# Soluciones al Taller de Fundamentos de Python

Alumna: Valentina Tapia
Institución: Universitaria de Colombia
Fecha: 05 de Junio de 2025
Este documento presenta las soluciones a los ejercicios propuestos en el "Taller - Fundamentos de Python", siguiendo la estructura de las tres secciones definidas: Variables, Operadores Relacionales y Operadores Lógicos. Para cada ejercicio, se incluye una breve descripción del problema, la solución implementada en código Python y una explicación concisa.

## Sección 1: Variables

### 1. Intercambio sin variable auxiliar

- **Problema:** Intercambiar los valores de dos variables a y b sin usar una variable extra.
- **Solución:**
  ```
  a = 5
  b = 10
  print(f"Antes: a={a}, b={b}")
  a, b = b, a # Intercambio de valores en Python
  print(f"Después: a={a}, b={b}")
  ```
- **Explicación:** Python facilita el intercambio de valores entre variables de manera concisa mediante la asignación de tuplas.

### 2. Suma de cuadrados

- **Problema:** Declarar dos variables a=5, b=3, y almacenar el resultado de a2+b2 en una tercera variable c.
- **Solución:**
  ```
  a_sq = 5
  b_sq = 3
  c_sq = a_sq**2 + b_sq**2
  print(f"a={a_sq}, b={b_sq}, c={c_sq}")
  ```
- **Explicación:** Se utilizan las variables y el operador de exponenciación (**) para calcular los cuadrados y la suma.

### 3. Conversión de tipos

- **Problema:** Convertir un número entero en string, luego a float y finalmente de

nuevo a int. Mostrar los resultados.

- **Solución:**
  num_int = 10
  print(f"Número entero original: {num_int} (tipo: {type(num_int)})")
  num_str = str(num_int)
  print(f"Convertido a string: '{num_str}' (tipo: {type(num_str)})")
  num_float = float(num_str)
  print(f"Convertido a float: {num_float} (tipo: {type(num_float)})")
  num_int_again = int(num_float)
  print(f"Convertido de nuevo a int: {num_int_again} (tipo: {type(num_int_again)})")

- **Explicación:** Se utilizan las funciones str(), float(), e int() para realizar las conversiones de tipo.

**4. Redondeo y formato**

- **Problema:** Redondear el número 17.8567 a dos decimales y convertir el resultado en string para imprimirlo como: "Resultado: 17.86".
- **Solución:**
  num_round = 17.8567
  rounded_num = round(num_round, 2)
  result_str = f"Resultado: {rounded_num}"
  print(f"Original: {num_round}, Formateado: \"{result_str}\"")

- **Explicación:** La función round() se usa para redondear, y las f-strings (f"") para formatear la salida.

**5. Concatenación de variables**

- **Problema:** Declarar dos variables nombre y edad, y construir un mensaje como: "Hola, mi nombre es Juan y tengo 25 años."
- **Solución:**
  nombre = "Juan"
  edad = 25
  mensaje = f"Hola, mi nombre es {nombre} y tengo {edad} años."
  print(mensaje)

- **Explicación:** Se utiliza una f-string para insertar los valores de las variables directamente en la cadena de texto.

**6. Cálculo de edad actual**

- **Problema:** Usando anio_actual y anio_nacimiento, calcular y mostrar la edad

actual.

- **Solución:**
  ```
  anio_actual = 2025
  anio_nacimiento = 2000
  edad_actual = anio_actual - anio_nacimiento
  print(f"Año actual={anio_actual}, Año nacimiento={anio_nacimiento}, Edad actual={edad_actual}")
  ```

- **Explicación:** Se realiza una simple resta entre el año actual y el año de nacimiento.

**7. Cuenta regresiva con variables**

- **Problema:** Simular una cuenta regresiva desde 10 hasta 0 usando una sola variable.
- **Solución:**
  ```
  count = 10
  while count >= 0:
      print(count)
      count -= 1
  ```

- **Explicación:** Se utiliza un bucle while para decrementar la variable count en cada iteración hasta llegar a 0.

**8. Valor absoluto sin usar abs()**

- **Problema:** Implementar una lógica para obtener el valor absoluto de un número entero sin usar abs().
- **Solución:**
  ```
  num_abs = -7
  print(f"Número original: {num_abs}")
  valor_absoluto = num_abs if num_abs >= 0 else -num_abs
  print(f"Valor absoluto: {valor_absoluto}")

  num_abs = 5
  print(f"Número original: {num_abs}")
  valor_absoluto = num_abs if num_abs >= 0 else -num_abs
  print(f"Valor absoluto: {valor_absoluto}")
  ```

- **Explicación:** Se usa un operador ternario (if-else en una línea) para cambiar el signo del número solo si es negativo.

**9. Incremento/decremento según paridad**

- **Problema:** Si una variable n es par, sumar 1. Si es impar, restarle 1. Imprimir el nuevo valor.
- **Solución:**

```
n_parity = 4
print(f"n={n_parity}")
if n_parity % 2 == 0:
    n_parity += 1
else:
    n_parity -= 1
print(f"Nuevo valor de n: {n_parity}")

n_parity = 7
print(f"n={n_parity}")
if n_parity % 2 == 0:
    n_parity += 1
else:
    n_parity -= 1
print(f"Nuevo valor de n: {n_parity}")
```

- **Explicación:** Se usa el operador módulo (%) para verificar la paridad (n % 2 == 0 para par) y luego se ajusta el valor.

**10. Máximo entre tres números**

- **Problema:** Dadas tres variables a, b y c, encontrar y mostrar cuál es la mayor sin usar max().
- **Solución:**

```
a_max = 15
b_max = 20
c_max = 12
print(f"a={a_max}, b={b_max}, c={c_max}")
max_val = a_max
if b_max > max_val:
    max_val = b_max
if c_max > max_val:
    max_val = c_max
print(f"El número mayor es: {max_val}")
```

- **Explicación:** Se inicializa una variable max_val con el primer número y se

compara secuencialmente con los otros, actualizando max_val si se encuentra un número mayor.

## Sección 2: Operadores Relacionales

### 11. Verificación de rango

- **Problema:** Verificar si un número entero x está entre 10 y 100 (inclusive). Imprimir el resultado booleano.
- **Solución:**
  x_range = 55
  in_range = 10 <= x_range <= 100
  print(f"x={x_range}, ¿Está entre 10 y 100? {in_range}")

- **Explicación:** Python permite encadenar operadores relacionales para verificar un rango de forma concisa.

### 12. Comparación de strings (ignorar mayúsculas)

- **Problema:** Dadas dos cadenas a y b, comprobar si son iguales ignorando mayúsculas/minúsculas.
- **Solución:**
  str1 = "Hola Mundo"
  str2 = "hola mundo"
  compare_strings = str1.lower() == str2.lower()
  print(f"'{str1}' y '{str2}', ¿Son iguales? {compare_strings}")

- **Explicación:** Se utiliza el método .lower() para convertir ambas cadenas a minúsculas antes de la comparación, asegurando que el caso no afecte el resultado.

### 13. Igualdad entre tres variables

- **Problema:** Verificar si tres variables distintas tienen el mismo valor.
- **Solución:**
  var1 = 7
  var2 = 7
  var3 = 7
  all_equal = (var1 == var2) and (var2 == var3)
  print(f"var1={var1}, var2={var2}, var3={var3}, ¿Son todos iguales? {all_equal}")

- **Explicación:** Se utilizan operadores de igualdad (==) y el operador lógico and para comprobar si las tres variables son iguales entre sí.

**14. Presencia en lista**

- **Problema:** Declarar una lista de números y verificar si un número x está contenido en ella.
- **Solución:**
  my_list = [1, 5, 8, 12, 20]
  x_list = 8
  is_present = x_list in my_list
  print(f"Lista={my_list}, Número a buscar={x_list}, ¿Está presente? {is_present}")

- **Explicación:** El operador in se utiliza para verificar la pertenencia de un elemento en una secuencia (como una lista).

**15. Divisibilidad por 3 y 5**

- **Problema:** Dado un número n, determinar si es divisible simultáneamente por 3 y 5.
- **Solución:**
  n_divisible = 30
  divisible_by_3_and_5 = (n_divisible % 3 == 0) and (n_divisible % 5 == 0)
  print(f"n={n_divisible}, ¿Es divisible por 3 y 5? {divisible_by_3_and_5}")

- **Explicación:** Se utiliza el operador módulo (%) para verificar si el residuo de la división por 3 y por 5 es cero, y el operador and para asegurar que ambas condiciones se cumplan.

**16. Números en orden estricto**

- **Problema:** Verificar si tres variables a, b, c están en orden ascendente estricto (a<b<c).
- **Solución:**
  a_order = 10
  b_order = 20
  c_order = 30
  in_strict_order = (a_order < b_order) and (b_order < c_order)
  print(f"a={a_order}, b={b_order}, c={c_order}, ¿Están en orden estricto? {in_strict_order}")

- **Explicación:** Al igual que con el rango, Python permite encadenar operadores de comparación para verificar el orden estricto.

**17. Comparación doble**

- **Problema:** Dado un número x, y dos límites a y b, verificar si x está estrictamente entre a y b.
- **Solución:**
  x_between = 7
  a_limit = 5
  b_limit = 10
  is_strictly_between = (x_between > a_limit) and (x_between < b_limit)
  print(f"x={x_between}, a={a_limit}, b={b_limit}, ¿Está estrictamente entre a y b? {is_strictly_between}")

- **Explicación:** Se usan los operadores > y < junto con and para establecer límites estrictos.

**18. Relación proporcional**

- **Problema:** Verificar si una variable a es exactamente el doble de otra variable b.
- **Solución:**
  a_prop = 14
  b_prop = 7
  is_double = (a_prop == 2 * b_prop)
  print(f"a={a_prop}, b={b_prop}, ¿a es el doble de b? {is_double}")

- **Explicación:** Se compara la variable a con el doble de la variable b usando el operador de igualdad.

**19. Cambio de signo si negativo**

- **Problema:** Si una variable x es negativa, convertirla en positiva. Si ya es positiva, dejarla igual.
- **Solución:**
  x_sign = -10
  print(f"Original: {x_sign}")
  x_sign_changed = x_sign if x_sign >= 0 else -x_sign
  print(f"Después: {x_sign_changed}")

  x_sign = 5
  print(f"Original: {x_sign}")
  x_sign_changed = x_sign if x_sign >= 0 else -x_sign
  print(f"Después: {x_sign_changed}")

- **Explicación:** Se utiliza un operador ternario: si x es mayor o igual a 0, se

mantiene, de lo contrario, se le cambia el signo.

**20. Detección de tipo**

- **Problema:** Dada una variable x, imprimir si es un entero (int), flotante (float) o cadena (str).
- **Solución:**
  x_type_int = 100
  x_type_float = 3.14
  x_type_str = "Python"

  print(f"Variable {x_type_int}: tipo {type(x_type_int).__name__}")
  print(f"Variable {x_type_float}: tipo {type(x_type_float).__name__}")
  print(f"Variable '{x_type_str}': tipo {type(x_type_str).__name__}")

- **Explicación:** La función type() devuelve el tipo de un objeto, y .__name__ obtiene el nombre de ese tipo como cadena.

## Sección 3: Operadores Lógicos

**21. ¿Puede votar?**

- **Problema:** Verificar si una persona puede votar. Debe tener al menos 18 años y tener documento de identidad (True o False).
- **Solución:**
  edad_votar = 19
  tiene_documento = True
  puede_votar = (edad_votar >= 18) and tiene_documento
  print(f"Edad={edad_votar}, Tiene documento={tiene_documento}, ¿Puede votar? {puede_votar}")

- **Explicación:** Se combinan dos condiciones con el operador lógico and: la edad debe ser 18 o más, Y debe tener documento.

**22. Control de acceso lógico**

- **Problema:** Una persona puede entrar si: Tiene pase VIP, O Paga entrada y no está ebria.
- **Solución:**
  tiene_pase_vip = False
  paga_entrada = True
  esta_ebria = False
  puede_entrar = tiene_pase_vip or (paga_entrada and not esta_ebria)

print(f"Pase VIP={tiene_pase_vip}, Paga entrada={paga_entrada}, Ebria={esta_ebria}, ¿Puede entrar? {puede_entrar}")

- **Explicación:** Se utiliza el operador or para la condición principal (VIP o pago válido), y and junto con not para la condición de pago y sobriedad.

**23. Validación XOR**

- **Problema:** Dadas dos condiciones booleanas cond1 y cond2, verificar si exactamente una es verdadera (operación XOR lógica).
- **Solución:**
  cond1 = True
  cond2 = False
  xor_result = (cond1 and not cond2) or (not cond1 and cond2)
  print(f"cond1={cond1}, cond2={cond2}, ¿Exactamente una es verdadera? {xor_result}")

- **Explicación:** La operación XOR (o exclusivo) se implementa con una disyunción (or) de dos conjunciones (and): (cond1 y no cond2) O (no cond1 y cond2).

**24. Validación compuesta múltiple**

- **Problema:** Dado un número n, verificar si es mayor que 0 y (múltiplo de 2 o de 3).
- **Solución:**
  n_multiple = 6
  is_valid_multiple = (n_multiple > 0) and ((n_multiple % 2 == 0) or (n_multiple % 3 == 0))
  print(f"n={n_multiple}, ¿Es mayor que 0 y (múltiplo de 2 o 3)? {is_valid_multiple}")

- **Explicación:** Se combinan una condición de rango (> 0) con una disyunción de divisibilidad (% 2 == 0 o % 3 == 0), usando and para unir ambas partes.

**25. Condición contradictoria**

- **Problema:** Escribir una condición lógica que nunca se cumple (una contradicción), como x>5 and x<1.
- **Solución:**
  x_contradiction = 3 # El valor de x no importa, la condición es siempre False
  contradictory_condition = (x_contradiction > 5) and (x_contradiction < 1)
  print(f"Condición contradictoria (ejemplo con x={x_contradiction}): {contradictory_condition}")

- **Explicación:** Una contradicción ocurre cuando se requiere que una variable

cumpla dos condiciones mutuamente excluyentes al mismo tiempo, como ser mayor que 5 y menor que 1.

**26. Negación lógica**

- **Problema:** Dada una condición x>10, reescribirla usando not para que tenga el mismo efecto lógico.
- **Solución:**
  x_negation = 15
  negation_effect_true = (x_negation > 10)
  negation_rewritten_true = not (x_negation <= 10) # Equivalente a x > 10
  print(f"Negación lógica (x={x_negation}): Original ('x > 10'):
  {negation_effect_true}, Reescribiendo con 'not': {negation_rewritten_true}")

  x_negation = 8
  negation_effect_false = (x_negation > 10)
  negation_rewritten_false = not (x_negation <= 10)
  print(f"Negación lógica (x={x_negation}): Original ('x > 10'):
  {negation_effect_false}, Reescribiendo con 'not': {negation_rewritten_false}")

- **Explicación:** Negar la condición opuesta (not (x <= 10)) es lógicamente equivalente a la condición original (x > 10).

**27. Aprobación de estudiante**

- **Problema:** Verificar si un estudiante aprueba: nota >=3.0 y asistencia >=80.
- **Solución:**
  nota = 3.5
  asistencia = 85
  aprueba_estudiante = (nota >= 3.0) and (asistencia >= 80)
  print(f"Nota={nota}, Asistencia={asistencia}, ¿Aprueba el estudiante?
  {aprueba_estudiante}")

- **Explicación:** Ambas condiciones (nota y asistencia) deben ser verdaderas para que el estudiante apruebe, lo que se modela con un and.

**28. Simulación de alarma**

- **Problema:** Si la temperatura es menor a 0 o mayor a 38 y la humedad es mayor a 80%, se activa una alarma.
- **Solución:**
  temperatura = -5
  humedad = 90

```
activa_alarma = ((temperatura < 0) or (temperatura > 38)) and (humedad > 80)
print(f"Temperatura={temperatura}, Humedad={humedad}, ¿Se activa la alarma? {activa_alarma}")
```

- **Explicación:** Se usa or para la condición de temperatura (extremos) y and para combinarla con la condición de humedad alta. Los paréntesis son cruciales para el orden de evaluación.

**29. Contraseña segura**

- **Problema:** Verificar si una contraseña cumple con: Longitud mayor a 8 caracteres y Contiene al menos un número.
- **Solución:**

```
password = "MiContraseña123"
longitud_valida = len(password) > 8
contiene_numero = any(char.isdigit() for char in password)
contrasena_segura = longitud_valida and contiene_numero
print(f"Contraseña='{password}', ¿Es segura? {contrasena_segura}")
```

- **Explicación:** len() verifica la longitud. La expresión any(char.isdigit() for char in password) itera sobre cada carácter y retorna True si al menos uno es un dígito. Ambas condiciones se combinan con and.

**30. Doble negación lógica**

- **Problema:** Evaluar una expresión como: "No es falso que no tenga acceso" usando operadores lógicos. Traducir eso a Python.
- **Solución:**

```
# La expresión "No es falso que no tenga acceso" se traduce a:
# not (False (not (tiene_acceso)))
# Simplificando: not (not (tiene_acceso))
# Que es equivalente a: tiene_acceso

tiene_acceso = True
expresion_logica = not (not (tiene_acceso))
print(f"'No es falso que no tenga acceso' (Acceso: {tiene_acceso}) -> Resultado: {expresion_logica}")

tiene_acceso = False
expresion_logica = not (not (tiene_acceso))
print(f"'No es falso que no tenga acceso' (Acceso: {tiene_acceso}) -> Resultado:
```

{expresion_logica}")

- **Explicación:** La doble negación (not (not x)) es lógicamente equivalente a la afirmación original (x). Si "no tengo acceso" es False, entonces "no es falso que no tengo acceso" significa que tengo acceso.