Control de flujo

1 Estructuras de control: Decisiones

1.1 Decisiones en la vida diaria

Hasta ahora, los programas que se han construido se ejecutan de manera secuencial y solo se resuelven expresiones aritméticas para operar sobre los datos. No obstante, en ocasiones es necesario realizar comparaciones o incluso tomar decisiones para optar por un camino en particular.

A modo de ejemplo, considera la plataforma Netflix, la cual, luego de ver 3 episodios de una serie de manera consecutiva y sin tocar el control remoto, entrega el siguiente mensaje:

Lo anterior deja en claro que la toma de decisiones es algo que se encuentra en el día a día y, como la finalidad de programación es facilitar las cosas, se debe considerar en la realización de los programas.

En los lenguajes de programación, cada vez que se desea tomar una decisión respecto a expresiones o situaciones, estas se deben comparar y con base en ello, se escoje la opción más adecuada.

2 Operadores booleanos

2.1 Operadores de comparación

Para poder realizar comparaciones entre 2 expresiones aritméticas, se deben utilizar los operadores de comparación. Python provee un listado de los operadores de comparación más utilizados.

Cabe señalar que el resultado de utilizar estos operadores genera el tipo de dato lógico o `booleano` (`bool`).

Operación	Operador	Ejemplo	Resultado
Mayor	`>`	`2.0 > 5`	`False`
Mayor o igual	`>=`	`8 >= 3.7`	`True`
Menor	`<`	`1.5 < 1.4`	`False`
Menor o igual	`<=`	`3.3 <= 3.3`	`True`
Igual	`==`	`2.0 == -2`	`False`
Distinto	`!=`	`4.0 != 4.01`	`True`

Ahora que se conocen los operadores de comparación, veamos los siguientes ejemplos:

[1]:
```
x = 10
expresion_1 = x != 10
print(expresion_1)
```

```
False
```

[2]:
```
expresion_2 = x > 5
print(expresion_2)
```

```
True
```

[3]:
```
expresion_3 = x < 10
print(expresion_3)
```

```
False
```

[4]:
```
expresion_4 = x >= 5
print(expresion_4)
```

```
True
```

[5]:
```
expresion_5 = x<= 10
print(expresion_5)
```

```
True
```

2.2 Operadores lógicos

Para poder agrupar operaciones lógicas/booleanas, se utilizan los operadores lógicos. Para ello, Python provee tres operadores básicos, los cuales se muestran a continuación:

- Negación (`not`)
- Conjunción o *y lógico* (`and`)
- Disyunción u *o lógico* (`or`)

Cada operador lógico trabaja con un conjunto de reglas resumidas en tablas de verdad.

La tabla de verdad ilustra el resultado de operar con 1 o 2 valores lógicos, según corresponda.

2.2.1 Negación

El operador de negación (`not`) es un operador unario (trabaja solo con una expresión). Puede generar dos resultados, dependiendo la expresión evaluada:

- Genera un valor verdadero (`True`) si y solo si la expresión evaluada es falsa (`False`)
- Genera un valor falso (`False`) si y solo si la expresión evaluada es verdadera (`True`)

expresión_1	not expresion_1
True	**False**
False	**True**

[6]:
```
expresión_1 = 3 < 5
evaluacion_1 = not expresion_1

print(evaluacion_1)
```

```
True
```

[7]:
```
expresión_2 = 45 >= 45.1
evaluacion_2 = not expresion_2

print(evaluacion_2)
```

```
False
```

2.2.2 Conjunción o y lógico

El operador conjunción (`and`) es un operador binario (necesita dos expresiones para trabajar). Puede generar dos resultados, dependiendo de las expresiones evaluadas:

- Genera un valor verdadero (`True`) si y solo si ambas expresiones evaluadas son verdaderas (`True` **and** `True`)
- Genera un valor falso (`False`) si por lo menos alguna de las expresiones evaluadas es falsa (`False`)

expresión_1	expresión_2	expresion_1 and expresion_2
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

[8]:
```
expresion_1 = 4.8 >= 4.2
expresion_2 = 2 == 2

evaluacion_1 = expresion_1 and expresion_2

print(evaluacion_1)
```

```
True
```

[9]:
```
expresion_1 = 4.8 >= 4.2
expresion_3 = 3 > 5

evaluacion_2 = expresion_1 and expresion_3

print(evaluacion_2)
```

```
False
```

[10]:
```
expresion_2 = 2 == 2
expresion_3 = 3 > 5

evaluacion_3 = expresion_3 and expresion_2

print(evaluacion_3)
```

```
False
```

[11]:
```
expresion_3 = 3 > 5
expresion_4 = 91 <= 90

evaluacion_4 = expresion_3 and expresion_4

print(evaluacion_4)
```

```
False
```

2.2.3 Disyunción u o lógico

El operador de disyunción (`or`) es un operador binario (necesita dos expresiones para trabajar). Puede generar dos resultados, dependiendo de las expresiones evaluadas:

- Genera un valor verdadero (`True`) si por lo menos alguna de las expresiones evaluadas es verdadera (`True`)
- Genera un valor falso (`False`) si y solo si ambas expresiones evaluadas son falsas (`False` **or** `False`)

expresión_1	expresión_2	expresion_1 or expresion_2
True	True	**True**
True	False	**True**
False	True	**True**
False	False	**False**

[12]:
```
expresion_1 = 4.8 >= 4.2
expresion_2 = 2 == 2

evaluacion_1 = expresion_1 or expresion_2
```

```
print(evaluacion_1)
```

```
True
```

[13]:
```
expresion_1 = 4.8 >= 4.2
expresion_3 = 3 > 5

evaluacion_2 = expresion_1 or expresion_3

print(evaluacion_2)
```

```
True
```

[14]:
```
expresion_2 = 2 == 2
expresion_3 = 3 > 5

evaluacion_3 = expresion_3 or expresion_2

print(evaluacion_3)
```

```
True
```

[15]:
```
expresion_3 = 3 > 5
expresion_4 = 91 <= 90

evaluacion_4 = expresion_3 or expresion_4

print(evaluacion_4)
```

```
False
```

2.3 Expresiones booleanas

Al igual que los operadores aritméticos, los operadores lógicos tienen su propia **precedencia establecida**:

1. Operadores de comparación
2. Negación (`not`)
3. Conjunciones (`and`)
4. Disyunciones (`or`)

Cabe señalar que si una expresión contiene operaciones aritméticas, estas son resueltas antes de utilizar los operadores de comparación (**precedencia de operadores**)

Realice los siguientes ejercicios y compare con los resultados entregados por Python. Para ello, considere que la variable `x` será evaluada primero con el valor 5 y luego, con el valor 10.

[16]:
```
# Se solicitará el valor de la variable x al usuario
x = input("Favor ingresar el valor de la variable x: ")

# Se realiza el cambio de tipo de dato a entero
x = int(x)

# Se evalua la expresión booleana
expresion_1 = x + 1 > x**2 or x + 5 <= 2 * x

# Se muestra por pantalla el resultado obtenido
print(expresion_1)
```

```
True
```

[17]:
```
# Se solicitará el valor de la variable x al usuario
x = input("Favor ingresar el valor de la variable x: ")

# Se realiza el cambio de tipo de dato a entero
x = int(x)

# Se evalua la expresión booleana
expresion_2 = x < 6 and not x >= 10 or not 2**x < 16

# Se muestra por pantalla el resultado obtenido
print(expresion_2)
```

```
True
```

[18]:
```
# Se solicitará el valor de la variable x al usuario
x = input("Favor ingresar el valor de la variable x: ")

# Se realiza el cambio de tipo de dato a entero
x = int(x)

# Se evalua la expresión booleana
expresion_3 = not(x > 3 and x<=10) and (x**2 <= 16 or 2 * x <= x + 10)

# Se muestra por pantalla el resultado obtenido
print(expresion_3)
```

```
False
```

La utilidad de las expresiones booleanas, en programación, es la de entregar la posibilidad a un programa de **controlar el flujo de ejecución**, es decir, escoger entre un camino y otro.

3 Sentencias condicionales

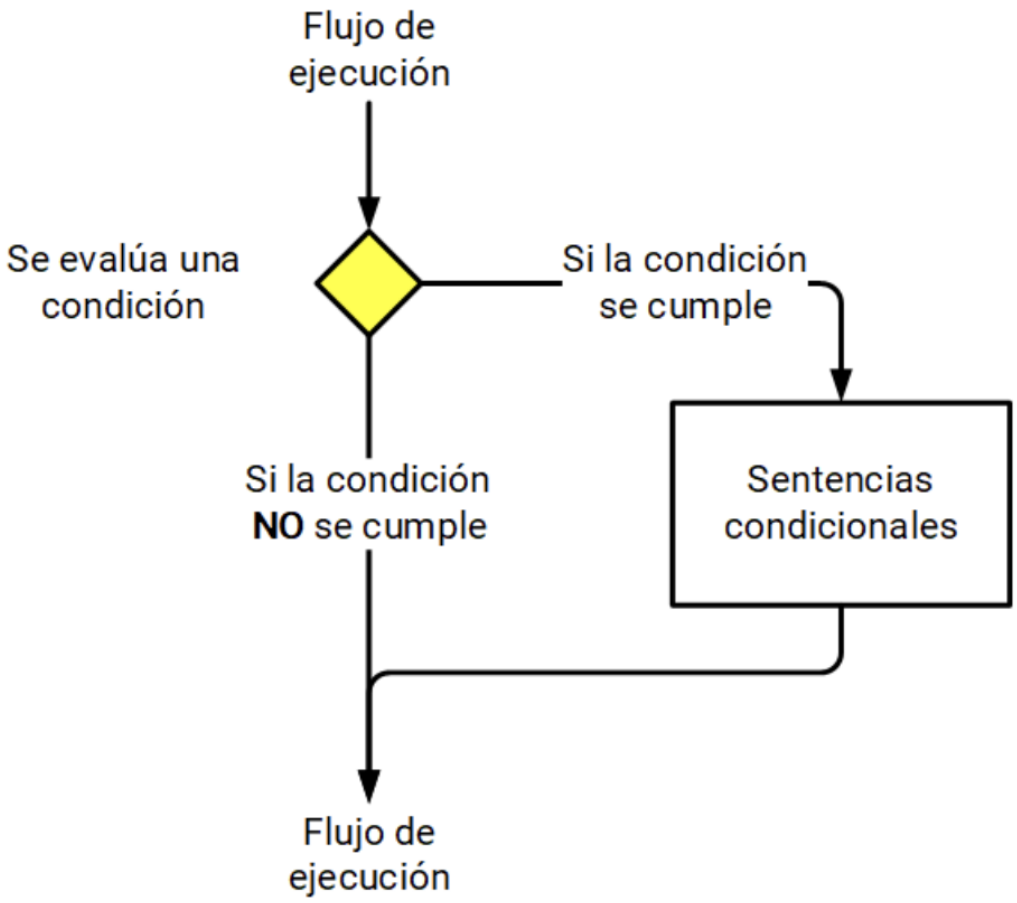
Ahora que ya se han dado a conocer los operadores de comparación y operadores lógicos, es posible indicarle a los programas cómo trabajar en distintas situaciones.

Para poder indicarle a Python sobre qué camino seguir en la ejecución de un código, se utilizarán las sentencias `if` e `if-else`, las cuales serán definidas a continuación:

3.1 Sentencia `if`

Los lenguajes de programación proveen distintas **estructuras de decisión**. La más simple se compone de **una condición**, que no es otra cosa que una expresión booleana y un bloque de sentencias condicionales que solo se ejecutará si la condición se evalúa con un valor verdadero (`True`).

El siguiente diagrama de flujo ilustra la ejecución de una estructura de decisión simple:



La **sintaxis** (forma de escribir) que Python utiliza para una estructura de decisión simple es la siguiente:

```
if <condición>:
    # Se ejecuta si la condición se cumple
    <Bloque de sentencias condicionales>
# Se ejecuta independiente de si al condición se cumple o no
<Bloque de sentencias que sigue>
```

A continuación, se da un ejemplo donde se entregará un mensaje al usuario si el número que se ingresa por teclado es par:

[19]:
```
# ENTRADA
# Se solicita el número al usuario
numero = input("Favor ingresar número para evaluar su paridad: ")

# PROCESAMIENTO
# Se realiza el cambio de tipo de dato a número entero
numero = int(numero)

# Se evalúa la paridad del número ingresado
if numero%2 == 0:
  print("El número ingresado es par")
print("Fin del programa")
```

```
El número ingresado es par
Fin del programa
```

En caso de que el número ingresado no sea par, el programa ignorará las ordenes que estén dentro del bloque condicionado.

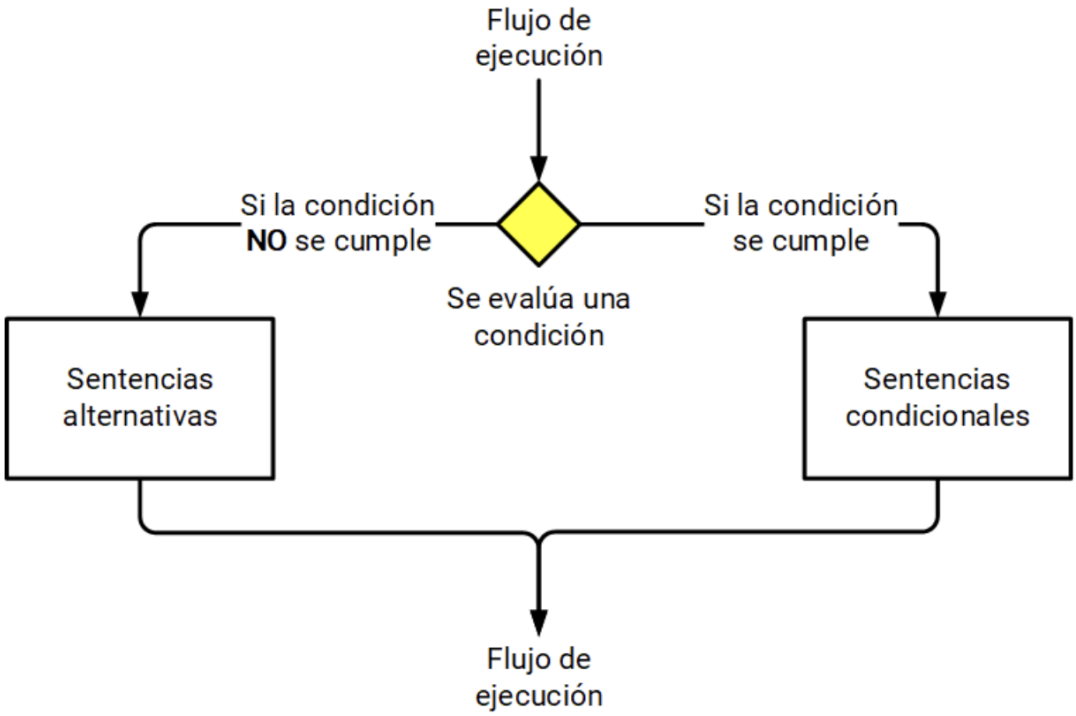
[20]:
```
# ENTRADA
# Se solicita el número al usuario
numero = input("Favor ingresar número para evaluar su paridad: ")

# PROCESAMIENTO
# Se realiza el cambio de tipo de dato a número entero
numero = int(numero)

# Se evalúa la paridad del número ingresado
if numero%2 == 0:
  print("El número ingresado es par")
print("Fin del programa")
```

```
Fin del programa
```

3.2 Sentencia `if-else`

Una alternativa muy utilizada al momento de trabajar con sentencias condicionales es el uso de decisiones alternativas. Esta se compone de una expresión booleana, y un bloque de sentencias condicionales que solo se ejecutará si la condición se evalúa con un valor verdadero (`True`). En caso de que la condición evaluada arroje un valor falso (`False`), se ejecutará el bloque de sentencias alternativas que sucede a la palabra reservada `else`.

El siguiente diagrama de flujo ilustra la ejecución de una estructura de decisión simple.



La **sintaxis** (forma de escribir) que Python utiliza para una estructura de decisión alternativa es la siguiente:

```
if <condición>:
    # Se ejecuta si la condición se cumple
    <Bloque de sentencias condicionales>
else:
    # Se ejecuta si la condición NO se cumple
    <Bloque de sentencias alternativo>
# Se ejecuta independiente de si al condición se cumple o no
<Bloque de sentencias que sigue>
```

A continuación se utilizará el ejemplo anterior, donde se entregará un mensaje al usuario si el número que se ingresa por teclado es par o, en su defecto, es impar:

[21]:
```
# ENTRADA
# Se solicita el número al usuario
numero = input("Favor ingresar número para evaluar su paridad: ")

# PROCESAMIENTO
# Se realiza el cambio de tipo de dato a número entero
numero = int(numero)

# Se evalúa la paridad del número ingresado
if numero%2 == 0:
  print("El número ingresado es par")
else:
```

```
  print("El número ingresado NO es par")
print("Fin del programa")
```

```
El número ingresado NO es par
Fin del programa
```

NOTA: La sentencia `else` nunca lleva condición, pues su ejecución se activa automáticamente cuando el if inmediatamente superior a este resulta en `False`.

Si intento añadir una condición, provocaré un error de sintaxis:

[22]:
```
valor = 5

if valor%2 == 0:
  print('El número almacenado en valor es par')
else valor%2 != 0:
  prinr('El número almacenado en valor es impar')
```

```
  Cell In[22], line 5
    else valor%2 != 0:
         ^
SyntaxError: expected ':'
```

Si existiesen varias sentencias `if`, la sentencia `else` solo responde a la que esté inmediatamente antes del `else`.

En este caso, como el `else`, solo está revisando la condición `valor == 0`, si el número fuese positivo imprimiría tanto `"El número es positivo"`, por la condición `valor > 0`, como `"El número es negativo"`, porque no se está cumpliendo la condición `valor == 0`.

[23]:
```
valor = 32

if valor > 0:
  print('El número es positivo')
if valor == 0:
  print('El número es cero')
else:
  print('El número es negativo')
```

```
El número es positivo
El número es negativo
```

3.3 Decisiones consecutivas y anidadas

Antes de continuar, se propone el siguiente ejercicio para practicar los conceptos vistos:

Diseñe un programa que solicite por teclado la edad de 2 personas e indique cuál es la más joven. Tenga en consideración que ambas personas pueden tener la misma edad, por lo que se debe generar un mensaje adecuado para cada caso

[24]:
```
# ENTRADA
# Se solicita la edad de las personas respectivas
persona_1 = input("Favor ingresar la edad de la primera persona: ")
persona_2 = input("Favor ingresar la edad de la segunda persona: ")

# PROCESAMIENTO
# Se realiza el cambio de tipo de dato, en este caso se debe cambiar a número␣
 ↪entero
persona_1 = int(persona_1)
persona_2 = int(persona_2)

# Se realizan las sentencias condicionales que se solicitan en el enunciado
if persona_1 < persona_2:
  print("La primera persona es menor")
if persona_1 > persona_2:
  print("La segunda persona es menor")
if persona_1 == persona_2:
  print("Ambas personas tienen la misma edad")
```

```
La segunda persona es menor
```

En el ejercicio anterior, se puede ver como se trabajó con **múltiples decisiones consecutivas**, donde en cada caso se especifica la condición respectiva.

Aprovecharemos que las condiciones son mutuamente excluyentes para presentar el mismo ejemplo, pero considerando para la solución el uso de **decisiones anidadas**.

[25]:
```
# ENTRADA
# Se solicita la edad de las personas respectivas
persona_1 = input("Favor ingresar la edad de la primera persona: ")
persona_2 = input("Favor ingresar la edad de la segunda persona: ")

# PROCESAMIENTO
# Se realiza el cambio de tipo de dato, en este caso se debe cambiar a número␣
 ↪entero
persona_1 = int(persona_1)
persona_2 = int(persona_2)

# Se realizan las sentencias condicionales que se solicitan en el enunciado
if persona_1 < persona_2:
  print("La primera persona es menor")
else:
  if persona_1 > persona_2:
    print("La segunda persona es menor")
  else:
    print("Ambas personas tienen la misma edad")
```

```
La segunda persona es menor
```

Al ver la resolución del ejercicio anterior se puede apreciar el uso de `if-else`, pero dentro de la

condición `else` existen más condiciones. Lo anterior se conoce como **decisiones anidadas**.

Cabe señalar que el `else` es la negación absoluta del `if` que se encuentra **inmediatamante superior a este**.

Para dejar más clara la idea anterior, se desarrollará nuevamente el ejercicio anterior pero usando decisiones consecutivas y `else`:

[27]:
```
#ENTRADA
#Se solicita la edad de las personas respectivas
persona_1 = input("Favor ingresar la edad de la primera persona: ")
persona_2 = input("Favor ingresar la edad de la segunda persona: ")

#PROCESAMIENTO
#Se realiza el cambio de tipo de dato, en este caso se debe cambiar a número␣
 ↪entero
persona_1 = int(persona_1)
persona_2 = int(persona_2)

#Se realizan las sentencias condicionales que se solicitan en el enunciado
if persona_1 < persona_2:
  print("La primera persona es menor")
if persona_1 > persona_2:
  print("La segunda persona es menor")
else:
  print("Ambas personas tienen la misma edad")
```

```
La primera persona es menor
Ambas personas tienen la misma edad
```

Como se puede observar, si la primera decisión se cumple (si la primera persona es menor) se entrega el mensaje por pantalla y como la segunda decisión no puede cumplirse (si la segunda persona es menor), se entrega el mesaje que conlleva el `else`. Esto se debe a lo mencionado anteriormente:

El `else` es la negación absoluta del `if` que se encuentra inmediatamante superior a este.

Para reforzar los contenidos y aplicar las últimas indicaciones entregadas se propone el siguiente ejercicio:

En Chile, si las personas que tienen menos de 18 años, son consideradas menores de edad; aquellas que están en el rango etario de 18 a 35 años se les considera adultos-jóvenes; quienes superan los 35 años, pero aún no alzanzan los 65 años, son consideradas adultos, y los que tiene 65 años o más son consideradas adultos mayores o miembros de la "tercera edad".

Se solicita que construya un programa en Python que solicite por teclado la edad de una persona e indique su categoría a partir del rango etario dado.

Resolución 1: **Decisiones consecutivas**

[28]:
```
# ENTRADA
# Se solicita la edad a la persona
edad_persona = input("Favor ingresar la edad de la persona en estudio: ")
```

```
# PROCESAMIENTO
# Se realiza el cambio de tipo de dato, en este caso en particular se␣
 ↪transformará a número entero
edad_persona = int(edad_persona)

# Se plantean las condiciones de acuerdo a lo estipulado en el enunciado
if edad_persona < 18:
  print("La persona es menor de edad")
if edad_persona >= 18 and edad_persona <= 35:
  print("La persona es adulto-joven")
if edad_persona > 35 and edad_persona < 65:
  print("La persona es adulto")
if edad_persona >= 65:
  print("La persona es adulto mayor")
```

```
La persona es adulto-joven
```

Resolución 2: **Decisiones anidadas**

[29]:
```
# ENTRADA
# Se solicita la edad a la persona
edad_persona = input("Favor ingresar la edad de la persona en estudio: ")

# PROCESAMIENTO
# Se realiza el cambio de tipo de dato, en este caso en particular se␣
 ↪transformará a número entero
edad_persona = int(edad_persona)

# Se plantean las condiciones de acuerdo a lo estipulado en el enunciado
if edad_persona < 18:
  print("La persona es menor de edad")
else:
  if edad_persona <= 35:
    print("La persona es adulto-joven")
  else:
    if edad_persona < 65:
      print("La persona es adulto")
    else:
      print("La persona es adulto mayor")
```

```
La persona es adulto-joven
```

Como podemos ver, al usar `if-else`, no tenemos que escribir condiciones tan extensas, pues se ingresa a `else` solo cuando el `if` no se cumple. Lo que implica que no puedo ingresar al código del primer `else` si la persona es menor que 18, por lo que no vale la pena verificarlo de nuevo.

Esto no implica que repetir las condiciones signifique que el código sea incorrecto, solo que está haciendo algunas operaciones que podrían considerarse redundantes.

3.4 Sentencia `if-elif-else`

Como vimos anteriormente, podemos usar `if` y `else` para definir distintas salidas para nuestro código. Sin embargo, a partir del último ejemplo, pudimos ver que, a medida que las instrucciones anidadas aumentan, la **complejidad** del código también lo hace.

Para reducir las anidaciones de código, en Python existe la sentencia `elif` (contracción de "*else if*"), la cuál nos permite reducir las anidaciones de `if-else` usando el equivalente a decir: "*Si no se cumple el primer* ***`if`****, revisa la siguiente condición (****`elif`****), si no se cumple esta, revisa la siguiente, y así sucesivamente hasta alcanzar el* ***`else`****.*"

Revisemos como quedaría para el caso del ejemplo anterior:

[30]:
```
# ENTRADA
# Se solicita la edad a la persona
edad_persona = input("Favor ingresar la edad de la persona en estudio: ")

# PROCESAMIENTO
# Se realiza el cambio de tipo de dato, en este caso en particular se␣
 ↪transformará a número entero
edad_persona = int(edad_persona)

# Se plantean las condiciones de acuerdo a lo estipulado en el enunciado
if edad_persona < 18:
  print("La persona es menor de edad")
elif edad_persona <= 35:
  print("La persona es adulto-joven")
elif edad_persona < 65:
    print("La persona es adulto")
else:
    print("La persona es adulto mayor")
```

```
La persona es adulto
```

En este caso, se asume que todas las sentencias `if-elif-elif-else` corresponden al mismo bloque. Por lo que Python revisará las condiciones *una a una* hasta encontrar una que se cumpla (es más, **no puede** haber una sentencia `elif` sin una sentencia `if` que la anteceda).

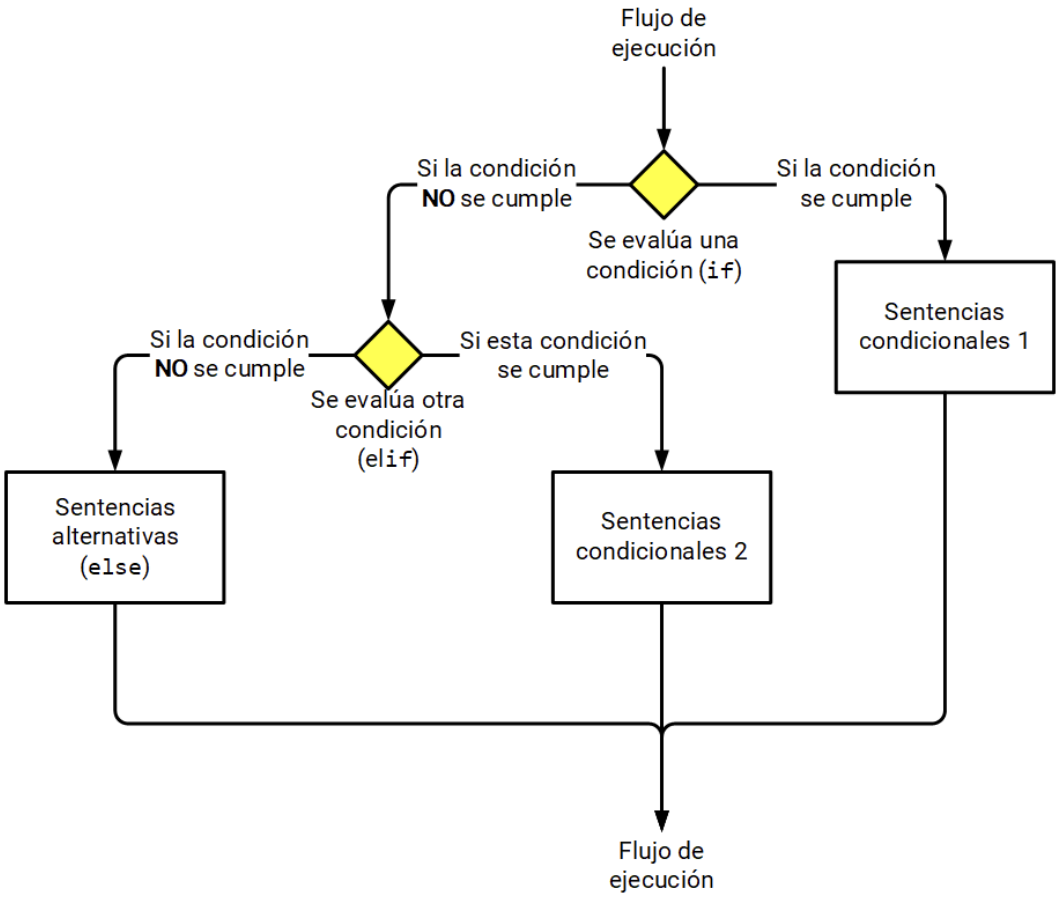
Una vez que encuentra una condición que se cumple, es decir, resulte en `True`, ejecuta las sentencias condicionadas. Por ejemplo, si la persona tuviese 29 años, imprimirá `"La persona es adulto-joven"` y luego saldrá del bloque, sin revisar las sentencias que siguen.

La **sintaxis** (forma de escribir) que Python utiliza para una estructura de decisión compuesta de `if`, `elif` y `else` es la siguiente:

```
if <condición>:
  # Se ejecuta si la condición se cumple
  <Bloque de sentencias condicionales>
# Si la condición de arriba no se cumple
elif <condición>:
  # Se ejecuta si la condición se cumple
```

```
  <Bloque de sentencias condicionales>
# Si las condiciones de arriba no se cumplen
elif <condición>:
  # Se ejecuta si la condición se cumple
  <Bloque de sentencias condicionales>
#...
# Puedo poner cuantos elif yo necesite
# Si las condiciones de arriba no se cumplen
elif <condición>:
  # Se ejecuta si la condición se cumple
  <Bloque de sentencias condicionales>
# Si ninguna de las condiciones anteriores
else:
  # Se ejecuta si NINGUNA de las condiciones anteriores se cumplen
  <Bloque de sentencias alternativo>
# Se ejecuta independiente de si al condición se cumple o no
<Bloque de sentencias que sigue>
```

El siguiente diagrama de flujo ilustra la ejecución de una estructura de decisión que utiliza un `if`, un `elif` y un `else`.



Vale la pena destacar que este bloque puede terminar sin `else`, o que puedo agregar cuantos elif quiera entre `if` y `else`.

Del mismo modo, si incluyera un `else` en medio del bloque o un `elif`, sin un `if` previamente definido, provocaría un error.

3.5 Indentación

A diferencia de otros lenguajes de programación, Python no utiliza símbolos para señalar el comienzo y el fin de un bloque de sentencias (`if-else`). En su lugar, Python utiliza la indentación (sangría).

La importancia de la indentación es separar el bloque de sentencias condicionales a ejecutar en caso de cumplir la condición dada. Su aparición se genera posterior a la utilización del caracter dos puntos (`:`).

El bloque de sentencias condicionales termina cuando la indentación está al mismo nivel que la condición condicional que la originó.

No utilizar la indentación genera un `IndentationError` (**error de indentación**):

[31]:
```
x = int(input("Ingrese un número: "))
if x < 5:
print("Opción 1.")
else:
print("Opción 2.")
```

```
  Cell In[31], line 3
    print("Opción 1.")
    ^
IndentationError: expected an indented block after 'if' statement on line 2
```

4 Bibliografía

4.1 Formato y buenas prácticas

Van Rossum, G., Warsaw, B., & Coghlan, N. (2001, 21 julio). *PEP 8 – Style Guide for Python Code*. Python Enhancement Proposals. Recuperado 1 de agosto de 2022, de https://peps.python.org/pep-0008/

4.2 Operadores

GeeksforGeeks. (2022, 15 julio). *Python Operators*. Recuperado 3 de agosto de 2022, de https://www.geeksforgeeks.org/python-operators/

Shaw, Z. (2017). Memorizing Logic. En *Learn Python 3 the Hard Way* (p. 96). Addison-Wesley.

4.3 Sentencias condicionales

Beecher, K. (2017). Effective Building Blocks. En Computational Thinking (p. 115). BCS.

Shaw, Z. (2017). What If. En *Learn Python 3 the Hard Way* (p. 104). Addison-Wesley.