# **Módulo 3**

**Modificador de flujos**

# **Manejo de excepciones.**

* + **try:**
  + **except:**
  + **else:**
  + **finally:**
  + **Raise:**

**h[ttps//docs.python.org/3/library/exceptions.html#exception-hierarchy](https://docs.python.org/3/library/exceptions.html" \l "exception-hierarchy)**

## **Manejo de Errores / excepciones**

En la prehistoria de la programación ( hace unos 10 años) si un programa corria y se encontraba con un error este se rompía, se cortaba el flujo del programa y terminaba. Todo hemos visto la pantalla azul de windows.

El manejo de errores permitió en un inicio que si encontraba un error se derivara el flujo del programa a alguna parte (función o método) que pudiera salvarlo (zafar) .

Luego se permitió capturar el error, saber que fue lo que causo el problema y poder derivar el flujo alternativamente a varias (funciones o métodos) dependiendo de como se pudiera saltar o solucionar el problema.

Bien. Sabemos que hay personas que le das la mano y se toman hasta el codo. En este caso todo esto derivo a que se tomen hasta el cuello y mas también.

Hoy uno puede crear una excepción simplemente si un valor no esta dentro de lo procesable y se crea una excepción(ya no lleva el nombre de error, porque ya no lo es) para salir del flujo normal

En Python, el manejo de excepciones es una técnica fundamental que te permite controlar y manejar situaciones excepcionales o errores que pueden ocurrir durante la ejecución de tu programa. Esto se logra mediante bloques `try:`, `except:`, `else:` y `finally:`.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ·   |  | | --- | | **try:** Este bloque es donde se coloca el código que podría generar una excepción. El intérprete de Python intentará ejecutar el código en este bloque y monitoreará si ocurre alguna excepción. |   ·   |  | | --- | | **except**: Si ocurre una excepción dentro del bloque `try`, el flujo del programa se dirigirá a el o los bloque(s) `except`. Aquí es donde manejas la(s) excepción(es) en función del tipo de error que ocurrió. Puedes tener múltiples bloques `except` para manejar diferentes tipos de excepciones. |   ·   |  | | --- | | **else**: Este bloque es opcional y se ejecutará solo si no se ha producido ninguna excepción en el bloque `try`. Es útil para separar el código que puede generar excepciones del código que debería ejecutarse solo cuando no se producen errores. |   ·   |  | | --- | | **finally**: Este bloque también es opcional y se ejecutará sin importar si se ha producido una excepción o no en el bloque `try`. Se utiliza para tareas que deben realizarse siempre, como la liberación de recursos, sin importar si se produjo una excepción o no. |   · |

Nuestro paradigma ahora tiene una m¡nueva modificación, trata de anticipar posibles excepciones y diseñar tu código para evitar errores en lugar de simplemente capturarlos.

**Raise:** se utiliza para generar o lanzar manualmente una excepción en un punto específico del código donde el programador desea cambiar el flujo normal del programa. Puedes utilizarla para indicar que ha ocurrido una situación excepcional (o no tanto).

|  |
| --- |
| **Errores en tiempo de ejecución.** **Python no se rompe, se puede obtener un código con salidas correctas sin cuelgues, ya que no sólo es posible identificar y rastrear los errores, sino tomar acciones correctivas y preventivas.** |

·

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ·Código Python   |  | | --- | | try:  # Es ley. Nunca el usuario hará lo que el programador espera.  num\_usuario = input("Ingrese un número: ")#ingreso desde el usuario como str  num\_usuario = int(num\_usuario)#casting de str a int  resultado = 10 / num\_usuario  print(f"Correcto: El resultado es:{resultado}")  except ZeroDivisionError:  print("No se puede dividir por cero.")  except ValueError:  print(f" '{num\_usuario}' no es valido: Ingrese un número en caracteres decimales.")  else:  print(f"Salida sin errores")  finally:  print("the end…..") # o no?????? |   ·Salida esperada por consola   |  | | --- | | Ingrese un número: 20 **< ----------------------------------------------- ingreso por usuario**  Correcto: El resultado es:0.5----------------------------------------------- > salida del try  Salida sin errores------------------------------------------------------------- > salida del else  the end…..-------------------------------------------------------------------- > salida del finally |   ·Salida esperada por consola   |  | | --- | | Ingrese un número: veinte  **< ----------------------------------------------- ingreso por usuario**  'veinte' no es valido: Ingrese un número en caracteres decimales.--- > excepción ValueError  the end…..-------------------------------------------------------------------- > salida del finally |   ·Salida esperada por consola   |  | | --- | | Ingrese un número:  **0 < ------------------------------------------------- ingreso por usuario**  **No se puede dividir por cero.**----------------------------------- > excepción ZeroDivisionError  the end…..-------------------------------------------------------------------- > salida del finally |   · |

·

Se puede usar **except Exception as variable** en bloques except para capturar la excepción en una variable y luego acceder a sus atributos:

|  |  |  |
| --- | --- | --- |
| ·Código Python   |  | | --- | | for intento in range (1,4):  try:  num\_usuario = input(f"Ingrese una edad ({intento=}): ")#ingreso desde el usuario como str  num\_usuario = int(num\_usuario)#casting de str a int  if 0<num\_usuario<125:  print(f"Correcto: puede ser una edad válida:{num\_usuario}")  break  except Exception as error:  print (f"otro tipo de {error=}")  else:  print (f"cantidad de intentos {intento}") |   ·   |  | | --- | | **Sigue el flujo y piensa los tipos de salida** |   · |

·

En Python siguen una jerarquía de clases. BaseException es la clase base de todas las excepciones. Puedes capturar todas las excepciones utilizando except BaseException.

Se pueden consultar los diversos tipos de excepciones en el siguiente link:

h[ttps://docs.python.org/3/library/exceptions.html#exception-hierarchy](https://docs.python.org/3/library/exceptions.html" \l "exception-hierarchy)

Aquí lanzo una excepción que no existe. Es simplemente que espero un valor distinto a 0

|  |  |  |
| --- | --- | --- |
| ·Código Python   |  | | --- | | def funcion\_personalizada(valor):  try:  if valor > 0:  salida = "positivo"  elif valor < 0:  salida = "negativo"  else:  raise Exception("El valor no es negativo ni positivo")  return salida  except Exception as e:  print("Error:", e)  resultado = funcion\_personalizada(0)  print (f"{resultado=}") |   ·   |  | | --- | | **Sigue el flujo y piensa los tipos de salida** |   · |

·

En resumen, la palabra clave raise te permite crear excepciones y controlar manualmente el flujo de excepciones en tu código, lo que puede ser útil para manejar situaciones excepcionales que no son manejadas por las excepciones incorporadas de Python.

|  |  |  |
| --- | --- | --- |
| ·Código Python   |  | | --- | | while True:  try:  nombre= input("Ingrese su nombre:").title()  if len (nombre)>3 and nombre.replace(" ","").isalpha():  print (f"{nombre=} ingrese por favor")  break  raise NameError('Tu nombre no esta en lista')  except NameError:  print('Adios!, siguiente') |   ·   |  | | --- | | **Sigue el flujo y piensa los tipos de salida**  **nombres de menos de 3 caracteres o números o signos en el string** |   · |

·

|  |  |  |
| --- | --- | --- |
| ·Código Python   |  | | --- | | def func():  raise ConnectionError  try:  func()  except ConnectionError as exc:  raise RuntimeError('no se pudo conectar') from exc |   ·   |  | | --- | | **Sigue el flujo y piensa los tipos de salida** |   · |

·