CONTENIDO:

1. Programación estructurada.
2. Condiciones.
3. Estructura de control condicional "if".
   (a) Condicional simple "if"
   (b) Condicional doble "if - else"
   (c) Condicional anidado "if - elif - else"
4. Estructuras de control iterativas.
   (a) Ciclo de repetición "while"
   (b) Ciclo de repetición "for"
5. Contadores.
6. Banderas.

1. PROGRAMACIÓN ESTRUCTURADA

Hasta este momento, todas las instrucciones se ejecutaban secuencialmente en el orden en que estaban escritas en el código fuente. Esta ejecución, como ya se ha comentado, se denomina programación secuencial.

Uno de los más importantes avances fue el reconocimiento a finales de los sesenta de que cualquier algoritmo, no importaba su complejidad, podía ser construido utilizando combinaciones de tres estructuras de control de flujo estandarizadas (secuencial, selección, repetitiva o iterativa).

Los ciclos condicionales son: si (if) y según-sea (switch); las sentencias de repetición o iterativas son: desde (for), mientras (while), hacer-mientras (do-while) o repetir-hasta que (repeat-until),

El término flujo de control se refiere al orden en que se ejecutan las instrucciones de un programa. A menos que se especifique expresamente, el flujo normal de control de todos los programas es el secuencial. Este término significa que las sentencias se ejecutan en secuencia, una después de otra, en el orden en que se sitúan dentro del programa. Las estructuras condicionales y de repetición permiten que el flujo secuencial del programa sea modificado en un modo preciso y definido con anterioridad. Las estructuras condicionales se utilizan para seleccionar cuáles instrucciones se han de ejecutar a continuación y las estructuras de repetición (repetitivas o iterativas) se utilizan para repetir un conjunto de instrucciones.

2. CONDICIONES

Se presentan situaciones donde la computadora debe tomar decisiones antes de ejecutar una instrucción u otra, basándose en el resultado de una condición. En otros casos, el resultado de la condición indicará si se repite una instrucción (o bloque de instrucciones), si no se repite; o si finaliza la repetición.

Las decisiones que tomará el computador estarán basadas en la evaluación de una condición.

CONDICIÓN

Es todo elemento (u objeto) que al ser evaluado nos arroja un resultado de tipo de dato booleano (bool).

La ejecución de instrucciones dependerá del valor que resulte de la evaluación de la condición (True o False).

Una condición puede ser:

- Una expresión lógica.
- Una expresión relacional.
- Una expresión mixta.
- Un valor bool almacenado en una variable.
- Una llamada a función que retorna un valor bool.
- En Python cero es False y distinto de cero es True.

3. ESTRUCTURA DE CONTROL CONDICIONAL "if"

3.1. "if" Simple o condicional simple

El "if" ejecuta solo una instrucción o bloque de instrucciones de acuerdo a las siguientes reglas de funcionamiento:

i. Lo primero que hace el "if" es evaluar la condición.
ii. Si el resultado de evaluar la condición es «True», se ejecuta la instrucción o bloque de instrucciones.
iii. Si la condición es «False» finaliza el "if".

Sintaxis para el "if" simple:

```
if «condición»:
    «instrucción»
```

Una instrucción puede ser:

- Una expresión aritmética.
- Una asignación.
- Una entrada "input()".
- Una salida "print()".
- Un ciclo de control.
- La llamada a una función.

"if" doble o condicional doble

Sintaxis para el "if" doble: solo con una instrucción

```
if «condición»:
    «instrucción_A»
else:
    «instrucción_B»
```

Sintaxis para el "if" doble: con más de una instrucción

```
if «condición»:
    «instrucción_A»
    «instrucción_B»
else:
    «instrucción_C»
    «instrucción_D»
```

El "if" doble ejecuta solo una instrucción o bloque de instrucciones de acuerdo a las siguientes reglas de funcionamiento:

i. Lo primero que hace el "if" es evaluar la condición. Luego guiándonos por la anterior muestra de la sintaxis con una instrucción, tenemos que:
ii. Si el resultado de evaluar la condición es «True», se ejecuta la «instrucción_A» o el bloque de instrucciones que están en el "if" y finaliza.
iii. Si la condición es «False», se ejecuta la «instrucción_B» o el bloque de instrucciones que están en el "else" y finaliza.

Nota: Solo ejecuta una instrucción o bloque de instrucciones, no es posible que ejecute juntas la «instrucción_A» y la «instrucción_B».

El condicional "if" no tiene la capacidad de repetirse por sí solo

3.2. "if" anidado o condicional múltiple

Sintaxis para el "if" anidado:
Python permite expresar fragmentos de código de forma compacta, cuando el contenido de un "else" requiere otro "if" lo podemos escribir con "elif", el siguiente ejemplo solo puede ejecutar una instrucción, o finalizar:

```
if «condición_1»:
    «instrucción_A»
elif «condición_2»:
    «instrucción_B»
elif «condición_3»:
    «instrucción_C»
```

El "if" anidado ejecuta solo una instrucción o bloque de instrucciones de acuerdo a las siguientes reglas de funcionamiento:

1. Lo primero que hace el "if" es evaluar la condición. Luego guiándonos por la anterior muestra de la sintaxis tenemos que:
2. Si el resultado de evaluar la «condición_1» es «True», se ejecuta la «instrucción_A» o el bloque de instrucciones que están en el "if" y finaliza.
3. Si la condición es «False», se va al primer "elif" y evalua la «condición_2»:
   3.1. Si el resultado de la «condición_2» es "True" ejecuta la «instrucción_B» o el bloque de instrucciones que están en el "elif" y finaliza.
   3.2. Si el resultado de la «condición_2» es "False", se va al siguiente "elif" y evalúa la «condición_3»:
      3.2.1. Si el resultado de la «condición_3» es "True" ejecuta la «instrucción_C» o el bloque de instrucciones que están en el "elif" y finaliza.
      3.2.2. Si el resultado de la «condición_3» es "False" finaliza el ciclo "if" anidado.

Nota: Solo ejecuta una instrucción o bloque de instrucciones, no es posible que ejecute juntas la «instrucción_A», «instrucción_B» y la «instrucción_C».

A continuación, se muestra un nuevo caso donde combinamos "if", "elif" y "else". Este ejemplo nos muestra un "if" anidado donde sin importar lo que pase siempre se ejecutara una instrucción, ya que, si la «condición_1», «condición_2» y la «condición_3» son "False" siempre se ejecutara la «instrucción_C» que esta en el "else".

```
if «condición_1»:
    «instrucción_A»
elif «condición_2»:
    «instrucción_B»
elif «condición_3»:
    «instrucción_C»
else:
    «instrucción_C»
```

Recuerde que, la indentación de cuatro espacios es lo que permite que las instrucciones de "if", "elif" y "else", pertenezcan al condicional.

4. ESTRUCTURA DE CONTROL ITERATIVAS O CICLOS DE REPETICIÓN

Python permite repetir una sección de código de programa de dos formas distintas: mediante el ciclo "while" y el ciclo "for".

Un ciclo de repetición "se repite", cada ejecución de instrucciones es una repetición. Si un ciclo ejecuta su bloque de instrucciones solo una vez, se dice, que tuvo una repetición, en cualquier caso, tendrá tantas repeticiones como el número de veces que se ejecutaron sus instrucciones.

Recordemos que una condición puede ser:

- Una expresión lógica.
- Una expresión relacional.
- Una expresión mixta.
- Un valor bool almacenado en una variable.
- Una llamada a función que retorna un valor bool.
- En Python cero es False y distinto de cero es True.

El ciclo de repetición "while" utiliza una condición para controlar sus repeticiones, como ya hemos hablado de condiciones en la sección 3 de esta guía, este será el primer ciclo iterativo que estudiaremos.

4.1. CICLO DE REPETICIÓN "while"

En inglés, «while» significa «mientras». La sintaxis para "while" se muestra a continuación.

Cuando tiene solo una instrucción:

```
while «condición»:
    «instrucción»
```

El ciclo de repetición «while», evalúa la condición, y se repetirá mientras la condición evaluada de como resultado el valor «True»

Pasos de funcionamiento:

1. Lo primero que hace es evaluar la condición.
   i. Si la condición es «True» ejecuta la instrucción o bloque de instrucciones y vuelve al ciclo para evaluar de nuevo la condición.

   ii. Si la condición es «False» el ciclo finaliza.

   Nota: La condición siempre debe cambiar dentro del ciclo lo que permite que se repita y finalice.

Cuando tiene más de una instrucción:

```
while «condición»:
    «instrucción_A»
    «instrucción_B»
    «instrucción_C»
```

El resultado de evaluar la expresión es lo que determina el comportamiento del "while", por lo que puede ocurrir algunas de las siguientes situaciones:

- Si la primera vez la condición resulta «False», el ciclo finalizara y no ejecuta ninguna instrucción.

- Si la primera vez la condición resulta «True», el ciclo inicia, ejecuta el bloque de instrucciones y vuelve al ciclo para evaluar la condición.

- Durante la ejecución, si al volver al ciclo la condición es «True» el ciclo se repite, si es «False» el ciclo finaliza.

En el "while", la indentación de cuatro espacios es lo que indica al intérprete de Python que las instrucciones pertenezcan al ciclo de repetición.

4.2. CICLO DE REPETICIÓN «for»

El ciclo de repetición «for» fue dejado de último intencionalmente, debido a que este ciclo no utiliza una condición para controlar su funcionamiento.

Habrá situaciones donde necesitamos repetir un bloque de instrucciones, y el número de veces que se repiten es conocido o se puede conocer antes de iniciar las repeticiones, esta es la situación ideal para implementar un ciclo de repetición «for».

Sintaxis para el ciclo «for» con una instrucción:

```
for «vc» in range(«vi»,«vf»):
    «instrucción»
```

En el ejemplo de sintaxis utilizamos una nomenclatura genérica que a continuación describimos:

- «vc» es la variable de control, debe ser de tipo entero (int) y ella llevará el control del número de iteraciones o vueltas que ejecuta el ciclo.
- «vi» el valor inicial del ciclo, debe ser un número entero y aunque normalmente inicia con cero o uno, de ser necesario puede iniciar en un valor distinto.
- «vf» el valor final: es un valor entero que indica el número total de iteraciones del ciclo. "for" se ejecutará hasta el último valor entero menor que el valor final, ejemplo:

```
for k in range(1,10):
```

La variable k iniciará en uno (1) y finalizará en nueve (9), el número de la última iteración será el valor anterior al valor final.

El ciclo «for», se repetirá el número de veces que le indique el valor final menos uno.

Pasos de funcionamiento:

1. Se asigna el valor inicial a la variable de control, ejecuta las instrucciones y vuelve al ciclo.
   i. Si la variable de control **no es igual** al valor final menos uno, se le **suma uno (1)** (a la «vc») y ejecuta nuevamente las instrucciones, luego vuelve al ciclo.

   ii. Si la variable de control **es igual** al valor final menos uno el ciclo finaliza.

Nota: solo el «**for**» debe cambiar el valor de la variable de control, una vez iniciado el ciclo no debe leer valores, ni asignarlos sobre la variable de control.

```
for «vc» in range(«vi»,«vf»):
    «instrucción_A»
    «instrucción_B»
    «instrucción_C»
```

Hasta ahora hemos visto el ciclo "for" en combinación con la función "range()" a continuación se muestran ejemplos utilizando valores:

1. Solo indicándole a la función "range()" el número total de vueltas:

```
>>> for k in range(10):
...     print(k, end=' ')
...
0 1 2 3 4 5 6 7 8 9
```

Utilizando "range(10)" la variable de control "k" tomará valores desde cero (0) hasta (9), el ciclo se repetirá diez veces.

2. Indicándole a "range()" el valor inicial (1) y el valor final (10):

```
>>> for k in range(1, 10):
...     print(k, end=' ')
...
1 2 3 4 5 6 7 8 9
```

Utilizando "range(1,10)" la variable de control "k" tomará valores desde cero (1) hasta nueve (9), el ciclo se repetirá nueve veces.

3. Usando "range(2, 10, 2)" el valor inicial (2), el valor final (10) y de cuanto en cuanto se aplicará el incremento (2):

```
>>> for k in range(2, 10, 2):
...     print(k, end=' ')
...
2 4 6 8
```

Utilizando "range(2, 10, 2)" la variable de control "k" tomará valores desde dos (2) hasta ocho (8), el ciclo se repetirá cuatro veces porque se contará de dos en dos, "range" siempre llega al valor anterior al valor final en este caso es ocho (8).

4. Usando "range(5, 0, -1)" el valor inicial (5), el valor final (0) y se le indica que reste de un en uno (-1):

```
>>> for k in range(5, 0, -1):
...     print(k, end=' ')
...
5 4 3 2 1
```

Utilizando "range(5, 0, -1)" la variable de control "k" tomará valores desde cinco (5) hasta uno (1), el ciclo se repetirá cinco veces haciendo una cuenta hacia atrás porque se contará menos uno (-1) en cada iteración. Recordemos que "range" siempre llega al valor anterior al valor final en este caso es uno (1).

5. CONTADORES

Un contador es una variable numérica (int o float) que permite acumular (contar) valores todo en una misma variable.

La sintaxis para un contador:

```
cont = cont + 5

prod = prod * 2
```

Los contadores utilizan dos estrategias para acumular los valores:

1. Utilizan el elemento neutro de la operación (operador) que se esté utilizando, por ejemplo (a) si es una suma el contador se iniciará en cero, (b) cuando es una multiplicación se iniciará en uno (1).

   La inicialización con el elemento neutro le permitirá acumular todos los valores que se necesiten dentro de un ciclo de repetición. Si la solución del problema lo requiere se pueden inicializar en otros valores.

2. Junto al valor que inicializa la variable usada como contador, la segunda estrategia aplica la prioridad de operadores, ejemplo:

   ```
   Suma = 0
   ...

   suma = suma + 3
   ```

   Un contador es una variable que será utilizada en una expresión (suma = suma + 3), en el ejemplo anterior, de acuerdo con la prioridad de operadores y las reglas para evaluar expresiones tenemos que:

   i. Primero se evalúa la suma (suma + 3), lo que permite usar el valor actual que suma esta almacenando.
   ii. Luego el resultado de la suma se asigna a la misma variable suma, debido a que la asignación (=) tiene la menor prioridad en esta expresión. Esto permite acumular los valores a medida que avancen las iteraciones donde esta incluido el contador.

   El ejemplo (suma = suma + 3), acumulará valores de tres en tres.

6. BANDERAS

Una bandera es un valor booleano utilizado como condición para controlar el funcionamiento de un ciclo de control “if” o “while”.

Recordemos que una condición puede ser:

- Un valor bool almacenado en una variable.
- Una llamada a función que retorna un valor bool.
- En Python cero es False y distinto de cero es True.

Sintaxis para el uso de banderas con un “while” :

```
bandera = True
...
if bandera:
    «instrucción»
```

Sintaxis para el uso de banderas con un "while" :

```
bandera = True
...
while bandera:
    «instrucción»
```

Cuando ocurra alguna situación esperada, la bandera podrá cambiar de valor y este cambio indicará so que el ciclo de control donde es usada finalice o se detenga.

Una equivocación común al usar banderas, Ud. debe estar atento para que no ocurra:

```
salir = False

if salir == True:
    «instrucción_A»

#------------------------------

band = True

while band != False:
    «instrucción_A»
```

En el ejemplo, resaltado en color rojo, se muestra la forma equivocada de utilizar una bandera, la bandera (variable o llamada a función), por si sola es una expresión evaluable, al incluir algún operador relacional deja de ser una bandera para convertirse en una expresión relacional.

Las convenciones para escribir el código Python (PEP28) establecen que no se debe usar como muestra en este ejemplo (ver la guía de estilo Python).