

Manejo de excepciones

(Data Engineer)



Introducción

Bienvenidos al módulo 9 - Manejo de excepciones en Python. En el mismo, los objetivos de aprendizaje estarán orientados a incorporar conocimientos generales acerca del manejo de excepciones.

Una vez finalizado este módulo estarás capacitado para las siguientes acciones:

- Utilizar las excepciones pre configuradas que nos ofrece Python.
- Comprender la diferencia entre un error de sintaxis y una excepción.
- Generar excepciones a voluntad.
- Manejar excepciones a través de diferentes estructuras de control.



Tema 1. Excepciones

Objetivos

Una vez finalizado este tema estarás capacitado para las siguientes acciones:

- Comprender que es una excepción y diferenciarla de un error de sintaxis.
- Conocer los diferentes tipos de excepciones pre configuradas que ofrece Python
- Generar una excepción mediante la sentencia **raise**.
- Utilizar **assert** para generar una afirmación y generar excepciones del tipo `AssertionError`.

Introducción

Un programa en Python termina tan pronto como encuentra un error. Este error puede ser un error de sintaxis o una excepción.

El manejo de excepciones en Python nos permite controlar el comportamiento de un programa cuando se produce un error.

Excepciones frente a errores de sintaxis

Los errores de sintaxis ocurren cuando el módulo “parser” de Python detecta una declaración incorrecta.

Cuando ejecutamos código de Python, el intérprete primero lo analizará para convertirlo en código de bytes de Python, que luego ejecutará. El intérprete encontrará cualquier sintaxis no válida en Python durante esta primera etapa de ejecución del programa, también conocida como **etapa de análisis**. Si el intérprete no puede analizar correctamente el código de Python, significa que utilizamos una sintaxis no válida en alguna parte del código. El intérprete intentará mostrarnos dónde ocurrió ese error.

Por ejemplo:

```
print(0/0))

Input In [5]
      print(0/0))
           ^
SyntaxError: unmatched ')'
```

La flecha indica dónde se encontró el analizador con el error de sintaxis . En este ejemplo, había un paréntesis de más. Si lo eliminamos y lo ejecutamos nuevamente:

```
print(0/0)

-----
ZeroDivisionError                                Traceback (most recent call last)
Input In [6], in <cell line: 1>()
----> 1 print(0/0)

ZeroDivisionError: division by zero
```

Esta vez, vemos que nos encontramos con un error de excepción . Este tipo de error ocurre siempre que el código Python sintácticamente correcto da como resultado un error. La última línea del mensaje indicaba qué tipo de error de excepción se encontró.

En lugar de mostrar el mensaje **exception error**, Python detalla qué tipo de error de excepción se encontró. En este caso, fue un **ZeroDivisionError**. Python viene con varias excepciones integradas, así como con la posibilidad de crear excepciones autodefinidas.

En la siguiente imagen podemos ver un resumen de ellas.

Class	Description
Exception	A base class for most error types
AttributeError	Raised by syntax obj.foo, if obj has no member named foo
EOFError	Raised if “end of file” reached for console or file input
IOError	Raised upon failure of I/O operation (e.g., opening file)
IndexError	Raised if index to sequence is out of bounds
KeyError	Raised if nonexistent key requested for set or dictionary
KeyboardInterrupt	Raised if user types ctrl-C while program is executing
NameError	Raised if nonexistent identifier used
StopIteration	Raised by next(iterator) if no element; see Section 1.8
TypeError	Raised when wrong type of parameter is sent to a function
ValueError	Raised when parameter has invalid value (e.g., sqrt(-5))
ZeroDivisionError	Raised when any division operator used with 0 as divisor

También podemos consultar el siguiente [link](#) a la documentación oficial para más detalles.

Por ejemplo, para la excepción encontrada anteriormente podemos encontrar la siguiente definición.

exception **ZeroDivisionError**

Raised when the second argument of a division or modulo operation is zero. The associated value is a string indicating the type of the operands and the operation.

The following exceptions are kept for compatibility with previous versions; starting from Python 3.3, they are aliases of `OSError`.

También podemos observar otro tipo de excepciones, en la imagen que se encuentra a continuación.

```
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Generación de una excepción

Anteriormente generamos una excepción al realizar una operación no permitida, como dividir por cero.

Python también nos permite lanzar o levantar una excepción utilizando la palabra reservada **raise**, deteniendo el control de flujo del programa.

Por ejemplo podemos levantar una excepción del tipo **IndexError**.

```
raise IndexError ('error de índice')
```

```
-----
IndexError                                Traceback (most recent call last)
Input In [11], in <cell line: 1>()
----> 1 raise IndexError ('error de índice')

IndexError: error de índice
```

También podemos levantar una excepción si se cumple una condición lógica.

```
a = 3
if (a == 3):
    raise Exception ('a no puede ser igual a 3')
```

```
Exception                                Traceback (most recent call last)
Input In [14], in <cell line: 2>()
      1 a = 3
      2 if (a == 3):
----> 3     raise Exception ('a no puede ser igual a 3')

Exception: a no puede ser igual a 3
```

Excepción AssertionError

La palabra reservada **Assert** nos permite hacer una afirmación, es decir algo que siempre es verdadero. Cuando esa afirmación toma un valor falso, entonces tendremos una excepción del tipo AssertionError.

Por ejemplo siempre que el sistema operativo sea win32, no se levantará la excepción.

```
import sys
sist_operativo = sys.platform
sist_operativo

'win32'

assert('win32' in sist_operativo)
```

Cuando la afirmación no sea verdadera se generará la excepción.

```
#forzamos la excepción
sist_operativo = 'linux'

assert('win32' in sist_operativo)
```

```
AssertionError                                Traceback (most recent call last)
Input In [24], in <cell line: 1>()
----> 1 assert('win32' in sist_operativo)

AssertionError:
```



Tema 2. Manejo de excepciones

Objetivos

Una vez finalizado este tema estarás capacitado para las siguientes acciones:

- Manejar excepciones utilizando los bloques try - except.
- Manejar excepciones utilizando las sentencias else y finally.

Introducción

Hasta ahora vimos que cuando se genera algún tipo de excepción, nuestro programa se detiene por completo. Afortunadamente, las excepciones pueden ser capturadas y manejadas de forma adecuada sin que el programa se detenga.

Bloque Try - Except

El bloque Try - Except se utiliza para capturar y manejar excepciones. Python ejecuta el código siguiendo la declaración **try** como una parte "normal" del programa. El código que sigue a la declaración **except** es la respuesta del programa a cualquier excepción en la declaración Try anterior.

En el siguiente ejemplo podemos observar cómo estamos manejando la excepción **ZeroDivisionError**

```
a = 10
b = 0

try:
    a/b
except ZeroDivisionError:
    print('No se puede dividir por cero')

No se puede dividir por cero
```

Siguiendo esa estructura podemos utilizar el bloque try - except para reformular el script anterior y adecuarlo a nuestras necesidades.

```
def get_numbers(msj):
    print(msj)
    a = float(input())
    b = float(input())

    division(a,b)

def division(a,b):
    try:
        res = a/b
        print('La división es:',res)
    except ZeroDivisionError:
        get_numbers('--->No se puede dividir por cero. Ingresá los números nuevamente')

get_numbers('Ingresá dos números')

Ingresá dos números
10
0
--->No se puede dividir por cero. Ingresá los números nuevamente
10
5
La división es: 2.0
```

En el ejemplo anterior capturamos la excepción ZeroDivisionError, pero puede ser que no conozcamos el tipo de excepción que puede suceder. En ese caso podemos usar **Exception**.

Por ejemplo si queremos realizar la siguiente operación tendremos una excepción del tipo **TypeError**.


```
d = 2 + "Hola"

-----
TypeError                                Traceback (most recent call last)
Input In [57], in <cell line: 1>()
----> 1 d = 2 + "Hola"

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

En el caso de no saber a priori que tipo de excepción vamos a manejar podemos utilizar:

```
try:
    d = 2 + "Hola"
except Exception:
    print('No se pudo realizar la operación')

No se pudo realizar la operación
```

Si en el manejo de la excepción queremos conocer de qué tipo de excepción se trata, podemos utilizar **type()**.

```
try:
    d = 2 + "Hola"
except Exception as exp:
    print('No se pudo realizar la operación debido a la excepción:', type(exp))

No se pudo realizar la operación debido a la excepción: <class 'TypeError'>
```

Uso de la sentencia else

Al bloque **try - except** podemos agregar la sentencia **else**, que se ejecutará si no ha ocurrido ninguna excepción.

```
a = 10
b = 2

try:
    a/b
except ZeroDivisionError:
    print('No se puede dividir por cero')
else:
    print('No ha ocurrido ninguna excepción')

No ha ocurrido ninguna excepción
```

Uso de la sentencia finally

Al bloque **try - except** también podemos agregarle la sentencia **finally**, que se ejecutará **haya o no** una excepción.

Por ejemplo, si generamos una excepción.

```
a = 10
b = 0

try:
    a/b
except ZeroDivisionError:
    print('No se puede dividir por cero')
else:
    print('No ha ocurrido ninguna excepción')
finally:
    print('Este mensaje se ejecuta siempre')

No se puede dividir por cero
Este mensaje se ejecuta siempre
```

En caso de que no genere la excepción, vemos que también se ejecuta el código que se agrega a la sentencia **finally**.

```
a = 10
b = 2

try:
    a/b
except ZeroDivisionError:
    print('No se puede dividir por cero')
else:
    print('No ha ocurrido ninguna excepción')
finally:
    print('Este mensaje se ejecuta siempre')
```

No ha ocurrido ninguna excepción
Este mensaje se ejecuta siempre

Ejemplo

En este ejemplo vamos a recapitular los conceptos vistos anteriormente. En primer lugar utilizaremos una función que afirma que el tipo de sistema operativo es Windows de 32 bits.

```
import sys
def windows_interaction():
    assert ('win32' in sys.platform), "Esta función funciona solamente en Windows"
    print('Sistema operativo Win32...')
```

A continuación generamos el siguiente bloque try - except, donde en caso de que:

- La función **windows_interaction** genere una excepción, nos imprimirá la excepción **AssertionError**.
- La plataforma sea Windows 32 bits, intentará abrir el archivo file.log, leerlo e imprimirlo por la consola.
- En caso de que exista una excepción del tipo **FileNotFoundError**, imprimirá el mensaje de esa excepción.

```
try:
    windows_interaction()

except AssertionError as error:
    print(error)

else:
    try:
        with open('file.log') as file:
            read_data = file.read()
            print(read_data)
    except FileNotFoundError as fnf_error:
        print(fnf_error)
```

Probamos el programa con las siguiente condiciones y visualizamos los resultados:

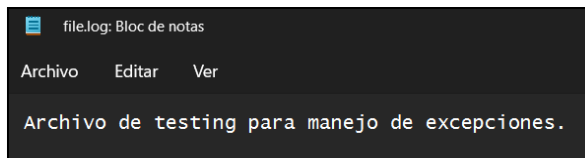
- Sistema operativo = Linux

Esta función funciona solamente en Windows

- Sistema operativo = win32 & ningún archivo file.log

Sistema operativo Win32...
[Errno 2] No such file or directory: 'file.log'

- Sistema operativo = win32 & archivo file.log



Sistema operativo Win32...
Archivo de testing para manejo de excepciones.



Cierre

Durante esta unidad aprendimos sobre excepciones.

Comenzamos diferenciando una excepción de un error, y conocimos acerca de las excepciones pre establecidas que nos ofrece Python.

Posteriormente aprendimos a generar excepciones y a manejarlas utilizando diferentes bloques de control, permitiendo la continuación del flujo de nuestros programas.

Quedamos a disposición de las consultas que puedan surgir.



Referencias

Errors and Exceptions | Python 3.10.17 documentation. Recuperado de: <https://docs.python.org/3/tutorial/errors.html>

Exception handling in Python try/except/else/finally | Dev Inline.
Recuperado de:
<https://www.devinline.com/2015/04/exception-handling-in-python.html>

Python Exceptions - An Introduction | Real Python. Recuperado de:
<https://realpython.com/python-exceptions/>

