Tema 1: Fundamentos del Lenguaje Python

1. Estructura de un Programa Python

1.1 Estructura Básica

Python es un lenguaje interpretado que enfatiza la legibilidad del código. A diferencia de otros lenguajes, Python usa la indentación como parte de su sintaxis.

```
# mi_primer_programa.py

# Este es un programa básico en Python
print("¡Hola Mundo!")

# Python ejecuta el código línea por línea
nombre = "Python"
version = 3.11
print(f"Bienvenido a {nombre} versión {version}")
```

Características clave:

- No requiere función `main()` obligatoria
- No requiere declaración de tipos de variables
- No necesita punto y coma al final de las líneas
- La indentación es obligatoria y define los bloques de código
- Los archivos Python tienen extensión `.py`

1.2 Programa con Función Principal

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Módulo de ejemplo que demuestra la estructura de un programa Python.
Este es un docstring que documenta el módulo.
"""

# Importaciones
import sys
import os
from datetime import datetime

# Constantes (por convención en MAYÚSCULAS)
VERSION = "1.0.0"
AUTOR = "Tu Nombre"

# Variables globales
contador_global = 0

# Definición de funciones
def saludar(nombre):
    """
    Función que saluda a una persona.
```

```
    Args:
        nombre (str): El nombre de la persona a saludar

    Returns:
        str: El mensaje de saludo
    """
    return f"Hola, {nombre}!"

def main():
    """Función principal del programa"""
    print(f"Programa versión {VERSION}")
    print(f"Autor: {AUTOR}")

    nombre_usuario = input("¿Cómo te llamas? ")
    mensaje = saludar(nombre_usuario)
    print(mensaje)

    print(f"Hora actual: {datetime.now()}")
    return 0

# Punto de entrada del programa
if __name__ == "__main__":
    # Este bloque solo se ejecuta si el archivo se ejecuta
directamente
    # No se ejecuta si el archivo es importado como módulo
    sys.exit(main())
```

1.3 Comentarios y Documentación

```
# Comentario de una línea

"""
Comentario de múltiples líneas.
También llamado docstring cuando se usa
para documentar módulos, clases o funciones.
"""

'''
También se pueden usar comillas simples
para comentarios multilínea
'''

def funcion_documentada(parametro1, parametro2):
    """
    Descripción breve de la función.

    Descripción más detallada explicando el comportamiento
    de la función y cualquier detalle importante.

    Args:
        parametro1 (int): Descripción del primer parámetro
        parametro2 (str): Descripción del segundo parámetro

    Returns:
        bool: Descripción del valor de retorno

    Raises:
        ValueError: Cuando parametro1 es negativo
        TypeError: Cuando parametro2 no es string
```

```
    Examples:
        >>> funcion_documentada(10, "texto")
        True
        >>> funcion_documentada(-5, "texto")
        ValueError: parametro1 debe ser positivo
    """
    if parametro1 < 0:
        raise ValueError("parametro1 debe ser positivo")
    if not isinstance(parametro2, str):
        raise TypeError("parametro2 debe ser string")
    return True

# Acceder a la documentación
print(funcion_documentada.__doc__)
```

1.4 Codificación y Shebang

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
El shebang (#!) indica qué intérprete usar en sistemas Unix/Linux.
La declaración de codificación especifica el encoding del archivo.
UTF-8 es el estándar recomendado.
"""

# Con UTF-8 podemos usar caracteres especiales
mensaje_español = "¡Hola! ¿Cómo estás?"
emoji = "Python es genial "
símbolos = "π = 3.14159, ∑, ∫, √"

print(mensaje_español)
print(emoji)
print(símbolos)
```

2. Tipos de Datos en Python

Python es un lenguaje de tipado dinámico pero fuertemente tipado. Los tipos se asignan automáticamente según el valor.

2.1 Tipos Numéricos

```
# int - Números enteros (sin límite de tamaño)
edad = 25
temperatura = -10
poblacion = 7_800_000_000  # Se pueden usar _ para legibilidad
numero_grande = 10**100  # Python maneja números arbitrariamente
grandes

print(f"Edad: {edad}, Tipo: {type(edad)}")
print(f"Población mundial: {poblacion:,}")  # Formato con comas
print(f"10^100 = {numero_grande}")

# float - Números decimales (punto flotante de 64 bits)
precio = 19.99
pi = 3.14159265359
cientifico = 1.23e-4  # Notación científica
```

```
infinito = float('inf')  # Infinito
no_numero = float('nan')  # Not a Number

print(f"Precio: ${precio:.2f}")
print(f"Pi: {pi}")
print(f"Notación científica: {cientifico}")
print(f"Infinito: {infinito}")
print(f"NaN: {no_numero}")

# complex - Números complejos
complejo = 3 + 4j
otro_complejo = complex(2, -3)

print(f"Número complejo: {complejo}")
print(f"Parte real: {complejo.real}, Parte imaginaria:
{complejo.imag}")
print(f"Módulo: {abs(complejo)}")

# Operaciones entre tipos numéricos
entero = 10
decimal = 3.5
resultado = entero + decimal  # Promoción automática a float
print(f"{entero} + {decimal} = {resultado} (tipo:
{type(resultado).__name__})")
```

2.2 Tipo Booleano

```
# bool - Valores booleanos
verdadero = True  # Nota: T mayúscula
falso = False     # Nota: F mayúscula

print(f"Verdadero: {verdadero}, Tipo: {type(verdadero)}")
print(f"Falso: {falso}")

# Los booleanos son subclase de int
print(f"True como número: {int(True)}")   # 1
print(f"False como número: {int(False)}")  # 0
print(f"True + True = {True + True}")      # 2

# Valores truthy y falsy
# Falsy: False, 0, 0.0, "", [], {}, (), None
# Truthy: Todo lo demás

valores = [True, False, 0, 1, "", "texto", [], [1,2], None, 42]
for valor in valores:
    if valor:
        print(f"{repr(valor)} es truthy")
    else:
        print(f"{repr(valor)} es falsy")
```

2.3 Tipo None

```
# None - Representa la ausencia de valor
vacio = None

print(f"Valor: {vacio}, Tipo: {type(vacio)}")

# Uso común: valores por defecto
def buscar_usuario(id):
    # Simulación de búsqueda
```

```
    if id == 1:
        return "Juan"
    return None

usuario = buscar_usuario(1)
if usuario is not None:  # Usar 'is' para comparar con None
    print(f"Usuario encontrado: {usuario}")
else:
    print("Usuario no encontrado")

# None es singleton (solo existe una instancia)
a = None
b = None
print(f"a is b: {a is b}")  # True
```

2.4 Tipos de Secuencia

```
# list - Lista mutable
frutas = ["manzana", "naranja", "plátano"]
numeros = [1, 2, 3, 4, 5]
mixta = [1, "dos", 3.0, True, None]  # Pueden contener diferentes
tipos

print(f"Frutas: {frutas}")
print(f"Primer elemento: {frutas[0]}")
print(f"Último elemento: {frutas[-1]}")

# Modificar listas
frutas.append("uva")
frutas.insert(1, "pera")
frutas.remove("naranja")
print(f"Frutas modificadas: {frutas}")

# tuple - Tupla inmutable
coordenadas = (10, 20)
rgb = (255, 128, 0)
persona = ("Juan", 25, "España")  # Pueden contener diferentes tipos

print(f"Coordenadas: {coordenadas}")
print(f"Color RGB: {rgb}")
# coordenadas[0] = 15  # Error: las tuplas son inmutables

# Desempaquetado de tuplas
x, y = coordenadas
print(f"x = {x}, y = {y}")

nombre, edad, pais = persona
print(f"{nombre} tiene {edad} años y es de {pais}")

# range - Secuencia de números
rango = range(5)  # 0, 1, 2, 3, 4
rango_personalizado = range(1, 10, 2)  # 1, 3, 5, 7, 9

print(f"Range(5): {list(rango)}")
print(f"Range(1, 10, 2): {list(rango_personalizado)}")
```

2.5 Tipos de Conjunto

```
# set - Conjunto mutable sin duplicados
numeros = {1, 2, 3, 4, 5}
```

```
letras = set("abracadabra")  # Elimina duplicados automáticamente
vacio = set()  # Conjunto vacío (no usar {})

print(f"Números: {numeros}")
print(f"Letras únicas: {letras}")

# Operaciones de conjuntos
pares = {2, 4, 6, 8, 10}
multiplos_3 = {3, 6, 9}

print(f"Unión: {pares | multiplos_3}")
print(f"Intersección: {pares & multiplos_3}")
print(f"Diferencia: {pares - multiplos_3}")
print(f"Diferencia simétrica: {pares ^ multiplos_3}")

# frozenset - Conjunto inmutable
inmutable = frozenset([1, 2, 3, 4, 5])
print(f"Frozenset: {inmutable}")
# inmutable.add(6)  # Error: frozenset no tiene método add
```

2.6 Tipo Diccionario

```
# dict - Diccionario (mapa clave-valor)
persona = {
    "nombre": "María",
    "edad": 30,
    "ciudad": "Madrid",
    "profesion": "Ingeniera"
}

print(f"Diccionario: {persona}")
print(f"Nombre: {persona['nombre']}")
print(f"Edad: {persona.get('edad')}")

# Modificar diccionarios
persona["edad"] = 31
persona["email"] = "maria@email.com"
del persona["ciudad"]

print(f"Diccionario modificado: {persona}")

# Iterar sobre diccionarios
print("\nClaves:")
for clave in persona.keys():
    print(f"  - {clave}")

print("\nValores:")
for valor in persona.values():
    print(f"  - {valor}")

print("\nPares clave-valor:")
for clave, valor in persona.items():
    print(f"  {clave}: {valor}")

# Diccionarios con diferentes tipos de claves
mixto = {
    1: "uno",
    "dos": 2,
    (3, 3): "tupla",
    # [4, 4]: "lista"  # Error: las listas no pueden ser claves
```

```
}
print(f"\nDiccionario mixto: {mixto}")
```

3. El Tipo String

Las cadenas en Python son inmutables y pueden definirse con comillas simples, dobles o triples.

3.1 Declaración y Uso Básico

```
# Diferentes formas de crear strings
comillas_simples = 'Hola Python'
comillas_dobles = "Hola Python"
comillas_triples = """Este es un string
que ocupa varias
líneas"""

# Strings vacíos
vacio1 = ""
vacio2 = ''
vacio3 = str()  # Constructor

print(f"Comillas simples: {comillas_simples}")
print(f"Comillas dobles: {comillas_dobles}")
print(f"Comillas triples:\n{comillas_triples}")

# Strings con caracteres especiales
con_salto = "Primera línea\nSegunda línea"
con_tab = "Columna1\tColumna2\tColumna3"
con_comillas = "Ella dijo: \"Hola\""
con_backslash = "C:\\Users\\Python"
raw_string = r"C:\Users\Python\nuevo"  # Raw string (sin escape)

print(f"\nCon salto de línea:\n{con_salto}")
print(f"Con tabulación:\n{con_tab}")
print(f"Con comillas: {con_comillas}")
print(f"Con backslash: {con_backslash}")
print(f"Raw string: {raw_string}")

# Concatenación
nombre = "Python"
version = "3.11"
concatenado = nombre + " " + version
repetido = "=" * 50

print(f"\nConcatenado: {concatenado}")
print(f"Repetido: {repetido}")

# Acceso a caracteres (indexing)
palabra = "Python"
print(f"\nPrimer carácter: {palabra[0]}")
print(f"Último carácter: {palabra[-1]}")
print(f"Penúltimo: {palabra[-2]}")

# Slicing (subcadenas)
texto = "Programación"
print(f"\nTexto completo: {texto}")
print(f"Primeros 5: {texto[:5]}")
print(f"Últimos 4: {texto[-4:]}")
```

```
print(f"Del 3 al 7: {texto[3:7]}")
print(f"Cada 2 caracteres: {texto[::2]}")
print(f"Invertido: {texto[::-1]}")
```

3.2 Métodos de String

```
texto = "  Python es un Lenguaje de Programación  "

# Métodos de transformación
print(f"Original: '{texto}'")
print(f"Strip: '{texto.strip()}'")  # Elimina espacios
print(f"Mayúsculas: '{texto.upper()}'")
print(f"Minúsculas: '{texto.lower()}'")
print(f"Título: '{texto.title()}'")
print(f"Capitalizar: '{texto.capitalize()}'")
print(f"Intercambiar: '{texto.swapcase()}'")

# Métodos de búsqueda
frase = "Python es genial y Python es popular"
print(f"\nFrase: {frase}")
print(f"Índice de 'Python': {frase.index('Python')}")
print(f"Índice de 'Python' desde 10: {frase.index('Python', 10)}")
print(f"Contar 'Python': {frase.count('Python')}")
print(f"Empieza con 'Python': {frase.startswith('Python')}")
print(f"Termina con 'popular': {frase.endswith('popular')}")
print(f"Contiene 'genial': {'genial' in frase}")

# Métodos de validación
print("\nValidaciones:")
print(f"'123'.isdigit(): {'123'.isdigit()}")
print(f"'abc'.isalpha(): {'abc'.isalpha()}")
print(f"'abc123'.isalnum(): {'abc123'.isalnum()}")
print(f"'   '.isspace(): {'   '.isspace()}")
print(f"'Title Case'.istitle(): {'Title Case'.istitle()}")
print(f"'UPPER'.isupper(): {'UPPER'.isupper()}")
print(f"'lower'.islower(): {'lower'.islower()}")

# Reemplazar y dividir
texto = "manzana,pera,naranja,uva"
print(f"\nTexto: {texto}")
print(f"Reemplazar: {texto.replace(',', ' - ')}")
print(f"Split: {texto.split(',')}")

palabras = ["Python", "es", "genial"]
print(f"Join: {' '.join(palabras)}")

# Formateo
precio = 19.99
cantidad = 3
print(f"\nFormateo:")
print(f"F-string: {cantidad} items cuestan ${precio * cantidad:.2f}")
print("Format: {} items cuestan ${:.2f}".format(cantidad, precio *
cantidad))
print("% formato: %d items cuestan $%.2f" % (cantidad, precio *
cantidad))
```

3.3 Formateo Avanzado de Strings

```
# F-strings (Python 3.6+) - Método recomendado
nombre = "Ana"
```

```
edad = 25
altura = 1.68

# Básico
print(f"Me llamo {nombre} y tengo {edad} años")

# Expresiones
print(f"En 10 años tendré {edad + 10} años")
print(f"Mi edad al cuadrado es {edad**2}")

# Formato de números
pi = 3.14159265359
print(f"Pi con 2 decimales: {pi:.2f}")
print(f"Pi con 5 decimales: {pi:.5f}")
print(f"Pi en notación científica: {pi:.2e}")

# Alineación y relleno
print(f"{'Izquierda':<20}|")
print(f"{'Centro':^20}|")
print(f"{'Derecha':>20}|")
print(f"{'Relleno con ceros':0>20}")
print(f"{'Relleno con guiones':-^20}")

# Números con formato
numero = 1234567.89
print(f"Con separador de miles: {numero:,.2f}")
print(f"Porcentaje: {0.1234:.1%}")
print(f"Binario: {42:b}")
print(f"Hexadecimal: {255:x}")
print(f"Hexadecimal mayúsculas: {255:X}")

# Fecha y hora
from datetime import datetime
ahora = datetime.now()
print(f"Fecha: {ahora:%Y-%m-%d}")
print(f"Hora: {ahora:%H:%M:%S}")
print(f"Fecha completa: {ahora:%c}")

# Debug con = (Python 3.8+)
x = 10
y = 20
print(f"{x=}, {y=}, {x+y=}")  # Imprime: x=10, y=20, x+y=30

# Multiline f-strings
mensaje = f"""
Resumen de {nombre}:
  - Edad: {edad} años
  - Altura: {altura:.2f} m
  - IMC aproximado: {70 / (altura**2):.1f}
"""
print(mensaje)
```

4. Expresiones y Operadores

4.1 Operadores Aritméticos

```
# Operadores básicos
a = 15
b = 4
```

```
print(f"a = {a}, b = {b}")
print(f"Suma: {a} + {b} = {a + b}")
print(f"Resta: {a} - {b} = {a - b}")
print(f"Multiplicación: {a} * {b} = {a * b}")
print(f"División: {a} / {b} = {a / b}")  # División real
print(f"División entera: {a} // {b} = {a // b}")
print(f"Módulo: {a} % {b} = {a % b}")
print(f"Potencia: {a} ** {b} = {a ** b}")
print(f"Negación: -{a} = {-a}")

# Operadores con diferentes tipos
print(f"\nOperaciones mixtas:")
print(f"String * int: {'Python' * 3}")
print(f"List * int: {[1, 2] * 3}")
print(f"String + String: {'Py' + 'thon'}")
print(f"List + List: {[1, 2] + [3, 4]}")

# Operadores de asignación aumentada
x = 10
print(f"\nValor inicial de x: {x}")
x += 5;  print(f"x += 5: {x}")
x -= 3;  print(f"x -= 3: {x}")
x *= 2;  print(f"x *= 2: {x}")
x //= 4; print(f"x //= 4: {x}")
x **= 2; print(f"x **= 2: {x}")
x %= 5;  print(f"x %= 5: {x}")

# División y redondeo
import math
numero = 7.8
print(f"\nNúmero: {numero}")
print(f"Redondeo: {round(numero)}")
print(f"Redondeo a 1 decimal: {round(numero, 1)}")
print(f"Techo: {math.ceil(numero)}")
print(f"Piso: {math.floor(numero)}")
print(f"Truncar: {math.trunc(numero)}")
```

4.2 Operadores de Comparación

```
x = 10
y = 20
z = 10

print("Comparaciones básicas:")
print(f"{x} == {y}: {x == y}")  # Igualdad
print(f"{x} != {y}: {x != y}")  # Desigualdad
print(f"{x} < {y}: {x < y}")    # Menor que
print(f"{x} > {y}: {x > y}")    # Mayor que
print(f"{x} <= {z}: {x <= z}")  # Menor o igual
print(f"{x} >= {z}: {x >= z}")  # Mayor o igual

# Comparación encadenada (característica única de Python)
a = 5
print(f"\nComparación encadenada:")
print(f"1 < {a} < 10: {1 < a < 10}")
print(f"10 <= {a} <= 20: {10 <= a <= 20}")

# Comparación de diferentes tipos
print(f"\nComparaciones especiales:")
```

```
print(f"'10' == 10: {'10' == 10}")  # False - tipos diferentes
print(f"True == 1: {True == 1}")    # True - bool es subclase de int
print(f"False == 0: {False == 0}")  # True

# Comparación de secuencias
lista1 = [1, 2, 3]
lista2 = [1, 2, 3]
lista3 = [1, 2, 4]

print(f"\nComparación de listas:")
print(f"{lista1} == {lista2}: {lista1 == lista2}")
print(f"{lista1} < {lista3}: {lista1 < lista3}")  # Comparación
lexicográfica

# is vs ==
a = [1, 2, 3]
b = [1, 2, 3]
c = a

print(f"\nIdentidad vs Igualdad:")
print(f"a == b: {a == b}")  # True - mismo contenido
print(f"a is b: {a is b}")  # False - diferentes objetos
print(f"a is c: {a is c}")  # True - mismo objeto
```

4.3 Operadores Lógicos

```
# and, or, not
verdadero = True
falso = False

print("Operadores lógicos básicos:")
print(f"True and True: {verdadero and verdadero}")
print(f"True and False: {verdadero and falso}")
print(f"True or False: {verdadero or falso}")
print(f"False or False: {falso or falso}")
print(f"not True: {not verdadero}")
print(f"not False: {not falso}")

# Evaluación de cortocircuito
print("\nCortocircuito:")
def funcion_costosa():
    print("  Ejecutando función costosa...")
    return True

print("False and funcion_costosa():")
resultado = False and funcion_costosa()  # No ejecuta la función
print(f"  Resultado: {resultado}")

print("True or funcion_costosa():")
resultado = True or funcion_costosa()  # No ejecuta la función
print(f"  Resultado: {resultado}")

# Valores retornados por and/or
# and retorna el primer valor falsy o el último valor
# or retorna el primer valor truthy o el último valor
print("\nValores retornados:")
print(f"5 and 10: {5 and 10}")  # 10
print(f"0 and 10: {0 and 10}")  # 0
print(f"5 or 10: {5 or 10}")    # 5
print(f"0 or 10: {0 or 10}")    # 10
```

```
print(f"'' or 'default': {'' or 'default'}")  # 'default'

# Uso práctico
nombre = input("Introduce tu nombre (o Enter para default): ") or
"Anónimo"
print(f"Nombre: {nombre}")
```

4.4 Operadores de Pertenencia e Identidad

```
# in, not in - Pertenencia
lista = [1, 2, 3, 4, 5]
texto = "Python es genial"
diccionario = {"a": 1, "b": 2, "c": 3}

print("Operador in:")
print(f"3 in {lista}: {3 in lista}")
print(f"10 in {lista}: {10 in lista}")
print(f"'Python' in '{texto}': {'Python' in texto}")
print(f"'Java' in '{texto}': {'Java' in texto}")
print(f"'a' in {diccionario}: {'a' in diccionario}")

print("\nOperador not in:")
print(f"6 not in {lista}: {6 not in lista}")
print(f"'Java' not in '{texto}': {'Java' not in texto}")

# is, is not - Identidad
a = [1, 2, 3]
b = [1, 2, 3]
c = a
ninguno = None

print("\nOperador is:")
print(f"a is b: {a is b}")  # False - diferentes objetos
print(f"a is c: {a is c}")  # True - mismo objeto
print(f"ninguno is None: {ninguno is None}")  # True

print("\nOperador is not:")
print(f"a is not b: {a is not b}")  # True
print(f"ninguno is not None: {ninguno is not None}")  # False

# Casos especiales con números pequeños (interning)
x = 256
y = 256
z = 257
w = 257
print("\nInternado de enteros pequeños:")
print(f"x = 256, y = 256")
print(f"x is y: {x is y}")  # True (Python interna -5 a 256)
print(f"z = 257, w = 257")
print(f"z is w: {z is w}")  # Puede ser False
```

4.5 Operadores de Bits

```
# Operadores bitwise
a = 60  # 0011 1100
b = 13  # 0000 1101

print(f"a = {a} ({bin(a)})")
print(f"b = {b} ({bin(b)})")
print(f"AND: {a} & {b} = {a & b} ({bin(a & b)})")
```

```
print(f"OR:  {a} | {b} = {a | b} ({bin(a | b)})")
print(f"XOR: {a} ^ {b} = {a ^ b} ({bin(a ^ b)})")
print(f"NOT: ~{a} = {~a} ({bin(~a & 0xFF)})")  # Mostrando solo 8 bits
print(f"Left shift:  {a} << 2 = {a << 2} ({bin(a << 2)})")
print(f"Right shift: {a} >> 2 = {a >> 2} ({bin(a >> 2)})")

# Uso práctico: flags
LEER = 0b100    # 4
ESCRIBIR = 0b010  # 2
EJECUTAR = 0b001  # 1

permisos = LEER | ESCRIBIR  # Combinar permisos
print(f"\nPermisos: {bin(permisos)}")
print(f"¿Puede leer? {bool(permisos & LEER)}")
print(f"¿Puede escribir? {bool(permisos & ESCRIBIR)}")
print(f"¿Puede ejecutar? {bool(permisos & EJECUTAR)}")
```

5. Conversiones de Tipo

5.1 Conversiones Básicas

```
# int() - Convertir a entero
print("Conversión a int:")
print(f"int('123'): {int('123')}")
print(f"int(45.67): {int(45.67)}")  # Trunca decimales
print(f"int(True): {int(True)}")
print(f"int(False): {int(False)}")
print(f"int('1010', 2): {int('1010', 2)}")  # Binario a decimal
print(f"int('FF', 16): {int('FF', 16)}")    # Hexadecimal a decimal

# float() - Convertir a decimal
print("\nConversión a float:")
print(f"float('3.14'): {float('3.14')}")
print(f"float(10): {float(10)}")
print(f"float('inf'): {float('inf')}")
print(f"float('-inf'): {float('-inf')}")
print(f"float('nan'): {float('nan')}")

# str() - Convertir a string
print("\nConversión a str:")
print(f"str(123): {repr(str(123))}")
print(f"str(45.67): {repr(str(45.67))}")
print(f"str(True): {repr(str(True))}")
print(f"str([1,2,3]): {repr(str([1,2,3]))}")

# bool() - Convertir a booleano
print("\nConversión a bool:")
valores = [0, 1, "", "texto", [], [1], None, 42, 0.0, 0.1]
for valor in valores:
    print(f"bool({repr(valor)}): {bool(valor)}")

# list() - Convertir a lista
print("\nConversión a list:")
print(f"list('Python'): {list('Python')}")
print(f"list((1, 2, 3)): {list((1, 2, 3))}")
print(f"list(range(5)): {list(range(5))}")
print(f"list({{1, 2, 3}}): {list({1, 2, 3})}")

# tuple() - Convertir a tupla
```

```
print("\nConversión a tuple:")
print(f"tuple([1, 2, 3]): {tuple([1, 2, 3])}")
print(f"tuple('abc'): {tuple('abc')}")

# set() - Convertir a conjunto
print("\nConversión a set:")
print(f"set([1, 2, 2, 3, 3, 3]): {set([1, 2, 2, 3, 3, 3])}")
print(f"set('aabbcc'): {set('aabbcc')}")

# dict() - Convertir a diccionario
print("\nConversión a dict:")
print(f"dict([('a', 1), ('b', 2)]): {dict([('a', 1), ('b', 2)])}")
print(f"dict(zip(['x', 'y'], [10, 20])): {dict(zip(['x', 'y'], [10,
20]))}")
```

5.2 Conversiones Seguras con Validación

```
def convertir_a_entero(valor):
    """Convierte de forma segura un valor a entero"""
    try:
        return int(valor)
    except (ValueError, TypeError) as e:
        print(f"Error: No se puede convertir {repr(valor)} a entero")
        return None

# Pruebas
valores = ["123", "45.67", "abc", None, [1,2,3], 10]
for v in valores:
    resultado = convertir_a_entero(v)
    print(f"{repr(v)} -> {resultado}")

def entrada_numero(mensaje, tipo=float):
    """Solicita un número al usuario con validación"""
    while True:
        try:
            entrada = input(mensaje)
            return tipo(entrada)
        except ValueError:
            print(f"Error: Debe introducir un {tipo.__name__} válido")

# Uso
# numero = entrada_numero("Introduce un número: ")
# entero = entrada_numero("Introduce un entero: ", int)

# Conversión con valores por defecto
def obtener_entero(valor, default=0):
    """Convierte a entero o retorna valor por defecto"""
    try:
        return int(valor)
    except:
        return default

print("\nConversión con default:")
print(f"obtener_entero('123'): {obtener_entero('123')}")
print(f"obtener_entero('abc'): {obtener_entero('abc')}")
print(f"obtener_entero('xyz', -1): {obtener_entero('xyz', -1)}")
```

5.3 Conversiones entre Bytes y Strings

```
# str a bytes
```

```
texto = "Python "
print(f"Texto original: {texto}")

# Codificar a bytes
bytes_utf8 = texto.encode('utf-8')
bytes_latin = texto.encode('latin-1', errors='ignore')
bytes_ascii = texto.encode('ascii', errors='replace')

print(f"UTF-8: {bytes_utf8}")
print(f"Latin-1: {bytes_latin}")
print(f"ASCII: {bytes_ascii}")

# bytes a str
datos_bytes = b'Python'
texto_decodificado = datos_bytes.decode('utf-8')
print(f"\nBytes: {datos_bytes}")
print(f"Decodificado: {texto_decodificado}")

# Conversión hexadecimal
hex_string = "48656c6c6f"  # "Hello" en hex
bytes_from_hex = bytes.fromhex(hex_string)
print(f"\nHex string: {hex_string}")
print(f"Bytes: {bytes_from_hex}")
print(f"Texto: {bytes_from_hex.decode('ascii')}")

# Bytes a hex
mensaje = b"Python"
hex_from_bytes = mensaje.hex()
print(f"\nBytes: {mensaje}")
print(f"Hex: {hex_from_bytes}")
```

6. Enumerados (Enum)

Los enumerados proporcionan una forma de crear conjuntos de constantes nombradas.

6.1 Enum Básico

```
from enum import Enum, auto

# Definición básica
class DiaSemana(Enum):
    LUNES = 1
    MARTES = 2
    MIERCOLES = 3
    JUEVES = 4
    VIERNES = 5
    SABADO = 6
    DOMINGO = 7

# Uso
hoy = DiaSemana.MIERCOLES
print(f"Hoy es: {hoy}")
print(f"Nombre: {hoy.name}")
print(f"Valor: {hoy.value}")

# Comparación
if hoy == DiaSemana.MIERCOLES:
    print("Es mitad de semana")
```

```
# Iterar sobre enum
print("\nTodos los días:")
for dia in DiaSemana:
    print(f"  {dia.name}: {dia.value}")

# Acceso por valor o nombre
dia_por_valor = DiaSemana(5)
dia_por_nombre = DiaSemana["LUNES"]
print(f"\nDía 5: {dia_por_valor}")
print(f"LUNES: {dia_por_nombre}")

# Enum con auto()
class Estado(Enum):
    PENDIENTE = auto()
    PROCESANDO = auto()
    COMPLETADO = auto()
    CANCELADO = auto()

print("\nEstados con auto():")
for estado in Estado:
    print(f"  {estado.name}: {estado.value}")
```

6.2 Enum con Tipos Específicos

```
from enum import IntEnum, Flag, IntFlag
import enum

# IntEnum - Compatible con int
class Prioridad(IntEnum):
    BAJA = 1
    MEDIA = 2
    ALTA = 3
    URGENTE = 4

# Se puede usar como int
prioridad = Prioridad.ALTA
print(f"Prioridad: {prioridad}")
print(f"Prioridad > 2: {prioridad > 2}")
print(f"Prioridad + 1: {prioridad + 1}")

# Flag - Para combinaciones
class Permiso(Flag):
    LEER = 1
    ESCRIBIR = 2
    EJECUTAR = 4
    ELIMINAR = 8

# Combinar flags
permisos = Permiso.LEER | Permiso.ESCRIBIR
print(f"\nPermisos: {permisos}")
print(f"¿Puede leer?: {Permiso.LEER in permisos}")
print(f"¿Puede ejecutar?: {Permiso.EJECUTAR in permisos}")

# Agregar permiso
permisos |= Permiso.EJECUTAR
print(f"Permisos actualizados: {permisos}")

# IntFlag - Flag compatible con int
class EstadoConexion(IntFlag):
    DESCONECTADO = 0
```

```
    CONECTANDO = 1
    CONECTADO = 2
    ERROR = 4
    RECONECTANDO = 8

estado = EstadoConexion.CONECTADO | EstadoConexion.ERROR
print(f"\nEstado: {estado}")
print(f"Valor numérico: {int(estado)}")
```

6.3 Enum Funcional y Avanzado

```
from enum import Enum, unique
import enum

# Asegurar valores únicos con @unique
@unique
class Color(Enum):
    ROJO = 1
    VERDE = 2
    AZUL = 3
    # CYAN = 3  # Error: valor duplicado

# Enum funcional (alternativa)
Animal = Enum('Animal', 'PERRO GATO PAJARO PEZ')
print("Animales:")
for animal in Animal:
    print(f"  {animal.name}: {animal.value}")

# Enum con métodos personalizados
class TipoArchivo(Enum):
    TEXTO = ("txt", "text/plain", "📄")
    IMAGEN = ("jpg", "image/jpeg", "🖼")
    VIDEO = ("mp4", "video/mp4", "🎬")
    AUDIO = ("mp3", "audio/mpeg", "🎵")
    PDF = ("pdf", "application/pdf", "📑")

    def __init__(self, extension, mime_type, icono):
        self.extension = extension
        self.mime_type = mime_type
        self.icono = icono

    @classmethod
    def desde_extension(cls, ext):
        """Obtener tipo desde extensión"""
        for tipo in cls:
            if tipo.extension == ext:
                return tipo
        raise ValueError(f"Extensión {ext} no reconocida")

    def es_multimedia(self):
        """Verificar si es archivo multimedia"""
        return self in [TipoArchivo.VIDEO, TipoArchivo.AUDIO,
TipoArchivo.IMAGEN]

# Uso
archivo = TipoArchivo.VIDEO
print(f"\nTipo: {archivo.name}")
print(f"Extensión: {archivo.extension}")
print(f"MIME: {archivo.mime_type}")
```

```
print(f"Icono: {archivo.icono}")
print(f"¿Es multimedia?: {archivo.es_multimedia()}")

# Buscar por extensión
tipo = TipoArchivo.desde_extension("pdf")
print(f"\nArchivo .pdf es de tipo: {tipo.name} {tipo.icono}")

# Enum con property
class EstadoPedido(Enum):
    NUEVO = (1, "El pedido ha sido creado", True)
    PROCESANDO = (2, "El pedido se está procesando", True)
    ENVIADO = (3, "El pedido ha sido enviado", True)
    ENTREGADO = (4, "El pedido ha sido entregado", False)
    CANCELADO = (5, "El pedido ha sido cancelado", False)

    def __init__(self, codigo, descripcion, activo):
        self.codigo = codigo
        self.descripcion = descripcion
        self.activo = activo

    @property
    def puede_cancelar(self):
        return self in [EstadoPedido.NUEVO, EstadoPedido.PROCESANDO]

    @property
    def es_final(self):
        return self in [EstadoPedido.ENTREGADO,
EstadoPedido.CANCELADO]

# Uso
estado = EstadoPedido.PROCESANDO
print(f"\nEstado del pedido: {estado.name}")
print(f"Descripción: {estado.descripcion}")
print(f"¿Puede cancelar?: {estado.puede_cancelar}")
print(f"¿Es estado final?: {estado.es_final}")
```

7. Entrada de Datos del Usuario

7.1 Función input() Básica

```
# input() siempre retorna un string
nombre = input("¿Cómo te llamas? ")
print(f"Hola, {nombre}!")

# Con prompt vacío
dato = input()
print(f"Has escrito: {dato}")

# Con prompt multilínea
mensaje = """
Por favor, introduce tu opción:
1. Opción A
2. Opción B
3. Opción C
Tu elección: """
opcion = input(mensaje)
print(f"Has elegido: {opcion}")

# Input con valor por defecto (simulado)
```

```
def input_con_default(prompt, default=""):
    """Input con valor por defecto"""
    entrada = input(f"{prompt} [{default}]: ")
    return entrada if entrada else default

ciudad = input_con_default("Ciudad", "Madrid")
print(f"Ciudad seleccionada: {ciudad}")
```

7.2 Conversión y Validación de Tipos

```
# Conversión básica (puede fallar)
try:
    edad = int(input("Introduce tu edad: "))
    print(f"Dentro de 10 años tendrás {edad + 10} años")
except ValueError:
    print("Error: Debes introducir un número")

# Función robusta para enteros
def leer_entero(prompt, min_val=None, max_val=None):
    """Lee un entero con validación de rango"""
    while True:
        try:
            valor = int(input(prompt))
            if min_val is not None and valor < min_val:
                print(f"El valor debe ser mayor o igual a {min_val}")
                continue
            if max_val is not None and valor > max_val:
                print(f"El valor debe ser menor o igual a {max_val}")
                continue
            return valor
        except ValueError:
            print("Por favor, introduce un número entero válido")

# Uso
edad = leer_entero("Edad (0-120): ", 0, 120)
print(f"Edad registrada: {edad}")

# Función para decimales
def leer_decimal(prompt, decimales=2):
    """Lee un número decimal"""
    while True:
        try:
            valor = float(input(prompt))
            return round(valor, decimales)
        except ValueError:
            print("Por favor, introduce un número válido")

precio = leer_decimal("Precio del producto: ")
print(f"Precio: ${precio:.2f}")

# Función para booleanos
def leer_booleano(prompt):
    """Lee un valor booleano (s/n)"""
    while True:
        respuesta = input(f"{prompt} (s/n): ").lower()
        if respuesta in ['s', 'si', 'sí', 'y', 'yes']:
            return True
        elif respuesta in ['n', 'no']:
            return False
        else:
```

```
            print("Por favor, responde 's' o 'n'")

es_estudiante = leer_booleano("¿Eres estudiante?")
print(f"Estudiante: {es_estudiante}")
```

7.3 Entrada de Múltiples Valores

```
# Múltiples valores en una línea
print("Introduce tres números separados por espacios:")
entrada = input()
numeros = entrada.split()
print(f"Números como strings: {numeros}")

# Conversión a enteros
try:
    numeros_int = [int(n) for n in numeros]
    print(f"Números como enteros: {numeros_int}")
    print(f"Suma: {sum(numeros_int)}")
except ValueError:
    print("Error: Todos los valores deben ser números")

# Usando unpacking
print("\nIntroduce nombre, edad y ciudad separados por comas:")
entrada = input()
try:
    nombre, edad, ciudad = entrada.split(',')
    nombre = nombre.strip()
    edad = int(edad.strip())
    ciudad = ciudad.strip()
    print(f"Nombre: {nombre}, Edad: {edad}, Ciudad: {ciudad}")
except ValueError:
    print("Error: Formato incorrecto")

# Función para leer lista de números
def leer_lista_numeros(prompt, separador=','):
    """Lee una lista de números"""
    while True:
        entrada = input(prompt)
        try:
            numeros = [float(x.strip()) for x in
entrada.split(separador)]
            return numeros
        except ValueError:
            print(f"Error: Introduce números separados por
'{separador}'")

notas = leer_lista_numeros("Introduce las notas (separadas por comas):
")
promedio = sum(notas) / len(notas) if notas else 0
print(f"Notas: {notas}")
print(f"Promedio: {promedio:.2f}")
```

7.4 Entrada con Menús y Opciones

```
def mostrar_menu():
    """Muestra el menú de opciones"""
    print("\n" + "=" * 30)
    print("MENÚ PRINCIPAL")
    print("=" * 30)
    print("1. Opción A")
```

```
    print("2. Opción B")
    print("3. Opción C")
    print("0. Salir")
    print("=" * 30)

def leer_opcion(opciones_validas):
    """Lee una opción del menú"""
    while True:
        opcion = input("Selecciona una opción: ").strip()
        if opcion in opciones_validas:
            return opcion
        print(f"Opción no válida. Opciones disponibles: {',
'.join(opciones_validas)}")

# Uso del menú
while True:
    mostrar_menu()
    opcion = leer_opcion(['0', '1', '2', '3'])

    if opcion == '0':
        print("¡Hasta luego!")
        break
    elif opcion == '1':
        print("Has seleccionado la Opción A")
    elif opcion == '2':
        print("Has seleccionado la Opción B")
    elif opcion == '3':
        print("Has seleccionado la Opción C")

    input("\nPresiona Enter para continuar...")
```

7.5 Entrada Avanzada y Segura

```
import getpass
import sys
from typing import Optional, Union, List

# Entrada de contraseñas (oculta el texto)
# password = getpass.getpass("Introduce la contraseña: ")
# print(f"Contraseña tiene {len(password)} caracteres")

# Clase para gestionar entrada de datos
class InputManager:
    """Gestor avanzado de entrada de datos"""

    @staticmethod
    def leer_string(prompt: str, min_len: int = 0, max_len: int =
None) -> str:
        """Lee un string con validación de longitud"""
        while True:
            valor = input(prompt).strip()
            if len(valor) < min_len:
                print(f"Mínimo {min_len} caracteres")
                continue
            if max_len and len(valor) > max_len:
                print(f"Máximo {max_len} caracteres")
                continue
            return valor

    @staticmethod
```

```
    def leer_email(prompt: str = "Email: ") -> str:
        """Lee y valida un email"""
        import re
        patron = r'^[\w\.-]+@[\w\.-]+\.\w+$'

        while True:
            email = input(prompt).strip()
            if re.match(patron, email):
                return email
            print("Email inválido. Formato: usuario@dominio.com")

    @staticmethod
    def leer_telefono(prompt: str = "Teléfono: ") -> str:
        """Lee y valida un número de teléfono"""
        while True:
            telefono = input(prompt).strip()
            # Eliminar espacios y guiones
            telefono_limpio = telefono.replace(" ", "").replace("-",
"")

            if telefono_limpio.isdigit() and 9 <= len(telefono_limpio)
<= 15:
                return telefono_limpio
            print("Teléfono inválido. Debe contener entre 9 y 15
dígitos")

    @staticmethod
    def leer_fecha(prompt: str = "Fecha (DD/MM/AAAA): ") -> tuple:
        """Lee y valida una fecha"""
        while True:
            fecha = input(prompt).strip()
            try:
                partes = fecha.split('/')
                if len(partes) != 3:
                    raise ValueError

                dia = int(partes[0])
                mes = int(partes[1])
                año = int(partes[2])

                if not (1 <= dia <= 31 and 1 <= mes <= 12 and 1900 <=
año <= 2100):
                    raise ValueError

                return (dia, mes, año)
            except (ValueError, IndexError):
                print("Formato inválido. Usa DD/MM/AAAA")

    @staticmethod
    def leer_opcion_multiple(prompt: str, opciones: List[str]) -> str:
        """Lee una opción de una lista"""
        print(prompt)
        for i, opcion in enumerate(opciones, 1):
            print(f"  {i}. {opcion}")

        while True:
            try:
                seleccion = int(input("Selección: "))
                if 1 <= seleccion <= len(opciones):
                    return opciones[seleccion - 1]
```

```
                print(f"Selecciona un número entre 1 y
{len(opciones)}")
            except ValueError:
                print("Por favor, introduce un número")

# Ejemplos de uso
if __name__ == "__main__":
    manager = InputManager()

    # nombre = manager.leer_string("Nombre (3-20 caracteres): ", 3,
20)
    # email = manager.leer_email()
    # telefono = manager.leer_telefono()
    # fecha = manager.leer_fecha()
    # color = manager.leer_opcion_multiple("Elige un color:", ["Rojo",
"Verde", "Azul"])

    # print(f"\nDatos registrados:")
    # print(f"Nombre: {nombre}")
    # print(f"Email: {email}")
    # print(f"Teléfono: {telefono}")
    # print(f"Fecha: {fecha[0]:02d}/{fecha[1]:02d}/{fecha[2]}")
    # print(f"Color: {color}")
```

Ejercicios Propuestos

Ejercicio 1: Conversor de Unidades

Crea un programa que convierta entre diferentes unidades (longitud, peso, temperatura). Usa enumerados para las unidades, validación de entrada, y formateo adecuado de resultados.

Ejercicio 2: Analizador de Texto

Desarrolla un programa que analice un texto: cuenta palabras, caracteres, encuentra la palabra más larga, frecuencia de letras. Usa métodos de string y estructuras de datos apropiadas.

Ejercicio 3: Calculadora de Propinas

Implementa una calculadora que solicite el importe, número de personas, y porcentaje de propina. Debe manejar diferentes monedas usando enum y validar todas las entradas.

Ejercicio 4: Generador de Contraseñas

Crea un generador de contraseñas con diferentes niveles de seguridad. Permite al usuario elegir longitud, incluir mayúsculas, números, símbolos. Usa random y validación.

Ejercicio 5: Agenda de Contactos

Desarrolla una agenda simple que permita agregar, buscar y listar contactos. Usa diccionarios, validación de email/teléfono, y permite exportar a JSON.

Resumen de Buenas Prácticas en Python

1. **PEP 8**: Sigue las convenciones de estilo
    - snake_case para variables y funciones
    - PascalCase para clases
    - MAYUSCULAS para constantes
2. **Type Hints**: Usa anotaciones de tipo para claridad
3. `def funcion(param: str) -> int:`
4. **Docstrings**: Documenta módulos, clases y funciones
5. **F-strings**: Usa f-strings para formateo (Python 3.6+)
6. **Context Managers**: Usa `with` para manejo de recursos
7. **Valores por defecto**: Define valores por defecto en funciones
8. **Validación**: Siempre valida entrada del usuario
9. **EAFP**: "Es más fácil pedir perdón que permiso"
    - Usa try/except en lugar de muchas comprobaciones
10. **Comprensiones**: Usa list/dict/set comprehensions cuando sea apropiado
11. **Zen de Python**: Recuerda `import this`
    - Explícito es mejor que implícito
    - Simple es mejor que complejo
    - Legibilidad cuenta

Este tema establece las bases fundamentales para programar en Python. Es esencial dominar estos conceptos antes de avanzar a estructuras de control, funciones y programación orientada a objetos.