PROGRAMACIÓN PYTHON

UNIDAD 3 - Manejo de excepciones.

Una excepción ocurre cuando sucede algo inesperado en el programa, generalmente ocurren cuando se produce un error, un método no termina correctamente o como su nombre lo indica, por alguna razón, el programa termina de manera excepcional por alguna situación anormal.

Es muy importante aprender a manejar las excepciones ya que ocasionan que las aplicaciones finalicen de manera abrupta y generalmente los usuarios no obtienen información de lo que sucedio.

Para evitar situaciones de ese tipo, se debe recurrir al manejo de las excepciones, de esta manera, es posible controlar el comportamiento del programa cuando alcanza una situación de error.

Como ejemplo, se usará el caso de la división:

```
a = 20
b = 7
print(a/b)
```

```
2.857142857142857
```

En el caso anterior, la división puede realizarse son problema debido a que los valores de las variables son válidos, sin embargo ¿Qué sucede si la variable `b` tiene el valor de 0?

```
b = 0
print(a/b)

print("El resultado es")
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call
last)
<ipython-input-8-760a633042ce> in <module>
      1 b = 0
----> 2 print(a/b)
      3
      4 print("El resultado es")

ZeroDivisionError: division by zero
```

Se alcanza la siguiente excepción: `ZeroDivisionError: division by zero`, esto ocurre por que matematicamente la división entre cero no está definida, por tanto, ningún lenguaje de programación sabe como realizar dicha operación.

Otra excepción podría suceder si alguna variable fuera un objeto que no puede realizar la operación de división:

```
b = "Hola"

print (a/b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
last)
<ipython-input-9-00f724d8c81c> in <module>
      1 b = "Hola"
      2
----> 3 print (a/b)

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

En este caso se alcanza la excepción de: `TypeError: unsupported operand type(s) for /: 'int' and 'str'`.

Aunque es posible agregar robustez al código mediante estructuras de selección `if - else`, generalmente resulta imposible e impráctico realizar todas las comprobaciones de forma manual de una en una.

```
b = 0

if b!= 0:
  print(a/b)
else:
  print("No se puede dividir por 0")
```

```
No se puede dividir por 0
```

Otra manera de lanzar excepciones es cuando se conozca de algún caso en el que el programa puede incurrir en un error. Para esas situaciones se hace uso de la función `raise()`:

```
b = 85

if type(b) is not int:
  raise Exception("Error de tipo - Unicamente se permiten enteros.")
elif b != 0:
  print(a/b)
else:
  print("La división entre 0 no esta definida")
```

```
0.23529411764705882
```

En el ejemplo anterior, `b` toma el valor de una cadena, por tanto, el programa cae en una excepción de tipo en la cual para realizar la división únicamente se aceptan enteros, de este modo lanzamos una excepción personalizada.

Para ver la lista de todas las excepciones que tiene Python, es necesario consultar la documentación oficial respecto a las excepciones.

Si bien la solución anterior permite visualizar las posibles condiciones de error, aún no se están manejando correctamente y el programa continua deteniéndose en la excepción. Para evitarlo es necesario utilizar los bloques `try - except`. La sintaxis es la siguiente:

```
try:
  Código que lanzará alguna o varias excepciones.
except:
  Lo que realizará el programa en caso de alcanzar la excepción especificada.
```

Por ejemplo, para capturar las dos posibles excepciones de la división y hacer que el programa no termine anormalmente el código sería:

```
b = "hola"
try:
  print(a/b)
except ZeroDivisionError:
  print("No se puede dividir entre 0")
except TypeError:
  print("No se puede dividir una cadena")
```

```
No se puede dividir una cadena
```

El segmento anterior permite tratar cada excepción de manera separada, aunque si se requiere tratar ambas excepciones de la misma manera, es posible definirlas dentro de un bloque `except()`:

```
b = "hola"

try:
  print(a/b)
except (ZeroDivisionError, TypeError):
  print("No se puede hacer división")
```

```
No se puede hacer división
```

Si son muchas las posibles excepciones o no se conocen, se puede utilizar la clase base de las mismas, la cual es `Exception`, una vez que caiga dentro de ella, es posible identificarla:

```
b = "hola"

try:
  print(a/b)
except Exception as e:
  print("No se puede hacer división", type(e))
```

```
No se puede hacer división <class 'TypeError'>
```

Además de los bloques anteriores, Python permite añadir un bloque `else` el cual se ejecutará sino se alcanzó alguna excepción:

```
b = 4
a = 20

try:
  print(a/b)
except Exception as e:
  print("No se puede realizar la divisón", type(e))
else:
  print("No hubo excepcion alguna")
```

```
5.0
No hubo excepcion alguna
```

Un último bloque que es posible agregar a los anteriores, es `finally`. Este bloque siempre se ejecutará haya habido o no alguna excepción:

```
b = "hola"

try:
  print(a/b)
except Exception as e:
  print("No se puede realizar la división.", type(e))
else:
  print("No hubo excepcion alguna.")
finally:
  print("Programa terminado")
```

```
No se puede realizar la división. <class 'TypeError'>
Programa terminado
```

```
***ESCRIBE Y COMPARTE TUS REFLEXIONES DE LOS TEMAS VISTOS ANTERIO
Prueba:

b = "hola"

if type(b) is not int:
  raise Exception("Error de tipo - Unicamente se permiten enteros
elif b != 0:
  print(a/b)
else:
  print("La división entre 0 no esta definida")

**Exception producida**
---------------------------------------------------------------------
Exception                                 Traceback (most recent
<ipython-input-12-5b29e5a58f4a> in <module>
      2
      3 if type(b) is not int:
----> 4   raise Exception("Error de tipo - Unicamente se permiten
      5 elif b != 0:
      6   print(a/b)

Exception: Error de tipo - Unicamente se permiten enteros.

Me pareció muy interesante la forma validar valores de tipo caden
ejemplo en el caso cuando la variable tenía asignado valor "Hola
validar por medio de cadena de caracteres.
```

ESCRIBE Y COMPARTE TUS REFLEXIONES DE LOS TEMAS VISTOS ANTERIORMENTE. Prueba:

b = "hola"

if type(b) is not int: raise Exception("Error de tipo - Unicamente se permiten enteros.") elif b != 0: print(a/b) else: print("La división entre 0 no esta definida")

Exception producida

Exception Traceback (most recent call last) in 2 3 if type(b) is not int: ----> 4 raise Exception("Error de tipo - Unicamente se permiten enteros.") 5 elif b != 0: 6 print(a/b)

Exception: Error de tipo - Unicamente se permiten enteros.

Me pareció muy interesante la forma validar valores de tipo cadena por ejemplo en el caso cuando la variable tenía asignado valor "Hola" y se podía validar por medio de cadena de caracteres.

Haz doble clic (o ingresa) para editar

✓ 0 s se ejecutó 19:41