

Estructuras iterativas: bucles acotados `for ... in ...`

A través de los bucles, podemos hacer que una instrucción o secuencia de instrucciones se repitan (un número determinado) de veces. La instrucción básica es `for`.

Sintaxis:

```
for <<variable>> in <<secuencia>>:  
    <<cuerpo del bucle>>
```

Sumar los primeros n enteros:

Es decir, calcular $1 + 2 + 3 + \dots + n$

Algoritmo: establecemos una variable acumuladora con valor inicial 0 y luego un bucle: en cada vuelta, sumamos $i = 1, 2, \dots, n$:

```
In [1]: ▶ n = 10 # Límite superior  
        suma = 0  
        for i in range(1, n+1):  
            suma = suma + i  
  
        print(suma)
```

55

En forma de función. Para abreviar, no documentamos esta función (que está explicada), pero pongamos los tipos de datos de la función anotados:

```
In [2]: ▶ def sum_acot(n: int) -> int:  
        suma = 0  
        for i in range(1, n+1):  
            suma = suma + i  
        return suma  
  
        print(sum_acot(10))
```

55

Bucles `for_in` con listas

```

In [3]: ► def nota_media(lista_de_notas):
        """
        Calcula la media de una lista de notas numéricas

        Parameters:
        -----
        notas : [float]
            Lista de notas, no vacía

        Returns:
        -----
        float
            Media de las notas

        Example
        -----
        >>> nota_media([5, 6, 9, 10])
        7.5
        """
        suma = 0.0
        for nota in lista_de_notas:
            suma = suma + nota
        return suma / len(lista_de_notas)

nota_media([4.5, 6, 5]), nota_media([4, 6, 5, 7, 5, 6, 8]), nota_media([

```

Out[3]: (5.166666666666667, 5.857142857142857, 7.5)

```
In [4]: ▶ def nombres_cortos(lista_de_nombres, n):
        """
        La función filtra una lista_de_nombres,
        devolviendo únicamente los de longitud menor o igual que n.

        Parameters
        -----
        lista_de_nombres : [string]
            Lista of strings
        n : int
            Longitud máxima

        Returns
        -----
        [string]
            Lista de los nombres de lista_de_nombres con long <= n

        Example
        -----
        >>> nombres_cortos(['Ana', 'Marta', 'Patricia', 'Alba', 'Silvia', 'Gloria', 'Lara'], 5)
        ['Ana']
        """
        n_cortos = []
        for nombre in lista_de_nombres:
            if len(nombre) <= n:
                n_cortos.append(nombre)
        return n_cortos

lista = ['Ana', 'Marta', 'Patricia', 'Alba', 'Silvia', 'Gloria', 'Lara']
nombres_cortos(lista, 5), nombres_cortos(lista, 3)
```

```
Out[4]: (['Ana', 'Marta', 'Alba', 'Lara'], ['Ana'])
```

La función range()

La función range() genera un rango, algo muy parecido a una lista, y que se puede convertir en una lista sencillamente:

```
In [5]: ▶ range(10)
```

```
Out[5]: range(0, 10)
```

```
In [6]: ▶ list(range(10))
```

```
Out[6]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Esta función es muy flexible: con dos parámetros, toma el primero como el valor inicial; y con tres, el tercero da el incremento entre elementos:

```
In [7]: ▶ list(range(5, 10)), list(range(5, 20, 3)), list(range(10, 5, -1))
```

```
Out[7]: ([5, 6, 7, 8, 9], [5, 8, 11, 14, 17], [10, 9, 8, 7, 6])
```

Bucles `for_in` para listas generadas con `range()`

La función `range` se usa con frecuencia para diseñar bucles `for_in` :

```
In [8]: ▶ import randomd

def lista_aleatoria(n):
    """
    genera una lista aleatoria de n enteros aleatorios entre 0 y 100.

    Parameters
    -----
    n : int
        Longitud de la lista resultante
        n >= 0

    Returns
    -----
    [int]
        Lista de n enteros aleatorios entre 0 y 100

    Example
    -----
    >>> lista_aleatoria(3)
    [1, 88, 31]
    """
    lista = []
    for x in range(n):
        lista.append(random.randint(0, 100))
    return lista

lista_aleatoria(3), lista_aleatoria(3), lista_aleatoria(5)
```

```
Out[8]: ([64, 80, 56], [47, 73, 14], [81, 25, 92, 64, 71])
```

```
In [9]: ▶ def romano_a_entero(romano):
        """
        Convierte un numeral romano (un string) en su valor numérico entero.

        Parameters
        -----
        romano: str
            Un número romano

        Returns
        -----
        int : int
            El valor numérico del número romano dado

        Precondition
        -----
        romano es un string no vacío cuyas letras están
        entre las siguientes: {"I","V","X","L","C","D","M"}
        y están en mayúscula

        Example
        -----
        >>> romano_a_entero("MCDXCII")
        1492
        """
        valor = {"I": 1, "V": 5, "X": 10, "L": 50, "C": 100, "D": 500, "M":
        total = 0
        total = valor[romano[-1]] # la última cifra siempre suma
        for i in range(len(romano)-1):
            cifra, siguiente = valor[romano[i]], valor[romano[i+1]]
            total = total + (cifra if cifra >= siguiente else - cifra)
        return total

        for r in ["I", "IV", "XIV", "XXXIX", "XLV", "LXIV", "MCDXCII"]:
            print(r, romano_a_entero(r))
```

```
I 1
IV 4
XIV 14
XXXIX 39
XLV 45
LXIV 64
MCDXCII 1492
```

```
In [10]: ► def multiplos_de_7_y_5(n):
        """
        Genera la lista de enteros de [0...n] que son
        multiplos de 7 y de 5 simultáneamente.

        Parameters
        -----
        n : int
            Límite superior

        Returns
        -----
        [int]
            Lista de múltiplos de 7 y 5

        Example
        -----
        >>> multiplos_de_7_y_5(100)
        [0, 35, 70]
        """
        multiplos = []
        for x in range(n):
            if (x % 5 == 0) and (x % 7 == 0):
                multiplos.append(x)
        return multiplos

multiplos_de_7_y_5(100)
```

Out[10]: [0, 35, 70]

Otra versión, saltando ya en el bucle de 7 en 7:

```
In [11]: ► def multiplos_de_7_y_5(n):

        multiplos = []
        for x in range(7, n, 7):
            if x % 5 == 0:
                multiplos.append(x)
        return multiplos

multiplos_de_7_y_5(100)
```

Out[11]: [35, 70]

```
In [12]: ▶ def lista_inversa(lista_inicial):
        """
        Genera una lista como la dada, pero en orden invertido.

        Parameters
        -----
        lista_inicial : list
            Lista original

        Returns
        -----
        list:
            Lista invertida

        Example
        -----
        >>> lista_inversa([1, 2, 3, 4])
        [4, 3, 2, 1]
        """
        lista_inv = []
        for i in lista_inicial:
            lista_inv.insert(0, i)
        return lista_inv

lista_inversa([1,2,3,4]), lista_inversa(["hola","buenas","tardes"])
```

```
Out[12]: ([4, 3, 2, 1], ['tardes', 'buenas', 'hola'])
```

Bucles for_in en strings

```
In [13]: ▶ for c in 'hola':
        print(c, end='')
```

```
hola
```

```
In [14]: ▶ for c in 'Buenas Tardes Ángel':
        print(c.lower(), ord(c.lower()), c.upper(), ord(c.upper()))
```

```
b 98 B 66
u 117 U 85
e 101 E 69
n 110 N 78
a 97 A 65
s 115 S 83
 32 32
t 116 T 84
a 97 A 65
r 114 R 82
d 100 D 68
e 101 E 69
s 115 S 83
 32 32
á 225 Á 193
n 110 N 78
g 103 G 71
e 101 E 69
l 108 L 76
```

```
In [15]: ► for c in 'buenas tardes':  
           print('{0}\t{1}\t{2}\t{3}'.format(c.lower(), ord(c.lower()), c.upper,
```

b	98	B	66
u	117	U	85
e	101	E	69
n	110	N	78
a	97	A	65
s	115	S	83
	32		32
t	116	T	84
a	97	A	65
r	114	R	82
d	100	D	68
e	101	E	69
s	115	S	83

```
In [16]: ► def cuenta_letras(letra, palabra):  
           """  
           Cuenta el número de apariciones de una letra en la palabra dada  
  
           Parameters  
           -----  
           letra : string  
               Letter to count the occurrences  
           word : string  
               Word  
  
           Result  
           -----  
           int  
               Número de veces que la letra aparece en la palabra  
  
           Example  
           -----  
           >>> cuenta_letras('o', 'pelirrojo')  
           2  
           """"  
           contador = 0  
           for char in palabra:  
               if char == letra:  
                   contador = contador + 1  
           return contador
```

```
In [17]: ► cuenta_letras('o', 'pelirrojo')
```

```
Out[17]: 2
```

```
In [18]: ► cuenta_letras('j', 'pelirrojo')
```

```
Out[18]: 1
```

```
In [19]: ► cuenta_letras('a', 'pelirrojo')
```

```
Out[19]: 0
```



```
In [20]: ▶ cuenta_letras('J', 'pelirrojo')
```

```
Out[20]: 0
```

Palabras en un texto

Tengo el texto del Quijote completo en el archivo siguiente:

```
Quijote = ["En un lugar de la Mancha, de cuyo nombre no quiero acor  
darme, ",  
          "no ha mucho tiempo que vivía un hidalgo de los de lanza en ast  
illero, ",  
          "adarga antigua, rocín flaco y galgo corredor. ",  
          "Una olla de algo más vaca que carnero, salpicón las más noche  
s, ",  
          "duelos y quebrantos los sábados, lantejas los viernes, algún p  
alomino ",  
          "de añadidura los domingos, consumían las tres partes de su hac  
ienda. ",  
          "El resto della concluían sayo de velarte, calzas de velludo pa  
ra las ",  
          "fiestas, con sus pantuflos de lo mismo, y los días de entresem  
ana El ",  
          "se honraba con su vellorí de lo más fino. Tenía en su casa una  
ama que ",  
          "pasaba de los cuarenta, y una sobrina que no llegaba a los vei  
nte, y ",  
          "un mozo de campo y plaza, que así ensillaba el rocín como toma  
ba la El ",  
          "podadera. Frisaba la edad de nuestro hidalgo con los cincuenta  
años; "  
]
```

Vamos a diseñar una función que selecciona las palabras que tienen una cierta longitud. Para simplificar en este momento, consideramos que una palabra es una cadena de caracteres no blancos, separados del resto por espacios en blanco. Admitiremos (excepcionalmente) otros caracteres como puntos o comas.

```
In [21]: ▶ def palabras(cadena_lineas):
    lista = []
    for linea in cadena_lineas:
        palabras = linea.split()
        for palabra in palabras:
            # lista.append(palabra.strip()) # sin limpiar los caracteres
            lista.append(palabra.strip(".;:()[]"))
    return(lista)

Quijote = ["En un lugar de la Mancha, de cuyo nombre no quiero acordarme
"no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero
"adarga antigua, rocín flaco y galgo corredor. ",
"Una olla de algo más vaca que carnero, salpicón las más noches, ",
"duelos y quebrantos los sábados, lantejas los viernes, algún palomino
"de añadidura los domingos, consumían las tres partes de su hacienda
"El resto della concluían sayo de velarte, calzas de velludo para las
"fiestas, con sus pantuflos de lo mismo, y los días de entresemana
"se honraba con su vellorí de lo más fino. Tenía en su casa una ama
"pasaba de los cuarenta, y una sobrina que no llegaba a los veinte,
"un mozo de campo y plaza, que así ensillaba el rocín como tomaba la
"podadera. Frisaba la edad de nuestro hidalgo con los cincuenta años
"]

print(palabras(Quijote))
```

```
['En', 'un', 'lugar', 'de', 'la', 'Mancha', 'de', 'cuyo', 'nombre', 'n', 'o', 'quiero', 'acordarme', 'no', 'ha', 'mucho', 'tiempo', 'que', 'viví', 'a', 'un', 'hidalgo', 'de', 'los', 'de', 'lanza', 'en', 'astillero', 'ad', 'arga', 'antigua', 'rocín', 'flaco', 'y', 'galgo', 'corredor', 'Una', 'o', 'lla', 'de', 'algo', 'más', 'vaca', 'que', 'carnero', 'salpicón', 'las', 'más', 'noches', 'duelos', 'y', 'quebrantos', 'los', 'sábados', 'lantej', 'as', 'los', 'viernes', 'algún', 'palomino', 'de', 'añadidura', 'los', 'domingos', 'consumían', 'las', 'tres', 'partes', 'de', 'su', 'haciend', 'a', 'El', 'resto', 'della', 'concluían', 'sayo', 'de', 'velarte', 'calz', 'as', 'de', 'velludo', 'para', 'las', 'fiestas', 'con', 'sus', 'pantuflo', 's', 'de', 'lo', 'mismo', 'y', 'los', 'días', 'de', 'entresemana', 'El', 'se', 'honraba', 'con', 'su', 'vellorí', 'de', 'lo', 'más', 'fino', 'Te', 'nía', 'en', 'su', 'casa', 'una', 'ama', 'que', 'pasaba', 'de', 'los', 'cuarenta', 'y', 'una', 'sobrina', 'que', 'no', 'llegaba', 'a', 'los', 'veinte', 'y', 'un', 'mozo', 'de', 'campo', 'y', 'plaza', 'que', 'así', 'ensillaba', 'el', 'rocín', 'como', 'tomaba', 'la', 'El', 'podadera', 'Frisaba', 'la', 'edad', 'de', 'nuestro', 'hidalgo', 'con', 'los', 'cin', 'cuenta', 'años']
```

Variables aleatorias básicas

```
In [22]: ▶ # Bernuilli

import random

def moneda_cargada(prob_cara: float) -> bool:
    unif_01 = random.random() # se obtiene un número aleatorio del inter
    return unif_01 < prob_cara

for _ in range(10):
    print(moneda_cargada(0.7))

True
True
True
False
True
False
False
False
True
False
```

```
In [23]: ▶ for _ in range(10):
            print(moneda_cargada(0.7))

True
True
True
False
True
True
True
True
True
True
```

```
In [24]: ▶ # Binomial

def varios_lanzamientos(prob_cara, n):
    total = 0
    for _ in range(n):
        if moneda_cargada(prob_cara):
            total = total + 1
    return total

for _ in range(10):
    print(varios_lanzamientos(0.7, 10))

8
7
8
7
6
8
9
4
4
7
```

