



Laboratorio 3 - Ciclos While

Objetivos

Parte 1: Entendiendo el Ciclo While

[Actividad 1: Analizando el Ciclo While](#)

[Actividad 2: Ciclos Infinitos](#)

[Actividad 3: Ciclos dentro de ciclos](#)

Parte 2: Patrones Comunes con Ciclos While

[Patrón 1: Cantidad fija de repeticiones](#)

[Patrón 2: Contadores](#)

[Patrón 3: Valor específico como condición](#)

[Patrón 4: Variables como condición \(Flags\)](#)

Parte 3: Desafíos

[Desafío 1: Rayo que distingue entre los humanos y el pan integral](#)

[Desafío 2: Clasificador de Ballenas](#)

[Desafío 3: Nuevo Calculador de Promedio Avanzado V1.0 de Goddard](#)

[Desafío 4: La tienda de Doña Lucía](#)

[Desafío 5: Tiempo de reverberación](#)

Objetivos

1. Resolver problemas en Python a través de algoritmos con ciclos que contemplen una cantidad predefinida de iteraciones. Los ciclos se implementan con la instrucción `while` y un contador.
2. Resolver problemas en Python a través de algoritmos con ciclos basados en condiciones generales (uso de flags, en caso de ser necesario). Los ciclos se implementan con la instrucción `while`.
3. Reconocer la necesidad de uso e implementar patrones conocidos de ciclos: buscar mayor/menor, buscar algo, acumular (con + y *), contar
4. Escribir programas que repiten un proceso que ya es repetitivo, creando entonces una estructura de ciclo anidado. Todos los ciclos se implementan con la instrucción `while` y pueden ser con cantidad de iteraciones definidas o controlados por condiciones generales.

5. Diseñar algoritmos que utilizan ciclos anidados para resolver problemas que, por su naturaleza, requieren de esa estructura de control.

Parte 1: Entendiendo el Ciclo While

Imagina que necesitas escribir un programa que imprima los números del 1 al 10 en la pantalla. Podrías hacerlo escribiendo una instrucción `print()` para cada número, como esto:

```
print(1)
print(2)
print(3)
print(4)
print(5)
print(6)
print(7)
print(8)
print(9)
print(10)
```

Este código funciona, pero es **ineficiente** y **poco flexible**. ¿Qué pasaría si ahora quisieras imprimir los números del **1 al 100**? ¿O del **1 al 1.000.000**? ¿O un valor cualquiera que **ingrese un usuario**?

Aquí es donde entran los **ciclos**, una herramienta clave en la programación que nos permite **repetir acciones sin escribir el mismo código una y otra vez**.

En Python, el ciclo `while` nos permite ejecutar un bloque de código **mientras** una condición se mantenga verdadera. Veamos cómo es su estructura:

Identación como en el caso del if `while condición:` Formadas como las que vimos para el caso del if
sentencias a ejecutar si se cumple la condición

Mientras la condición se **cumpla** el ciclo `while` continúa, es decir, se ejecuta todo lo que esté **dentro** de él.

Observa cómo podemos escribir el mismo programa que imprime 10 números, pero de manera más eficiente y acotada con un ciclo `while`:

```
n = 1
while n <= 10:
    print(n)
    n += 1
```

- En este caso, primero se define la variable `n` y se le da como valor inicial `1`.
- Luego, como cumple la condición del ciclo `while` de `n <= 10`, se va a imprimir el valor actual de `n`, y luego se le sumará `1` a su valor.
- Ahora `n` vale `2`, como sigue cumpliendo la condición del ciclo `while` de **ser menor o igual a 10**, se imprimirá su valor actual `2` y se le sumará `1`, quedando `n = 3`.
- Y así sucesivamente irá imprimiendo los números del 1 al 10. Cuando el valor de `n` sea `11`, **ya no se cumplirá la condición del ciclo, por lo que no se repetirá más**.



¿Qué ocurre cuando hacemos `n += 1`?

Cuando hacemos `n += 1`, estamos aumentando el valor de `n` en 1. Es una forma abreviada de escribir `n = n + 1`.

Esta operación se usa comúnmente en ciclos para actualizar una variable y evitar ciclos infinitos.

También podemos sumar o restar otros valores, por ejemplo:

- `n += 2` (suma de 2 en 2)
- `n -= 1` (resta de 1 en 1)
- `n += 5` (suma de 5 en 5)

Esto nos permite controlar la velocidad y dirección del cambio de `n` en un ciclo.

💻 Manos al teclado:

Entendamos cómo funciona el ciclo `while` usando el ejemplo anterior:

1. Copia y pega el código anterior que imprime los números del 1 al 10 usando el ciclo `while`, y ejecútalo.

Ahora cambia el valor `10` de la condición del ciclo a otros números y ve qué ocurre:

1. ¿Qué pasa si ponemos un número mayor que 10?
2. ¿Y si ponemos un número menor que 1? ¿Por qué ocurre esto?



Un ciclo `while` solo se ejecuta si la condición inicial es **verdadera**.

Si la condición es **falsa desde el inicio**, el código dentro del ciclo nunca se ejecuta.

En este caso, por ejemplo si el usuario ingresa `-2`, como `n` empieza 1 y `1 <= -2` es **falso**, el ciclo `while` **nunca se ejecuta**.

Este comportamiento es útil cuando queremos asegurarnos de que solo entremos al ciclo si se cumple una determinada condición. Si la condición no se cumple, el programa simplemente sigue con las instrucciones que están **después** del `while`.

2. Modifica el código para que ahora imprima los números del 1 a un valor ingresado por el usuario.
3. Ya que el código no funciona cuando se ingresa un número menor a 1, agrega una condición para que cuando pase esto, se le avise al usuario por consola que el programa no podrá contar.

Actividad 1: Analizando el Ciclo While

Vamos a realizar algunos ejercicios para familiarizarnos con el ciclo `while`, e ir entendiendo diferentes tipos de situaciones en las que es útil utilizarlo:

1. ¿Qué imprimirá este código en la consola? Responde, y luego copia y pega el código, y ejecútalo, para ver si acertaste o no.

```
contador = 1
while contador <= 3:
```

```
print("Hola")
contador += 1
```

2. ¿Cuántas veces se ejecuta el siguiente ciclo? ¿Por qué?

```
n = 0
while n < 8:
    print("Python")
    n += 2
```

3. ¿Qué pasará si ejecutamos este código?

```
x = 1
while x != 10:
    print(x)
    x += 3
```

a. ¿Por qué no imprime el número 10?

4. Analiza qué hace el siguiente código:

```
contador = 0
num = int(input("Ingresa un número: "))

while num >= 0:
    contador += 1
    num = int(input("Ingresa otro número: "))

print("Ingresaste", contador, "números positivos.")
```

Modifica el código anterior para que, además de contar los números positivos ingresados, también se obtenga la suma de todos los números y se imprima su valor al final.

▼ 🔍 Pista

Vas a necesitar definir otra variable al inicio que vaya acumulando la suma total. Recuerda darle un valor inicial. Luego, dentro del ciclo deberás ir sumando en esa variable cada número ingresado por el usuario.

5. Completa el código para contar cuántos números entre 1 y 20 son múltiplos de 3.

```
contador = 0
n = 1
while n <= 20:
    if _____: # (Completa la condición)
        contador += 1
    n += 1
print("Cantidad de múltiplos de 3:", contador)
```

6. Completa el siguiente código para que el jugador intente adivinar un número secreto. El ciclo debe continuar hasta que el jugador adivine correctamente.

```
import random
numero_secreto = random.randint(1, 10)
intento = 0

while intento != _____: # Completa la condición para que el ciclo termine cuando adivine
    intento = int(input("Adivina el número (1-10): "))

    if intento _____ numero_secreto: # Completa la comparación
        print("¡Correcto! Has adivinado.")
    elif intento < numero_secreto:
        print("El número es mayor.")
    else:
        print("El número es menor.")
```

Analiza:

- ¿Por qué la variable `intento` se inicializa en 0?
- ¿Hubiese servido cualquier otro valor inicial?
- ¿Qué pasaría si en vez de darle un valor inicial a la variable `intento`, le preguntáramos de inmediato al usuario una opción? ¿Habría que cambiar el resto del programa?

7. Trabajas en el Registro Civil y tienes que verificar las edades de varias personas para determinar si son mayores de edad y pueden acceder a ciertos servicios. Tu jefe te indica al inicio del día la cantidad de personas que se atenderán hoy, y debes verificar cada edad ingresada para clasificar si la persona es mayor o menor de edad.

a. Escribe un programa que reciba la cantidad de personas que atenderás ese día, y por cada una pida ingresar la edad e indique si es mayor o menor de edad. Guíate con los siguientes ejemplos de ejecución:

Ingrese la cantidad de personas: 2

Ingrese la edad de la persona: 5

Es menor de edad.

Ingrese la edad de la persona: 19

Es mayor de edad.

Ingrese la cantidad de personas: 3

Ingrese la edad de la persona: 17

Es menor de edad.

Ingrese la edad de la persona: 45

Es mayor de edad.

Ingrese la edad de la persona: 13

Es menor de edad.

b. Modifica tu programa para que cuente cuántas personas fueron mayor de edad, y al final muestre un mensaje con esa suma. Guíate por el siguiente ejemplo:

Ingrese la cantidad de personas: 3

Ingrese la edad de la persona: 18

Es mayor de edad.

Ingrese la edad de la persona: 6

Es menor de edad.

Ingrese la edad de la persona: 37

Es mayor de edad.

El número de mayores de edad es: 2

- c. ¿Qué ocurre con tu programa si el usuario ingresa como cantidad de personas inicial un número menor o igual a 0?
- d. Modifica tu programa para que, si el usuario ingresa un número inválido de personas, muestre un mensaje de error. Guíate por el siguiente ejemplo:

```
Ingrese la cantidad de personas: -19  
ERROR: número de personas inválido.
```



Ahora que ya hemos visto diferentes ejemplos podemos preguntarnos: ¿Cuándo usar un ciclo `while`?

El ciclo `while` es útil cuando:

- No sabemos de antemano cuántas veces se repetirá el código.
- Necesitamos repetir una acción con una cantidad fija de repeticiones, pero sin escribir cada paso manualmente.
- Queremos ejecutar algo hasta que se cumpla una condición (por ejemplo, hasta que el usuario ingrese una respuesta válida).

Actividad 2: Ciclos Infinitos

Prueba ejecutar este código en Python y observa qué sucede:

```
n = 1  
while n > 0:  
    print(n)  
    n += 1
```



Para detener la ejecución de un programa que generó un ciclo infinito en Python, en la consola puedes presionar las teclas **ctr + c**.

Pregunta: ¿Por qué este ciclo nunca termina?

Un **ciclo infinito** ∞ ocurre cuando un bucle `while` **nunca deja de ejecutarse** porque la condición que lo controla **siempre es verdadera**. Esto puede hacer que un programa se quede atrapado en un bucle sin fin, bloqueando su ejecución.

🔍 ¿Qué sucede aquí?

- La condición `n > 0` **siempre es verdadera**, porque `n` comienza en 1 y solo sigue aumentando.
- Como nunca hay una instrucción que haga que `n` deje de cumplir la condición, **el ciclo nunca termina**.

✓ Cómo evitar ciclos infinitos

Para que un ciclo termine correctamente, debemos asegurarnos de que **en algún momento la condición se vuelva falsa**.

Analicemos y arreglemos algunos ciclos infinitos:

1. Corrige este ciclo para que se detenga cuando `n` llegue a 5:

```
n = 1
while n > 0:
    print(n)
    # Falta algo aquí...
```

2. Analiza el siguiente código:

```
respuesta = "sí"
while respuesta == "sí":
    print("Sigues en el bucle...")
```

- a. ¿Qué ocurre si se ejecuta?



Este es otro ejemplo de un error común: olvidarse de actualizar la variable de la condición del ciclo `while`. El problema es que la variable `respuesta` **nunca cambia**, por lo que **la condición siempre es verdadera**.

- b. Corrige este ciclo para que termine cuando el usuario ingrese `"no"`:

▼ 00 Pista

Asegúrate de que la condición en `while` pueda volverse **falsa** en algún momento cambiando dentro del ciclo el valor de la variable `respuesta`.

Actividad 3: Ciclos dentro de ciclos

Tal como con los condicionales, podemos tener ciclos `while` anidados, es decir, ciclos dentro de otros ciclos. Los ciclos anidados se utilizan cuando necesitamos repetir un proceso dentro de otro proceso repetitivo.

Veamos un ejemplo para entender mejor el concepto: necesitamos hacer un programa en el que el usuario ingrese un número `n`, y se muestren las tablas de multiplicar del 1 al `n`. Un ejemplo de ejecución esperada es:

Ingresar un número para generar sus tablas de multiplicar: 3

Tabla del 1

$1 * 1 = 1$

$1 * 2 = 2$

$1 * 3 = 3$

$1 * 4 = 4$

$1 * 5 = 5$

$1 * 6 = 6$

$1 * 7 = 7$

$1 * 8 = 8$

$1 * 9 = 9$

$1 * 10 = 10$

Tabla del 2

$2 * 1 = 2$

$2 * 2 = 4$

$2 * 3 = 6$

$2 * 4 = 8$

$2 * 5 = 10$

$2 * 6 = 12$

$2 * 7 = 14$

$2 * 8 = 16$

$2 * 9 = 18$

$2 * 10 = 20$

Tabla del 3

```
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
```

En este caso vamos a necesitar un ciclo dentro de otro en nuestro código:

- El **ciclo externo** va ir iterando del 1 al n, y **representará la tabla de multiplicar que iremos calculando**. Por ejemplo, tabla del 1, tabla del 2, ..., hasta llegar al máximo ingresado por el usuario.
- El **ciclo interno** va a iterar siempre del 1 al 10, y va a representar la creación de cada tabla.

```
maximo = int(input("Ingrese un número para generar sus tablas de multiplicar"))
tabla = 1 #comenzamos con la tabla del 1
while tabla <= maximo:
    print("Tabla del", tabla)
    numero = 1 #por cada tabla, vamos a ir multiplicando del 1 al 10, por eso em
    while numero <= 10:
        multiplicacion = tabla * numero
        print(tabla, "*", numero, "=", multiplicacion)
        numero += 1
    tabla += 1 #al final de cada iteración, aumentamos el número de la tabla
```

Puntos importantes a tener en cuenta:

- Al inicializar la variable `numero = 1` (que es la que itera el ciclo interno), debe ser dentro del ciclo externo, ya que por cada tabla de multiplicar necesitamos que comience en 1 y llegue al 10.
- Dentro del ciclo interno debemos ir aumentando la variable `numero += 1`, ¡o generaremos un ciclo infinito! ∞

Veamos algunos ejercicios que requieren el uso de ciclos anidados:

1. Cinco concursantes participan en una competencia de canto y reciben puntuaciones de 3 jueces diferentes. Escribe un programa que por cada uno de los concursantes pida la puntuación de los 3 jueces, y muestre el promedio de cada uno. Guíate por el siguiente ejemplo de ejecución:

Concursante 1

Juez 1, ingrese la calificación (0-10): 4

Juez 2, ingrese la calificación (0-10): 3

Juez 3, ingrese la calificación (0-10): 5

Promedio del Concursante 1: 4.00

Concursante 2

Juez 1, ingrese la calificación (0-10): 6

Juez 2, ingrese la calificación (0-10): 6

Juez 3, ingrese la calificación (0-10): 7

Promedio del Concursante 2: 6.33

Concursante 3

Juez 1, ingrese la calificación (0-10): 8

Juez 2, ingrese la calificación (0-10): 9

Juez 3, ingrese la calificación (0-10): 5

Promedio del Concursante 3: 7.33

Concursante 4

Juez 1, ingrese la calificación (0-10): 10

Juez 2, ingrese la calificación (0-10): 8

Juez 3, ingrese la calificación (0-10): 8

Promedio del Concursante 4: 8.67

Concursante 5

Juez 1, ingrese la calificación (0-10): 4

Juez 2, ingrese la calificación (0-10): 3

Juez 3, ingrese la calificación (0-10): 6

Promedio del Concursante 5: 4.33

¡Calificaciones registradas!

▼  **Pista**

Necesitarás:

- Un **ciclo externo** para los concursantes.
- Un **ciclo interno** para los jueces.

2. Un cine vende boletos para **2 funciones** del día. Cada función tiene **5 clientes** que compran **entre 1 y 3 boletos**. Haz un programa que para cada función pida la cantidad de boletos de cada cliente, y al final muestre el total de boletos vendidos. Guíate con el siguiente ejemplo de ejecución:

Función 1 - Venta de boletos

Cliente 1, ¿cuántos boletos desea (1-3)? 2

Cliente 2, ¿cuántos boletos desea (1-3)? 3

Cliente 3, ¿cuántos boletos desea (1-3)? 1

Cliente 4, ¿cuántos boletos desea (1-3)? 1

Cliente 5, ¿cuántos boletos desea (1-3)? 3

Función 2 - Venta de boletos

Cliente 1, ¿cuántos boletos desea (1-3)? 2

Cliente 2, ¿cuántos boletos desea (1-3)? 3

Cliente 3, ¿cuántos boletos desea (1-3)? 1

Cliente 4, ¿cuántos boletos desea (1-3)? 1

Cliente 5, ¿cuántos boletos desea (1-3)? 2

Total de boletos vendidos en el día: 19

▼ 🕵️ Pista

Necesitarás:

- Un **ciclo externo** para las funciones del cine.
- Un **ciclo interno** para los clientes que compran boletos.
- Una variable que acumule la **suma total de boletos comprados**.

Parte 2: Patrones Comunes con Ciclos While

Como hemos ido viendo en los ejercicios anteriores, los ciclos while sirven para programar muchas situaciones diferentes. Veamos algunos patrones comunes de uso del ciclo while:

Patrón 1: Cantidad fija de repeticiones

Cuando conocemos la cantidad exacta de repeticiones que vamos a necesitar hacer, podemos utilizar un **contador** para saber cuándo salir del ciclo. Por ejemplo:

```
contador = 0
while contador < 10:
    sentencias a ejecutar
    contador += 1
```

Initializamos en cero para comenzar a contar.
En este caso, queremos repetir 10 veces el ciclo.
Cada vez que iteramos, aumentamos en 1 nuestro contador

Veamos cómo aplicar este patrón en algunos ejercicios:

1. Completa el código para hacer una cuenta regresiva antes de un despegue del 10 al 1.

```
contador = _____
while _____:
    print("Despegue en...", contador)
    contador _____ 1
    print("¡Despegue! 🚀")
```

2. Vamos a crear un ciclo que pida las calificaciones de 5 estudiantes, y luego calcule el promedio de todas las calificaciones. La cantidad de calificaciones es fija (5), por lo que el ciclo se debe repetir 5 veces.

- a. Completa el siguiente código para que calcule correctamente el promedio:

```
calificaciones_totales = 0
num_estudiantes = _____
while _____:
    calificacion = int(input("Ingrese la calificación del estudiante: "))
    calificaciones_totales += calificacion
    num_estudiantes _____
```

```
promedio = _____  
print("El promedio de calificaciones es:", promedio)
```

- b. Modifica el código anterior para que, si el usuario ingresa una calificación inválida (que sea menor a 0 o mayor a 100), se le avise por consola con el mensaje "Calificación inválida. Debe estar entre 0 y 10.", y pueda ingresar la nota nuevamente.

▼  **Pista**

Vas a necesitar un condicional del tipo `if-else` dentro del ciclo `while`.

Patrón 2: Contadores

También podemos utilizar los **contadores** para ir guardando el valor acumulado de alguna variable:

```
contador = 0 → Inicializamos en cero para comenzar a contar.  
while condición:  
    sentencias a ejecutar  
    contador += valor → Cada vez que iteramos, vamos aumentando nuestro contador el algún valor.
```

Podemos utilizar distintos contadores para ir guardando el total de diferentes cosas en nuestro programa.

Veamos cómo aplicar este patrón en un ejercicio:

1. En un sistema de autenticación, se le da al usuario 5 intentos para ingresar correctamente su contraseña. Si se alcanzan los 5 intentos fallidos, el sistema debe bloquear al usuario.

Escribe un programa que se comporte como el sistema descrito, asumiendo que la contraseña correcta es "12345" (la contraseña más segura, obvio). Guíate por los siguientes ejemplos de ejecución:

```
Ingrese la contraseña: 1234  
Contraseña incorrecta. Intentos fallidos: 1  
Ingrese la contraseña: hola123  
Contraseña incorrecta. Intentos fallidos: 2  
Ingrese la contraseña: contraseña  
Contraseña incorrecta. Intentos fallidos: 3
```

```
Ingrese la contraseña: 7654  
Contraseña incorrecta. Intentos fallidos: 4  
Ingrese la contraseña: 9876abc  
Contraseña incorrecta. Intentos fallidos: 5  
Se han alcanzado 5 intentos fallidos. El acceso está bloqueado.
```

```
Ingrese la contraseña: hola  
Contraseña incorrecta. Intentos fallidos: 1  
Ingrese la contraseña: contraseña  
Contraseña incorrecta. Intentos fallidos: 2  
Ingrese la contraseña: 12345  
Acceso permitido.
```

▼ 🔍 Pista

Vas a necesitar un condicional del tipo `if-else` dentro del ciclo `while`. Y la condición del ciclo while deberá incluir 2 cosas: que aún queden intentos **y** que no se haya ingresado la contraseña correcta.

Patrón 3: Valor específico como condición

También podríamos necesitar que una variable sea distinta de cierto valor para continuar nuestro ciclo. Por ejemplo, si necesitaramos iterar hasta ingresar un valor específico:

```
variable = input()  
while variable != valor:  
    sentencias a ejecutar  
    variable = input()
```

Leemos algún valor de la variable (podría ser otro tipo de dato también)

Mientras variable sea distinto a algún valor específico (cualquiera).

Al final del ciclo actualizamos el valor de la variable (o podríamos generar un loop infinito).

Veamos cómo aplicar este patrón en un ejercicio:

1. Un mago ha escondido un número secreto y solo revelará su hechizo más poderoso a quien lo adivine. Los jugadores deben intentar descubrir el número correcto. Escribe un programa que genere un número aleatorio entre el 1 y el 10 como número secreto. Luego, deberá comenzar a pedirle

al usuario ingresar un número hasta que adivine. Guíate por el siguiente ejemplo de ejecución:

```
Adivina el número mágico (entre 1 y 10): 2
¡Incorrecto! Inténtalo de nuevo.
Adivina el número mágico (entre 1 y 10): 3
¡Incorrecto! Inténtalo de nuevo.
Adivina el número mágico (entre 1 y 10): 4
¡Incorrecto! Inténtalo de nuevo.
Adivina el número mágico (entre 1 y 10): 9
🎩✨ ¡Has adivinado el número mágico! ✨🎩
```

▼ ¿No recuerdas cómo generar un número aleatorio?

La biblioteca `random` que posee la función `randint(x, y)` que genera un número al azar entero entre el rango `x` e `y`:

```
import random
aleatorio = random.randint(1,10) #genera un entero aleatorio entre el 1
```

Patrón 4: Variables como condición (Flags)

A veces, es necesario utilizar una **variable** como **condición de término** de un ciclo, la que definirá si el ciclo continúa o finaliza. A este tipo de condiciones se les suele llamar *flags* (banderas en inglés), ya que las variables actúan como una bandera que indica si el programa debe continuar o no. Es por eso que suelen tener valor `True` o `False`.

Por ejemplo:

```
debo_seguir = 1 ——————> Podría ser True
while debo_seguir:
    sentencias a ejecutar
    if condición de salida:
        debo_seguir = 0 ——————> Podría ser False
```

En estos casos es muy importante recordar cambiar el valor del *flag* dentro del ciclo, o se generará un loop infinito ∞ .

Este patrón podría ser útil si hay una o varias condiciones que deban ser tomadas en cuenta para salir del ciclo `while`.

Veamos cómo aplicar este patrón en un contexto específico: eres el cajero de una tienda y debes registrar los precios de los productos que un cliente está comprando. El sistema seguirá pidiendo precios hasta que el usuario indique que ha terminado la compra. Para ello, deberá ingresar `"fin"`. Al final se debe mostrar el total a pagar.

Un posible programa que resuelva este enunciado usando una variable como flag es:

```
total = 0 # Acumulador del total a pagar
comprando = True # Flag para controlar el ciclo

print("Bienvenido a la caja registradora. Ingresa los precios de los productos.")
print("Escribe 'fin' para terminar.")

while comprando:
    precio = input("Ingresa el precio del producto (o 'fin' para terminar): ")

    if precio.lower() == "fin":
        comprando = False # Cambia el flag para salir del ciclo
    else:
        total += float(precio) # Suma el precio al total

print("El total a pagar es:", total)
print("¡Gracias por tu compra!")
```

Parte 3: Desafíos

Desafío 1: Rayo que distingue entre los humanos y el pan integral

El Dr. Heinz Doofenshmirtz es un científico loco. Su nueva invención se conoce como "Rayo que distingue entre los humanos y el pan integral". Si bien este invento parece ser pacífico, es importante destacar que su funcionamiento se basa en el análisis sofisticado de las cadenas de ADN.



Figura 1: Dr. Heinz Doofenshmirtz y su nuevo invento.

Una cadena ADN es una secuencia de 4 bases: adenina (**A**), citosina (**C**), guanina (**G**), y timina (**T**). Por ejemplo: **AGTA ACCC TTAG CCGG**

El rayo dice que el objeto analizado es un **pan integral** cuando la **cantidad total de apariciones de las bases T y G es mayor que la cantidad de apariciones de las bases A y C**. Es un **humano** en caso contrario.

Se te propone simular el código de Dr. Doofenshmirtz. Tu programa debe recibir las bases, hasta que se ingrese "X". En este momento se debe mostrar si el objeto analizado es un humano o pan integral.

Para poder obtener el resultado, debe contar las apariciones de cada base en la cadena ADN ingresada. Guíate con los siguientes ejemplos de ejecución:

```
Ingrese la base: A
Ingrese la base: G
Ingrese la base: T
Ingrese la base: A
Ingrese la base: A
Ingrese la base: X
Resultado: el objeto analizado es un humano.
```

```
Ingrese la base: A
Ingrese la base: G
Ingrese la base: T
Ingrese la base: G
```

Ingrese la base: A

Ingrese la base: C

Ingrese la base: T

Ingrese la base: T

Ingrese la base: C

Ingrese la base: X

Resultado: el objeto analizado es un pan integral.

Ingrese la base: A

Ingrese la base: A

Ingrese la base: C

Ingrese la base: T

Ingrese la base: G

Ingrese la base: A

Ingrese la base: T

Ingrese la base: C

Ingrese la base: G

Ingrese la base: G

Ingrese la base: X

Resultado: no tenemos idea de qué es el objeto analizado.

Desafío 2: Clasificador de Ballenas

Juan Carlos Bodoque, un reconocido periodista estrella, en afán de ver las ballenas viajó a Punta Arenas. Ahí, los científicos marinos le contaron que las ballenas, en realidad, se clasifican según su tamaño:

- Ballena azul: 25-30[m]
- Ballena de aleta: 21-24[m]
- Ballena franca austral: 14-20[m]
- Ballena rorqual sol: menos de 14[m]



Figura 2: Señor Bodoque sorprendido por las riquezas del Sur.

Cuando Juan Carlos salió al mar, logró ver muchas ballenas, pero solo con la ayuda de un científico marino que lo acompañó logró medirlas todas. Una vez en Santiago, Bodoque contrató a un equipo de programadores para crear un programa que permite clasificar a las ballenas. Eres parte de este equipo.

Crea un programa que permita recibir el número de ballenas a analizar y el tamaño de cada ballena en metros. Por cada una debe indicar qué tipo de ballena es.

Guíate con el siguiente ejemplo de ejecución:

```
Cuantas ballenas quieres analizar? 3
Ingresa el tamaño de la ballena: 13
Es una ballena rorcual sol
Ingresa el tamaño de la ballena: 29
Es una ballena azul
Ingresa el tamaño de la ballena: 22
Es una ballena de aleta
Fin de programa creado para J.C.Bodoque
```

Desafío 3: Nuevo Calculador de Promedio Avanzado V1.0 de Goddard

Con cada año que pasa, los colegios y universidades usan más tecnologías para sus clases. A su vez, los alumnos y alumnas también inventan más maneras de usar la tecnología disponible para organizar sus estudios. Por ejemplo, muchos usan calendarios digitales, o toman notas en sus computadores o tables.

Jimmy Neutron, un ingenioso alumno universitario, quiere instalar un nuevo programa a su perro robótico Goddard: una calculadora de promedio semestral de un ramo llamado Komputación Snekítífica KIWI285:



Figura 3: Goddard, el mejor amigo de Jimmy

Para calcular el promedio del ramo se necesita saber las notas de los 3 certámenes y el promedio aritmético de las N tareas:

$$P_{tareas} = \frac{T_1 + T_2 + \dots + T_n}{n}$$

$$P_{certamen} = \sqrt[3]{C_3 \left(\frac{C_1 + C_2}{2} \right)^2}$$

$$P_{ramo} = 0,75P_{certamen} + 0,25P_{tarea}$$

Escribe un programa que reciba primero las notas de los 3 certámenes. Después debe recibir como entrada varios números enteros entre 0 y 100, que indican las notas de N tareas. Una vez que se ingrese -1, el programa debe imprimir el promedio del ramo.

Tip: La raíz n-ésima es lo mismo que elevar un número a $1/n$.

Guíate con el siguiente ejemplo de ejecución:

Ingresé la nota del certamen 1: 70

Ingresé la nota del certamen 2: 50

Ingresé la nota del certamen 3: 55

```
Ingrese la nota de la tarea: 90
Ingrese la nota de la tarea: 91
Ingrese la nota de la tarea: 55
Ingrese la nota de la tarea: 70
Ingrese la nota de la tarea: -1
Nota final: 62.83
```

Desafío 4: La tienda de Doña Lucía

Doña Lucía tiene una tienda y le pide ayuda para ordenar sus cuentas. Quiere tener un registro del monto total que gana al día, y del monto que gasta cada cliente en promedio.



Figura 4: Doña Lucía feliz en su tienda.

Para esto, necesita un programa que pida el nombre de un cliente, cuántos artículos lleva, y solicite de a uno los precios de los artículos. Al terminar de ingresar los artículos de ese cliente, el programa deberá solicitar otro. Se deberán solicitar clientes hasta que se ingrese la palabra “cerrar” o hasta que se iguale o supere la meta de ventas del día (\$200.000), momento en que se cerrará la tienda.

Cuando finalice el programa, se deberá mostrar el monto total ganado en el día, y la cantidad de dinero promedio que gastan cada cliente.

Guíate por los siguientes ejemplos de ejecución:

Ingrese cliente: Felipe
Ingrese cantidad de artículos: 1
Ingrese precio: 5000

Ingrese cliente: Camila
Ingrese cantidad de artículos: 2
Ingrese precio: 10000
Ingrese precio: 5000

Ingrese cliente: cerrar

Total del día: 20000
Promedio gastado por cliente: 10000

Ingrese cliente: Carla
Ingrese cantidad de artículos: 1
Ingrese precio: 150000

Ingrese cliente: Alejandro
Ingrese cantidad de artículos: 2
Ingrese precio: 50000
Ingrese precio: 30000

Total del día: 230000
Promedio gastado por cliente: 115000

▼ 00 Pista

Vas a necesitar utilizar ciclos anidados: el ciclo externo será para ir pidiendo clientes hasta que se cumpla alguna de las dos condiciones (que se llegue a la meta, o que se cierre la tienda), y el ciclo interno para pedir cada uno de los artículos de un cliente.

Desafío 5: Tiempo de reverberación

Una ingeniera en sonido quiere asegurarse de que el tiempo de reverberación de una sala de clase sea adecuado, ya la universidad ha tenido quejas de que se genera mucho eco.

El **tiempo de reverberación** de un recinto es el tiempo que tarda el sonido en disminuir **60 decibeles** después de que la fuente sonora se ha apagado. En términos simples, si un lugar tiene un **tiempo de reverberación alto**, los sonidos persisten más tiempo y pueden generar eco. Si el **tiempo de reverberación es bajo**, el sonido se apaga rápidamente, lo que es ideal para lugares como salas de grabación o teatros.



Figura 5: La ingeniera en sonido reventando globitos para ver qué tanto eco tiene la sala de clases.

Para medir el tiempo de reverberación de un recinto se requieren de **10 mediciones** de la respuesta impulsiva. La respuesta impulsiva es una señal de audio que contiene información del **tiempo de reverberación (en segundos)**. Para que la medición sea válida, **el voltaje máximo de la señal no puede sobrepasar los 24 dBu**.

El **tiempo de reverberación final** es el promedio de los tiempos de reverberación de 10 mediciones **válidas**.

Genera un programa que ayude a la ingeniera en sonido, recibiendo **10 mediciones válidas**. Para esto, en cada medición se debe pedir el voltaje máximo y el tiempo de reverberación. Si es que el voltaje sobrepasa lo permitido, se debe indicar que la medición fue errónea y medir nuevamente. Al final, el programa debe imprimir el tiempo de reverberación total.

Guíate por el siguiente ejemplo de ejecución:

Medición 1

Ingrese voltaje (dBu): 25

Ingrese tiempo (s): 3

Medición inválida, intente nuevamente.

Ingrese voltaje (dBu): 24.9

Ingrese tiempo (s): 1.5

Medición inválida, intente nuevamente.

Ingrese voltaje (dBu): 23.9

Ingrese tiempo (s): 1.36

Medición 2

Ingrese voltaje (dBu): 23.7

Ingrese tiempo (s): 2.5

Medición 3

Ingrese voltaje (dBu): 23.75

Ingrese tiempo (s): 1.67

Tiempo de reverberación final (s): 1.269

▼  **Pista**

Vas a necesitar utilizar ciclos anidados: el ciclo externo será para obtener las 10 mediciones, y el ciclo interno por si una medición es inválida, se repita hasta obtener un valor válido.