

3

Tipos de datos y operadores

Los programas informáticos gestionan información en forma de datos. En este capítulo aprenderemos *cómo definir y manipular datos en Python, así como los operadores y conversiones de tipos* disponibles en el lenguaje.

3.1. Tipos de datos

En Python, cada dato tiene un tipo asociado. Los tipos de datos más básicos del lenguaje se muestran en la Tabla 1.

Tipo	Representa	Ejemplo (literales del dato)
int	Números enteros	-10972, -13, 0, 5, 17, 209
float	Números reales	-1.0E34, -32.17, 0.0, 1.0E-12, 0.47
str	Cadenas de caracteres	"Hola", 'Python', "" (cadena vacía)
bool	Valores lógicos	True, False

Tabla 1: Tipos de datos más básicos en Python.

Para conocer el tipo de un dato en Python se utiliza la función `type()`:

```
1 print(type(42))      # <class 'int'>
2 print(type(3.1416))  # <class 'float'>
3 print(type("A"))     # <class 'str'>
4 print(type(True))    # <class 'bool'>
```

Además de estos tipos básicos, Python incluye un valor especial llamado `None`, que se utiliza para representar la *ausencia de valor*. Su tipo es `NoneType` (no aparece en la Tabla 1 porque no es un tipo en el sentido de los anteriores: `None` es el *único* valor que existe de `NoneType`). Es habitual usar `None` para inicializar variables que aún no tienen un valor significativo, o como valor de retorno de una función que no devuelve nada

explícitamente.

```
1 resultado = None
2 print(type(resultado)) # <class 'NoneType'>
```

Una nota sobre el tamaño en memoria. En el módulo estándar `sys` existe la función `getsizeof()`, que devuelve el número de bytes que ocupa un objeto en memoria. Conviene saber, sin embargo, que el valor obtenido no se corresponde con la representación matemática del dato: por ejemplo, en CPython, un entero pequeño como 0 ocupa 28 bytes, no 4 u 8, porque cada objeto de Python lleva consigo metadatos (tipo, contador de referencias, etc.). Aun así, es útil para hacerse una idea del coste relativo de los distintos tipos:

```
1 import sys
2
3 print(sys.getsizeof(42))      # tamaño de un entero
4 print(sys.getsizeof(3.1416)) # tamaño de un flotante
5 print(sys.getsizeof('A'))    # tamaño de una cadena de un
↪   carácter
6 print(sys.getsizeof(True))   # tamaño de un booleano
```

Python incluye muchas funcionalidades en módulos estándar para mantener el lenguaje ligero. Cuando se necesita acceder a herramientas adicionales, como en el ejemplo anterior, se usa `import` para traer módulos al espacio de nombres del programa.

3.2. Operaciones

Para trabajar con datos, Python proporciona operadores que permiten hacer cálculos y comparaciones. Existen distintos tipos de operadores, que se describen a continuación.

3.2.1. Operadores de comparación

Los operadores de comparación se usan para comparar dos valores y devuelven un resultado de tipo `bool` (`True` o `False`). En la Tabla 2 se muestran todos estos operadores, su significado, un ejemplo y el resultado de evaluar dicha operación.

Veamos un ejemplo:

```
1 a, b = 10, 20 # define a=10 y b=20 simultaneamente
2 print(a < b)  # True
3 print(a == b) # False
4 print(a >= b) # False
```

Operador	Significado	Ejemplo	Resultado
<	Menor que	3 < 5	True
<=	Menor o igual que	5 <= 5	True
>	Mayor que	10 > 3	True
>=	Mayor o igual que	4 >= 8	False
==	Igual a	10 == 10	True
!=	Distinto de	7 != 3	True

Tabla 2: Operadores de comparación en Python.

Operador	Nombre	Ejemplo	Resultado
+	Suma	3 + 4	7
-	Resta	10 - 5	5
*	Multiplicación	4 * 6	24
/	División (real)	9 / 2	4.5
//	División entera	9 // 2	4
%	Módulo (resto)	9 % 2	1
**	Exponente	2 ** 3	8

Tabla 3: Operadores aritméticos en Python.

3.2.2. Operadores aritméticos

Los operadores aritméticos permiten realizar operaciones matemáticas básicas. En la Tabla 3 se muestran los operadores aritméticos disponibles, con un ejemplo y el resultado.

Un ejemplo de uso:

```

1 a, b = 27, 5
2 print(a / b)    # 5.4 (división real)
3 print(a // b)   # 5 (división entera)
4 print(a % b)    # 2 (resto de la división)
5 print(3 ** 2)   # 9 (3 elevado a 2)

```

3.2.3. Operadores lógicos

Los operadores lógicos trabajan con valores booleanos para construir expresiones lógicas. En la Tabla 4 se muestran estos operadores, su significado, un ejemplo y el resultado.

El operador **and** devuelve **True** si *ambos* operandos son verdaderos, y **False** en cualquier otro caso. En cambio, **or** devuelve **True** si *al menos uno* de los operandos es verdadero, y **False** solo cuando ambos son falsos. Las tablas de verdad de **and** y **or** son:

3. Tipos de datos y operadores

Operador	Significado	Ejemplo	Resultado
not	Negación ($\neg$)	not True	False
and	Conjunción ($\wedge$)	True and False	False
or	Disyunción ($\vee$)	True or False	True

Tabla 4: Operadores lógicos en Python.

<i>a</i>	<i>b</i>	<i>a and b</i>	<i>a</i>	<i>b</i>	<i>a or b</i>
True	True	True	True	True	True
True	False	False	True	False	True
False	True	False	False	True	True
False	False	False	False	False	False

Un ejemplo de uso:

```
1 a, b = True, False
2 print(not a)      # False
3 print(a and b)    # False
4 print(a or b)     # True
```

3.2.4. Operadores de asignación

Los operadores de asignación permiten modificar el valor de una variable. La Tabla 5 muestra los más habituales.

Operador	Ejemplo	Equivalente a
=	x = 5	Asigna 5 a x
+=	x += 3	x = x + 3
-=	x -= 2	x = x - 2
*=	x *= 4	x = x * 4
/=	x /= 2	x = x / 2

Tabla 5: Operadores de asignación en Python.

Un ejemplo de uso:

```
1 x = 10
2 x += 5
3 print(x)    # 15
```

3.3. Prioridad de operadores

Cuando una expresión contiene varios operadores, es importante saber en qué orden se evalúan. Python sigue un conjunto de reglas de *prioridad* (también llamada *precedencia*) que determinan qué operación se ejecuta primero.

A continuación se describe el orden de evaluación, de mayor a menor prioridad:

- (1) Paréntesis (): tienen la mayor prioridad. Cualquier operación dentro de paréntesis se evalúa primero, sin importar qué otros operadores estén presentes. Permiten cambiar el orden de evaluación cuando se desee:

```
1 resultado = (3 + 5) * 2
2 # Primero se evalúa (3 + 5), y luego el resultado se
↪   multiplica por 2.
3 print(resultado) # 16
```

- (2) Exponente **: se evalúa antes que multiplicaciones, divisiones, sumas o restas:

```
1 resultado = 2 ** 3 + 4
2 # Primero se evalúa 2 ** 3 (que es 8), y luego se le suma
↪   4.
3 print(resultado) # 12
```

- (3) Multiplicación *, división /, división entera //, módulo %: tienen la misma prioridad y se evalúan de izquierda a derecha:

```
1 resultado = 4 * 3 // 2
2 # Primero se evalúa 4 * 3 (que es 12), y luego 12 // 2 (
↪   que es 6).
3 print(resultado) # 6
```

- (4) Suma + y resta -: prioridad menor que las multiplicativas; se evalúan de izquierda a derecha:

```
1 resultado = 5 + 2 * 3
2 # Primero se evalúa 2 * 3 (que es 6), y luego 5 + 6.
3 print(resultado) # 11
```

- (5) Operadores de comparación (<, <=, >, >=, ==, !=): se evalúan después de las operaciones aritméticas y devuelven un valor booleano:

```
1 resultado = 5 + 2 * 3 < 20
2 # Primero se evalúan las operaciones aritmeticas
3 # y al final la comparación.
4 print(resultado) # True
```

3. Tipos de datos y operadores

- (6) Negación lógica **not**: tiene mayor prioridad que **and** y **or**, pero menor que las operaciones aritméticas y de comparación.
- (7) Conjunción lógica **and**: prioridad menor que **not**.
- (8) Disyunción lógica **or**: la menor prioridad de los operadores lógicos:

```
1 resultado = 5 + 2 * 3 < 20 and 3 > 2
2 # Primero las operaciones aritmeticas, luego las
  ↪ comparaciones,
3 # y finalmente la conjuncion lógica.
4 print(resultado) # True
```

Cuando varios operadores tienen la misma prioridad, Python evalúa la expresión de izquierda a derecha:

```
1 resultado = 5 + 3 - 2
2 # Primero se evalua 5 + 3, y luego se resta 2.
3 print(resultado) # 6
```

Si en algún momento tienes dudas sobre el orden de evaluación, lo más sensato es usar paréntesis explícitamente para garantizar el orden deseado:

```
1 resultado = (5 + 3) * 2
2 # Aquí, primero se evalua 5 + 3, y luego se multiplica por 2.
3 print(resultado) # 16
```

La Tabla 6 resume las reglas anteriores.

Prioridad	Operador	Descripción
1	()	Paréntesis (más alta prioridad)
2	**	Exponente
3	*, /, //, %	Multiplicación, división, división entera, módulo
4	+, -	Suma y resta
5	<, <=, >, >=, ==, !=	Operadores de comparación
6	not	Negación lógica
7	and	Conjunción lógica
8	or	Disyunción lógica (más baja prioridad)

Tabla 6: Prioridad de los operadores básicos de Python (de mayor a menor).

3.4. Conversión de tipos

En Python los tipos de las variables son dinámicos, lo que significa que no es necesario declararlos. Sin embargo, en muchos casos hay que convertir un dato de un tipo a otro:

bien para que las operaciones tengan sentido, bien para garantizar que los valores se ajusten a un formato concreto. Existen dos tipos de conversión: *implícita* y *explícita*.

3.4.1. Conversión implícita

Python realiza automáticamente algunas conversiones de tipo cuando los operandos involucrados son compatibles, especialmente entre tipos numéricos. Este proceso se conoce como *conversión implícita*. Por ejemplo, al sumar un entero con un flotante, Python convierte automáticamente el entero a flotante para que la operación sea coherente:

```
1 x = 5          # entero
2 y = 2.3        # flotante
3 resultado = x + y  # Python convierte x a flotante
↪      automaticamente
4 print(resultado)  # 7.3
```

En este caso, `x` es un entero e `y` es un flotante, pero Python convierte `x` a flotante antes de realizar la suma, lo que evita errores y garantiza que el resultado sea correcto.

3.4.2. Conversión explícita

En otros casos, el programador debe indicar explícitamente cómo realizar la conversión. Esto se hace usando funciones de conversión como `int()`, `float()` o `str()`. La *conversión explícita* es necesaria cuando el tipo de origen no es compatible de forma implícita con el tipo de destino.

Por ejemplo, para convertir un flotante a entero (con truncamiento de la parte decimal):

```
1 x = 5.9
2 y = int(x)    # convierte x a entero, truncando la parte
↪      decimal
3 print(y)      # 5
```

O para convertir un entero a cadena de texto:

```
1 x = 10
2 y = str(x)    # convierte x a cadena de texto
3 print(y)      # '10'
```

3.4.3. Conversiones de tipo más comunes

Las conversiones más habituales al trabajar con datos en Python son:

- **De int a float:** con `float()`. Por ejemplo:

```
1 entero = 7
2 flotante = float(entero)
3 print(flotante)  # 7.0
```

3. Tipos de datos y operadores

- **De float a int:** con `int()`, que trunca (no redondea) la parte decimal:

```
1 | flotante = 3.75
2 | entero = int(flotante)
3 | print(entero)    # 3
```

- **De cadena a número:** con `int()` o `float()`, siempre que la cadena tenga el formato adecuado:

```
1 | texto_entero = "123"
2 | texto_flotante = "45.67"
3 |
4 | numero_entero = int(texto_entero)
5 | numero_flotante = float(texto_flotante)
6 |
7 | print(numero_entero)    # 123
8 | print(numero_flotante) # 45.67
```

- **De número a cadena:** con `str()`. Útil cuando se quiere mostrar un número en un mensaje o concatenarlo con otro texto:

```
1 | numero = 25
2 | cadena = str(numero)
3 | print(cadena)    # '25'
```

3.5. Cadenas formateadas (f-strings)

Concatenar cadenas con valores de variables usando el operador `+` requiere que todos los operandos sean cadenas: si se intenta concatenar una cadena con un entero, Python lanza un `TypeError`. Por ejemplo:

```
1 | edad = 25
2 | print("Tienes " + edad + " años")    # TypeError: can only
↪ concatenate str
```

Para resolver esto, hay que convertir explícitamente el entero a cadena con `str()`:

```
1 | print("Tienes " + str(edad) + " años")
```

Sin embargo, Python ofrece una forma más cómoda y legible: las cadenas formateadas o *f-strings* (de *formatted string literals*). Una f-string se identifica por la `f` delante de las comillas y permite incrustar expresiones entre llaves `{ }`:

```
1 | nombre = "Alicia"
2 | edad = 25
3 | print(f"Hola, {nombre}, tienes {edad} años")
4 | # Salida: Hola, Alicia, tienes 25 años
```

Dentro de las llaves se puede escribir cualquier expresión válida de Python, no solo nombres de variables:

```
1 print(f"En 5 años, {nombre} tendrá {edad + 5} años")
```

Las f-strings también permiten especificar el formato de presentación del valor mediante el carácter `:` seguido de una especificación. Por ejemplo, para mostrar un número con 2 decimales:

```
1 PI = 3.1415926535
2 print(f"Pi con 2 decimales es {PI:.2f}")
3 # Salida: Pi con 2 decimales es 3.14
```

A lo largo del libro usaremos f-strings de forma sistemática, ya que producen un código más claro que la concatenación con `+` y `str()`. En el Capítulo 9 se profundizará en otras opciones de formato.

3.6. Manejo de errores en la conversión

Las conversiones entre tipos pueden fallar en tiempo de ejecución, lo que generaría una *excepción* que detendría el programa si no se trata adecuadamente. Para que el programa pueda recuperarse y dar una respuesta útil al usuario en lugar de detenerse abruptamente, Python proporciona el mecanismo de *manejo de excepciones* mediante los bloques `try`, `except`, `else`, `finally` y la sentencia `raise`.

Los errores más comunes al convertir datos son:

Error de tipo (TypeError). Se produce cuando se intenta convertir un valor a un tipo de dato incompatible. Por ejemplo, convertir el objeto nulo `None` a entero:

```
1 num = int(None) # TypeError: int() argument must be a string
↪ ...
```

Error de valor (ValueError). Se produce cuando el valor a convertir no tiene un formato válido para el tipo de destino. Por ejemplo, `int("12.5")` y `float("abc")` generan ambos un `ValueError`, ya que ninguna de las dos cadenas representa un número válido para el tipo solicitado:

```
1 num = int("12.5") # ValueError: invalid literal for int()
↪ with base 10
```

3.6.1. Captura específica con `try / except`

El bloque `try` contiene la operación que puede fallar; el bloque `except` se ejecuta *solo* si dentro del `try` se ha producido el error indicado. La práctica recomendada es indicar siempre el tipo de excepción que se quiere capturar, para no enmascarar errores no esperados.

```
1 entrada = input("Introduce un número entero: ")
2
3 try:
4     numero = int(entrada)
5     print(f"El número introducido es {numero}")
6 except ValueError:
7     print(f"Error: '{entrada}' no es un número válido.")
```

Si el usuario escribe 42, la conversión funciona y se imprime `El número introducido es 42`. Si escribe hola, el bloque `except ValueError` atrapa la excepción y se imprime el mensaje de error sin que el programa se detenga.

3.6.2. Captura genérica con `except Exception as e`

A veces interesa capturar un conjunto amplio de errores. La forma recomendada es `except Exception as e`: se atrapan todas las excepciones derivadas de `Exception` y, además, se obtiene una referencia `e` al objeto de error, lo que permite mostrar información útil sobre lo ocurrido:

```
1 entrada = input("Introduce un número entero: ")
2
3 try:
4     numero = int(entrada)
5     print(f"El número introducido es {numero}")
6 except Exception as e:
7     print(f"Se ha producido un error: {e}")
```

Es preferible `except Exception` a `except:` a secas. Esta última forma captura *absolutamente todo*, incluyendo `KeyboardInterrupt` (Ctrl+C) y `SystemExit`, lo que generalmente es indeseable: el usuario no podría siquiera abortar un programa colgado.

3.6.3. Cláusula `else`: código sin error

La cláusula `else` se ejecuta *solo si* el bloque `try` termina sin excepciones. Es útil para separar el código que puede fallar (en `try`) del que solo tiene sentido cuando todo ha ido bien (en `else`):

```
1 entrada = input("Introduce un número entero: ")
2
3 try:
4     numero = int(entrada)
5 except ValueError:
6     print(f"Error: '{entrada}' no es un número válido.")
7 else:
8     print(f"La conversión fue exitosa: {numero}")
```

3.6.4. Cláusula finally: código que siempre se ejecuta

La cláusula `finally` se ejecuta *siempre*, haya o no haya habido excepción. Es ideal para liberar recursos (cerrar ficheros, conexiones de red, bloqueos) sin importar si la operación tuvo éxito:

```
1 entrada = input("Introduce un número entero: ")
2
3 try:
4     numero = int(entrada)
5     print(f"Doble: {numero * 2}")
6 except ValueError:
7     print(f"Error: '{entrada}' no es un número válido.")
8 finally:
9     print("Operación finalizada (con o sin error).")
```

3.6.5. Lanzar errores con raise

A veces el programa detecta una situación errónea y debe avisar de ella. La sentencia `raise` permite *lanzar* explícitamente una excepción del tipo deseado, opcionalmente con un mensaje:

```
1 edad = int(input("Edad: "))
2 if edad < 0:
3     raise ValueError(f"La edad no puede ser negativa: {edad}")
4 print(f"Edad válida: {edad}")
```

Esta excepción puede capturarse después con un `try/except` en cualquier punto del código que llame a esta porción de programa, lo que permite separar la *detección* del problema de su *tratamiento*.

3.7. Ejercicios

EJERCICIO 3.1

Escribe un programa en Python que calcule el resultado de la siguiente expresión:

$$\text{resultado} = (a + b) \cdot c - \frac{a}{b}$$

El programa debe solicitar al usuario los valores de a , b y c y mostrar el resultado. Controla el error de división entre cero si $b = 0$.

Ejemplo de ejecución esperada

```
Introduce el valor de a: 10
Introduce el valor de b: 5
Introduce el valor de c: 2
El resultado de la expresión es 28.0
```

Solución (girar para leer)

```
1 try:
2     a = float(input("Introduce el valor de a: "))
3     b = float(input("Introduce el valor de b: "))
4     c = float(input("Introduce el valor de c: "))
5     # Calculamos la expresión
6     resultado = (a + b) * c - (a / b)
7
8     # Mostramos el resultado
9     print("El resultado de la expresión es", resultado)
10
11 except ZeroDivisionError:
12     print("Error: No se puede dividir por cero.")
13 except ValueError:
14     print("Error: Por favor, introduce valores numé-
15         ricos válidos.")
```

Código 3.1: Solución al Ejercicio 3.1.

EJERCICIO 3.2

Escribe un programa en Python que convierta una cantidad en kilómetros a metros, centímetros y milímetros. El programa debe solicitar al usuario la cantidad en kilómetros y mostrar los resultados.

Ejemplo de ejecución esperada

```
Introduce la cantidad en kilómetros: 3.5
3.5 km equivale a:
3500.0 metros
350000.0 centímetros
3500000.0 milímetros
```

Solución (girar para leer)

```

1 kilómetros = float(input("Introduce la cantidad en kiló
2 metros = kilómetros * 1000
3 centímetros = kilómetros * 100000
4 milímetros = kilómetros * 1000000
5 print(kilómetros, "km equivale a:")
6 print(metros, "metros")
7 print(centímetros, "centímetros")
8 print(milímetros, "milímetros")

Código 3.2: Solución al Ejercicio 3.2.
```

3. Tipos de datos y operadores

EJERCICIO 3.3

Escribe un programa en Python que realice las siguientes operaciones entre un número entero y un número flotante:

- Suma
- Multiplicación
- División
- División entera
- Módulo

El programa debe solicitar al usuario los dos números, uno entero y otro flotante, y mostrar los resultados.

Ejemplo de ejecución esperada

```
Introduce un número entero: 7
Introduce un número flotante: 2.3
Suma: 9.3
Multiplicación: 16.1
División: 3.0434782608695654
División entera: 3.0
Módulo: 0.1
```

Solución (girar para leer)

```
1 entero = int(input("Introduce un número entero: "))
2 flotante = float(input("Introduce un número flotante: "))
3 suma = entero + flotante
4 multiplicacion = entero * flotante
5 division = entero / flotante
6 division-entera = entero // flotante
7 modulo = entero % flotante
8 print("Suma:", suma)
9 print(f"Multiplicación: {multiplicacion:.1f}")
10 print("División:", division)
11 print("División entera:", division-entera)
12 print(f"Módulo: {modulo:.1f}")
```

Código 3.3: Solución al Ejercicio 3.3.

EJERCICIO 3.4

Escribe un programa en Python que evalúe las siguientes expresiones booleanas:

- $a \wedge b$
- $a \vee c$
- $\neg b$
- $a \wedge (b \vee c)$

El programa debe solicitar al usuario los valores de las variables a , b y c (como True o False), y mostrar los resultados de las expresiones.

Ejemplo de ejecución esperada

```
Introduce True o False para a: True
Introduce True o False para b: False
Introduce True o False para c: True
a and b: False
a or c: True
not b: True
a and (b or c): True
```

Solución (girar para leer)

```
1 # Solicitar los valores booleanos
2 a = input("Introduce True o False para a: ")
3 b = input("Introduce True o False para b: ")
4 c = input("Introduce True o False para c: ")
5
6 # Convertir las entradas en valores booleanos
7 a == "True"
8 b == "True"
9 c == "True"
10
11 # Evaluar las expresiones booleanas
12 print("a and b:", a and b)
13 print("a or c:", a or c)
14 print("not b:", not b)
15 print("a and (b or c):", a and (b or c))
```

Código 3.4: Solución al Ejercicio 3.4.

3. Tipos de datos y operadores

EJERCICIO 3.5

Escribe un programa en Python que utilice los operadores de asignación `+=`, `-=`, `*=`, `/=`, `**=` para modificar el valor de una variable. El programa debe solicitar al usuario un número inicial y luego aplicar los operadores mencionados, mostrando el valor de la variable después de cada operación.

Ejemplo de ejecución esperada

```
Introduce un número inicial: 10
Después de += 5: 15
Después de -= 3: 12
Después de *= 2: 24
Después de /= 4: 6.0
Después de **= 2: 36.0
```

Solución (gíralo para leer)

```
1 x = int(input("Introduce un número inicial: "))
2 x += 5
3 print("Después de += 5:", x)
4 x -= 3
5 print("Después de -= 3:", x)
6 x *= 2
7 print("Después de *= 2:", x)
8 x /= 4
9 print("Después de /= 4:", x)
10 x **= 2
11 print("Después de **= 2:", x)
```

Código 3.5: Solución al Ejercicio 3.5.

EJERCICIO 3.6

Escribe un programa en Python que calcule la suma de tres números. El programa debe solicitar al usuario los tres números y luego mostrar el resultado de la suma. Incorpora un control de errores ante entradas incorrectas por parte del usuario.

Ejemplo de ejecución esperada

```
Introduce el primer número: 3
Introduce el segundo número: 5
Introduce el tercer número: 7
La suma de los tres números es 15
```

Solución (girar para leer)

```

1  try:
2      num1 = int(input("Introduce el primer número: "))
3      num2 = int(input("Introduce el segundo número: "))
4      num3 = int(input("Introduce el tercer número: "))
5      suma = num1 + num2 + num3
6      print("La suma de los tres números es", suma)
7
8  except ValueError:
9      print("Error: Por favor, ingresa valores numéricos válidos.")
10

```

Código 3.6: Solución al Ejercicio 3.6.

3. Tipos de datos y operadores

EJERCICIO 3.7

Escribe un programa en Python que reciba tres números y determine cuál es el mayor de los tres. El programa debe solicitar al usuario los tres números y mostrar el número mayor. Incorpora un control de errores ante entradas incorrectas por parte del usuario.

Ejemplo de ejecución esperada

```
Introduce el primer número: 8
Introduce el segundo número: 12
Introduce el tercer número: 5
El número mayor es 12
```

Solución (girar para leer)

```
1 try:
2     num1 = int(input("Introduce el primer número: "))
3     num2 = int(input("Introduce el segundo número: "))
4     num3 = int(input("Introduce el tercer número: "))
5     # calculamos el número mayor
6     mayor = max(num1, num2, num3)
7     print("El número mayor es", mayor)
8 except ValueError:
9     print("Error: Por favor, ingresa valores numéricos válidos.")
```

Código 3.7: Solución al Ejercicio 3.7.

EJERCICIO 3.8

Escribe un programa en Python que cuente el número de caracteres en una cadena introducida por el usuario. El programa debe solicitar al usuario una cadena de texto y luego mostrar el número de caracteres que contiene. Incorpora un control de errores ante entradas incorrectas por parte del usuario.

Ejemplo de ejecución esperada

Introduce una cadena de texto: Hola Mundo
El número de caracteres es 10

Solución (girar para leer)

```

1  try:
2      cadena = input("Introduce una cadena de texto: ")
3
4      # Contamos el número de caracteres
5      longitud = len(cadena)
6      print("El número de caracteres es", longitud)
7
8      except ValueError:
9          print("Error: Por favor, ingresa una cadena de
          texto válida.")

```

Código 3.8: Solución al Ejercicio 3.8.

3. Tipos de datos y operadores

EJERCICIO 3.9

Escribe un programa en Python que convierta una cantidad de segundos a horas, minutos y segundos. El programa debe solicitar al usuario el número de segundos e imprimir el resultado de la conversión en el formato:

horas : minutos : segundos

Para realizar la conversión, utiliza las siguientes fórmulas:

- Las horas son el cociente de la división entera de los segundos entre 3600 (número de segundos en una hora).
- Los minutos son el cociente de la división entera del resto anterior entre 60 (número de segundos en un minuto).
- Los segundos resultantes son el resto de esa última división.

Ejemplo de ejecución esperada

Introduce la cantidad de segundos: 3665
La conversión es 1:1:5

Solución (girar para leer)

```
1 segundos = int(input("Introduce la cantidad de segundos\n"))  
2  
3 # Cálculo de horas, minutos y segundos  
4 horas = segundos // 3600  
5 minutos = (segundos % 3600) // 60  
6 segundos = segundos % 3600  
7  
8 # Mostrar el resultado  
9 print("La conversión es", horas, ":", minutos, ":", segundos)  
10
```

Código 3.9: Solución al Ejercicio 3.9.

EJERCICIO 3.10

Escribe un programa en Python que calcule la distancia entre dos puntos en el plano cartesiano, dadas sus coordenadas (x_1, y_1) y (x_2, y_2) . El programa debe solicitar al usuario las coordenadas de ambos puntos y luego calcular la distancia entre ellos utilizando la fórmula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

donde d es la distancia entre los puntos.

Ejemplo de ejecución esperada

Introduce las coordenadas del primer punto (x1, y1): 1 2
 Introduce las coordenadas del segundo punto (x2, y2): 4 6
 La distancia entre los dos puntos es 5.0

Solución (girar para leer)

```

1 # Solicitar las coordenadas de los dos puntos
2 x1, y1 = map(float, input("Introduce las coordenadas
3 del primer punto (x1, y1): ").split())
4 x2, y2 = map(float, input("Introduce las coordenadas
5 del segundo punto (x2, y2): ").split())
6 # split() se usa para separar las coordenadas, permite
7 dividir la cadena en dos partes,
8 basándose en el espacio en blanco entre los valores
9 dados por el usuario
10 # Calcular la distancia utilizando la fórmula de
11 distancia Euclídea
12 distancia = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
13 # Mostrar el resultado
14 print("La distancia entre los puntos es", distancia)

```

Código 3.10: Solución al Ejercicio 3.10.