Agarrando la Pitón (Ejercicio final)

Lluc Galceran Jodar
21/05/2023

Índice:

¿Por qué Python?:

Sintaxis sencilla:
Python se destaca por su sintaxis fácil de leer y comprender. Utiliza una estructura clara y utiliza espacios en blanco (indentación) para definir bloques de código, en lugar de utilizar llaves o palabras clave especiales.

Lenguaje interpretado:
Python es un lenguaje interpretado, lo que significa que no se necesita compilar el código antes de ejecutarlo. Esto hace que el desarrollo en Python sea más rápido y eficiente.

Tipado dinámico:
En Python, no es necesario declarar explícitamente el tipo de una variable. El tipo de una variable se infiere automáticamente en tiempo de ejecución. Esto facilita la escritura de código rápido y flexible.

Amplia biblioteca estándar:
Python cuenta con una amplia biblioteca estándar que proporciona una gran cantidad de módulos y funciones listos para usar. Esto permite realizar tareas comunes de manera eficiente sin tener que escribir mucho código adicional.

Enfoque en legibilidad del código:
La filosofía de diseño de Python se centra en la legibilidad del código, lo que hace que sea más fácil de entender y mantener. Es decir no se utilizan corchetes “{}” eso al principio es confuso, pero el código Python tiende a ser más claro y conciso en comparación con otros lenguajes.

Multiplataforma:
Python es compatible con varias plataformas, lo que significa que puedes escribir un programa en Python y ejecutarlo en diferentes sistemas operativos sin necesidad de hacer cambios significativos en el código.

Orientación a objetos:
Python es un lenguaje orientado a objetos, lo que significa que permite la programación orientada a objetos y el uso de conceptos como clases, objetos, herencia y polimorfismo.

Pequeño trozo de código básico:

```python
#!/usr/bin/python3


import random

numero_adivinar = random.randrange(100)
#print(numero_adivinar)

archivo = open("intro")

print(archivo.read())

teclado = input("Introduce tu numero: ")

while numero_adivinar != teclado:

    if int(teclado) > numero_adivinar:
        print("El numero que quieres adivinar es mas pequeño ")

        teclado = input("Introduce tu numero: ")

    elif int(teclado) < numero_adivinar:
        print("El numero que quieres adivinar es mas grande ")

        teclado = input("Introduce tu numero: ")

    else:
        print("Lo adivinaste papurri!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!")
        break

```

Funciones i Condiciones en Python:

Condiciones:

La condición es una expresión que se evalúa como verdadera o falsa. Si la condición es verdadera, se ejecuta el bloque de código indentado bajo el "if". Si la condición es falsa, se ejecuta el bloque de código indentado bajo el "else" además podemos añadir más opciones con el “elif”, cuando terminemos de escribir la condición siempre se ponen dos puntos al final.

Pequeño ejemplo:

```
if int(teclado) > numero_adivinar:
    print("El numero que quieres adivinar es mas pequeño ")

    teclado = input("Introduce tu numero: ")

elif int(teclado) < numero_adivinar:
    print("El numero que quieres adivinar es mas grande ")

    teclado = input("Introduce tu numero: ")

else:
    print("Lo adivinaste papurri!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!")
    break
```

Funciones:

La declaración comienza con la palabra clave "def", seguida del nombre de la función, que debe seguir las convenciones de nomenclatura de Python. Entre paréntesis, se pueden especificar los parámetros que la función espera recibir (opcionalmente). Estos parámetros son variables que se utilizan dentro de la función.

```
def obten_datos ():
    name = input("Nombre: ")
    health = int(input("Vida: "))
    damage = int(input("Fuerza: "))

    return {
            "name": name,
            "damage": damage,
            "health": health
    }
```

El bloque de código de la función está indentado y puede contener cualquier número de declaraciones, como operaciones, condiciones, bucles, etc. Dentro de la función, también puedes utilizar variables locales que solo son visibles dentro de la función.

```
return {
        "name": name,
        "damage": damage,
        "health": health
}
```

Opcionalmente, la función puede utilizar la palabra clave "return" para devolver un resultado. Esta declaración finaliza la ejecución de la función y devuelve el valor especificado.

Juego básico con condiciones y xmltodict.

```python
enemy_files= ["enemy1.xml", "enemy2.xml", "enemy3.xml"]
random_file= random.choice(enemy_files)

xml_file_enemy= open(random_file, "r")

diccionario = xmltodict.parse(xml_file_enemy.read())

enemy1= diccionario["enemys"]["enemy"]
("Nombre: "+enemy1["name"])
("Daño: "+enemy1["damage"])
("Vida: "+enemy1["health"])
("Nivel: "+enemy1["level"])


vida= int(30)

vida_enemy1= int((enemy1["health"]))
daño_enemy= int((enemy1["damage"]))
nombre_enemy1= str((enemy1["name"]))

while True:

    teclado = input ("Introduce que quieres hacer, para ver tu vida escribe: vida, para atacar escirbe: ataca, para ver las stats de los enemigos escribe: stats y para salir escribe: salir\n")

    if teclado == "vida":

        print ("Tu vida es: " + str(vida))

    elif teclado == "ataca":

        daño= int(random.randrange(5))
        daño_enemy1= int(random.randrange(daño_enemy))

        vida_enemy1= vida_enemy1 - daño
        vida= vida - daño_enemy1
        print ("Le has sacado " + str(daño) + " de vida, ahora el enemigo tiene: "+ str(vida_enemy1) + " de vida\n")
        print (str(nombre_enemy1) + " tambien te ha atacado, te ha sacado "+ str(daño_enemy1) + " de vida, ahora tienes " + str(vida) + " de vida\n")

    elif teclado == "stats":

        enemy1= diccionario["enemys"]["enemy"]

        print("Nombre: "+enemy1["name"])

        print("Daño: "+enemy1["damage"])

        print("Vida: "+enemy1["health"])

        print("Nivel: "+enemy1["level"])

    elif teclado == "salir":
        break

    if vida <= 0:
        print("Mala suerte papu")
        break

    elif vida_enemy1 <= 0:
        print("Acabaste con el")
        break
```

Parse:

Aquí utilizamos una librería que se llama xmltodict, esta sirve para convertir archivos de XML a diccionarios de python, la función principal en xmltodict es **xmltodict.parse()**, que se utiliza para convertir XML en un diccionario, ejemplo de código:

```python
xml_file_enemy= open(random_file, "r")

diccionario = xmltodict.parse(xml_file_enemy.read())

enemy1= diccionario["enemys"]["enemy"]
("Nombre: "+enemy1["name"])
("Daño: "+enemy1["damage"])
("Vida: "+enemy1["health"])
("Nivel: "+enemy1["level"])
```

Unparse:

Otra función es la de unparse, lo que hace es del revés convertir el diccionario en un archivo **XML**, ejemplo de código:

```
def save_game_xml(self, filename):
    data = {
        'current_enemy': self.current_enemy,
        'enemy_health': self.enemies[self.current_enemy].health,
        'player_health': self.player.health,
        'player_experience': self.player.experience,
        'player_level': self.player.level,
    }

    xml_data = xmltodict.unparse({'game': data}, pretty=True)
    with open(filename, 'w') as xml_file:
        xml_file.write(xml_data)
    print("Partida guardada en XML correctamente.")
```

P.O.O. (Programación Orientada a Objetos)

La programación orientada a objetos en Python se basa en la creación de clases y objetos. Una clase es como una plantilla que define las propiedades y comportamientos que tendrán los objetos. Un objeto, por otro lado, es una instancia de una clase. Ejemplo de una clase:

enti@lluc (192.168.1.199) - byobu

```
1 class Perro:
2     def __init__(self, nombre, edad, raza):
3         self.nombre = nombre
4         self.edad = edad
5         self.raza = raza
6
7     def ladrar(self):
8         print(¡Guau!)
9
10    def saludar(self):
11        print("Hola, soy "+str(self.nombre)+" tengo " +str(self.edad)+" años. ¡Soy un" +str(self.raza))
12
13 # Crear una instancia de la clase Perro
14 mi_perro = Perro(Fido, 3, Labrador)
15
16 # Acceder a los atributos del perro
17 print(mi_perro.nombre)
18 print(mi_perro.edad)
19 print(mi_perro.raza)
20
21 # Llamar a los métodos del perro
22 mi_perro.ladrar()
23 mi_perro.saludar()
```

En este ejemplo, hemos definido la clase **"Perro"** con tres atributos: **"nombre", "edad"** y **"raza"**. También hemos definido dos métodos, que es como se llaman a las funciones que están dentro de una clase: **"ladrar"** y **"saludar"**. El método "init" es un método especial en Python que se llama automáticamente al crear un nuevo objeto y se utiliza para inicializar los atributos.

Luego, hemos creado una instancia de la clase "Perro" llamada **"mi_perro"** y hemos accedido a sus atributos y llamado a sus métodos utilizando la notación de punto.