Condicionales

Unidad 3

Apunte de cátedra

Pensamiento computacional (90)
Cátedra: Camejo

.UBA XXI

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 sucede si existen casos o patrones que requieren tratamientos diferenciados? Para lidiar con esa situación (por demás frecuente, cabe decir) necesitaremos escribir programas que exhiban dos capacidades importantes. En primer lugar, 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 construído 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.

Haremos una brevísima introducción a Lógica…

Lógica Simbólica, Lógica Bivalente o Binomial es un Modelo Lógico que nos viene de maravillas para estos propósitos. Todos los Lenguajes (como PYTHON) disponen de mecanismos para preguntar y responder haciendo uso de expresiones y operadores lógicos.

En Lógica (igual que en aritmética) 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: *U={V,F}*

*Verdadero **V** o Falso **F***

Estos valores son disjuntos y complementarios; es decir, en Lógica binomial no existen las verdades a medias, si algo no es Verdadero, entonces es Falso; y viceversa.

El nombre teórico de una expresión lógica es **proposición**, en programación también las llamamos frecuentemente **expresiones booleanas** ya que la Lógica Binomial es una aplicación de uso extendido de un modelo matemático especial denominado Álgebra de Boole.

Algunos ejemplos:

- **p= Hoy llueve**
- **q= Estoy en Sudamérica**
- **r=Las zanahorias son celestes**
- **s= 4 es un número par**

<u>Nota:</u> Normalmente se usan las últimas letras del alfabeto en minúsculas para denotar una expresión lógica

Si evaluamos cualquiera de las expresiones de los ejemplos obtendremos su valor (como 2+2=4). Pero sólo tengo dos posibles resultados: Verdadero o Falso.

Tanto en Matemáticas como en Lógica existen operadores que permiten escribir expresiones complejas. ¿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. Funciona como un juego enigmático donde los jugadores pueden hacer infinitas preguntas, pero sólo obtienen Si o No de respuesta. De la astucia para organizar sus preguntas y el conocimiento parcial que van obteniendo con las respuestas alcanzarán la verdad. Así que, usaremos estas operaciones para elaborar preguntas tan sofisticadas como sea necesario.

Pero…

Para poder usar correctamente un operador debemos conocer muy bien cómo se resuelve esa operación; igual que en matemáticas.

Estamos de acuerdo en que (en sistema decimal): 2+2=4 o 33-11=22

¿Por qué estamos tan seguros de esto? Simple:

Para cada operación existe un Teorema que permite garantizar cómo se obtienen los resultados . Teorema de la Suma, de la Resta, etc.

En el colegio, donde te mantuvieron alejado de comprobaciones teóricas, por una cuestión de edad, simplemente te enseñaron diferentes algoritmos que permiten resolverlas. Y en algunos casos te hicieron estudiar de memoria tablas (de multiplicar y dividir). ¿Para qué las usabas? Al saber de memoria el resultado de esas operaciones simples, lograbas atajos en tus algoritmos.

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

Negación **~** (equivalente a no)

Conjunción **^** (agregado, suma,equivalente a y)

Disyunción **v** (opcionalidad, alternativas, equivalente a o)

Todas ellas tienen su Teorema Fundamental. Sin embargo, aún cuando hayas visto Lógica en tus estudios previos, es probable que no lo conozcas. Y tampoco que sepas diferentes algoritmos para resolverlas. Raro, ¿no? O no tanto… La razón es de índole práctica: (algo que nos gusta mucho a los programadores)

Como en Lógica Bivalente sólo tenemos un espacio de valores (y posibles resultados) de dos elementos: **{V,F}**

1. Si un operador es **monario** (se aplica a un solo término, como la negación **~**), sólo habrá (2x1) resultados posibles

2. Si un operador es **binario** (se aplica a dos términos, como la conjunción ^ o la disyunción v), sólo habrá (2x2) resultados posibles

3. En lugar de aprender los algoritmos correspondientes, es más sencillo aprenderse de memoria los resultados (como en las tablas de multiplicar o dividir)

4. Las tablas lógicas equivalentes a las de multiplicar y dividir se llaman ***Tablas de Verdad***

Conjunción		
p^q	V	F
V	V	F
F	F	F

Disyunción		
p√q	V	F
V	V	V
F	V	F

Negación	
~p	
V	F
F	V

¿Cómo se leen? Te ejemplifico con la tabla de la conjunción:

Sean dos proposiciones p y q.

Si p es V y q es V la conjunción es V

Si p es V y q es F la conjunción es F

Si p es F y q es V la conjunción es F

Si p es F y q es F la conjunción es F

Ni siquiera precisamos recordar estas tablitas. Basta con saber lo siguiente:

✔ La **Negación** de una expresión siempre da el valor complementario

✔ Una **Conjunción** es **V**, sólo si las dos expresiones son **V**

✔ Una **Disyunción** es **F**, sólo si las dos expresiones son **F**

Resta decir que 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 **¡Y con esto ya estamos listos para operar en Lógica!**

Repasemos:

Juan Semiamargo vive en Chocolatolandia y quiere realizar un viaje a México para aprender nuevas recetas con chocolate. Pero no tiene suficientes ahorros. Así que planea hacer una visita a su Banco para solicitar un préstamo personal.
Pide una entrevista con su agente bancario y le explica la situación. El agente cordialmente lo pone al tanto de la condición del banco para otorgar un crédito personal:
*Debe tener un ingreso mensual en su cuenta no menor a 10 monedas de chocolate **Y** debe tener un saldo promedio en ella en los últimos 3 meses de 5 monedas de chocolate*
De vuelta en su casa, Juan evalúa tranquilo la situación y se alegra al pensar que es su cuenta sueldo y mensualmente le pagan allí 15 monedas de chocolate, así que respira aliviado. Sin embargo al revisar sus resúmenes de los últimos tres meses descubre que en uno retiró todas sus monedas para comprar una entrada a un recital. Y ahora no le da el promedio!
*¿Le dará el Banco el préstamo a Juan? La respuesta es categórica: **NO***
*Ya que el Banco fue muy claro y para considerar a Juan como calificable requería el cumplimiento de las **2** condiciones simples.*
Grrrr! El rock cuesta caro!
Sin embargo decide probar suerte con la competencia y solicita información para un préstamo, a pesar de no tener una cuenta allí.
La condición de la nueva entidad bancaria es diferente:
*Debe abrir una cuenta y depositar 100 monedas de chocolate por el término de 6 meses **O** registrar un ingreso mensual no menor a 15 monedas, aunque sea en otro Banco*
Claramente Juan no desea abrir una nueva cuenta, y menos dejar allí 100 monedas que no dispone (además viene otra banda en el verano) Pero como su ingreso es exactamente de 15 monedas por mes y el Banco le dio opciones, es calificable para solicitar el crédito y volar feliz al país de los jalapeños!

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**.

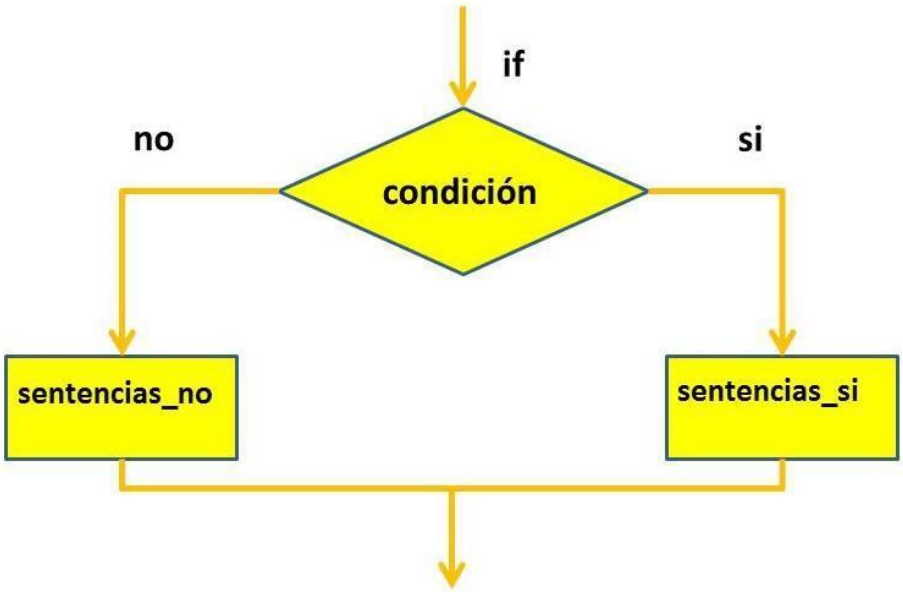
Una **Sentencia Condicional** tiene dos componentes o partes. Una, que permite al programa formularse y responderse una pregunta (simple o compleja), denominada ***condición;*** de allí el nombre de **Sentencia Condicional**. La otra, le permite al programa tomar sólo uno de dos (o más) posibles cursos de acción.

Por esta razón las **Sentencias Condicionales** pertenecen a una larga lista de tipos de sentencias que permiten alterar el **Flujo de Control Normal** (**FCP**) de un programa, llamadas también **Estructuras de Control**.

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.

Intentaremos comprender cómo funciona usando gráficos de **Diagramación Lógica** (un lenguaje gráfico de programación que sirve para aprender a programar genéricamente).



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 sea, si la respuesta a la pregunta es **SI**), se ejecutarán las sentencias asociadas a esa elección y **no** a la salida por **F** (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** se ejecutarán otras sentencias, las que están asociadas a esa respuesta (en la gráfica, la rama de la izquierda).

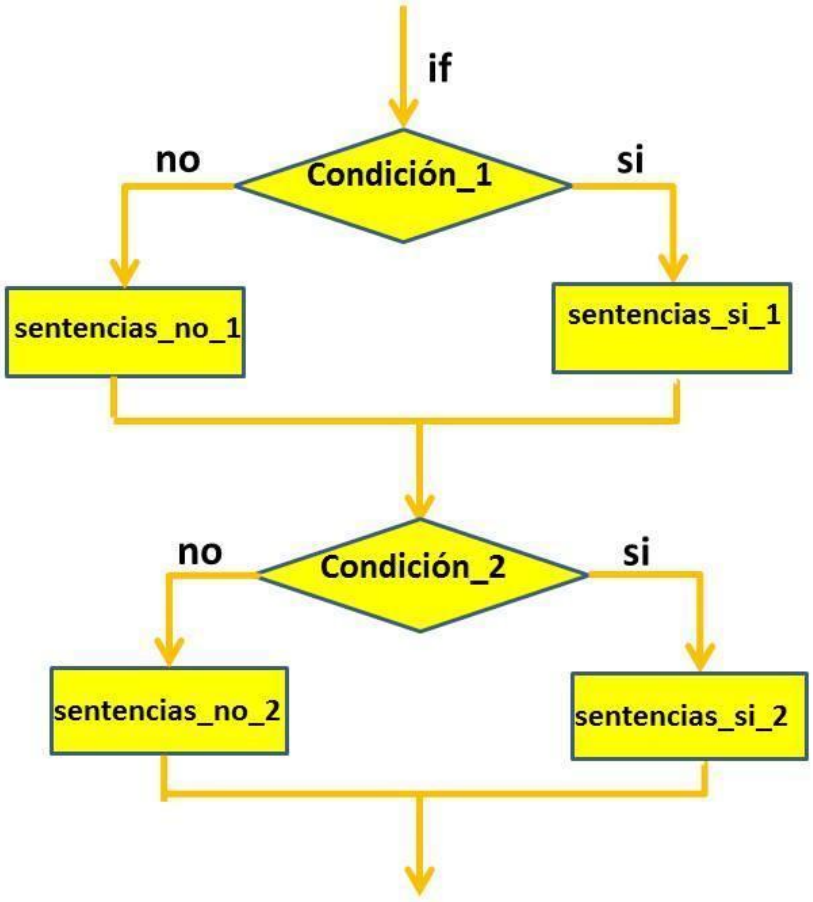
Es decir, el **FCP** llega a la **condición**, la evalúa y luego **bifurca**, por derecha o izquierda. Hay un grupo de sentencias que, en ese caso (debido a la respuesta obtenida), no se ejecutarán.

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).

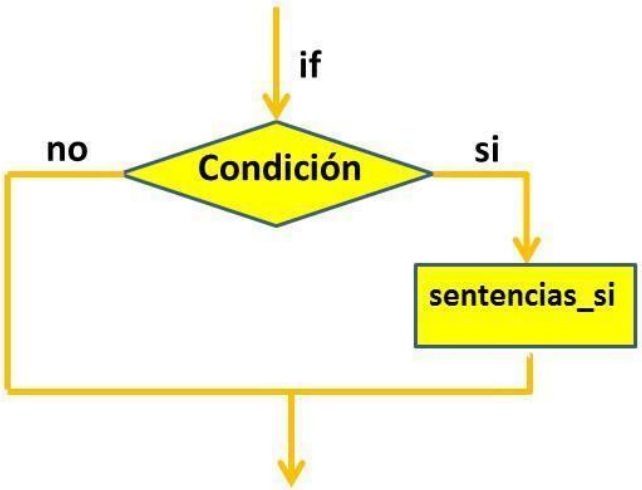
Dicho en otros términos, cuando la sentencia **if** finaliza, no importa si alcanzamos ese punto por la rama del V o del F, en cualquiera de los dos casos se ejecutarán las sentencias a continuación de ella.

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 **FCP** tiene amnesia regular, sólo sabe cuál es la sentencia que está ejecutando en ese momento y cuál será la próxima.

El siguiente esquema muestra cómo sería un Diagrama Lógico para dos **if** continuos:

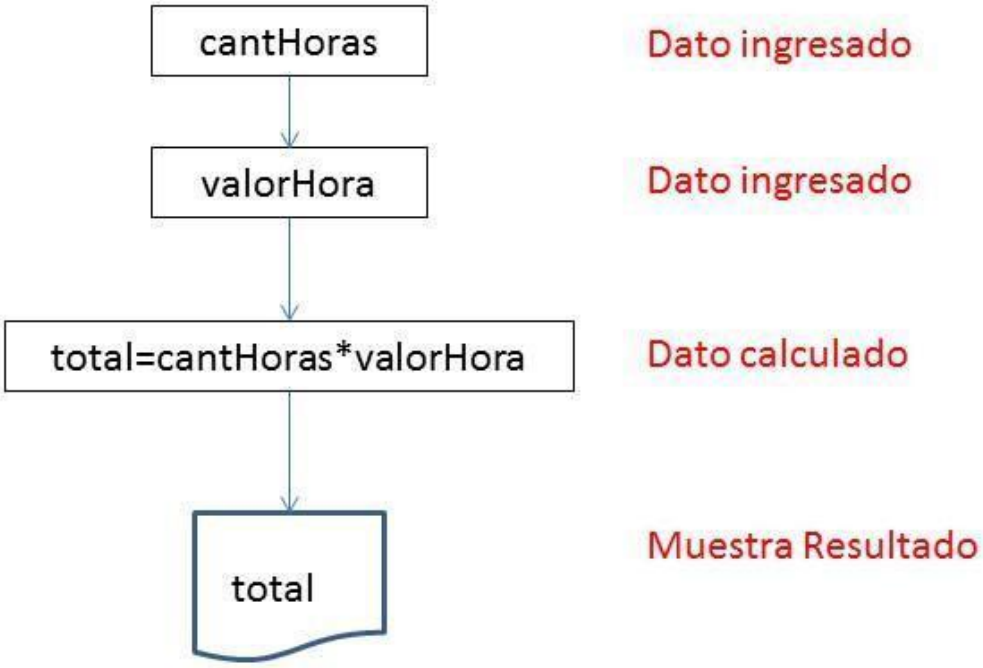


Es muy frecuente también escribir **if** que no hace nada por la salida del **F**, sólo por **V**. Esto es para cuando hay que sumar alguna acción a lo general para ciertos casos. La gráfica sería algo así:



Veamos un ejemplo:

Debemos calcular el pago a un trabajador. El cálculo debe hacerse por la cantidad de horas trabajadas. Por supuesto, necesitaremos que nos ingresen el valor por hora y la cantidad de horas trabajadas. Igual que en los gráficos anteriores, las flechas señalan los **FCP** posibles.



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.

Claramente ahora no tenemos un solo modelo o caso de liquidación, sino varios:

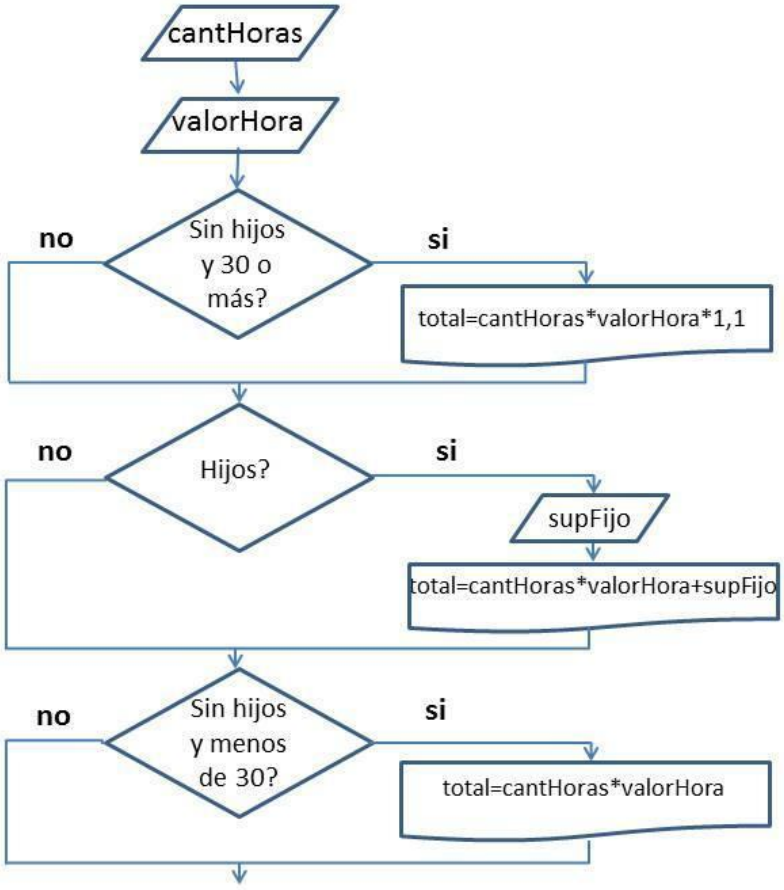
Trabajador con menos de 30 horas y sin hijos	total=cantHoras*valorHora
Trabajador con 30 horas o más y sin hijos	total=cantHoras*valorHora*1.1
Trabajador con menos de 30 horas y con hijos	total=cantHoras*valorHora+plusFijo
Trabajador con 30 horas o más y con hijos	total=cantHoras*valorHora+plusFijo

Observemos que los dos últimos casos reciben el **mismo tratamiento**, por lo tanto no son casos diferentes para nosotros y no es necesario distinguirlos.

¡Cuidado!

Los programas no son chismosos, sólo preguntan lo que es relevante para identificar situaciones que conduzcan a acciones diferenciadas

Veamos una solución para este problema:



En este caso, como hay varios **if** independientes y contiguos, cuando el **FCP** arriba al tercer **if** ignora por cuál rama del anterior pasó; de modo que no puede saber si la persona tenía hijos o no y debe repreguntar para asegurarse de no pagar de más.

Sentencia if

```
if  condición:
        -
        -
        -
[elif condición:
        -
        -
        -]k
[else:
        -
        -
```

```
    -]
```

Notas:

1. Lo que está entre [ ] es opcional

*2. 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. En términos generales, a la primera sentencia que encuentra con una indentación alineada con el Encabezado (o aún más a la izquierda que éste) decide que el Cuerpo terminó y esa es la sentencia siguiente a la Sentencia Estructurada previa*

3. 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

4. ¿Qué puede conformar un cuerpo en una Sentencia Estructurada? Cualquier sentencia o invocación a función válida de Python, incluso otra Sentencia Estructurada

Los operadores lógicos que usaremos en PYTHON son:

(^) conjunción: **and**

(v) disyunción: **or**

(~) negación:: **not**

¿Un programa puede preguntarse cualquier cosa? Esta pregunta tiene una respuesta ambivalente. Puede preguntarse lo que quiera. Pero sólo pocas respuestas se obtendrán de manera directa.

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.

Ojo:

No debería preguntarse lo que ya se sabe…

¿Y qué puede preguntarse PYTHON en forma directa?

Básicamente puede asociarle un valor de verdad a la evaluación de cualquier expresión aritmética (normalmente F si es 0, V si es ≠0), puede comparar valores de datos usando los **operadores de comparación**, o averiguar si un valor está dentro de un grupo, **operadores de pertenencia**.

Operadores de Comparación o Desigualdad:

- **==** igual

- **!=** distinto
- **<** menor
- **<=** menor o igual
- **>** mayor
- **>=** mayor o igual

Operadores de Pertenencia:

- **in** pertenece
- **not in** no pertenece

Estos operadores permiten formular condiciones simples tales como preguntar:

`a==2`	si el contenido de la variable **a** es **2**
`b+4>=c`	si el resultado de sumarle **4** al valor de **b** es **mayor o igual** que el valor de **c**

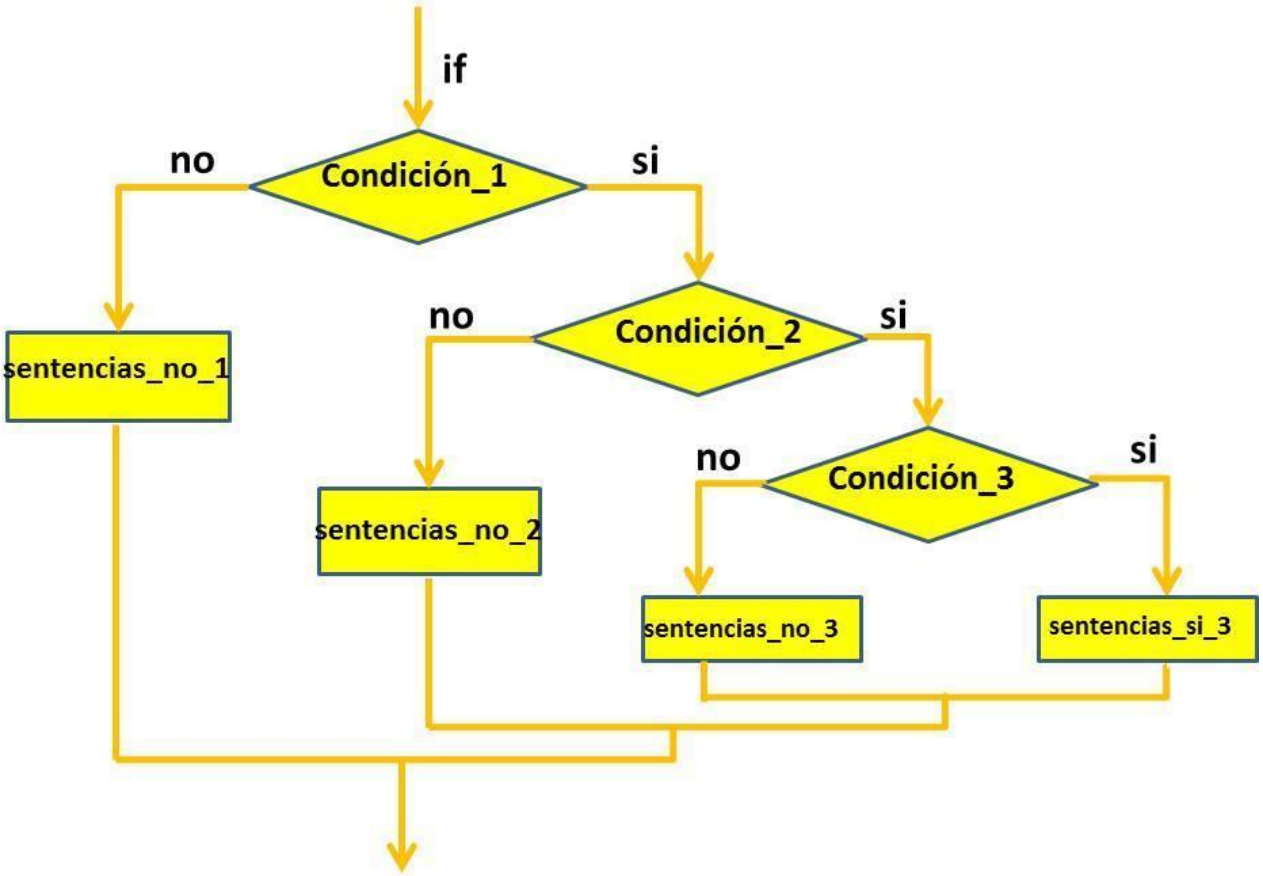
Con estos pocos elementos podemos arreglarnos para hacer una colección de preguntas simples o sofisticadas, que en su conjunto nos permitan identificar cualquier situación o patrón que busquemos.

¿Cómo escribiríamos el programa de cálculo del pago por horas que graficamos antes en PYTHON?

```
#Ejemplo de Cálculo de Pago por horas

cantHoras=int(input('Ingrese la cantidad de horas trabajadas: '))
valorHora=float(input('Ingrese valor de la hora de trabajo: '))
hijos=input('Tiene hijos? (si/no): ')
if (hijos=='no')and(cantHoras>=30):
      print('Debe cobrar %.2f'%(cantHoras*valorHora*1.1))
if hijos=='si':
      plusFijo=float(input('Ingrese adicional guardería: '))
      print('Debe cobrar %.2f'%(cantHoras*valorHora+plusFijo))
if (hijos!= 'si')and(cantHoras<30):
      print('Debe cobrar %.2f'%(cantHoras*valorHora))
```

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**. Dependiendo de los condicionales que estén dentro de las ramas de un **if** podemos tener diferentes esquemas. Cuando dentro de una rama de **if** aparece otro **if** decimos que los **if** están **anidados**.

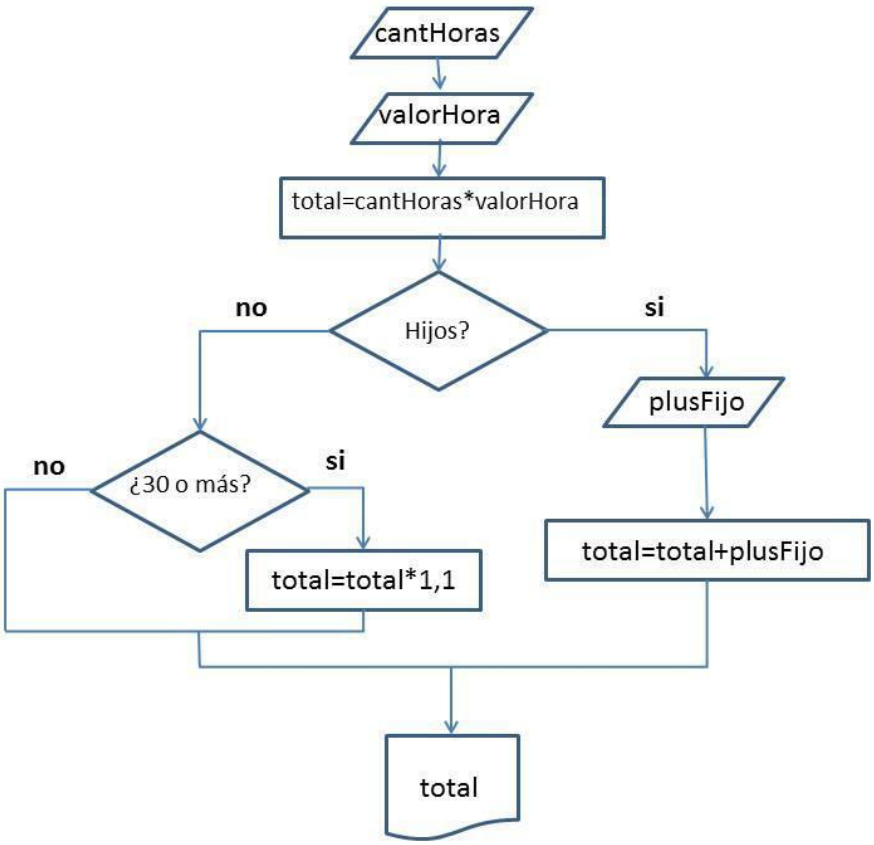


En la gráfica de arriba observamos que hay un **if anidado** dentro del mayor o más externo por la rama del **si**, y a su vez, dentro de esa rama hay otro **if anidado**, también en la rama del **si**. Las combinaciones posibles son infinitas.

Esta capacidad de PYTHON (y en general de cualquier Lenguaje y sus Condicionales) nos permite organizar nuestras preguntas (**condiciones**), en algunos casos, de manera más eficiente.

Revisemos el ejemplo del cálculo de pago que vimos antes:

¿Cómo sería la organización gráfica de nuestra versión usando la ventaja de **if anidados**?



Este no es el único modelo posible. Existen infinitas maneras de resolver esta situación. En la opción que graficamos 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!

También elegimos ir constituyendo el total por partes, calculando los importes comunes en una sola posición del diagrama. Podríamos haber elegido hacer el cálculo completo y específico en cada rama y también estaría bien.

Es posible resolver el problema anidando **if,** o no. Sin embargo, en casos como este la anidación permite condiciones más simples y una lógica de organización también más sencilla.

¿Cómo escribiríamos el programa de cálculo del pago por horas que graficamos antes en PYTHON?

```
#Ejemplo de Cálculo de Pago por horas com if anidados

cantHoras=int(input('Ingrese la cantidad de horas trabajadas: '))
valorHora=float(input('Ingrese valor de la hora de trabajo: '))
hijos=input('Tiene hijos? (si/no): ')
total=cantHoras*valorHora
if hijos=='si':
     plusFijo=float(input('Ingrese adicional fijo por hijos: '))
     total=total+plusFijo
else:
if cantHoras>29:
          total=total*1.1
print('Debe cobrar %.2f'%total)
exit()
```

Nota:

Prestá atención a cómo se van indentando las sentencias si se anidan if

Para evitar que el programa se desplace mucho a la derecha por indentación y le resulte incómodo visualmente al programador, Python incorpora una cláusula dentro del **if** bastante popular. De hecho, muchos lenguajes la disponen. Ya sabemos: El pragmatismo de Python…

¿Cómo podemos escribir el programa anterior utilizando esta cláusula, **elif**, que se emplea **sólo cuando por un else hay una única sentencia, y esta es un if**?

```
#Ejemplo de Cálculo de Pago por horas com if anidados

cantHoras=int(input('Ingrese la cantidad de horas trabajadas: '))
valorHora=float(input('Ingrese valor de la hora de trabajo: '))
hijos=input('Tiene hijos? (si/no): ')
total=cantHoras*valorHora
if hijos=='si':
     plusFijo=float(input('Ingrese adicional fijo por hijos: '))
     total=total+plusFijo
elif cantHoras>29:
     total=total*1.1
print('Debe cobrar %.2f'%total)
exit()
```

En el próximo apunte completaremos las Estructuras de Control básicas con Ciclos y Rangos.

Bibliografía Adicional

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