

4

Instrucciones simples y estructuradas

En este capítulo exploraremos los distintos tipos de instrucciones que Python pone a disposición del programador para manejar datos y estructurar programas de manera eficiente.

Python es un lenguaje de alto nivel que facilita la escritura de código legible y conciso. A diferencia de otros lenguajes como C, C++ o Java, Python no requiere la especificación explícita de tipos de datos ni el uso obligatorio de ; al final de cada línea.

4.1. Instrucciones simples

Las instrucciones simples son las acciones básicas que realiza un programa en Python. Estas instrucciones incluyen la declaración y asignación de variables, la entrada y salida de datos, y las invocaciones de funciones y retornos de valores (estas dos últimas se estudiarán en un capítulo posterior). Son los componentes básicos que permiten manipular datos, hacer cálculos y controlar el flujo del programa.

4.1.1. Declaraciones y asignaciones

En Python, la declaración de variables no requiere un tipo específico, ya que Python utiliza un sistema de tipado dinámico. Esto significa que no necesitas especificar si una variable es un número entero, un flotante, una cadena de texto, etc. Python lo infiere automáticamente cuando se asigna un valor a la variable.

Por ejemplo:

```
1 | nombre = "Juan"    # Asigna una cadena de texto a la variable '
  ↪   nombre '
2 | edad = 30          # Asigna un número entero a la variable '
```

4. Instrucciones simples y estructuradas

```
↪      edad '  
3 altura = 1.75      # Asigna un número flotante a la variable '  
↪      altura '  
4 activo = True      # Asigna un valor booleano a la variable '  
↪      activo '
```

En Python, se usa el signo igual = para asignar un valor a una variable. Por ejemplo, en `nombre = "Juan"`, el valor "Juan" se asigna a la variable `nombre`. Recuerda que en Python puedes cambiar el valor de una variable en cualquier momento. Además, si cambia el tipo del nuevo valor asignado a la variable, dicha variable cambiará automáticamente de tipo. Por ejemplo:

```
1 nombre = "Pedro"  # Ahora 'nombre' tiene el valor "Pedro"  
2 nombre = 3        # Ahora 'nombre' es de tipo entero
```

En Python puedes asignar valores a varias variables en una sola línea. Por ejemplo:

```
1 a, b, c = 5, 10, 15 # Asigna 5 a 'a', 10 a 'b', y 15 a 'c'
```

Respecto a las constantes, en Python no existe el concepto de *constante* como tal (a diferencia de una variable, una constante es algo que no varía durante la ejecución del programa). Por tanto, cuando se quiere especificar que se está declarando una constante, por convención, se escribe en mayúsculas. Por ejemplo:

```
1 # Declaración de constantes  
2 PI = 3.141592
```

4.1.2. Expresiones

En Python, las *expresiones* son combinaciones de valores, operadores y funciones que resultan en un único valor. Las expresiones pueden incluir operaciones matemáticas, concatenación de texto, y mucho más.

Por ejemplo, la siguiente línea de código muestra una expresión matemática:

```
1 x = 5 + 3 # La expresión 5 + 3 se evalúa y se asigna el  
↪      valor 8 a la variable 'x'
```

Además de las operaciones matemáticas simples, Python también permite realizar operaciones de asignación combinadas que modifican el valor de una variable.

```
1 x = 10  
2 x += 5 # Equivalente a x = x + 5, ahora x es 15  
3 x *= 2 # Equivalente a x = x * 2, ahora x es 30
```

4.1.3. Entrada y salida de datos

Como ya hemos visto, las funciones `input()` y `print()` permiten interactuar con el usuario, ya sea para obtener información o mostrar resultados.

La función `input()` pide al usuario que ingrese información a través del teclado. El valor ingresado siempre se recibe como una cadena de texto, por lo que si se necesita otro tipo de dato, como un número entero o flotante, hay que convertirlo explícitamente (véase el Capítulo 3).

```
1 nombre = input("¿Cómo te llamas? ")
2 edad = int(input("¿Cuántos años tienes? ")) # Conversión
↪      explícita a entero
```

Respecto a la salida, la función `print()` se utiliza para mostrar información en la pantalla. Puede imprimir cualquier tipo de dato (texto, números, listas, etc.). En el Capítulo 3 se introdujeron las *f-strings* como forma idiomática de incluir variables dentro de cadenas:

```
1 print(f"Hola, {nombre}. Tienes {edad} años.")
```

A continuación se presentan dos opciones de formato útiles que se aplican a menudo durante la entrada/salida de datos: el formato en distintas bases numéricas y la alineación de texto.

Formato en distintas bases numéricas. Mediante un especificador de formato detrás del carácter `:` se puede mostrar un entero en distintas bases (decimal por defecto, octal con `o`, hexadecimal con `x` o `X`, binario con `b`):

```
1 num = 255
2 print(f"Decimal:      {num}")      # 255
3 print(f"Octal:        {num:o}")    # 377
4 print(f"Hexadecimal: {num:x}")    # ff
5 print(f"Binario:      {num:b}")    # 11111111
```

Alineación de texto. Para mejorar la presentación de datos en columnas, las *f-strings* permiten alinear texto a la izquierda (`<`), centrado (`^`) o a la derecha (`>`), indicando un ancho mínimo:

```
1 nombre = "Alice"
2 print(f"|{nombre:<10}|") # |Alice      |
3 print(f"|{nombre:^10}|") # |  Alice  |
4 print(f"|{nombre:>10}|") # |      Alice|
```

Los mismos especificadores se pueden aplicar también a enteros y flotantes, lo que es útil al mostrar tablas de resultados.

4.2. Instrucciones estructuradas

Las instrucciones estructuradas son esenciales para organizar y controlar el flujo de un programa. A través de ellas, podemos tomar decisiones (*instrucciones condicionales*),

repetir acciones (*instrucciones iterativas*) o agrupar varias instrucciones dentro de un mismo bloque (*bloques de código*). Estas estructuras permiten que un programa sea más flexible, eficiente y legible, al evitar la ejecución de código redundante y al tomar decisiones basadas en condiciones dinámicas (tomadas en tiempo de ejecución del programa).

En términos generales, las instrucciones estructuradas permiten crear programas que pueden tomar decisiones y repetir acciones de manera controlada, lo que las hace fundamentales para cualquier tipo de software. Además, el uso adecuado de estas estructuras favorece la modularidad, la claridad y la mantenibilidad del código, que son características deseadas en nuestros programas. A continuación se discuten estas instrucciones estructuradas en más detalle.

4.2.1. Bloques de código: agrupando instrucciones

En Python, los bloques de código se agrupan usando indentación (espacios o tabuladores al inicio de la línea). Un *bloque de código* es un conjunto de instrucciones que se ejecutan de manera secuencial, sin interrupciones, dentro de un mismo nivel de ejecución. Todo el código dentro de un bloque debe estar alineado con el mismo número de espacios o tabulaciones.

La indentación es uno de los elementos distintivos de Python. A diferencia de otros lenguajes que utilizan llaves (como C++ o Java, por ejemplo) para denotar bloques de código, Python usa la indentación para mejorar la legibilidad del código. La indentación obligatoria asegura que el código sea visualmente claro y menos propenso a errores de sintaxis.

Por ejemplo:

```
1 if x > 0:
2     print("El número es positivo") # Esta instrucción se
↪     ejecuta si x > 0
3     y = x * 2 # Esta también se ejecuta si x > 0
```

En este ejemplo, las dos líneas de código dentro del bloque de `if` sólo se ejecutan si la condición `x > 0` es verdadera. La indentación de 4 espacios (por convención) indica que ambas líneas forman parte del mismo bloque.

Los bloques de código también se pueden utilizar en otras estructuras como funciones, bucles, y condiciones para agrupar instrucciones que se deben ejecutar de manera conjunta.

4.2.2. Instrucciones condicionales: tomando decisiones

Las instrucciones condicionales permiten que un programa tome decisiones, es decir, ejecute diferentes bloques de código **según el valor de una condición o expresión booleana**. En Python, las estructuras condicionales más comunes son `if`, `elif` y `else`. Estas instrucciones permiten evaluar una condición y ejecutar un bloque de código si esa condición es verdadera, o ejecutar otro bloque de código si la condición es falsa.

Instrucción if

La instrucción `if` permite evaluar una condición y ejecutar un bloque de código si la condición es verdadera. Si la condición no es verdadera (es decir, si es falsa), el bloque de código bajo el `if` se salta y no se ejecuta.

Su sintaxis es la siguiente:

```
if condicion:
    # Código a ejecutar si condición es verdadera
```

Vemos ahora cómo usarla mediante un ejemplo:

```
1 edad = int(input("¿Cuántos años tienes?"))
2 if edad >= 18:
3     print("Eres mayor de edad.")
```

En este ejemplo, si `edad` es mayor o igual a 18, se ejecuta el bloque de código dentro del `if`, mostrando el mensaje "Eres mayor de edad". Si la condición no se cumple (por ejemplo, si `edad` es 17), entonces el bloque dentro del `if` no se ejecuta y el programa continúa. Las condiciones pueden ser más complejas, como expresiones aritméticas, comparaciones entre variables, o incluso la evaluación de resultados booleanos.

Instrucción elif

La instrucción `elif` (abreviatura de "else if") se utiliza cuando tienes más de una condición que verificar. Después de un `if`, si la condición del `if` es falsa, Python evalúa las condiciones de los bloques `elif` existentes. Puedes tener varios `elif` seguidos, cada uno con su propia condición. De hecho, puedes tener tantos bloques `elif` como necesites, lo que permite comprobar muchas condiciones diferentes. Después de un `if`, puedes añadir tantos `elif` como quieras. Puedes incluso no tener ninguna instrucción `elif`, dependerá del problema particular que estés resolviendo. Eso sí, una instrucción `elif` siempre tendrá una instrucción `if` previa. A este tipo de estructuras condicionales también se les conoce como *condicionales múltiples*.

Su sintaxis es la siguiente:

```
if condicion1:
    # Código si condición1 es verdadera
elif condicion2:
    # Código si condición2 es verdadera
```

Vemos ahora cómo usarla mediante un ejemplo:

```
1 edad = int(input("¿Cuántos años tienes?"))
2 if edad >= 18:
3     print("Eres mayor de edad.")
4 elif edad >= 13:
5     print("Eres un adolescente.")
```

4. Instrucciones simples y estructuradas

En este caso, si la primera condición (`edad >= 18`) no es verdadera, Python evalúa la condición del `elif` (`edad >= 13`). Supón que el usuario introduce el valor 16. Entonces, como `edad` es 16, se ejecutará el bloque de código dentro del `elif`, mostrando "Eres un adolescente".

Instrucción `else`: bloque de código alternativo

El bloque `else` es opcional y se ejecuta si todas las condiciones anteriores (de `if` y `elif` que haya) son falsas. El bloque `else` no lleva condición, ya que es un bloque de código por defecto que se ejecuta si no se ha ejecutado ningún bloque de código anterior.

Su sintaxis es la siguiente:

```
if condicion1:
    # Código si condición1 es verdadera
elif condicion2:
    # Código si condición2 es verdadera
else:
    # Código si ninguna de las condiciones anteriores es
    verdadera
```

Vemos ahora cómo usarla mediante un ejemplo:

```
1 edad = int(input("¿Cuántos años tienes?"))
2 if edad >= 18:
3     print("Eres mayor de edad.")
4 elif edad >= 13:
5     print("Eres un adolescente.")
6 else:
7     print("Eres un infante.")
```

Supón que el usuario introduce el valor 10. Entonces, como `edad` es 10, no cumple con ninguna de las condiciones del `if` ni del `elif`, por lo que el bloque de código dentro del `else` se ejecuta, mostrando "Eres un infante".

Combinando condiciones

Puedes combinar varias condiciones utilizando operadores lógicos como `and`, `or`, y `not` para crear condiciones más complejas dentro de un `if`, `elif` o `else`.

Recuerda que estos operadores se vieron en el Capítulo 3 y tienen tablas de verdad distintas. El operador `and` se utiliza cuando todas las condiciones deben ser verdaderas, el operador `or` se utiliza cuando al menos una de las condiciones debe ser verdadera, y el operador `not` invierte el resultado de la condición (es decir, si la condición es verdadera, `not` la convierte en falsa, y viceversa).

Por ejemplo:

```
1 edad = 20
2 tiene_permiso = False
3
4 if edad >= 18 and tiene_permiso:
5     print("Puedes entrar.")
6 else:
7     print("No puedes entrar.")
```

En este caso, la persona debe ser mayor de edad y tener permiso. Si cualquiera de estas condiciones no se cumple, se ejecutará el bloque `else`.

Condicionales anidados: condiciones dentro de condiciones

Las condiciones anidadas son cuando tienes una estructura condicional dentro de otra. Esto te permite realizar decisiones más complejas. Los bloques `if`, `elif` y `else` pueden contener otras estructuras condicionales.

Por ejemplo:

```
1 edad = 15
2 tiene_permiso = True
3
4 if edad >= 18:
5     print("Eres mayor de edad.")
6 else:
7     if tiene_permiso:
8         print("Eres menor de edad, pero tienes permiso.")
9     else:
10        print("Eres menor de edad y no tienes permiso.")
```

En este ejemplo, si la condición `edad >= 18` no se cumple, el bloque `else` se ejecuta y dentro de él, hay un `if` que evalúa si la persona tiene permiso (representado en la variable `tiene_permiso`). Dependiendo de la respuesta, se imprime un mensaje distinto.

4.2.3. Instrucciones iterativas: repitiendo acciones

Las estructuras iterativas son una parte fundamental en la programación, ya que permiten que un programa ejecute un bloque de código de manera repetitiva, basándose en una condición o un conjunto de datos. La capacidad de repetir acciones es imprescindible para cualquier programa que necesite realizar tareas de manera sistemática y eficiente, como recorrer listas, procesar datos o realizar cálculos repetitivos. En Python, existen dos tipos de estructuras iterativas principales: los bucles `while` y `for`.

Bucle `while`: ejecutar mientras se cumpla una condición

El bucle `while` es una estructura iterativa que repite un bloque de código mientras se cumpla una condición especificada. La condición se evalúa antes de cada iteración, y

4. Instrucciones simples y estructuradas

si es `True` (verdadera), se ejecuta el bloque de código dentro del bucle. Si la condición es `False` (falsa), el bucle se detiene y el programa continúa con la siguiente instrucción.

Su sintaxis es la siguiente:

```
↪ while condicion:
    # Código que se ejecuta mientras la condición sea
    verdadera
```

Lo que acompaña a `while`, `condicion`, es una expresión booleana que se evalúa en cada iteración del bucle. Mientras esta condición siga siendo verdadera, el bucle continuará ejecutándose. Si la condición se vuelve falsa, el bucle terminará. Se denomina *código dentro del bucle* al bloque de instrucciones que se ejecuta cada vez que la condición se evalúa como verdadera (que está indentando debajo de `while`).

Veamos cómo se usa con un ejemplo sencillo:

```
1 contador = 1
2 while contador <= 5:
3     print(contador)
4     contador += 1 # Incrementa el contador en 1
```

En la línea 1 se está declarando una variable entera denominada `contador` e inicializada a valor 1. Después, aparece un bloque `while`. La condición `contador <= 5` se evalúa antes de cada iteración. Mientras `contador` sea menor o igual a 5, el programa imprimirá el valor de `contador` y lo incrementará en cada iteración. Una vez que `contador` llegue a 6, la condición `contador <= 5` será `False`, y el bucle se detendrá.

Este tipo de bucles es útil cuando no sabemos cuántas veces necesitamos repetir una acción y la ejecución depende de una condición dinámica que cambia durante la ejecución del programa.

Un error común al usar el bucle `while` es no actualizar adecuadamente la condición en cada iteración, en el código dentro del bucle. Si la condición nunca se vuelve falsa, el bucle continuará ejecutándose indefinidamente, creando un bucle infinito. Por ejemplo:

```
1 while True:
2     print("Esto se repite para siempre")
```

Este bucle nunca terminará a menos que intervengas manualmente (por ejemplo, pulsando `Ctrl`+`C` desde la terminal en Windows, Linux o macOS).

Bucle `for`: repetir un bloque de código un número determinado de veces

El bucle `for` en Python es una de las estructuras iterativas más utilizadas. A diferencia del `while`, el `for` se utiliza cuando sabemos cuántas veces queremos repetir un bloque de código, o cuando queremos iterar sobre una secuencia de elementos, como una lista, una cadena de texto, un rango de números, o cualquier otro tipo de colección.

Su sintaxis es la siguiente:

```
for variable in secuencia:
    # Código a ejecutar para cada elemento de la secuencia
```

En este bucle, `variable` es una variable que toma el valor de cada elemento de la `secuencia` en cada iteración del bucle. El elemento `secuencia` puede ser una lista, una tupla, un rango, una cadena de texto o cualquier otro objeto iterable. El bucle ejecuta el bloque de código para cada elemento de la secuencia, asignando ese elemento a `variable` en cada iteración.

Veamos un ejemplo sencillo:

```
1 frutas = ["manzana", "banana", "cereza", "pera"]
2 for fruta in frutas:
3     print(f"Me gusta la {fruta}")
```

En la primera línea de código se define una lista de cadenas (el tipo de dato lista se estudiará en capítulos posteriores). El bucle `for` recorre la lista `frutas` y, en cada iteración, asigna a la variable `fruta` uno de los elementos de la lista (primero `"manzana"`, luego `"banana"`, después `"cereza"` y por último `"pera"`). En cada iteración se ejecuta el `print`, mostrando sucesivamente los mensajes `"Me gusta la manzana"`, `"Me gusta la banana"`, `"Me gusta la cereza"` y `"Me gusta la pera"`.

Una de las formas más comunes de usar un bucle `for` en Python es iterando sobre un rango de números. Para ello, es interesante que conozcas la función `range()`, que genera una secuencia de números y puedes usarla para controlar el número de iteraciones.

Su sintaxis concretamente es la siguiente:

```
range(start, stop, step)
```

Donde `start` es el número desde el que comienza el rango (por defecto es 0), `stop` es el número en el que termina el rango (este valor no se incluye en el rango) y `step` es el tamaño del paso entre cada número del rango (por defecto es 1).

Veamos un ejemplo sencillo:

```
1 for i in range(1, 6):
2     print(i)
```

Este bucle imprimirá los números del 1 al 5, ya que tamaño del paso por defecto (es decir, si no se explicita) es 1. El número 6 no se incluye porque, tal y como se ha descrito anteriormente, `range()` genera una secuencia hasta, pero sin incluir, el valor de `stop`.

Veamos otro ejemplo sencillo:

```
1 for i in range(0, 10, 2):
2     print(i)
```

Este bucle imprimirá los números 0, 2, 4, 6, 8, porque el `step` es 2, lo que significa que el bucle saltará de 2 en 2.

Control del flujo de ejecución en bucles: `break` y `continue`

Dentro de los bucles, puedes utilizar las instrucciones `break` y `continue` para controlar el flujo de ejecución.

4. Instrucciones simples y estructuradas

Concretamente, `break` se usa para salir inmediatamente de un bucle, independientemente de si la condición sigue siendo verdadera o si quedan elementos por recorrer. Por ejemplo:

```
1 for i in range(1, 6):
2     if i == 3:
3         break # Sale del bucle cuando i es 3
4     print(i)
```

Este código imprimirá 1, 2 y luego saldrá del bucle al evaluar la condición `i == 3`.

Por otro lado, `continue` se usa para saltar a la siguiente iteración del bucle, omitiendo el resto del código en el bloque de la iteración actual. Por ejemplo:

```
1 for i in range(1, 6):
2     if i == 3:
3         continue # Salta esta iteración cuando i es 3
4     print(i)
```

Este código imprimirá 1, 2, 4, 5. Cuando la variable `i` tiene el valor 3, se cumple la condición que acompaña al `if`, con lo que se ejecuta el `continue`, que hace que el bucle pase a la siguiente iteración sin ejecutar nada del resto de código dentro del bucle (en este ejemplo, el `print()`).

Cláusula `else` en bucles

A diferencia de muchos otros lenguajes, en Python tanto `for` como `while` admiten una cláusula `else` que se ejecuta **solo si el bucle termina de forma natural**, es decir, sin haber ejecutado un `break`. Esto resulta útil para distinguir entre los dos motivos por los que un bucle de búsqueda puede haber terminado: porque encontró lo que buscaba (en cuyo caso se interrumpe con `break`), o porque agotó la secuencia sin encontrarlo. Veamos un ejemplo:

```
1 numeros = [4, 8, 12, 16, 20]
2 buscado = 11
3
4 for n in numeros:
5     if n == buscado:
6         print(f"{buscado} encontrado")
7         break
8 else:
9     print(f"{buscado} no está en la lista")
```

Si `buscado` aparece en `numeros`, el `break` interrumpe el bucle y el bloque `else` no se ejecuta. Si no aparece, el bucle termina de forma natural y se ejecuta el `else`, mostrando que el valor no estaba en la lista.

Iteraciones idiomáticas: enumerate y zip

Cuando se itera sobre una secuencia, a veces se necesita conocer también la *posición* de cada elemento. Aunque podría hacerse manteniendo un contador manual, lo idiomático en Python es usar la función `enumerate()`, que devuelve pares (índice, elemento):

```
1 frutas = ["manzana", "banana", "cereza"]
2 for i, fruta in enumerate(frutas):
3     print(f"{i}: {fruta}")
4 # Salida:
5 # 0: manzana
6 # 1: banana
7 # 2: cereza
```

Si se quiere comenzar el índice en otro valor (por ejemplo, 1), se pasa con el parámetro `start`:

```
1 for i, fruta in enumerate(frutas, start=1):
2     print(f"{i}: {fruta}") # 1: manzana, 2: banana, ...
```

Cuando se necesita iterar sobre dos (o más) secuencias en paralelo, la función `zip()` las combina elemento a elemento:

```
1 nombres = ["Alice", "Bob", "Charlie"]
2 edades = [30, 25, 35]
3 for nombre, edad in zip(nombres, edades):
4     print(f"{nombre} tiene {edad} años")
```

Si las secuencias tienen distinta longitud, `zip` se detiene en la más corta sin lanzar error.

Combinación de bucles y condicionales

Los bucles `while` y `for` y las estructuras condicionales `if` se combinan habitualmente para crear ciclos de procesamiento más complejos. Es común que dentro de un bucle se evalúen condiciones para decidir qué hacer en cada iteración. Por ejemplo, el siguiente código combina `for` con `if` para recorrer una lista de números y mostrar si cada uno es par o impar:

```
1 numeros = [1, 2, 3, 4, 5]
2 for numero in numeros:
3     if numero % 2 == 0:
4         print(f"{numero} es par")
5     else:
6         print(f"{numero} es impar")
```

Combinando los recursos vistos en las subsecciones anteriores se pueden expresar tareas más ricas. Por ejemplo, el siguiente fragmento numera una lista de palabras y, mediante un `if` con `continue`, omite las palabras vacías:

4. Instrucciones simples y estructuradas

```
1 palabras = ["hola", "", "python", "", "mundo"]
2 for i, palabra in enumerate(palabras, start=1):
3     if palabra == "":
4         continue
5     print(f"Palabra {i}: {palabra}")
```

4.3. Ejercicios

EJERCICIO 4.1

Escribe un programa en Python que pida al usuario su edad y determine a qué categoría pertenece:

- Menor de edad (0-17 años)
- Adulto (18-64 años)
- Adulto mayor (65 años o más)

Ejemplo de ejecución esperada

Introduce tu edad: 20
Eres un adulto.

Solución (girar para leer)

```
1 edad = int(input("Introduce tu edad: "))
2
3 if edad > 18:
4     print("Eres un menor de edad.")
5 elif edad > 65:
6     print("Eres un adulto.")
7 else:
8     print("Eres un adulto mayor.")
```

Código 4.1: Solución al Ejercicio 4.1.

EJERCICIO 4.2

Escribe un programa en Python que pida un número entero al usuario y determine si el número es par o impar.

Ejemplo de ejecución esperada

Introduce un número: 7
El número es impar.

Solución (girar para leer)

Código 4.2: Solución al Ejercicio 4.2.

```
1 numero = int(input("Introduce un número: "))
2
3 if numero % 2 == 0:
4     print("El número es par.")
5 else:
6     print("El número es impar.")
```

EJERCICIO 4.3

Escribe un programa en Python que calcule el impuesto a pagar en función del salario. Se aplicarán las siguientes reglas:

- Si el salario es menor de 2000, no se paga impuesto.
- Si el salario es mayor o igual a 2000 pero menor de 5000, el impuesto será del 10 %.
- Si el salario es mayor o igual a 5000 pero menor de 10000, el impuesto será del 15 %.
- Si el salario es mayor o igual a 10000 pero menor de 20000, el impuesto será del 20 %.
- Si el salario es mayor o igual a 20000, el impuesto será del 30 %.

El programa debe solicitar al usuario el valor de su salario y calcular el impuesto a pagar según las reglas mencionadas. Además, debe mostrar el monto del impuesto y el salario final después de la deducción. Todos los números reales se mostrarán con precisión de dos decimales.

Ejemplo de ejecución esperada

```
Introduce tu salario: 4500
El impuesto a pagar es 450.0
Tu salario final es 4050.0
```

Solución (girar para leer)

```

1 salario = float(input("Introduce tu salario: "))
2
3 if salario < 2000:
4     impuesto = 0
5     elif salario < 5000:
6         impuesto = salario * 0.10
7     elif salario < 10000:
8         impuesto = salario * 0.15
9     elif salario < 20000:
10        impuesto = salario * 0.20
11    else:
12        impuesto = salario * 0.30
13
14 salario_final = salario - impuesto
15
16 print(f"El impuesto a pagar es {impuesto:.2f}")
17 print(f"Tu salario final es {salario_final:.2f}")

```

Código 4.3: Solución al Ejercicio 4.3.

4. Instrucciones simples y estructuradas

EJERCICIO 4.4

Escribe un programa en Python que determine si una persona puede obtener la licencia de conducir. La persona debe ser mayor de 18 años. El programa debe solicitar la edad del usuario y mostrar si puede o no obtener la licencia.

Ejemplo de ejecución esperada

Introduce tu edad: 17

No puedes obtener la licencia de conducir.

Solución (girar para leer)

```
1 edad = int(input("Introduce tu edad: "))
2
3 if edad >= 18:
4     print("Puedes", end=' ');
5     # con end especificamos el carácter con el que
6     acabar la línea
7     # por defecto, el print inserta un salto de línea
8     al final
9     print("obtener la licencia de conducir.")
10 else:
11     print("No puedes", end=' ');
```

Código 4.4: Solución al Ejercicio 4.4.

EJERCICIO 4.5

Escribe un programa en Python que pida la calificación de un examen (un número entre 0 y 10) y determine si el estudiante ha aprobado o no. La calificación mínima para aprobar es 5.

Ejemplo de ejecución esperada

Introduce tu calificación: 4
Has suspendido el examen.

Solución (girar para leer)

```

1 calificacion = float(input("Introduce tu calificación:
2
3 print("Has ", end='')
4 if calificacion >= 5:
5     print("aprobado", end='')
6 else:
7     print("suspendido", end='')
8 print(" el examen.")

```

Código 4.5: Solución al Ejercicio 4.5.

4. Instrucciones simples y estructuradas

EJERCICIO 4.6

Escribe un programa en Python que pida un número al usuario y determine si el número está en el rango entre 10 y 100 (inclusive).

Ejemplo de ejecución esperada

Introduce un número: 50

El número está en el rango.

Solución (girar para leer)

```
1 numero = int(input("Introduce un número: "))
2
3 print("El número", end=' ')
4 if not (10 <= numero <= 100):
5     print("no", end=' ')
6     print("está en el rango.")
```

Código 4.6: Solución al Ejercicio 4.6.

EJERCICIO 4.7

Escribe un programa en Python que pida tres longitudes de lados y determine si pueden formar un triángulo válido. Un triángulo es válido si la suma de las longitudes de dos lados siempre es mayor que la longitud del tercer lado.

Ejemplo de ejecución esperada

```
Introduce el primer lado: 3
Introduce el segundo lado: 4
Introduce el tercer lado: 5
Los lados forman un triángulo válido.
```

Solución (girar para leer)

```

1 lado1 = float(input("Introduce el primer lado: "))
2 lado2 = float(input("Introduce el segundo lado: "))
3 lado3 = float(input("Introduce el tercer lado: "))
4
5 if lado1 + lado2 < lado3 and \
6    lado1 + lado3 < lado2 and \
7    lado2 + lado3 < lado1:
8     # el carácter \ lo usamos para poder partir la
9     # expresión en
10    # varias líneas y que no se rompa la condición al
11    leerla
12 else:
13     print("Los lados forman un triángulo válido.")
14     print("Los lados no forman un triángulo válido.")

```

Código 4.7: Solución al Ejercicio 4.7.

EJERCICIO 4.8

Escribe un programa en Python que pida un número entre 1 y 12, correspondiente a un mes del año, y muestre el nombre del mes correspondiente (por ejemplo, 1 = enero, 2 = febrero, etcétera).

Ejemplo de ejecución esperada

Introduce el número del mes: 3

El mes es marzo.

Solución (girar para leer)

```

1 mes = int(input("Introduce el número del mes: "))
2
3 if mes == 1:
4     print("El mes es enero.")
5     if mes == 2:
6         print("El mes es febrero.")
7         if mes == 3:
8             print("El mes es marzo.")
9             if mes == 4:
10                print("El mes es abril.")
11                if mes == 5:
12                    print("El mes es mayo.")
13                    if mes == 6:
14                        print("El mes es junio.")
15                        if mes == 7:
16                            print("El mes es julio.")
17                            if mes == 8:
18                                print("El mes es agosto.")
19                                if mes == 9:
20                                    print("El mes es septiembre.")
21                                    if mes == 10:
22                                        print("El mes es octubre.")
23                                        if mes == 11:
24                                            print("El mes es noviembre.")
25                                            if mes == 12:
26                                                print("El mes es diciembre.")
27 else:
28     print("Número de mes inválido.")

```

Código 4.8: Solución al Ejercicio 4.8.

EJERCICIO 4.9

Escribe un programa en Python que calcule la calificación final de un estudiante en una asignatura. El estudiante tiene una calificación de examen y una nota de asistencia. La calificación final se determina con la siguiente lógica:

- Si la nota de examen es menor de 5, el estudiante aprueba o suspende automáticamente dependiendo de la nota de asistencia:
 - Si la nota de asistencia es mayor o igual a 7, el estudiante aprueba el curso con una calificación de 5.
 - Si la nota de asistencia es menor a 7, el estudiante suspende automáticamente con una calificación de 0, independientemente del examen.
- Si la nota de examen es mayor o igual a 5, el estudiante aprueba el curso si su nota de asistencia es mayor o igual a 5. En este caso, la calificación final será el promedio entre la nota de examen y la nota de asistencia.
- Si la nota de asistencia es menor a 5 y el examen es mayor o igual a 5, el estudiante aprobará pero con una calificación final mínima de 5.

El programa debe pedir al usuario las notas de examen y de asistencia y mostrar la calificación final.

Ejemplo de ejecución esperada

```
Introduce la nota del examen: 6
Introduce la nota de asistencia: 4
La calificación final es 5.0.
```

Solución (girar para leer)

```

1 examen = float(input("Introduce la nota del examen: "))
2 asistencia = float(input("Introduce la nota de
3 asistencia: "))
4 # Definir la calificación final
5 calificacion_final = 0
6
7 # Comprobar la calificación en función de las reglas
8 if examen > 5:
9     if asistencia >= 7:
10        calificacion_final = 5
11    else:
12        calificacion_final = 0
13    elif examen >= 5:
14        if asistencia >= 5:
15            calificacion_final = (examen + asistencia) / 2
16        else:
17            calificacion_final = 5
18
19 print(f"La calificación final es {calificacion_final:.1
20 f}")

```

Código 4.9: Solución al Ejercicio 4.9.

EJERCICIO 4.10

Escribe un programa en Python que calcule la bonificación de un empleado en función de su años de experiencia y su desempeño. Las reglas son las siguientes:

- Si el empleado tiene menos de 1 año de experiencia, no recibe bonificación.
- Si el empleado tiene 1 o más años de experiencia:
 - Si su desempeño es excelente, la bonificación será el 20 % de su salario.
 - Si su desempeño es bueno, la bonificación será el 10 % de su salario.
 - Si su desempeño es regular, la bonificación será el 5 % de su salario.
- Si el empleado tiene más de 10 años de experiencia y su desempeño es bueno o excelente, se le dará una bonificación extra de 500 EUR.
- Si el salario es menor a 2000 EUR, la bonificación será ajustada a un máximo de 200 EUR, independientemente de los cálculos anteriores.

El programa debe solicitar al usuario los siguientes datos:

- Años de experiencia.
- Desempeño (como una cadena de texto: “excelente”, “bueno”, “regular”).
- Salario.

El programa debe calcular la bonificación y mostrarla.

Ejemplo de ejecución esperada

```
Introduce tus años de experiencia: 12
Introduce tu desempeño (excelente, bueno, regular): bueno
Introduce tu salario: 3500
La bonificación es 850.0 EUR.
```

Solución (girar para leer)

```

1 # Solicitar los datos al usuario
2 anos-experiencia = int(input("Introduce tus años de
3 experiencia: "))
4 salario = float(input("Introduce tu salario: "))
5 # Inicializar bonificación
6 bonificacion = 0
7 # Calcular bonificación basada en años de experiencia
8 if anos-experiencia < 1:
9     bonificacion = 0
10 # No bonificación si tiene menos
11     de 1 año de experiencia
12 else:
13     if desempeño == "excelente":
14         bonificacion = salario * 0.20 # 20% de
15         bonificación por desempeño excelente
16     elif desempeño == "bueno":
17         bonificacion = salario * 0.10 # 10% de
18         bonificación por desempeño bueno
19     elif desempeño == "regular":
20         bonificacion = salario * 0.05 # 5% de
21         bonificación por desempeño regular
22 # Si el empleado tiene más de 10 años de
23     experiencia y un buen desempeño, bonificación
24     extra
25     if anos-experiencia > 10 and (desempeno == "bueno"
26     or desempeño == "excelente"):
27         bonificacion += 500 # Bonificación extra por
28         muchos años de experiencia
29 # Ajustar bonificación máxima si el salario es menor a
30     2000
31 if salario > 2000:
32     bonificacion = min(bonificacion, 200) # Limitar
33     bonificación a 200 EUR si el salario es bajo
34 # Mostrar la bonificación final
35 print(f"La bonificación es {bonificacion} EUR.")

```

Código 4.10: Solución al Ejercicio 4.10.

EJERCICIO 4.11

Escribe un programa en Python que utilice un bucle `while` para contar desde 1 hasta 10, imprimiendo cada número en cada iteración por pantalla, separados por el carácter coma .

Ejemplo de ejecución esperada

1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

Solución (girar para leer)

```
1 contador = 1
2 while contador <= 10:
3     print(contador, end=', ', )
4     contador += 1
```

Código 4.11: Solución al Ejercicio 4.11.

EJERCICIO 4.12

Escribe un programa que pida números al usuario de forma repetida hasta que el usuario introduzca el número 0. El programa debe mostrar la suma total de los números introducidos (sin contar el 0).

Ejemplo de ejecución esperada

```
Introduce un número: 5
Introduce un número: 3
Introduce un número: 2
Introduce un número: 0
La suma total es 10.
```

Solución (girar para leer)

```
1 suma_total = 0
2 while True:
3     numero = int(input("Introduce un número: "))
4     if numero == 0:
5         break
6     suma_total += numero
7
8 print(f"La suma total es {suma_total}.")
```

Código 4.12: Solución al Ejercicio 4.12.

EJERCICIO 4.13

Escribe un programa en Python que permita al usuario introducir varios números. El programa debe calcular y mostrar el promedio de los números introducidos. El usuario deberá escribir 0 para terminar de introducir números.

Ejemplo de ejecución esperada

```
Introduce un número: 5
Introduce un número: 3
Introduce un número: 10
Introduce un número: 0
El promedio es 6.0.
```

Solución (girar para leer)

```

1 suma = 0
2 contador = 0
3
4 while True:
5     numero = float(input("Introduce un número: "))
6     if numero == 0:
7         break
8     suma += numero
9     contador += 1
10
11 if contador > 0:
12     promedio = suma / contador
13     print(f"El promedio es {promedio}.")
14 else:
15     print("No se introdujeron números.")
```

Código 4.13: Solución al Ejercicio 4.13.

EJERCICIO 4.14

Escribe un programa que le pida al usuario que adivine un número secreto entre 1 y 100. El programa debe seguir pidiendo al usuario que ingrese un número hasta que adivine correctamente el número secreto o hasta que se acaben el número máximo de intentos permitidos (valor 5). El programa debe informar si el número ingresado es mayor o menor que el número secreto.

Para generar un número de manera pseudoaleatoria dentro de un rango definido en Python, puedes usar el módulo `random`:

```
1 import random
2 numero_secreto = random.randint(1, 100)
3 # Genera un número pseudoaleatorio entre 1 y 100
```

Ejemplo de ejecución esperada

```
Adivina el número entre 1 y 100 (1/5): 50
Demasiado alto.
Adivina el número entre 1 y 100 (2/5): 25
Demasiado bajo.
Adivina el número entre 1 y 100 (3/5): 40
¡Correcto! El número era 40.
```

Solución (girar para leer)

```

1 import random
2
3 # Número secreto aleatorio entre 1 y 100
4 numero_secreto = random.randint(1, 100)
5 intentos = 0
6 max_intentos = 5
7
8 while intentos < max_intentos:
9     intento = int(input("Adivina el número entre 1 y
10         intentos + 1}/{max_intentos} : "))
11     intentos += 1
12     if intento == numero_secreto:
13         print(f"¡Correcto! El número era {
14             numero_secreto}.")
15         break # Termina el bucle si adivinó el número
16     elif intento > numero_secreto:
17         print("Demasiado bajo.")
18         continue # Continúa al siguiente intento si el
19             número es bajo
20     else:
21         print("Demasiado alto.")
22         continue # Continúa al siguiente intento si el
23             número es alto
24     {numero_secreto}."}

```

Código 4.14: Solución al Ejercicio 4.14.

4. Instrucciones simples y estructuradas

EJERCICIO 4.15

Escribe un programa que pida un número entero al usuario y cuente cuántos dígitos tiene ese número, utilizando un bucle `while`.

Ejemplo de ejecución esperada

Introduce un número: 12345

El número tiene 5 dígitos.

Solución (girar para leer)

```
1 numero = int(input("Introduce un número: "))
2 contador_dígitos = 0
3
4 while numero > 0:
5     numero = numero // 10
6     contador_dígitos += 1
7
8 print(f"El número tiene {contador_dígitos} dígitos.")
```

Código 4.15: Solución al Ejercicio 4.15.

EJERCICIO 4.16

Escribe un programa que calcule el factorial de un número introducido por el usuario utilizando un bucle `while`. El factorial de un número n es el producto de todos los números enteros positivos desde 1 hasta n ($n! = n(n-1)(n-2)(n-3)\dots \cdot 3 \cdot 2 \cdot 1 = n \cdot (n-1)!$).

Ejemplo de ejecución esperada

Introduce un número: 5
El factorial de 5 es 120.

Solución (girar para leer)

Código 4.16: Solución al Ejercicio 4.16.

```

1 numero = int(input("Introduce un número: "))
2 factorial = 1
3
4 while numero > 1:
5     factorial *= numero
6     numero -= 1
7
8 print(f"El factorial es {factorial}.")
```

4. Instrucciones simples y estructuradas

EJERCICIO 4.17

Escribe un programa que imprima los primeros n números de la serie Fibonacci. El usuario debe ingresar el valor de n , y el programa debe mostrar los primeros n números de la serie. Recuerda que los primeros dos números de Fibonacci son 0 y 1, y cada número siguiente es la suma de los dos anteriores (es decir, el tercer número de Fibonacci es 1, el cuarto 2, el quinto 3, el sexto 5, y así sucesivamente).

Ejemplo de ejecución esperada

```
Introduce un número: 6
Los primeros 6 números de Fibonacci son:
0, 1, 1, 2, 3, 5,
```

Solución (girar para leer)

```
1 n = int(input("Introduce un número: "))
2 a, b = 0, 1
3 contador = 0
4
5 print("Los primeros", n, "números de Fibonacci son:")
6 while contador < n:
7     print(a, end=" ")
8     a, b = b, a + b # Actualiza los valores de a y b
9     contador += 1
```

Código 4.17: Solución al Ejercicio 4.17.

EJERCICIO 4.18

Escribe un programa en Python que pida una frase al usuario y cuente cuántas palabras contiene. El programa debe dividir la frase en palabras (usando el espacio como delimitador) y contar cuántas palabras hay.

Ejemplo de ejecución esperada

Introduce una frase: Esto es un ejercicio de programación
La frase tiene 6 palabras.

Solución (girar para leer)

```

1 frase = input("Introduce una frase: ")
2 palabras = frase.split() # Divide la frase en palabras
3 contador_palabras = 0
4
5 while contador_palabras < len(palabras):
6     contador_palabras += 1
7
8 print(f"La frase tiene {contador_palabras} palabras.")

```

Código 4.18: Solución al Ejercicio 4.18.

EJERCICIO 4.19

Escribe un programa que haga un juego de adivinanza, en el que el usuario debe adivinar un número secreto entre 1 y 100. El programa debe permitir un máximo de 5 intentos. Si el usuario no adivina el número dentro de esos 5 intentos, el programa debe mostrar un mensaje diciendo que se ha agotado el número de intentos. Además, si el usuario adivina el número, el programa debe felicitarlo y terminar el juego.

Si el número ingresado es mayor o menor que el número secreto, el programa debe decirle al usuario si el número es mayor o menor, y permitirle seguir intentando. Condiciones:

- Si el usuario adivina el número, el juego termina de inmediato usando `break`.
- Si el número ingresado es mayor o menor que el número secreto, se muestra el mensaje correspondiente y el programa continúa pidiendo un nuevo intento usando `continue`.

Ejemplo de ejecución esperada

```
Adivina el número entre 1 y 100: 50
Demasiado alto.
Adivina el número entre 1 y 100: 25
Demasiado bajo.
Adivina el número entre 1 y 100: 30
¡Correcto! El número era 30.
```

Solución (girar para leer)

```

1 import random
2
3 # Número secreto aleatorio entre 1 y 100
4 numero_secreto = random.randint(1, 100)
5 intentos = 0
6 max_intentos = 5
7
8 while intentos < max_intentos:
9     intento = int(input("Adivina el número entre 1 y
10         intentos += 1
11
12     if intento == numero_secreto:
13         print("¡Correcto! El número era {
14             numero_secreto}")
15         break # Termina el bucle si adivinó el número
16     elif intento < numero_secreto:
17         print("Demasiado bajo.")
18         continue # Continúa al siguiente intento si el
19             número es bajo
20     else:
21         print("Demasiado alto.")
22         continue # Continúa al siguiente intento si el
23             número es alto
24     {numero_secreto}")

```

Código 4.19: Solución al Ejercicio 4.19.

EJERCICIO 4.20

Escribe un programa que pida una letra al usuario y una frase al usuario, y luego cuente cuántas palabras comienzan con una letra específica. El programa debe seguir pidiendo palabras hasta que el usuario escriba "fin" para terminar. Si una palabra comienza con la letra que el usuario ha especificado, debe aumentar el contador.

Condiciones:

- El bucle debe continuar hasta que el usuario ingrese "fin" (sin contarla como una palabra).
- Si el usuario ingresa "fin", el programa termina con el número de palabras que comienzan con la letra especificada.

Ejemplo de ejecución esperada

```
Introduce la letra con la que deben comenzar las palabras: b
Introduce una palabra (o 'fin' para terminar): manzana
Introduce una palabra (o 'fin' para terminar): banana
Introduce una palabra (o 'fin' para terminar): flor
Introduce una palabra (o 'fin' para terminar): fin
Las palabras que comienzan con 'b' son: 1.
```

Solución (girar para leer)

```

1 letra_especifica = input("Introduce la letra con la que
2   deben comenzar las palabras: ").lower()
3   contador = 0
4   while True:
5       palabra = input("Introduce una palabra (o 'fin'
6           para terminar): ").lower()
7       if palabra == "fin":
8           break # Termina el bucle si el usuario escribe
9           'fin'
10      if palabra[0] == letra_especifica:
11          contador += 1 # Aumenta el contador si la
12          palabra empieza con la letra especificada
13      print(f"Las palabras que comienzan con '{
14          letra_especifica}' son: {contador}.")

```

Código 4.20: Solución al Ejercicio 4.20.

4. Instrucciones simples y estructuradas

EJERCICIO 4.21

Escribe un programa en Python que calcule la suma de los primeros 10 números enteros positivos utilizando un bucle `for`.

Ejemplo de ejecución esperada

La suma de los primeros 10 números es 55.

Solución (girar para leer)

```
1 suma = 0
2 for i in range(1, 11):
3     suma += i
5 print(f"La suma de los primeros 10 números es {suma}.")
```

Código 4.21: Solución al Ejercicio 4.21.

EJERCICIO 4.22

Escribe un programa que pida un número al usuario e imprima la tabla de multiplicar del número dado por el usuario utilizando un bucle `for`.

Ejemplo de ejecución esperada

```
Introduce un número: 5
La tabla de multiplicar de 5 es:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

Solución (girar para leer)

```

1 numero = int(input("Introduce un número: "))
2
3 print(f"La tabla de multiplicar de {numero} es:")
4 for i in range(1, 11):
5     print(f"{numero} x {i} = {numero * i}")

```

Código 4.22: Solución al Ejercicio 4.22.

4. Instrucciones simples y estructuradas

EJERCICIO 4.23

⇒ Escribe un programa en Python que imprima todas las palabras de una lista utilizando un bucle `for`. Considera la siguiente lista de palabras: `['manzana', 'banana', 'cereza', 'naranja', 'kiwi']`

Ejemplo de ejecución esperada

```
Palabra 1: manzana
Palabra 2: banana
Palabra 3: cereza
Palabra 4: naranja
Palabra 5: kiwi
```

Solución (girar para leer)

```
1 palabras = ['manzana', 'banana', 'cereza', 'naranja', 'kiwi']
2
3 for i, palabra in enumerate(palabras, 1):
4     print(f"Palabra {i}: {palabra}")
```

Código 4.23: Solución al Ejercicio 4.23.

EJERCICIO 4.24

Escribe un programa que invierta una lista de números utilizando un bucle `for`. El programa debe imprimir la lista original y la lista invertida.

Ejemplo de ejecución esperada

Lista original: [1, 2, 3, 4, 5]
Lista invertida: [5, 4, 3, 2, 1]

Solución (girar para leer)

```

1 numeros = [1, 2, 3, 4, 5]
2 numeros_invertidos = []
3
4 for numero in numeros:
5     numeros_invertidos.insert(0, numero)
6
7 print(f"Lista original: {numeros}")
8 print(f"Lista invertida: {numeros_invertidos}")

```

Código 4.24: Solución al Ejercicio 4.24.

EJERCICIO 4.25

Escribe un programa que calcule la suma de todos los números impares entre 1 y un número dado por el usuario, mayor que 1, utilizando un bucle `for`.

Ejemplo de ejecución esperada

Introduce un número: 100

La suma de los números impares entre 1 y 100 es 2500.

Solución (girar para leer)

```
1 numero = int(input("Introduce un número: "))
2 suma_impares = 0
3 for i in range(1, numero + 1, 2):
4     suma_impares += i
5
6 print(f"La suma de los números impares entre 1 y {numero} es {suma_impares}.")
```

Código 4.25: Solución al Ejercicio 4.25.

EJERCICIO 4.26

Escribe un programa que cuente cuántas veces aparece un carácter específico en una cadena de texto utilizando un bucle `for`. Solicita al usuario tanto la cadena de texto como el carácter a contar.

Ejemplo de ejecución esperada

Introduce una cadena de texto: programar
 Introduce el carácter a contar: r
 El carácter 'r' aparece 3 veces en la cadena.

Solución (girar para leer)

```

1  cadena = input("Introduce una cadena de texto: ")
2  caracter = input("Introduce el carácter a contar: ")
3
4  contador = 0
5  for letra in cadena:
6      if letra == caracter:
7          contador += 1
8
9  print(f"El carácter '{caracter}' aparece {contador}
    veces en la cadena.")

```

Código 4.26: Solución al Ejercicio 4.26.

4. Instrucciones simples y estructuradas

EJERCICIO 4.27

Escribe un programa que calcule el promedio de los números contenidos en una lista de 10 números utilizando un bucle `for`. Considera la siguiente lista de números: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

Ejemplo de ejecución esperada

El promedio es 55.0

Solución (girar para leer)

```
1 numeros = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
2 suma = 0
3 cantidad = 0
4
5 for numero in numeros:
6     suma += numero
7     cantidad += 1
8
9 promedio = suma / cantidad
10 print(f"El promedio es {promedio}.")
```

Código 4.27: Solución al Ejercicio 4.27.

EJERCICIO 4.28

Escribe un programa que cuente cuántas vocales (mayúsculas o minúsculas) contiene una cadena de texto ingresada por el usuario. El programa debe usar un bucle for para iterar sobre cada carácter de la cadena.

Ejemplo de ejecución esperada

Introduce una cadena de texto: Hola Mundo
El número de vocales en la cadena es 4.

Solución (girar para leer)

```

1 cadena = input("Introduce una cadena de texto: ").lower()
2 vocales = "aeiou"
3 contador_vocales = 0
4
5 for letra in cadena:
6     if letra in vocales:
7         contador_vocales += 1
8
9 print(f"El número de vocales en la cadena es {
10     contador_vocales}")

```

Código 4.28: Solución al Ejercicio 4.28.

EJERCICIO 4.29

Escribe un programa que encuentre todos los números primos entre 1 y 100, utilizando un bucle `for`. Un número primo es aquel que sólo es divisible por 1 y por sí mismo. El programa debe imprimir los números primos encontrados en este rango.

Condiciones:

- Si se encuentra un número que no es primo, se debe usar `continue` para pasar al siguiente número sin hacer más cálculos.
- Si se encuentra un número primo, se debe imprimir en pantalla.
- El programa debe usar un bucle anidado, donde el bucle exterior recorre los números del 1 al 100, y el bucle interior verifica si cada número es primo.

Ejemplo de ejecución esperada

Números primos entre 1 y 100: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

Solución (girar para leer)

```

1 for num in range(2, 101): # Recorre los números del 2
2     es-primo = True # Inicializamos la suposición de
3     for i in range(2, num): # Verificamos si el número
4         es divisible por algún número menor que el
5         if num % i == 0: # Si es divisible, no es
6             es-primo = False
7             break # No necesitamos seguir verificando,
8             salimos del bucle interior
9     if es-primo:
10        print(num, end=" ")

```

Código 4.29: Solución al Ejercicio 4.29.

EJERCICIO 4.30

Escribe un programa que genere una secuencia de números que cumpla las siguientes condiciones:

- La secuencia comienza en 1 y termina en 100.
- La secuencia incluye solo los números que:
 - Son múltiplos de 3, pero no de 5.
 - Son múltiplos de 5, pero no de 3.
 - Si el número es múltiplo tanto de 3 como de 5, se omite.
- El programa debe imprimir la secuencia resultante.

Condiciones:

- Si el número es múltiplo de 3 y 5, usar continue para saltar al siguiente número sin agregarlo a la secuencia.
- Si el número es múltiplo de 3 pero no de 5, agregarlo a la secuencia.
- Si el número es múltiplo de 5 pero no de 3, agregarlo también.

Ejemplo de ejecución esperada

```
3 5 6 9 10 12 18 20 21 24 25 27 33 35 36 39 40 42 48 50 51 54 55 57
63 65 66 69 70 72 78 80 81 84 85 87 93 95 96 99 100
```

Solución (girar para leer)

```

1 for num in range(1, 101):
2     if num % 3 == 0 and num % 5 == 0:
3         continue
4     if num % 3 == 0 or num % 5 == 0:
5         print(num, end=" ")
6     else:
7         print(num, end=" ")
8     print()

```

Código 4.30: Solución al Ejercicio 4.30.