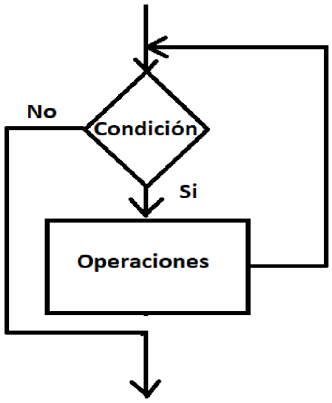
ESTRUCTURAS REPETITIVAS
Instrucción while

A diferencia de *for*, cuando en un programa se requiere repetir una serie de operaciones donde se desconoce el número de veces, pero el control del ciclo depende de una condición, la instrucción *while* es la más indicada para realizar tal bucle. Su diagrama de flujo puede representarse de la siguiente forma:



Normalmente se recurre a una variable de control que permita valorar la condición, para que el circuito se realice mientras la condición es verdadera (*True*). Esta estructura también recibe el nombre de circuito *while*, y se debe ser muy cuidadoso en la definición de la condición (o condiciones) de la variable de control para que se pueda salir del circuito, porque se corre el riesgo de quedar en un circuito infinito o generar un desbordamiento (*inf*).

Esta estructura es muy utilizada para repetir procesos o partes de un programa hasta que se oprima determinada tecla o se establezca cierta condición. Por ejemplo, el programa *EcuacionCuadratica.py* puede modificarse para que se repita hasta que se oprima una tecla diferente de *"S"*, de la forma que se muestra en la figura 12.

```
"""Programa para calcular las raíces de la ecuación cuadrática
Ax2+Bx+C=0 usando las expresiones de la fórmula general
x1=(-B+sqrt(B**2-4*A*C)/(2*A) y x2=(-B-sqrt(B**2-A*C)/(2*A)
y matemática compleja"""
import math as m
import cmath as cm
tecla="S"
while tecla=="S" or tecla=="s":
    print("Introducir los coeficientes de la ecuación cuadrática")
    A=float(input("Introducir coeficiente A: "))
    B=float(input("Introducir coeficiente B: "))
    C=float(input("Introducir coeficiente C: "))
    D=B**2-4*A*C
    if D>=0:
        x1=(-B+m.sqrt(D))/(2*A)
        x2=(-B-m.sqrt(D))/(2*A)
        print("Raíz x1= {:8.4f}".format(x1))
        print("Raíz x2= {:8.4f}".format(x2))
    else:
        x1=(-B+cm.sqrt(D))/(2*A)
        x2=(-B-cm.sqrt(D))/(2*A)
        print("Raíz x1= {:8.4f}".format(x1))
        print("Raíz x2= {:8.4f}".format(x2))

    print()
    tecla=input("Otro cálculo (S/N): ")
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47)
1)] on win32
Type "copyright", "credits" or "license()" for more inf
>>>
 RESTART: G:/PythonParaIngenierosCiviles/ProgramasPytho
.py
Introducir los coeficientes de la ecuación cuadrática
Introducir coeficiente A: 3
Introducir coeficiente B: 5
Introducir coeficiente C: 4
Raíz x1= -0.8333+0.7993j
Raíz x2= -0.8333-0.7993j

Otro cálculo (S/N): s
Introducir los coeficientes de la ecuación cuadrática
Introducir coeficiente A: 3
Introducir coeficiente B: -4
Introducir coeficiente C: 2
Raíz x1= 0.6667+0.4714j
Raíz x2= 0.6667-0.4714j

Otro cálculo (S/N): s
Introducir los coeficientes de la ecuación cuadrática
Introducir coeficiente A: 47
Introducir coeficiente B: 6
Introducir coeficiente C: -9
Raíz x1=   0.3784
Raíz x2=  -0.5061

Otro cálculo (S/N): n
>>>
```

Figura 12. Programa EcuacionCuadraticaWhile.py

En el programa anterior, la variable de control es *tecla*, es de tipo *string* y se introduce desde el teclado. La condición de repetición es *tecla=="S" or tecla=="s"*. Cuando se oprima una tecla diferente, el circuito termina.

En ciertos problemas de métodos numéricos, se requiere determinar la raíz de una función no polinómica, y el proceso debe repetirse hasta que se cumpla cierta condición de evaluación de la función, por ejemplo, que se aproxime a un valor cercano a cero establecido por un número muy pequeño. El método de la secante permite obtener mediante aproximaciones sucesivas a partir de la evaluación de la recta secante que intercepta dos puntos sobre la curva, acercándose a la solución

mediante un proceso iterativo. El pseudo programa en *Mathcad 14* mostrado en la figura 13 permite visualizar el resultado la raíz de la funcion y(x)=exp(-x)-π, cuya gráfica se muestra ahí mismo.

$$\text{Secante}(f, tol, x_0, x_1) := \left| \begin{array}{l} i \leftarrow 1 \\ \text{while } |f(x_0)| > tol \\ \quad \left| \begin{array}{l} x_2 \leftarrow x_1 - \dfrac{x_1 - x_0}{f(x_1) - f(x_0)} \cdot f(x_1) \\ X_i \leftarrow x_2 \\ x_0 \leftarrow x_1 \\ x_1 \leftarrow x_2 \\ i \leftarrow i + 1 \end{array} \right. \\ X_{i-1} \end{array} \right.$$

$$f(x) := \exp(-x) - \pi \qquad tol := 0.00001$$

$$x_0 := -3$$

$$x_1 := 0$$

$$\text{Secante}(f, tol, x_0, x_1) = -1.145$$

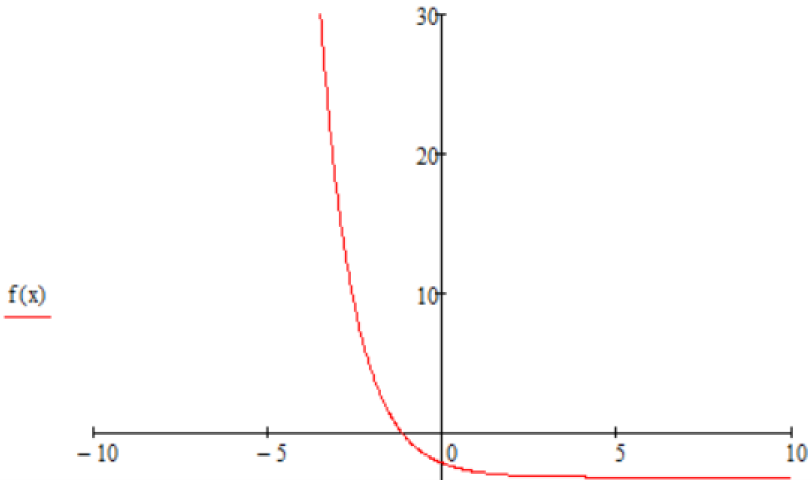


Figura 13. Pseudo programa Secante.xmcd

El programa en *Python* y su corrida se muestran en la figura 14.

```
"""Programa del método de la secante"""
import math as m
tol=1e-05
x0=-3
x1=0
fx0=m.exp(-x0)-m.pi
i=1
while abs(fx0)>tol:
    fx1=m.exp(-x1)-m.pi
    x2=x1-(x1-x0)/(fx1-fx0)*fx1
    print(i,"++",x0,"++",fx0,"++",x1,"++",fx1,"++",x2,"++",abs(fx0))
    Xi=x2
    x0=x1
    x1=x2
    fx0=m.exp(-x0)-m.pi
    i=i+1
print("La raíz es: ",Xi)
```

```
== RESTART: G:/PythonParaIngenierosCiviles/ProgramasPython/MetodoSecante.py ==
1 ++ -3 ++ 16.943944269597875 ++ 0 ++ -2.141592653589793 ++ -0.3366307160561827 ++ 16.943944269597875
2 ++ 0 ++ -2.141592653589793 ++ -0.3366307160561827 ++ -1.7413707647988947 ++ -1.8013154419328887 ++ 2.141592653589793
3 ++ -0.3366307160561827 ++ -1.7413707647988947 ++ -1.8013154419328887 ++ 2.9160180071899617 ++ -0.8842678763098257 ++ 1.7413707647988947
4 ++ -1.8013154419328887 ++ 2.9160180071899617 ++ -0.8842678763098257 ++ -0.7203815371706224 ++ -1.0659377194948605 ++ 2.9160180071899617
5 ++ -0.8842678763098257 ++ -0.7203815371706224 ++ -1.0659377194948605 ++ -0.2380322204848615 ++ -1.1555890876477097 ++ 0.7203815371706224
6 ++ -1.0659377194948605 ++ -0.2380322204848615 ++ -1.1555890876477097 ++ 0.03430109276633164 ++ -1.1442972634528379 ++ 0.2380322204848615
7 ++ -1.1555890876477097 ++ 0.03430109276633164 ++ -1.1442972634528379 ++ -0.0013508293916111382 ++ -1.1447275409653053 ++ 0.03430109276633164
8 ++ -1.1442972634528379 ++ -0.0013508293916111382 ++ -1.1447275409653053 ++ -7.366662008845282e-06 ++ -1.1447298863566615 ++ 0.0013508293916111382
La raiz es:  -1.1447298863566615
>>>
```

Figura 14. Programa MetodoSecante.py

Algunos ejemplos del uso de las dos instrucciones repetitivas se muestran en los siguientes programas.

El primero es para calcular el factorial de un número mediante la expresión:

$$n! = n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot(3)\cdot(2)\cdot(1)$$

$$5! = 5\cdot4\cdot3\cdot2\cdot1 = 120$$

$$10! = 3628800$$

$$0! = 1$$

```
"""Factorial de un número entero"""
tecla="S"
while tecla=="S" or tecla=="s":
    n=1.0
    while type(n)!=int:
        n=eval(input("Introduzca un número entero: "))
    f=1
    for i in range(1,n+1):
        f=f*i
    print("El factorial de ",n," es: ",f)
    print()
    tecla=input("Otro (S/N): ")
```

```
Introduzca un número entero: 5
El factorial de  5  es:  120

Otro (S/N): s
Introduzca un número entero: 10
El factorial de  10  es:  3628800

Otro (S/N): s
Introduzca un número entero: 5.3
Introduzca un número entero: 5.5
Introduzca un número entero: 6
El factorial de  6  es:  720

Otro (S/N): n
>>>
```

En este programa puede observarse el uso de la función *type()* para verificar el tipo de variable, el uso de *f* como variable para referenciar el producto de la variable iterativa *i*, *n* con un valor inicial *float* para servir como variable de control en el segundo circuito *while()* y la función *eval()* para evaluar el tipo de dato que se introduce desde el teclado con la instrucción *input()*.

Así como se dijo que para una sumatoria se requiere un acumulador, para una multiplicatoria se requiere una variable que registre el producto (multiplicador) en cada vuelta del circuito, como se observa en el calculo de la multiplicatoria de 1 a 10, su pseudo-programa en *Mathcad 14* y el programa en *Python*.

$$\prod_{k=1}^{10} k = 3628800$$

$$\begin{vmatrix} f \leftarrow 1 \\ \text{for } k \in 1..10 \\ \quad f \leftarrow f \cdot k \\ f \end{vmatrix} = 3628800$$

```
f=1
for k in range(1,n+1):
    f=f*k
    print(k,"---",f)
print("La multiplicatoria de ",1," a ",n," es ",f)
```

```
1 --- 1
2 --- 2
3 --- 6
4 --- 24
5 --- 120
6 --- 720
7 --- 5040
8 --- 40320
9 --- 362880
10 --- 3628800
La multiplicatoria de  1  a  10  es  3628800
```

Para calcular las cifras decimales del número *pi (π)*, existe una serie infinita propuesta por *James Gregory y Gottfried Leibniz* expresada por la siguiente sumatoria; en este ejemplo se calcula para un máximo de $k = 1e^6$:

$$4 \cdot \sum_{k=1}^{10^6} \frac{(-1)^{k+1}}{2 \cdot k - 1} = 3.1415916535897743$$

El programa en *Python* y su corrida se muestran a continuación:

```
"""Cálculo de pi con la serie de Gregory-Leibniz"""
import math as m
n=eval(input("Introduzca el valor máximo de la serie: "))
suma=0
for k in range(1,n+1):
    suma=suma+(-1)**(k+1)/(2*k-1)
Pi=4*suma
dif=Pi-m.pi
print("Número pi de Gregory-Leibniz: {:32.31f}".format(Pi))
print("Número pi de Python: {:32.31f}".format(m.pi))
print("Diferencia de cifras decimales: {:32.31f}".format(abs(dif)))
```

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (I
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: G:/PythonParaIngenierosCiviles/ProgramasPython/PiLeibnizGregory.py
Introduzca el valor máximo de la serie: 1000000
Número pi de Gregory-Leibniz: 3.1415916535897743244731827871874
Número pi de Python: 3.1415926535897931159979634685442
Diferencia de cifras decimales: 0.0000010000000187915247806813568
```