
# Errores y Excepciones

Los errores y las excepciones son situaciones inesperadas que pueden ocurrir durante la ejecución de un programa y que impiden que se ejecute correctamente.

## Tipos de errores

En un programa pueden existir diferentes tipos de errores:

- **De Sintaxis**: Estos errores ocurren cuando el código no sigue la gramática y reglas de escritura del lenguaje de programación.
- **De lógica:** ocurren cuando el código no produce el resultado esperado debido a un error en el razonamiento o la lógica detrás de él. Estos errores pueden ser más difíciles de detectar porque el programa se ejecuta sin lanzar excepciones, pero produce resultados incorrectos.
- **De ejecución:** se detectan durante la ejecución del programa, mientras que los errores de lógica pueden producir resultados incorrectos pero no generan errores durante la ejecución.

Define que tipo de errores son los siguientes ejemplos:

```
print ("Hola mundo)
pritn ("Hola mundo")
```

```
if edad > 18:
    print("mayor de edad")
else:
    print("menor de edad")
```

```
def division (x, y):
   return x/y
```

```
print(division(2,0))


def encontrar_maximo(lista_numeros):
    maximo = lista_numeros[0]
    for numero in lista_numeros:
        if numero > maximo:
            maximo = numero
    return maximo


numeros = []
maximo = encontrar_maximo(numeros)
print("El máximo es:", maximo)
```

## Gestión de Excepciones

La gestión de excepciones se da para manejar errores ocurridos cuando estamos ejecutando nuestro programa y el usuario no sigue las indicaciones o suministra información equivocada.

## Sintaxis

```
try:
  codigo_vigilado
except:
  lo_que_se_hace_en_caso
  de_error_en_el_codigo_vigilado


def division (x, y):
  return x/y

print(division(int(input("Ingrese x:")),int(input("Ingrese y:"))))
```

```
def division (x, y):
  return x/y

try:
  print(division(int(input("Ingrese x:")),int(input("Ingrese y:"))))
except:
  print("No se puede dividir")
```

## Tipos Comunes de excepciones:

**a. SyntaxError:** Se produce cuando hay un error de sintaxis en el código.

**b. IndentationError:** Se produce cuando hay problemas con la indentación del código.

**c. NameError:** Se produce cuando se intenta acceder a una variable que no está definida.

**d. TypeError:**Se produce cuando se realiza una operación no válida para un tipo de dato.

**e. ValueError:** Se produce cuando una función recibe un argumento con un valor inapropiado.

**f. ZeroDivisionError:** Se produce cuando se intenta dividir por cero.

**g. IndexError:** Se produce cuando se intenta acceder a un índice fuera de rango en una lista.

**h. KeyError:** Se produce cuando se intenta acceder a una clave que no existe en un diccionario.

**i. FileNotFoundError:** Se produce cuando se intenta abrir un archivo que no existe.

## Excepciones específicas

Algunos códigos vigilados en el bloque **try** pueden generar diferentes tipos de excepciones. Si se tratan con un solo **except** general, el mensaje será el mismo. Es posible que queramos un mensaje diferente para cada tipo de excepción ocurrida...

```
try:
    codigo_vigilado
except TipoError1:
    codigo_que_se_ejecuta
    cuando_ocurre_error1
except TipoError2:
    codigo_que_se_ejecuta
    cuando_ocurre_error2
```

A continuación ejecuta el siguiente código ingresando como dividendo el valor de cero (0). Toma nota del tipo de error. Luego vuelve a ejecutar el código e ingresa una letra como valor del divisor y toma de nuevo nota del tipo de error.

```
def division (x, y):
    return x/y

divisor = int(input("ingrese divisor:"))
dividendo = int(input("ingrese dividendo:"))
print(division(divisor,dividendo))
```

Como puedes ver, en el primer caso (cero como divisor) el tipo error es **ZeroDivisionError** y en el segundo caso (ingreso de una letra) el error es de tipo **ValueError** con estos dos tipos de errores claros, podemos hacer una mejor gestión de excepciones en nuestro código. Veamos, ejecuta de nuevo el siguiente código con los valores que producían error...

```
def division (x, y):
  return x/y

try:
  divisor = int(input("ingrese divisor:"))
  dividendo = int(input("ingrese dividendo:"))
  print(division(divisor,dividendo))
except ZeroDivisionError:
  print("No es posible dividir por cero (0)")
except ValueError:
  print("Número ingresado no válido")
except:
  print("Error al dividir")
```

Ahora podemos ver que generamos un mensaje específico para cada posible tipo de excepción. También podemos notar que en la línea 12 se ha dejado un bloque except de caracter general, el cual sólo se ejecutará si ocurre una excepción en el código vigilado que no ha sido gestionada de manera específica.

En el código anterior el programa termina después de mostrar que ocurrió algún error. **¿Cómo podemos adaptarlo para que solo termine después de ser ejecutado correctamente?**

```
def division (x, y):
  return x/y

while True:
  try:
    divisor = int(input("ingrese divisor:"))
    dividendo = int(input("ingrese dividendo:"))
    print(division(divisor,dividendo))
    break
  except ZeroDivisionError:
    print("No es posible dividir por cero (0)")
  except ValueError:
    print("Número ingresado no válido")
  except:
    print("Error al dividir")
```

## Bloque **else** y **finally**

La gestión de excepciones en python admite otros dos bloques opcionales

El bloque *else* al final de los bloques except se utiliza como el bloque que se ejecuta si ninguna except se ejecutó.

El bloque *finally* se ejecuta siempre que se defina. Este es el último paso que se ejecuta cuando hay gestión de excepciones y su ejecución siempre se da hayan o no excepciones en el programa.

```
try:
    codigo_vigilado
except TipoError1:
    codigo_que_se_ejecuta
    cuando_ocurre_error1
except TipoError2:
    codigo_que_se_ejecuta
    cuando_ocurre_error2
except:
    codigo_que_se_ejecuta
    cuando_ocurre_cualquier
    otro_tipo_de_excepcion
else:
    codigo_que_se_ejecuta
    si_ninguna_excepcion_ocurre
finally:
    codigo_que_siempre_se_ejecuta
    al_final_de_la_gestion_de
```

```
def division (x, y):
  return x/y

try:
  divisor = int(input("ingrese divisor:"))
  dividendo = int(input("ingrese dividendo:"))
  print(division(divisor,dividendo))
except ZeroDivisionError:
  print("No es posible dividir por cero (0)")
except ValueError:
  print("Número ingresado no válido")
except:
  print("Error al dividir")
else:
  print("La división se pudo llevar a cabo")
finally:
  print("gracias por usar la función division()")
```

## Capturando el tipo de excepción

Cuando no estamos seguros sobre el tipo de excepción que puede arrojar un código, podemos capturar la excepción en una variable y a partir de ella acceder a su tipo. Miremos:

```
def division (x, y):
  return x/y

try:
  divisor = int(input("ingrese divisor:"))
  dividendo =int(input("ingrese dividendo:"))
  print(division(divisor,dividendo))
except Exception as captura_excepcion:
  print("Error ", captura_excepcion)
```

## Lanzamiento manual de excepciones

Con la instrucción **raise** podemos lanzar manualmente una excepción en nuestro código. Veamos, ingresa una edad negativa en el siguiente programa

```
edad = int (input("Ingrese su edad:"))
if edad < 0:
  raise ValueError("La edad no puede ser negativa")
```

Ahora, tomando el código anterior y gestionando esta excepción

```
try:
  edad = int (input("Ingrese su edad:"))
  if edad < 0:
    raise ValueError("La edad no puede ser negativa")
except Exception as e:
  print("Error ", e)
```

## Apropiación

1. Finaliza el siguiente código con una gestión completa de sus excepciones. El objetivo del programa es solicitar al usuario una posición de una lista y devolver el valor que se encuentra en dicha posición. Prueba el programa enviando una posición inexistente y enviando una letra como posición. El programa sólo terminará cuando se ejecute correctamente

```
lista = [3,5,6,8]
pos = int(input("ingrese la posición del elemento que desea obtener:"))
print(f"El valor en la posicion {pos} es {lista[pos]}")
```

2. Realiza una función llamada agregar_una_vez(lista, elemento) que reciba una lista y un elemento. La función debe añadir el elemento al final de la lista con la condición de no repetir ningún elemento. Además si este elemento ya se encuentra en la lista se debe invocar un error de tipo ValueError que debes capturar y mostrar este mensaje en su lugar:

```
Error: Imposible añadir elementos duplicados => [elemento].
```

Mostrar el siguiente mensaje siempre que se ejecute independientemente de que la ejecución haya sido exitosa o no

```
Gracias por usar este programa
```