Índice

1 | Fundamentos de programación

1.1 | Conceptos básicos

Python es un lenguaje de programación interpretado, de alto nivel y con semántica dinámica, esto nos indica que al momento de crear una variable en memoria, automáticamente asignará el tipo de variable a ese objeto. Entre las características más destacables de Python se mencionan las siguientes:

- **Es orientado a objetos: La programación orientada a objetos (POO) es notablemente fácil de aprender y aplicar con Python.**
- **Es libre: Es software de código abierto, no existen restricciones para copiarlo, integrarlo en sus sistemas o enviarlo con sus productos.**
- **Es portable: Python está escrito en ANSI C portátil, por eso se compila y ejecuta en prácticamente todas las plataformas principales como Windows, Linux, MacOS, entre otras.**
- **Es poderoso: Su conjunto de herramientas lo sitúa entre lenguajes de secuencias de comandos tradicionales (como Tcl, Scheme y Perl) y lenguajes de sistemas (como C, C++ y Java).**
- **Es combinable: Python puede fácilmente integrarse a componentes escritos en otros lenguajes de programación.**
- **Es fácil de usar: La combinación de respuesta rápida y simplicidad del lenguaje de Python hace que la programación sea más divertida que el trabajo.**
- **Es fácil de aprender: De hecho, puedes esperar estar codificando programas significativos en Python en cuestión de días (y quizás en sólo horas, si ya eres un programador experimentado).**

En Python hay una gran colección de funcionalidades preconstruidas y portátiles, conocidas como librerías estándares, las cuales se pueden invocar con "import". Estas librerías admiten una serie de tareas de programación a nivel de aplicación, desde la comparación de patrones de texto hasta la creación de scripts de red. Entre las librerías que usaremos se mencionan las siguientes:

- time: **Este módulo ofrece varias funciones relacionadas con el tiempo.**
- pin: **Controla los pines de E/S, también conocido como General-Purpose Input/Output (GPIO). Los objetos pin se asocian comúnmente con un pin físico que puede controlar un voltaje de salida y leer voltajes de entrada.**
- dht: **El módulo dht proporciona funciones relacionadas con la lectura del sensor de temperatura y humedad de la serie dht.**
- ujson: **UltraJSON es un codificador y decodificador JSON escrito en C puro con enlaces para Python 3.7+.**

Además, escribir en el lenguaje de Python es casi como escribir en pseudocódigo en inglés. Por ejemplo:

Pseudocódigo

```
Si e no está en [12,34,25,17,24]
    imprimir"No está en la lista"
```

Python 3

```
if e not in [12,34,25,17,24]:
    print("No está en la lista")
```

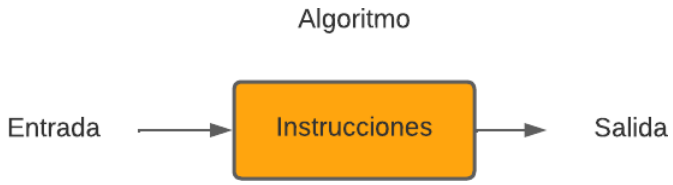

Figura 1.1: **Algoritmo**

1.2 | Intérprete y compilador

1.2.1 | Intérprete

Un intérprete es un tipo de programa que ejecuta otros programas. Cuando escribes un programa Python, el intérprete de Python lee tu programa y ejecuta las instrucciones que contiene. En efecto, el intérprete es una capa de lógica de software entre tu código y el hardware de tu máquina. Cuando el paquete de Python se instala en su máquina, genera una serie de componentes -mínimamente, un intérprete y una librería de apoyo-. Dependiendo de cómo lo utilices, el intérprete de Python puede adoptar la forma de un programa ejecutable o de un conjunto de librerías vinculadas a otro programa [1].

1.2.2 | Compilador

Un compilador traduce un programa a instrucciones específicas, es decir traduce código fuente de alto nivel a lenguaje de bajo nivel como lenguaje de máquina. Compilan y enlazan programas completos. La principal diferencia entre el intérprete y el compilador radica en el rendimiento. Una vez que se ejecuta el programa, los servicios del compilador ya no son necesarios, mientras que el intérprete continúa utilizando los recursos informáticos. En Python, el compilador está siempre presente en tiempo de ejecución y forma parte del sistema que ejecuta los programas.

1.3 | Algoritmos y sus representaciones

1.3.1 | Algoritmo

Un algoritmo es una sucesión ordenada de instrucciones que deben ejecutarse para resolver un problema en un tiempo finito. Su estructura se basa en la entrada de datos, estos son transformados produciendo una serie de acciones que se convertirán en la salida del algoritmo, como se muestra en la Figura1.1.

Para desarrollar algoritmos se debe usar una metodología que permita resolver problemas de forma organizada. Primero definir el objetivo, luego identificar los componentes o variables y finalmente escribir las instrucciones que permitan llegar al objetivo propuesto. Se deben realizar pruebas para validar que el resultado sea el correcto.

1.3.2 | Ejemplo de algoritmo

Desarrollar un algoritmo que permita comprar entradas para la película Avengers desde la aplicación de Supercines. Para realizar la compra tenemos la información de una tarjeta de débito.

Para desarrollar el algoritmo primero identificamos nuestra entrada y nuestra salida esperada.

- Entrada: **Información de la tarjeta de débito.**
- Salida esperada: **Entradas para la película.**

Algoritmo

1. **Ingresar a la aplicación de supercines**
2. **Seleccionar la película Avengers**
3. **Seleccionar el lugar**
4. **Seleccionar la hora de la película**
5. **Ingresar los datos de la tarjeta de débito para realizar el pago**
6. **Finalizar la compra**

1.3.3 | Construcción de algoritmos

Para poder construir algoritmos computacionales se pueden usar diferentes notaciones. En este caso aprenderemos una notación simple y clara para ser usada en problemas básicos. Esta notación es útil cuando se está aprendiendo a diseñar algoritmos, una vez que se tenga mejor dominio del desarrollo, puede ser omitida.

Para esto se debe identificar los componentes que la comprende, cada componente incluye una instrucción simple o un conjunto de instrucciones. Para la representación gráficas representaremos cada componente como un bloque como se muestra en la Figura1.2.

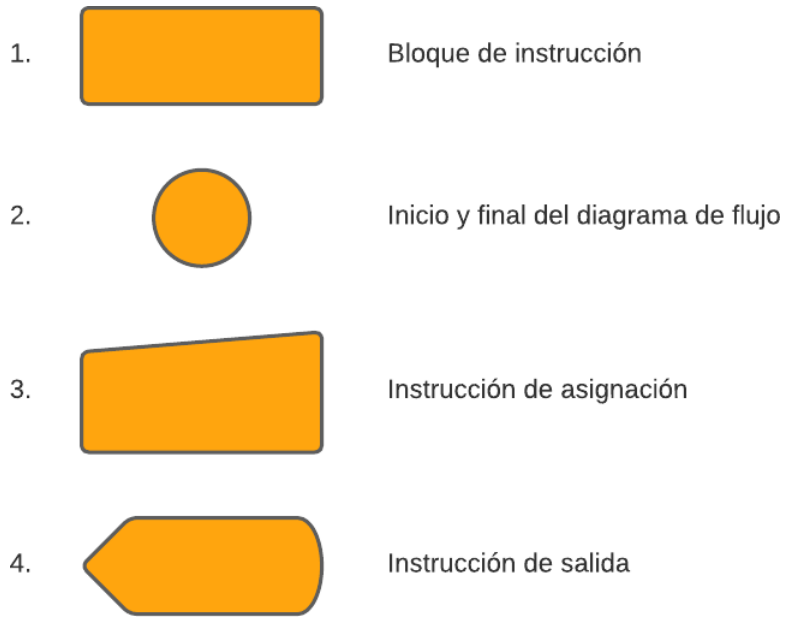

Figura 1.2: **Bloques usados para elaborar diagramas de flujo**

Un algoritmo no solo cuenta con una instrucción, sino una serie organizada de instrucciones que son ejecutadas secuencialmente, por lo que se muestra esta organización a través de líneas de flujo que unen cada uno de los bloques del componente. Además, encontramos los siguientes símbolos en los diagramas de flujo.

1. **Bloque de instrucción: En este bloque se representa una acción o línea de código, usualmente se realizan operaciones con los operadores presentados más adelante.**
2. **Inicio y final del diagrama de flujo: Este bloque representa el inicio o fin del algoritmo.**

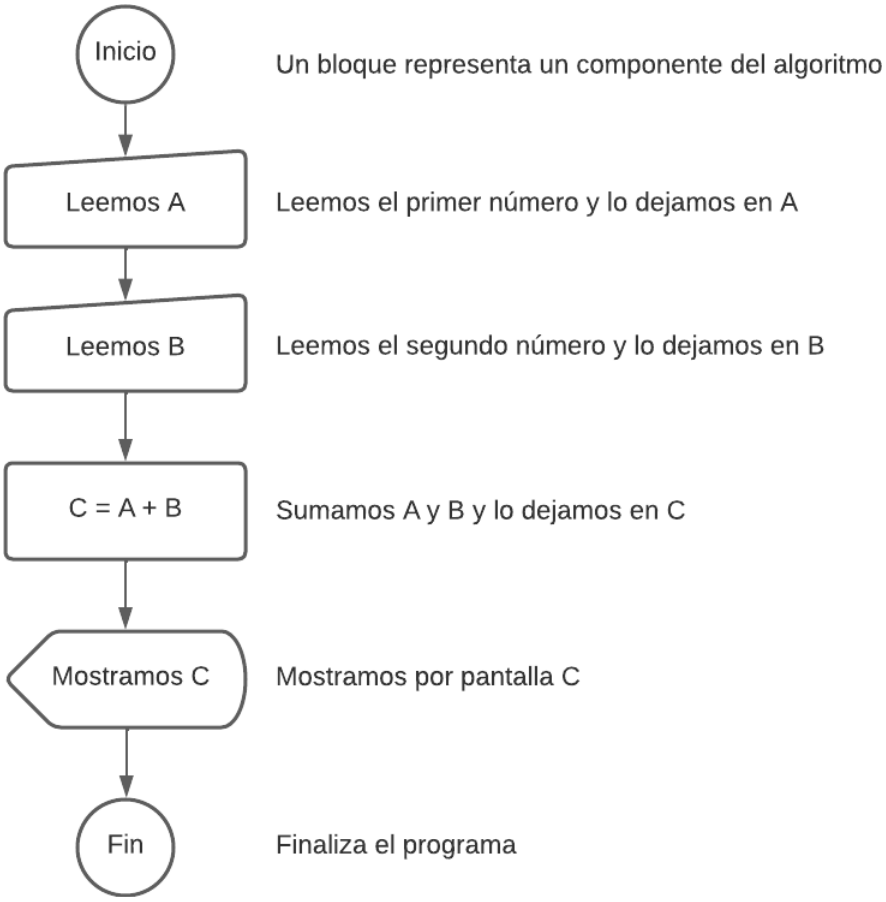

Figura 1.3: **Ejemplo de diagrama de flujo**

3. **Instrucción de asignación: Este bloque permite representar la entrada por teclado, o asignación de un valor a una variable.**
4. **Instrucción de salida: Este bloque representa la salida esperada del algoritmo. También se puede usar para presentar una salida por pantalla.**
5. **Líneas de flujo: Estas permiten conectar los diferentes bloques para mostrar la secuencia que debe seguir el algoritmo.**

Un ejemplo de un diagrama se muestra en la Figura1.3.

1.4 | Ejercicios

Realizar los diagramas de flujo para representar los algoritmos de los siguientes procesos:

- **Ingrese el nombre del usuario y se guarde en la variable** *nombre*. **Luego ingrese el apellido del usuario y se guarde en la variable** *apellido*. **Por último, unir estas dos variables en una nueva variable llamada** *nombre_completo* **y mostrarlo por pantalla.**
- **El programa pide el costo de 3 productos A, B, y C. Se debe realizar la suma de los 3 productos y mostrar el resultado por pantalla.**
- **El programa pide el ingreso de la temperatura en Fahrenheit y retorna la temperatura convertida en Celsius.**

2 | Tipos de datos

2.1 | Datos primitivos

Los datos primitivos son los que cubren los formatos básicos de los elementos que se utilizan para formar un programa. Estos son los caracteres numéricos y las cadenas de caracteres.

Los tipos de datos primitivos son los siguientes:

1. int (enteros): **Son números sin punto decimal. Ejemplo: 4, 89, 230, -30**
2. float(númerosdepuntoflotante):**Números con punto decimal.**
   Ejemplo: 20.6, 0.36, -52.9
3. string(cadenadecaracteres):**Hace referencia a un conjunto de caracteres, estas pueden ser una letra, una palabra, una oración, un párrafo, etc.; que son establecidos entre comillas simples o dobles.**
   Ejemplo: "Hola", "Welcome to python"
4. bool(verdaderooFalso):**Este tipo de dato acepta valores por verdad = True o falso = False**

2.2 | Variables

Las variables son contenedores de información que puede ir cambiando en el transcurso del tiempo de ejecución de un programa. Una variable está formada por un nombre que esté relacionado con el programa a desarrollar. Un valor, éste será el contenido de la variable. El nombre de la variable puede estar conformado por letras, números o el caracter guión bajo.

Por ejemplo, humedad, HUMEDAD, humedad_jardin.

Para python humedad y HUMEDAD son variables diferentes, ya que este lenguaje reconoce las mayúsculas y minúsculas como diferentes.

Reglas

- **No debe iniciar con un número.**
- **El nombre no debe ser una** palabrareservada**de python. Las palabras reservadas tienen un significado especial para python por lo tanto no se debe usar como nombre de una variable.**
  Ejemplos de palabras reservadas:

print	input	False	True	None	as	asset	break	continue
def	del	elif	else	except	finally	for	from	global
if	try	while	with	yield	not	in	and	input

Buenas prácticas

- **Empezar el nombre de una variable con minúscula**
- **Elegir un nombre que represente el dato a guardar**
  Ejemplos:

- temperatura
- humedad

- **Nombres que contengan más de una palabra se debe capitalizar o separar con subguión cada palabra esta práctica se la conoce como** camelCase **y** snake_case **respectivamente.**
  Ejemplos:
  - **temperaturaCelsius**
  - **temperatura_celsius**

Para crear una variable simplemente se debe establecer el nombre de la variable y asignando valor mediante el signo igual, por ejemplo:

Python 3

```
temperatura_celsius=28
sensor="DHT11"
```

Para presentar el contenido de una variable se usa la función print(), por ejemplo:

Python 3

```
print(temperatura_celsius)
```

Salida

```
28
```

2.3 | Operadores Aritméticos

En python existen operadores aritméticos que nos permiten calcular un valor de dos operandos. Los operadores aritméticos disponibles en python son: suma, resta, multiplicación, división, división entera, módulo y potenciación.

Operador	Operación	Ejemplo	Resultado
+	Suma	2 + 4	6
-	Resta	8 - 5	3
*	Multiplicación	6 * 2	12
/	División	9 / 2	4.5
//	División Entera	9 // 2	4
%	Módulo	9 % 2	1
**	Potenciación	2 ** 3	8

Ejemplo Suma

```
temperatura_celsius=28
temperatura_celsius=temperatura_celsius+1
print(temperatura_celsius)
```

Salida

29

Prioridad de Operadores

En la sección anterior se muestran ejemplos de operaciones simples con un operador, en python es posible realizar una operación más compuesta, pero hay que tener en claro la prioridad de los operadores, el orden de operación es el siguiente, siendo el paréntesis de mayor prioridad, suma y resta el de menor:

- **Paréntesis ()**
- **Exponente ****
- **Multiplicación *, División /, División Entera //, Módulo %**
- **Suma +, Resta -**

Conversión de Fahrenheit a Grados Celsius

Por ejemplo, supongamos que se posee un sensor de temperatura que lee en unidades Fahrenheit si deseamos convertir a grados Celsius podríamos hacer lo siguiente:

$$C = \frac{5(F-32)}{9}$$

```
temperaturaF=82.4
temperaturaC=(5*(temperaturaF-32))/9
print(temperaturaC)
```

Salida

28

Ejercicio

```
a=3.0
b=4.0
c=(a**2+b**2)**0.5
print(c)

#¿Cuál será el resultado?
```

2.4 | Operadores lógicos

Estos símbolos se utilizan para construir expresiones lógicas, nos permiten trabajar con valores de tipo booleano. Un valor booleano o bool es un tipo que solo puede tomar valores True o False. Por lo tanto, estos operadores nos permiten realizar diferentes operaciones con estos tipos y su resultado será otro booleano.

Matemáticas	Símbolo Python
Conjunción	and
Disyunción	or
Negación	not

Las tablas de verdad son las siguientes:

1. Tabla de not

A	not A
True	False
False	True

Ejemplo

```
print( notTrue )
print( notFalse)
print( notnotnotnotTrue        )
```

Salida

```
False
True
True
```

2. Tabla de and

A	B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

Ejemplo 1

```
print( TrueandTrue    )
print( TrueandFalse     )
print( FalseandTrue     )
print( FalseandFalse      )
```

Salida

```
True
False
False
False
```

Ejemplo 2

```
x=0
y=10
print(x<5   and y>2)
```

Salida

```
True
```

Ejemplo 3

```
x=0
y=10
print(x and y)
```

Salida

```
0
```

3. Tabla de or

A	B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

Ejemplo 1

```
print( TrueorTrue )
print( TrueorFalse )
print( FalseorTrue )
print( FalseorFalse )
```

Salida

```
True
True
True
False
```

Ejemplo 2

```
x=0
y=10
print(x or y)
```

Salida

```
10
```

4. Ejemplo general

Ejemplo general

```
print(0 andnot 1 or1 andnot 0 or 1 and0)
```

Salida

```
True
```

2.5 | Conversión de datos

En Python, la conversión de tipos es un proceso en el que convertimos un literal de un tipo a otro. Las funciones incorporadas int(), float() y str() se utilizarán para el typecasting.

- int()
  Puede tomar un literal float o string como argumento y devuelve un valor de tipo class 'int'.
- float()
  Puede tomar como argumento un literal de int o de cadena y devuelve un valor de la class 'float'.
- str()
  Puede tomar un literal de float o int como argumento y devuelve un valor de la class 'str'.

1. Conversión de tipo Entero (int) a Flotante (float)
   El método de Python float() le permitirá convertir los enteros en flotantes.

Ejemplo 1

```
print(float(57))
```

Salida

```
57.0
```

También puede utilizar esto con una variable.

Ejemplo 2

```
f=68
print(float(f))
```

Salida

```
68.0
```

Ejemplo 3

```
n=100
f=float(n)
print(f)
```

Salida

```
100.0
```

2. Conversión de tipo Entero (int) a Cadena (str)

Podemos convertir números en cadenas usando el método str().

Ejemplo 1

```
print("El resultado es "+str(12))
```

Salida

```
El resultado es 12
```

También puede utilizar esto con una variable.

Ejemplo 2

```
numero=50
print("El resultado es "+str(numero))
```

Salida

```
El resultado es 50
```

Ejemplo 3

```
n=100
s="El resultado es "+str(n)
print(s)
```

Salida

```
El resultado es 100
```

3. Conversión de tipo Flotante (float) a Entero (int)

Python también incluye una función integrada para convertir flotantes en enteros: int().

Ejemplo 1

```
print(int(390.8))
```

Salida

```
390
```

Ejemplo 2

```
b=125.0
print(int(b))
```

Salida

```
125
```

Ejemplo 3

```
f=100.05
n=int(f)
print(n)
```

Salida

```
100
```

Cuando se convierten flotantes en enteros con la función int(), Python corta el decimal y los números restantes de un flotante para crear un entero. Aunque posiblemente desee redondear 390.8 a 391, Python no lo hará con la función int().

4. Conversión de tipo Flotante (float) a Cadena (str)

Cuando colocamos un flotante en el método str(), se mostrará un valor de cadena del flotante.

Ejemplo 1

```
print("El resultado es "+str(421.034))
```

Salida

```
El resultado es 421.034
```

Ejemplo 2

```
f=5524.53
print("El resultado es "+str(f))
```

Salida

```
El resultado es 5524.53
```

Ejemplo 3

```
f=100.05
s="El resultado es "+str(f)
print(s)
```

Salida

```
El resultado es 100.05
```

5. Conversión de tipo Cadena (str) a Flotante (float)

Queremos convertir estas cadenas en flotantes con la función float().

Ejemplo 1

```
totalPuntos="5524.53"
nuevosPuntos="45.30"
resultado=float(totalPuntos)+float(nuevosPuntos)
print(resultado)
```

Salida

```
5569.83
```

Ejemplo 2

```
s=   "132.65"
f=float(s)
print(f)
```

Salida

```
132.65
```

6. Conversión de tipo Cadena (str) a Entero (int)

Si su cadena no tiene posiciones decimales, probablemente querrá convertirla en un entero utilizando el método int().

Ejemplo 1

```
temperaturaPasada="34"
temperaturaActual="28"
diferencia=int(temperaturaPasada)-int(temperaturaActual)
print(diferencia)
```

Salida

```
6
```

Ejemplo 2

```
s="80"
i=int(s)
print(i)
```

Salida

```
80
```

Solo se realizará la conversión de datos si el contenido es compatible, sería ilógico intentar convertir t45 **a entero porque tiene una t, para algo así tendríamos que separar primero todo el texto.**

2.6 | Manejo de entrada y salida

Para que un programa sea útil, generalmente necesita comunicarse con el mundo exterior obteniendo datos de entrada del usuario y mostrando los datos de resultados al usuario. Un desarrollador puede querer tomar la entrada del usuario en algún punto del programa. Para hacer esto, Python proporciona la función llamada input()**.**

Sintaxis

```
variable=input('solicitud')
```

Donde *'solicitud'*, **es una cadena de caracteres** opcional, **que es mostrada al momento de pedir la entrada.**

2.7 | Obtener entrada del usuario

La entrada del usuario es lo que una persona ingresa en el teclado, ya sea eso un caracter, una flecha presionada o tecla enter, u otra cosa. Para convertirla a cualquier otro tipo de datos, tenemos que hacerlo explícitamente.

Python 3

```
# Tomando entrada del usuario
nombre =input( "Ingresa tu nombre: ")

# Salida
print("Hola, "+ nombre)
print(type(nombre))
```

Salida

```
Ingresa tu nombre: Alex
Hola, Alex
<class 'str'>
```

2.8 | Usar entradas con enteros

Como se puede observar Python toma todas las entradas como una entrada de cadena de forma predeterminada. Para convertirlo a cualquier otro tipo de datos, tenemos que convertir la entrada explícitamente. Por ejemplo, para convertir la entrada a int o float tenemos que usar el método int() **y** float() **respectivamente.**

Python 3

```
# Tomando entrada de usuario como entero
edad =int(input(   "Ingresa tu edad: "))
nueva_edad = edad + 1
# Salida
print("En tu siguiente cumpleaños tendrás"+nueva_edad+ "años")
```

Salida

```
Ingresa tu edad: 18
En tu siguiente cumpleaños tendrás 19 años
```

3 | Cadenas de caracteres

En términos de programación, generalmente llamamos *string* a una cadena. Cuando se piensa que una cadena es una colección de letras, el término tiene sentido. Todas las letras, números y símbolos de este folleto podrían ser una cadena. De hecho, su nombre podría ser una cadena, al igual que su número telefónico.

3.1 | Crear cadenas

En Python, creamos una cadena poniendo comillas alrededor del texto. Por ejemplo, podríamos etiquetar una cadena, así:

Python 3

```
monster = "¿Por qué los monstruos tienen tanto pelaje?"
Luego, para ver qué hay dentro de monster, podríamos escribirprint(monster),
como  esto:
print(monster)
```

Output

```
¿Por qué los monstruos tienen tanto pelaje?
```

También se pueden usar comillas simples para crear cadenas, así:

Python 3

```
monster = '¿Qué es morado y muy peludo?'
print(monster)
```

Salida

```
¿Qué es morado y muy peludo?
```

3.2 | Operaciones con cadenas

Python tiene una clase de cadena incorporada llamada str **con muchas funciones útiles. Las cadenas pueden ser modificadas usando diversos operadores. A continuación se presentan algunos de los operadores de cadena más comunes.**

Operador de concatenación "+" **Dos cadenas se pueden concatenar o unir usando el operador** + **en python, como se explica en el siguiente ejemplo:**

Python 3

```
string1="PyTime"
string2="IoT"
cadena=string1+string2
print(cadena)
```

Salida

```
PyTime IoT
```

Operador de repetición de cadenas "*"

La misma cadena se puede repetir en python n veces usando cadena*n, como se explica en el siguiente ejemplo:

Python 3

```
string1="PyTime"
print(string1*2)
print(string1*3)
print(string1*4)
print(string1*5)
```

Salida

```
PyTime PyTime
PyTime PyTime PyTime
PyTime PyTime PyTime PyTime
PyTime PyTime PyTime PyTime PyTime
```

Operador de división de cadenas "[]"

Se puede acceder a los caracteres de un índice específico de la cadena con el operador cadena[índice]. El índice se interpreta como un índice positivo a partir de 0 desde el lado izquierdo y un índice negativo a partir de -1 desde el lado derecho.

cadena	H	E	L	L	O
índice positivo	0	1	2	3	4
índice negativo	-5	-4	-3	-2	-1

- string[a] **Devuelve un carácter de un índice positivo de la cadena del lado izquierdo como se muestra en el gráfico de índice anterior.**
- string[-a] **Devuelve un carácter de un índice negativo a de la cadena del lado derecho como se muestra en el gráfico de índice anterior.**

- string[a:b] **Devuelve caracteres del índice positivo a al índice positivo b de como se muestra en el gráfico de índice anterior.**
- string[a:-b] **Devuelve caracteres del índice positivo a al índice negativo b de la cadena como se muestra en el gráfico de índice anterior.**
- string[a:] **Devuelve caracteres desde el índice positivo a hasta el final de la cadena.**
- string[:b] **Devuelve los caracteres desde el inicio de la cadena hasta el índice positivo b.**
- string[-a:] **Devuelve caracteres desde el índice negativo a hasta el final de la cadena.**
- string[:-b] **Devuelve los caracteres desde el inicio de la cadena hasta el índice negativo b.**
- string[::-1] **Devuelve una cadena en orden inverso.**

Python 3

```
string1="PyTime"
print(string1[1])
print(string1[-3])
print(string1[1:5])
print(string1[1:-3])
print(string1[2:])
print(string1[:5])
print(string1[:-2])
print(string1[-2:])
print(string1[::-1])
```

Salida

```
y
i
yTim
yT
Time
PyTim
PyTi
me
emiTyP
```

Operadores de comparación de cadenas “==” “!=”

El operador de comparación de cadenas en python se usa para comparar dos cadenas.

- **El operador “==” devuelve Boolean True si dos cadenas son iguales y devuelve Boolean False si dos cadenas no son iguales.**
- **El operador “!=” devuelve Boolean True si dos cadenas no son iguales y devuelve Boolean False si dos cadenas son iguales.**

Estos operadores se utilizan principalmente junto con la condición if para comparar dos cadenas en las que la decisión se debe tomar en función de la comparación de cadenas.

Python 3

```
string1="PyTime"
string2="PyTime, IoT"
string3="PyTime, IoT"
string4="IoT"
print(string1==string4)
print(string2==string3)
print(string1!=string4)
print(string2!=string3)
```

Salida

```
False
True
True
False
```

Operador de pertenencia “in” y “not in”

El operador de pertenencia se usa para buscar si el carácter específico es parte/miembro de una cadena de Python de entrada determinada.

- **“a” en la cadena: devuelve True booleano si “a” está en la cadena y devuelve False si “a” no está en la cadena.**
- **“a” no está en la cadena: devuelve True booleano si “a” no está en la cadena y devuelve False si “a” está en la cadena.**

Un operador de membresía también es útil para encontrar si una subcadena específica es parte de una cadena dada.

Python 3

```
string1="PyTimeIoT"
print("i" in string1)
print("I" in string1)
print("a" in string1)
print("a" notin     string1)
print("pytime" instring1)
print("Pytime" instring1)
print("pytime" notin    string1)
```

Salida

```
True
True
False
True
False
False
True
```

3.3 | Diferencia entre cadenas de caracteres y números

La entrada del usuario ingresa en Python como una cadena, lo que significa que cuando escribe el número 10 en su teclado, Python guarda el número 10 en una variable como una cadena, no como un número, pero ¿Cuál es la diferencia entre el número 10 y la cadena '10'? Ambos nos parecen iguales, con la única diferencia de que uno está rodeado de comillas. Pero para una computadora, los dos son muy diferentes.

Python 3

Por ejemplo, supongamos que comparamos el valor de la variable temperatura con un número:

```
comparar=temperatura==10:
```

Luego establecemos la variable temperatura en el número 10:

```
temperatura=10
comparar=temperatura==10:
print("¿Son iguales?", comparar)
```

Salida

```
¿Son iguales? True
```

Como puede ver, el resultado de la expresión booleana es `True`

Python 3

A continuación, establecemos temperatura en la cadena '10' (entre comillas), así:

```
temperatura=  "10"
comparar=temperatura==10:
print("¿Son iguales?",comparar)
```

Salida

```
¿Son iguales? False
```

Aquí, el resultado es `False` **porque Python no ve el número en comillas (una cadena) como un número. Afortunadamente, Python tiene funciones mágicas que pueden convertir cadenas en números y números en cadenas. Por ejemplo, puedes convertir la cadena '10' en un número con** `int()`**:**

Python 3

```
temperatura=  "10"
temperatura_convertida=int(temperatura)
comparar=temperatura_convertida==10:
print("¿Son iguales?", comparar)
```

Salida

```
¿Son iguales? True
```

3.4 | Operador de formato de cadena "%"

El operador de formato de cadena se utiliza para formatear una cadena según el requisito. Para insertar otro tipo de variable junto con la cadena, se usa el operador "%" junto a la cadena de python. "%" tiene el prefijo de otro carácter que indica el tipo de valor que queremos insertar junto con la cadena de Python. Consulte la tabla a continuación para conocer algunos de los diferentes especificadores de formato de cadena comúnmente utilizados:

OPERADOR	DESCRIPCIÓN
%d	Entero decimal con signo
%s	Cadena
%f	Número real de punto flotante

Python 3

```
sensor="DTH11"
temperatura=19.0
humedad=74

ejemplo1="El sensor usado es:  %s"%(sensor)
print(ejemplo1)

ejemplo2="Temperatura:  %f°C, Humedad:  %d"%(temperatura,humedad)
print(ejemplo2)
```

Salida

```
El sensor usado es: DTH11
Temperatura: 19.0 °C, Humedad: 74
```

Ejercicio

Realice un programa que usando las variables temperatura y humedad del ejemplo anterior, muestre la siguiente salida:

Salida

```
{ "temperatura": 19.0, "humedad":74 }
```

4 | Sentencia condicional

4.1 | Sentencia if

If es una estructura de control que nos permite ejecutar un bloque de código cuando se cumpla una expresión booleana. Podemos identificar dos partes principales en la estructura de control if (ver Figura4.1):

- Expresión booleana: **es la condición que se debe cumplir para que se ejecute bloque de código.**
- Bloque de código: **son las acciones a realizar en caso de cumplirse la expresión booleana.**

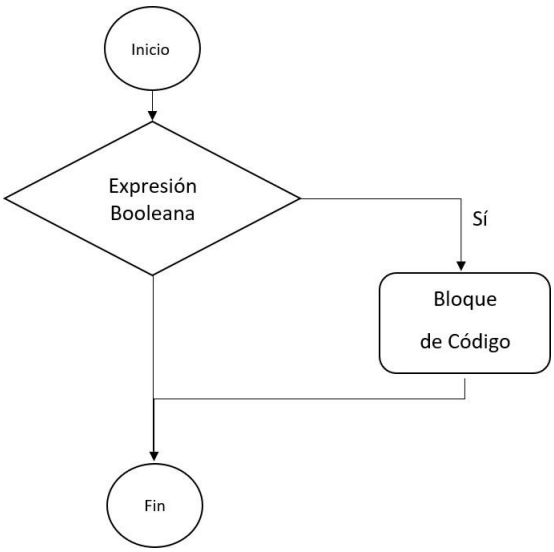

Figura 4.1: **Diagrama de flujo de la estructura de control if**

Por ejemplo, si el porcentaje de humedad del jardín es menor a 40 % se debe encender el sistema de riego, en un diagrama de flujo se observa de la siguiente manera

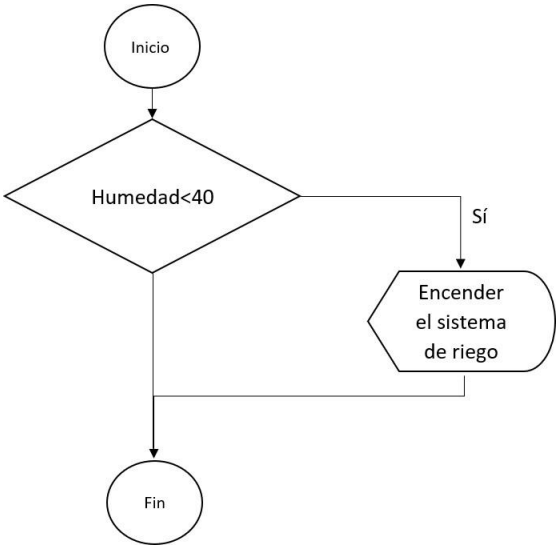

Figura 4.2: **Ejemplo de diagrama de flujo de sentencia condicional**

Python 3

```
humedad=34
print("Inicio del programa")
if humedad<40:
    print("Encender el sistema de riego")
print("Fin del programa")
```

Salida

```
Inicio del programa
Encender el sistema de riego
Fin del programa
```

Python 3

```
humedad=50
print("Inicio del programa")
if humedad<40:
    print("Encender el sistema de riego")
print("Fin del programa")
```

Salida

```
Inicio del programa
Fin del programa
```

4.2 | Sentencia if... else...

En la sección anterior observamos como se ejecutaba un bloque de código cuando la expresión en la sentencia if **daba como resultado** True**, caso contrario se continuaba con el programa. Si se desea ejecutar código en el caso que no se cumpla la sentencia** if**, podemos incluir la sentencia** else**, el diagrama sería el siguiente**

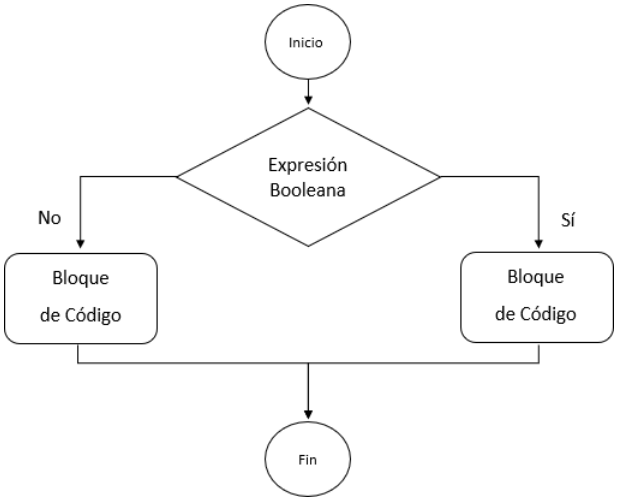

Figura 4.3: **Diagrama de flujo de la estructura de control if... else...**

Python 3

```
humedad=50
print("Inicio del programa")
if humedad<40:
    print("Encender el sistema de  riego")
else:
    print("El jardín está en excelentes condiciones")
print("Fin del programa")
```

Salida

```
Inicio del programa
El jardín está en excelentes condiciones
Fin del programa
```

5 | Bucles o lazos

Una vez aprendido los conceptos básicos de programación, los tipos y operadores de datos, presentamos una *estructura de control* **para ejecutar acciones repetitivas según una condición dada.**

Pero, ¿Qué es una *estructura de control*?. **Es un bloque o sección de código que nos permite agrupar instrucciones de manera controlada. Esto es fundamental al desarrollar sistemas puesto que hay aplicaciones donde tenemos que realizar acciones repetitivas y no siempre vamos a saber cuántas veces esto va a ocurrir. A continuación lo explicamos.**

5.1 | Bucle While

Supongamos que estamos en una habitación y deseamos mostrar la temperatura del ambiente mediante una aplicación escrita con Python y la ayuda de sensores. En principio queremos mostrar la temperatura dos **veces. A continuación se muestra su representación en diagrama de flujo y código.**

Python 3

Para el siguiente código, vamos a asumir que el comando *sensor.leer()* **nos devuelve la temperatura del ambiente.**

```
temperatura=sensor.leer()
print("La temperatura del ambiente es: "+temperatura+"      ºC")
temperatura=sensor.leer()
print("La temperatura del ambiente es: "+temperatura+"      ºC")
```

Salida

```
La temperatura del ambiente es 24.1 ºC
La temperatura del ambiente es 24.5 ºC
```

Si ahora queremos mostrar cinco **veces el valor de temperatura deberíamos repetir el fragmento de código donde leemos el valor del sensor e imprimimos por pantalla las veces deseadas. Pero, ¿Y si queremos mostrarlo**

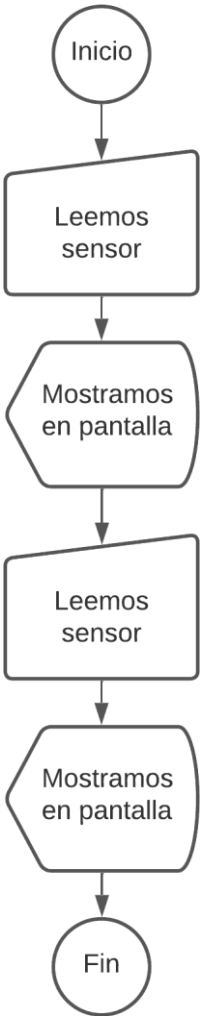

Figura 5.1: **Ejemplo de algoritmo para mostrar secuencialmente dos lecturas de sensor**

cien**veces? esto de repetir el código se volvería extenso. Incluso ¿Qué pasa si queremos repetirlo por siempre o bajo una cierta condición? Aquí entra en juego el** Bucle While**.**

5.2 | Creación de Bucle While

Podemos identificar dos partes principales en el Bucle While (ver Figura5.2):

- Expresión de control: **es la condición que se debe cumplir para que se ejecute el código.**
- Bloque de código: **son las acciones a realizar dentro del Bucle.**

¡Tencuidado!**Un mal uso del Bucle While puede dar lugar a ciclos infinitos y la ejecución del programa puede no ser la deseada. Tratamos de siempre tener una condición de escape que permita terminar el Bucle While. Esta condición suele estar relacionada con el cambio de una variable.**

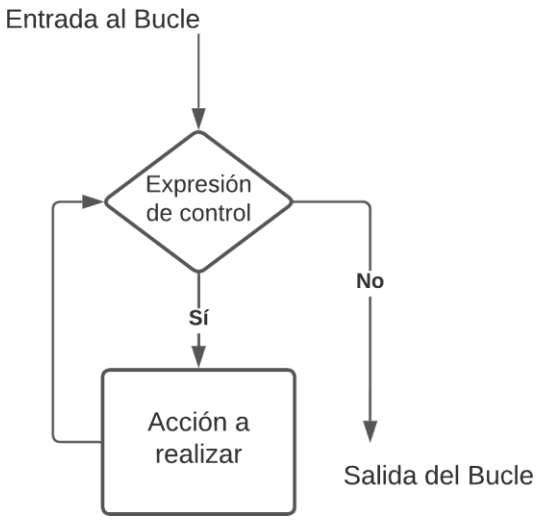

Figura 5.2: **Equivalente en diagrama de flujo del Bucle While**

Por ejemplo, siguiendo el problema planteado en la subsección anterior, ahora queremos mostrar la temperatura solo cuando sea menor a 25°C. Una vez que la temperatura sea mayor o igual a 25°C el programa terminará. A continuación te presentamos el código.

Python 3

Para el siguiente código, vamos a asumir que el comando *sensor.leer()* **nos devuelve la temperatura del ambiente.**

```
temperatura=sensor.leer()
while temperatura<25:
    print("La temperatura del ambiente es: "+temperatura+"      ºC")
    temperatura=sensor.leer()
print("La temperatura ha alcanzado los 25ºC")
```

Salida

```
# La temperatura oscila entre 24 y 24.9 ºC

La temperatura del ambiente es 24.1 ºC
La temperatura del ambiente es 24.5 ºC
La temperatura del ambiente es 24.7 ºC

# La temperatura pasa los 24.9 ºC

La temperatura ha alcanzado los 25ºC
```

Existen casos donde si deseamos que el programa se quede en un ciclo infinito, por ejemplo, si quisiéramos mostrar siempre la temperatura independientemente del cambio. Para ello podemos hacer uso del dato booleano *True* **como se muestra a continuación.**

Python 3

Para el siguiente código, vamos a asumir que el comando *sensor.leer()* **nos devuelve la temperatura del ambiente.**

```
temperatura=sensor.leer()
whileTrue :
    print("La temperatura del ambiente es: "+temperatura+"    ºC")
    temperatura=sensor.leer()
print("Esta línea nunca se va a imprimir")
```

Salida

```
# La temperatura toma cualquier valor

La temperatura del ambiente es 24.1 ºC
La temperatura del ambiente es 24.5 ºC
La temperatura del ambiente es 24.7 ºC
.
.

# Continuará hasta que se interrumpa la ejecución del programa
```