1. Proposiciones lógicas

Las **proposiciones** (enunciados) son oraciones declarativas al las que se le pueden determinar su *valor de verdad*. Es una expresión lingüística del razonamiento, que se caracteriza por ser verdadera o falsa empíricamente, sin ambigüedades. Son proposiciones las oraciones aseverativas, las leyes científicas, las fórmulas matemáticas, las fórmulas y/o esquemas lógicos, los enunciados cerrados o claramente definidos.

El **valor de verdad** de una proposición depende no solamente de las relaciones entre las palabras del lenguaje y los objetos en el mundo, sino también del *estado del mundo* y del conocimiento acerca de ese estado. El valor de verdad de la oración *"Juan canta"* depende no solamente de la persona denotada en *Juan* y el significado del verbo *cantar*, sino también *del momento* cuando esta oración es expresada. Juan probablemente canta ahora, pero ciertamente que no siempre está cantando.

Se debe hacer una distinción entre la oración gramatical propiamente dicha, a la que se llama enunciado, y el contenido o significado del enunciado, que es la proposición.

Los siguientes *enunciados* representan en realidad a **la misma proposición**:

- En Buenos Aires hay mucha humedad.
- Buenos Aires es una ciudad muy húmeda.
- La humedad en Buenos Aires es bastante alta.

Ejemplos de expresiones las cuales **no son** proposiciones:

- El hombre más fuerte del mundo.
- ¿Quién ganará el Mundial de Pelota Panamá 2011?
- 13 + 7
- ¡Hable en voz baja!

1.1. Proposiciones atómicas y compuestas

Las **proposiciones atómicas** (o simples) son las que carecen de conectores lógicos, es decir que sólo se evalúa un valor de verdad. Por ejemplo *"Matías es profesor"* es una proposición simple la cual puede ser verdadera o falsa. En cambio las **proposiciones compuestas** (o moleculares) son aquellas que tienen dos o más proposiciones atómicas vinculadas entre sí por conectores. Por ejemplo, *"Matías es profesor* ***Y*** *Matías dicta el curso de programación"* es una proposición compuesta entre las proposiciones simples *"Matías es profesor"* y *"Matías dicta el curso de programación"* conectadas entre sí por una **conjunción**. Por lo tanto, **para que la proposición molecular sea verdadera, ambas deben serlo**.

A medida que avancemos en la construcción de algoritmos con sentencias de control veremos cómo aplicar estos conectores.

2. Sentencias y operaciones lógicas

Sabemos que un algoritmo es un modelo de resolución para un problema, el cual estaba caracterizado por ser secuencial, ordenado, preciso, definido y finito. Es decir que es una lista ordenada de instrucciones o pasos que debemos seguir para llegar a la solución del problema. Pero hay situaciones en las que debemos tomar decisiones, las cuales pueden llevarnos a tomar un camino u otro al momento de implementar la solución.

Las **sentencias de control**, son *instrucciones* que permiten romper la secuencialidad de la ejecución, esto significa que nos permiten la realización de algunas instrucciones y omitir otras, de acuerdo a la evaluación de una condición o expresión lógica. Por otra parte, hay sentencias que nos permiten volver a ejecutar un conjunto de instrucciones dada una condición.

Resumiendo, existen dos tipos de *sentencias de control*:

1. **Selectivas:** un camino entre dos o más opciones se ejecuta "por única vez".
2. **Repetitivas:** permiten ejecutar un conjunto de instrucciones "varias veces" dada una condición.

2.1. Implementación en el lenguaje `Python`

2.1.1. Variable booleanas

En `Python` existe un tipo de dato denominado `bool` el cual solo puede tener uno de dos valores: `True` o `False`. Un ejemplo de declaración y asignación de dos variables de este tipo se vería así:

```
es_lunes = True
es_martes = False
```

2.1.2. Operadores de comparación

Existen operadores que permiten comparar dos valores, que usualmente conocemos de la matemática. Por ejemplo: $a > b$, $a \leq b$, $a = b$, $a \neq b$, etc.

Operadores de comparación en `Python`

- `==` : igual
- `>` : mayor
- `<` : menor
- `!=` : distinto
- `>=` : mayor igual
- `<=` : menor igual

Veamos aplicaciones concretas sobre la aplicación de estos operadores:

```
a, b = 2, 3
es_a_mayor_b = (a > b)        # ¿a es mayor que b?
es_a_menorigual_b = (a <= b) # ¿a es menor o igual que b?
es_a_distinto_b = (a != b)   # ¿a es distinto que b?
```

2.1.3. Operadores lógicos

Los operadores lógicos son los ya mencionados **conectores**, qué sirven para poder programar *proposiciones compuestas*. Ya que por ejemplo es imposible hacer la siguiente expresión: "$0 < a < x$", ya que el operador de comparación < es un operador binario, es decir que tiene solo dos operandos.

Es por ello que disponemos de 3 operadores lógicos:

Operadores lógicos en Python

- `and` : conjunción (Y)
- `or` : disyunción (O)
- `not` : negación (NO)

Por lo tanto si se deseara averiguar si una variable está dentro de determinado rango:

```
esta_entre_0y10 = (a > 0 and a < 10) # True: si 0 < a < 10
```

Otros ejemplos serían:

```
es_0o10 = (a == 0 or a == 10)  # True: si a es 0 ó 10
b = True
no_b = not b                   # True: si b es False
```

3. Sentencias selectivas en Python

En este encuentro nos centraremos en las sentencias **selectivas** únicamente, que nos permitirán ejecutar porciones de código y omitir otras.

3.1. Condicional simple

La sentencia de control condicional *"Si"* es la más básica de todas. Se evalúa una **expresión lógica**. En caso verdadero, las instrucciones declaradas se ejecutarán, pero en caso contrario, estas serán omitidas y se continuarán con las siguientes a continuación del bloque. En `Python` se utiliza la palabra en ingles `if` que se traduce literalmente como *si*. Su implementación se puede observar en el Programa 1.

En el diagrama se detalla que *"Si n es igual a 2"* **entonces** se escribirá el texto *"Ingresó un 2."*. Lo cual se codifica en `Python` con un `if` que a continuación tiene la expresión lógica "`n == 2`" utilizando el operador correspondiente de **igualación**, el cual no debe confundirse con el de **asignación** que es un solo igual "=".

Inmediatamente después de la expresión a igualar se coloca '`:`', en la linea subsiguiente se escribirán las instrucciones que se ejecutarán en caso de que dicha expresión *sea verdadera* dejando una sangría conocida como *indentado*[1]. Aquí únicamente se mostrará el texto *"Ingresó un 2."*. En caso de que no se ingrese un 2 por la linea de comandos, entonces este texto no se verá. Finalmente, sin importar si el caso resultara verdadero o

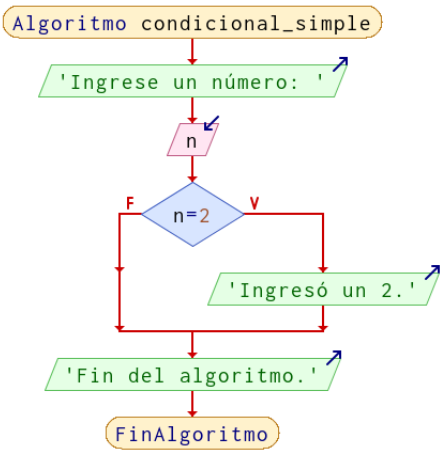
[1] **anglicismo** (de la palabra inglesa *indentation*) de uso común en informática.

Programa 1: Condicional simple

```
1 n=float(input("Ingrese un número: "))
2
3 if n == 2:
4     print("Ingresó un 2.")
5
6 print("Fin del algoritmo.")
```

```
Ingrese un número: 5
Fin del algoritmo.
```

```
Ingrese un número: 2
Ingresó un 2.
Fin del algoritmo.
```



falso se mostrará el texto *"Fin del algoritmo."*, ya que este se encuentra por fuera del bloque del **`if`**.

¡Cuidado!

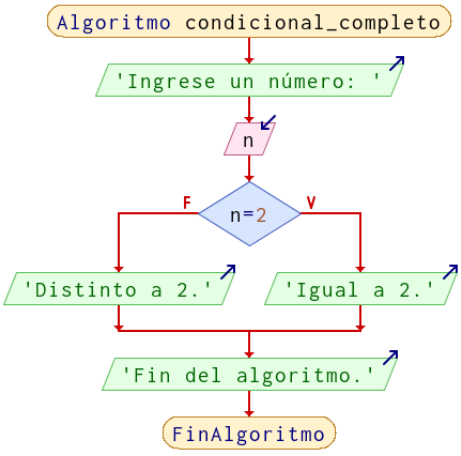
Es importante para el programador no descuidar la sintaxis del lenguaje, ser lo más claro y ordenado posible. En **`Python`** es obligatorio el espacio (*indentado*) para indicar que las instrucciones están dentro de un *"bloque"* o *"ámbito"* específico y al finalizar la sangría se abandona dicho bloque.

3.2. Condicional completo

Los condicionales completos son aquellos que poseen dos bloques de código, uno que se ejecuta en caso verdadero y otro que se ejecuta en el caso falso. El bloque ejecutado por el lado falso se indico con la palabra reservada **`else`** (*"sino"*):

Programa 2: Condicional completo

```
1 n=float(input("Ingrese un número: "))
2
3 if n == 2:
4     print("Igual a 2.")
5 else:
6     print("Distinto a 2.")
7
8 print("Fin del algoritmo.")
```



En esta variante obtendremos dos tipos de salida dependiendo del valor ingresado:

```
Ingese un número: 2
Igual a 2.
Fin del algoritmo.
```

```
Ingese un número: 3
Distinto a 2.
Fin del algoritmo.
```

Al finalizar el primer bloque y volver a la sangría original debemos anteponer la palabra reservada **`else`**, la cual abrirá un nuevo bloque indicado por el carácter '`:`' y la indentación a continuación.

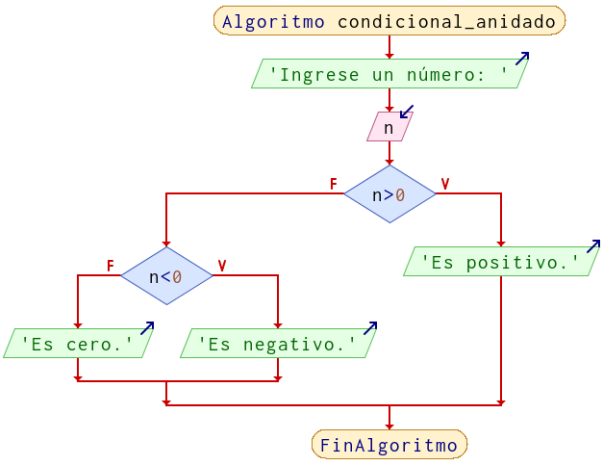
Los `else` no deben tener condiciones, es decir, no hay una condición como en el `if`, ya que justamente el bloque se ejecuta cuando esta primera es falsa. *Nunca habrá un* ***`else`*** *sin un* ***`if`*** *previo.*

3.3. Condicionales anidados

Muchas veces debemos corroborar algo que no puede resolverse con una sola expresión lógica. Por ejemplo, saber si un número es positivo, negativo o cero. Aquí tenemos 3 posibilidades. Para ello, podemos pasar por más de una instancia de evaluación **adentro** de otra. En este sencillo algoritmo observamos que luego de un `if` es posible colocar

Programa 3: Condicional anidado

```
1 n=float(
2     input("Ingrese un número: "))
3
4 if n > 0:
5     print("Es positivo.")
6 elif n < 0:
7     print("Es negativo.")
8 else:
9     print("Es cero.")
```



`elif` con una nueva condición. Esto viene de la contracción de `else-if` (sino-si). El nuevo condicional puede ser simple o completo dependiendo de la necesidad, es decir que el último `else` puede estar o no dependiendo de la necesidad del algoritmo.

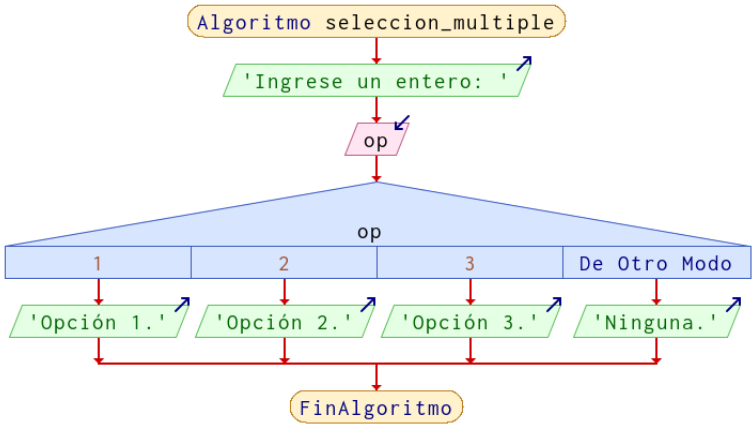
Es posible poner un nuevo `elif` todas las veces que se requieran:

```
if '''condición 1''':
  # ...
elif '''condición 2''':
  # ...
elif '''condición 3''':
  # ...
else: # Opcional...
  # ...
```

3.4. Selección múltiple

Frecuentemente nos encontraremos con casos en los cuales debemos tomar un camino dependiendo de un valor fijo. El caso más típico es al crear un *menú de opciones*. En el cual debemos seleccionar un valor predeterminado y preestablecido. Para ello podremos evaluar una ***variable*** y ejecutar el bloque de código correspondiente al valor de la misma.

Como vimos anteriormente, esto se puede lograr concatenando una serie de `if-else-if`, pero esto no se estila porque existe la sentencia de control de selección `match-case`.



Programa 4: Selección múltiple

```
1  op=int(input("Ingrese el entero 1, 2 ó 3: "))
2
3  match op:
4      case 1:          # Caso op == 1
5        print("Opción 1.")
6        # FinCaso
7
8      case 2:          # Caso op == 2
9        print("Opción 2.")
10       # FinCaso
11
12     case 3:          # Caso op == 3
13       print("Opción 3.")
14       # FinCaso
15
16     case _:          # op != 1 and op != 2 and op != 3
17       print("Ninguna.")
18 # FinMatch
```

La sentencia empieza con la palabra reservada `match` que evaluará una variable. Inmediatamente después dentro del bloque se evaluarán los `case` compuestos por un valor constante (*literal*: 1, 2, 3, etc.) seguido de '`:`'.

En cada bloque `case` se incluirán las instrucciones a ejecutar cuando la variable sea igual al cual se está comparando y terminará al volver una sangría.

Por último, hay un bloque opcional, el cual puede estar o no (como el último `else` en

un bloque de condicionales) que es el '`case _`' (guion bajo), el cual se ejecuta cuando la evaluación no coincide con ninguno de los *casos* anteriores.

4. Ejercitación

Intente resolver los siguientes problemas planteando los algoritmos en diagramas y codifíquelos en `Python`:

1. Pida ingresar un número, indique si es par o impar.
2. Ingrese 2 números, muestre la distancia entre ambos.
3. Pida ingresar una temperatura en grados centígrados, muéstrelos en fahrenheit. Si supera o iguala los 113°F muestre la cadena "Alerta roja por altas temperaturas".
4. Pida ingresar un número y muestre un texto que diga si el número es positivo, negativo o cero.
5. Pida ingresar 2 nombres, muéstrelos ordenados alfabeticamente (de forma ascendente: A→Z).
6. Pida ingresar un ángulo y verifique que esté entre 0° y 180°, muestre el texto "ángulo válido" en caso afirmativo y "ángulo inválido" de lo contrario.
7. Pida ingresar un ángulo entre 0° y 180°, determine y muestre si es: "nulo", "agudo", "recto", "obtuso", "llano" o "no válido".
8. Pida ingresar 2 ángulos internos de un triangulo, muestre el que falta e indique si es un triangulo equilátero.
9. Extienda el algoritmo del ejercicio anterior para que indique si el triangulo es rectángulo.
10. Pida ingresar 2 números y determine cuál es el mayor.
11. Pida ingresar 3 números y determine cuál es el menor.
12. Pida ingresar 3 nombres, muéstrelos ordenados de manera descendente (Z→A).
13. Muestre 3 opciones:

    1 - Doble

    2 - Triple

    3 - Mitad

    Pida ingresar una de las opciones, luego un número y muestre el doble, triple o la mitad del mismo según corresponda. Si no se ingresó un opción válida, muestre el mensaje de error: *"Opción no válida"*.