Estructuras iterativas: bucles condicionales

A través de los bucles, podemos hacer que una instrucción o secuencia de instrucciones se repitan (un número determinado o no) de veces. La instrucción básica es `while` .

Bucle while

Sintaxis:

```
while <<condición>>:
    <<instrucciones>>
```

Las instrucciones ser repiten mientras se verifique la condición.

He aquí un ejemplo muy sencillo. Deseamos sumar los primeros números enteros hasta uno dado. Por ejemplo, si el límite superior es 10, la suma sería 1 + 2 + ... + 10, lo cual vale 55:

In [1]:

```
lim_sup = 10
suma = 0
i = 1
while i <= lim_sup:
    suma = suma + i
    i = i + 1

print(suma)
```

```
55
```

En forma de función, sería así

In [2]:

```python
def suma_hasta(lim_sup):
    """
    Esta función calcula la suma de los enteros desde 1 hasta lim_sup,
    incluyendo ambos. Si lim_sup < 1, la función devuelve 0
    (puesto que no hay ningún entero entero que cumpla la condición).

    Parameters
    ----------
    lim_sup : int
        límite superior

    Returns
    -------
    int
        Suma de 1 + 2 + 3 + ... lim_sup

    Example
    -------
    >>> suma_hasta(4)
    10
    """
    suma = 0
    i = 1
    while i <= lim_sup:
        suma = suma + i
        i = i + 1
    return suma
```

In [3]:

```python
suma_hasta(8), suma_hasta(7), suma_hasta(-3), suma_hasta(400)
```

Out[3]:

```
(36, 28, 0, 80200)
```

Factorización $n = 2^k * m$

Descomponer un número, separando su potencia de dos y el factor restante.

Ejemplos: `12 = 2^2 * 3`, `16 = 2^4 * 1`, `7 = 2^0 * 7` .

Una primera aproximación.

In [4]:

```
def mayor_pot_2(n):
    """
    Esta función calcula la mayor potencia de 2
    que es divisor de un entero dado, n, dando además
    el factor complementario:
    Si tenemos n = 2^k * r, y 2^(k+1) ya no es divisor de n:
        mayor_pot_2(n) = 2^k, r

    Parameters
    ----------
    n: int
        Entero, del que deseamos extraer la mayor potencia de 2

    Returns
    -------
    (int, int)
        Par (pow2, r) donde pow2 es la mayor potencia de 2 que divide a n,
        y pow2*r = n

    Example
    -------
    >>> mayor_pot_2(12)
    (4, 3)
    """
    pow2, resto = 1, n
    while (resto % 2) == 0:
        pow2 = pow2 * 2
        resto = resto // 2
    return pow2, resto

mayor_pot_2(12), mayor_pot_2(8), mayor_pot_2(7)
```

Out[4]:

```
((4, 3), (8, 1), (1, 7))
```

Ya estamos muy cerca, ahora contamos el exponente de la potencia en lugar de calcularlo.

In [5]:

```python
def mayor_pot2_exp(n):
    """
    Esta función calcula el mayor exponente de 2, digamos k,
    tal que 2^k es divisor de un entero dado, n, dando además
    el factor complementario:
    Si tenemos n = 2^k * r, y 2^(k+1) ya no es divisor de n:
        mayor_pot_2(n) = k, r

    Parameters
    ----------
    n: int
        Entero, del que deseamos extraer la mayor potencia de 2

    Returns
    -------
    (int, int)
        Par (exp2, r) donde 2^exp2 es la mayor potencia de 2 que divide a n,
        y 2^exp2 * k = n

    Example
    -------
    >>> mayor_pot2_exp(12)
    (2, 3)
    """
    exp2, resto = 0, n
    while (resto % 2) == 0:
        exp2 = exp2 + 1
        resto = resto // 2
    return exp2, resto

mayor_pot2_exp(2), mayor_pot2_exp(8), mayor_pot2_exp(7), mayor_pot2_exp(12)
```

Out[5]:

```
((1, 1), (3, 1), (0, 7), (2, 3))
```

Si queremos *ver* los resultados más bonitos...

In [6]:

```python
n = 3214134134
a, b = mayor_pot2_exp(n)
print("{0} = 2^{1} * {2}".format(n, a, b))
```

```
3214134134 = 2^1 * 1607067067
```

Nota: No es buena idea introducir la visualización en el `return` de la función. Es mejor diseñar funciones que calculan valores y, si luegon queremos ver el resultado en un forma más legible, diseñamos algo parecido a lo que acabamos de hacer con la función `print` .

Suma de las cifras de un número

Para trabajar con las cifras de un número, hay dos expresiones de gran utilidad: el cociente y el resto de dividir por 10, que nos dan elmúmero sin su última cifra y dicha úlria cifra:

In [7]:

```
n = 1536
n // 10, n % 10
```

Out[7]:

```
(153, 6)
```

Ahora podemos usar una asignación añadir la última cifra a una variable acumulador (digmaos que está a cero) y otra para transformar el número eliminando su última cifra:

In [8]:

```
n = 174539
acum = 0
print(n, acum)
acum = acum + n % 10
n = n // 10
print(n, acum)
```

```
174539 0
17453 9
```

Hagámoslo de nuevo:

In [9]:

```
acum = acum + n % 10
n = n // 10
print(n, acum)
```

```
1745 12
```

Unas pocas veces más:

In [10]:

```
acum = acum + n % 10
n = n // 10
print(n, acum)

acum = acum + n % 10
n = n // 10
print(n, acum)

acum = acum + n % 10
n = n // 10
print(n, acum)

acum = acum + n % 10
n = n // 10
print(n, acum)

acum = acum + n % 10
n = n // 10
print(n, acum)
```

```
174 17
17 21
1 28
0 29
0 29
```

Vemos que, cuando n == 0, las instrucciones no tienen efecto: se acumula un 0 y el número no cambia. Podíamos haber parado cuando n== 0.

Con `while` la cosa es más sencilla, general y clara

In [11]:

```python
def suma_de_cifras(n):
    """
    Esta función calcula la suma de las cifras de un entero positivo

    Parameters
    ----------
    n : int
        Un entero positivo

    Returns
    -------
    int
        La suma de los dígitos de n

    Example
    -------
    >>> suma_de_cifras(123)
    6
    """
    suma = 0
    while n != 0:
        suma = suma + n % 10
        n = n // 10
    return suma

print(suma_de_cifras(123))
print(suma_de_cifras(239814065983))
```

```
6
58
```

Criterios de divisibilidad

¿Es el número `2334324365987645235 78` divisible por `3` y por `9` ?

In [12]:

```
def divisible_by_3(n):
    """
    This function decides if a positive integer is divisible by 3. n >= 0.

    Parameters
    ----------
    n : int
        Integer positive number

    Returns
    -------
    bool
        Whether n is divisible by 3 or not

    Example
    -------
    >>> divisible_by_3(14)
    False
    """
    copy = n
    while copy > 9:
        copy = suma_de_cifras(copy)
    if (copy == 0) or (copy == 3) or (copy == 6) or (copy == 9):
        return True
    else:
        return False
```

In [13]:

```
print(divisible_by_3(334132413413241231))
print(divisible_by_3(14))
```

```
True
False
```

In [14]:

```
def divisible_by_9(n):
    """
    This function decides if a positive integer is divisible by 9. n >= 0.

    Parameters
    ----------
    n : int
        Integer positive number

    Returns
    -------
    bool
        Whether n is divisible by 9 or not

    Example
    -------
    >>> divisible_by_9(19)
    False
    """
    copy = n
    while copy > 9:
        copy = suma_de_cifras(copy)
    return (copy == 0) or (copy == 9)
```

In [15]:

```
divisible_by_9(18), divisible_by_9(3413413413414), divisible_by_9(19)
```

Out[15]:

```
(True, True, False)
```

In [16]:

```
divisible_by_3(233432436598764523578) and \
divisible_by_9(233432436598764523578)
```

Out[16]:

```
True
```

Divisivilidad por 11: las suma de las cifras en posición par es igual a la suma de las cifras en posición impar.

In [17]:

```
def sum_par_impar(n):
    pos_par = True
    pares = 0
    impares = 0

    while  n!=0:
        digit = n%10
        if pos_par :
            pares = pares + digit
        else:
            impares = impares + digit
        n= n // 10
        pos_par = not pos_par
    return (pares,impares)
```

In [18]:

```
sum_par_impar(12123)
```

Out[18]:

```
(5, 4)
```

In [19]:

```
def divisible_11(n):
    while n > 11:
        pares, impares = sum_par_impar(n)
        if pares > impares:
            n = pares - impares
        else:
            n = impares - pares
    return n==0 or n==11
```

In [20]:

```
divisible_11(11)
```

Out[20]:

```
True
```

In [21]:

```
divisible_11(135777972)
```

Out[21]:

```
True
```

In [22]:

```
divisible_11(135776972)
```

Out[22]:

```
False
```

¿Es primo un número?

¿Es el número `233432436598764523577` primo? ¿Lo es n? Para saberlo, basta con tantear si son divisores los números $2, 3, \dots, \sqrt{n}$.

In [23]:

```
def es_primo(n):
    """
    Function that checks if n is prime

    Pamarameters
    ============
    n : int

    Returns
    =======
    bool

    Precondition
    ============
    n>1
    """
    i = 2
    while i*i <= n and (n%i!=0):
        i += 1
    return i*i>n
```

In [24]:

```
print([(i, es_primo(i)) for i in range(2, 100)])
```

```
[(2, True), (3, True), (4, False), (5, True), (6, False), (7, True), (8, Fal
se), (9, False), (10, False), (11, True), (12, False), (13, True), (14, Fals
e), (15, False), (16, False), (17, True), (18, False), (19, True), (20, Fals
e), (21, False), (22, False), (23, True), (24, False), (25, False), (26, Fal
se), (27, False), (28, False), (29, True), (30, False), (31, True), (32, Fal
se), (33, False), (34, False), (35, False), (36, False), (37, True), (38, Fa
lse), (39, False), (40, False), (41, True), (42, False), (43, True), (44, Fa
lse), (45, False), (46, False), (47, True), (48, False), (49, False), (50, F
alse), (51, False), (52, False), (53, True), (54, False), (55, False), (56,
False), (57, False), (58, False), (59, True), (60, False), (61, True), (62,
False), (63, False), (64, False), (65, False), (66, False), (67, True), (68,
False), (69, False), (70, False), (71, True), (72, False), (73, True), (74,
False), (75, False), (76, False), (77, False), (78, False), (79, True), (80,
False), (81, False), (82, False), (83, True), (84, False), (85, False), (86,
False), (87, False), (88, False), (89, True), (90, False), (91, False), (92,
False), (93, False), (94, False), (95, False), (96, False), (97, True), (98,
False), (99, False)]
```

Ejercicio propuesto

Diseña un algoritmo que realice la descomposición clásica de un número en factores, a la manera clásica: se comienza dividiendo el número original entre el divisor más pequeño posible (2), se actualiza el dividendo y se continúa con ese divisor o con el siguiente, cuando haya de ser así:

```
60|2
30|2
15|3
 5|5
 1|
```