

Condicionales

Unidad 3

Apunte de cátedra

2do Cuatrimestre 2023

Pensamiento computacional (90)
Cátedra: Camejo

.UBA XXI

1. Sentencias condicionales

Los programas se hacen para obtener resultados con distintos datos de entrada. Al definir reglas formales de transformación, logramos un mismo modelo genérico que se puede aplicar a distintos valores. Pero, ¿qué pasa si existen casos o patrones que requieren tratamientos diferenciados? Para lidiar con esa situación necesitaremos escribir programas que exhiban dos capacidades importantes. En primer lugar, se deben poder **identificar los casos o patrones**; es decir, las alternativas, en general, que se puedan presentar (pueden ser patrones ingresados o producidos internamente). En segundo término, tendrán que saber **elegir diferentes caminos de acción para cada caso**.

- ✓ *NO TIENE SENTIDO CONOCER LAS DIFERENCIAS SI NO PODEMOS TOMAR ACCIONES DISTINTAS*
- ✓ *NO PODEMOS REALIZAR TRATAMIENTOS DIFERENCIADOS SI NO SABEMOS RECONOCER CUÁLES SON LOS CASOS ESPECIALES*

Necesitaremos programas más inteligentes que los que hemos construido hasta ahora. No sólo deben transformar datos y obtener resultados; también deben decidir qué tipo de tratamiento darán a cada caso o situación. Es decir, tendrán que tener la habilidad de preguntarse y responderse, además de emplear **Flujos de Control alternativos**.

La idea es que un programa pueda formular una pregunta, o varias, cuando lo necesite; de modo que la respuesta, o combinación de respuestas obtenidas, le permita decidir sin cabos sueltos qué camino tomar. Para este objetivo, los modelos matemáticos que venimos empleando no alcanzan. Necesitamos otro Modelo Formal; y el Modelo Lógico es la respuesta.

1.1 Introducción a lógica

La **lógica simbólica o lógica bivalente** es un modelo lógico que nos viene de maravilla para estos propósitos. Todos los lenguajes (como Python) disponen de mecanismos para preguntar y responder haciendo uso de expresiones y operadores lógicos.

En **lógica** tenemos expresiones (lógicas) a las que se asocia algún valor (valor de verdad) posible, del Universo de valores disponibles. A diferencia de las Matemáticas, la Lógica **Binomial** tiene un espacio de resultados finito: **Verdadero V o Falso F**.

Estos valores son disjuntos (no tienen elementos en común) y complementarios (cada evento y su evento complementario no pueden ocurrir simultáneamente); es decir, en lógica binomial no existen las verdades a medias: si algo no es Verdadero, entonces es Falso; y viceversa. Pero algo no puede ser verdadero y falso al mismo tiempo.

1.1.1. Expresiones Booleanas

El nombre teórico de una expresión lógica es **proposición**. En programación también las llamamos frecuentemente **expresiones booleanas**.

Algunos ejemplos:

- Hoy llueve
- Estoy en Sudamérica
- Las zanahorias son celestes
- 4 es un número par

Cada una de las expresiones dichas anteriormente pueden evaluarse como Verdaderas o Falsas. Es decir, les podemos dar un valor de verdad: si abro la app del clima y dice que va a estar soleado, “hoy llueve” es falso. Argentina está en Sudamérica, entonces “estoy en Sudamérica” es verdadero. De la misma forma, las otras dos expresiones son falso y verdadero respectivamente.

Si evaluamos cualquiera de las expresiones de los ejemplos obtendremos su valor. Y ese valor tiene sólo dos posibles resultados: Verdadero o Falso.

¿Por qué nos importa esto? Porque todas las preguntas que nuestros programas necesiten formularse deberán estar escritas como expresiones lógicas.

En programación las llamamos **condiciones**. Y si sólo obtendrán como respuesta Si o No, entonces lo más probable es que con una pregunta simple no le alcance al programa para identificar perfectamente una situación, caso o patrón. Así que, usaremos estas operaciones para elaborar preguntas tan sofisticadas como sea necesario.

1.1.2. Operadores lógicos

Para realizar estas condiciones vamos a usar lo que llamamos **operadores lógicos**, que aplicados a una o dos expresiones nos devuelven *Verdadero* o *Falso*. Para poder usar correctamente un operador, debemos conocer muy bien cómo se resuelve esa operación; igual que en matemáticas.

Las operaciones lógicas que usaremos para escribir las condiciones en nuestros programas son las siguientes:

Operador	Descripción	Símbolo en Python
Negación (~)	Es equivalente a “ no ”: si algo es Verdadero, lo convierte en Falso y viceversa.	<ul style="list-style-type: none"> • <code>not</code>
Conjunción (^)	Es equivalente a “ y ”. Solamente da <i>Verdadero</i> si todas las expresiones que evalúa son <i>verdaderas</i> .	<ul style="list-style-type: none"> • <code>and</code>

Disyunción (v)	Es equivalente a “o”, es opcionalidad, alternativas o alternativas. Da verdadera si alguna de las expresiones que evalúa es Verdadera. Solamente si es <i>Falso</i> cuando todas las expresiones son <i>falsas</i> .	<ul style="list-style-type: none"> or
-------------------------	--	--

Para cada operador lógico se tienen **Tablas de Verdad**, las cuales nos dan el resultado de usar estos operadores con diferentes valores de verdad:

AND


A	B	Resultado
V	V	V
V	F	F
F	V	F
F	F	F

OR

A	B	Resultado
V	V	V
V	F	V
F	V	V
F	F	F

NOT

A	Resultado
V	F
F	V

 **Nota:** Las expresiones lógicas complejas (más de un operador) se resuelven (igual que en matemáticas) respetando precedencias y de izquierda a derecha. También admiten el uso de () para alterar precedencias.

Sin embargo, si no tenemos alteraciones de precedencias (paréntesis), vamos a priorizar los AND por sobre los OR. Esto es algo específico de *algunos* lenguajes de programación y, en este caso, también de Python.

Ejemplo:

- V or F and F = V or (F and F)* = V or F = V * precedencia de AND por sobre OR
 - V or F or F = (V or F) or F = V or F = V
 - V and V and F = (V and V) and F = V and F = F

Pero si al mismo ejemplo les agregamos paréntesis para alterar precedencias, podríamos obtener un resultados diferente:

- (V or F) and F = V and F = F -> sin modificación de precedencias, este resultado era V
 - V or (F or F) = V or F = V -> no cambia
 - V and (V and F) = V and F = F -> no cambia

<pre> main.py 1 # Sin modificar precedencias 2 valor1 = True or False and False 3 valor2 = True or False or False 4 valor3 = True and True and False 5 6 # Modificando precedencias 7 valor1_modificado = (True or False) and False 8 valor2_modificado = True or (False or False) 9 valor3_modificado = True and (True and False) 10 11 print("Valor1:", valor1) 12 print("Valor2:", valor2) 13 print("Valor3:", valor3) 14 15 print("\nModificando precedencias:") 16 print("Valor1:", valor1_modificado) 17 print("Valor2:", valor2_modificado) 18 print("Valor3:", valor3_modificado) </pre>	<pre> Valor1: True Valor2: True Valor3: False Modificando precedencias: Valor1: False Valor2: True Valor3: False </pre>
--	--

Conclusión

El operador lógico OR (or) es asociativo. Esto significa que cuando tenés múltiples condiciones conectadas con el operador OR en una expresión, la evaluación se realiza de izquierda a derecha y se detiene tan pronto como se encuentra un valor True. Si ninguno de los valores es True, entonces se devuelve False.

Además, el operador lógico AND (and) también es asociativo. Al igual que con el operador OR, cuando tenés múltiples condiciones conectadas con el operador AND en una expresión, la evaluación se realiza de izquierda a derecha y se detiene tan pronto como se encuentra un valor False. Si todos los valores son True, entonces se devuelve True.

Sin embargo, en expresiones combinadas, en Python AND tiene precedencia por sobre OR. El resto de la expresión se evalúa de izquierda a derecha.

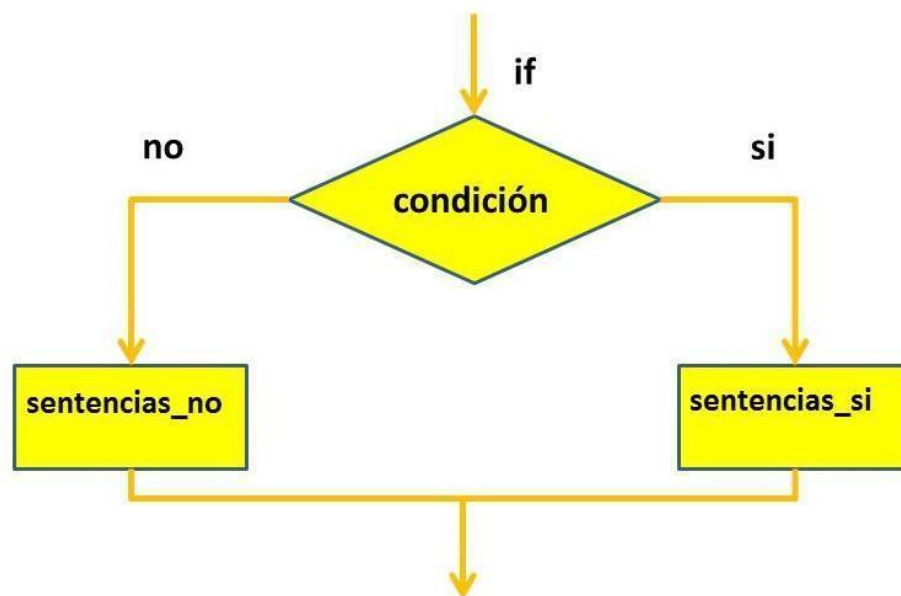
1.2. If

Todo Lenguaje de Programación Procedural dispone de una o más herramientas para poder trabajar en estas circunstancias. El tipo de herramientas que se precisa se llama genéricamente **sentencias condicionales**.

Las sentencias condicionales pertenecen a lo que llamamos **estructuras de control condicionales**, y se denominan de esta manera porque permiten alterar el **Flujo de Control Normal** de un Programa (FCP).

Python dispone de una **sentencia condicional** bastante genérica y versátil; la sentencia **if**.

Esta es la sentencia que usaremos para tomar decisiones y hacer bifurcaciones en general, y funciona de la siguiente manera:



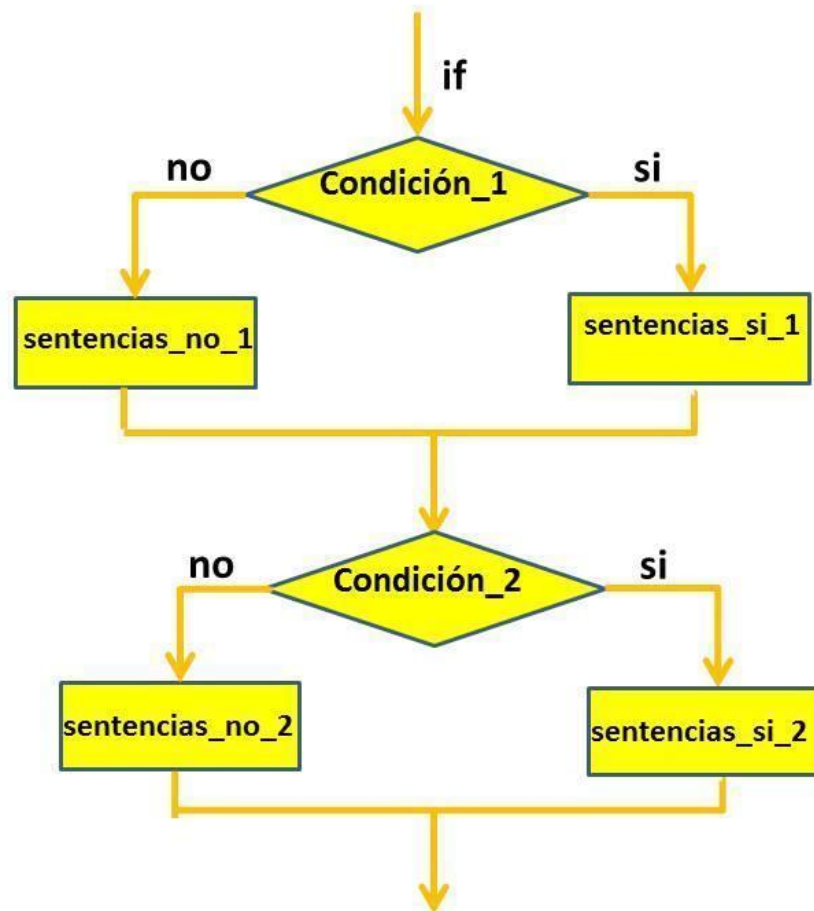
Cuando el **Flujo de Control de un Programa (FCP)** alcanza una sentencia **if**, lo primero que se hace es evaluar la **condición** asociada a ella. Si la **condición** da **V o Verdadero** (o sea, si la respuesta a la pregunta es **SI**), se ejecutarán las sentencias asociadas a esa elección y **no** a la salida por **F o Falso** (la respuesta es **NO**). En la gráfica, si da **V** se ejecutan las sentencias de la rama de la derecha. En cambio, si da **F o Falso**, se ejecutarán otras sentencias, las que están asociadas a esa respuesta (en la gráfica, la rama de la izquierda).

Es decir, el **FCP** llega a la **condición**, la evalúa y luego **bifurca**, por derecha o izquierda. Hay un grupo de sentencias que, en ese caso (debido a la respuesta obtenida), no se ejecutarán.

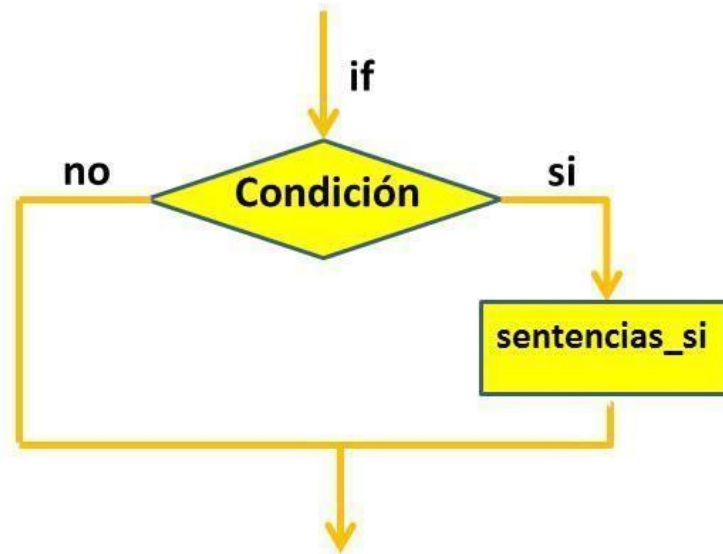
El punto de unión al final de la gráfica indica el fin de la sentencia **if**. Es decir, que en algún punto la bifurcación acaba y el **FCP** sigue su curso de manera unificada (al menos hasta la próxima bifurcación).

Dicho en otros términos, cuando la sentencia **if** finaliza, no importa si alcanzamos ese punto por la rama del V o del F, en cualquiera de los dos casos se ejecutarán las sentencias a continuación de ella.

Puede ser que a un **if**, le siga otra sentencia **if**. Pero, en realidad el **FCP** al arribar a la segunda sentencia, no sabe por dónde atravesó la primera. El siguiente esquema muestra cómo sería un Diagrama Lógico para dos **if** continuos:



Es muy frecuente también escribir **if** que no hace nada por la salida del **F**, sólo por **V**. Esto es para cuando hay que sumar alguna acción a lo general para ciertos casos. La gráfica sería algo así:



A continuación se ve la sintaxis del if en Python:

```
if condición 1:
    instrucción 1
    instrucción 2
elif condición 2:
    instrucción A
    instrucción B
else:
    instrucción x
    instrucción y
```

*elif es como decir “else if”, es el camino “no”, pero evaluando una nueva condición. Es una mezcla entre else + if

La sentencia **if** es una de las Sentencias Estructuradas de Python (que se escribe en más de un renglón). Las sentencias estructuradas normalmente tienen una primera línea de **encabezado** y un **cuerpo**. El **cuerpo** siempre se escribe con indentación (desplazamiento del inicio del renglón a la derecha). Esta indentación debe ser igual para todas las sentencias del cuerpo y es la forma que tiene Python para identificar cuáles sentencias pertenecen a él.

Esta estructura nos va a permitir ejecutar una porción de código u otra. Si la condición que acompaña al if es Verdadera, van a correr las instrucciones 1, 2 y el resto de las que estén dentro del cuerpo del if. Si en cambio, la condición correcta es la que acompaña elif van a ejecutarse las instrucciones del cuerpo del elif, y lo mismo con el else.

La traducción al lenguaje coloquial sería “**si** la condición 1 es Verdadera ejecutá la instrucción 1 y 2, **si en cambio** la condición 2 es correcta ejecutá las instrucciones A, B y **si no** ejecutá las instrucciones x e y.

Notas:

1. Lo que está entre [] es opcional
2. En el caso de `if PYTHON` puede encontrar un Encabezado `elif` o un Encabezado `else`; sabe entonces que continúa dentro del `if`, pero esa es otra rama. Si viene ejecutando el Cuerpo de la rama del `si`, de todas maneras va a ignorar esas ramas. Y sigue ignorando hasta encontrar la primera sentencia alineada al `if` (o más a la izquierda) que no sea ni `elif`, ni `else`

Dentro del grupo de sentencias de cada rama de un **if** puede aparecer cualquier sentencia válida de Python, esto incluye una o varias sentencias **if**. Cuando dentro de una rama de **if** aparece otro **if** decimos que los **if** están **anidados**.

1.3. Operadores en Python

Las **expresiones simples** son las preguntas que puede formularse Python directamente. Esas preguntas pueden tener una respuesta conocida de antemano, es decir que actúan como constantes, o no.

En la sesión anterior vimos operadores aritméticos (por ejemplo +, -, *), pero existen otros tipos de operadores que también dan un resultado cuando se evalúan.

Básicamente puede asociar un valor de verdad a la evaluación de cualquier expresión aritmética (normalmente F si es 0 -cero-, V si es $\neq 0$), puede comparar valores de datos usando los **operadores de comparación**, o averiguar si un valor está dentro de un grupo, **operadores de pertenencia**. También existen los **operadores lógicos** que ya vimos su definición anteriormente.

1.3.1. Operadores de Comparación o Desigualdad:

Símbolo	Definición	Ejemplo
<code>==</code>	igual	<code>a == 4</code> # La expresión va a valer True si 'a' vale 4
<code>!=</code>	distinto	<code>b != 4</code> # La expresión va a valer True si 'b' NO vale 4
<code><</code>	menor	<code>2 < 4</code> # devuelve True <code>3 < 1</code> # devuelve False

<=	menor o igual	<pre>2 <= 4 # devuelve True 3 <= 1 # devuelve False 3 <= 3 # devuelve True</pre>
>	mayor	<pre>2 > 4 # devuelve False 3 > 1 # devuelve True</pre>
>=	mayor o igual	<pre>2 >= 4 # devuelve False 3 >= 1 # devuelve True 3 >= 3 # devuelve True</pre>

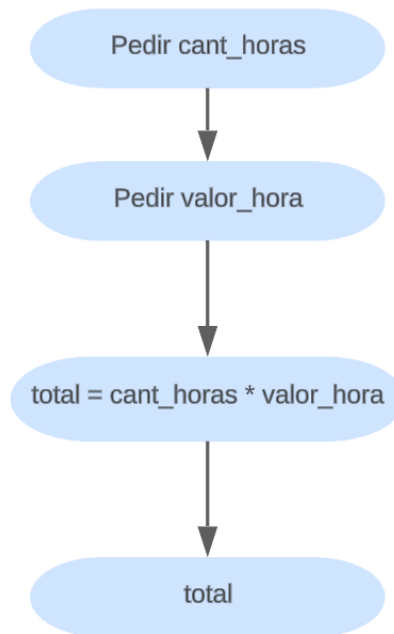
1.3.2. Operadores de Pertenencia:

Símbolo	Definición	Ejemplo
in	Pertenece	<pre>"razón" in "corazón" # devuelve True "tazón" in "corazón" # devuelve False</pre>
not in	NO pertenece	<pre>"razón" not in "corazón" # devuelve False "tazón" not in "corazón" # devuelve True</pre>

Con estos pocos elementos podemos arreglarnos para hacer una colección de preguntas simples o sofisticadas, que en su conjunto nos permitan identificar cualquier situación o patrón que busquemos.

1.4. Ejemplo

Debemos calcular el pago a un trabajador. El cálculo debe hacerse por la cantidad de horas trabajadas. Le vamos a pedir al usuario por pantalla la cantidad de horas que trabaja y cuánto vale cada hora de trabajo. En el siguiente gráfico se muestra el flujo (**FCP**) del programa.

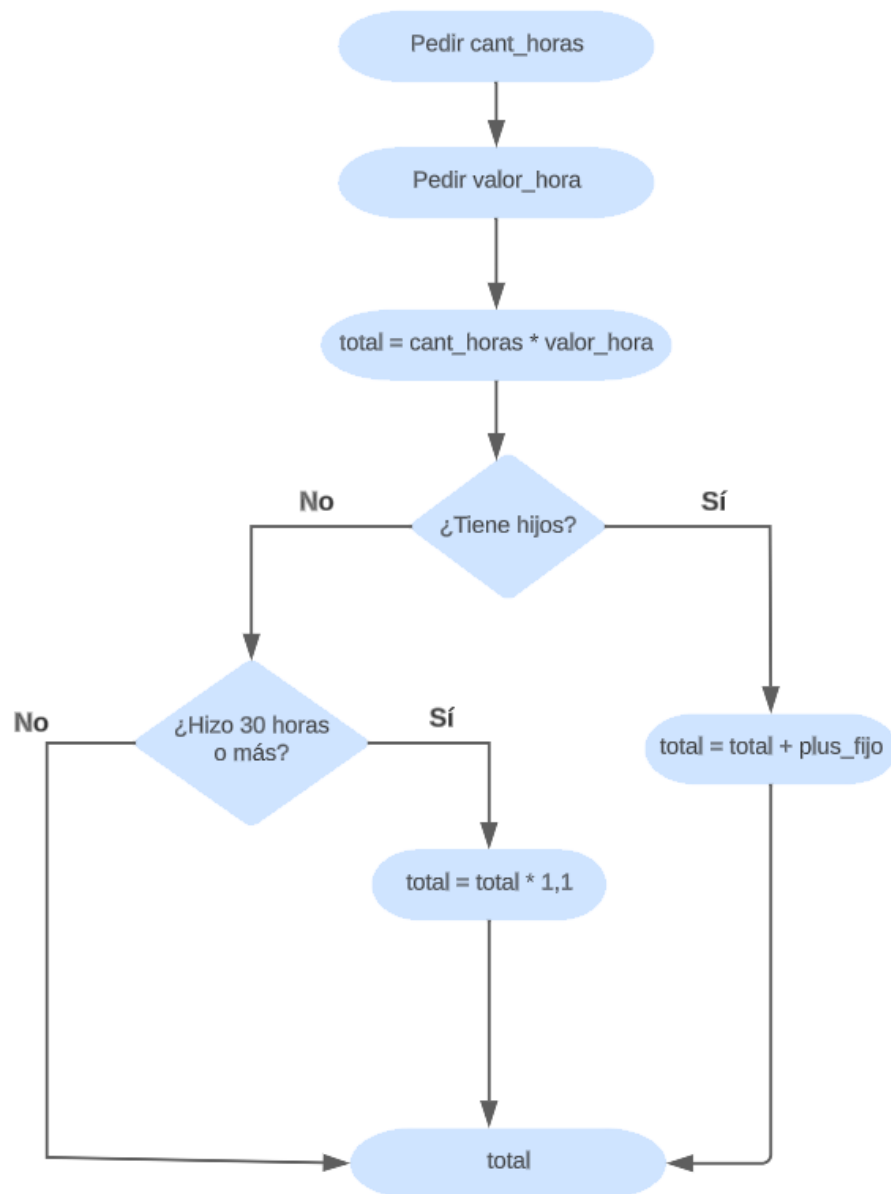


Ahora imaginemos que en la empresa se decide abonar un plus fijo de guardería a todo trabajador que tiene hijos. Y pagar un 10% de incentivo a todo trabajador que haya hecho 30 horas o más y **NO** reciba el plus por guardería.

Es decir, antes el cálculo era simplemente $\text{cant_horas} * \text{valor_hora}$, pero ahora se abren 4 casos posibles para calcular el sueldo del trabajador. **No tenemos un solo modelo o caso de liquidación, sino varios:**

Trabajador con menos de 30 horas y sin hijos	$\text{total} = \text{cant_horas} * \text{valor_hora}$
Trabajador con 30 horas o más y sin hijos	$\text{total} = \text{cant_horas} * \text{valor_hora} * 1.1$
Trabajador con menos de 30 horas y con hijos	$\text{total} = \text{cant_horas} * \text{valor_hora} + \text{plus_fijo}$
Trabajador con 30 horas o más y con hijos	$\text{total} = \text{cant_horas} * \text{valor_hora} + \text{plus_fijo}$

Veamos una posible solución al problema en forma de gráfico a partir del anterior, en el que vemos la toma de decisiones y los diferentes caminos que se pueden tomar según las condiciones.



Podemos ver que se complejizó el problema que habíamos solucionado anteriormente. Ahora, además del cálculo del total de las horas por el valor de las horas, tenemos que preguntarnos si el empleado trabajó más de 30 horas y si tiene hijos.

Como se puede observar, cada pregunta abre dos caminos que van a cambiar el valor del total de diferentes formas. Si tiene hijos le sumamos el plus, y si no nos hacemos la siguiente pregunta: si hizo más de 30 horas, se le aumenta el total un 10% y sino el total queda igual.

Notemos que la pregunta dentro de una pregunta es un **if anidado** (un if dentro de otro).

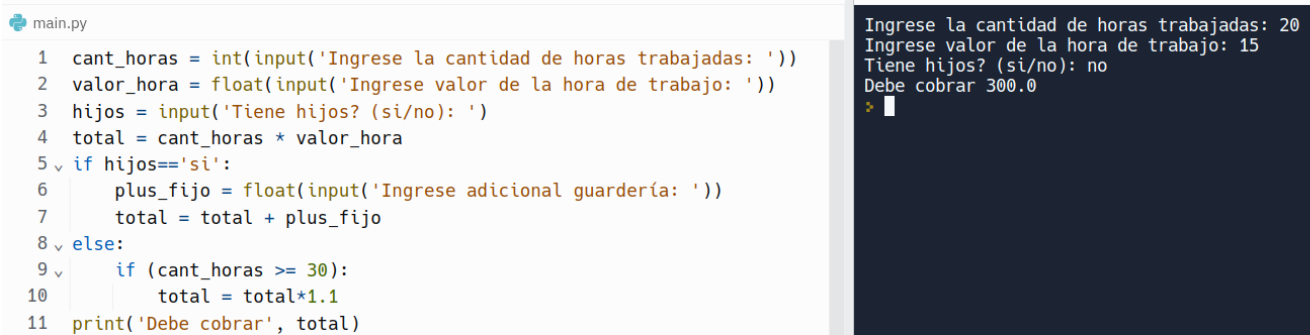
¿Cómo escribiríamos el programa de cálculo del pago por horas en Python?

Ejemplo de cálculo de pago por horas

```
cant_horas = int(input('Ingrese la cantidad de horas trabajadas: '))
valor_hora = float(input('Ingrese valor de la hora de trabajo: '))
hijos = input('Tiene hijos? (si/no): ')
total = cant_horas * valor_hora
if hijos=='si':
    plus_fijo=float(input('Ingrese adicional guardería: '))
    total = total + plus_fijo
else:
    if (cant_horas >= 30):
        total = total*1.1
print('Debe cobrar', total)
```

Veamos si funciona, lo ejecutamos y probamos los casos posibles:

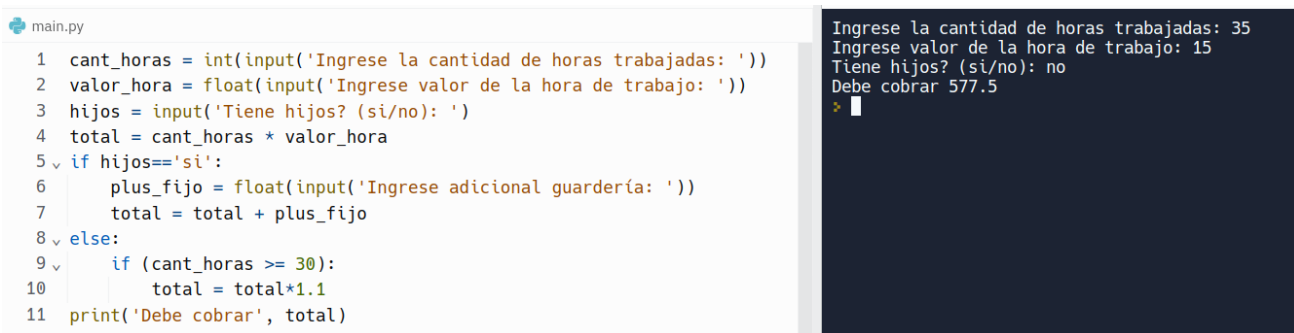
Este es el caso en el que NO tiene hijos y NO trabaja 30 horas o más, por lo tanto no ingresa dentro del primer if (fíjate que no imprime la pregunta 'Ingrese adicional guardería') sino que ejecuta el código dentro del else, y como no trabajó 30 horas o más, el total es directamente $20 * 15$:



```
main.py
1 cant_horas = int(input('Ingrese la cantidad de horas trabajadas: '))
2 valor_hora = float(input('Ingrese valor de la hora de trabajo: '))
3 hijos = input('Tiene hijos? (si/no): ')
4 total = cant_horas * valor_hora
5 if hijos=='si':
6     plus_fijo = float(input('Ingrese adicional guardería: '))
7     total = total + plus_fijo
8 else:
9     if (cant_horas >= 30):
10        total = total*1.1
11 print('Debe cobrar', total)
```

Ingrese la cantidad de horas trabajadas: 20
Ingrese valor de la hora de trabajo: 15
Tiene hijos? (si/no): no
Debe cobrar 300.0

Este es el caso en el que NO tiene hijos y SÍ trabaja 30 horas o más. Como el usuario indicó que trabaja 35 horas, la cuenta es $35 * 15 * 1,1$:



```
main.py
1 cant_horas = int(input('Ingrese la cantidad de horas trabajadas: '))
2 valor_hora = float(input('Ingrese valor de la hora de trabajo: '))
3 hijos = input('Tiene hijos? (si/no): ')
4 total = cant_horas * valor_hora
5 if hijos=='si':
6     plus_fijo = float(input('Ingrese adicional guardería: '))
7     total = total + plus_fijo
8 else:
9     if (cant_horas >= 30):
10        total = total*1.1
11 print('Debe cobrar', total)
```

Ingrese la cantidad de horas trabajadas: 35
Ingrese valor de la hora de trabajo: 15
Tiene hijos? (si/no): no
Debe cobrar 577.5

Este es el caso en el que Sí tiene hijos y NO trabaja 30 horas o más. Como la respuesta a la pregunta si tiene hijos es “sí”, entra en el código del if y le pregunta el adicional de la guardería. La cuenta que se hace es $20 * 15 + 50$ que es lo que se indicó que es el adicional.

```
main.py
1 cant_horas = int(input('Ingrese la cantidad de horas trabajadas: '))
2 valor_hora = float(input('Ingrese valor de la hora de trabajo: '))
3 hijos = input('Tiene hijos? (si/no): ')
4 total = cant_horas * valor_hora
5 if hijos=='si':
6     plus_fijo = float(input('Ingrese adicional guardería: '))
7     total = total + plus_fijo
8 else:
9     if (cant_horas >= 30):
10         total = total*1.1
11 print('Debe cobrar', total)
```

```
Ingrese la cantidad de horas trabajadas: 20
Ingrese valor de la hora de trabajo: 15
Tiene hijos? (si/no): si
Ingrese adicional guardería: 50
Debe cobrar 350.0
```

Este es el caso en el que Sí tiene hijos y Sí trabaja 30 horas o más, como tiene hijos, se ignora que trabaja más de 30 horas y no se le agrega el 10%. La cuenta es $35 * 15 + 50$.

```
main.py
1 cant_horas = int(input('Ingrese la cantidad de horas trabajadas: '))
2 valor_hora = float(input('Ingrese valor de la hora de trabajo: '))
3 hijos = input('Tiene hijos? (si/no): ')
4 total = cant_horas * valor_hora
5 if hijos=='si':
6     plus_fijo = float(input('Ingrese adicional guardería: '))
7     total = total + plus_fijo
8 else:
9     if (cant_horas >= 30):
10         total = total*1.1
11 print('Debe cobrar', total)
```

```
Ingrese la cantidad de horas trabajadas: 35
Ingrese valor de la hora de trabajo: 15
Tiene hijos? (si/no): si
Ingrese adicional guardería: 50
Debe cobrar 575.0
```

Observá que al preguntar si trabajó 30 o más horas no estamos chequeando que tenga hijos; esto es porque al estar el **if anidado** por la salida del **no** tenemos garantía de que si llegamos a ese punto, es porque en la pregunta anterior tomamos la rama del **no**.

O sea, no preguntamos si tiene hijos, porque sabemos que no los tiene, ya que si los tuviera hubiese ingresado a las sentencias correspondientes al if y no hubiese entrado al código dentro del else

Este no es el único modelo posible. Existen infinitas maneras de resolver esta situación.

⚠ ¡Atención!

Prestá atención a cómo se van indentando las sentencias si se anidan if. Es muy importante poner un TAB (un espacio con la tecla TAB del teclado) para la sintaxis del if, ya que esta indentación es lo que le permite a Python distinguir el cuerpo del ciclo del resto de las instrucciones del programa.

Veamos un par más de ejemplos:

```
if clima_hoy == "lluvia":  
    llevar_paraguas()  
llevar_campera()
```

Este código lo que hace es: si el clima de hoy es "lluvia", me dice que lleve el paraguas. Si no llueve, no hace nada particular. Pero, independientemente de si llueve o no, me llevo la campera. Esto es porque 'llevar_campera()' está por fuera del "bloque" del if. Es decir, es donde la bifurcación del si/no del if se une, y el programa sigue con su flujo de control.

```
if nivel_sueño == "mucho":  
    apagar_alarma()  
    seguir_durmiendo()  
else:  
    apagar_alarma()  
    levantarse()
```

Suponiendo que este programa se ejecuta cuando nos suena la alarma a la mañana, tenemos dos caminos:

- Si mi nivel de sueño es "mucho", elijo apagar la alarma y seguir durmiendo.
- Si mi nivel de sueño NO es "mucho" (es decir, es cualquier otro valor distinto a "mucho" como "poquito", "nada", etc), elijo apagar la alarma y levantarme.

```
if necesidad == "hambre":  
    comer_algo()  
elif necesidad == "sed":  
    tomar_agua()  
else:  
    relajarme_en_el_sillon()
```

En este código, verificamos si mi necesidad es igual a "hambre". Si es así, como algo. Si mi necesidad no es "hambre", entonces verifico si es "sed". Si lo es, tomo agua. Y si no es ninguna de las dos anteriores, simplemente me relajo en el sillón de mi casa.

Eso sí: si necesidad es "hambre", ejecuto el código dentro del if y *no verifico si necesidad es "sed" ni me relajo en el sillón*. Como la primera condición del if ya se cumplió, no es necesario seguir con el bloque del if, simplemente ejecutamos la instrucción asociada y listo.

Esto nos dice que sólo vamos a verificar que necesidad sea "sed" si y sólo si necesidad fue distinta de "hambre" (por eso el 'elif').

Al mismo tiempo, sólo me voy a relajar en el sillón si necesidad no fue “hambre” ni “sed”.

```
if clima == "lluvia":  
    if transporte == "colectivo":  
        llevar_paraguas()  
    elif transporte == "auto":  
        llevar_llaves()  
salir_de_casa()
```

En este ejemplo, si llueve, verifico además en qué tipo de transporte voy a viajar. Si es en colectivo, llevo paraguas. Si no es en colectivo, verifico si es en auto. Si lo es, agarro las llaves.
Si no llueve, no verifico nada más.
Si llueve y tomo colectivo, no verifico si voy en auto. Sólo si no voy en colectivo, verifico si voy en auto.
Independientemente de todo lo anterior, salgo de casa al final.

Bibliografía Adicional

Wachenchauzer, R. (2018). *Aprendiendo a programar usando Python como herramienta*. Capítulos 2, 4.