INTRODUCCIÓN A PYTHON

CLASE 8

Contenido

“

Este espacio curricular permite fortalecer el espíritu crítico y la actitud creativa del futuro graduado. Lo alienta a integrar los conocimientos previos, recreando la práctica en el campo real, al desarrollar un proyecto propio.

”

Encuentro 7

Cuando empezamos a trabajar con clases y objetos, experimentamos uno de los primeros avances en programación. El impacto en nuestro código es significativo, ya que el uso adecuado de estas clases nos permite desarrollar códigos más legibles y fáciles de mantener. En esta sección, ampliaremos nuestro conocimiento sobre el paradigma orientado a objetos. Además, abordaremos brevemente el sistema de Python para manejar errores en tiempo de ejecución.

Introducción

Concluyendo esta primera introducción a Python y la programación estructurada, discutiremos acerca de los módulos y su potencial como bibliotecas de trabajo. Al desarrollar, resultan muy útiles ya que nos permiten crear nuestras propias herramientas para proyectos futuros. Asimismo, abordaremos la importancia de gestionar los datos de entrada en nuestros programas. Por último, un tema adicional: exploraremos cómo almacenar la información relevante del juego en archivos de texto.

Paradigmas

Un paradigma de programación es, podríamos decir, un estilo de programación que adoptamos para nuestras soluciones. Este estilo o método de programación, no es otra cosa que un conjunto de acuerdos sistémicos que se aplican en los distintos niveles de nuestro código

con el surgimiento de los paradigmas, los lenguajes se pensaban para un paradigma especifico, pero Python, es muy distinto a todos. en este sentido, algunos lenguajes sólo abonan a un paradigma, mientras que otros lenguajes a más de uno. en referencia a esto último, es donde Python saca ventaja.

Estructurado

Este paradigma, define estructuras que determinan el flujo de ejecución. Este flujo se define por las características de dichas estructuras, módulos o bucles. Claro, existió un paradigma previo al estructurado, y fue con el surgimiento de este que dejamos de utilizar el GOTO. Lenguajes como C o pascal, son clásicos en este tipo de paradigma. a eta altura, todos coincidimos que Python también me permite aprovechar las directivas del paradigma de programación estructurado. claro, lo vimos en los anteriores encuentros. Pero, no conforme con eso, Python nos deja más tela para cortar.

Paradigma Orientado a objetos

Aquí es donde se nos queman los papeles, todo lo que vimos hasta el momento pareciera no ser relevante. Con este nuevo paradigma, aparecen las clases como recursos necesarios para organizar nuestro código. el cómo miramos nuestro entorno y analizamos el diseño del programa, cambio por completo. Es este momento es cuando aparecen las abstracciones, tenemos que representar los elementos que observamos y eso no es una tarea simple. Nuestra responsabilidad, como desarrolladores, es diseñar cada una de esas abstracciones.

La programación orientada objetos, mejorar la legibilidad y nos permite disminuir la cantidad de errores. Además, nos facilita la reutilización del código y su escalabilidad. siguiente con este tema, presentamos algunos ejemplos de lenguajes orientados a objetos como C++, C# o Python.

Para entender que es el paradigma orientado a objetos, vamos a describir algunos puntos que se relevantes, estos conceptos son:

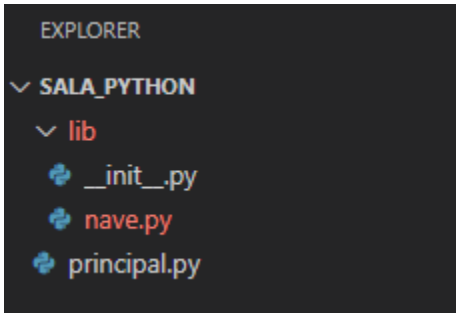
- Abstracción de datos
- Encapsulación
- Herencia
- Polimorfismo

Ingenuo paralelismo de módulos a clases y objetos

Aunque solo es un artilugio pedagógico, debido a que son cosas totalmente distintas. Vamos a intentar comparar lo que venimos hasta el momento, la programación estructurada, para poder comprender este nuevo paradigma.

Sí volvemos atrás, en el Paradigma Estructurado, ordenamos las funciones en módulos o librerías. Por ejemplo, podemos tener una librería llamada naves que utilizamos mucho en nuestro programa. Sí utilizamos, lo que aprendimos, podemos declarar las variables que representan y describen nuestra nave. En este caso, podríamos encontrar valores y las funciones al describir que es la nave y que cosas que la nave puede hacer. al considerar esto último, el código podría ser algo similar a lo siguiente:

#creamos una carpeta con la librería con código para la nave



El código dentro del archivo sería como el siguiente:

```
velocidad=5
cantodad_balas=100

def disparar():
    #codigo de la funcion
```

Esto parece ser una buena solución, pero...

¿Qué pasa si tenemos que crear muchas naves?

Bien, acá entra la programación orientada a Objetos. el concepto de clase, lo entenderíamos como un molde para construir cosas. Por ejemplo, si necesitamos usar naves en nuestro programa, deberíamos corear una clase nave. Esa clase, será el molde para instanciar otras naves. El código, sería muy similar a lo que mostramos a continuación:

```
class nave:
    velocidad=5
    cantida_balas=100

    def disparar(self):
        #codigo de disparar
```

Ya con ese molde, creamos las instancias que serán las naves de nuestro programa. Luego de crear una instancia de la clase nave, vamos a tener ahora si... nuestros objetos (o muchas naves 😊)

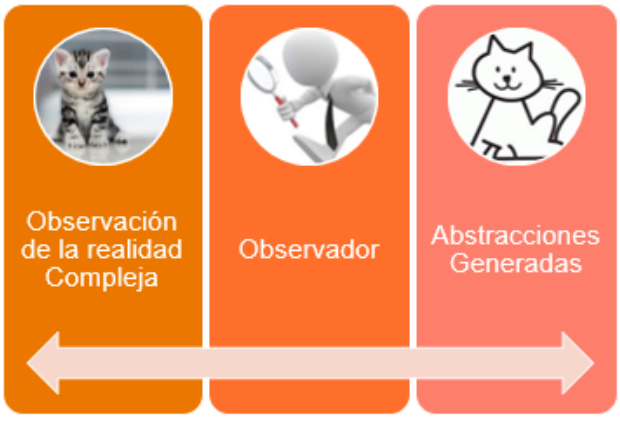
```
nave1=nave()
nave2=nave()
```

Pilares de la programación orientada a Objetos

La **Programación Orientada a Objetos (POO)** emerge como un paradigma fundamental, ofreciendo un enfoque innovador para el desarrollo de software. A diferencia de los métodos tradicionales, la POO se basa en la creación de **objetos**, entidades que encapsulan datos y comportamiento, imitando el funcionamiento del mundo real. Este paradigma, le permite al equipo de desarrollo cumplir con la misión de producir una ilusión de simplicidad al momento de crear soluciones tecnológicas.

Abstracción

Este es uno de los aspectos más importantes, cuando decimos abstracción, en el paradigma orientado a objetos, hablamos de las clases. Estas abstracciones, representan distintos tipos de entidades. Un ejemplo de esto se podría visualizar en la gráfica siguiente.



Encapsulación

Una de las cosas más evidentes en POO, es que el código queda escondido dentro de los objetos. Es por eso, que solemos decir "El código se encuentra encapsulado dentro del objeto".

Esto significa, que se puede interactuar con el objeto, pero "no sabemos" cómo resuelve nuestra petición. entendamos que, hablamos de los objetos interactuando entre sí, es claro que el programador si puede ver el código fuente.

Instancia

Venimos hablando de observar el mundo real, para diseñar una abstracción de algún elemento observable, debemos crear una instancia. Luego de esto, y saltando muchos pasos en la teoría de Objetos, convertimos las clases en objetos. Cuando tenemos la clase, podemos crear todos los objetos que necesitemos. La acción de crear un objeto se llama **instanciar**.

Las clases y los Objetos

Para simplificar la teoría de objetos, vamos a decir simplemente que como programadores debemos crear clases y objetos… fin del comunicado 😊. Las clases serán únicas, no se repiten. Pero, por otro lado, los objetos si… se pueden repetir. tomando las clases programadas, podemos crear todos los objetos como necesitemos.

Definir una clase

Las clases son descripciones de una abstracción de elementos similares. La descripción consta de expresar las características comunes que observamos. Las acciones que esos elementos puede implementar se programan como parte del comportamiento.

A modo de práctica, nos hacemos la siguiente pregunta

¿Qué pasa si observamos este grupo de animales que se muestran en la foto?

¡¡Opiniones!!

Podemos notamos que son muy parecidos, creo, todos negros y tiene rayas blancas (*menos Marty que es blanco con rayas negras* 😊). como conclusión, podemos notar que todos pertenecen a una misma familia de animales. Esto último, nos acerca a pensar que se puede describir a cada uno de la misma manera, teniendo descriptores comunes y posibles datos diferentes. Por ejemplo, cada animal tiene:

- Color: blancos con rayas negras
- Cantidad de patas: 4
- Altura: 1 metro
- Peso: 180 kg

Y por otro lado podemos describir lo que pueden hacer:

- Correr
- Saltar
- Comer

De esta forma, creamos un molde para describir un conjunto de animales similares. Entonces que me ofrece teoría de objetos, el paradigma nos ofrece herramientas para describir sus propiedades y sus acciones. por ejemplo, cuál sería la descripción de una cebra:

Cebra

Atributos o propiedades

- Color
- Cantidad de patas
- Altura
- Peso

Acciones o métodos

- Correr
- Saltar
- Comer

Interfaz o contrato

Al detallar las características y funciones de la clase, podemos especificar su alcance. Es decir, cuáles son privadas y cuáles son públicas. Todo lo público puede ser accesible desde fuera del modelo. Lo que describimos como características y funciones es la interfaz. Esta interfaz es el medio a través del cual podemos comunicarnos con el modelo. Decimos que es un contrato porque, al querer comunicarnos con este modelo, debemos utilizar su interfaz. Por lo tanto, si modificamos las interfaces sin avisar al resto, nos enfrentaremos a un error.

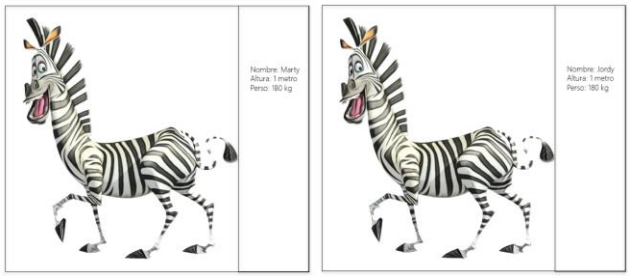
Definir un objeto

Un objeto es algo que representa una idea o un concepto, y que además tiene identidad única. También, reforzando, podemos decir que es una instancia de una clase

Todos los objetos, tienen una identidad que los distingue de otros. Pero, además tienen una estructura (estado) que los define y acciones (comportamiento) que los controlan. a continuación, describimos estos conceptos.

Estado

Es el conjunto de todas las propiedades y los valores para cada una de estas propiedades. En general todos los objetos tienen las mismas propiedades, pero cambian sus valores. Esto tiene sentido porque todos los objetos son instancias de una clase, Pero, pero no son el mismo objeto. en este camino, podemos decir que dos personas salen del mismo molde y ambas tienen un nombre, pero una se llama Luana y el otro Martín.



Comportamiento

Es todo lo que el objeto puede hacer, es decir sus acciones. no hay mucho más por decir…

Figure 2 Caminar

Figure 2 Saltar

Identidad

Cada instancia de un Objeto es única, por lo que debe existir una manera, para que estos, se puedan diferenciar. Sin ánimo de complejizar este concepto, aclaramos que los valores que toman las propiedades no es la manera de diferenciar un objeto de otro – y lo dejamos ahí-

Relaciones entre clases

Es la manera como los elementos pueden colaborar entre sí. Para simplificar, solo presentaremos dos tipos de relaciones, la herencia y la asociación.

Herencia

Herencia: una clase nueva se crea a partir de una existente.

Mediante la herencia, las instancias de una clase hija (o subclase) pueden acceder tanto a los atributos como a los métodos públicos y protegidos de la clase padre (o superclase). Se suele hacer referencia a la relación como "es una".

Asociación

Se establece cuando dos clases tienen una dependencia de utilización, es decir una clase utiliza atributos y/o métodos de otra para funcionar. Estas dos clases no necesariamente están en jerarquía, es decir, no necesariamente una clase es padre de la otra. A diferencia de las otras relaciones de clases, la asociación expresa un tipo de colaboración entre ambos elementos. Esto último, también se lo suele llamar **"una relación de uso".**

Clases y objetos en Python

Crear una clase

Como ocurre con las variables, debemos definirlas. La palabra reservada que se usa en **Python** es **class**

```
class Nave:
    pass
```

(**pass** es una palabra reservada que usamos para indicar que, por ahora, la clase está vacía)

Instancia de la clase

Crear un objeto

Se llama instancia, y el código es el siguiente:

```
nave1=Nave()
```

Atributos de una clase

Son las propiedades, es decir la estructura que lo define. Es un par de nombre y valor.

```
class Nave:
    vidas=3
    cantida_balas=100
    __velocidad=5 #privado
```

El alcance del atributo determina si se puede ver desde el exterior. En el caso de **Python** no podemos definir el alcance como privado, pero podemos emular algo similar al anteponer un doble guion bajo.

Para utilizar los atributos, solo navegamos su interfaz:

```
nave1=nave() #instancia
print (f"Cantidad de balas
{nave1.cantida_balas}")
```

Métodos de una clase

Se programan como si fuesen funciones, con el agregado que siempre debe tener por lo menos el parámetro **self**.

```
class Nave:
    vidas=3
    cantida_balas=100
    __velocidad=5 #privado
    def disparar(self):
        self.vidas -=1
```

Método constructor y método destructor

Son métodos especiales que se ejecutar en primer lugar cuando se instancia el objeto, y en último lugar cuando se destruye el objeto.

#Constructor y Destructor

```
def __init__(self):
    #código para inicializar
    pass

def __del__(self):
    #código para eliminar
    pass
```

Bibliografía

Sweigart, A. (2012). Making Games with Python & Pygame: A Guide to Programming with Graphics, Animation, and Sound. Recuperado de https://inventwithpython.com/makinggames.pdf

Pygame. (s.f.). [Página oficial de pygame]. Recuperado 2 julio, 2019, de https://www.pygame.org/news