Recorridos y búsquedas en listas y strings

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 []:
```
def mean(numberlst):
    """This function returns the mean of a list of numbers
    @type numberlst: [float]
    @rtype float
    """
    s = 0.0
    i = 0
    while i<len(numberlst):
        s = s + numberlst[i]
        i = i + 1
    return s / len(numberlst)
mean(numberlst)
```

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

    @type n: int
    @rtype: int
    """
    result = 0
    while n != 0:
        digit = n%10
        result = result + digit
        n = n//10
    return result
```

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

    @type name_list: [string]
    @type n: int
    @rtype: [string]
    """
    short = []
    for name in name_list:
        if len(name) <= n:
            short.append(name)
    return short
```

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

    @type letter: string
    @type word: string
    @rtype: int
    """
    cont = 0
    for char in word:
        if char == letter:
            cont = cont + 1
    return cont
```

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.

Buscar si un elemento está o no en una lista

In [1]:
```
def search1(name,staff):
    """
    fuction that indicates if the list staff contains the person name
    @type name:string
    @type staff:list of string
    """
    result = False
    for p in staff:
        if p == name:
            result = True
    return result
```

In [3]:
```
search("María",["Juan","Luisa","María","Eva"]),find("Petra",["Juan","Lu
isa","María","Eva"])
```

Out[3]: (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 []:
```
def search2(name,staff):
    """
    fuction that indicates if the list staff contains the person name
    @type name:string
    @type staff:list of string
    """
    i = 0
    while i<len(staff) and name!=staff[i]:
        i = i+1
    return i<len(staff)
```

Ahora con números

In [15]:
```
def random_list(n):
    """
    Returns a list of n random integer between 0 and 100.

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

Generamos una lista aleatoria de enteros.

In [17]:
```
l = random_list(100)
```

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

In [18]:
```
def search(e,l):
    """
    fuction that indicates if the list l contains the element e
    @type e:x
    @type l:list of x
    """
    i = 0
    while i<len(l) and e!=l[i]:
        i = i+1
    return i<len(l)
```

In [20]:
```
search(2,l),search(48,l)
```

Out[20]: (True, False)

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

<<elemento>> in <<secuencia>>

In [30]:
```
2 in l
```

Out[30]: True

In [22]:
```
"Pedro" in "Pedro y Juan"
```

Out[22]: True

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 [35]:
```
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 antigu
a,
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 vein
te,
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; er
a de
complexión recia, seco de carnes, enjuto de rostro, gran madrugador y a
migo
de la caza. Quieren decir que tenía el sobrenombre de Quijada, o Quesad
a,
que en esto hay alguna diferencia en los autores que deste caso escribe
n;
aunque, por conjeturas verosímiles, se deja entender que se llamaba
Quejana. Pero esto importa poco a nuestro cuento; basta que en la narra
ción
dél no se salga un punto de la verdad."""
```

In [36]:
```
texto.find("en"), texto.find("internet")
```

Out[36]: (119, -1)

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

In [58]:
```
texto.find("en",120)
```

Out[58]: 384

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

In [60]:
```
def all_occurences(text,word):
    """
    function that returns the positions of all occurences of word in te
xt
    @type text:string
    @type word:string
    """
    result = []
    last = text.find(word)
    while last!=-1:
        result.append(last)
        last = text.find(word,last+1)
    return result
```

In [62]:
```
all_occurences(texto,"en")
```

Out[62]:
```
[119,
 384,
 515,
 572,
 578,
 619,
 705,
 798,
 852,
 911,
 925,
 938,
 973,
 997,
 1003,
 1039,
 1088,
 1091,
 1156,
 1172]
```

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 [70]:
```
def search_ord(e,l):
    """
    fuction that indicates if the list l contains the element e
    @type e:x
    @type l:list of x
    @preconditions: elements of type x must be comparables with < and
                    l is ordered.
    """
    i = 0
    while i<len(l) and l[i]<e:
        i = i+1
    return i<len(l) and e==l[i]
```

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 [74]:
```
def binary_search(e,l):
    """
    fuction that indicates if the list l contains the element e
    @type e:x
    @type l:list of x
    @preconditions: elements of type x must be comparables with < and
                    l is ordered.
    """
    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 [77]:
```
binary_search(8,[3,4,5,8,10])
```

Out[77]: True

In []: