

CLASE 14

Guia de Estudio: Excepciones I

Cuando las cosas salen mal... y como manejarlo

Contenido:

1. Que es una excepcion
2. Sintaxis try/except/else/finally
3. Excepciones comunes en Python
4. Cheat Sheet de referencia rapida
5. Conexion con el proyecto de tareas
6. Ejercicios adicionales (8 ejercicios)
7. Soluciones completas

1. Que es una Excepcion?

Una **excepcion** es un error que ocurre **durante la ejecucion** del programa. No es un error de sintaxis (que Python detecta antes de ejecutar), sino algo que sucede mientras el programa esta corriendo.

Analogia: Imagina que alguien te lanza una pelota. Si la atrapas, todo bien. Si no la atrapas, te pega en la cara. Las excepciones son como pelotas que Python te lanza cuando algo sale mal. Con try/except, te pones el guante para atraparlas.

El problema sin manejo de excepciones:

```
# Sin try/except - el programa MUERE
edad = int(input("Tu edad: ")) # Si escriben "hola" -> CRASH
resultado = 10 / 0 # Division entre cero -> CRASH
archivo = open("no_existe.json") # Archivo inexistente -> CRASH
```

Flujo de una excepcion:

1. Python detecta un problema
2. "Lanza" una excepcion
3. Si nadie la atrapa: el programa MUERE
4. Si alguien la atrapa (con try/except): el programa SIGUE

2. Sintaxis try/except/else/finally

2.1 try/except basico

```
try:
    #Codigo que PODRIA fallar
    resultado = 10 / 0
except:
    #Que hacer SI falla
    print("Algo salio mal!")
```

2.2 except con tipo especifico (RECOMENDADO)

```
try:
    numero = int(input("Escribe un numero: "))
except ValueError:
    print("Eso no es un numero valido")
```

REGLA: Siempre especifica el tipo de excepcion. except: solo (sin tipo) atrapa TODO, incluyendo bugs que deberias arreglar.

2.3 Multiples except

```
def dividir_numeros():
    try:
        a = int(input("Primer numero: "))
        b = int(input("Segundo numero: "))
        resultado = a / b
        print(f"Resultado: {resultado}")
    except ValueError:
        print("Debes escribir un numero")
    except ZeroDivisionError:
        print("No puedes dividir entre cero")
```

2.4 Capturar el mensaje del error con 'as e'

```

try:
    numero = int("hola")
except ValueError as e:
    print(f"Error: {e}")
# Output: Error: invalid literal for int() with base 10: 'hola'

```

2.5 finally - Se ejecuta SIEMPRE

```

try:
    archivo = open("datos.txt", "r")
    contenido = archivo.read()
except FileNotFoundError:
    print("Archivo no encontrado")
finally:
    print("Esto se ejecuta SIEMPRE")
    # Cerrar conexiones, limpiar recursos, etc.

```

Nota: with open() ya hace el "finally" automáticamente. Por eso usamos with para archivos.

2.6 else - Solo si NO hubo error

```

try:
    numero = int(input("Numero: "))
except ValueError:
    print("No es un numero")
else:
    print(f"Perfecto, escribiste {numero}")
finally:
    print("Fin del intento")

```

Estructura completa:

```

try:      -> Código que puede fallar
except:   -> Si HUBO error
else:     -> Si NO hubo error (opcional)
finally:  -> SIEMPRE (opcional)

```

3. Excepciones Comunes en Python

Excepcion	Cuando ocurre	Ejemplo
ValueError	Valor incorrecto	<code>int("hola")</code>
TypeError	Tipo incorrecto	<code>"hola" + 5</code>
KeyError	Clave no existe en dict	<code>mi_dict["clave_falsa"]</code>
IndexError	Indice fuera de rango	<code>lista[99]</code> (con 3 items)
FileNotFoundError	Archivo no existe	<code>open("no_existe.json")</code>
ZeroDivisionError	Division entre cero	<code>10 / 0</code>
AttributeError	Atributo no existe	<code>"texto".metodo_falso()</code>
NameError	Variable no definida	<code>print(variable_inexistente)</code>

Consejo: No necesitas memorizar esta tabla. El NOMBRE del error te dice que paso: ValueError = el VALOR no sirve, KeyError = la CLAVE no existe, etc.

4. Cheat Sheet - Referencia Rapida

Patron: while + try/except + break

Pedir datos hasta que sean validos:

```
while True:
    try:
        numero = int(input("Escribe un numero: "))
        break # Si llego aqui, es valido -> salir del loop
    except ValueError:
        print("Eso no es un numero. Intenta de nuevo.")

print(f"El doble es: {numero * 2}")
```

Funcion reutilizable: pedir_opcion()

```
def pedir_opcion(mensaje="Opcion: ", minimo=0, maximo=5):
    """Pide un numero validando tipo y rango."""
    while True:
        try:
            opcion = int(input(mensaje))
            if minimo <= opcion <= maximo:
                return opcion
            else:
                print(f"Debe ser entre {minimo} y {maximo}")
        except ValueError:
            print("Escribe un numero valido")

# Uso:
opcion = pedir_opcion("Elige (1-5): ", minimo=1, maximo=5)
```

Cargar JSON de forma segura

```
import json

def cargar_tareas(archivo="tareas.json"):
    """Carga tareas manejando errores."""
    try:
        with open(archivo, "r") as f:
            return json.load(f)
    except FileNotFoundError:
        print("Primera vez. Creando archivo nuevo.")
        return []
    except json.JSONDecodeError:
        print("Archivo corrupto. Iniciando vacio.")
        return []
```

Buenas Practicas

HACER:

- Siempre especificar el tipo de excepcion
- Usar "as e" para obtener detalles del error
- Manejar cada tipo de error de forma diferente
- try solo alrededor del codigo que puede fallar
- Retornar valores por defecto cuando tenga sentido

EVITAR:

- except: solo (sin tipo) -> atrapa TODO, esconde bugs
- try: con 50 lineas -> dificil saber que fallo
- Usar excepciones para controlar flujo normal
- Ignorar errores: except: pass (NUNCA hacer esto)

REGLA DE ORO: "Atrapa solo los errores que SABES manejar."

5. Conexion con el Proyecto de Tareas

En la Clase 13 creamos un sistema de gestion de tareas con archivos JSON. Ahora podemos hacerlo mas robusto con excepciones.

Mejoras a implementar:

1. **cargar_tareas():** Manejar FileNotFoundError y JSONDecodeError
2. **Menu principal:** Validar opciones con try/except ValueError
3. **Agregar tarea:** Validar entrada del usuario
4. **Editar tarea:** Manejar indices invalidos con IndexError

Menu robusto completo:

```
import json

def cargar_tareas(archivo="tareas.json"):
    try:
        with open(archivo, "r") as f:
            return json.load(f)
    except FileNotFoundError:
        return []
    except json.JSONDecodeError:
        return []

def guardar_tareas(tareas, archivo="tareas.json"):
    with open(archivo, "w") as f:
        json.dump(tareas, f, indent=2)

def pedir_opcion(mensaje, minimo, maximo):
    while True:
        try:
            opcion = int(input(mensaje))
            if minimo <= opcion <= maximo:
                return opcion
            print(f"Debe ser entre {minimo} y {maximo}")
        except ValueError:
            print("Escribe un numero valido")

def main():
    tareas = cargar_tareas()
    while True:
        print("\n--- MENU ---")
        print("1. Ver tareas")
        print("2. Agregar tarea")
        print("0. Salir")

        opcion = pedir_opcion("Opcion: ", 0, 2)

        if opcion == 0:
            guardar_tareas(tareas)
            print("Hasta luego!")
            break
        elif opcion == 1:
            # Mostrar tareas
            pass
        elif opcion == 2:
            # Agregar tarea
            pass
```

6. Ejercicios Adicionales

NIVEL PRINCIPIANTE

Ejercicio P1: Divisor Seguro

Crea una funcion **dividir(a, b)** que:

- Retorne el resultado de a / b
- Si b es cero, retorne None y muestre un mensaje
- Use try/except ZeroDivisionError

```
# Ejemplo de uso:
print(dividir(10, 2))    # 5.0
print(dividir(10, 0))    # "No se puede dividir entre cero" -> None
```

Ejercicio P2: Conversor de Edad

Crea un programa que:

- Pida la edad al usuario con input()
- Valide que sea un numero (ValueError)
- Valide que este entre 0 y 150
- Siga pidiendo hasta que sea valido

Ejercicio P3: Acceso Seguro a Diccionario

Crea una funcion **obtener_valor(diccionario, clave, valor_defecto=None)** que:

- Retorne el valor de la clave si existe
- Si la clave no existe (KeyError), retorne el valor_defecto
- Muestre un mensaje indicando que uso el valor por defecto

```
# Ejemplo de uso:
datos = {"nombre": "Ana", "edad": 25}
print(obtener_valor(datos, "nombre"))    # "Ana"
print(obtener_valor(datos, "telefono", "N/A")) # "N/A"
```

NIVEL INTERMEDIO

Ejercicio I1: Lector de Archivos Multiples

Crea una funcion **leer_archivos(lista_nombres)** que:

- Reciba una lista de nombres de archivos
- Intente abrir cada uno
- Retorne un diccionario: {nombre: contenido} o {nombre: "Error: mensaje"}
- Maneje FileNotFoundError por cada archivo

```
# Ejemplo de uso:
resultado = leer_archivos(["datos.txt", "no_existe.txt", "config.json"])
# {"datos.txt": "contenido...",
#  "no_existe.txt": "Error: archivo no encontrado",
#  "config.json": "..."}

```

Ejercicio I2: Conversor de Lista

Crea una funcion **convertir_lista(lista_strings)** que:

- Reciba una lista de strings como ["1", "2", "hola", "3"]
- Intente convertir cada uno a numero
- Retorne solo los numeros validos: [1, 2, 3]
- Maneje ValueError por cada elemento

Ejercicio I3: Mini Agenda

Crea un programa de agenda con menu (agregar contacto, buscar, listar) que:

- Maneje ValueError al pedir opciones del menu
- Maneje KeyError al buscar contactos que no existen
- Guarde los contactos en un diccionario

NIVEL AVANZADO

Ejercicio A1: Procesador de CSV Manual

Crea una funcion **procesar_csv(nombre_archivo)** que:

- Lea un archivo CSV linea por linea
- Maneje FileNotFoundError si no existe
- Maneje ValueError en conversiones de numeros
- Maneje IndexError si faltan columnas
- Retorne: (datos_limpios, lista_errores)

```
# Archivo productos.csv:
# nombre,precio,cantidad
# Laptop,999.99,10
# Mouse,invalido,5
# Teclado,49.99

# Uso:
datos, errores = procesar_csv("productos.csv")
# datos: [{"nombre": "Laptop", "precio": 999.99, "cantidad": 10}, ...]
# errores: ["Linea 3: precio invalido"]
```

Ejercicio A2: Sistema de Notas Robusto

Crea un sistema completo de calificaciones que:

- Pida notas (validar 0-100) con manejo de ValueError
- Calcule promedio
- Guarde en JSON con manejo de errores
- Cargue al iniciar manejando FileNotFoundError y JSONDecodeError
- Permita agregar, eliminar y ver notas

7. Soluciones Completas

Solucion P1: Divisor Seguro

```
def dividir(a, b):
    """Divide a entre b de forma segura."""
    try:
        return a / b
    except ZeroDivisionError:
        print("No se puede dividir entre cero")
        return None

# Pruebas
print(dividir(10, 2))    # 5.0
print(dividir(10, 0))    # None
print(dividir(15, 3))    # 5.0
```

Solucion P2: Conversor de Edad

```
def pedir_edad():
    """Pide edad validando tipo y rango."""
    while True:
        try:
            edad = int(input("Tu edad: "))
            if 0 <= edad <= 150:
                return edad
            else:
                print("La edad debe estar entre 0 y 150")
        except ValueError:
            print("Debes escribir un numero")

edad = pedir_edad()
print(f"Tu edad es: {edad}")
```

Solucion P3: Acceso Seguro a Diccionario

```
def obtener_valor(diccionario, clave, valor_defecto=None):
    """Obtiene valor de diccionario de forma segura."""
    try:
        return diccionario[clave]
    except KeyError:
        print(f"Clave '{clave}' no encontrada. Usando valor defecto.")
        return valor_defecto

# Pruebas
datos = {"nombre": "Ana", "edad": 25}
print(obtener_valor(datos, "nombre"))    # "Ana"
print(obtener_valor(datos, "telefono", "N/A")) # "N/A"
print(obtener_valor(datos, "ciudad", "Desconocida")) # "Desconocida"
```

Solucion I1: Lector de Archivos Multiples

```
def leer_archivos(lista_nombres):
    """Lee multiples archivos manejando errores."""
    resultado = {}
    for nombre in lista_nombres:
        try:
            with open(nombre, "r") as f:
                resultado[nombre] = f.read()
        except FileNotFoundError:
            resultado[nombre] = "Error: archivo no encontrado"
        except PermissionError:
            resultado[nombre] = "Error: sin permisos de lectura"
    return resultado

# Pruebas
archivos = ["datos.txt", "no_existe.txt"]
resultado = leer_archivos(archivos)
for nombre, contenido in resultado.items():
    print(f"{nombre}: {contenido[:50]}...")
```

Solucion I2: Conversor de Lista

```
def convertir_lista(lista_strings):
    """Convierte lista de strings a numeros, ignorando invalidos."""
    numeros_validos = []
    for item in lista_strings:
        try:
            numero = float(item)
            numeros_validos.append(numero)
        except ValueError:
            print(f"Ignorando valor invalido: '{item}'")
    return numeros_validos

# Pruebas
datos = ["1", "2.5", "hola", "3", "abc", "4.0"]
resultado = convertir_lista(datos)
print(f"Numeros validos: {resultado}") # [1.0, 2.5, 3.0, 4.0]
```

Solucion I3: Mini Agenda

```
def mini_agenda():
    contactos = {}

    def pedir_opcion():
        while True:
            try:
                return int(input("Opcion: "))
            except ValueError:
                print("Escribe un numero")

    while True:
        print("\n--- AGENDA ---")
        print("1. Agregar contacto")
        print("2. Buscar contacto")
        print("3. Listar todos")
        print("0. Salir")

        opcion = pedir_opcion()

        if opcion == 0:
            break
        elif opcion == 1:
            nombre = input("Nombre: ")
            telefono = input("Telefono: ")
            contactos[nombre] = telefono
            print("Contacto agregado!")
        elif opcion == 2:
            nombre = input("Buscar nombre: ")
            try:
                print(f"Telefono: {contactos[nombre]}")
```

```
        except KeyError:
            print("Contacto no encontrado")
    elif opcion == 3:
        if contactos:
            for n, t in contactos.items():
                print(f"  {n}: {t}")
        else:
            print("Agenda vacia")
mini_agenda()
```

Solucion A1: Procesador de CSV Manual

```
def procesar_csv(nombre_archivo):
    """Procesa CSV manejando todos los errores posibles."""
    datos_limpios = []
    errores = []

    try:
        with open(nombre_archivo, "r") as f:
            lineas = f.readlines()
    except FileNotFoundError:
        return [], [f"Archivo '{nombre_archivo}' no encontrado"]

    # Primera linea es encabezado
    if len(lineas) < 2:
        return [], ["Archivo vacio o sin datos"]

    encabezados = lineas[0].strip().split(",")

    for i, linea in enumerate(lineas[1:], start=2):
        try:
            valores = linea.strip().split(",")

            if len(valores) != len(encabezados):
                errores.append(f"Linea {i}: columnas incorrectas")
                continue

            registro = {}
            for j, encabezado in enumerate(encabezados):
                valor = valores[j]
                # Intentar convertir a numero si es posible
                try:
                    if "." in valor:
                        registro[encabezado] = float(valor)
                    else:
                        registro[encabezado] = int(valor)
                except ValueError:
                    registro[encabezado] = valor

            datos_limpios.append(registro)

        except IndexError:
            errores.append(f"Linea {i}: error de indice")

    return datos_limpios, errores

# Prueba
datos, errores = procesar_csv("productos.csv")
print("Datos:", datos)
print("Errores:", errores)
```

Solucion A2: Sistema de Notas Robusto

```
import json

class SistemaNotas:
    def __init__(self, archivo="notas.json"):
        self.archivo = archivo
        self.notas = self.cargar()

    def cargar(self):
        try:
            with open(self.archivo, "r") as f:
                return json.load(f)
        except FileNotFoundError:
            print("Primera vez. Creando archivo nuevo.")
            return []
        except json.JSONDecodeError:
            print("Archivo corrupto. Iniciando vacio.")
            return []

    def guardar(self):
        try:
            with open(self.archivo, "w") as f:
                json.dump(self.notas, f, indent=2)
        except PermissionError:
            print("Error: sin permisos para guardar")

    def pedir_nota(self):
        while True:
            try:
                nota = float(input("Nota (0-100): "))
                if 0 <= nota <= 100:
                    return nota
                print("Debe estar entre 0 y 100")
            except ValueError:
                print("Escribe un numero valido")

    def agregar(self):
        nombre = input("Nombre del estudiante: ")
        nota = self.pedir_nota()
        self.notas.append({"nombre": nombre, "nota": nota})
        self.guardar()
        print("Nota agregada!")

    def promedio(self):
        if not self.notas:
            print("No hay notas")
            return
        total = sum(n["nota"] for n in self.notas)
        prom = total / len(self.notas)
        print(f"Promedio: {prom:.2f}")

    def listar(self):
        if not self.notas:
            print("No hay notas")
            return
        for n in self.notas:
            print(f" {n['nombre']}: {n['nota']}")

    def menu(self):
        while True:
            print("\n--- SISTEMA DE NOTAS ---")
            print("1. Agregar nota")
            print("2. Ver promedio")
            print("3. Listar notas")
            print("0. Salir")

            try:
                opcion = int(input("Opcion: "))
            except ValueError:
                print("Escribe un numero")
                continue

            if opcion == 0:
                break
```

```
        elif opcion == 1:
            self.agregar()
        elif opcion == 2:
            self.promedio()
        elif opcion == 3:
            self.listar()

# Ejecutar
sistema = SistemaNotas()
sistema.menu()
```

Recuerda: La practica hace al maestro. Intenta resolver los ejercicios ANTES de ver las soluciones. El manejo de excepciones es fundamental para crear programas robustos y profesionales.