# **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

# **Дополнения**

**API (Application Programming Interface)** - интерфейс для взаемодейтвия различных програм между собой или програм с пользователями по опеределеным правилам

# **Структура Кода**

**Модуля -** добавление файла к коду, который розширяет функцыонал програмирования (новые методы и свойства). Также деление кода на различные файлы.

**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

**Операнды** - значения на которые действует оператор

**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. 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.

**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) -** показывает какие методы будут вызываться при множественном наследовании. Показывает порядок вызова етих методо и поряд наследования класов

**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 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

# **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**

**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 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 pethon 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

# **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

**HTML Страница** - результатом работы фреймоврка 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**

**Шаблоны Jinja -** позволяют вставлять переменные python непостредственно в html документ та выводить на стринце в браузере

# **----------- 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

**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

**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 для создания и управлениями контенерами

**Контейнер** - 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 -> Образ -> Контейнер**

**Doker Файл** - набор инструкций записаный програмистом, который указывает что нужно ложить в doker образ

**Doker Образ** - готовое к запуску приложение, которое внутри хранить весь код, инструменты, библиотеик и настройки

**Doker Контейнер** - запущений екзепляр нашого приложения

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

# **----------- 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 -> GitLab (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. При переключении веток по-умолчанию указывает на последний коммит в новой ветке. У каждой ветки свой указатель. Может указывать на любой выбранный коммит

**Смещенный Указатель** - показывает не на последний коммит в ветке. Не имеет ничего общего ни з какой веткой

**.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

**Быстрое Слияние ( Fast - Forward ) -** тип слияния **-** ветка master и ветка second\_branch не имеют разных паралельных коммитов. Слияние не создает новых коммитов.

**git merge second\_branch -** указатель текущей ветки начинает указывать на последний коммит second\_branch. После слияния обе ветки указывают на один коммит. Показывает историю двох веток одновременно. Не создается новый коммит (не нужно создавать). Можна откатить изменения git reset --hard HEAD~1 , просто перемещается указатель текущей ветки - коммит не удаляется↓

**master** --> ( m\_1 ) -> ( m\_2 ) ↓

**( sb\_1 ) <- HEAD-M & HEAD-S**

**second\_branch** --> ( m\_1 ) -> ( sb\_2 ) -> ( sb\_1 )  ↑

**git merge --squash second\_branch** - копируються данные из second\_branch в текущую ветки. Указатель не перемещается. Показывает историю только текущей ветки. Не создается новый коммит (нужно создавать). Можно откатит изменения git reset --hard HEAD~**1-2-3** удалит созданный нами коммит и переместить указатель по текущей ветке назад ↓

**master**  --> ( m\_1 ) -> ( m\_2 ) -> **( changes\_master\_3 )** -> **<- HEAD-M**

**second\_branch** --> ( m\_1 ) -> ( m\_2 ) -> ( sb\_1 ) -> ( sb\_2 ) **<-** **HEAD-S**

**Рекурсивное Слияние** - тип слияния - ветка master и ветка second\_branch имеют разные паралельные коммиты. Создается новый коммит в текущей ветке, который состоит из двох последних коммитов обеих веток. Указатель текущей ветки указывает на етот новосозданный коммит. Имеем историю двох веток одновременно. Название етому коммиту дает сам git ( Merge branch “second\_branch”). Можно отменить слияние git reset --hard HEAD~**1**

**git merge --no-ff second\_branch** - указываем рекурсивное слияние в независимости от того, есть ли разные паралельные коммиты в обеих ветках

**git merge second\_branch** - git автоматичесик выполнит рекурсивное слияние, если точно есть разные паралельные коммиты в обеих ветках ↓

**master** --> ( m\_1 ) -> ( m\_2 ) -> *( m\_3 )* -> ( ------ ) -> **( new\_commit\_in master\_4 ) <- HEAD-M**

**second\_branch** --> ( m\_1 ) -> ( m\_2 ) -> ( sb\_1 ) -> ( sb\_2 )**<- HEAD-S**

**Rebase** - тип слияния - изменяет разположения коммитов в ветках - пребазирование коммитов. Не перемещает коммиты, а создает копии из выбраной ветки в текущую. Новых коммитов не создает.Указатель указывате на последний коммит в текущей ветке. Коммиты из выбраной ветки перемещаються назад етого коммита. Изменяеть идентификаторы коммитов в текущей ветке. Обычно использують git rebase master в отношении second\_brach для пемещения(копирования) важных коммитов из master ветки в second\_branch ветку. Не Рекомендуется Использовать

**git rebase second\_branch** берет копии коммитов из текущей ветки и ставвит их повверху указаной второй ветки second\_branch . Определяет последовательность коммитов по времени их создания. После слияния указатели обеих веток указывають на один самый последний коммит по времени ↓

**master** --> ( m\_1 ) -> ( m\_2 ) -> *( m\_3 )* ->

**=> ( m\_1 ) -> ( m\_2 ) -> ( sb\_1 ) -> (s\_b2) -> ( m\_3 ) <- HEAD-M**

**second\_branch** --> ( m\_1 ) -> ( m\_2 ) -> ( sb\_1 ) -> *( sb\_2 )* **<- HEAD-S**

**Chery-Pick** - слияние указаного коммита из выбраной ветки с коммитом из текущей ветки. Копия слитого коммита становиться последней в ветке и на него указывает указатель текущей ветки. Не переносятся коммита , а создаються копии коммитов с новыми уникальными индетификаторами

**Теги** - дают доплнительные названия для коммитов. Помечают важные етапы в разработке проекта

**Легковесные Теги** - добавление нового названия на разположение коммита в ветке и обращение к нему по тегу

**Анотированные Теги** - тег с текстом, который прикреплен к коммиту. Сщитаються полноценными обьектами в git

# **----------- 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