VIDEOJUEGOS
INTRODUCCIÓN A PYTHON

CLASE 7

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.

”

PRORAMACIÓN I

Uno de los primeros avances que se tiene en programación es cuando comenzamos a trabajar con funciones, el cambio de nuestro código es enorme. Hace un uso correcto de las funciones, nos permite desarrollar códigos mas legibles y mas mantenibles.

En este apartado, vamos a extender nuestro conocimiento en el manejo de funciones. Incorporamos el uso de módulos y librerías que son muy útiles para asegurar la mantenibilidad del código. Por otro lado, trataremos ligeramente el sistema de Python para gestionar los errores del código en tiempo de ejecución.

Introducción

Terminando ya con esta primera aproximación de Python y la programación estructurada, hablaremos sobre los módulos y como estos pueden convertirse en nuestras librerías de trabajo. Cuando desarrollamos, estos es muy útil puesto que nos permite crear nuestras primeras herramientas para futuros trabajos. También, vamos a debatir sobre la importancia de controlar los datos de entrada de nuestros juegos. Por último, un tema de regalo. Veremos como persistir la información que nos resulte útil en el juego en archivos de texto.

Paradigmas

Un paradigma de programación es un estilo de programación que adoptamos para nuestras soluciones. No es otra cosa que un conjunto de métodos sistémicos que se aplican en los distintos niveles en el diseño de nuestros programas.

Cada lenguaje se pensó para un paradigma en particular. Algunos solo abonan a un paradigma, y otros lenguajes a mas de uno de ellos.

Estructurado

El flujo del programa se define por medio de estructuras, módulos o bucles. Algo que fue muy representativo en este paradigma es que se dejó de utilizar el GOTO. Lenguajes como C o pascal son los clásicos en este tipo de paradigma. Claramente Python también me permite aprovechar las directivas del paradigma de programación estructurado.

Orientado a objetos

En este caso los objetos con los referentes para organizar nuestro código. Tenemos que representar los elementos que observamos, y nuestra función es diseñar alguna abstracción. La programación orientada objetos mejorar la legibilidad y nos permite disminuir la cantidad de errores. Además, nos permite reutilizar el código con mas facilidad. Algunos ejemplos de lenguajes que nos permiten trabajar con este paradigma son C++, C# o Python.

Algunos puntos que se van a abordar en este paradigma son:

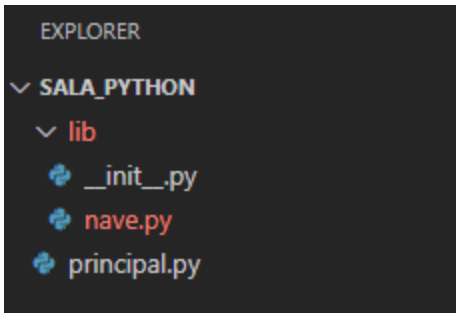
- Abstracción de datos
- Encapsulación
- Herencia
- Polimorfismo

Ingenuo paralelismo de módulos a clases y objetos

Aunque no es el fin, porque son cosas totalmente distintas. Vamos a intentar comparar lo que venimos usando en modo estructurado, para compararlo con este nuevo concepto llamado orientación a objetos (OO)

En PE, ordenamos las funciones en módulos o librerías según su afinidad. Por ejemplo, podemos tener una librería llamada naves. Declaramos las variables que representan algunos valores y las funciones con las cosas que la nave puede hacer. El código podría ser algo parecido 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 muchas naves que se compartan de la misma manera?

Bien, acá entra la programación orientada a Objetos. Programamos una clase para la nave:

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

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

Luego creamos una instancia (Objeto) de la clase nave, para cada nave que necesitamos:

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

Pilares de la programación orientada a Objetos

Este paradigma le permite al equipo de desarrollo cumplir con la misión de producir una ilusión de simplicidad.

Abstracción

La principal abstracción de este paradigma son los objetos, que representan distintos tipos de entidades.

Encapsulación

En POO, el código queda escondido dentro de los objetos. Por lo que decimos, que el código se encuentra encapsulado dentro del objeto. Esto significa, que se puede interactuar con el objeto, pero no sabemos como resuelve nuestra petición.

Instancia

Venimos hablando de observar el mundo real, para diseñar una abstracción de ese elemento observable. Luego de esto, y saltando muchos pasos en la teoría de OO, obtenemos un molde (clase). Cuando tenemos la clase, podemos crear todos los objetos que necesitemos. Esa acción 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. Las clases serán únicas, no se repiten. Por otro lado, los objetos si se repiten. Partiendo de las clases, podemos crear todos los objetos que necesitemos.

Definir una clase

Las clases es una descripción de un conjunto de elementos similares. La descripción consta de expresar las características comunes que observamos en ese conjunto, mas las acciones que esos elementos puede implementar.

¿Qué pasa si observamos este grupo de animales?

Notamos que son muy parecidos, todos negros y tiene rayas blancas (*menos Marty que es blanco con rayas negras* 😊). Notamos que todos pertenecen a una misma familia de animales, lo que nos acerca a la conclusión que podemos describir a cada uno de la misma manera. Por ejemplo:

- 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 a estos. Para describirlos detalles sus propiedades o atributos y sus acciones:

Cebra

Atributos o propiedades

- Color
- Cantidad de patas
- Altura
- Peso

Acciones o métodos

- Correr
- Saltar
- Comer

Interfaz o contrato

Al describir las propiedades y métodos de la clase, podemos indicar su alcance. Es decir, cuales son privados y cuales son públicos. Todo lo público se puede ver desde fuera del modelo. Esto que describimos como propiedades y métodos es la interfaz. Esta interfaz, es el medio por el cual nos podemos comunicar con el modelo. Por decimos que es un contrato, porque cuando queremos comunicarnos con este modelo, debemos usar su interfaz. Entonces, si cambiamos las interfaces y no avisamos al resto nos enfrentaríamos a un error.

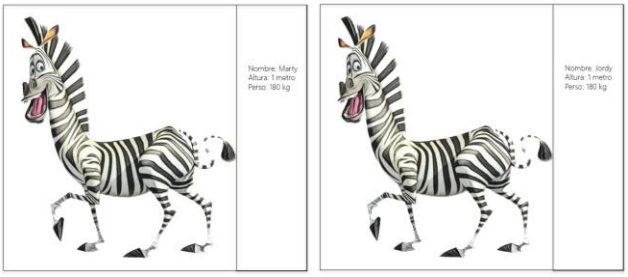
Definir un objeto

Un objeto es algo que representa una idea o un concepto, y que tiene identidad. También podemos decir que es una instancia de una clase, podemos tener todos los objetos que necesitemos

Todos los objetos, una identidad que los distingue del resto de los objetos. Pero, además tienen una estructura (estado) que los define y acciones (comportamiento) que los controlan.

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. Esto tiene sentido, porque todos los objetos son instancias de una clase. Pero, los valores que tienen las propiedades pueden ser diferentes.



Comportamiento

Es todo lo que el elemento puede hacer, es decir sus acciones.

Figure 2 Caminar

Figure 2 Saltar

Identidad

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

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. Suele identificarse como 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, 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 tenemos alcance privada, 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 ultimo 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