UTN.BA | Centro de e-Learning
Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Diplomatura en Python:

Python nivel intermedio

UTN.BA | Centro de e-Learning
Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 1:

Nivel Intermedio II

Unidad III:

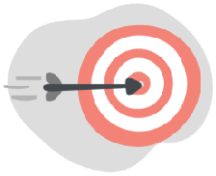
Programación y Excepciones.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Presentación

En esta unidad veremos cómo trabajar con excepciones, las cuales nos sirven como una señal de que se ha producido un error u otra condición inusual en nuestro código. Hay una serie de excepciones incorporadas en cada distribución de python, que indican condiciones cómo buscar valores fuera de un rango de índice dado, o dividir por cero, también es posible definir nuestras propias excepciones.
En Python, las excepciones se activan automáticamente en caso de errores, y podemos utilizarlas para interceptar los errores y proporcionar una respuesta en su lugar.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning



Objetivos

Comprendan cómo trabajar con excepciones.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Bloques temáticos

1.- Introducción.
2.- Declaración try.
3.- Raise.
4.- Try – finally.
5.- La cláusula assert.
6.- La cláusula raise y el uso de clases.
7.- Métodos en clases de excepciones.
8.- Propagación de excepciones.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

1.- Introducción.

En general utilizaremos las excepciones con varios fines, entre los que podemos citar el manejo de errores, la notificación de eventos, las acciones de cierre y algunos casos especiales. En el caso del manejo de errores, Python genera excepciones siempre que detecta errores en los programas en tiempo de ejecución, para nosotros, será posible detectar y generar una respuesta a los errores en nuestro código, o ignorar las excepciones que se generan. Si se ignora un error, se inicia el comportamiento predeterminado de manejo de excepciones, en el cual el programa se detiene y se imprime un mensaje de error. Si no deseamos que el programa se detenga deberemos capturar y recuperar la excepción.

Veamos un ejemplo en el cual creamos una función que está esperando recibir dos valores numéricos, si como en la línea 5 le pasamos dos enteros, vemos que nos retorna la suma de ambos valores, sin embargo si como segundo parámetro le pasamos un string, como en la línea 6 podemos ver que nos retorna un error con una descripción del tipo de error:

excepcion1.py

```
1  def sumar(a, b):
2      c = a + b
3      return c
4
5  print(sumar(3, 4))
6  print(sumar(3, 'Manzana'))
7  print(sumar(2, 3))
```

El código anterior retorna.

```
7
Traceback (most recent call last):
  File "C:\Users\juanb\Desktop\excepcion1.py", line 6, in <module>
    print(Sumar(3, 'Manzana'))
  File "C:\Users\juanb\Desktop\excepcion1.py", line 2, in Sumar
    c = a + b
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Como podemos ver la línea 7 no se ejecuta ya que como en la línea 6 fue encontrado un error, el programa se detuvo.

Para evitar la detención podemos atrapar el error de la siguiente forma:

excepcion2.py

```
1  def sumar(a, b):
2      c = a + b
3      return c
4
5  print(sumar(3, 4))
6
7  try:
8      print(sumar(3, 'Manzana'))
9  except:
10     print("Ha surgido un error")
11
12 print(sumar(2, 3))
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Retorna:

```
7
Ha surgido un error
5
```

La estructura try/except nos ha permitido que el programa intente (try) ejecutar la llamada a la función con un segundo parámetro y de existir un problema en la ejecución (except) se imprima un mensaje informando de que ha existido un error. Sin embargo a diferencia de lo que pasaba antes, podemos ver que la línea 12 se ha ejecutado, retornando el valor de "5".

2. - Declaración try

La declaración de prueba funciona de la siguiente manera.

- Primero, se ejecuta la cláusula try (la/s declaración/es entre las palabras clave try y except).
- Si no se produce una excepción, la cláusula de excepción se omite y la ejecución de la sentencia try finaliza.
- Si se produce una excepción durante la ejecución de la cláusula try, el resto de la cláusula se omite. Luego, si su tipo coincide con la excepción nombrada después de la palabra clave except, se ejecuta la cláusula except, y luego la ejecución continúa después de la instrucción try.
- Si se produce una excepción que no coincide con la excepción nombrada en la cláusula de excepción, se pasa a las declaraciones de prueba externas; si no se encuentra ningún controlador, es una excepción no controlada y la ejecución se detiene con un mensaje de error.

Una declaración de prueba puede tener más de una cláusula de excepción, para especificar manejadores para diferentes excepciones. A lo sumo se ejecutará un controlador. Los manejadores solo manejan las excepciones que ocurren en la cláusula try correspondiente, no en otros manejadores de la misma declaración try. Una cláusula de excepción puede nombrar múltiples excepciones como una tupla entre paréntesis, por ejemplo:

```
... except (RuntimeError, TypeError, NameError):
...     pass
```

3.- Raise

En algunos casos en los cuales sabemos que una condición puede causar un error, podemos lanzar una excepción antes de que el error ocurra y presentar un mensaje asociado. La estructura es:

```
raise TypeError("----------mensaje----------")
```

Veamos un ejemplo en el cual a partir de un número aleatorio intentamos calcular la raíz cuadrada:

excepcion3.py

```
1  import random
2
3  numero = random.randint(-5, 5)
4
5  try:
6      if numero < 0:
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

```
7         raise TypeError("No se puede calcular la raiz")
8       print("numero %.2f y su raíz cuadrada %.2f" % (numero, numero ** 0.5))
9   except (TypeError) as mensaje:
10      print("Ocurrió una excepción identificada.", mensaje)
```

Si el valor es un entero negativo se lanza una excepción y es atrapa en la línea 9 y presentado el mensaje de error en la línea 10:

```
Ocurrió una excepción identificada. No se puede calcular la raiz
```

En caso contrario, nos retorna el número y su valor de raíz cuadrada.

```
numero 5.00 y su raíz cuadrada 2.24
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

4.- Try - finally

La combinación **try / finally** especifica acciones de terminación que siempre se ejecutan "en el camino de salida", independientemente de si se produce una excepción en el bloque try o no, incluso podemos agregar un condicional “else”. Modifiquemos un poco el ejemplo anterior:

excepcion4.py

```
1  import random
2
3  error = True
4  numero = random.randint(-5, 5)
5
6  try:
7      if numero < 0:
8          raise TypeError("No se puede calcular la raíz")
9      print("numero %.2f y su raíz cuadrada %.2f" % (numero, numero ** 0.5))
10 except (TypeError) as mensaje:
11     print("Ocurrió una excepción identificada.", mensaje)
12 else:
13     print("No hubo errores.")
14     error = False
15 finally:
16     if error:
17         print("Se ha dado un error.")
```

```
18      else:
19          print("¡Se ha podido encontrar un resultado!")
```

Al ejecutarlos podemos ver dos tipos de salidas, en el caso de no existir error nos da:

```
número 3.00 y su raíz cuadrada 1.73
No hubo errores.
¡Se ha podido encontrar un resultado!
```

O si el valor del número es negativo:

```
Ocurrió una excepción identificada. No se puede calcular la raíz


Se ha dado un error.
```

5.- Cláusulas de prueba.

Cuando se escribe una declaración de prueba, pueden aparecer varias cláusulas después del encabezado de prueba como se muestra a continuación.

Forma de la cláusula	Interpretación
except:	Atrapa todos los tipos de excepciones.
except nombre:	Atrapa una excepción específica.

except nombre as valor:	Atrapa el tipo de excepción y le asigna una instancia.
except (nombre1, nombre2):	Atrapa cualquiera de los tipos de excepciones especificadas.
except (nombre1, nombre2) as value:	Atrapa cualquiera de las excepciones listadas y le asigna una instancia.
else:	Se ejecuta si no se levanta ninguna excepción dentro del bloque try.
finally:	Siempre ejecuta este bloque al salir.

En la práctica podemos especificar cualquier número de cláusulas de excepción, pero para especificar la cláusula "else" debe de existir al menos una cláusula "except" (recordar que estamos trabajando con python 3.7, en python 2.x no se podía poner una cláusula "finally" junto a las cláusulas "else" o "except").

Dado que Python busca una coincidencia dentro de un intento determinado al inspeccionar las cláusulas de excepción de arriba abajo, es lo mismo poner:

```
... except (RuntimeError, TypeError, NameError):
...     pass
```

Que poner:

```
... except RuntimeError:
...     pass
... except TypeError,
```

```
...     pass
... except NameError:
...     pass
```

Poner la excepción entre paréntesis tiene el mismo efecto que enumerar cada excepción en su propia cláusula de excepción, pero debe codificar el cuerpo de la declaración asociado a cada una solo una vez. . Este es un ejemplo de varias cláusulas de excepción en el trabajo, que demuestra cuán específicos pueden ser sus manejadores:

5.- La cláusula assert

La cláusula "assert" se puede usar como una forma resumida de utilización de la cláusula "raise", y es utilizada mayormente para verificar las condiciones de un programa durante la etapa de desarrollo, como testear errores de programación cuando no estás seguro de código escrito por otra persona o realizar testeos de prueba para ver si los valores esperados son correctos. Veamos un ejemplo para clarificar su uso, supongamos que debemos calcular la raíz de un número retornado por un programa que realizo otra persona, en donde el número debe ser siempre positivo, y no estamos seguros de que el otro programador haya realizado bien su trabajo, en tal caso podemos utilizar un assert para tomar en cuenta esta posibilidad:

excepcion5.py

```
1  import para_calcular_raiz
2
3  assert para_calcular_raiz.numero >= 0, 'El valor de numero debe ser mayor a cero'
4  print(para_calcular_raiz.numero**0.5)
```

En este ejemplo en el caso de que la variable “valor” sea menor que cero lanzamos una excepción.

```
Traceback (most recent call last):
  File "C:\Users\juanb\Desktop\excepcion5.py", line 3, in <module>
    assert numero >= 0
NameError: name 'numero' is not defined
```

Es importante tener en cuenta que assert está diseñado principalmente para atrapar restricciones definidas por el usuario, no para detectar errores de programación genuinos. Python genera excepciones en los errores de forma automática, así que casi no hay necesidad de utilizar un “assert” aunque en algunos casos nos podemos como siempre encontrar con excepciones.

6.- La cláusula raise y el uso de clases.

Ya hemos visto que podemos utilizar la cláusula “raise” para lanzar una excepción, opcionalmente podemos a continuación de raise poner una clase o instancia de la misma. Desde python 2.6 y python 3.x en adelante las excepciones son instancias de clases.

Python trae incorporadas varios tipos de excepciones, como por ejemplo “IndexError”, la cual permite analizar si existe un error con un determinado índice, un ejemplo del uso de “IndexError” puede ser el buscar una determinada posición dentro de un string, sin embargo si la posición que estamos buscando está fuera de la longitud de caracteres del string podemos evitar que se dé un error atrapandolo con IndexError.

excepcion6.py

```
1  def buscar(objeto, clave):
2      return objeto[clave]
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

```
3
4  palabra = "Manzana"
5  print(buscar('manzana', 3))
6
7  try:
8      print(buscar('manzana', 31))
9  except IndexError:
10     print('Ha existido un error')
```

Retorna:

```
z
Ha existido un error
```

Cuando trabajamos con las clases de excepciones incorporadas en el núcleo de python, podemos adoptar cualquiera de los siguientes formatos:

```
raise IndexError
raise IndexError()
```

En ambos casos se está lanzando una instancia de la excepción, pero en el primer caso lo estamos realizando de forma implícita. También nos está permitido crear una instancia fuera de tiempo de la siguiente forma:

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

```
excepcion = IndexError()
raise excepcion
```

Ó

```
excepcion = [IndexError, TypeError]
raise excepcion[0]
```

No solo podemos lanzar clases incluidas en el núcleo, sino que además podemos crear nuestras propias clases de excepciones si las hacemos heredar de la clase "Exception", de esta forma podemos utilizar la cláusula "raise" para lanzar clases de excepción personalizadas, en donde una clase propia en una cláusula "except" es compatible con una excepción si la misma está en la misma clase o una clase base de la misma

excepcion7.py

```
1  class B(Exception):
2      color = "ROJO"
3
4  class C(B):
5      color = "VERDE"
6
7  class D(C):
8      color = "AZUL"
9
10 for cls in [B, C, D]:
```

```
11    try:
12        raise cls()
13    except D:
14        print("D")
15        print(D.color)
16    except C:
17        print("C")
18        print(C.color)
19    except B:
20        print("B")
21        print(B.color)
```

La ejecución anterior nos retorna:

```
B
ROJO
C
VERDE
D
AZUL
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Como vemos, los programas pueden nombrar sus propias excepciones creando una nueva clase para manejar un tipo particular de excepción, las cuales deben heredar de la clase Exception, directa o indirectamente.

Las clases de Excepciones pueden ser definidas de la misma forma que cualquier otra clase, pero usualmente se mantienen simples, a menudo solo ofreciendo un número de atributos con información sobre el error que leerán los manejadores de la excepción. Al crear un módulo que puede lanzar varios errores distintos, una práctica común es crear una clase base para excepciones definidas en ese módulo y extenderla para crear clases excepciones específicas para distintas condiciones de error. La estructura podría ser así:

```
class Error(Exception):
    """Clase base para excepciones en el módulo."""
    pass


class EntradaError(Error):
    """Excepción lanzada por errores en las entradas.

    Atributos:
        expresion -- expresión de entrada en la que ocurre el error
        mensaje -- explicación del error
    """

    def __init__(self, expresion, mensaje):
        self.expresion = expresion
        self.mensaje = mensaje
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

```
class TransicionError(Error):
    """Lanzada cuando una operación intenta una transición de estado no permitida.

    Atributos:
        previo -- estado al principio de la transición
        siguiente -- nuevo estado intentado
        mensaje -- explicación de por qué la transición no está permitida
    """
    def __init__(self, previo, siguiente, mensaje):
        self.previo = previo
        self.siguiente = siguiente
        self.mensaje = mensaje
```

Nota: La mayoría de las excepciones son definidas con nombres que terminan en "**Error**", similares a los nombres de las excepciones estándar.

Nota: Muchos módulos estándar definen sus propias excepciones para reportar errores que pueden ocurrir en funciones propias.

Podemos encontrar un listado completo de los tipos de excepciones que python 3.7 tiene incluido por defecto en: ***https://docs.python.org/3/library/exceptions.html***

7.- Métodos en clases de excepciones

La utilización de métodos en las clases de excepción es una excelente herramienta para tener en cuenta en nuestros desarrollos, una aplicación muy útil para ver el potencial que tenemos, podría ser registrar los errores que se dan en nuestra plataforma para posteriormente poder informar al administrador de la misma que se ha producido un evento inesperado. Muchas aplicaciones vienen con un registro de log (archivo en el cual se graban los errores) que nos brinda la información necesaria para poder ubicar los problemas.

excepcion8.py

```
1  import os
2  import datetime
3
4  class RegistroLogError(Exception):
5
6      ruta = os.path.dirname(os.path.abspath(__file__))+"\\log.txt"
7      def __init__(self, linea, archivo, fecha):
8          self.linea = linea
9          self.archivo = archivo
10         self.fecha = fecha
11
12     def registrar_error(self):
13         log = open(self.ruta, 'a')
14         print('Se ha dado un error:', self.archivo, self.linea, self.fecha, file=log)
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

```
15
16  def registrar():
17      raise RegistroLogError(7, 'archivo1.txt', datetime.datetime.now())
18
19  try:
20      registrar()
21  except RegistroLogError as log:
22      log.registrar_error()
```

Nota: Este script permite registrar en el archivo log.txt los datos de error, si el archivo inicialmente no existe lo crea.

8.- Anidamiento de excepciones.

Hasta aquí solo hemos utilizado un intento para detectar excepciones, pero las declaraciones de prueba pueden anidarse, tanto en términos de sintaxis como de flujo de control de tiempo de ejecución a través de su código. Cuando se produce una excepción, Python regresa a la última declaración de prueba ingresada con una cláusula de excepción coincidente. Debido a que cada declaración de prueba deja un marcador, Python puede saltar a las versiones anteriores al inspeccionar los marcadores apilados.

Consideremos un ejemplo para clarificar la idea, en el cual tenemos un nivel de anidamiento.

excepcion9.py

```
1  def evento2():
2      print(1 + 'Manzana')
3
4  def evento1():
5      try:
6          evento2()
7      except TypeError:
8          print('try interno')
9
10 try:
11     evento1()
12 except TypeError:
13     print('try externo')
```

Nos retorna:

```
ry interno
```

Podemos observar cómo si bien existen dos declaraciones del tipo “try” activas (la que está en el nivel superior y la que está dentro de evento1()), Python selecciona y ejecuta solo el intento más reciente con una coincidencia except, que en este caso es el intento dentro de evento1. El lugar donde una excepción termina saltando depende del flujo de control a través del programa en tiempo de ejecución, debido a esto, para saber a dónde irá, es necesario saber en dónde ha estado.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Bibliografía utilizada y sugerida

Libros y otros manuscritos:

Programming Python 5th Edition – Mark Lutz – O'Reilly 2013

Programming Python 4th Edition – Mark Lutz – O'Reilly 2011

Manual online

https://docs.python.org/3.7/tutorial/

https://docs.python.org/3.7/library/index.html

https://docs.python.org/3/distutils/introduction.html