

Manejo de Excepciones en POO (Python)

Alumno: Puma Huanca Anthony Rusbel

Docente: Ing. Coyla Idme Leonel

Lenguajes de Programación II – FINESI

Universidad Nacional del Altiplano

Facultad de Ingeniería Estadística e Informática

Introducción

El **manejo de excepciones** en la Programación Orientada a Objetos (POO) con Python es un mecanismo que permite identificar y gestionar errores en tiempo de ejecución. Utilizando bloques como `try`, `except`, `else` y `finally`, se puede interrumpir el flujo normal del programa y ejecutar código específico para tratar errores de forma controlada, garantizando la estabilidad y confiabilidad del sistema.

Ejercicio 1 División entre cero

```
1 class Division:
2     def __init__(self, a, b):
3         self.a = a
4         self.b = b
5
6     def dividir(self):
7         try:
8             resultado = self.a / self.b
9             return resultado
10        except ZeroDivisionError:
11            return "Error: No se puede dividir entre cero."
12        except Exception as e:
13            return f"Ocurrió un error: {e}"
14        finally:
15            print("Proceso de división concluido.")
16
17 valor_a = 10
18 valor_b = 0
19 operacion = Division(valor_a, valor_b)
20 print(f"Resultado de la operación: {operacion.dividir()}")
```

Listing 1: Clase Division con manejo de excepciones

Ejecución:

Proceso de división concluido.

Resultado de la operación: Error: No se puede dividir entre cero.

Ejercicio 2 Cálculo de la hipotenusa

```
1 import math
2
3 def calcular_hipotenusa(cateto_a, cateto_b):
4     return math.sqrt(cateto_a**2 + cateto_b**2)
5
6 def main():
7     try:
8         a = float(input("Ingrese el valor de A:"))
9         b = float(input("Ingrese el valor de B:"))
10        if a<=0 or b<=0:
11            raise ValueError("Los catetos deben ser números positivos")
12
13        hipotenusa = calcular_hipotenusa(a,b)
14        print(f"La hipotenusa es: {hipotenusa:.2f}")
15
16    except ValueError as ve:
17        print("Error",ve)
18
19    except Exception as e:
20        print("Ocurrió un problema inesperado",e)
21
22 if __name__ == "__main__":
23     main()
```

Listing 2: Validación de entradas numéricas positivas

Ejecución (entrada inválida):

Entrada 1:

```
Ingrese el valor de A:3
Ingrese el valor de B:4
La hipotenusa es: 5.000000
```

Entrada 2:

```
Ingrese el valor de A:cinco
Error invalid literal for float(): cinco
```

Entrada 3:

```
Ingrese el valor de A:3
Ingrese el valor de B:-4
Error Los catetos deben ser números positivos
```

Ejercicio 3 Fibonacci con manejo de excepciones

```
1 def generar_fibonacci(n):
2
3     if n <= 0:
4         print("La cantidad de términos debe ser un número
5             positivo mayor que cero.")
5         return
```

```

6
7     a = 0
8     b = 1
9
10    print("Serie de Fibonacci:")
11
12    print(a)
13
14    if n > 1:
15        print(b)
16
17    for i in range(3, n + 1):
18        siguiente_termino = a + b
19        print(siguiente_termino)
20
21        a = b
22        b = siguiente_termino
23
24
25 def main():
26     try:
27         n_terminos = float(input("Ingrese la cantidad de
28         términos de Fibonacci a generar: "))
29
30         if n_terminos != int(n_terminos) or n_terminos <= 0:
31             raise ValueError("Debe ingresar un número entero
32             positivo (ej: 10).")
33
34         generar_fibonacci(int(n_terminos))
35
36     except ValueError as ve:
37         print("Error:", ve)
38
39     except Exception as e:
40         print("Ocurrió un problema inesperado:", e)
41
42 if __name__ == "__main__":
43     main()

```

Listing 3: Generación controlada de la secuencia de Fibonacci

Ejecución (entrada inválida):

Entrada 1:

```

Ingresar la cantidad de términos de Fibonacci a generar: 5
Serie de Fibonacci:
0
1
1
2
3

```

Entrada 2:

Ingrese la cantidad de términos de Fibonacci a generar: -5

Error: Debe ingresar un número entero positivo (ej: 10)

Entrada 3:

Ingrese la cantidad de términos de Fibonacci a generar: letras

Error: could not convert string to float: 'letras'