Programación de Computadores

Manejo de Excepciones

Jonatan Gómez Perdomo, Ph. D.
jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.
aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D. (c)
eccubidesg@unal.edu.co

Research Group on Artificial Life – Grupo de investigación en vida artificial – (Alife)
Computer and System Department
Engineering School
Universidad Nacional de Colombia

Contenido

Agenda

Definición I

En general los programas usan librerías, operaciones preexistentes o funciones nuevas que son especificadas para que funcionen con un cierto tipo de datos, con unos ciertos valores, o bajo unas ciertas condiciones. En algunas ocasiones este tipo de especificaciones no se pueden garantizar. Por ejemplo, cuando se lee un archivo, es posible que el dispositivo externo falle y no se permita la lectura del mismo, o que al realizar una división, el divisor resulte aproximado a 0 y no se pueda realizar dicha operación. Algunas veces estas situaciones son derivadas del mal uso que realizan los usuarios del programa al ingresar valores inválidos.

Definición II

Este tipo de situaciones, que son consideradas **excepcionales**, pueden afectar el flujo "normal" del programa interrumpiendo en muchos casos su ejecución.
Un programador puede considerar estas situaciones inesperadas y darle un **manejo excepcional** a las mismas para que no interrumpan la ejecución del programa.

Agenda

Ejemplo I. Dividiendo por cero

Para el programa.

```
def division(a, b):
    coc = a//b
    res = a % b
    return (coc, res)
division(4,5)
print(division(10, 0))
print(division(1024,10))
```

La salida obtenida es:

Ejemplo I. División por cero

```
ZeroDivisionError  Traceback (most recent call last)
<ipython-input-24-af58c2d4968e> in <module>
      6 division(4,5)
      7
----> 8 print(division(10, 0))
      9 print(division(1024,10))
<ipython-input-24-af58c2d4968e> in division(a, b)
      1 def division(a, b):
----> 2     coc = a//b
      3     res = a % b
      4     return (coc, res)
      5
ZeroDivisionError: integer division or modulo by zero
```

Agenda

Sintaxis

El manejo de excepciones tiene la siguiente forma:

```
 try:
    aquí van las operaciones
except Exception_name: #Primer tipo de excepción
    código a ejecutar si sucedió la excepción Exception_name
except Exception_name2: #Segundo tipo de excepción (opcional)
    código a ejecutar si sucedió la excepción Exception_name2
 ...
except Exception_nameN: #Segundo tipo de excepción (opcional)
    código a ejecutar si sucedió la excepción Exception_nameN
else: #opcional
    si no hay excepción se ejecuta este código
```

Se pueden definir varios except uno por cada tipo de excepción, si se quiere un manejo diferenciado.

Ejemplo I. Manejo excepción división por cero

Para el programa.

```
def division(a, b):
  try:
    coc = a//b
    res = a % b
    return (coc, res)
  except:
    print(f'Error en la división de a entre b')
    return ''
print(division(10, 0))
print(division(1024,10))
```

Ejemplo I. Manejo excepción división por cero

La salida obtenida es:

```
Error en la división de 10 entre 0
(102, 4)
```

Agenda

Ejemplo II. Valor no apropiado

Para el programa, posibles errores incluyen digitar texto en num o div.

```
def division(a, b):
  try:
    coc = a//b
    res = a % b
    return (coc, res)
  except:
    print('Error en la división de  entre '.format(a, b))
def main():
  num = int(input('digite el dividendo: '))
  div = int(input('digite el divisor: '))
  print(division(num, div))
main()
```

La salida obtenida es:

Ejemplo II. Valor no apropiado

```
digite el dividendo: hola
---------------------------------------------------------------------------
ValueError Traceback (most recent call last)
<ipython-input-33-e4c231c802f0> in <module>
     13     print(division(num, div))
     14
---> 15 main()
<ipython-input-33-e4c231c802f0> in main()
      9
     10 def main():
---> 11     num = int(input('digite el dividendo: '))
     12     div = int(input('digite el divisor: '))
     13     print(division(num, div))
ValueError: invalid literal for int() with base 10: 'hola'
```

Ejemplo II. Manejo excepción valor no apropiado

Para el programa.

```
def division(a, b):
  try:
    coc = a//b
    res = a % b
    return (coc, res)
  except ZeroDivisionError:
    print('La división por cero no está definida.')
    return ''
def main():
  try:
    num = int(input('digite el dividendo: '))
    div = int(input('digite el divisor: '))
    print(division(num, div))
  except ValueError:
    print('El valor digitado no es entero.')
main()
```

Ejemplo II. Manejo excepción valor no apropiado

La salida obtenida es:

```
digite el dividendo: diez
El valor digitado no es entero.
```

Agenda

El bloque finalmente `finally`

Se especifica el bloque `finally` para determinar acciones que se deben ejecutar sin importar si se produce una excepción o no

```
try:
    #run this action first
except:
    # Run if exception occurs
Finally :
    #Always run this code
```

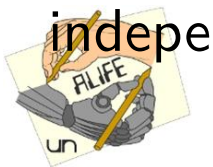
El orden de ejecución de las excepciones es el siguiente:
`try -> except -> else -> finally`

Ejemplo: El bloque finalmente `finally`

Para el siguiente código:

```
try:
    num = int(input("Enter the number "))
    re = 100/num
except:
    print("Something is wrong")
else:
    print ("result is ",re)
finally :
    print ("finally program ends")
```

Al ejecutar este programa se mostrará el texto `finally program ends` independientemente de que se produzca una excepción o no.



Agenda

Capturar excepciones de varios tipos

Como es tedioso determinar el tipo de excepción se puede utilizar una sóla línea de código. Para el código:

```
try:
    num = int(input("Enter the number ")) # se puede digiart un texto para producir
    re = 100/num #se puede digitar 0
    print(re)
except Exception as e:
    print(e, type(e))
```

Se tiene como salida:

Si el usuario digita cero:

```
Enter the number 0
division by zero <class 'ZeroDivisionError'>
```

Si el usuario digita hola:

```
Enter the number hola
invalid literal for int() with base 10: 'hola' <class 'ValueError'>
```

Agenda

Lanzar una excepción (raise):

Es posible lanzar excepciones utilizando raise. Para el código:

```
raise ValueError('error de división por cero')
```

La salida del programa anterior es:

```
ValueError                                Traceback (most recent call last)
<ipython-input-40-496f78253296> in <module>
----> 1 raise ValueError('error de división por cero')

ValueError: error de división por cero
```

Agenda

Definiendo mis propias excepciones:

Para definir nuestras propias excepciones es necesario crear una clase que hereda de la clase Exception. Por ahora puede verse el ejemplo siguiente como una plantilla pues no se han visto clases. Para el código:

```
class NoPuedeDigitarDosException(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return self.value

def main():
    try:
        num = int(input())
        if num == 2: #si el usuario digita 2 se lanza la excepción
            raise NoPuedeDigitarDosException('El usuario digitó 2 y no se puede!')
        else:
            print('all is well it is not 2')
    except Exception as e:
        print(e, type(e))
main()
```