

**Контролеры** - предсталвения в views.py которые отвечают за отправку данных в формате JSON

# **Celery**

**Celery** - обработчик задач на нашем веб-сайте, который выполянет задачи, которые делаеть пользователь асинхроно(паралельно)

**Таски (Tasks)** - задачи на нашем сайте, которые приходят от нашего пользователя. Загрузить картинку , скачать музыку , зделать денежный перевод. Ети задачи асинхронные

**Брокеры (Broker)** - создает очередь из принятых тасков для выполнения Воркером. Хранит ести такси внутри себя

**Воркер (Worker** или **Consumer) -** выполняет таски. Ожыдает поступления тасков от брокера

**Task -> Broker -> Worker -> Задача Выполнена**

**Внутри Celery НЕ можна работать с ORM Django**

**Flower -** позволяет наблюдать за выполеннием тасков не из консоли

# **Redis**

**Redis** **Сервер** - специальная база данных, котора хранит асинхронные операции на нашем сайте. По сути ето брокер для Celery. Данные в етой базе хранять по типу ключ - значение

**redis-cli** инструмент для работы с redis сервер

# **----------- Database - SQL -----------**

**Пользователь -> СУБД -> База Данных -> Жесткий Диск (Облако)**

**Можно делать с базами -> Create / Read / Update / Delete**

**Реляционные (Realitons - Отношения) Базы Данных** **-** берет информацию от пользователя -> структурирует -> записывает в хранилище. Упорядоченый набор данных, которые сохраняться по типу таблици и размещаються по строкам и столбцам. Одни таблицы могут ссылатся на другие таблици.

**Не Реляционые NoSQL Базы Данных** - упорядоченый набор данных, который может храниться по типу ключ - значение / документация / графы. Ето MongoDB - Cassandra

**СУБД (Система Управления Базами Данных)** - програма , которая отдает команды базе данных. Базы данных понимають чего хочет пользователь, а пользователь понимает что ответила база данных.Также отвечает за оптимизацию процеса извлечения данных из базы. Ето PostgreSQL - MySQL - Microsoft SQL Server

**SQL (Structured Query Language)**  - языки запросов для получения информации из базы данных.

**Отношения -** связи между несколькими таблицами

**Транзакции Базы Данных** - операция или операции над базой данных. Добавление , Чтение , Изменение , Удаление записей в базе

**Атрибут** - столбец в таблице

**Запись (Кортеж)** - строки в таблице

**Результирующий Набор** - результат запроса SQL из базы данных. Резульзат запроса всегда таблица

**Значение** - елемент таблици, который находиться на пересечении столбцов и строк

**Ключ** - столбец в таблице , который связывает разные таблици между собой

**Первичный Ключ (Primary Key)** - столбец, который индетифицирует каждую запись в таблице. Только уникальные значения

**Потенциальный** **Ключ** стобец, который который индетифицырует каждую запись в таблице , но с названием которое отличается от первичного ключа

**Внешний Ключ (Foreign Key)** - столбец, который ссылается на записи в других таблицах.

**SQL Операторы** - ключевые слова, символы которые используються для конкретных операций над базами данных

**Схема SQL (Schema SQL)** - опередяет стурктура базы данных. Указывает на размеры таблиц, значения строк и столбоц, также связь между таблицами

**Запрос SQL (Query SQL)** - запрос данных из базы данных (из таблицы). Запрос стоит на трех китах -> **Что делаем ? - Где Делаем? - Как делаем ?**

**Выборка -** когда пользователь берет данные из одной или нескольких таблиц одновременно

**Отношение Один ко Одному -** когда каждая запись в каждой таблице используеться только один раз. Создаеться третья таблица для етого

**Отношение Один ко Многим -** когда каждая запись в первой таблице используеться один раз , а записи из второй таблици - id могут повторяються. Вторая табличка какбы пренадлежит первой(самая популярная)

**Отношение Многие ко Многим** - каждая запись в каждой таблице может повторяться. Такое отношение только возможно в третьей(другой) таблице. В первой колонке третье таблици числа удит от меньше к большему, а во второй колонке как попало

↓ **Соединения** ↓

**Left Таблица -** таблица к которой будем добавлять таблицу

**Right Таблица** - таблица, которую будем добавлять

**Таблици соединяться по внешним ключам**

**INNER JOIN (Внутрешнее Соединение) -** к левой таблице добавляються значения по ключу из правой таблици. В левой таблице все значения, в правой только найденные

**LEFT JOIN (Левое Соединение)** - беруться все значения из текущей (левой) таблици и значения, которые будут найденые из правой таблици, если найденыне по ключу. Пустые значения заполняются NULL

**RIGHT JOIN (Правое Соединение)** - беруться все данные из правой таблички и данные которые будут найдение из левой таблички, если найденные по ключу. Пустые значения заполняются NULL

**FULL JOIN (Полное Соединение) -** беруться абсолютно все данные из двох таблиц, даже если нет ключей. Пустые значения заполняются NULL

**Cross Join (Декартовое Соеднение)** - соединение основаное на соединении каждого ключа из левой таблици со всеми ключами из правой таблици

# **------- Doker and DokerHub ---------**

**Doker** - технология упаковки , доставки и развертывания приложения на любом устройстве в виде контейнера. Главаная задача doker запустить приложение на любом устройстве. Не нужно настраивать приложения для каждого устройства. Имеет один механиз запуска для каждого приложения

**Виртуализация** - для запуска Doker в Windows нужно включать виртуализацию

**Контейнер** - виртуальные машины, которые хранит внутри себя код приложения, системные инструменты, среду выполнения , библиотеки, файлы конфигурации. Создан для того, чтоб приложение могло работать на любом устройстве, где установлен doker. Использует ядро операционой системы той на которой работает наше приложение

**Doker Файл (Doker File) -> Doker Image (Образ Приложения) -> Контейнер**

**Doker File (Сценарий)** - набор записанных комманд, который указывает что нужно ложить один в doker образ

**Docker Compose** - повзволяет запускать сразу нескольок контейнеров одновременно. Также связывает ети контейнеры между собой

**Doker Образ** или **Doker Image** - готовое к запуску приложение, которое внутри хранить весь код, инструменты, библиотети и настройки. Doker Образ хранит внутри себя другой Doker Образ, который являеться основой нашего приложения

**Doker Контейнер** - запущений екзепляр нашого приложения

**Сервис** - docker контейнер который используеться в docker-compose. Cервис изолирован от других и работает независимо от других

**Docker Registry** - сервер для хранение docker images (докер образов). В нашем случае ето Docker Hub

**Docker Deаmon** - название служби с помощью которой мы пользуемся функциями докер и взаемодействуем с его обьектами

**.dockerignore** - записываев папки и файлы, которые не должны попадать в docker imege проекта

**Переменные Окружения** - переменные, которые изменяют свое значение во время создания виртуального окружение на сервере. Ети переменные имеют указанные значение только при запущеном виртульном окружении. Такие переменные как SECRET\_KEY , ключи для баз данны (пользователи, пароли) , токены , адресса електронных почт

**Маппинг (Mapping)** - возможность обращаться пользователея сайта или другим сервисами к контейнерам из вне. По-умолчанию docker контейнеры запускаються изолировано, поетому нужно связывать его с локальными сервисами такими как PostgreSQL , Nginx , Redis , Celery

**DokerHub** - облако в котором можна хранить doker образы для запуска приложений, или скачивать

**Volume -** позволяет сохранять данные даже после удаление контейнера. Перекидывает данные из нашего локального проекта в docker контейнер

# **----------- Nginx ------------**

**Nginx** - служит для направления запросов от пользователей к сервепу. Позволяет создавать прокси для сервера. Делать кеширование данных, которые принимает сервер от пользователя. Может также выступать web-серевером для сайта

# **----------- Git and GitHub ------------**

**Git -** система розпределеного контроля различных версий кода. Контроль и отслеживание изменений в коде за период жизни проекта. Не хранит в себе файлы, только отслеживает изменения

**GitHub** - облачное хранилище предоставляющее доступ для совместной работы програмистов над проектов. Хранит в себе git - репозитории. Для глобальной разработки

**Рабочая Директория** - название папки с проектом, которая хранит состояние файлов проекта и находиться под контролем версий git

**Git Репозиторий** **.git** - область, где происходит само управление версиями проекта. Храниться внутри скрытой папки **.git** Делиться на две зоны **↓**

**Staging Area Репозиторий (черновая зона)** - область репозитория, где храниться индексный файл. Перед добавлением в commit файлы сохраняються здесь. При разовом добавлении файла и зделаному коммиту будет храниться всегда в етой области, если только не удалим сам файл из проекта.

**Commits Area Репозитория (зона коммитов) -** область репозитория, где содержаться комиты проекта

**Commit** - сохранение состояния проекта, кода. Каждый коммит имеет свой идентификатор

**Хеш Коммита** - уникальное название для каждого коммита -> 06beccaf9af64ea48b1ffd6

**Файл -> Staging Area Репозитория -> Commits Area Репозитория -> GitHub (сохранение в облаке)**

**Ветка -** паралельное состояние проекта с другими изменениями в коде. Нужен для предотвращениея ризка сломать текущую версию проекта (+ добавить баги). Можно работать

**Ветка master ( main )** - название первой ветки в репозиторее, по-умолчанию все комиты сохраняються в ету ветку

**Merge ( слияние )** - обьединение воедину розделеную историю двух веток, они становятса одинаковыми

**Указатель ( Head -> )** - указывате на определенный коммит в текущей ветке. Каждая ветка имеет свой указатель. По-умолчанию указывает на последний коммит в текущей ветке. Можна переключать указатель на любой выбранный коммит

**Смещенный Указатель** - показывает не на последний коммит в текущей ветке, а на указаный в етой ще ветке

**.gitignore** - папка содержит внутри файлы которые на которых не будет контроля версий git и которые не будут записываться в коммиты ( например файлы логинов - файлы настройки ide). Git не может взаемодествовать с етими файлами. Работает один раз только после самого создания файла и еще не добавлен в stage (потом не работает)

**Git Stash** - хранит в себе не подготовленные и не закоммиченные изменения в проекте. Можно создавать несколько стешей. Можно использовать одновременно только один стеш

**Git Tags** - теги позволяют зафиксировать в истории ветки оперделенные моменты разработотки (сообщения, подсказки)

**Легковесные Теги** - указатель с сообщением

**Анотированные Теги -** полноценные обьекты git, имеют свой хеш, автора и дату когда был зделан

# **----------- Криптография ------------**

**Хеширование** - преобразование любого обьема данных в уникальынй набор символов, который связан с етими данными. Набор символов имеет фиксированую длину. При незначительном изменении входной информации ее хеш полностью меняеться. Хеширование нобратимо , нельзя вернуть данные преобразовав их хеш. В Python хешируються только неизменяемые обьекты.

**Хеш**  - уникальный набор символов фиксированой длины, который выходит после преобразования входящих данных

**Создан Для** - Проверки целосности файлов, цифрових подписей. Хеширование паролей для невозможности разгадивания самого значения пароля. Пароли храняться в базах данных всегда в виде хешей

**Равные Обьекты** - если обьекты равные a == b, то их хеш тоже равный

**Равные Хеши** - если хеши равны hash(a) == hash(a) , они не гарантируют равенство обьектов

**Не Равные Хеши** - если хеши не равны hash(a) != hash(b) , они гарантируют неравество обьектов