UD08 – Excepciones Python

Contenido

Introducción a las excepciones 2
¿Qué es una excepción? 2
¿Por qué necesitamos manejar excepciones? 2
2. Tipos de excepciones 2
3. Manejo de excepciones usando try/except 3
4. Uso de else y finally 4
5. Lanzar excepciones 5
6.- Buenas prácticas 6

Introducción a las excepciones

¿Qué es una excepción?

Una excepción en Python es un evento que ocurre durante la ejecución de un programa que interrumpe el flujo normal de las instrucciones del programa. En términos generales, cuando un error de Python ocurre, se genera una excepción que puede ser manejada para evitar que el programa se detenga completamente.

Las excepciones son eventos que se generan cuando ocurre un error en el código. Por ejemplo, si intentas dividir un número por cero, Python genera una excepción de tipo ZeroDivisionError. Otro ejemplo común es el FileNotFoundError, que se genera cuando intentas abrir un archivo que no existe.

Las excepciones proporcionan una forma de reaccionar a situaciones excepcionales (de ahí el nombre) en nuestro código, permitiéndonos manejarlas de manera especial sin tener que detener completamente la ejecución del programa.

¿Por qué necesitamos manejar excepciones?

Manejar excepciones es importante por varias razones:

1. **Control de flujo**: Las excepciones cambian el flujo de control de un programa. Cuando ocurre una excepción, el programa se detiene inmediatamente y el control se pasa al bloque de código más cercano que puede manejar la excepción (un bloque except, en el caso de Python). Esto permite a los programas manejar errores de manera elegante en lugar de fallar por completo.
2. **Depuración**: Las excepciones proporcionan información detallada sobre los errores que ocurren durante la ejecución de un programa, lo que puede ser muy útil para la depuración. La información proporcionada por una excepción a menudo incluye el tipo de error, una descripción del error y una traza de la pila que muestra dónde ocurrió el error.
3. **Robustez**: Manejar excepciones permite a los programas recuperarse de errores y continuar ejecutándose, lo que puede ser crucial en aplicaciones de producción. Por ejemplo, si estás escribiendo un servidor web, no quieres que todo el servidor se caiga solo porque una sola solicitud causó un error.
4. **Comunicación de errores**: Las excepciones permiten a los programas comunicar errores a otras partes del programa o a los usuarios. Por ejemplo, una función puede lanzar una excepción para indicar que no pudo realizar una tarea, permitiendo que el código que llamó a la función maneje el error de manera apropiada.

2. Tipos de excepciones

los tipos de excepciones más comunes en Python:

1. **Exception**: Es la clase base para todas las excepciones integradas en Python. Casi todas las excepciones que se lanzan en Python heredan de esta clase.

2. **AttributeError**: Se lanza cuando se intenta acceder a un atributo o método que no existe en un objeto.

```
x = 5
print(x.non_existent_attribute)  # Lanza AttributeError
```

3. **TypeError**: Se lanza cuando se realiza una operación o función con un tipo de objeto inapropiado.

```
"2" + 2  # Lanza TypeError
```

4. **ValueError**: Se lanza cuando una función recibe un argumento del tipo correcto pero con un valor inapropiado.

```
int("abc")  # Lanza ValueError
```

5. **IndexError**: Se lanza cuando se intenta acceder a un índice fuera del rango en una lista.

```
my_list = [1, 2, 3]
print(my_list[5])  # Lanza IndexError
```

6. **KeyError**: Se lanza cuando se intenta acceder a una clave que no existe en un diccionario.

```
my_dict = {"name": "John"}
print(my_dict["age"])  # Lanza KeyError
```

7. **FileNotFoundError**: Se lanza cuando se intenta abrir un archivo que no existe.

```
with open("non_existent_file.txt") as f:  # Lanza FileNotFoundError
    print(f.read())
```

8. **ZeroDivisionError**: Se lanza cuando se intenta dividir un número por cero.

```
print(1 / 0)  # Lanza ZeroDivisionError
```

Estos son solo algunos ejemplos de las excepciones integradas en Python. Puedes encontrar más información en la documentación oficial de Python sobre excepciones:

https://docs.python.org/es/3/library/exceptions.html

3. Manejo de excepciones usando try/except

El manejo de excepciones en Python se realiza mediante el uso de bloques try/except.

- **Bloque try:** Aquí es donde colocas el código que podría lanzar una excepción. Python intentará ejecutar este código y si se lanza una excepción, inmediatamente detendrá la ejecución del bloque try y pasará al bloque except.

- **Bloque except:** Este bloque se ejecuta si se lanza una excepción en el bloque try. Puedes tener varios bloques except para manejar diferentes tipos de excepciones.

Aquí tenemos un ejemplo básico:

```
try:
    # Intentamos dividir por cero, lo que lanza una excepción
    result = 1 / 0
except ZeroDivisionError:
    # Este bloque se ejecuta cuando se lanza una excepción
ZeroDivisionError
    print("¡Oops! No se puede dividir por cero.")
```

En este caso, como estamos intentando dividir por cero, se lanza una excepción ZeroDivisionError. Como tenemos un bloque except que maneja este tipo de excepción, ese bloque se ejecuta y se imprime el mensaje "¡Oops! No se puede dividir por cero.".

Puedes tener varios bloques except para manejar diferentes tipos de excepciones. Ejemplo:

```
try:
    # Código que podría lanzar varias excepciones
    x = "2"
    y = 2
    result = x / y
except ZeroDivisionError:
    print("¡Oops! No se puede dividir por cero.")
except TypeError:
    print("¡Oops! Los tipos de los operandos no son compatibles para
la división.")
```

En este caso, como estamos intentando dividir una cadena por un número, se lanza una excepción TypeError. Como tenemos un bloque except que maneja este tipo de excepción, ese bloque se ejecuta y se imprime el mensaje "¡Oops! Los tipos de los operandos no son compatibles para la división.".

4. Uso de else y finally

Además de try y except, los bloques de manejo de excepciones en Python pueden incluir las cláusulas else y finally.

- **Bloque else**: Este bloque se ejecuta si el código dentro del bloque try se ejecutó sin lanzar ninguna excepción. Es útil para código que debe ejecutarse si no se producen errores, pero que no debe ejecutarse si se produce una excepción. Por ejemplo:

```
try:
    # Intentamos convertir una cadena a un entero
    num = int("123")
except ValueError:
    # Este bloque se ejecuta si se lanza una excepción ValueError
    print("¡Oops! Eso no es un número válido.")
else:
```

```
    # Este bloque se ejecuta si no se lanzó ninguna excepción
    print("La conversión fue exitosa. El número es:", num)
```

- **Bloque finally**: Este bloque se ejecuta siempre, independientemente de si se lanzó una excepción o no. Es útil para código que debe ejecutarse sin importar qué suceda, como la limpieza de recursos. Por ejemplo:

```
try:
    # Intentamos abrir un archivo y leer su contenido
    f = open("myfile.txt")
    content = f.read()
except FileNotFoundError:
    # Este bloque se ejecuta si se lanza una excepción
FileNotFoundError
    print("¡Oops! El archivo no se encontró.")
else:
    # Este bloque se ejecuta si no se lanzó ninguna excepción
    print("El archivo se leyó exitosamente. Su contenido es:",
content)
finally:
    # Este bloque se ejecuta siempre, sin importar si se lanzó una
excepción o no
    print("Cerrando el archivo...")
    f.close()
```

En este último ejemplo, el bloque finally se asegura de que el archivo se cierre correctamente, independientemente de si se pudo leer con éxito o no.

5. Lanzar excepciones

En Python, puedes lanzar tus propias excepciones utilizando la palabra clave raise. Esto es útil cuando quieres indicar que algo ha ido mal en tu programa y quieres interrumpir la ejecución normal.

```
if not isinstance(x, int):
    raise TypeError("x debe ser un entero")
```

En este caso, si x no es un entero, lanzamos una excepción TypeError con un mensaje que explica el problema.

También puedes definir tus propias clases de excepciones para lanzar errores más específicos a tu aplicación. Para hacerlo, simplemente defines una nueva clase que hereda de la clase base Exception o de cualquier otra clase de excepción. Por ejemplo:

```
class MiExcepcionPersonalizada(Exception):
    pass

if algo_malo:
    raise MiExcepcionPersonalizada("Algo malo ocurrió")
```

En este caso, si algo_malo es verdadero, lanzamos una excepción MiExcepcionPersonalizada con un mensaje que explica el problema.

Lanzar tus propias excepciones puede ser útil para indicar problemas que son específicos a tu aplicación y que no están cubiertos por las excepciones integradas de Python. También te permite proporcionar mensajes de error más detallados y específicos para ayudar a depurar el problema.

6.- Buenas prácticas

1. **No manejar todas las excepciones**: Es posible capturar todas las excepciones con un bloque `except` sin especificar ningún tipo de excepción. Sin embargo, esto no es una buena práctica porque puede ocultar errores inesperados y hacer que el código sea más difícil de depurar. En su lugar, debes tratar de capturar solo las excepciones que esperas y sabes cómo manejar.

```
try:
    # Código que puede lanzar una excepción
except:  # No es una buena práctica
    pass
```

2. Evitar el uso de excepciones desnudas: Similar al punto anterior, debes evitar el uso de excepciones desnudas. Una excepción desnuda es cuando capturas una excepción, pero no haces nada con ella. Esto puede ocultar problemas en tu código.

```
try:
    # Código que puede lanzar una excepción
except Exception:  # Es una excepción desnuda si no haces nada con
ella
    pass
```

3. **Lanzar excepciones apropiadas**: Cuando lances tus propias excepciones, asegúrate de lanzar el tipo de excepción que mejor describa el problema que ocurrió. Python tiene muchas excepciones integradas que puedes usar, o puedes crear tus propias excepciones personalizadas. Esto hace que tu código sea más fácil de entender y depurar.

```
if not isinstance(x, int):
    raise TypeError("x debe ser un entero")  # Lanza una excepción
apropiada
```