# **--------- Python -----------**

# **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

**Сортировка Елементов** - количество елементов не изменяеться, меняеться порядок елементов по определеному указаному признаку

**Фильтрация Елементов** - количество елементов могут изменяться, последовательность тоже может изменяться

**Big-O Notation** - метод 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

**CRM-система (Customer Relationship Managment)** - автоматизиурет процесы на нашем сайте(преприятии)

# **Структура Кода**

**Модуля -** файл с розишрением **.py** , который вы добавляем в другой файл. Добавляем функционал из подключаемого файла в наше текущий 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 Python -------------**

**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.