Manejo de exepeciones

March 30, 2022

Ejecuta el siguiente bloque de código siempre antes de ejecutar el resto del notebook.

[ ]:
```
from IPython.core.magic import  Magics, magics_class, cell_magic, line_magic

@magics_class
class Helper(Magics):

  def __init__(self, shell=None,  **kwargs):
    super().__init__(shell=shell, **kwargs)

  @cell_magic
  def debug_cell_with_pytutor(self, line, cell):
    import urllib.parse
    url_src = urllib.parse.quote(cell)
    str_begin = '<iframe width="1000" height="500" frameborder="0" src="https://
↪pythontutor.com/iframe-embed.html#code='
    str_end   = '&cumulative=false&py=3&curInstr=0"></iframe>'
    import IPython
    from google.colab import output
    display(IPython.display.HTML(str_begin+url_src+str_end))

get_ipython().register_magics(Helper)
```

1 Manejo de Excepciones

En esta sección vamos a eliminar la suposición de que el usuario sabe cómo funciona nuestro programa, es natural que no sepa cómo debe ingresar la información para que nuestra aplicación le entregue el resultado que espera. Incluso, es muy fácil que "rompa" nuestra ejecución por ejemplo al ingresar un string cuando esperábamos un número. Vamos a aprender a anticiparnos a ese tipo de fallos.

Los errores en Python pueden ser de dos tipos: errores de sintaxis y excepciones. Los errores son los problemas en un programa debido a los cuales el programa detendrá la ejecución, por ejemplo, escribir `fir` en lugar de `for` o no poner `:` luego al final de una sentencia `if`. Por otro lado, las excepciones se producen cuando ocurren algunos eventos internos que cambian el flujo normal del programa.

1.1 Diferencia entre errores de sintaxis y excepciones

- **Error de sintaxis**: Como su nombre lo indica, este error es causado por una sintaxis incorrecta en el código. Lleva a la terminación del programa.

[ ]:
```
fir i in range(1,10):
    print(i)
```

```
  File "<ipython-input-1-42ca5e5fbf09>", line 1
    fir i in range(1,10):
        ^
SyntaxError: invalid syntax
```

Excepciones: Las excepciones se plantean cuando el programa es sintácticamente correcto, pero el código dio lugar a un error. Este error no detiene la ejecución del programa, sin embargo, cambia el flujo normal del programa.

[ ]:
```
#Creemos uno de las excepciones más comunes,
#La división por cero y en uno de los escenarios más comunes

particiones = 1

for i in range(10):
    particiones /= i

print(particiones)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-631a4623e90b> in <module>()
      5
      6 for i in range(10):
----> 7     particiones /= i
      8
      9 print(particiones)

ZeroDivisionError: division by zero
```

En el reporte de error anterior, `ZeroDivisionError` es la excepción que generó nuestro código al ejecutarse de forma inapropiada. Python tiene muchos tipos de excepciones que responden a distintos tipos de error, en este enlace encontrarás una lista con la gran cantidad que hay dentro del lenguaje, sin embargo, no estudiaremos en detalle todas y cada una.

Algunos de los Errores de Excepción más comunes son :

- `IOError`: si el archivo no se puede abrir
- `KeyboardInterrupt`: cuando el usuario pulsa una tecla no requerida

- `ValueError`: cuando la función incorporada recibe un argumento erróneo
- `EOFError`: si se llega al final del archivo sin leer ningún dato
- `ImportError`: si no se puede encontrar el módulo
- `TypeError`: cuando se realizan operaciones con tipos no compatibles entre sí.

1.2 Try Except en Python

Las sentencias `try` y `except` se utilizan para manejar estas excepciones dentro de nuestro código en Python. El bloque `try` se utiliza para comprobar si hay errores en el código, es decir, el código dentro del bloque `try` se ejecutará cuando no haya ningún error en el programa. Mientras que el código dentro del bloque `except` se ejecutará cuando el programa encuentre algún error en el bloque `try` anterior.

```
try:
    instrucciones
except:
    instrucciones
    #se ejecuta si hubo error en try
```

1.2.1 ¿Cómo funciona try()?

Primero se ejecuta la sentencia `try`, es decir, el bloque de código asociado a dicha sentencia. Si no hay ninguna excepción, entonces sólo se ejecutará el `try`, y se salta el `except`.

Si se produce alguna excepción, el bloque del `try` se interrumpe y se ejecuta el bloque del `except`. Si ocurre alguna excepción, pero la sentencia `except` dentro del código no la maneja, se pasa a las sentencias try externas, si es que las hay (es decir, si tenemos bloques try-except anidados). Si la excepción no se maneja, entonces la ejecución se detiene. Una sentencia try puede tener más de un bloque `except`

[ ]:
```
#Resolvamos la excepción del ejercicio anterior

particiones = 1

for i in range(10):
    try:
        particiones /= i
    except:
        print('división por cero, el ciclo ignorará la operación')

print(particiones)
```

```
división por cero, el ciclo ignorará la operación
2.7557319223985893e-06
```

Podemos controlar el tipo de excepción específica añadiendo el tipo de error al bloque `except`, además, podemos asignar una variable para almacenar el mensaje de error usando la sentencia `as`:

[ ]:
```
a = input('digite un número: ')
try:
```

```
    print(a + 15)
except TypeError as e:
    print('La variable no se almacenó como número')
    print('error: ',e, sep='\n')
```

```
digite un número: 465
La variable no se almacenó como número
error:
can only concatenate str (not "int") to str
```

1.3 Palabra clave finally en Python

Python cuenta, además, con la instrucción `finally`, que siempre se ejecuta después de los bloques try y except. El bloque final siempre se ejecuta después de la terminación normal del bloque try o después de que el bloque try termine debido a algunas excepciones.

```
try:
    instrucciones
except:
    instrucciones
    #se ejecuta si hubo error en try
finally:
    instrucción
    #siempre se ejecuta
```

[ ]:
```
particiones = 1

for i in range(10):
    try:
        particiones /= i
    except:
        print('división por cero, el ciclo ignorará la operación')
    finally:
        print('\ndivisión parcial: ',particiones)

print('\nresultado final',particiones)
```

```
división por cero, el ciclo ignorará la operación

división parcial:  1

división parcial:  1.0

división parcial:  0.5

división parcial:  0.16666666666666666

división parcial:  0.041666666666666664
```

```
división parcial:  0.008333333333333333

división parcial:  0.001388888888888889

división parcial:  0.0001984126984126984

división parcial:  2.48015873015873e-05

división parcial:  2.7557319223985893e-06

resultado final 2.7557319223985893e-06
```