RETO 1
CONCEPTO GENERAL DE ELECTRONICA MENTEFACTOS E IAS

PROGRAMA: INGENIERIA MECATRONICA

ASIGNATURA: FUNDAMENTOS DE ELECTRONICA

1. Concepto teórico

En Python, la entrada de datos permite recibir información del usuario mediante la función input(), que siempre retorna un valor de tipo str (string).
La salida de datos se realiza principalmente con print(), que puede mostrar texto, variables y resultados.

Strings: cadenas de caracteres delimitadas por comillas simples (' '), dobles (" ") o triples (''' ''' / """ """).
Tipos de datos básicos:
- str: cadena de texto
- int: número entero
- float: número decimal
- bool: valores lógicos (True o False)

Casteo: conversión de un tipo de dato a otro, por ejemplo:
- int("5") convierte '5' a número entero 5
- float("3.14") convierte '3.14' a número decimal 3.14
- str(25) convierte el número 25 a cadena '25'

2. Instrucciones básicas para conocer el lenguaje y el tema

```
# Entrada de datos
nombre = input('Ingrese su nombre: ')
edad = input('Ingrese su edad: ')

# Salida de datos
print('Hola,', nombre)
print(f'Tienes {edad} años')

# Manejo de strings
```

```
texto = 'Hola Python'
print(texto.upper())   # Mayúsculas
print(texto.lower())   # Minúsculas
print(len(texto))      # Longitud de la cadena
print(texto[0])        # Primer carácter

# Tipos de datos
numero = 10
pi = 3.1416
es_mayor = True
print(type(numero))   # <class 'int'>
print(type(pi))       # <class 'float'>
print(type(es_mayor)) # <class 'bool'>

# Casteo
numero_str = '50'
numero_int = int(numero_str)
print(numero_int + 10)  # 60
```

3. Ejercicio práctico

Crea un programa que:
1. Pida al usuario su nombre y edad.
2. Muestre un mensaje diciendo: 'Hola [nombre], el próximo año tendrás [edad+1] años'.
3. Use casteo para poder sumar la edad.

4. Reto de nivel medio

Haz un programa que:
1. Pida tres datos: nombre completo, año de nacimiento y ciudad de residencia.
2. Calcule la edad actual (suponiendo el año actual como 2025).
3. Muestre en pantalla un mensaje del estilo: 'Hola [nombre], tienes [edad] años y vives en [ciudad]' en mayúsculas.
4. Indique cuántos caracteres tiene el nombre.

5. Reto de nivel alto

Desarrolla un programa que:
1. Pida al usuario nombre, apellido, año, mes y día de nacimiento.
2. Calcule la edad exacta en años usando la fecha actual del sistema.
3. Muestre un saludo personalizado con el formato: 'Hola, [Apellido, Nombre]. Naciste el [día/mes/año] y tienes [edad] años'.
4. Indique cuántas letras y cuántos espacios tiene el nombre completo.
5. Aplique casteo cuando sea necesario y valide que las edades sean números enteros.

Fase 2: operadores lógicos, aritméticos, booleanos y de comparación.

1. Concepto teórico

En Python, los operadores permiten realizar cálculos, comparaciones y evaluaciones lógicas.

- Operadores aritméticos: +, -, *, /, //, %, **
- Operadores de comparación: ==, !=, >, <, >=, <=
- Operadores booleanos y lógicos:
  - and → verdadero si ambas condiciones lo son.
  - or → verdadero si al menos una lo es.
  - not → invierte el valor lógico.

Los operadores de comparación y lógicos retornan siempre valores booleanos (True o False).

2. Instrucciones básicas para conocer el lenguaje y el tema

```
# Operadores aritméticos
a = 10
b = 3
print(a + b)   # 13
print(a - b)   # 7
print(a * b)   # 30
print(a / b)   # 3.333...
print(a // b)  # 3
print(a % b)   # 1
print(a ** b)  # 1000

# Operadores de comparación
x = 5
y = 10
print(x == y)  # False
print(x != y)  # True
print(x < y)   # True
print(x >= 5)  # True

# Operadores lógicos y booleanos
edad = 20
es_estudiante = True
```

```
print(edad > 18 and es_estudiante)  # True
print(edad < 18 or es_estudiante)   # True
print(not es_estudiante)            # False
```

3. Ejercicio práctico

Crea un programa que:
1. Pida dos números enteros.
2. Muestre la suma, resta, multiplicación, división y módulo.
3. Indique si el primer número es mayor, menor o igual que el segundo.

4. Reto de nivel medio

Realiza un programa que:
1. Pida la edad y si la persona es estudiante (True o False).
2. Determine si puede acceder a un descuento (la condición es: tener menos de 18 años o ser estudiante).
3. Muestra el resultado con un mensaje claro.

5. Reto de nivel alto

Desarrolla un programa que:
1. Pida tres números.
2. Calcule su promedio y determine si es mayor o igual a 10.
3. Verifique si todos son positivos usando and.
4. Verifique si al menos uno es par usando or.
5. Muestra mensajes explicativos para cada caso.

Fase 3: list, tuple, set, dict

1. Concepto teórico

En Python existen varias estructuras de datos para almacenar colecciones de elementos:

• Listas (list): Colecciones ordenadas, modificables y que permiten elementos duplicados. Se definen con corchetes [].
• Tuplas (tuple): Colecciones ordenadas, inmutables y que permiten elementos duplicados. Se definen con paréntesis ().
• Conjuntos (set): Colecciones no ordenadas, sin índices, no permiten elementos duplicados. Se definen con llaves {}.

• Diccionarios (dict): Colecciones no ordenadas de pares clave-valor, donde las claves son únicas. Se definen con llaves {} usando la sintaxis clave: valor.

2. Instrucciones básicas para conocer el lenguaje y el tema

```
# ===== LISTAS =====
frutas = ['manzana', 'banana', 'cereza']
print(frutas)
frutas.append('naranja')      # Agregar al final
frutas.insert(1, 'kiwi')      # Insertar en posición específica
frutas.remove('banana')       # Eliminar por valor
frutas.pop(0)                 # Eliminar por índice
frutas.sort()                 # Ordenar alfabéticamente
frutas.reverse()              # Invertir orden
print(frutas)
print(len(frutas))            # Longitud de la lista

# ===== TUPLAS =====
coordenadas = (10, 20, 30)
print(coordenadas)
# Las tuplas no se pueden modificar, pero sí desempaquetar:
x, y, z = coordenadas
print(x, y, z)

# ===== SETS =====
colores = {'rojo', 'verde', 'azul'}
colores.add('amarillo')       # Agregar elemento
colores.remove('rojo')        # Eliminar elemento
colores.discard('negro')      # No da error si no existe
print('verde' in colores)     # Comprobar existencia
otros_colores = {'rosa', 'azul'}
union = colores.union(otros_colores)
interseccion = colores.intersection(otros_colores)
print(union)
print(interseccion)

# ===== DICCIONARIOS =====
persona = {'nombre': 'Juan', 'edad': 30, 'ciudad': 'Madrid'}
print(persona['nombre'])      # Acceder a valor
persona['edad'] = 31          # Modificar valor
persona['profesion'] = 'Ingeniero'  # Agregar nuevo par clave-valor
del persona['ciudad']         # Eliminar clave
```

```
print(persona.keys())       # Obtener claves
print(persona.values())     # Obtener valores
print(persona.items())      # Obtener pares clave-valor
```

3. Ejercicio práctico

Crea un programa que:

1. Cree una lista de 5 frutas y las muestre ordenadas alfabéticamente.
2. Agregue una fruta al inicio y otra al final.
3. Elimine una fruta por nombre y otra por posición.
4. Muestre la lista final.

4. Reto de nivel medio

Realiza un programa que:

1. Pida al usuario los nombres de 5 ciudades y las guarde en un set.
2. Solicite otra ciudad y la agregue al set.
3. Cree otro set con 3 ciudades y muestre la unión e intersección de ambos conjuntos.
4. Elimine una ciudad si existe.

5. Reto de nivel alto

Desarrolla un programa que:

1. Cree un diccionario donde las claves sean nombres de estudiantes y los valores sean listas con sus calificaciones.
2. Permita agregar nuevos estudiantes y calificaciones.
3. Calcule el promedio de cada estudiante.
4. Muestre el estudiante con el promedio más alto.
5. Elimine a un estudiante por nombre.
6. Liste todos los estudiantes con sus calificaciones.

Fase 4: Estructuras de decisión if else elif y match

1. Concepto teórico y propiedades

Las estructuras de decisión permiten que el programa tome diferentes caminos según condiciones lógicas. En Python se utilizan principalmente if, elif y else, además de match (pattern matching) desde Python 3.10.

Propiedades clave:
• Encadenamiento de comparaciones: 1 < x < 10
• Anidación de condiciones (if dentro de if)
• Evaluación de verdad (truthy/falsy): valores no vacíos o distintos de cero suelen ser considerados True
• Operador condicional (expresión ternaria): a if condición else b
• Clausula else opcional en if; elif para múltiples ramas
• match-case para coincidencia estructural de patrones, con comodines y guardas (if)

2. Instrucciones básicas: if / elif / else

```
# Validación de umbral de sensor
valor = 72
if valor > 80:
    print('ALERTA: sobre-umbral')
elif 60 <= valor <= 80:
    print('RANGO SEGURO')
else:
    print('POR DEBAJO DEL UMBRAL')

# Expresión condicional (ternaria)
mensaje = 'OK' if valor >= 60 else 'BAJO'
print(mensaje)

# Encadenamiento de comparaciones
x = 7
if 1 < x < 10:
    print('x está entre 2 y 9')

# Anidación
if valor >= 60:
    if valor > 90:
        print('Crítico')
    else:
        print('Aceptable')
```

3. Instrucciones básicas: match / case (pattern matching)

match permite comparar un valor contra múltiples patrones y extraer partes de estructuras (tuplas, listas, dicts). Admite comodín '_' y '|' para alternativas. Las guardas 'if' refinan el patrón.

```
# Ejemplo 1: Menú simple con alternativas
op = 'iniciar'
match op:
    case 'iniciar' | 'start':
        print('Inicializando sistema')
    case 'detener' | 'stop':
        print('Deteniendo sistema')
    case _:
        print('Opción no reconocida')

# Ejemplo 2: Pattern matching sobre tuplas (x, y)
evento = ('medicion', 28.5)
match evento:
    case ('medicion', valor) if valor > 30:
        print('Temperatura alta:', valor)
    case ('medicion', valor):
        print('Temperatura ok:', valor)
    case _:
        print('Evento desconocido')

# Ejemplo 3: Patrones de diccionario
pkt = {'tipo': 'cmd', 'accion': 'reset', 'id': 42}
match pkt:
    case {'tipo': 'cmd', 'accion': accion, 'id': ident}:
        print(f'Comando {accion} para id {ident}')
    case _:
        print('Paquete no válido')
```

4. Ejercicio práctico

Crea un programa que solicite un valor de temperatura y clasifique el estado:

- > 80 → 'CRÍTICO'
- 60–80 → 'SEGURO'
- < 60 → 'BAJO'

Usa if/elif/else y la versión con expresión condicional para imprimir un resumen.

5. Reto de nivel medio

Implementa un menú de control por texto usando match con estas opciones: 'encender', 'apagar', 'reporte', 'config [clave]=[valor]'.
• Para 'reporte', muestra un estado simulado (voltaje, corriente, temperatura) con rangos y etiquetas usando if anidados.
• Para 'config', reconoce el patrón de cadena, extrae clave y valor con match y valida tipos (int/float/str) según la clave.
• Para cualquier otra entrada, responde 'comando inválido'.

6. Reto de nivel alto

Diseña un validador de paquetes para un sistema embebido. La entrada es un diccionario con diferentes formatos, por ejemplo:
{'tipo': 'sensor', 'id': 7, 'datos': [22.4, 22.6, 22.3]} o {'tipo': 'cmd', 'accion': 'calibrar', 'target': {'id': 7, 'eje': 'x'}}.

Requisitos:
• Usa match para distinguir los tipos ('sensor', 'cmd', 'alarma', etc.)
• En 'sensor', valida que 'datos' sea lista no vacía y calcula promedio y alerta (if con guardas).
• En 'cmd', extrae 'accion' y 'target'; si falta info, lanza mensaje de error.
• Si el paquete no coincide con ningún patrón, marca como 'paquete inválido'.

Fase 5: Loops for y while

1. Concepto teórico y propiedades

Los bucles permiten repetir bloques de código. En Python:
• for itera sobre secuencias (listas, tuplas, strings, rangos, diccionarios, sets).
• while repite mientras una condición sea verdadera.

Propiedades clave:
• range(inicio, fin, paso) para generar secuencias numéricas.
• enumerate(iterable) para obtener índice y valor.
• zip(a, b, ...) para iterar en paralelo.
• break para salir del bucle; continue para saltar a la siguiente iteración.
• else en bucles: se ejecuta si el bucle finaliza sin break.
• Iteración sobre dict: keys(), values(), items().

2. Instrucciones básicas: for

```
# Contadores y rangos
for i in range(0, 10, 2):
    print(i)

# enumerate y zip
sensores = ['T1', 'T2', 'H1']
lecturas = [22.4, 23.1, 45.0]
for idx, s in enumerate(sensores):
    print(idx, s)
for s, v in zip(sensores, lecturas):
    print(f'{s}: {v}')

# Iterar diccionarios
estado = {'voltaje': 12.1, 'corriente': 1.8}
for k, v in estado.items():
    print(k, v)

# else en for (solo se ejecuta si no hay break)
objetivo = 23.1
for v in lecturas:
    if v == objetivo:
        print('Encontrado')
        break
else:
    print('No encontrado')
```

3. Instrucciones básicas: while

```
# Bucle while con condición y control de salida
contador = 5
while contador > 0:
    print('Cuenta regresiva:', contador)
    contador -= 1
else:
    print('¡Despegue!')

# Validación de entrada con while y continue
intentos = 0
max_intentos = 3
while intentos < max_intentos:
    dato = input('Ingrese número (>0): ')
```

```
    try:
        n = int(dato)
        if n <= 0:
            print('Debe ser >0')
            intentos += 1
            continue
        print('Número válido:', n)
        break
    except ValueError:
        print('Entrada inválida')
        intentos += 1
else:
    print('Se agotaron los intentos')
```

4. Ejercicio práctico (Fase 2)

Escribe un programa que:

• Use for para imprimir los números del 1 al 50, mostrando 'Fizz' si es múltiplo de 3, 'Buzz' si es múltiplo de 5, y 'FizzBuzz' si es múltiplo de ambos.

• Use while para pedir contraseñas hasta acertar una clave predeterminada o agotar 3 intentos.

5. Reto de nivel medio (Fase 2)

Gestor de buffers: Dada una lista de paquetes con tamaños (bytes), recorre con for acumulando en un buffer de capacidad N. Cuando el siguiente paquete no quepa, envía el buffer (imprime contenido) y reinicia acumulador. • Usa continue para ignorar paquetes de tamaño 0 o negativos.

• Al final, si quedan paquetes sin enviar, envíalos.

• Prueba con N=100 y tamaños variados.

6. Reto de nivel alto (Fase 2)

Planificador de tareas con ventanas: Genera tareas con (id, prioridad, tiempo) y simula su ejecución por rondas.

Requisitos:

• Crea una lista de tareas (diccionarios o tuplas). En cada ronda (for), ejecuta solo tareas cuya prioridad sea >= umbral.

• Por cada tarea, reduce 'tiempo' en 1 (while interno). Si llega a 0, marca como completada y retírala del ciclo siguiente.

• Si en una ronda no se ejecutó ninguna tarea, usa el else del for para imprimir 'esperando nuevas tareas'.

• Termina cuando todas las tareas se completen.

Fase 6: Funciones

1. Concepto teórico

Las funciones son bloques de código reutilizables que realizan una tarea específica. Permiten estructurar el código, evitar repeticiones y mejorar la legibilidad.

Características clave:
• Se definen con la palabra clave def seguida del nombre y paréntesis.
• Pueden recibir parámetros (argumentos) y devolver valores con return.
• Pueden tener valores por defecto en los parámetros.
• Admiten argumentos posicionales, nombrados (*args para múltiples posicionales, **kwargs para múltiples nombrados).
• El ámbito de las variables puede ser local (dentro de la función) o global (fuera de cualquier función).
• Pueden incluir anotaciones de tipo (type hints) para documentar el tipo de datos esperado.

2. Instrucciones básicas para conocer el lenguaje y el tema

```
# Función simple
def saludar():
    print('Hola, bienvenido')
saludar()

# Función con parámetros
def suma(a, b):
    return a + b
resultado = suma(5, 3)
print(resultado)

# Función con valor por defecto
def potencia(base, exponente=2):
    return base ** exponente
print(potencia(5))     # 25
print(potencia(5, 3))  # 125

# Función con *args y **kwargs
def mostrar_datos(*args, **kwargs):
    print('Argumentos posicionales:', args)
```

```
    print('Argumentos nombrados:', kwargs)
mostrar_datos(1, 2, 3, nombre='Ana', edad=25)

# Ámbito de variables
x = 10
def cambiar():
    global x
    x = 20
cambiar()
print(x)  # 20
```

3. Ejercicio práctico

Crea una función llamada 'calcular_area_rectangulo' que reciba ancho y alto, y retorne el área. Pide los valores al usuario y muestra el resultado.

4. Reto de nivel medio

Crea una función llamada 'analizar_lista' que reciba una lista de números y retorne:

- El promedio.
- El número más alto.
- El número más bajo.
- La cantidad de elementos.

Solicita una lista al usuario, llama la función y muestra los resultados.

5. Reto de nivel alto

Diseña un sistema de registro usando funciones:

- 'registrar_usuario(nombre, edad, ciudad)': agrega un usuario a un diccionario global con ID autoincremental.
- 'mostrar_usuarios()': lista todos los usuarios registrados.
- 'buscar_usuario(id)': retorna los datos del usuario con ese ID.
- 'eliminar_usuario(id)': elimina un usuario por ID.

Crea un menú interactivo que permita ejecutar estas funciones y probar su funcionamiento.