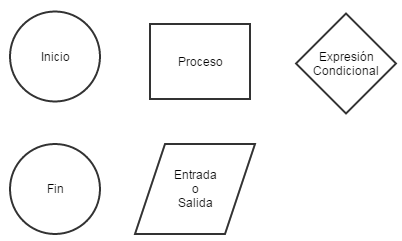
# Simbología de diagramas de flujo

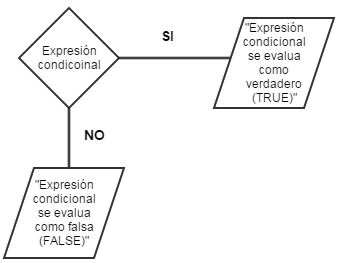


# Expresiones condicionales y Decisiones

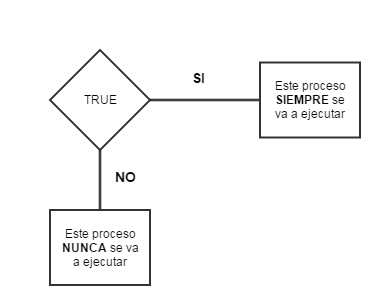
Las **expresiones condicionales** (ejemplo: **10 > 2**) son expresiones que se evalúan para dar un resultado **booleano** (verdadero o falso, TRUE o FALSE). Este **resultado** puede ser almacenado una variable de tipo **Boolean**. Nótese que el **resultado** es lo que se almacena, no la expresión condicional como tal, en otras palabras es necesario evaluar la condición periódicamente (como en un ciclo) para saber si el **resultado** que está almacenado en la variable ha cambiado.

Las **expresiones condicionales** tiene un amplio uso, uno de ellos es para cambiar el flujo de la lógica de un programa, esto se logra haciendo uso de **decisiones**.

Las **decisiones** (bloques **if** **else** en lenguaje de programación) se expresan en diagramas de flujo por medio del **rombo**. Tiene dos extremos por donde continúa la lógica dependiendo del resultado al evaluar la **expresión condicional**:



Ahora **unamos** los conceptos anteriormente explicados para profundizar y entender mejor como cada uno de estos conceptos puede ser utilizado como bloques de construcción para nuestros programas. Ya que los bloques de **decisión** evalúan **expresiones condicionales** para utilizar un resultado **booleano** a la hora de dirigir el flujo de la lógica, es posible “ahorrarle” el esfuerzo al bloque de **decisión** y darle la “respuesta” (el verdadero o falso) en vez de darle una **expresión condicional** para evaluar.



El ejemplo de la imagen puede parecer un poco tonto ya que hay un proceso, o un **flujo de lógica** el cual nunca va a ser ejecutado, pero nos ayuda a entender que los bloques de condición no sólo reciben **expresiones condicionales**, sino también valores booleanos que pueden ser almacenados en **variables booleanas**.

Cuando veamos operadores lógicos (para concatenar expresiones condicionales), es posible tener expresiones condicionales muy largas, al punto que ya uno no está seguro qué se desea evaluar. Es por esto que se acostumbra almacenar estos resultados en variables booleanas, ya que se le puede nombrar la variable, haciendo la lógica más legible y explícita. Estas variables pueden luego ser utilizado en bloques de decisión y en ciclos (ya que los ciclos, como vamos a ver, hacen uso de bloques de decisión).

Es importante tener conciencia de que las **decisiones** evalúan **expresiones condicionales**, pero estos dos conceptos son distintos el uno del otro a pesar de que se complementan muy bien.

# 

# Ciclos

Los ciclos permiten **repetir** una sección de lógica hasta que “ya no sea necesario”. Esta descripción se puede ver un poco escueta, pero verdaderamente es lo único que el ciclo hace.

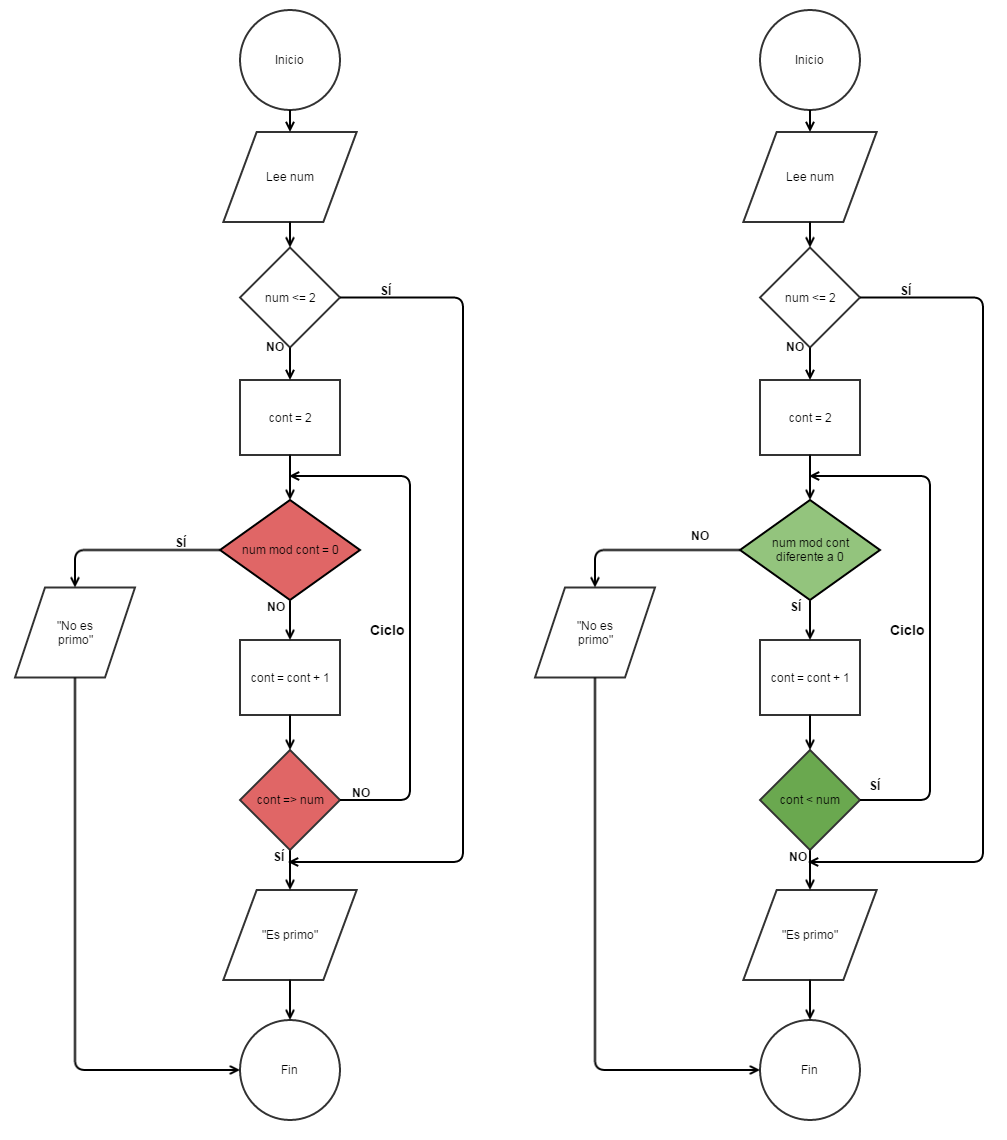
Los ciclos se aprovechan de la herramienta de **decisión** anteriormente explicada para definir el: hasta que “ya no sea necesario”, o en otras palabras el fin del ciclo.

Para finalizar el ciclo, es necesario **evaluar** por medio de una **decisión** que la **condición de parada** del ciclo sea **verdadera** o la **condición de ciclo** deje de cumplirse (sea **falsa**).

La **condición de ciclo** normalmente se lee “mientras que **expresión condicional** haga el ciclo”.

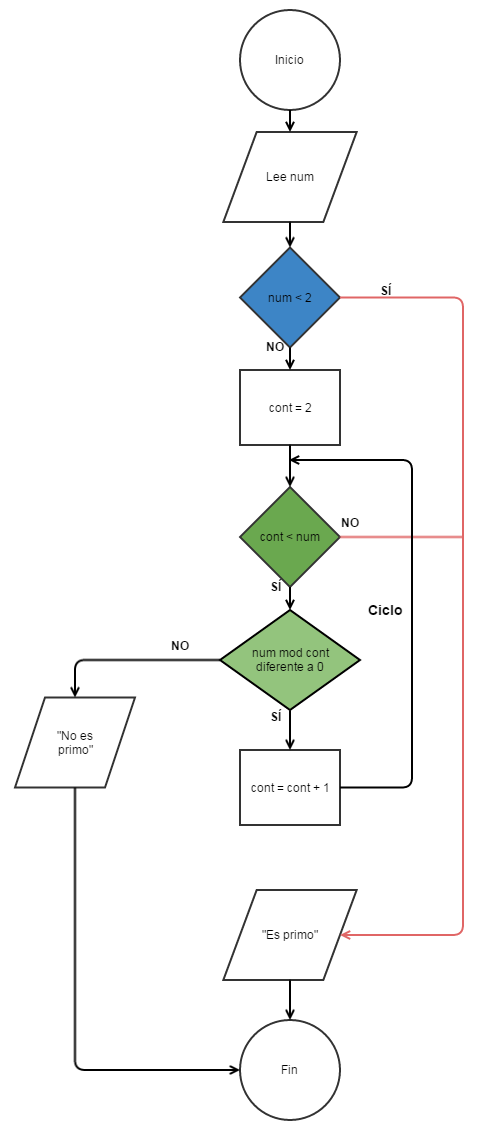
Nótese que la **condición de ciclo** y **condición de parada** son conceptualmente opuestas, por lo que invertir la lógica normalmente es suficiente para cambiar una **condición de ciclo** a una **condición de parada** y viceversa.

Veamos el ejemplo del programa que identifica números primos el cual es un poco especial ya que tiene 2 condiciones para definir si el ciclo continúa. Nota: el programa no está tomando en cuenta manejar excepciones como num < 1:



En rojo vemos las **condiciones de parada** del ciclo, mientras que en verde estas mismas condiciones fueron cambiadas a **condiciones de ciclo** y sus salidas invertidas ya que la lógica se invirtió.

Dependiendo de si es necesario o no evaluar la **condición de ciclo** antes de empezar el proceso, nos ayuda a identificar la ubicación del bloque de **decisión**. Normalmente uno se da cuenta de esto en la etapa de **verificación del programa** (correr el programa en papel, anotando cómo cambia el valor de las variables). En el ejemplo anterior, se cambiaron las **condiciones de salida** por **condiciones de ciclo**, pero ya que la lógica cambió, se puede aprovechar mejor la condición de ciclo y variar un poco la lógica sin cambiar el resultado del programa:

En este caso la **condición de ciclo** es **cont < num**, y mientras esta condición se cumpla el ciclo va a seguir corriendo. Ya que se tiene esta condición, se cambió la **expresión condicional** de la **decisión** en **azul**. Ahora no es necesario revisar si num es igual a 2 ya que esta revisión se hace en el encabezado del ciclo (en verde oscuro), donde se pregunta si **cont < num**. En la primar iteración **cont** es igual a 2, entonces 2 < 2 evalúa a **falso**, si se sigue la lógica en rojo se devuelve el mensaje “Es primo” y termina el program. Este es el mismo comportamiento que se tenía antes, pero la lógica cambió un poco al hacer uso del **condicional de ciclo**.