# Tema 2: Estructuras de Control en Python

## 1. Estructuras de Selección

Las estructuras de selección permiten que un programa tome decisiones y ejecute diferentes bloques de código según las condiciones especificadas.

### 1.1 Estructura if

La estructura `if` ejecuta un bloque de código solo si la condición es verdadera.

```
# Estructura if básica
edad = 18

# if simple
if edad >= 18:
    print("Eres mayor de edad")

# if con múltiples condiciones
nota = 7.5
asistencia = True

if nota >= 5.0 and asistencia:
    print("Has aprobado la asignatura")
    print(f"Tu nota es: {nota}")

# if anidados
hora = 14

if hora >= 6:
    if hora < 12:
        print("Buenos días")

# if con operadores diversos
nombre = "Carlos"
lista_vip = ["Ana", "Carlos", "Diana"]

if nombre in lista_vip:
    print(f"{nombre} está en la lista VIP")

# if con valores truthy/falsy
texto = "Hola"
numero = 0
lista_vacia = []

if texto:  # String no vacío es True
    print("El texto tiene contenido")

if not numero:  # 0 es False
    print("El número es cero")

if not lista_vacia:  # Lista vacía es False
    print("La lista está vacía")
```

## 1.2 Estructura if-else

La estructura `if-else` permite ejecutar un bloque de código si la condición es verdadera y otro diferente si es falsa.

```
# if-else simple
edad = int(input("Introduce tu edad: "))

if edad >= 18:
    print("Puedes votar")
    print("Puedes conducir")
else:
    print("Eres menor de edad")
    print("No puedes votar ni conducir")

# if-elif-else encadenados
nota = float(input("Introduce tu nota (0-10): "))

if nota < 0 or nota > 10:
    print("Nota inválida")
elif nota < 5:
    print("Suspenso")
elif nota < 6:
    print("Aprobado")
elif nota < 7:
    print("Bien")
elif nota < 9:
    print("Notable")
else:  # nota <= 10
    print("Sobresaliente")

# Ejemplo práctico: Calculadora de IMC
peso = float(input("Introduce tu peso (kg): "))
altura = float(input("Introduce tu altura (m): "))

imc = peso / (altura ** 2)
print(f"Tu IMC es: {imc:.2f} - ", end="")

if imc < 18.5:
    print("Bajo peso")
elif imc < 25:
    print("Peso normal")
elif imc < 30:
    print("Sobrepeso")
else:
    print("Obesidad")

# if-else con múltiples condiciones complejas
dia_semana = "sábado"
hora_actual = 14
tienda_especial = True

if (dia_semana in ["sábado", "domingo"] or
    hora_actual >= 20 or
    tienda_especial):
    descuento = 0.20
    print(f"Descuento especial del {descuento*100:.0f}%")
else:
    descuento = 0.05
    print(f"Descuento estándar del {descuento*100:.0f}%")
```

## 1.3 Operador Condicional (Expresión Ternaria)

Python tiene una forma compacta de escribir expresiones if-else simples.

```
# Sintaxis: valor_si_verdadero if condicion else valor_si_falso

# Ejemplo básico
edad = 20
mensaje = "Mayor de edad" if edad >= 18 else "Menor de edad"
print(mensaje)

# Asignación de valores
a, b = 10, 20
mayor = a if a > b else b
print(f"El mayor es: {mayor}")

# En impresiones directas
nota = 6.5
print(f"Estado: {'Aprobado' if nota >= 5 else 'Suspenso'}")

# Operadores ternarios anidados (usar con precaución)
numero = 0
tipo = "Positivo" if numero > 0 else ("Negativo" if numero < 0 else
"Cero")
print(f"El número es: {tipo}")

# En comprensiones de lista
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
pares_impares = ["Par" if n % 2 == 0 else "Impar" for n in numeros]
print(pares_impares)

# Uso en funciones
def verificar_par(num):
    return True if num % 2 == 0 else False

es_par = verificar_par(8)
print(f"8 es {'par' if es_par else 'impar'}")

# Ejemplo práctico: Descuento
precio = 100.0
es_vip = True
precio_final = precio * (0.8 if es_vip else 1.0)  # 20% descuento si
es VIP
print(f"Precio final: {precio_final:.2f}€")

# Con funciones lambda
obtener_estado = lambda edad: "Adulto" if edad >= 18 else "Menor"
print(obtener_estado(25))
print(obtener_estado(15))
```

## 1.4 Estructura match-case (Python 3.10+)

Python 3.10 introdujo `match-case`, similar al switch de otros lenguajes pero más poderoso.

```
# match-case básico
dia = int(input("Introduce un día (1-7): "))

match dia:
```

```
    case 1:
        print("Lunes")
    case 2:
        print("Martes")
    case 3:
        print("Miércoles")
    case 4:
        print("Jueves")
    case 5:
        print("Viernes")
    case 6:
        print("Sábado")
    case 7:
        print("Domingo")
    case _:  # Caso por defecto
        print("Día inválido")

# match con múltiples valores
mes = int(input("Introduce un mes (1-12): "))

match mes:
    case 1 | 3 | 5 | 7 | 8 | 10 | 12:
        print("Este mes tiene 31 días")
    case 4 | 6 | 9 | 11:
        print("Este mes tiene 30 días")
    case 2:
        print("Este mes tiene 28 o 29 días")
    case _:
        print("Mes inválido")

# match con guardias (condiciones adicionales)
nota = float(input("Introduce una nota: "))

match nota:
    case n if n < 0 or n > 10:
        print("Nota inválida")
    case n if n < 5:
        print("Suspenso")
    case n if n < 7:
        print("Aprobado")
    case n if n < 9:
        print("Notable")
    case _:
        print("Sobresaliente")

# match con patrones estructurales
punto = (2, 3)  # Tupla representando coordenadas

match punto:
    case (0, 0):
        print("Origen")
    case (x, 0):
        print(f"En el eje X: {x}")
    case (0, y):
        print(f"En el eje Y: {y}")
    case (x, y):
        print(f"Punto en ({x}, {y})")

# match con tipos de datos
def procesar_dato(dato):
    match dato:
```

```
        case int(n) if n > 0:
            return f"Entero positivo: {n}"
        case int(n) if n < 0:
            return f"Entero negativo: {n}"
        case float(f):
            return f"Número decimal: {f}"
        case str(s):
            return f"Cadena de texto: {s}"
        case list(l):
            return f"Lista con {len(l)} elementos"
        case _:
            return "Tipo de dato no reconocido"

# Pruebas
print(procesar_dato(42))
print(procesar_dato(-10))
print(procesar_dato(3.14))
print(procesar_dato("Hola"))
print(procesar_dato([1, 2, 3]))
```

### Alternativa a match-case para versiones anteriores

```
# Diccionario como alternativa a switch/match
def dia_semana(dia):
    dias = {
        1: "Lunes",
        2: "Martes",
        3: "Miércoles",
        4: "Jueves",
        5: "Viernes",
        6: "Sábado",
        7: "Domingo"
    }
    return dias.get(dia, "Día inválido")

print(dia_semana(3))
print(dia_semana(10))

# Con funciones
def suma(a, b):
    return a + b

def resta(a, b):
    return a - b

def multiplicar(a, b):
    return a * b

def dividir(a, b):
    return a / b if b != 0 else "Error: División por cero"

operaciones = {
    '+': suma,
    '-': resta,
    '*': multiplicar,
    '/': dividir
}

operacion = input("Introduce una operación (+, -, *, /): ")
num1, num2 = 10, 5
```

```
if operacion in operaciones:
    resultado = operaciones[operacion](num1, num2)
    print(f"Resultado: {resultado}")
else:
    print("Operación no válida")
```

# 2. Estructuras de Repetición

Las estructuras de repetición (bucles) permiten ejecutar un bloque de código múltiples veces.

## 2.1 Bucle while

El bucle `while` ejecuta el código mientras la condición sea verdadera.

```
# while básico - contador
contador = 1
print("Contando del 1 al 5:")

while contador <= 5:
    print(f"Contador: {contador}")
    contador += 1

# while con condición compleja
suma = 0
print("Introduce números (0 para terminar):")
numero = int(input())

while numero != 0:
    suma += numero
    print(f"Suma actual: {suma}")
    numero = int(input())

print(f"Suma total: {suma}")

# Validación de entrada con while
edad = -1

while edad < 0 or edad > 120:
    try:
        edad = int(input("Introduce una edad válida (0-120): "))
        if edad < 0 or edad > 120:
            print("Edad inválida. Inténtalo de nuevo.")
    except ValueError:
        print("Por favor, introduce un número.")
        edad = -1

print(f"Edad registrada: {edad}")

# Bucle infinito con break
while True:
    respuesta = input("¿Quieres continuar? (s/n): ").lower()
    if respuesta == 'n':
        print("Saliendo...")
        break
    elif respuesta == 's':
        print("Continuando...")
    else:
```

```
        print("Respuesta no válida")

# Ejemplo práctico: Adivinar número
import random

numero_secreto = random.randint(1, 100)
intentos = 0
adivinado = False

print("¡Adivina el número (1-100)!")

while not adivinado:
    try:
        intento = int(input("Tu intento: "))
        intentos += 1

        if intento < numero_secreto:
            print("Demasiado bajo. Intenta de nuevo:")
        elif intento > numero_secreto:
            print("Demasiado alto. Intenta de nuevo:")
        else:
            adivinado = True
            print(f"¡Correcto! Lo adivinaste en {intentos} intentos")
    except ValueError:
        print("Por favor, introduce un número válido")

# while con else (característica única de Python)
contador = 0
while contador < 5:
    print(f"Contador: {contador}")
    contador += 1
else:  # Se ejecuta cuando la condición es False (no por break)
    print("Bucle completado normalmente")
```

## 2.2 Simulación de do-while en Python

Python no tiene bucle do-while nativo, pero se puede simular.

```
# Simulación de do-while con while True y break
# El código se ejecuta al menos una vez

# Menú con simulación de do-while
while True:
    print("\n=== MENÚ PRINCIPAL ===")
    print("1. Saludar")
    print("2. Mostrar fecha")
    print("3. Contar hasta 10")
    print("0. Salir")

    try:
        opcion = int(input("Elige una opción: "))

        if opcion == 1:
            print("¡Hola! ¿Cómo estás?")
        elif opcion == 2:
            from datetime import datetime
            print(f"Fecha actual: {datetime.now().strftime('%Y-%m-%d
%H:%M:%S')}")
        elif opcion == 3:
            for i in range(1, 11):
```

```
                print(i, end=" ")
            print()
        elif opcion == 0:
            print("¡Hasta luego!")
            break
        else:
            print("Opción no válida")
    except ValueError:
        print("Por favor, introduce un número")

# Validación con simulación de do-while
PASSWORD_CORRECTA = "python2024"
MAX_INTENTOS = 3
intentos = 0

while True:
    password = input("Introduce la contraseña: ")
    intentos += 1

    if password == PASSWORD_CORRECTA:
        print("Acceso concedido")
        break

    if intentos >= MAX_INTENTOS:
        print("Acceso denegado. Máximo de intentos alcanzado")
        break

    print(f"Contraseña incorrecta. Te quedan {MAX_INTENTOS - intentos}
intentos")

# Ejemplo práctico: Calculadora con repetición
def calculadora_simple():
    while True:
        try:
            num1 = float(input("\nPrimer número: "))
            num2 = float(input("Segundo número: "))

            print(f"Suma: {num1 + num2}")
            print(f"Resta: {num1 - num2}")
            print(f"Multiplicación: {num1 * num2}")
            if num2 != 0:
                print(f"División: {num1 / num2}")
            else:
                print("División: No se puede dividir por cero")

        except ValueError:
            print("Error: Introduce números válidos")

        continuar = input("\n¿Deseas hacer otro cálculo? (s/n):
").lower()
        if continuar != 's':
            break

calculadora_simple()
```

## 2.3 Bucle for

El bucle `for` en Python es muy versátil y se usa principalmente para iterar sobre secuencias.

```python
# for con range
print("Números del 1 al 10:")
for i in range(1, 11):
    print(i, end=" ")
print()

# for descendente
print("\nCuenta atrás:")
for i in range(10, -1, -1):
    print(i)

# for con incremento diferente
print("\nNúmeros pares del 0 al 20:")
for i in range(0, 21, 2):
    print(i, end=" ")
print()

# for con enumerate (índice y valor)
frutas = ["manzana", "naranja", "plátano"]
print("\nFrutas con índices:")
for indice, fruta in enumerate(frutas):
    print(f"{indice}: {fruta}")

# for con zip (múltiples listas)
nombres = ["Ana", "Luis", "María"]
edades = [25, 30, 28]
ciudades = ["Madrid", "Barcelona", "Valencia"]

print("\nDatos combinados:")
for nombre, edad, ciudad in zip(nombres, edades, ciudades):
    print(f"{nombre} tiene {edad} años y vive en {ciudad}")

# for anidados - Tabla de multiplicar
print("\nTabla de multiplicar del 1 al 5:")
for i in range(1, 6):
    print(f"\nTabla del {i}:")
    for j in range(1, 11):
        print(f"{i} x {j} = {i * j}")

# for con listas
numeros = [10, 20, 30, 40, 50]

print("\nRecorriendo lista:")
for numero in numeros:
    print(f"Valor: {numero}")

# List comprehension (comprensión de listas)
cuadrados = [x**2 for x in range(1, 11)]
print(f"\nCuadrados: {cuadrados}")

pares = [x for x in range(1, 21) if x % 2 == 0]
print(f"Números pares: {pares}")

# for con diccionarios
estudiante = {
    "nombre": "Carlos",
    "edad": 20,
    "curso": "Python",
    "nota": 8.5
}
```

```
print("\nDatos del estudiante:")
for clave, valor in estudiante.items():
    print(f"{clave}: {valor}")

# for con strings
palabra = "Python"
print("\nLetras de la palabra:")
for letra in palabra:
    print(letra, end=" ")
print()

# Ejemplo práctico: Factorial
n = 5
factorial = 1

for i in range(1, n + 1):
    factorial *= i

print(f"\nFactorial de {n} = {factorial}")

# Ejemplo: Números primos
print("\nNúmeros primos del 2 al 50:")
for num in range(2, 51):
    es_primo = True

    for divisor in range(2, int(num ** 0.5) + 1):
        if num % divisor == 0:
            es_primo = False
            break

    if es_primo:
        print(num, end=" ")
print()

# for con else (característica única de Python)
print("\nBuscando número divisible por 7:")
for i in range(1, 10):
    if i % 7 == 0:
        print(f"Encontrado: {i}")
        break
else:  # Se ejecuta si el bucle termina sin break
    print("No se encontró ningún número divisible por 7")
```

## Itertools - Herramientas avanzadas de iteración

```
import itertools

# cycle - Repetir una secuencia infinitamente
colores = itertools.cycle(['rojo', 'verde', 'azul'])
print("Primeros 7 colores del ciclo:")
for i, color in enumerate(colores):
    if i >= 7:
        break
    print(color, end=" ")
print()

# count - Contador infinito
print("\nContador con step:")
contador = itertools.count(start=10, step=2)
for i, valor in enumerate(contador):
    if i >= 5:
```

```
        break
    print(valor, end=" ")
print()

# product - Producto cartesiano
letras = ['A', 'B']
numeros = [1, 2]
print("\nProducto cartesiano:")
for combinacion in itertools.product(letras, numeros):
    print(combinacion, end=" ")
print()

# combinations - Combinaciones sin repetición
elementos = ['a', 'b', 'c']
print("\nCombinaciones de 2 elementos:")
for combo in itertools.combinations(elementos, 2):
    print(combo, end=" ")
print()
```

# 3. Sentencias de Salto

## 3.1 Sentencia break

La sentencia `break` termina la ejecución del bucle más interno.

```
# break en bucle for
print("Buscando el primer múltiplo de 7:")
for i in range(1, 101):
    if i % 7 == 0:
        print(f"Encontrado: {i}")
        break  # Sale del bucle

# break en while
print("\nIntroduce números (999 para salir):")
while True:  # Bucle infinito
    numero = int(input())
    if numero == 999:
        print("Saliendo del bucle...")
        break
    print(f"Has introducido: {numero}")

# break en bucles anidados
print("\nBuscando en matriz:")
matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

buscar = 5
encontrado = False

for i, fila in enumerate(matriz):
    for j, valor in enumerate(fila):
        print(f"Verificando posición [{i}][{j}]")
        if valor == buscar:
            print(f"¡Encontrado {buscar} en [{i}][{j}]!")
            encontrado = True
            break  # Solo sale del bucle interno
```

```
    if encontrado:
        break  # Sale del bucle externo

# Uso de función para salir de bucles anidados
def buscar_en_matriz(matriz, objetivo):
    for i, fila in enumerate(matriz):
        for j, valor in enumerate(fila):
            if valor == objetivo:
                return i, j  # Return actúa como break en ambos bucles
    return None

resultado = buscar_en_matriz(matriz, 8)
if resultado:
    print(f"Encontrado en posición: {resultado}")

# Ejemplo práctico: Validación de entrada
while True:
    try:
        edad = int(input("\nIntroduce tu edad (0-120): "))
        if 0 <= edad <= 120:
            print(f"Edad válida: {edad}")
            break
        print("Edad fuera de rango")
    except ValueError:
        print("Por favor, introduce un número válido")

# break con comprensión de generadores
def primer_par(numeros):
    return next((n for n in numeros if n % 2 == 0), None)

lista = [1, 3, 5, 6, 7, 8]
print(f"\nPrimer número par: {primer_par(lista)}")
```

## 3.2 Sentencia continue

La sentencia `continue` salta a la siguiente iteración del bucle.

```
# continue en for - Números impares
print("Números impares del 1 al 10:")
for i in range(1, 11):
    if i % 2 == 0:
        continue  # Salta los números pares
    print(i, end=" ")
print()

# continue en while
print("\nNúmeros no divisibles por 3:")
contador = 0
while contador < 15:
    contador += 1
    if contador % 3 == 0:
        continue  # Salta los múltiplos de 3
    print(contador, end=" ")
print()

# continue con múltiples condiciones
print("\n\nProcesando valores:")
for i in range(-5, 11):
    if i == 0:
        print("Saltando división por cero")
```

```
        continue
    if i < 0:
        print(f"Saltando número negativo: {i}")
        continue
    resultado = 100.0 / i
    print(f"100 / {i} = {resultado:.2f}")

# continue en bucles anidados
print("\nTabla de multiplicar (saltando diagonal):")
for i in range(1, 6):
    for j in range(1, 6):
        if i == j:
            print("   X", end="")
            continue  # Salta cuando i == j
        print(f"{i*j:4d}", end="")
    print()

# Ejemplo práctico: Suma de números positivos
numeros = [10, -5, 20, -3, 15, 0, -8, 25]
suma = 0

print("\nSumando solo números positivos:")
for num in numeros:
    if num <= 0:
        print(f"Saltando: {num}")
        continue
    suma += num
    print(f"Sumando: {num} (Total: {suma})")

print(f"Suma final: {suma}")

# continue con try-except
print("\nProcesando lista de strings como números:")
datos = ["10", "20", "abc", "30", "xyz", "40"]

total = 0
for dato in datos:
    try:
        numero = int(dato)
        total += numero
        print(f"Sumado: {numero}")
    except ValueError:
        print(f"Saltando valor no numérico: {dato}")
        continue

print(f"Total: {total}")

# Filtrado con continue vs comprensión
print("\nFiltrando con continue:")
palabras = ["Python", "es", "un", "lenguaje", "de", "programación"]
for palabra in palabras:
    if len(palabra) < 3:
        continue
    print(palabra, end=" ")
print()

# Equivalente con comprensión (más pythónico)
palabras_largas = [p for p in palabras if len(p) >= 3]
print(f"Con comprensión: {palabras_largas}")
```

# 4. Manejo de Excepciones

Las excepciones son errores que ocurren durante la ejecución y alteran el flujo normal del programa.

## 4.1 Captura de Excepciones (try-except)

```
# try-except básico
try:
    resultado = 10 / 0
    print(f"Resultado: {resultado}")
except ZeroDivisionError:
    print("Error: División por cero")

# try-except con información del error
try:
    numero = int("abc")
except ValueError as e:
    print(f"Error: {e}")
    print(f"Tipo de error: {type(e).__name__}")

# try-except con múltiples excepciones
try:
    numero = int(input("Introduce un número: "))
    lista = [1, 2, 3]
    indice = int(input("Introduce un índice (0-2): "))

    resultado = lista[indice] / numero
    print(f"Resultado: {resultado}")

except ValueError:
    print("Error: Debes introducir un número entero")
except IndexError:
    print("Error: Índice fuera de rango")
except ZeroDivisionError:
    print("Error: División por cero")

# try-except-else-finally
try:
    archivo = open("datos.txt", "r")
    contenido = archivo.read()
except FileNotFoundError:
    print("Error: Archivo no encontrado")
else:
    print("Archivo leído correctamente")
    print(f"Contenido: {contenido}")
finally:
    print("Limpieza: Cerrando recursos...")
    if 'archivo' in locals() and not archivo.closed:
        archivo.close()

# Captura de múltiples excepciones en un solo except
try:
    entrada = input("Introduce un número para dividir 100: ")
    divisor = int(entrada)
    resultado = 100 / divisor
    print(f"Resultado: {resultado}")
except (ValueError, ZeroDivisionError) as e:
    print(f"Error: {e}")
```

```
    print(f"Tipo de error: {type(e).__name__}")

# Captura genérica con Exception
try:
    # Código que puede lanzar cualquier excepción
    import random
    opciones = [
        lambda: 1/0,
        lambda: int("abc"),
        lambda: [1,2,3][10],
        lambda: {"key": "value"}["noexiste"]
    ]
    random.choice(opciones)()

except Exception as e:
    print(f"Error general: {e}")
    print(f"Tipo: {type(e).__name__}")

    # Información detallada del error
    import traceback
    print("\nDetalle del error:")
    traceback.print_exc()

# try-except anidados
try:
    try:
        numero = int(input("\nIntroduce un número: "))
        resultado = 100 / numero
    except ValueError:
        print("No es un número, usando valor por defecto (1)")
        resultado = 100 / 1

    print(f"Resultado: {resultado}")

except ZeroDivisionError:
    print("Error crítico: División por cero")

# Gestión de errores en bucles
numeros = ["10", "20", "abc", "0", "30"]
resultados = []

for dato in numeros:
    try:
        num = int(dato)
        resultado = 100 / num
        resultados.append(resultado)
        print(f"100/{num} = {resultado}")
    except ValueError:
        print(f"'{dato}' no es un número válido")
    except ZeroDivisionError:
        print(f"No se puede dividir por cero")

print(f"\nResultados calculados: {resultados}")
```

## 4.2 Lanzamiento de Excepciones (raise)

```
# raise básico
def dividir(a, b):
    if b == 0:
        raise ZeroDivisionError("División por cero no permitida")
```

```
    return a / b

try:
    resultado = dividir(10, 0)
except ZeroDivisionError as e:
    print(f"Error capturado: {e}")

# raise con validación
def calcular_raiz_cuadrada(numero):
    if not isinstance(numero, (int, float)):
        raise TypeError(f"Se esperaba un número, se recibió
{type(numero).__name__}")
    if numero < 0:
        raise ValueError(f"No se puede calcular la raíz de un número
negativo: {numero}")
    return numero ** 0.5

# Pruebas
try:
    print(calcular_raiz_cuadrada(16))      # 4.0
    print(calcular_raiz_cuadrada(-4))      # Error
except ValueError as e:
    print(f"Error: {e}")

try:
    print(calcular_raiz_cuadrada("16"))    # Error
except TypeError as e:
    print(f"Error: {e}")

# Re-lanzar excepciones
def procesar_archivo(nombre):
    try:
        with open(nombre, 'r') as archivo:
            return archivo.read()
    except FileNotFoundError:
        print(f"Advertencia: No se encontró {nombre}")
        raise  # Re-lanza la excepción original

try:
    contenido = procesar_archivo("inexistente.txt")
except FileNotFoundError:
    print("Error manejado en el nivel superior")

# raise from - Encadenar excepciones
def convertir_a_numero(texto):
    try:
        return int(texto)
    except ValueError as e:
        raise TypeError(f"No se pudo convertir '{texto}' a número")
from e

try:
    numero = convertir_a_numero("abc")
except TypeError as e:
    print(f"Error: {e}")
    print(f"Causado por: {e.__cause__}")

# Validación con assert (para debugging)
def calcular_descuento(precio, porcentaje):
    assert precio >= 0, "El precio no puede ser negativo"
```

```
    assert 0 <= porcentaje <= 100, "El porcentaje debe estar entre 0 y
100"

    return precio * (1 - porcentaje / 100)

# Con assert activado (modo debug)
try:
    print(calcular_descuento(100, 20))   # 80.0
    print(calcular_descuento(-50, 10))   # AssertionError
except AssertionError as e:
    print(f"Error de validación: {e}")
```

## 4.3 Excepciones Personalizadas

```
# Definición de excepciones personalizadas
class EdadInvalidaError(Exception):
    """Excepción para edad fuera de rango válido"""
    def __init__(self, edad, mensaje="Edad fuera de rango"):
        self.edad = edad
        self.mensaje = mensaje
        super().__init__(self.mensaje)

    def __str__(self):
        return f"{self.mensaje}. Edad proporcionada: {self.edad}"

class SaldoInsuficienteError(Exception):
    """Excepción para operaciones bancarias sin saldo suficiente"""
    def __init__(self, saldo_actual, cantidad_solicitada):
        self.saldo_actual = saldo_actual
        self.cantidad_solicitada = cantidad_solicitada
        mensaje = f"Saldo insuficiente. Saldo: {saldo_actual:.2f},
Solicitado: {cantidad_solicitada:.2f}"
        super().__init__(mensaje)

class PasswordDebilidadError(Exception):
    """Excepción para contraseñas que no cumplen requisitos"""
    def __init__(self, razon):
        self.razon = razon
        super().__init__(f"Contraseña débil: {razon}")

# Clase que usa las excepciones personalizadas
class CuentaBancaria:
    def __init__(self, titular, saldo_inicial=0):
        self.titular = titular
        self.__saldo = saldo_inicial  # Atributo privado

    @property
    def saldo(self):
        return self.__saldo

    def depositar(self, cantidad):
        if cantidad <= 0:
            raise ValueError("La cantidad debe ser positiva")
        self.__saldo += cantidad
        print(f"Depósito exitoso. Nuevo saldo: {self.__saldo:.2f}")

    def retirar(self, cantidad):
        if cantidad <= 0:
            raise ValueError("La cantidad debe ser positiva")
        if cantidad > self.__saldo:
```

```
            raise SaldoInsuficienteError(self.__saldo, cantidad)
        self.__saldo -= cantidad
        print(f"Retiro exitoso. Nuevo saldo: {self.__saldo:.2f}")

    def transferir(self, cantidad, cuenta_destino):
        try:
            self.retirar(cantidad)
            cuenta_destino.depositar(cantidad)
            print(f"Transferencia exitosa de {cantidad:.2f} a
{cuenta_destino.titular}")
        except SaldoInsuficienteError:
            print("Transferencia cancelada: Saldo insuficiente")
            raise

# Funciones de validación
def validar_edad(edad):
    if not isinstance(edad, int):
        raise TypeError("La edad debe ser un número entero")
    if edad < 0:
        raise EdadInvalidaError(edad, "La edad no puede ser negativa")
    if edad > 150:
        raise EdadInvalidaError(edad, "La edad no puede ser mayor a
150")
    return True

def validar_password(password):
    errores = []

    if len(password) < 8:
        errores.append("Debe tener al menos 8 caracteres")
    if not any(c.isdigit() for c in password):
        errores.append("Debe contener al menos un número")
    if not any(c.isupper() for c in password):
        errores.append("Debe contener al menos una mayúscula")
    if not any(c in "!@#$%^&*()" for c in password):
        errores.append("Debe contener al menos un carácter especial")

    if errores:
        raise PasswordDebilidadError(", ".join(errores))

    return True

# Uso de las excepciones personalizadas
print("=== VALIDACIÓN DE EDAD ===")
edades_prueba = [25, -5, 200, "veinte"]

for edad in edades_prueba:
    try:
        validar_edad(edad)
        print(f"✓ Edad {edad} válida")
    except (EdadInvalidaError, TypeError) as e:
        print(f"✗ Error: {e}")

print("\n=== OPERACIONES BANCARIAS ===")
cuenta1 = CuentaBancaria("Juan Pérez", 1000)
cuenta2 = CuentaBancaria("María García", 500)

try:
    cuenta1.retirar(500)
    cuenta1.retirar(600)  # Provocará SaldoInsuficienteError
except SaldoInsuficienteError as e:
```

```
    print(f"Operación rechazada: {e}")
    deficit = e.cantidad_solicitada - e.saldo_actual
    print(f"Déficit: {deficit:.2f}")

print("\n=== VALIDACIÓN DE CONTRASEÑAS ===")
passwords = ["abc", "abcdefgh", "Abcdefgh1", "Abcdefgh1!"]

for pwd in passwords:
    print(f"\nValidando: {pwd}")
    try:
        validar_password(pwd)
        print("✓ Contraseña fuerte")
    except PasswordDebilidadError as e:
        print(f"✗ {e}")
```

## 4.4 Context Managers y with

```
# with para manejo automático de recursos
# Archivo se cierra automáticamente
try:
    with open("datos.txt", "r") as archivo:
        contenido = archivo.read()
        print(contenido)
except FileNotFoundError:
    print("Archivo no encontrado")

# Context manager personalizado
class ConexionBD:
    def __init__(self, nombre_bd):
        self.nombre_bd = nombre_bd
        self.conexion = None

    def __enter__(self):
        print(f"Conectando a {self.nombre_bd}...")
        self.conexion = f"Conexión a {self.nombre_bd}"
        return self.conexion

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(f"Cerrando conexión a {self.nombre_bd}...")
        if exc_type:
            print(f"Se produjo un error: {exc_val}")
        self.conexion = None
        return False  # Propaga la excepción

# Uso del context manager
with ConexionBD("mi_base_datos") as conn:
    print(f"Trabajando con: {conn}")
    # El recurso se cierra automáticamente

# Context manager con contextlib
from contextlib import contextmanager

@contextmanager
def temporizador():
    import time
    inicio = time.time()
    print("Iniciando medición...")
    try:
        yield
    finally:
```

```
        fin = time.time()
        print(f"Tiempo transcurrido: {fin - inicio:.2f} segundos")

# Uso
with temporizador():
    import time
    time.sleep(1)
    print("Procesando...")

# suppress para ignorar excepciones específicas
from contextlib import suppress

# Sin suppress
try:
    lista = [1, 2, 3]
    print(lista[10])
except IndexError:
    pass

# Con suppress (más limpio)
with suppress(IndexError):
    lista = [1, 2, 3]
    print(lista[10])

print("El programa continúa...")
```

# Ejercicios Propuestos

## Ejercicio 1: Calculadora Científica

Crea una calculadora que incluya operaciones básicas y científicas. Usa match-case para el menú, manejo de excepciones personalizadas para operaciones inválidas, y permite historial de operaciones.

## Ejercicio 2: Gestor de Tareas

Desarrolla un sistema de gestión de tareas con prioridades. Incluye fechas de vencimiento, filtrado por estado, excepciones para tareas duplicadas, y persistencia en JSON.

## Ejercicio 3: Juego de Aventura por Texto

Implementa un juego con múltiples habitaciones y objetos. Usa diccionarios para el mapa, match-case para comandos, excepciones para movimientos inválidos, y sistema de inventario.

## Ejercicio 4: Analizador de Logs

Crea un programa que analice archivos de log. Debe filtrar por nivel de severidad, buscar patrones, generar estadísticas, y manejar archivos corruptos con excepciones.

## Ejercicio 5: Sistema de Reservas

Desarrolla un sistema de reservas (hotel, restaurante, etc.). Incluye validación de fechas, control de disponibilidad, excepciones personalizadas para conflictos, y reportes de ocupación.

# Resumen de Buenas Prácticas en Python

1. **PEP 8**: Sigue las convenciones de estilo de Python
    - Indentación de 4 espacios
    - Nombres en snake_case para funciones y variables
    - Nombres en PascalCase para clases
2. **Excepciones**: Captura excepciones específicas, no uses `except:` sin tipo
3. **Context Managers**: Usa `with` para gestionar recursos automáticamente
4. **Comprensiones**: Prefiere comprensiones de lista cuando sean legibles
5. **Valores Truthy/Falsy**: Aprovecha la evaluación booleana de Python
6. **Type Hints**: Usa anotaciones de tipo para código más claro
7. **Docstrings**: Documenta funciones y clases con docstrings
8. **EAFP vs LBYL**: "Es más fácil pedir perdón que permiso" - usa try/except en lugar de verificaciones previas cuando sea apropiado

Este tema establece las bases del control de flujo en Python, fundamental para crear programas robustos e interactivos.