Primeros pasos en el equipo de trabajo **DEV-Pro**

Issue 1

Actualizado el 09/04/2024

Antes de nada bienvenido al equipo de trabajo DEV-Pro este documento es una ntroduccion a los conceptos básicos que usaras a diario en el equipo, espero que te ayude a integrarte y ¡Adelante!

Antes de empezar con los conceptos me gustaría definirte que es Python y porque lo usamos en el equipo.

Python es un **lenguaje de programación** que tiene las 3 Bs (**bueno, bonito y barato**):

Es **bueno** porque es un lenguaje que puede con casi todo, se utiliza en aplicaciones webs, desarrollo de software, ciencia de datos y el machine learning (ML) y se integra muy bien con los sistemas y aumenta la velocidad de desarrollo.

Es **bonito** porque es eficiente y fácil de entender, es muy visual y se entiende el código rápidamente.

Es **barato** ya que es gratuito y se descarga en su web oficial (https://www.python.org/downloads/).

Clases en Python

Cuando empieces en el equipo te daras cuenta que usamos clases en programas y te preguntaras el motivo, pues paso a describírtelos

Una **clase** en Python es una **estructura** de programación que permite definir un conjunto de métodos y atributos que describen un objeto o entidad. Las clases son un **concepto fundamental** en la programación orientada a objetos, que se utilizan para modelar entidades del mundo real o abstracto en un programa de computadora.

Te esteras preguntando, ¿Para qué sirven las clases en Python?

Pues una clase no es mas que una definición de una plantilla o molde para crear objetos, los cuales son instancias de esa clase. Los objetos creados a partir de una clase tienen las mismas propiedades y comportamientos definidos por la clase, pero pueden tener valores diferentes para los atributos que se definen en la clase.

En Python, una clase se define mediante la palabra clave «class», seguida del nombre de la clase y dos puntos (:) y luego el cuerpo de la clase. El cuerpo de la clase contiene definiciones de métodos y atributos, que pueden ser públicos o privados según su acceso.
Te adjunto un ejemplo para que veas estos conceptos en la vida real

```
CLASS PERSONA:

    DEF __INIT__(SELF, NOMBRE, EDAD):

        SELF.NOMBRE = NOMBRE

        SELF.EDAD = EDAD

        DEF SALUDAR(SELF):

            PRINT("HOLA, MI NOMBRE ES " + SELF.NOMBRE)
```

Ventajas y desventajas del uso de las clases en Python

Ventajas

- **Reutilización de código:** las clases pueden reutilizarse en diferentes partes del programa o en distintos programas, lo que ahorra tiempo y reduce la duplicación de código.
- **Encapsulación:** permiten ocultar la complejidad de un objeto y exponer solo una interfaz simple y fácil de usar para interactuar con él.
- **Modularidad:** pueden descomponer un programa en componentes más pequeños y manejables, lo que facilita el mantenimiento y la solución de problemas.
- **Polimorfismo:** ayudan a implementar el mismo conjunto de métodos con diferentes comportamientos para distintos tipos de objetos, lo que permite una mayor flexibilidad y extensibilidad en el diseño de programas. (lo veremos más adelante con más detalle)

Desventajas

- **Sobrecarga de complejidad:** las clases pueden agregar complejidad adicional a un programa y hacer que sea más difícil de entender y depurar.
- **Curva de aprendizaje:** el aprendizaje de las clases y la programación orientada a objetos en general pueden requerir una curva de aprendizaje más pronunciada para los programadores principiantes.
- **Uso innecesario:** a veces, las clases se utilizan innecesariamente en situaciones en las que una función simple podría haber hecho el trabajo de manera más eficiente.

¿Qué método se ejecuta automáticamente cuando se crea una instancia de una clase?

Es un método especial (también llamado **método dunder**) en Python que se utiliza para inicializar una instancia de una clase. Cuando se crea una instancia de una clase, el método **__init__** es llamado automáticamente por el intérprete de Python y se utiliza para realizar cualquier inicialización que sea necesaria para la instancia.

El método __init__ se usa para asignar valores iniciales a los atributos de una instancia de la clase. Los atributos son las variables que pertenecen a una instancia particular de la clase. Al llamar al método __init__, podemos establecer los valores de estos atributos y configurar la instancia de la clase para su uso posterior.

Por ejemplo, supongamos que queremos crear una clase llamada Persona con dos atributos: nombre y edad. Podríamos definir la clase de la siguiente manera:

```
CLASS PERSONA:

    DEF __INIT__(SELF, NOMBRE, EDAD): SELF.NOMBRE = NOMBRE
    SELF.EDAD = EDAD
```

Ahora podremos llamar a la clase de la siguiente forma

```
PERSONA1 = PERSONA("JUAN", 30)
```

Al final de este documento te mostrare más métodos dunder.

¿Cuáles son los tres verbos de API?

Antes de responder a esta pregunta, seguro que te estas preguntando que es una API, pues bien una **API** no es más que Interfaz de Programación de Aplicaciones (**A**pplication **P**rogramming **I**nterface en inglés).

Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero. Las API le otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones; y ofrecen oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales).

A veces, las API **se consideran como contratos**, con documentación que **representa un acuerdo entre las partes**: si una de las partes envía una solicitud remota con cierta estructura en particular, esa misma estructura determinará cómo responderá el software de la otra parte.

Las API son un medio simplificado para conectar su propia infraestructura a través del desarrollo de aplicaciones nativas de la nube, pero también le permiten compartir sus datos con clientes y otros usuarios externos. Las API públicas aportan un valor comercial único

porque simplifican y amplían sus conexiones con los partners y, además, pueden rentabilizar sus datos (un ejemplo conocido es la API de Google Maps).

Aunque más adelante te explicare los verbos más comunes a usar en una API (en la sección Postman), te los adelanto:

- **GET**: sirve para obtener (leer) datos. (por ejemplo una lista de usuarios de un servidor)
- **POST**: sirve para crear nuevos datos.
- **DELETE**: sirve para eliminar datos.

¿Es MongoDB una base de datos SQL o NoSQL?

No te asustes por el nombre, **MongoDB** es una **base de datos NoSQL** orientada a documentos que apareció a mediados de la década de 2000. Se utiliza para almacenar volúmenes masivos de datos.

A diferencia de una base de datos relacional SQL tradicional, MongoDB **no se basa en tablas y columnas**. Los datos se almacenan como colecciones y documentos.

Los documentos son **pares value/key** que sirven como **unidad básica de datos**. Las colecciones contienen conjuntos de documentos y funciones. Son el equivalente a las tablas en las bases de datos relacionales clásicas.

Los documentos **no** tienen un **esquema predefinido** y los campos pueden añadirse a voluntad. El modelo de datos disponible en MongoDB facilita la representación de relaciones jerárquicas u otras estructuras complejas.

Otra característica importante de MongoDB es **la elasticidad de sus entornos**. Muchas empresas tienen clusters de más de 100 nodos para bases de datos que contienen millones de documentos.

A continuación te explico cómo se usa MongoDB como es su arquitectura y sus definiciones básicas

La arquitectura de MongoDB se basa en varios **componentes principales**. En primer lugar, **«_id» es un campo obligatorio** para cada documento. Representa un **valor único** y puede considerarse como la **clave principal del documento** para identificarlo dentro de la colección.

Un documento es el equivalente a un registro en una base de datos tradicional. Se compone de campos de nombre y valor. Cada campo es una asociación entre un nombre y un valor y es similar a una columna en una base de datos relacional.

Una colección es un grupo de documentos de MongoDB, y se corresponde con una tabla creada con cualquier otro RDMS como Oracle o MS SQL en una base de datos relacional. Como ya mencione antes, tienes que recordar que no tiene una estructura predefinida.

Una base de datos es un contenedor de colecciones, al igual que un RDMS es un contenedor de tablas para las bases de datos relacionales. Cada uno tiene su propio conjunto de archivos en el sistema de archivos. **Un servidor MongoDB puede almacenar múltiples bases de datos**.

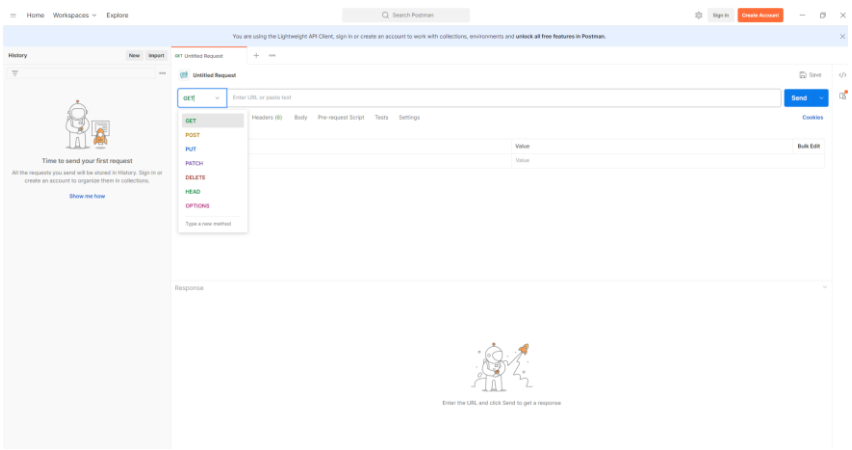
Por último, **JSON** (JavaScript Object Notation) es un formato de texto plano para expresar datos estructurados. Está soportado por muchos lenguajes de programación.

¿Qué es Postman?

Postman es una herramienta de **colaboración y desarrollo** que permite a los desarrolladores interactuar y probar el funcionamiento de servicios web y aplicaciones, proporcionando una interfaz gráfica intuitiva y fácil de usar para **enviar solicitudes a servidores web y recibir las respuestas correspondientes**.

Con esta herramienta se puede gestionar diferentes entornos de desarrollo, organizar las solicitudes en colecciones y realizar pruebas automatizadas para verificar el comportamiento de los sistemas.

Postman es utilizado por los desarrolladores para **testear colecciones y catálogos APIs** (tanto a nivel *front-end* como *back-end*), para gestionar el ciclo de vida de las APIs, mejorar el trabajo colaborativo y mejorar la organización del proceso de diseño y desarrollo.



Después de esta definición te estarás preguntando porque se elige Postman, a continuación te dejo las principales características y funcionalidades de Postman:

Envío de solicitudes. Permite enviar solicitudes GET, POST, PUT, DELETE y otros métodos HTTP a una API especificando los parámetros, encabezados y cuerpo de la solicitud.

Gestión de entornos. Facilita la configuración para diferentes entornos (por ejemplo, desarrollo, prueba, producción) y el cambio sencillo entre ellos (para realizar pruebas y desarrollo en diferentes contextos).

Colecciones de solicitudes. Agrupa las solicitudes relacionadas en colecciones, lo que facilita la organización y ejecución de pruebas automatizadas.

Pruebas automatizadas. Es ideal para crear y ejecutar pruebas automatizadas para verificar el comportamiento de una API (detectar errores e incrementar la calidad del software).

Documentación de API. Genera de forma automatizada, documentación detallada de la API a partir de las solicitudes y respuestas realizadas, lo que facilita su comprensión y uso por parte de otros desarrolladores.

Cada vez son más los desarrolladores y programadores que apuestan por un entorno como Postman para automatizar pruebas y mejorar sus procesos de trabajo. Los principales beneficios que se obtienen con esta herramienta son:

Facilidad a la hora de trabajar al disponer de una interfaz gráfica de usuario intuitiva, sencilla y personalizable.

Amplia compatibilidad con numerosas tecnologías y protocolos web, como por ejemplo; HTTP, HTTPS, REST, SOAP, GraphQL… (lo que permite interaccionar con diversos tipos de API sin complicaciones o problemas).

Ofrece una **amplia gama de funcionalidades para diseñar, probar y documentar APIs**, siendo probablemente la solución más completa del mercado para gestionar el ciclo de vida completo de desarrollo de APIs.

Fomenta y facilita la colaboración entre los miembros del equipo de desarrollo (con opciones interesantes como compartir colecciones de solicitudes con otros desarrolladores). Cuenta con una **comunidad amplia de usuarios que está en constante crecimiento** y que aporta una gran cantidad de recursos, como tutoriales, documentación, foros y grupos de discusión…

Se integra perfectamente con varias herramientas populares utilizadas en el desarrollo de software. Por ejemplo, se puede conectar con sistemas de control de versiones como GitHub, servicios de generación de documentación como Swagger o herramientas de automatización de pruebas como Jenkins, entre muchas otras.

Permite a los usuarios **agregar scripts personalizados utilizando JavaScript** (para automatizar tareas repetitivas, configurar pruebas avanzadas o agregar validaciones personalizadas a las respuestas de la API).

Las colecciones son una característica central de Postman que permite **organizar y agrupar solicitudes relacionadas**. Esto simplifica la administración de API complejas y facilita la reutilización de solicitudes y flujos de trabajo en diferentes proyectos.

Importante tener en cuenta que para saber si hemos tenido éxito o no al ejecutar nuestra solicitud, el programa nos devolverá un código. Te resumo los códigos principales:

200 : Éxito

400 : Mala petición

401 : No autorizado

¿Qué es el polimorfismo?

El **polimorfismo** es una técnica de la programación orientada a objetos que permite a distintos objetos responder de manera diferente a un mismo llamado de método. En Python, esto se logra gracias al uso de **clases y funciones**, lo que aumenta significativamente la **flexibilidad** de nuestras implementaciones.

Una de las ventajas del polimorfismo es que nos permite escribir código más genérico, lo que a su vez nos permite reutilizar nuestro código en una variedad de situaciones.

Veamos un ejemplo para entenderlo mejor. Supongamos que tenemos una clase Figura, que tiene un método abstracto area(). La idea detrás de esta clase es que cualquier figura que queramos modelar, sea un cuadrado, un círculo, un triángulo, etc., siempre tendrá una propiedad de área. Entonces, podemos crear una clase Cuadrado que herede de Figura y defina su propia implementación de area(), que calcularía el área del cuadrado. Lo mismo podemos hacer para otras figuras, como un Círculo o un Triángulo.

```
class Figura:
    def area(self):
        pass

class Cuadrado(Figura):
    def __init__(self, lado):
        self.lado = lado

    def area(self):
        return self.lado * self.lado
```

Una vez que hemos definido nuestras clases, podemos crear un método que acepte cualquier objeto de tipo Figura, y usar el método **area()** para calcular el área de esa figura particular:

```
def calcular_area(figura):
    return figura.area()
```

Ahora, podemos crear cualquier objeto de tipo Figura y pasarlo a nuestro método **calcular_area()**.

```
cuadrado = Cuadrado(5)
circulo = Circulo(3)
triangulo = Triangulo(4, 5)

print(calcular_area(cuadrado))
print(calcular_area(circulo))
print(calcular_area(triangulo))
```

¿Qué es un método dunder?

Son **métodos especiales**, también conocidos como **métodos Dunder o métodos Mágicos**. Se utilizan para emular el comportamiento de las funciones integradas.

Estos métodos tienen un significado particular para el intérprete de Python. Sus nombres empiezan y terminan en __ (doble guión bajo). Por ejemplo **init** (ya explicado en párrafos anteriores).

Normalmente estos métodos no son invocados directamente por el programador. Por ejemplo cuando haces una simple suma 2 + 2 se está invocando al método __add__ internamente.

Métodos de iniciación y constructores

__init__ Inicializa un objeto

Crea un nuevo objeto cuando se llama a la instancia de una clase.

```
class Car(object):
    def __init__(self):
        ...
    def __repr__(self):
        ...
```

__new__ Crea un objeto

__del__ Elimina un objeto

Métodos mágicos de comparación

__lt__ a < b
__gt__ a > b
__le__ a <= b
__ge__ a >= b
__ne__ a != b
__eq__ a == b

Métodos mágicos para matemáticas

__add__ obj + …
__sub__ obj - …
__mul__ obj * …
__floordiv__ obj //
__truediv__ obj /
__mod__ obj %
__pow__ obj ** …

Otros Métodos mágicos

__str__ Pretty print object. Devuelve una cadena de carácteres. Representación Legible para usuarios.
__repr__ Devuelve una cadena de carácteres. Representación no ambigua útil para desarrolladores.
__len__ Devuelve la cantidad de elementos que tiene una lista.

¿Qué es un decorador de python?

En Python los **decoradores** son una manera elegante de **extender o modificar el comportamiento de funciones y métodos sin cambiar el código fuente**. Se trata de una función que toma como referencia otra función y extiende sus atributos y métodos, es decir, extiende el comportamiento de la función original simplificando la sintaxis y **encapsulando sus funcionalidades.**

Ahora que ya sabes para qué sirven y qué son los decoradores en Python, vamos a ver cómo hacer uno de manera simple. Para ello, veremos paso a paso, qué es lo que tienes que considerar. Lo primero que debes saber es que crear un decorador implica definir una función que toma otra como argumento y devuelve una completamente nueva. Para hace esto hay que seguir estos pasos:

- Define la función que actúa como decorador mediante el nombre que quieras.
- Crea la función interna dentro del decorador que se llama ‘wrapper’. Esta será la que envuelva la función original.
- Personaliza el wrapper y agrega la lógica adicional que necesites. Esto puede ser, ejecutar el código antes o después de hacer la llamada a la función o modificar la entrada o manipulación del resultado.
- Llama a la función original desde wrapper ya que es lo que garantiza que la funcionalidad original se ejecute
- Devuelve el wrapper desde la función del decorador para que este funcione correctamente
- Aplica el decorador usando @ seguido del nombre que le hayas dado justo encima de la definición de la función.

```python
def my_decorator_name(name):
    def my_custome_decorator(function):
        def wrapper(*args, **kwargs):

            print('Name:', name)
            return function(*args, **kwargs)

        return wrapper

    return my_custome_decorator

@my_decorator_name('CodigoFácilito')
def suma(a, b):
    return a + b
```

Con esto finalizas tu introducción al equipo, ¡¡¡Mucho Ánimo y Bienvenido!!!