

# Estructuras de control

## Condicionales

*Ejemplo:* Resolución de la ecuación de primer grado

```
In [7]: # Solución de la ecuación ax+b=0
def solucion1grado(a, b):
    return -float(b) / a
```

```
In [8]: solucion1grado(2,4)
```

```
Out[8]: -2.0
```

```
In [9]: solucion1grado(0,3)
```

```
-----
ZeroDivisionError                                Traceback (most recent call last)
/home/jesus/Dropbox/docencia13-14/ipython/<ipython-input-9-0dc0bc36d40b> in <module>()
----> 1 solucion1grado(0,3)

/home/jesus/Dropbox/docencia13-14/ipython/<ipython-input-7-60f059795b6a> in solucion1grado(a, b)
      1 # Solución de la ecuación ax+b=0
      2 def solucion1grado(a, b):
----> 3     return -float(b) / a

ZeroDivisionError: float division by zero
```

Podemos evitar el error de división por cero con un *condicional*: la orden `if`

```
In [10]: def solucion1grado(a, b):
         if a != 0:
             return -float(b)/a
```

```
In [11]: solucion1grado(3,8)
```

```
Out[11]: -2.6666666666666665
```

```
In [12]: solucion1grado(0,2)
```

Podemos mejorar la respuesta.

```
In [13]: def solucion1grado(a, b):
         if a != 0:
             resultado = -float(b)/a
         if a == 0:
             resultado = 'ERROR'
         return resultado
```

```
In [14]: solucion1grado(3,4)
```

```
Out[14]: -1.3333333333333333
```

```
In [15]: solucion1grado(0,2)
```

```
Out[15]: 'ERROR'
```

La operación " $a == 0$ " y " $a != 0$ " es de hecho la misma. Sólo necesitamos calcular una, utilizando la orden **else**.

```
In [16]: def solucion1grado(a, b):  
    # solución de la ecuación de primer grado  
    #  $a x + b = 0$   
    if (a != 0):  
        resultado = -float(b)/a  
    else:  
        resultado = "ERROR"  
    return resultado
```

```
In [17]: solucion1grado(3,4)
```

```
Out[17]: -1.3333333333333333
```

```
In [18]: solucion1grado(0,2)
```

```
Out[18]: 'ERROR'
```

Podemos **anidar** diversos condicionales.

```
In [19]: def solucion1grado(a, b):  
    if a != 0:  
        resultado = -float(b)/a  
    if a == 0:  
        if b != 0: #estudio qué pasa si  $a = 0$   
            resultado = 'NO HAY SOLUCIÓN'  
        if b == 0:  
            resultado = 'HAY INFINITAS SOLUCIONES'  
    return resultado
```

o mejor

```
In [20]: def solucion1grado(a, b):  
    if a != 0:  
        resultado = -float(b)/a  
    else: #a == 0  
        if b != 0:  
            resultado = 'NO HAY SOLUCIÓN'  
        else: #b == 0  
            resultado = 'HAY INFINITAS SOLUCIONES'  
    return resultado
```

```
In [21]: solucion1grado(3,4)
```

```
Out[21]: -1.3333333333333333
```

```
In [22]: solucion1grado(0,2)
```

```
Out[22]: 'NO HAY SOLUCI\\xc3\\x93N'
```

```
In [23]: solucion1grado(0,0)
```

```
Out[23]: 'HAY INFINITAS SOLUCIONES'
```

## Otros ejemplos

Estudia si un número es par.

```
In [24]: def par(x):  
         if (x%2 == 0):  
             resultado = True  
         else:  
             resultado = False  
         return resultado
```

```
In [26]: par(30)
```

```
Out[26]: True
```

```
In [44]: #otra versión más sencilla sería  
         def par(x):  
             return (x%2 == 0)
```

¿Es un número el doble de un impar?

```
In [27]: def doble_de_impar(n):  
         if not par(n):  
             resultado = False  
             # no es el doble de nadie  
         else: # el número es par  
             if par(n/2):  
                 resultado = False  
             else:  
                 resultado = True  
         return resultado
```

```
In [30]: doble_de_impar(13)
```

```
Out[30]: False
```

Ser triángulo (con condicionales)

```
In [4]: def esTriangulo(a,b,c):
        return (a + b > c) and (a + c > b) and (c + b > a)

def esEscaleno(a,b,c):
    return esTriangulo(a,b,c) and (a <> b) and\
        (b <> c) and (a <> c)

def esEquilatero(a,b,c):
    return esTriangulo and (a == b) and (b == c)

def esIsosceles(a,b,c):
    return esTriangulo(a,b,c) and (not esEscaleno(a,b,c)) and\
        (not esEquilatero(a,b,c))

def tipo_triangulo(a,b,c):
    if esTriangulo(a,b,c):
        if esEscaleno(a,b,c):
            resultado = 'escaleno'
        else:
            if esEquilatero(a,b,c):
                resultado = 'equilatero'
            else:
                resultado = 'isósceles'
    else:
        resultado = 'no es un triángulo'
    return resultado
```

```
In [32]: tipo_triangulo(4,4,40)
```

```
Out[32]: 'no es un tri\xc3\xa1ngulo'
```

## La instrucción elif

Cuando se concatenan diversas secuencias else ... if podemos contraerlas utilizando "elif"

```
In [24]: def tipo_triangulo(a,b,c):
        if esTriangulo(a,b,c):
            if esEscaleno(a,b,c):
                resultado = 'escaleno'
            elif esEquilatero(a,b,c):
                resultado = 'equilatero'
            else:
                resultado = 'isósceles'
        else:
            resultado = 'no es un triángulo'
        return resultado
```

```
In [25]: tipo_triangulo(2,3,4)
```

```
Out[25]: 'escaleno'
```

## Sentencias iterativas (Bucles)

Para resolver determinados problemas, es necesario repetir una serie de instrucciones un número (determinado o no) de veces.

*Ejemplo* Suma los 10 primeros números

```
In [34]: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11
```

```
Out[34]: 66
```

```
In [35]: def suma(n):  
         return 1 + .. + n
```

```
File "<ipython-input-35-3a0a6513077e>", line 2  
    return 1 + .. + n  
              ^
```

```
SyntaxError: invalid syntax
```

¿Podemos sumar los 100 primeros números? ¿O los n primeros números?

```
In [45]: def suma(n):  
         # Suma los n primeros números  
         i = 1  
         parcial = 0  
         while i <= n:  
             #print 'parcial = ', parcial, 'i = ', i  
             parcial = parcial + i  
             i+=1  
         return parcial
```

```
In [46]: suma(100)
```

```
Out[46]: 5050
```

La sentencia while se usa así

```
while <condición> :
```

```
    acción
```

```
    acción
```

```
    acción
```

y permite expresar *Mientras se cumpla esta condición repite estas acciones.*

```
In [32]: def contador(n):  
         i = 0  
         while i < n:  
             print i  
             i += 1  
         print 'Hecho'
```

```
In [33]: contador(3)
```

```
0  
1  
2  
Hecho
```

```
In [36]: def contador():  
         i = 1 #IMPORTANCIA DEL VALOR INICIAL  
         while i > 3: #CONTROL DEL VALOR  
             print i  
             i += 1  
             #ACTUALIZAR EL VALOR DE CONTROL  
         print 'Hecho'
```

```
In [37]: contador()
```

Hecho

```
In [38]: #muestra los multiplos de n entre n y n.m, ambos incluidos  
def multiplos(n,m):  
    i = 1  
    while i<= m :  
        print n*i  
        i +=1
```

```
In [39]: multiplos(2,20)
```

2  
4  
6  
8  
10  
12  
14  
16  
18  
20  
22  
24  
26  
28  
30  
32  
34  
36  
38  
40

```
In [ ]:
```