

Estructuras de control

Condicionales

Ejemplo: Resolución de la ecuación de primer grado

```
In [1]: # Solución de la ecuación ax+b=0
def solucion1grado(a, b):
    return -float(b) / a
```

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

```
Out[2]: -2.0
```

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

```
-----
--
ZeroDivisionError                                Traceback (most recent call las
t)
<ipython-input-3-0dc0bc36d40b> in <module>()
----> 1 solucion1grado(0,3)

<ipython-input-1-60f059795b6a> 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 [4]: def solucion1grado(a, b):
        if a != 0:
            return -float(b)/a
```

```
In [5]: solucion1grado(3,8)
```

```
Out[5]: -2.6666666666666665
```

```
In [6]: solucion1grado(0,2)
```

Podemos mejorar la respuesta.

```
In [7]: def solucion1grado(a, b):
        if a != 0:
```

```
    resultado = -float(b)/a
if a == 0:
    resultado = 'ERROR'
return resultado
```

```
In [8]: solucion1grado(3,4)
```

```
Out[8]: -1.3333333333333333
```

```
In [9]: solucion1grado(0,2)
```

```
Out[9]: 'ERROR'
```

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

```
In [10]: 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 [11]: solucion1grado(3,4)
```

```
Out[11]: -1.3333333333333333
```

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

```
Out[12]: 'ERROR'
```

Podemos **anidar** diversos condicionales.

```
In [13]: 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 [14]: 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 [15]: `solucionlgrado(3,4)`

Out[15]: `-1.3333333333333333`

In [16]: `solucionlgrado(0,2)`

Out[16]: `'NO HAY SOLUCI\xc3\x93N'`

In [17]: `solucionlgrado(0,0)`

Out[17]: `'HAY INFINITAS SOLUCIONES'`

Otros ejemplos

Estudia si un número es par.

```
In [18]: def par(x):
          if (x%2 == 0):
              resultado = True
          else:
              resultado = False
          return resultado
```

In [19]: `par(30)`

Out[19]: `True`

```
In [20]: #otra versión más sencilla sería
          def par(x):
              return (x%2 == 0)
```

¿Es un número el doble de un impar?

```
In [21]: 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 [22]: doble_de_impar(13)
```

```
Out[22]: False
```

Ser triángulo (con condicionales)

```
In [23]: 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 [25]: tipo_triangulo(4,4,4)
```

```
Out[25]: 'equilatero'
```

La instrucción elif

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

```
In [26]: 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 [27]: tipo_triangulo(2,3,4)
```

```
Out[27]: '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 [28]: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10
```

```
Out[28]: 55
```

```
In [29]: def suma(n):
         return 1 + .. + n
```

```
File "<ipython-input-29-3a0a6513077e>", line 2
    return 1 + .. + n
                ^
```

```
SyntaxError: invalid syntax
```

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

```
In [30]: 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 [31]: suma(20)
```

```
Out[31]: 210
```

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 [34]: def contador(n):  
        i = 0 #IMPORTANCIA DEL VALOR INICIAL  
        while i < n: #CONTROL DEL VALOR  
            print i  
            i += 1  
            #ACTUALIZAR EL VALOR DE CONTROL  
        print 'Hecho'
```

```
In [35]: contador(4)
```

```
0  
1  
2  
3  
Hecho
```

```
In [36]: #cuidado con los bucles sin fin  
def numeros(n):  
    i = 1  
    while i > n:  
        print i  
        i+=1
```

La llamada numeros(0) genera un **bucle infinito**. Casi siempre se puede parar pulsando Control + c

```
In [37]: #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 [38]: multiplos(3,10)
```

```
3
6
9
12
15
18
21
24
27
30
```

Sumatorios

Un utilización típica de los bucles es la suma de cantidades (determinadas o no) de números.

```
In [39]: def sumatorio(n):
        suma = 0
        i = 1
        while i <= n:
            suma += i
            i += 1
        return suma
```

```
In [40]: sumatorio(100)
```

```
Out[40]: 5050
```

Calcula la suma $n + (n+1) + \dots + m$, si $n > m$ devuelve 0

```
In [41]: def suma(n,m):
        s = 0 #valor por defecto
        if n <= m :
            i = n
            while i <= m:
                s = s + i
                #código auxiliar
                #print "suma parcial", s, "valor",i
                i += 1
        return s
```

```
In [42]: suma(1,10)
```

```
Out[42]: 55
```

Cálculo del Factorial: n!

```
In [43]: def factorial(n):  
        fact = 1  
        if n > 0:  
            i = 1  
            while i <= n:  
                fact = fact * i  
                i += 1  
        return fact
```

```
In [44]: factorial(20)
```

```
Out[44]: 2432902008176640000
```

Números primos?

```
In [45]: def esprimo(n):  
        # Decide si el número n es primo  
        # válido para n>1  
        creo_que_es_primo = True  
        divisor = 2  
        while (divisor < n) and (creo_que_es_primo):  
            #print divisor  
            if n % divisor == 0:  
                creo_que_es_primo = False  
                #print 'tengo un divisor = ', divisor  
            divisor += 1  
        return creo_que_es_primo
```

```
In [46]: esprimo(103)
```

```
Out[46]: True
```

El bucle for-in

En Python hay otro tipo de bucles. El bucle *for-in* se puede leer como: **para todo elemento k de una serie, ejecuta una serie de instrucciones (que pueden depender de k)**

Tiene el siguiente aspecto

```
for variable in serie_de_valores:      acción      acción      ...      acción
```

Ejemplo Tabla de multiplicar

```
In [47]: def tabla_multiplicar(n):  
        for i in [1, 2, 3, 4, 5, 6, 8, 7, 9, 10]:  
            print n, 'x', i, '=', n*i
```

```
In [48]: tabla_multiplicar(5)
```



```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 8 = 40
5 x 7 = 35
5 x 9 = 45
5 x 10 = 50
```

Si la lista es muy larga, se puede utilizar **range**

```
In [49]: def tabla_multiplicar(n):
          for i in range(1,11):
              print n, 'x', i, '=', n*i
```

```
In [50]: tabla_multiplicar(8)
```

```
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80
```

range tiene vida propia

```
In [51]: range(20)
```

```
Out[51]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
In [52]: range(3,16)
```

```
Out[52]: [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

```
In [53]: range(1,20,3)
```

```
Out[53]: [1, 4, 7, 10, 13, 16, 19]
```

range es un caso particular de lista --> Introducción a las listas

```
In []:
```