**Módulo de 11:**

# **Manejo de excepciones:**

* try
* except
* else
* finally
* raiser

**Manejo de excepciones**

En un principio se trabajaba con manejo de errores. Estos fue mejorado hasta que no tengan que ser solo errores, sino que cualquier tipo de excepción o eventos que ocurren durante la ejecución y que interrumpen el flujo normal del programa.

El manejo de excepciones es una técnica utilizada para controlar y gestionar esta situaciones excepcionales que pueden ocurrir durante la ejecución de un programa.

|  |
| --- |
| En Python, el manejo de excepciones se realiza utilizando bloques   * + try   + except   + else   + finally   y la función   * + raiser |

El bloque try se utiliza para encapsular el código que puede generar una excepción,

El bloque except se utiliza para manejar las excepciones capturadas.

El bloque else se utiliza tras un evento que no ingresa por excepción.

El bloque finally se utiliza como último bloque. Esto es muy importante en una función con return

|  |
| --- |
| try:  resultado = 10 / 0 #<------------------------------- Dividimos sobre cero  except Exception as Error:  print(f"Error encontrado: {Error}") |

·

|  |
| --- |
| Error encontrado: division by zero |

·

|  |
| --- |
| try:  resultado = 2 \* x #<------------------------------- o x no existe  except Exception as Error:  print(f"Error encontrado: {Error}") |

·

|  |
| --- |
| Error encontrado: name 'x' is not defined |

·

Es posible utilizar varios bloques except para capturar diferentes tipos de excepciones y manejarlas de manera adecuada. Además, se puede utilizar un bloque else para especificar código que se ejecutará solo si no se generó ninguna excepción en el bloque try. También se puede utilizar un bloque finally para especificar código que se ejecutará siempre, independientemente de si se generó una excepción o no.

|  |
| --- |
| try:  ingreso = int(input("Ingrese un número: "))  resultado = 10 / ingreso  print (f"resultado = {resultado}")  except ValueError:  print("Error: Ingrese un número válido")#< ---------------------- el usuario no ingreso un entero  except ZeroDivisionError:  print("Error: División entre cero")#< --------------------------- el usuario ingreso 0 cero  #---------------------------------------------------------------------  else:  print("¡No se produjo ninguna excepción!")  finally:  print ("the end..... :)") |

·

|  |
| --- |
| Ingrese un número: **5#<-------------------------------ingreso de la altura del usuario**  resultado = 2.0  ¡No se produjo ninguna excepción!  the end..... :) |

·

Sin errores y al final el bloque finally

·

|  |
| --- |
| Ingrese un número: **dos#<--------------------------ingreso de la altura del usuario**  Error: Ingrese un número válido  the end..... :) |

·

En este ejemplo, se intenta realizar una casting de str a int no valido, lo cual generará una excepción ValueError. El bloque except captura esta excepción y ejecuta el código especificado dentro de él, en este caso, imprime un mensaje de error.

·

|  |
| --- |
| Ingrese un número: **0#<-------------------------------ingreso de la altura del usuario**  Error: División entre cero  the end..... :) |

·

En este ejemplo, se intenta realizar una división entre cero, lo cual generará una excepción ZeroDivisionError. El bloque except captura esta excepción y ejecuta el código especificado dentro de él, en este caso, imprime un mensaje de error.

Python no se rompe. El manejo de excepciones es una técnica para detectar y responder adecuadamente a situaciones excepcionales sin que el script se detenga abruptamente.

"raiser" es una declaración para generar manualmente no un error, sino una “excepción” programada, lanzar explícitamente en un lugar o tiempo específico del código.

La sintaxis básica de la declaración raise es la siguiente:

raise TipoDeExcepcion("Mensaje de error opcional")

Donde TipoDeExcepcion es el tipo de excepción que se desea lanzar, como ValueError, TypeError, FileNotFoundError, o un error opcional que se mostrará cuando se capture la excepción.

|  |
| --- |
| def dividir(a, b):  if b == 0:  raise ValueError("Nunca dividirás por cero")  return a / b  try:  resultado = dividir(10, 0)  except ValueError as Error:  print(f"Error encontrado: {Error}") |

·

|  |
| --- |
| Error encontrado: Nunca dividirás por cero |

La declaración raise se lanza cuando el programador encuentra un evento por el cual genera una excepción propia en este caso un ValueError con el mensaje "Nunca dividirás por cero".

Luego, se captura la excepción en el bloque except y se imprime el mensaje .

No hay error, no hay excepción por bloqueo de script. Raise permite controlar el flujo del programa y señalar condiciones excepcionales de acuerdo con tus necesidades.

**Finally:**

|  |
| --- |
| def mi\_funcion\_dividir(valor\_1,valor\_2):  try:  return valor\_1 / valor\_2  except Exception as Error:  return f"Error encontrado: {Error}"  finally:  return "Estes es el retorno final"  print (f"{mi\_funcion\_dividir(10,2)}")  print (f"{mi\_funcion\_dividir(10,0)}") |

·

mi\_funcion\_dividir(10,2)

|  |
| --- |
| Estes es el retorno final |

·

mi\_funcion\_dividir(10,0)

|  |
| --- |
| Estes es el retorno final |

**No importa que return se ejecute primero. Siempre el que vale es el de finally**

|  |
| --- |
| **Ea muy importante aclarar que el flujo del programa se altera con try except y finally** |

·