

Control de Flujo

Control de Flujo	1
¿Qué aprenderás?	2
Introducción	2
La instrucción IF	3
¿Qué hacer si una condición no se cumple?	5
¿Qué hacer si se quiere analizar más de 2 casos?	6
Ejercicio guiado 1: Números pares e impares	7



¡Comencemos!

¿Qué aprenderás?

- Reconocer las sentencias condicionales para la construcción de programas.
- Codificar un programa en Python utilizando diagramas de flujo con iteraciones para la implementación de un algoritmo que resuelve un problema.
- Codificar un programa en Python utilizando instrucciones de bloque para manejar el flujo en base a condiciones lógicas.

Introducción

En este capítulo se introducirán las instrucciones **If/elif/else en Python** los cuales permitirán generar estructuras condicionales para la implementación de programas de mayor complejidad. Las estructuras condicionales añaden flexibilidad a nuestro código permitiendo que algunas partes se ejecuten dependiendo de si ocurre o no ocurre una condición; y estas condiciones pueden depender de algún resultado, del ingreso de una variable o de la interacción del usuario.

Al inicio de la unidad hablamos y mostramos el siguiente diagrama de flujo, donde el rombo corresponde a una **evaluación condicional**; es en este punto donde se toma una decisión y el programa sigue en función de esa decisión. Y es por ello que en este capítulo aprenderemos a escribir programas que implementen estas decisiones.

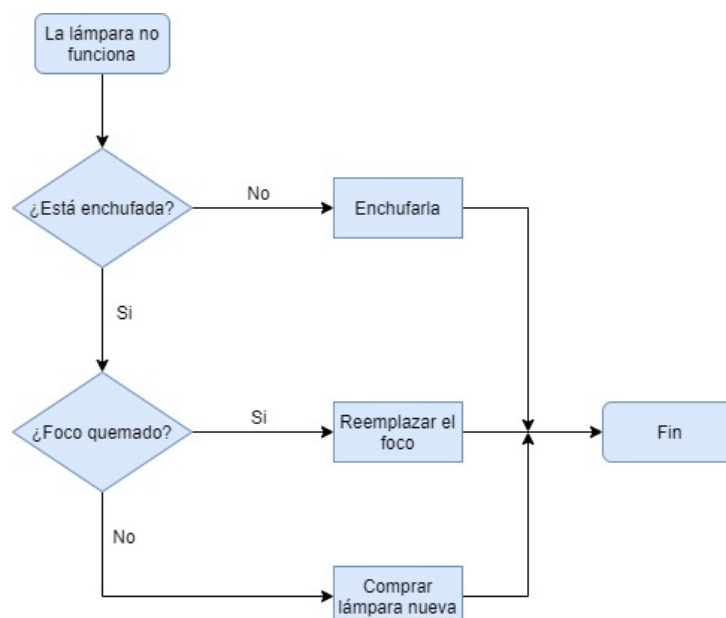


Imagen 1. Diagrama de una lámpara.

Fuente: Desafío Latam.

La instrucción IF

La instrucción `if` será la encargada de tomar una decisión en Python mediante una prueba lógica; normalmente en términos de control de flujo a estas pruebas lógicas se les denomina condiciones.

```
if condición:  
    # código que se ejecutará SÓLO si se cumple la condición
```

Lo anterior se lee de la siguiente manera: "Si se cumple la condición, **entonces** ejecuta el código". Python permite condiciones simples o condiciones compuestas tal como revisamos en el capítulo anterior.

Ejemplo:

```
edad = int(input("¿Qué edad tienes?"))  
  
if edad >= 18:  
    print("Eres mayor de edad")
```

```
¿Qué edad tienes? 33  
Eres mayor de edad
```



NOTA: Fíjate que el código que se ejecuta al cumplirse la condición se encuentra desplazado con respecto a la instrucción `if`. Esto se conoce como **indentación** y en Python está definida por convención en **4 espacios**, aunque hoy en día prácticamente todo editor de texto permite usar un **Tab**, el cual automáticamente se transformará en 4 espacios.

La indentación es fundamental en Python para determinar subcódigos que son parte de una Instrucción, por ejemplo:

```
edad = int(input("¿Qué edad tienes?"))  
  
if edad >= 18:  
    print("Eres mayor de edad")  
    print("El programa ha finalizado")
```

Si ejecutamos el programa, notaremos que en este caso se indica que se es mayor de edad y, además, se muestra que el programa finalizó:

```
¿Qué edad tienes? 33  
Eres mayor de edad  
El programa ha finalizado
```

Pero, si ejecutamos este programa no ocurre nada:

```
¿Qué edad tienes? 17
```

Esto sucede porque ambos `print()` son parte de la condición `if`, y solo se ejecutará si es que se cumple la condición estipulada.

Una propuesta más adecuada podría ser esta:

```
edad = int(input("¿Qué edad tienes?"))  
  
if edad >= 18:  
    print("Eres mayor de edad")  
  
print("El programa ha finalizado")
```



Nota: En este caso, el segundo `print` no se encuentra indentado, por lo tanto, independientemente de que se cumpla o no la condición, siempre indicará que el programa finalizó:

```
¿Qué edad tienes? 17  
El programa ha finalizado
```

¿Qué hacer si una condición no se cumple?

Quizás te preguntarás: ¿cómo podemos modificar nuestro programa para que muestre un mensaje cuando el usuario sea menor edad y otro mensaje cuando el usuario sea mayor de edad?

Una **buena práctica** es realizar un diagrama de flujo antes de comenzar a programar:



Imagen 2. Diagrama de flujo, mayor de edad 3.

Fuente: Desafío Latam.

Para implementarlo se introduce la instrucción **else**:

```
if condición:
    # código que se ejecutará SÓLO SI se cumple la condición
else:
    # código que se ejecutará si NO se cumple la condición
```

Lo anterior se lee de la siguiente manera: "**Si** se cumple la condición, **entonces** ejecuta el bloque de código, **sino** ejecuta el siguiente bloque de código".

```
edad = int(input("¿Qué edad tienes?"))

if edad >= 18:
    print("Eres mayor de edad")
else:
    print("Eres menor de edad")
```

¿Qué hacer si se quiere analizar más de 2 casos?

Tomando el caso anterior, es posible enfocarlo de la siguiente manera: “una persona puede ser mayor de 18 años, menor de 18 años o puede tener exactamente 18 años”.

Para eso, Python nos provee de una instrucción híbrida adicional llamada `elif`, la que nos permite realizar una evaluación condicional cuando no se cumplió una evaluación previa.

Además, es posible combinar `if` con uno o varios `elif`, además de un `else`.

```
edad = int(input("¿Qué edad tienes?"))

if edad > 18:
    print("Tienes más de 18 años")
elif edad == 18:
    print("Tienes 18 años")
else:
    print("Tienes menos de 18 años")
```

Un aspecto que hay que tener presente, es que las distintas condiciones se irán evaluando en orden secuencial, y el bloque terminará ejecutando el código asociado a la primera condición que se cumpla.

Ejercicio guiado 1: Números pares e impares

Supongamos que queremos determinar si el número que nuestro usuario ingresa es par o impar:

1. Abre tu editor de texto:
2. Solicitamos un valor al usuario de manera interactiva:

```
valor = int(input("Ingresa el valor a probar: "))
```

Nótese que estamos envolviendo en la función `int()` nuestro `input` para que al ingresar los datos desde la consola o terminal, estos sean convertidos a un entero dado que la función `input` por defecto retorna un `string` o cadena de texto.

3. Determinemos cual es la lógica para encontrar los pares:

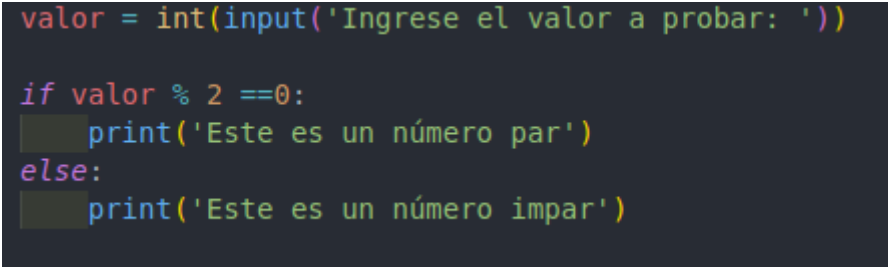
Sabemos que un número par se caracteriza por ser divisible por 2, por ello, si lo traducimos en una prueba lógica quedaría como:

```
valor % 2 == 0
```

4. Ahora armemos las decisiones:

```
if valor % 2 == 0:  
    print("Este es un número par")  
else:  
    print("Este es un número impar")
```

El código completo se ve así:

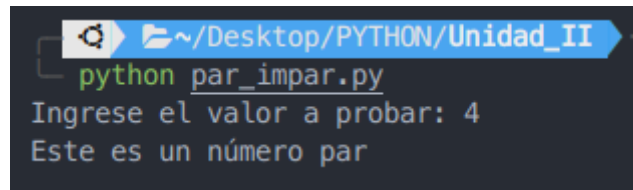


```
valor = int(input('Ingrese el valor a probar: '))  
  
if valor % 2 == 0:  
    print('Este es un número par')  
else:  
    print('Este es un número impar')
```

Imagen 3: Ejecución del código

Fuente: Desafío Latam

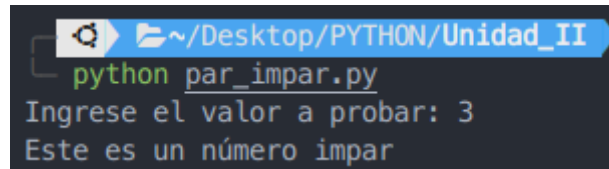
Si probamos su funcionamiento:



```
~/Desktop/PYTHON/Unidad_II
python par_impar.py
Ingrese el valor a probar: 4
Este es un número par
```

Imagen 4: Revisando para un valor par

Fuente: Fuente Desafío Latam

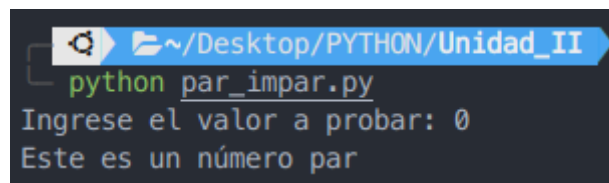


```
~/Desktop/PYTHON/Unidad_II
python par_impar.py
Ingrese el valor a probar: 3
Este es un número impar
```

Imagen 5: Revisando para un valor impar

Fuente: Desafío Latam

Vamos más allá, ¿qué pasa entonces si nuestro programa lo probamos con el valor 0? Esto siempre es un problema porque para algunos el 0 no es par ni tampoco es impar, ¡es un cero! Ahora bien, ¿qué es lo que dice nuestro programa al respecto?



```
~/Desktop/PYTHON/Unidad_II
python par_impar.py
Ingrese el valor a probar: 0
Este es un número par
```

Imagen 6: Revisando para el cero

Fuente: Desafío Latam

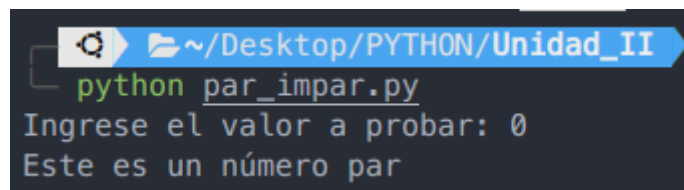
Si probamos con el cero, nuestro programa dice que es un número par, lo cual no está completamente correcto. La razón de este error, es porque la manera de encontrar nuestros pares es de manera indirecta, y para el caso particular $0 \% 2$ es cero, por lo tanto, nuestro programa está ejecutando de manera correcta lo que le pedimos, pero no está entregando el resultado que esperamos.

Para solucionar esto, nos basta con agregar dicha condición, naturalmente, podemos pensar en agregarla como un `elif`, previo al `else`.

```
valor = int(input('Ingrese el valor a probar: '))

if valor % 2 == 0:
    print('Este es un número par')
elif valor == 0:
    print('Este número es cero')
else:
    print('Este es un número impar')
```


Si ejecutamos el programa de esta manera se obtiene lo siguiente:



```
~ / Desktop / PYTHON / Unidad_II
python par_impar.py
Ingrese el valor a probar: 0
Este es un número par
```

Imagen 7: Analizando el problema
Fuente: Desafío Latam

Como se puede observar, se obtiene el mismo resultado, que el 0 es considerado par, a pesar de que se agregó una condición extra.

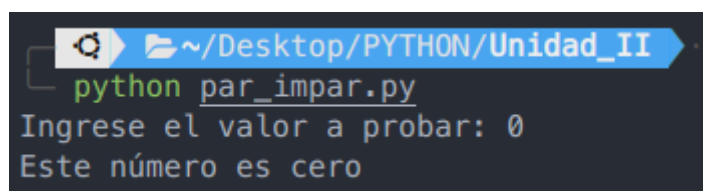
La razón por la que esto ocurre es que en caso de tener múltiples condiciones, estas se van a ir evaluando de manera secuencial, y ya sabemos que para un valor igual a `cero valor % 2` es igual a cero, por lo tanto, nunca llegará a cumplirse la segunda condición agregada.

Para solucionar este problema es necesario entonces intercambiar el orden de las condiciones de la siguiente manera:

```
valor = int(input('Ingrese el valor a probar: '))

if valor == 0:
    print('Este número es cero')
elif valor % 2 == 0:
    print('Este es un número par')
else:
    print('Este es un número impar')
```

Al realizar este simple cambio, se obtiene el resultado esperado:



```
~ / Desktop / PYTHON / Unidad_II
python par_impar.py
Ingrese el valor a probar: 0
Este número es cero
```

Imagen 8: Solución al problema
Fuente: Desafío Latam

Vamos aún más allá; si bien nuestro programa funciona como esperamos, aún existen oportunidades de mejora, y una de ellas es con respecto al uso de números negativos. Los números pares e impares normalmente son utilizados dentro del rango positivo.

Podríamos, por ejemplo, hacer una advertencia, o decir que normalmente estos criterios se usan en números positivos.

```
valor = int(input('Ingrese el valor a probar: '))

if valor < 0:
    print('Cuidado, los criterios de paridad normalmente se utilizan con  
números positivos.')
    print('Usar el resultado con Precaución.\n')

if valor == 0:
    print('Este número es cero')
elif valor % 2 == 0:
    print('Este es un número par')
else:
    print('Este es un número impar')
```

En este caso, se agregó un texto que advierte al usuario, el que solo se mostrará en el caso de que el valor sea negativo. Posteriormente, utilizamos nuestro programa que independientemente del valor, siempre responderá si se trata de un valor par o impar.

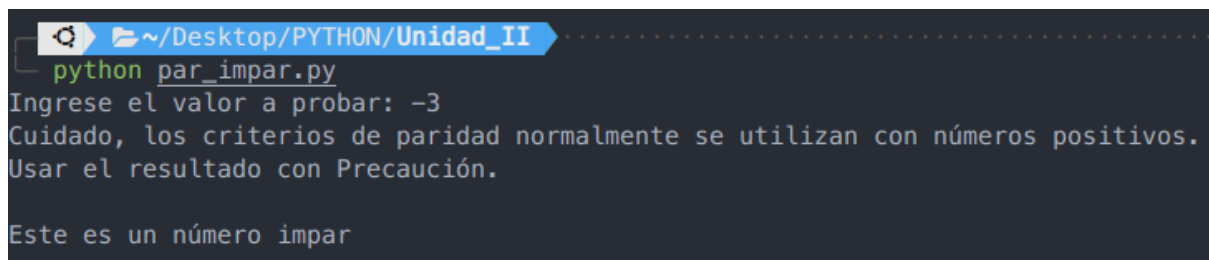


Imagen 9: Revisando el comportamiento del programa
Fuente: Desafío Latam

En la plataforma tendrás el código de este ejercicio con el nombre **Ejecución guiada - Números pares e impares**