Universidad De Oriente
Nucleo Anzoategui
Lenguajes de Programación
Sección #01

Exepciones en Python

Bachiller:
Valiant Molero
C.I.: 31.245.901

Profesor:
Manuel Carrasquero

1. TypeError

Descripción: `TypeError` se produce cuando se intenta realizar una operación en un tipo de dato inapropiado. Esto ocurre, por ejemplo, al intentar sumar una cadena de texto y un número entero, o al pasar un argumento de tipo incorrecto a una función.

Caso de uso común: Esta excepción es común al trabajar con entradas de usuario que no se han convertido al tipo correcto.

Ejemplo:

```
try:
    result = '5' + 5
except TypeError as e:
    print(f"Error: {e}")
```

Explicación: Aquí, al intentar sumar una cadena de texto (`'5'`) con un número entero (`5`), Python lanza un `TypeError` porque los tipos son incompatibles. Una solución es convertir explícitamente la cadena en un entero usando `int()`.

2. ZeroDivisionError

Descripción: La excepción `ZeroDivisionError` ocurre cuando se intenta dividir un número por cero. Matemáticamente, esta operación es indefinida y, por tanto, Python lanza un error.

Caso de uso común: Se presenta en cálculos matemáticos cuando el divisor es dinámico y podría llegar a ser cero.

Ejemplo:

```
try:
    result = 10 / 0
except ZeroDivisionError as e:
    print(f"Error: {e}")
```

Explicación: Aquí se intenta dividir `10` entre `0`, lo que genera un `ZeroDivisionError`. Una práctica común es verificar que el divisor no sea cero antes de realizar la operación de división.

3. OverflowError

Descripción: `OverflowError` ocurre cuando el resultado de una operación aritmética es demasiado grande para ser representado en el tipo de dato numérico de Python. Esto es raro en Python, ya que maneja automáticamente números grandes con tipos de datos de precisión extendida, pero puede ocurrir en cálculos matemáticos extremos.

Caso de uso común: A menudo se encuentra en cálculos científicos o financieros, como exponenciales grandes o factoriales de números grandes.

Ejemplo:

```
import math

try:
    result = math.exp(1000)
except OverflowError as e:
    print(f"Error: {e}")
```

Explicación: Aquí, `math.exp(1000)` intenta calcular un número extremadamente grande, provocando un `OverflowError`. En estos casos, se suele usar librerías como `decimal` o `fractions`, que pueden manejar números grandes con más precisión.

4. IndexError

Descripción: `IndexError` ocurre cuando se intenta acceder a un índice fuera del rango de una lista o secuencia. Las listas en Python tienen índices válidos que van de `0` a `len(lista) - 1`.

Caso de uso común: Aparece al manipular listas o arrays, especialmente cuando se trabaja con índices de manera dinámica o en bucles.

Ejemplo:

```
my_list = [1, 2, 3]

try:
    result = my_list[5]
except IndexError as e:
    print(f"Error: {e}")
```

Explicación: Aquí se intenta acceder al índice `5` en una lista con solo tres elementos (`0`, `1`, `2`). Para evitar este error, es común verificar si el índice está dentro del rango con `if index < len(my_list)`.

5. KeyError

Descripción: La excepción `KeyError` ocurre cuando se intenta acceder a un diccionario con una clave que no existe en él.

Caso de uso común: Es común al trabajar con diccionarios en los que no se tiene certeza de que todas las claves estén definidas.

Ejemplo:

```
my_dict = {'name': 'Alice', 'age': 25}

try:
    result = my_dict['address']
except KeyError as e:
    print(f"Error: {e}")
```

Explicación: Aquí, se intenta acceder a la clave `'address'`, que no existe en el diccionario `my_dict`. Para evitar el error, se puede usar el método `get()` del diccionario (`my_dict.get('address')`), que devuelve `None` si la clave no existe.

6. FileNotFoundError

Descripción: `FileNotFoundError` ocurre cuando se intenta abrir o manipular un archivo que no existe en la ruta especificada.

Caso de uso común: Común al intentar abrir archivos que el usuario puede haber movido, eliminado o renombrado.

Ejemplo:

```
try:
    with open('non_existent_file.txt', 'r') as file:
        content = file.read()
except FileNotFoundError as e:
    print(f"Error: {e}")
```

Explicación: Aquí se intenta abrir un archivo que no existe, lo que provoca un `FileNotFoundError`. Es buena práctica verificar la existencia del archivo antes de abrirlo usando `os.path.exists()`.

7. ImportError

Descripción: La excepción `ImportError` ocurre cuando Python no puede encontrar un módulo que se intenta importar, ya sea porque no existe, no está instalado o está mal escrito.

Caso de uso común: Ocurre al trabajar con dependencias externas o módulos opcionales que pueden no estar instalados en el entorno.

Ejemplo:

```
try:
    import non_existent_module
except ImportError as e:
    print(f"Error: {e}")
```

Explicación: Aquí se intenta importar un módulo que no existe. Para evitar este error, se puede verificar que el módulo esté instalado antes de importarlo o usar un bloque `try-except` para manejar la excepción y dar instrucciones al usuario.

8. ValueError

Descripción: `ValueError` ocurre cuando una función recibe un argumento con el tipo correcto, pero con un valor inaceptable.

Caso de uso común: Muy común al convertir datos de entrada de usuario o al trabajar con funciones que requieren valores específicos (como convertir cadenas a enteros).

Ejemplo:

```
try:
    result = int("abc")
except ValueError as e:
    print(f"Error: {e}")
```

Explicación: En este caso, se intenta convertir la cadena `"abc"` en un entero, lo cual no es posible y genera un `ValueError`. Una práctica útil es validar el valor antes de la conversión para asegurarse de que se pueda convertir.

9. AttributeError

Descripción: `AttributeError` ocurre cuando se intenta acceder o asignar un atributo que no existe en un objeto. Esto suele suceder cuando el nombre de un atributo o método está mal escrito o no está definido en la clase.

Caso de uso común: Frecuente en el desarrollo de clases y objetos en programación orientada a objetos, especialmente cuando se trabaja con dinámicas de herencia o polimorfismo.

Ejemplo:

```
class MyClass:
    def __init__(self):
        self.name = "Python"

obj = MyClass()

try:
    result = obj.age
except AttributeError as e:
    print(f"Error: {e}")
```

Explicación: Aquí se intenta acceder al atributo `age` que no existe en la clase `MyClass`. Esto provoca un `AttributeError`. Para evitarlo, se pueden definir todos los atributos necesarios en el método `__init__` o verificar con `hasattr(obj, 'age')` antes de acceder al atributo.