PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL

PYTHON

Primer método para evitar errores en la ejecución

VALIDACIÓN PRELIMINAR DE DATOS

```
first_number = int(input("Ingresa el primer número: "))
second_number = int(input("Ingresa el segundo número: "))

if second_number != 0:
    print(first_number / second_number)
else:
    print("Esta operación no puede ser realizada.")

print("FIN.")
```

MEJOR MANEJAR UN ERROR CUANDO OCURRE, QUE EVITARLO

```
try:
    print(first_number / second_number)
except:
    print("Esta operación no puede ser realizada.")
```

```
try:
    # código riesgoso
except:
    # código para manejar la excepción
```

El código riesgoso se ejecuta sin problema => No se ejecuta el código del bloque **except:**

El código riesgoso intenta hacer algo **erróneo** => Se detiene el programa Se genera una **excepción**

Se ejecuta el bloque **except** => Se proporciona una reacción adecuada a la excepción

Se termina la sección **try-except**

PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL

PYTHON

```
try:
    # codigo riesgoso
except ZeroDivisionError:
    # manejo de la excepcion
    print("No puedes dividir entre cero, lo siento.")
except ValueError:
    print("Debes ingresar un valor entero...")
except (IndexError, TypeError):
    # manejo conjunto de varias excepciones
    print("Operación inapropiada ")
except:
    # manejo por defecto de excepciones
    print("Oh cielos, algo salió mal...")
```

Al menos una ***except***

Cada excepción cae en la **primera coincidencia**

Tantas except como se necesiten, pero cada ***excepción solo se puede especificar una vez***

- Los bloques ***except*** se analizan según el orden de aparición en el código.
- El bloque ***except sin nombre*** SIEMPRE es el último

ZeroDivisionError → división entre cero.

ValueError → valor inaceptable en un argumento

```
>>> int("cadena")
```

TypeError → operación o función sobre objeto inapropiado

```
>>> lista[0.5]
```

AttributeError → invoca a propiedad o método de un elemento que no existe.

```
>>> lista.dapend(2)
```

IndexError → Indice de un iterable fuera de rango

SintaxError → error de sintaxis. Aparece en tiempo de compilación.

Keyboardinterrupt → usuario pulsa Ctrl + C

PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL

PYTHON

Bloques opcionales en el tratamiento de excepciones

el bloque **else** SIEMPRE después del último bloque **except**:

el bloque **finally** SIEMPRE es el último bloque

```
def inversa(n):
    try:
        n = 1 / n
    except ZeroDivisionError:
        print("División fallida")
        n = None
    else:
        print("Todo salió bien")
    finally:
        print("Es momento de decir adiós")
        return n

print(inversa(2))

print(inversa(0))
```

```
Todo salió bien
Es momento de decir adiós
0.5
División fallida
```

Se ejecuta el bloque **else** siempre y cuando no se haya generado ninguna excepción

el bloque **finally** se ejecuta SIEMPRE, haya ocurrido o no excepción

El identificador **e** que sigue a la palabra reservada **as**, captura información de la excepción con el fin de analizar su naturaleza y sacar conclusiones adecuadas.

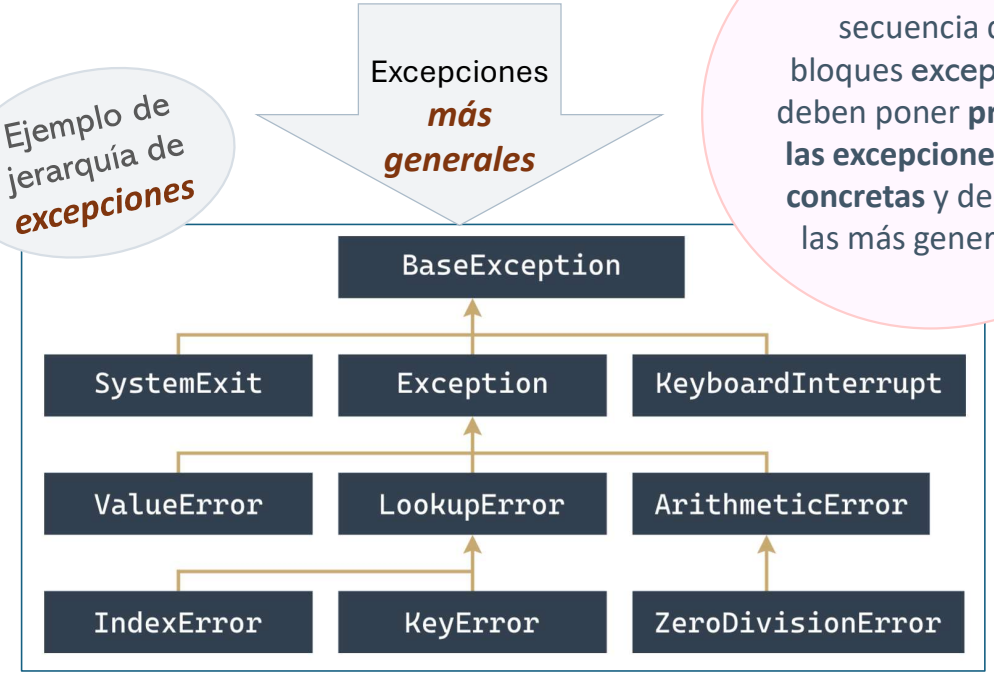
```
try:
    i = int("¡Hola!")
except Exception as e:
    print(e)
```

PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL

PYTHON

Ejemplo de jerarquía de ***excepciones***

Excepciones ***más generales***



Excepciones ***más concretas***

Dentro de la misma secuencia de bloques **except** , se deben poner **primero las excepciones más concretas** y después las más generales.

raise NombreDeExcepción
genera una excepción bajo demanda.

raise NombreDeExcepción (“error”)
genera una excepción con la información “error”

raise
sin NombreDeExcepción, solo se puede usar dentro del bloque except, y genera la misma excepción que se está manejando actualmente.

```
def dividir(a, b):
    if b == 0:
        # Lanzamos una excepción manualmente
        raise ValueError("No se puede dividir entre cero")
    return a / b
try:
    dividir(10, 0)
except ValueError as e:
    print("Se ha capturado una excepción ValueError:")
    print("Mensaje:", e)
```

assert expression

Evalúa la expresión y si es igual a False, a cero, a None o una cadena vacía genera la excepción AssertionError.
Puede usarse para proteger partes críticas del código de datos devastadores.

```
def procesar_edad(edad):
    # Verificamos que la edad sea un número positivo
    assert edad > 0, "La edad debe ser mayor que cero"
    return f"Edad válida: {edad}"

try:
    print(procesar_edad(25))   # Esto funciona
    print(procesar_edad(-3))   # Esto generará un AssertionError
except AssertionError as e:
    print("Se capturó un AssertionError:")
    print("Mensaje de error:", e)
```

```
def calcular_tangente(angle):
    assert angle % 180 != 90
    ....

def probar_angulo(angle):
    try:
        resultado = calcular_tangente(angle)
    except AssertionError:
        print(f"Error con el ángulo {angle}°: tangente  indefinida")

# Ejemplos
probar_angulo(30)   # Válido
probar_angulo(90)   # Provoca AssertionError
probar_angulo(270)  # También provoca AssertionError
```