CLASE 15

Guia de Estudio

Excepciones II + Debugging

raise | Excepciones personalizadas | assert | PEP 8

Python Fundamentals - Material de estudio

1. RAISE - Lanzar excepciones

En la Clase 14 aprendimos a **atrapar** excepciones con try/except. Ahora aprenderemos a **lanzar** excepciones nosotros mismos con la palabra clave **raise**.

1.1 Por que lanzar excepciones?

Cuando escribimos funciones, necesitamos proteger nuestro codigo de datos invalidos. Si alguien pasa una edad negativa o un nombre vacio, nuestro codigo deberia **rechazar** esos datos en lugar de crear registros basura.

> *Las excepciones son CONTRATOS: si no cumples las reglas, el codigo se niega a continuar.*

1.2 Sintaxis basica de raise

```
raise TipoDeError("Mensaje descriptivo")
```

Ejemplo:

```
def dividir(a, b):
    if b == 0:
        raise ValueError("No puedes dividir entre cero")
    return a / b

print(dividir(10, 2))   # 5.0
print(dividir(10, 0))   # ValueError: No puedes dividir entre cero
```

1.3 Raise para validar datos

```
def registrar_usuario(nombre, edad):
    """Registra un usuario con validaciones."""

    # Validar nombre
    if not isinstance(nombre, str):
        raise TypeError("El nombre debe ser string")
    if len(nombre.strip()) == 0:
        raise ValueError("El nombre no puede estar vacio")

    # Validar edad
    if not isinstance(edad, int):
        raise TypeError("La edad debe ser un numero entero")
    if edad < 0 or edad > 150:
        raise ValueError("La edad debe estar entre 0 y 150")

    return {"nombre": nombre.strip(), "edad": edad}
```

TypeError = tipo de dato incorrecto (str en vez de int)
ValueError = valor incorrecto (dentro del tipo pero invalido, ej: edad=-5)

1.4 Re-raise: propagar excepciones

A veces quieres atrapar una excepcion, hacer algo (como logging), y volver a lanzarla para que alguien mas la maneje:

```
def procesar_archivo(nombre):
    try:
        with open(nombre) as f:
            return f.read()
    except FileNotFoundError:
        print(f"Log: No se encontro {nombre}")
        raise  # Vuelve a lanzar la misma excepcion
```

> ***raise*** *solo (sin tipo de error) vuelve a lanzar la excepcion actual. Muy usado para logging.*

2. Excepciones Personalizadas

Podemos crear nuestros propios tipos de excepcion heredando de **Exception** o de otra excepcion existente.

2.1 Sintaxis basica

```
class MiError(Exception):
    """Descripcion de cuando usar este error."""
    pass
```

2.2 Jerarquia de excepciones

```
class TareaError(Exception):
    """Error base para el sistema de tareas."""
    pass

class TituloVacioError(TareaError):
    """El titulo esta vacio."""
    pass

class PrioridadInvalidaError(TareaError):
    """Prioridad no valida."""
    pass
```

Al crear una jerarquia, puedes atrapar todos los errores del sistema con **except TareaError**, o ser especifico con cada subtipo.

2.3 Uso completo

```
def crear_tarea(titulo, prioridad, categoria):
    if not titulo or len(titulo.strip()) == 0:
        raise TituloVacioError("El titulo no puede estar vacio")
    if prioridad not in ["alta", "media", "baja"]:
        raise PrioridadInvalidaError(f"'{prioridad}' no es valida")
    return {"titulo": titulo.strip(), "prioridad": prioridad}

# Manejo especifico:
try:
    tarea = crear_tarea("", "alta", "estudio")
except TituloVacioError:
    print("Escribe un titulo")
except PrioridadInvalidaError:
    print("Usa: alta, media o baja")
except TareaError:
    print("Error general en tarea")
```

> ***Orden de except:*** *siempre de MAS especifico a MENOS especifico. Si pones TareaError primero, nunca llegara a los especificos.*

2.4 Cuando usar excepciones personalizadas?

Usar ValueError/TypeError	Usar excepcion personalizada
Errores genericos	Errores de tu dominio
Scripts pequenos	Proyectos grandes
Una sola funcion	Multiples modulos
No diferenciar errores	Manejar cada error diferente

> *En duda, empieza con ValueError. Crea excepciones propias cuando las necesites.*

3. Debugging - Encontrar y arreglar bugs

Un **bug** es un error en la logica del codigo. No necesariamente causa un crash, pero produce resultados incorrectos.

3.1 Print Debugging

La tecnica mas basica: agregar prints temporales para ver los valores de las variables en cada paso:

```
def buscar_palabra(texto, palabra):
    palabras = texto.split()
    print(f"DEBUG: palabras = {palabras}")
    print(f"DEBUG: buscando = '{palabra}'")

    for i in range(len(palabras)):
        print(f"DEBUG: comparando '{palabras[i]}' con '{palabra}'")
        if palabras[i] == palabra:
            return i
    return -1
```

> *IMPORTANTE: Despues de encontrar el bug, BORRA los prints de debug.*

3.2 Assert - Validar suposiciones

```
assert condicion, "Mensaje si falla"
```

```
def calcular_promedio(notas):
    assert len(notas) > 0, "Lista no puede estar vacia"
    assert all(0 <= n <= 100 for n in notas), "Notas entre 0-100"
    return sum(notas) / len(notas)
```

raise	assert
Errores que PUEDEN ocurrir	Errores que NUNCA deberian ocurrir
Validar datos de usuario	Detectar bugs del programador
Siempre activo	Se desactiva con python -O

3.3 Debugger de VS Code

Pasos para usar el debugger:

1. Click en el margen izquierdo del editor para crear un **breakpoint** (punto rojo)

2. Presionar **F5** o ir a Run > Start Debugging

3. El codigo se pausa en el breakpoint

4. Ver valores de variables en el panel izquierdo

5. Usar **F10** (Step Over) para avanzar linea por linea

Tecla	Accion
`F5`	Continuar hasta siguiente breakpoint
`F10`	Step Over - avanzar una linea
`F11`	Step Into - entrar a una funcion
`Shift+F5`	Detener debugging

4. PEP 8 - Guia de estilo de Python

PEP 8 (Python Enhancement Proposal #8) es la guia oficial de estilo. Seguirla hace tu codigo mas legible y profesional.

4.1 Reglas principales

Regla	Mal	Bien
Indentacion: 4 espacios	`if x:\n  print()`	`if x:\n    print()`
Lineas <= 79 chars	`Linea muy larga...`	`Dividir en varias`
2 lineas entre funciones	`def a():\ndef b():`	`def a():\n\n\ndef b():`
Espacios en operadores	`x=1+2`	`x = 1 + 2`
Sin espacio antes de :	`def f() :`	`def f():`
Imports arriba	`Import en funcion`	`Imports al inicio`

4.2 Convenciones de nombres

Elemento	Convencion	Ejemplo
Variables	**snake_case**	`mi_variable, total`
Funciones	**snake_case**	`calcular_promedio()`
Constantes	**MAYUSCULAS**	`MAX_INTENTOS, PI`
Clases	**PascalCase**	`MiClase, TareaError`
Modulos	**snake_case**	`mi_modulo.py`

4.3 Ejemplo: Antes y despues

Antes (codigo feo):

```
def f(l):
    t=0
    for i in l:
        if i>0:t+=i
    return t/len(l) if len(l)>0 else 0
```

Despues (codigo limpio):

```
def calcular_promedio_positivos(numeros):
    """Calcula promedio de numeros positivos."""
    if not numeros:
        return 0

    suma_positivos = 0
```

```
    for numero in numeros:
        if numero > 0:
            suma_positivos += numero

    return suma_positivos / len(numeros)
```

5. CHEAT SHEET - Referencia rapida

raise

```
# Lanzar excepcion
raise ValueError("Mensaje")
raise TypeError("Mensaje")

# Re-raise (propagar)
except SomeError:
    print("Log")
    raise  # Vuelve a lanzar
```

Excepciones personalizadas

```
class MiError(Exception):
    pass

class ErrorEspecifico(MiError):
    pass

# Usar:
raise ErrorEspecifico("mensaje")

# Atrapar:
except ErrorEspecifico:  # Especifico
except MiError:          # Generico (atrapa todos)
```

assert vs raise

```
# ASSERT - bugs del programador (nunca deberian pasar)
assert len(lista) > 0, "Lista vacia"
assert isinstance(x, int), "Debe ser int"

# RAISE - errores de produccion (pueden pasar)
if edad < 0:
    raise ValueError("Edad invalida")
```

PEP 8 - Lo esencial

```
# Indentacion: 4 espacios
# Lineas: max 79 caracteres
# 2 lineas en blanco entre funciones
# Espacios: x = 1 + 2 (NO x=1+2)
#
# Nombres:
#   variables/funciones: snake_case
#   constantes: MAYUSCULAS
#   clases: PascalCase
```

```
#
# Imports al inicio, ordenados:
#   1. Libreria estandar
#   2. Terceros
#   3. Propios
```

6. Ejercicios Adicionales

NIVEL PRINCIPIANTE

Ejercicio 1: Validar edad

Crea una funcion **validar_edad(edad)** que reciba un valor y:
- Lance **TypeError** si no es un entero
- Lance **ValueError** si es menor a 0 o mayor a 150
- Retorne la edad si es valida

```
# Ejemplo de uso:
print(validar_edad(25))     # 25
print(validar_edad(-5))     # ValueError
print(validar_edad("abc"))  # TypeError
```

Ejercicio 2: Validar email basico

Crea una funcion **validar_email(email)** que:
- Lance ValueError si el email no contiene '@'
- Lance ValueError si el email no contiene '.' despues del '@'
- Lance ValueError si esta vacio
- Retorne el email en minusculas si es valido

```
print(validar_email("Ana@mail.com"))  # ana@mail.com
print(validar_email("invalido"))      # ValueError
```

Ejercicio 3: Validar contrasena

Crea una funcion **validar_contrasena(password)** que lance ValueError si:
- Tiene menos de 8 caracteres
- No contiene al menos un numero
- No contiene al menos una mayuscula

Retorne True si es valida.

NIVEL INTERMEDIO

Ejercicio 4: Sistema de login

Crea las excepciones **UsuarioNoExisteError** y **ContrasenaIncorrectaError** (ambas heredando de LoginError). Implementa una funcion **login(usuario, contrasena)** que valide contra un diccionario de usuarios.

```
USUARIOS = {
    "admin": "Admin123",
    "user1": "Pass456",
}
```

```
# login("admin", "Admin123")  -> "Bienvenido admin"
# login("noexiste", "x")     -> UsuarioNoExisteError
```

```
# login("admin", "mala")     -> ContrasenaIncorrectaError
```

Ejercicio 5: Calculadora con validaciones

Crea una funcion **calcular(a, operador, b)** que:
- Valide que a y b sean numeros (TypeError si no)
- Valide que operador sea +, -, *, / (ValueError si no)
- Lance ValueError si se divide entre cero
- Retorne el resultado de la operacion

Ejercicio 6: Refactorizar codigo complejo

Refactoriza el siguiente codigo aplicando PEP 8:

```
def pn(L,N):
    R=[]
    for X in L:
        if X['n']==N:R.append(X)
    return R

def ap(L):
    T=0
    for X in L:
        T+=X['p']
    if len(L)>0:return T/len(L)
    else:return 0

def mx(L):
    M=L[0]
    for X in L:
        if X['p']>M['p']:M=X
    return M
```

NIVEL AVANZADO

Ejercicio 7: Sistema bancario con excepciones

Crea una jerarquia de excepciones:
- **CuentaError** (base)
- **SaldoInsuficienteError** (hereda de CuentaError)
- **LimiteExcedidoError** (hereda de CuentaError)
- **CuentaNoExisteError** (hereda de CuentaError)

Implementa funciones: **depositar(cuenta, monto)**, **retirar(cuenta, monto)**, **transferir(origen, destino, monto)**. Limite de transferencia: $10,000.

```
cuentas = {
    "001": {"titular": "Ana", "saldo": 5000},
    "002": {"titular": "Luis", "saldo": 3000},
}
```

```
# transferir("001", "002", 2000)  -> OK
# transferir("001", "002", 6000)  -> SaldoInsuficienteError
# transferir("001", "002", 15000) -> LimiteExcedidoError
# transferir("999", "002", 100)   -> CuentaNoExisteError
```

Ejercicio 8: Validador de datos CSV

Crea un sistema que lea datos de una lista de diccionarios (simulando CSV) y valide cada registro. Debe:
- Crear excepciones: CampoVacioError, TipoInvalidoError, ValorFueraDeRangoError
- Validar campos: nombre (str no vacio), edad (int 0-150), email (contiene @)
- NO detenerse en el primer error: recolectar TODOS los errores
- Retornar un reporte con registros validos y lista de errores

```
datos = [
    {"nombre": "Ana", "edad": 25, "email": "ana@mail.com"},
    {"nombre": "", "edad": 25, "email": "x@mail.com"},
    {"nombre": "Luis", "edad": -5, "email": "luis@mail.com"},
    {"nombre": "Pedro", "edad": 30, "email": "invalido"},
]
```

```
reporte = validar_datos(datos)
# reporte["validos"] -> [registro de Ana]
# reporte["errores"] -> [lista de errores encontrados]
```

7. SOLUCIONES

Solucion Ejercicio 1: Validar edad

```
def validar_edad(edad):
    if not isinstance(edad, int):
        raise TypeError("La edad debe ser un numero entero")
    if edad < 0 or edad > 150:
        raise ValueError("La edad debe estar entre 0 y 150")
    return edad

# Pruebas
print(validar_edad(25))   # 25
try:
    validar_edad(-5)
except ValueError as e:
    print(f"Error: {e}")  # Error: La edad debe estar entre 0 y 150
try:
    validar_edad("abc")
except TypeError as e:
    print(f"Error: {e}")  # Error: La edad debe ser un numero entero
```

Solucion Ejercicio 2: Validar email

```
def validar_email(email):
    if not email or len(email.strip()) == 0:
        raise ValueError("El email no puede estar vacio")
    if "@" not in email:
        raise ValueError("El email debe contener '@'")
    partes = email.split("@")
    if "." not in partes[1]:
        raise ValueError("Debe haber '.' despues del '@'")
    return email.lower().strip()

print(validar_email("Ana@Mail.COM"))  # ana@mail.com
try:
    validar_email("invalido")
except ValueError as e:
    print(f"Error: {e}")
```

Solucion Ejercicio 3: Validar contrasena

```
def validar_contrasena(password):
    if len(password) < 8:
```

```
        raise ValueError("Minimo 8 caracteres")
    if not any(c.isdigit() for c in password):
        raise ValueError("Debe tener al menos un numero")
    if not any(c.isupper() for c in password):
        raise ValueError("Debe tener al menos una mayuscula")
    return True

print(validar_contrasena("MiPass123"))  # True
try:
    validar_contrasena("corta")
except ValueError as e:
    print(f"Error: {e}")
```

Solucion Ejercicio 4: Sistema de login

```
class LoginError(Exception):
    pass

class UsuarioNoExisteError(LoginError):
    pass

class ContrasenaIncorrectaError(LoginError):
    pass

USUARIOS = {"admin": "Admin123", "user1": "Pass456"}

def login(usuario, contrasena):
    if usuario not in USUARIOS:
        raise UsuarioNoExisteError(f"'{usuario}' no existe")
    if USUARIOS[usuario] != contrasena:
        raise ContrasenaIncorrectaError("Contrasena incorrecta")
    return f"Bienvenido {usuario}"

try:
    print(login("admin", "Admin123"))   # Bienvenido admin
except LoginError as e:
    print(f"Error: {e}")

try:
    login("ghost", "x")
except UsuarioNoExisteError as e:
    print(f"Error: {e}")  # 'ghost' no existe
```

Solucion Ejercicio 5: Calculadora

```
def calcular(a, operador, b):
    if not isinstance(a, (int, float)):
        raise TypeError(f"'{a}' no es un numero")
    if not isinstance(b, (int, float)):
        raise TypeError(f"'{b}' no es un numero")
    if operador not in ["+", "-", "*", "/"]:
        raise ValueError(f"Operador '{operador}' no valido")
    if operador == "/" and b == 0:
        raise ValueError("No se puede dividir entre cero")

    operaciones = {
        "+": a + b,
        "-": a - b,
        "*": a * b,
        "/": a / b,
```

```
    }
    return operaciones[operador]

print(calcular(10, "+", 5))   # 15
print(calcular(10, "/", 3))   # 3.333...
```

Solucion Ejercicio 6: Refactorizar

```
def buscar_por_nombre(productos, nombre):
    """Busca productos que coincidan con el nombre."""
    resultados = []
    for producto in productos:
        if producto['nombre'] == nombre:
            resultados.append(producto)
    return resultados

def calcular_precio_promedio(productos):
    """Calcula el precio promedio de una lista de productos."""
    if not productos:
        return 0
    total = 0
    for producto in productos:
        total += producto['precio']
    return total / len(productos)

def obtener_mas_caro(productos):
    """Retorna el producto con el precio mas alto."""
    mas_caro = productos[0]
    for producto in productos:
        if producto['precio'] > mas_caro['precio']:
            mas_caro = producto
    return mas_caro
```

Solucion Ejercicio 7: Sistema bancario

```
class CuentaError(Exception):
    pass

class SaldoInsuficienteError(CuentaError):
    pass

class LimiteExcedidoError(CuentaError):
    pass

class CuentaNoExisteError(CuentaError):
    pass

LIMITE_TRANSFERENCIA = 10000
cuentas = {
    "001": {"titular": "Ana", "saldo": 5000},
    "002": {"titular": "Luis", "saldo": 3000},
}

def obtener_cuenta(numero):
    if numero not in cuentas:
        raise CuentaNoExisteError(f"Cuenta '{numero}' no existe")
    return cuentas[numero]

def depositar(numero, monto):
    if monto <= 0:
        raise ValueError("El monto debe ser positivo")
    cuenta = obtener_cuenta(numero)
    cuenta["saldo"] += monto
    return cuenta["saldo"]

def retirar(numero, monto):
    if monto <= 0:
        raise ValueError("El monto debe ser positivo")
    cuenta = obtener_cuenta(numero)
    if monto > cuenta["saldo"]:
        raise SaldoInsuficienteError(
            f"Saldo insuficiente. Disponible: {cuenta['saldo']}")
    cuenta["saldo"] -= monto
    return cuenta["saldo"]

def transferir(origen, destino, monto):
    if monto > LIMITE_TRANSFERENCIA:
        raise LimiteExcedidoError(
            f"Limite de ${LIMITE_TRANSFERENCIA} excedido")
    retirar(origen, monto)
```

```
    depositar(destino, monto)
    return True

# Prueba
try:
    transferir("001", "002", 2000)
    print("Transferencia exitosa")
    print(f"Cuenta 001: {cuentas['001']['saldo']}")
    print(f"Cuenta 002: {cuentas['002']['saldo']}")
except CuentaError as e:
    print(f"Error: {e}")
```

Solucion Ejercicio 8: Validador CSV

```
class ValidacionError(Exception):
    pass

class CampoVacioError(ValidacionError):
    pass

class TipoInvalidoError(ValidacionError):
    pass

class ValorFueraDeRangoError(ValidacionError):
    pass

def validar_registro(registro, indice):
    errores = []
    # Validar nombre
    nombre = registro.get("nombre", "")
    if not nombre or len(str(nombre).strip()) == 0:
        errores.append(f"Fila {indice}: nombre vacio")
    # Validar edad
    edad = registro.get("edad")
    if not isinstance(edad, int):
        errores.append(f"Fila {indice}: edad no es entero")
    elif edad < 0 or edad > 150:
        errores.append(f"Fila {indice}: edad fuera de rango")
    # Validar email
    email = registro.get("email", "")
    if "@" not in str(email):
        errores.append(f"Fila {indice}: email sin @")
    return errores

def validar_datos(datos):
    reporte = {"validos": [], "errores": []}
    for i, registro in enumerate(datos):
        errores = validar_registro(registro, i + 1)
        if errores:
            reporte["errores"].extend(errores)
        else:
            reporte["validos"].append(registro)
    return reporte

# Prueba
datos = [
    {"nombre": "Ana", "edad": 25, "email": "ana@mail.com"},
```

```
    {"nombre": "", "edad": 25, "email": "x@mail.com"},
    {"nombre": "Luis", "edad": -5, "email": "luis@mail.com"},
    {"nombre": "Pedro", "edad": 30, "email": "invalido"},
]
reporte = validar_datos(datos)
print(f"Validos: {len(reporte['validos'])}")
print(f"Errores: {len(reporte['errores'])}")
for error in reporte["errores"]:
    print(f"  - {error}")
```