Programación Funcional en Python

6.1: La programación funcional es un paradigma de programación que se centra en el uso de funciones como bloques de construcción fundamentales y evita el uso de variables y estados mutables. Se basa en el concepto de funciones puras, que no tienen efectos secundarios y siempre producen el mismo resultado para los mismos argumentos.

6.2: Python proporciona varias herramientas para la programación funcional, como funciones de orden superior (funciones que pueden aceptar otras funciones como argumentos o devolver funciones como resultado), expresiones lambda (funciones anónimas), y los módulos functools y itertools que ofrecen funciones útiles para operaciones funcionales.

6.3: La función map() en Python permite aplicar una función a cada elemento de una secuencia y devuelve un nuevo iterable con los resultados. Facilita la transformación de los elementos de una secuencia sin la necesidad de utilizar bucles explícitos.

6.4: La función filter() en Python permite filtrar los elementos de una secuencia en función de una condición especificada en una función. Devuelve un nuevo iterable que contiene solo los elementos que cumplen con la condición. En programación estructurada, se necesitaría utilizar un bucle para recorrer la secuencia y agregar manualmente los elementos que cumplan con la condición a una nueva lista.

6.5: La función reduce() en Python se utiliza para aplicar una operación de reducción a una secuencia de elementos. Toma una función y una secuencia, y acumula los elementos de la secuencia utilizando la función proporcionada. Devuelve un único resultado que es el resultado de la operación de reducción. Por ejemplo, se puede usar para sumar todos los elementos de una lista o encontrar el máximo de una lista.

6.6: El código de ejemplo muestra cómo utilizar la programación funcional en Python. Importa la función reduce del módulo functools. Luego, se define una lista de diccionarios llamada personas que representa información sobre personas. Luego, se filtran los hombres utilizando la función filter() y una expresión lambda que verifica el valor de la clave 'Sexo'. Luego, se utiliza reduce() y una expresión lambda para sumar las edades de los hombres. Finalmente, se calcula la media de edad dividiendo la suma de las edades entre la cantidad de hombres y se imprime el resultado.

6.7: En comparación con C++, Python proporciona una sintaxis más concisa y expresiva para la programación funcional. El uso de funciones de orden superior, expresiones lambda y las funciones map(), filter() y reduce() facilitan la escritura de código funcional en Python. Además, Python es un lenguaje interpretado y no requiere una compilación previa, lo que hace que el desarrollo en Python sea más rápido y ágil en comparación con C++.

Excepciones en Python

7.1: Una excepción es un evento o condición anormal que ocurre durante la ejecución de un programa y interrumpe su flujo normal. Puede ser causada por errores en el código, condiciones inesperadas o situaciones excepcionales. Lanzar una excepción significa generar y señalar una situación de error o anormalidad en el código. Capturar una excepción significa manejar y responder adecuadamente a esa situación de error en el código.

7.2: Algunos ejemplos de excepciones comunes en Python son:

TypeError: Ocurre cuando se realiza una operación en un tipo de dato inapropiado.

KeyError: Ocurre al intentar acceder a una clave que no existe en un diccionario.

IndexError: Ocurre cuando se intenta acceder a un índice fuera del rango válido en una secuencia.

NameError: Ocurre cuando se utiliza una variable o nombre que no está definido en el ámbito actual.

RuntimeError: Ocurre cuando se produce un error en tiempo de ejecución que no se ajusta a ninguna otra categoría de excepción.

ZeroDivisionError: Ocurre cuando se intenta dividir un número entre cero.

7.3: La palabra clave raise se utiliza para generar manualmente una excepción en Python. Permite al programador lanzar una excepción específica en un lugar determinado del código. Al utilizar raise, se puede interrumpir el flujo normal del programa y señalar una situación de error o anormalidad según sea necesario. Esto permite personalizar y controlar el manejo de excepciones en el código.

8.1: La función open() en Python se utiliza para abrir un archivo y obtener un objeto de archivo que se puede utilizar para realizar operaciones de lectura o escritura en el archivo. La función close() se utiliza para cerrar el archivo después de haber terminado de trabajar con él, liberando los recursos asociados al archivo.

8.2: Los diferentes "modos" de apertura en Python indican cómo se abrirá el archivo y qué tipo de operaciones se permitirán. Algunos modos comunes son:

"r": Modo de lectura. El archivo se abre para lectura y debe existir previamente.

"w": Modo de escritura. El archivo se abre para escritura, se crea si no existe o se sobrescribe si ya existe.

"a": Modo de anexado. El archivo se abre para añadir contenido al final, se crea si no existe.

"x": Modo de creación exclusiva. El archivo se abre para escritura solo si no existe, de lo contrario, se genera una excepción.

"b": Modo binario. El archivo se abre en modo binario para lectura o escritura de datos binarios.

"t": Modo de texto. El archivo se abre en modo de texto para lectura o escritura de datos de texto.

8.3:

La función seek() se utiliza para mover el "puntero de archivo" a una posición específica dentro del archivo. Permite controlar la posición desde la cual se realizarán las operaciones de lectura o escritura posteriores.

La función readline() se utiliza para leer una línea completa del archivo desde la posición actual del puntero de archivo. Retorna una cadena que contiene la línea leída.

La función readlines() se utiliza para leer todas las líneas del archivo y retornar una lista donde cada elemento es una línea del archivo. Cada línea incluye los caracteres de nueva línea al final (a menos que se use strip() para eliminarlos).

# Ejemplo de uso de seek()

archivo = open("mi\_archivo.txt", "r")

archivo.seek(10) # Mueve el puntero a la posición 10

datos = archivo.read() # Lee desde la posición 10 hasta el final del archivo

archivo.close()

# Ejemplo de uso de readline()

archivo = open("mi\_archivo.txt", "r")

linea = archivo.readline() # Lee la primera línea del archivo

print(linea)

archivo.close()

# Ejemplo de uso de readlines()

archivo = open("mi\_archivo.txt", "r")

lineas = archivo.readlines() # Lee todas las líneas del archivo

for linea in lineas:

print(linea)

archivo.close()

Estos ejemplos muestran cómo usar seek() para mover el puntero a una posición específica, readline() para leer una línea y readlines() para leer todas las líneas del archivo. Recuerda cerrar el archivo utilizando close() una vez que hayas terminado de trabajar con él para liberar los recursos

Módulos y Paquetes.

9.1: Los módulos en Python son archivos que contienen código Python reutilizable. Proporcionan una forma de organizar y reutilizar el código en diferentes programas. Los paquetes, por otro lado, son directorios que contienen múltiples módulos relacionados entre sí. Los paquetes permiten una estructura de organización jerárquica para los módulos y facilitan la gestión y el acceso a un conjunto de funcionalidades relacionadas.

9.2: La función sys.path() en Python es una lista que contiene los directorios donde el intérprete de Python busca los módulos cuando se realiza una importación. Proporciona información sobre los directorios de búsqueda predeterminados y también permite agregar nuevos directorios a la lista en tiempo de ejecución.

9.3: El uso de la palabra clave "as" en Python permite cambiar o poner alias a los paquetes o módulos importados. Esto es útil cuando se desea utilizar un nombre más corto o más descriptivo para el paquete o módulo en el código. También puede ayudar a evitar conflictos de nombres si se importan múltiples paquetes con nombres similares.

9.4: Python maneja las excepciones al importar módulos utilizando el mecanismo de manejo de excepciones incorporado. Si se produce un error durante la importación de un módulo, Python genera una excepción ImportError. Esto puede ocurrir si el módulo no se encuentra en el sistema o si hay un problema en el código del módulo. El programador puede utilizar bloques try-except para capturar y manejar estas excepciones de importación según sea necesario.

9.5: Python maneja la inclusión múltiple de módulos utilizando un sistema de caché interno. Cuando se importa un módulo por primera vez, Python ejecuta el código del módulo y almacena el resultado en la caché de módulos importados. En las importaciones posteriores del mismo módulo, Python simplemente devuelve el objeto de módulo almacenado en la caché en lugar de ejecutar el código nuevamente. Esto evita la ejecución repetida y garantiza que cada módulo se importe solo una vez durante una sesión de ejecución.

Testing en Python.

10.1: Testing se refiere al proceso de evaluación de un sistema o componente para verificar su funcionamiento y calidad. Consiste en ejecutar el software con casos de prueba específicos para identificar errores, comportamientos inesperados o discrepancias entre los resultados esperados y los resultados reales.

10.2: La diferencia entre el test manual y automático radica en la forma en que se ejecutan los casos de prueba. En el test manual, un tester humano realiza las pruebas de forma manual, ejecutando pasos específicos y verificando los resultados. En el test automático, las pruebas se realizan utilizando scripts o herramientas automatizadas que ejecutan los casos de prueba y comparan los resultados esperados con los resultados obtenidos.

10.3: Un "assert" en los tests automáticos es una afirmación o expresión que verifica una condición esperada. Se utiliza para verificar que el resultado obtenido en una prueba coincide con el resultado esperado. Si la condición no se cumple, se genera una excepción que indica una falla en la prueba.

10.4: unittest es un módulo en Python que proporciona un framework para escribir y ejecutar pruebas unitarias. Permite definir casos de prueba como clases que heredan de la clase unittest.TestCase, y dentro de cada caso de prueba se definen métodos que representan las pruebas individuales. El framework proporciona métodos y aserciones útiles para realizar comprobaciones y validar los resultados esperados.

10.5: El código dado muestra un ejemplo de implementación de pruebas unitarias utilizando el módulo unittest en Python. El archivo "funciones.py" contiene una función llamada calcula\_media que calcula la media de una lista de números. El archivo "tests.py" importa la función calcula\_media y define una clase TestCalculaMedia que hereda de unittest.TestCase. Dentro de esta clase, se definen dos métodos de prueba (test\_1 y test\_2) que utilizan el método self.assertEqual() para verificar que el resultado obtenido de la función calcula\_media coincide con el resultado esperado. Al final del código, se utiliza unittest.main() para ejecutar las pruebas.

En resumen, el código realiza pruebas unitarias para verificar el correcto funcionamiento de la función calcula\_media. Cada prueba verifica un escenario diferente y utiliza el método assertEqual() para comparar el resultado esperado con el resultado obtenido.

.assertEqual(a, b): Verifica la igualdad de ambos valores.

.assertTrue(x): Verifica que el valor es True.

.assertFalse(x): Verifica que el valor es False.

.assertIs(a, b): Verifica que ambas variables son la misma (ver operador is).

.assertIsNone(x): Verifica que el valor es None.

.assertIn(a, b): Verifica que a pertenece al iterable b (ver operador in).

.assertIsInstance(a, b): Verifica que a es una instancia de b

.assertRaises(x): Verifica que se lanza una excepción.

Usando setUp y tearDown en unittest, se pueden realizar tareas de configuración y limpieza antes y después de la ejecución de cada método de prueba en una clase de pruebas. Aquí está lo que permiten:

setUp:

Permite realizar cualquier configuración necesaria antes de ejecutar cada método de prueba.

Se ejecuta antes de cada método de prueba en la clase de pruebas.

Puede utilizarse para inicializar variables, establecer conexiones a bases de datos, cargar datos de prueba u otras tareas de preparación.

tearDown:

Permite realizar cualquier limpieza necesaria después de ejecutar cada método de prueba.

Se ejecuta después de cada método de prueba en la clase de pruebas.

Puede utilizarse para cerrar conexiones a bases de datos, eliminar archivos temporales o cualquier otra tarea de limpieza necesaria para dejar el entorno en un estado consistente.

En resumen, setUp y tearDown proporcionan un mecanismo para preparar y limpiar el entorno de pruebas antes y después de la ejecución de cada método de prueba. Esto ayuda a mantener la consistencia y el aislamiento entre las pruebas individuales, ya que se asegura de que el entorno esté en el estado adecuado antes de cada prueba y se realice la limpieza necesaria después de cada prueba.

El código proporcionado es un ejemplo básico de una clase de pruebas unitarias utilizando el módulo unittest en Python. Aquí está la explicación del código:

import unittest: Importa el módulo unittest, que proporciona las funcionalidades para escribir y ejecutar pruebas unitarias en Python.

class TestEjemplos(unittest.TestCase): Define una clase llamada TestEjemplos que hereda de unittest.TestCase. Esta clase representa una colección de métodos de prueba relacionados.

def setUp(self): Define el método setUp. Este método se ejecuta antes de cada método de prueba en la clase. En este ejemplo, simplemente imprime el mensaje "Entra setUp".

def tearDown(self): Define el método tearDown. Este método se ejecuta después de cada método de prueba en la clase. En este ejemplo, simplemente imprime el mensaje "Entra tearDown".

def test\_1(self): Define el método de prueba test\_1. Este método representa una prueba individual y se ejecutará como parte del conjunto de pruebas. En este ejemplo, simplemente imprime el mensaje "Test: test\_1".

def test\_2(self): Define el método de prueba test\_2. Este método también representa una prueba individual y se ejecutará como parte del conjunto de pruebas. En este ejemplo, simplemente imprime el mensaje "Test: test\_2".

En resumen, el código define una clase de pruebas unitarias llamada TestEjemplos que contiene dos métodos de prueba (test\_1 y test\_2). El método setUp se ejecutará antes de cada prueba, y el método tearDown se ejecutará después de cada prueba. Los métodos de prueba en este ejemplo simplemente imprimen mensajes para indicar su ejecución.