AWAKELAB

Programación en Python

Objetivo de la jornada

1. Codifica un programa en Python utilizando instrucciones de bloque para manejar el flujo en base a condiciones lógicas.
2. Codifica un programa en Python utilizando estilos y convenciones de programación tales como indentación y estructura de código.

3.3.- Contenido 3: Codificar un programa en Python utilizando sentencias condicionales para el control del flujo que resuelve un problema

3.3.1.- Entendiendo las instrucciones condicionales.

En esta sección veremos un poco más de cerca las estructuras condicionales que revisamos resumidamente en el capítulo anterior.

¿Por qué usamos condicionales?

DECISIONES Y DIAGRAMA DE FLUJO

Hay muchas situaciones en las que nuestras acciones son condicionales, es decir, decidimos qué hacer en base a la respuesta a alguna pregunta o en alguna condición que observamos. Además frecuentemente tenemos varias alternativas para una decisión. Por ejemplo:

```
Si está lloviendo
   Tomar un paraguas
   Ponerse botas
Sino
   Ponerse lentes de sol
   Ponerse zapatillas
Ir a trabajar
```

Hay cierta estructura en la descripción anterior: un conjunto de cosas que hacemos si está lloviendo y un conjunto alternativo de cosas que hacemos si no está lloviendo. Es conveniente utilizar la sangría para

ayudarnos a agrupar visualmente estas acciones relacionadas. El hecho de que "ir a trabajar" no esté indentado junto con "ponerse zapatillas" sugiere que tenemos que ir a trabajar, esté lloviendo o no.

La estructura de decisiones como estas se puede visualizar muy bien con diagramas de flujo. En un diagrama de flujo de un programa los puntos de decisión se representan con un bloque en forma de diamante, por lo que nuestro ejemplo se vería de la siguiente forma:

```
Si está lloviendo
    Tomar un paraguas
    Ponerse botas
Sino
    Ponerse lentes de sol
    Ponerse zapatillas
Ir a trabajar
```

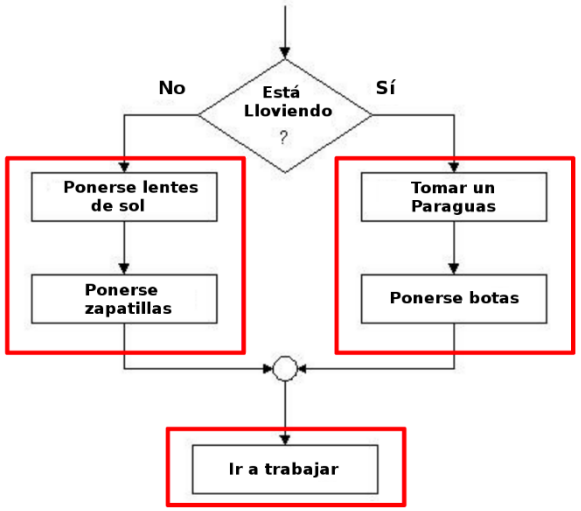

Diagrama de flujo de estructura condicional

De todos modos, los diagramas de flujo dejan una cosa muy clara. Hay dos ramas o rutas a través del diagrama, y hará las cosas en la rama "Sí", O hará las cosas en la rama "No", pero no hay ruta posible para realizar ambos caminos. Los cuadros en color rojo corresponden a acción(es) concretas a ejecutar.

También observamos que el pequeño círculo donde las ramas "Sí" y "No" se vuelven a unir. Esa es una característica importante que refleja la forma en que se utilizan las declaraciones condicionales en programación. Tomarás una de las dos rutas alternativas que tienen diferentes acciones en ellas, pero ambas rutas DEBEN terminar finalmente en el mismo punto nuevamente.

EXPRESIONES BOOLEANAS

Debemos poder escribir declaraciones en Python que representen acciones condicionales. ¿Qué es una "condición"? Por "condición", simplemente nos referimos a una expresión cuyo valor es verdadero o falso. Por ejemplo,

```
>>> x = 42
>>> print(x > 100)
False
```

Los booleanos son un tipo de datos que tiene solo dos valores posibles, representados por los literales de Python **True** y **False**. Podemos escribir expresiones booleanas simples usando los operadores relacionales para mayor que y menor que como el ejemplo anterior. Hay seis operadores relacionales en total que en Python corresponden a los que se muestran a continuación con su equivalente matemático:

Significado	Símbolo Matemático	Símbolo en Python
Menor que	<	<
Mayor que	>	>
Menor o igual que	≤	<=
Mayor o igual que	≥	>=
Distinto a	≠	!=
Igual a	=	==

Recuerde, el símbolo = no es un signo igual en Python, es el operador de asignación. Entonces, para probar si dos expresiones son iguales, necesitamos un símbolo diferente. Como muchos otros lenguajes de programación, Python usa un doble signo igual (==) para este propósito:

```
>>> x = 42          # "A x se le asigna el valor 42"
>>> print(x == 43)    # "Imprime resultado de x es igual a 43"
False
>>> print(x == 42)    # "Imprime resultado de x es igual a 42"
True
>>> print(x != 100)   # "Imprime resultado de x es distinto de 100"
True
```

SENTENCIAS CONDICIONALES

Comencemos con un ejemplo. A continuación, se muestra un breve Script que lee el puntaje de un estudiante en un examen y luego imprime una breve respuesta.

```
puntaje = int(input("Ingrese su puntaje:"))
if puntaje >= 65:
   print ("Al parecer estás aprobando")
else:
   print (puntaje, " es menor a 65.")
   print ("Debes estudiar un poco más.")
print ("Adiós!")
```

Todas las declaraciones entre **if** y **else** están indentadas en la misma cantidad, y las declaraciones después de **else** y antes de la última declaración impresa también tienen indentación. Esto es importante, porque es a partir de la indentación que el intérprete de Python determina qué declaraciones hay en cada bloque. El **print** al final no es parte de ninguno de los bloques, por lo que se ejecuta independientemente de la puntuación. En general, un bloque de sentencias es solo un grupo de ellas que tienen la misma indentación.

EJEMPLO PRÁCTICO
Supongamos que hay una empresa que vende un producto en línea. El producto cuesta $200 cada uno más $25 para el envío. Sin embargo, el envío es gratuito para pedidos de 30 o más productos Escriba una secuencia de comandos que lea la cantidad de unidades de productos requeridos por el usuario y luego imprima el costo total del pedido.

Primero, revisemos unos ejemplos para asegurarnos de que sabemos lo que estamos haciendo. ¿Cuánto costaría un pedido de 10 unidades del producto?

10 * 200 = 2000 es el total por 10 unidades
¿El pedido es mayor o igual a 30 unidades?
No, por lo tanto pagados 10 * 25= 250 por envío
Total 2000 + 250 = 2250

¿Y si pedimos 100 unidades del producto?

100 * 200 = 20000 es el total por 100 unidades
¿El pedido es mayor o igual a 30 unidades?
Sí, por lo tanto el envío es sin costo.
Total 20000

¿Y si pedimos exactamente 30?

30 * 200 = 6000 es el total por 30 unidades
¿El pedido es mayor o igual a 30 unidades?
Sí, por lo tanto el envío es sin costo.
Total 10000

La cantidad de exactamente 30 unidades de producto se denomina **Condición de Borde**, ya que representa el límite entre dos acciones alternativas. Éstas son, envío gratuito y envío no gratuito. (Es una buena idea verificar siempre las condiciones de los límites, porque ese es un lugar común para cometer errores).

Ahora que tenemos algunos ejemplos resueltos, debería ser fácil escribir el código. Solo mire los ejemplos y haga lo mismo. Supongamos que tenemos una variable **num** que contiene la cantidad de unidades del producto que se está ordenando.

Pedido de 10 unidades del producto	**Pedido de num unidades del producto**
10 * 200 = 2000 es el total por 10 unidades ¿El pedido es mayor o igual a 30 unidades? No, por lo tanto pagados 10 * 25= 250 por envío Total 2000 + 250 = 2250	total_productos = num * 200 ¿El pedido es mayor o igual a 30 unidades? costo_envio = num * 25 Total: total_productos + costo_envio

Un ejemplo de Script completo se vería como el siguiente. El código es exactamente como el anterior, sólo que agregamos una declaración **if** para distinguir los dos casos, envío no gratuito y envío gratuito.

```
# Este script calcula el costo total
# para un numero dado de unidades de
# un producto ingresado por el usuario

num = int(input("Cuantos productos: "))
Costo_productos = num * 200
if num < 30:
```

```
    costo_envio = num * 25
else:
    costo_envio = 0
total = costo_productos + costo_envio
print(total)
```

Finalmente, debemos leer el código y verificar que obtenemos la respuesta correcta para la condición de borde, exactamente 30 unidades del producto. Dado que la condición "30 < 30" es falsa, ejecutaremos el bloque **else** y tendremos un cargo de envío de cero.

Vale la pena pensar en el hecho de que hay más de una forma de resolver este problema. Por ejemplo, podríamos comenzar calculando el costo de envío y luego cambiarlo a cero si encontramos que hay 30 o más unidades de producto requeridas. En ese caso, terminaría con un Script como este:

```
num = int(input("Cuantos productos: "))
costo_productos = num * 2.00
costo_envio = num * .50
if num >= 30:
    costo_envio = 0
total = costo_productos + costo_envio
print(total)
```

Si nos preguntamos cuál es la forma correcta de resolver este problema. Ambas lo son. Por lo general, hay más de una forma de pensar sobre un problema determinado. Podemos abordar una solución de manera diferente a otra persona, pero siempre que traduzcamos fielmente los pasos de nuestra estrategia en declaraciones de Python, terminaremos con un programa que funciona.

CONDICIONALES ANIDADOS

A veces, tomar una decisión conduce a decisiones adicionales. En el siguiente ejemplo, solo nos enfrentamos a la decisión de ir al cajero automático si primero descubrimos que necesitamos combustible:

```
if necesito combustible
   if no tengo dinero
      Ir al cajero
   Cargar combustible
Ir a trabajar
```

Visualizada como un diagrama de flujo, esta secuencia de acciones se vería como se muestra a continuación. Si la respuesta a "¿Necesito combustible?" es Sí, entonces tenemos que hacer la pregunta: "¿Tengo dinero?" Si la respuesta es Sí, podemos simplemente comprar gasolina, pero si la respuesta es No, primero tenemos que ir al cajero automático y luego comprar gasolina.

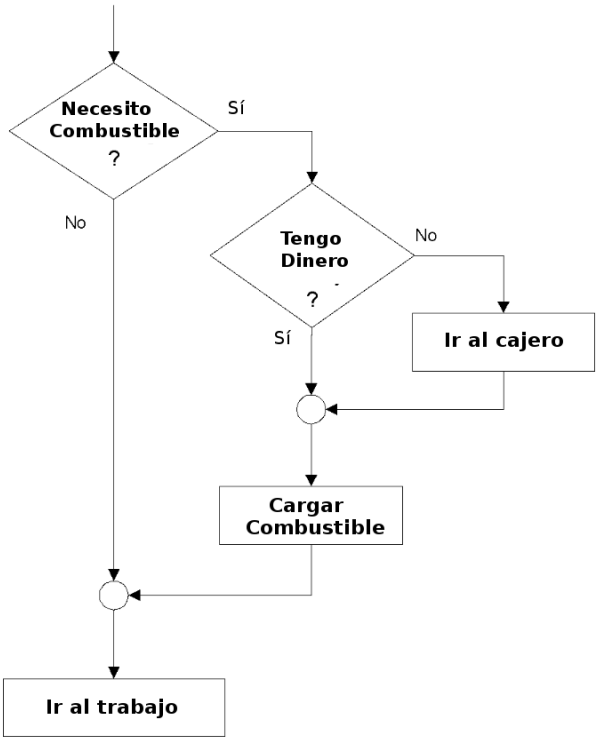

Diagrama de flujo de estructura condicional anidada

¿Cómo expresamos algo como esto en Python? Con lo que hemos revisado hasta ahora podríamos implementar algo como:

```
combustible = input("Necesito combustible?(S/N): ")
if combustible == "S":
    dinero = input("Tengo dinero?(S/N): ")
    if dinero == "N":
        print("Sacando dinero del cajero...")
    print("Cargando combustible...")
print("Yendo al trabajo")
```

En este caso hemos podido resolver nuestro problema solamente con sentencias **if,** sin necesidad de **else**.

Consideremos otro caso de ejemplo en el que necesitemos monitorear una temperatura. Queremos asegurarnos que ésta se encuentre entre 20 y 35 grados Celsius. En caso que sea menor a 20 deberemos encender un calefactor, y si es mayor a 35 deberemos encender un enfriador de aire.

En este caso podemos utilizar la siguiente lógica:

```
temp = int(input("Qué temperatura hay en este momento?"))
if temp > 35:
    print("Encendiendo enfriador de aire")
else:
    if temp < 20:
        print("Encendiendo calefactor")
    else:
        print("La temperatura está en el rango adecuado")
```

Observe que ahora hay dos "niveles de indentación", ya que toda la declaración "if temp <20 ..." debe estar indentada en sí misma para formar parte del "bloque else" exterior.

UN EJEMPLO DE MAYOR COMPLEJIDAD

Supongamos que queremos convertir puntajes de exámenes de estudiantes en 4 categorías:

Rango	Categoría
90 <= puntaje	Nivel 1
80 <= puntaje < 90	Nivel 2
70 <= puntaje < 80	Nivel 3
puntaje < 70	Nivel 4

Podemos ayudarnos con un diagrama de flujo previo a la implementación del código con las sentencias condicionales que correspondan:

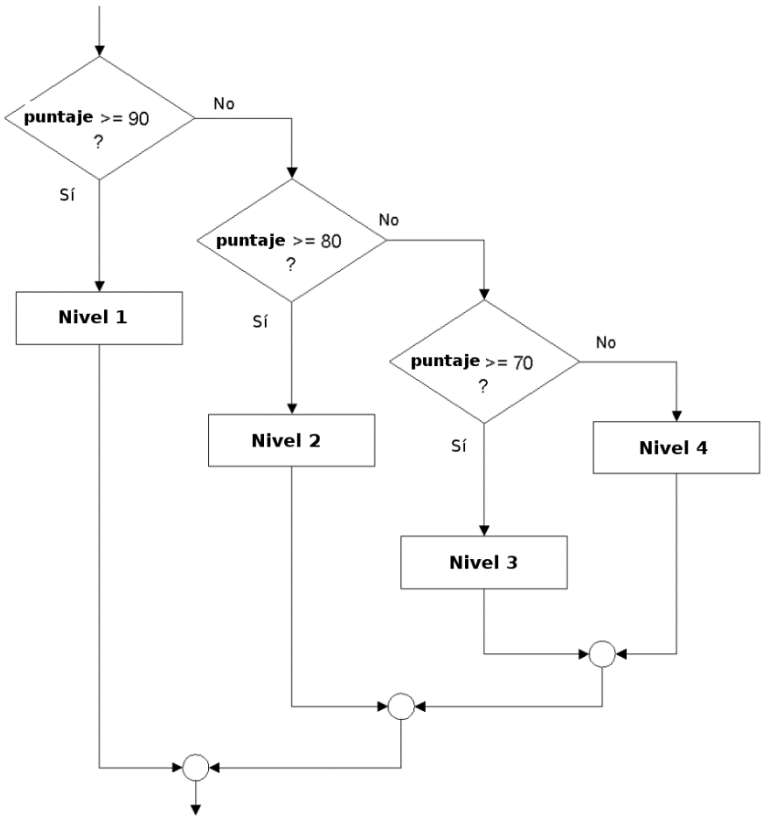

Diagrama de flujo de estructura condicional anidada multinivel

Una cosa que ayuda a pensar en un problema como este es intentar primero escribir el algoritmo de decisión en “pseudocódigo”. El pseudocódigo es una forma de describir un algoritmo de manera informal que se encuentra a medio camino entre el castellano y el código real, utilizando indentaciones para agrupar acciones. Cuando escribimos pseudocódigo, estamos pensando en escribir código real, por lo que

intentamos pensar en términos de bloques "if" y "else". Podría verse algo como esto:

```
if puntaje es mayor o igual a 90
   El nivel es 1
else
   if puntaje es mayor o igual a 80
      El nivel es 2
   else
      if puntaje es mayor o igual a 70
         El nivel es 3
      else
         El nivel es 4
```

Una vez que tenemos la estructura organizada, no es difícil traducir el esquema del pseudocódigo en código real. En particular, se eligió el uso de indentación en Python precisamente porque refleja los hábitos de los programadores que expresan algoritmos en pseudocódigo.

De acuerdo a lo anterior, el código en Python sería el siguiente:

```
puntaje = int(input("Cuál es tu puntaje?"))
if puntaje >= 90:
   nivel = 1
else:
   if puntaje >= 80:
      nivel = 2
   else:
      if puntaje >= 70
         nivel = 3
      else:
         nivel = 4
print(nivel)
```

Un error común en primeros intentos de resolver problemas como el planteado es considerar la siguiente solución:

```
puntaje = int(input("Cuál es tu puntaje?"))
if puntaje >= 90:
   nivel = 1
if puntaje >= 80:
   nivel = 2
if puntaje >= 70
   nivel = 3
else:
```

```
nivel = 4
print(nivel)
```

Sin embargo, en este caso, si probamos el código para un puntaje de 95 tendremos que éste cumple con todas las condiciones implementadas por las distintas sentencias **if**. De esta forma, la variable **nivel** será sobreescrita luego de cada comprobación de expresión booleana y el nivel resultante será igual a 3, que no es el valor correcto. Por esto, en etapas iniciales como programador es importante utilizar diagramas de flujo y pseudocódigo, lo que nos puede ayudar a evitar problemas como el expuesto.

LA PALABRA CLAVE ELIF

A veces, podemos encontrar que un condicional anidado se escribe de modo que inmediatamente después de un **else** hay otro **if,** como vimos en el ejemplo del control de temperatura algunos párrafos más arriba.

En situaciones como esta, cuando hay un **else** justo después de un **if**, puede utilizar la palabra clave especial **elif** para contraer el **if** y el **else** en la misma línea. Esta es otra forma de declaración condicional. La posible ventaja de usar **elif** es que reduce el número de niveles de indentación y hace que el código sea más fácil de leer. El ejemplo de control de temperatura se vería así:

```
temp = int(input("Qué temperatura hay en este momento?"))
if temp > 35:
   print ("Encendiendo enfriador de aire")
elif temp < 85:
   print ("Encendiendo calefactor")
else:
   print ("La temperatura está en el rango adecuado")
```

Por otro lado, el ejemplo de conversión de puntaje de exámenes a categorías por nivel sería expresado de esta forma:

```
Puntaje = int(input("Cuál es tu puntaje?"))
if puntaje >= 90:
   nivel = 1
elif puntaje >= 80:
   nivel = 2
elif puntaje >= 70:
   nivel = 3
else:
   nivel = 4
print(puntaje)
```

Tenga en cuenta que, si compara el formulario anterior con el diagrama de flujo, no hemos cambiado la lógica. Todo lo que hemos hecho es usar una sintaxis más compacta para expresar las mismas condiciones anidadas.

En esta forma de declaración condicional, se pueden tener tantos bloques **elif** como desee. Normalmente debería haber un **else** al final. Aquí está la sintaxis general, mostrando **n** posibles alternativas. Tenga en cuenta que se ejecutará uno y solo uno de los **n** bloques de instrucciones. Esta es una

buena forma de representar una serie de alternativas mutuamente excluyentes (Como asignar niveles a los puntajes).

```
if condición 1:
   bloque 1 de sentencias
elif condición 2:
   bloque 2 de sentencias
elif condición 3:
   bloque 3 de sentencias
:
:
elif condición n-1:
   bloque n-1 de sentencias
elif condición n:
   bloque n de sentencias
```

LECTURA DEL CÓDIGO Y ENTENDIENDO EL FLUJO

Cuando leemos código observe que no siempre leemos cada línea en el orden en que está escrito. En su lugar, seguimos el flujo del programa, es decir, el orden en el que el intérprete ejecuta realmente las declaraciones. Cuando hay una rama condicional, saltamos a las declaraciones que se ejecutarán, saltando la rama que no está seleccionada.

Es importante poder leer el código y rastrear a mano lo que está ocurriendo, como acabamos de hacer. ¿Por qué esto es útil? Sin embargo, lo que es muy usual que ocurra cuando se programa es que se cometen errores. El intérprete siempre hará exactamente lo que usted le diga. Entonces, si su programa no está haciendo lo que se supone que debe hacer, depende de nosotros. Es una tarea usual, luego de tener un código funcional, reflexionar sobre qué es lo que exactamente instruimos hacer al intérprete y analizar si efectivamente coincide con lo que deseábamos lograr. Muchas horas de desarrollo se invierten en este tipo de análisis y es bueno estar advertido de esto.

Referencias.

[1] Mark Lutz, Python Pocket Reference, Fifth Edition 2014.

[2] Matt Harrison, Ilustrated Guide to Python3, 2017.

[3] Eric Matthes, Python Crash Course, 2016.

[4] Problem Solving with Python – Flowcharts.
https://problemsolvingwithpython.com/08-If-Else-Try-Except/08.06-Flowcharts/

[4] Algoritmos y programación en pseudocodigo - Diego Fernando Duque.
https://repository.usc.edu.co/bitstream/handle/20.500.12421/59/ALGORITMOS.pdf?sequence=1&isAllowed=y