Recorridos y búsquedas en listas y strings

1 Recorridos

Ya hemos visto muchos ejemplos de recorridos. Un recorrido es cuando un bucle se ejecuta a lo largo de una secuencia sin pausa y sin necesidad de terminar antes del final.

In [7]:
```
def mean(numberlst):
    """This function returns the mean of a list of numbers

    Parameters
    ----------
    numberlst : [float]
        List of float numbers

    Returns
    -------
    float
        Mean of the list

    Example
    -------
    >>> mean([1,2,3,4])
    2.5
    """
    s = 0.0
    i = 0
    while i < len(numberlst):
        s = s + numberlst[i]
        i = i + 1
    return s / len(numberlst)
```

In [8]:
```
mean([1,2,3,4])
```

Out [8]:
```
2.5
```

In [9]:
```
def digit_sum(n):
    """
    This function computes the sum of the digits of a positive integer, n >= 0.

    Parameters
    ----------
    n : int
        Positive integer number

    Returns
    -------
    int
        Sum of the digits of n

    Example
    -------
    >>> digit_sum(1234)
    10
    """
    result = 0
    while n != 0:
        digit = n % 10
        result = result + digit
        n = n // 10
    return result
```

```
In [10]: digit_sum(1234)
Out [10]: 10
```

1.1 Filtros

Un tipo de recorrido muy utilizado son los filtros. En estos recorridos aplicamos una función para saber qué elementos de la secuencia original verifican una determinada propiedad.

```
In [13]:
def short_names(name_list, n):
    """
    From a list of names, the function chooses the names with length below of equal n.
    All the names that satisfy the condition are returned in a list.

    Parameters
    ----------
    name_list : [string]
        List of names
    n : int
        Maximum length for names

    Returns
    -------
    [string]
        List of names in name_list whose length is less than or equal to n

    Example
    -------
    >>> short_names(['Ana','Pedro','Eva'], 3)
    ['Ana', 'Eva']
    """
    short = []
    for name in name_list:
        if len(name) <= n:
            short.append(name)
    return short
```

```
In [14]: short_names(['Ana','Pedro','Eva'], 3)
Out [14]: ['Ana', 'Eva']
```

```
In [17]:
def letter_count(letter, word):
    """
    Counts the occurrences of letter in word.

    Parameters
    ----------
    letter: string
        Letter to search
    word: string
        Word

    Returns
    -------
    int
        Occurrences of letter in word

    Example
    -------
    >>> letter_count('o', 'abogado')
    2
    """
    cont = 0
    for char in word:
        if char == letter:
            cont = cont + 1
    return cont
```

```
In [18]: letter_count('o', 'abogado')
```

```
Out [18]: 2
```

```
In [23]: def substr(word, ini, num):
             """This function returns the substring starting in end with num letters.
             If ini>=len(word) the function returns the empty string. If ini<len(word) and ini+
             function returns the string starting in ini with len(word)-ini letters

             Parameters
             ----------
             word : str
                 String
             ini : int
                 Starting position of the substring (ini >= 0)
             num : int
                 Length of the substring (num >= 0)

             Returns
             -------
             str
                 Substring of word starting at position ini with length num

             Example
             -------
             >>> substr("Bienvenido", 4, 6)
             "venido"
             """
             i = 0
             sub = ""
             while (i + ini) < len(word) and (i < num):
                 sub += word[ini + i]
                 i += 1
             return sub
```

```
In [24]: substr("Hola",2,2), substr("Hola",4,2), substr("Hola",3,4), substr("Bienvenido", 4, 6)
```

```
Out [24]: ('la', '', 'a', 'venido')
```

2 Búsquedas

Las búsquedas tienen que hacer un recorrido, pero a diferencia de éstos, en cualquier momento este recorrido puede terminar al haber encontrado lo que se deseaba.

2.1 Buscar si un elemento está o no en una lista

```
In [28]: def search1(name, staff):
             """
             fuction that indicates if the list staff contains the person name

             Parameters
             ----------
             name : string
                 Person name
             staff : [str]
                 List of names

             Returns
             -------
             bool
                 Whether name appears in staff

             Example
             -------
```

```
    >>> search1("María",["Juan","Luisa","María","Eva"])
    True
    """
    result = False
    for p in staff:
        if p == name:
            result = True
    return result
```

In [27]:
```
search1("María",["Juan","Luisa","María","Eva"]),search1("Petra",["Juan","Luisa","María
```

Out [27]:
```
(True, False)
```

La función anterior no es muy eficiente, una vez encontrada la persona sigue recorriendo la lista hasta el final. Podemos hacer mejor las cosas.

In [29]:
```
def search2(name, staff):
    """
    fuction that indicates if the list staff contains the person name

    Parameters
    ----------
    name : string
        Person name
    staff : [str]
        List of names

    Returns
    -------
    bool
        Whether name appears in staff

    Example
    -------
    >>> search2("María",["Juan","Luisa","María","Eva"])
    True
    """
    i = 0
    while i<len(staff) and name!=staff[i]:
        i = i+1
    return i < len(staff)
```

In [32]:
```
search2('pepa',['ana','luis','pep']), search2("María",["Juan","Luisa","María","Eva"])
```

Out [32]:
```
(False, True)
```

2.2 Ahora con números

In [33]:
```
def random_list(n):
    """
    Returns a list of n random integer between 0 and 100.

    Parameters
    ----------
    n : int
        Number of randon integers

    Returns
    -------
    [int]
        List of random integers of length n

    Example
    -------
    >>> random_list(3)
    [14, 47, 77]
```

```
    """
    import random
    result = []
    for x in xrange(n):
        result.append(random.randint(0,100))
    return result
```

Generamos una lista aleatoria de enteros.

In [35]:
```
l = random_list(100)
print l
```

```
[14, 47, 77, 48, 85, 90, 67, 99, 87, 49, 85, 50, 42, 45, 63, 1, 80,
65, 54, 23, 42, 56, 35, 62, 89, 43, 38, 26, 14, 51, 14, 58, 53, 57, 2,
52, 43, 44, 89, 62, 28, 94, 54, 95, 27, 92, 48, 24, 44, 86, 50, 80,
11, 44, 62, 51, 85, 5, 52, 9, 61, 3, 54, 62, 40, 74, 95, 1, 8, 20, 35,
24, 46, 84, 12, 36, 47, 73, 10, 40, 93, 67, 39, 62, 64, 18, 67, 82,
42, 39, 24, 70, 5, 34, 24, 36, 82, 64, 70, 30]
```

Para buscar si un número está en l, ¡¡¡el código es el mismo que antes!!!. Podemos hacer las cosas "genéricas"

In [39]:
```
def search(e,l):
    """
    fuction that indicates if the list l contains the element e

    Parameters
    ----------
    e : x
        Element
    l : [x]
        List of values of type x

    Returns
    -------
    bool
        Whether l contains the element e or not

    Example
    -------
    >>> search(1, [3,2,1])
    True
    >>> search("pepe", ["ana","juan"])
    False
    """
    i = 0
    while (i < len(l)) and (e != l[i]):
        i = i + 1
    return (i < len(l))
```

In [40]:
```
search(2,l),search(48,l)
```

Out [40]:
```
(True, True)
```

Lo que hemos hecho es tan útil que los creadores de Python lo han incorporado al lenguaje:

```
<<elemento>> in <<secuencia>>
```

In [41]:
```
2 in l
```

Out [41]:
```
True
```

In [42]:
```
'pepas' in ['ana','luis','pepa']
```

Out [42]:
```
False
```

Buscar una palabra en un texto.

In [47]:
```
def find(text, word):
    """
    This function searches the word in text. The function returns the position of the
    word in the text. If the word is not in the text, the function returns -1

    Parameters
    ----------
    text : str
        Text
    word : str
        Word

    Returns
    -------
    int
        Position of word in text, or -1 if the word does not appear in text

    Example
    -------
    >>> find("Esta es mi casa", "mi")
    8
    >>> find("Esta es mi casa", "ventana")
    -1
    """
    i = 0
    lenw = len(word) #this is just an abbreviature
    while (i < len(text)) and (substr(text, i, lenw) != word):
        i += 1
    if i < len(text):
        return i
    else:
        return -1
```

In [48]:
```
s="Las patatas me gustan"
find(s, "pata"), find(s, "zana"), find("Esta es mi casa", "mi"), find("Esta es mi casa
```

Out [48]:
```
(4, -1, 8, -1)
```

In [50]:
```
"Pedro" in "Pedro y Juan", "Lorena" in "Pedro y Juan"
```

Out [50]:
```
(True, False)
```

Para las cadenas de carácteres disponemos del método find, `cadena1.find(cadena2)` devuelve la posición en que `cadena2` se encuentra como subcadena de `cadena1`. Devuelve -1 si `cadena1` no contiene a `cadena2`.

In [51]:
```
texto="""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 mesmo, y los días de
entresemana 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 veinte,
y 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; era de
complexión recia, seco de carnes, enjuto de rostro, gran madrugador y amigo
de la caza. Quieren decir que tenía el sobrenombre de Quijada, o Quesada,
que en esto hay alguna diferencia en los autores que deste caso escriben;
aunque, por conjeturas verosímiles, se deja entender que se llamaba
Quejana. Pero esto importa poco a nuestro cuento; basta que en la narración
dél no se salga un punto de la verdad."""
```

In [55]:
```
texto.find("en"), texto.find("internet")
```

Out [55]:
```
(119, -1)
```

No tenemos que conformarnos con la primera aparición podemos pedir la aparición posterior a una posición:

In [56]:
```
texto.find("en",120)
```

Out [56]:
```
384
```

Con esto y nuestra creciente habilidad en el manejo de los bucles podemos escribir una función que nos dé todas las apariciones.

In [66]:
```
def all_occurrences(text,word):
    """
    function that returns the positions of all occurrences of word in text

    Parameters
    ----------
    text : str
        Text
    word : str
        Word

    Returns
    -------
    [int]
        List with the position of all the occurrences of word in text

    Example
    -------
    >>> all_occurrences("el capitán y el timonel", "el")
    [0, 14, 22]
    """
    result = []
    last = text.find(word)
    while last != -1:
        result.append(last)
        last = text.find(word, last + 1)
    return result
```

In [67]:
```
all_occurrences(texto,"en"), all_occurrences("el capitán y el timonel", "el")
```

Out [67]:
```
([119,
  384,
  515,
  572,
  578,
  619,
  705,
  798,
  852,
  911,
  925,
  938,
  973,
  997,
  1003,
  1039,
  1088,
  1091,
  1156,
  1172],
 [0, 14, 22])
```

3 Cuando los elementos están ordenados

Si nuestra lista está ordenada las búsquedas pueden ser mas eficientes. Antes de llegar al final podemos saber que el elemento no está.

In [71]:
```
def search_ord(e,l):
    """
    fuction that indicates if the list l contains the element e

    Parameters
    ----------
    e : x
        Element that can be compared using <=
    l : [x]
        Ordered list of values of type x

    Returns
    -------
    bool
        Whether e appears in l or not

    Example
    -------
    >>> search_ord( 3, [1,3,4,9])
    True
    """
    i = 0
    while (i < len(l)) and (l[i] < e):
        i = i + 1
    return (i < len(l)) and (e == l[i])
```

In [70]:
```
search_ord( 3, [1,3,4,9])
```

Out [70]:
```
True
```

Lo podemos mejorar bastante. La idea es parecida a la forma en que buscamos en un diccionario. Supongamos que buscamos “marmota”. Abrimos el diccionario por la mitad y nos encontramos con la palabra “integridad” en la página 272, sabemos que está entre aquí y el final. Probamos en la mitad entre la 172 y el final y nos encontramos con “paz” en la 351, siguiendo el proceso encontramos muy rápido la palabra . . . . . Nuestro caso es más fácil pues cada número (índice de la lista) sólo tenemos un elemento.

In [72]:
```
def binary_search(e,l):
    """
    fuction that indicates if the list l contains the element e using binary search

    Parameters
    ----------
    e : x
        Element of type x. It must be comparable with <=
    l : [x]
        Ordered list of values of type x

    Returns
    -------
    bool
        Whether e appears in l or not

    Example
    -------
    >>> binary_search(8,[3,4,5,8,10])
    True
    """
    i = 0
    j = len(l) - 1
    result = False
    while (i <= j) and (not result):
```

```
        med = (i + j) // 2
        if l[med] == e:
            result = True
        elif l[med] < e:
            i = med + 1
        else:
            j = med - 1
    return result
```

In [73]:
```
binary_search(8,[3,4,5,8,10])
```

Out [73]:
```
True
```

In []: