Repaso de conceptos básicos
Cadenas

Fundamentos de Programación

Enero 2019

Cadenas

- Literales: `'Hola'`, `"Hola"`
- Asignación:

```
s = "¡Hola mundo!"
```

- Recorrido mediante índices:

```
i=0
count=0
while i<len(s):
    if s[i]=='e':
        count+=1
    i=i+1
```

- Recorrido mediante un for:

```
for c in s:
    if c=='!':
        break
```

- Trozos de cadenas:

```
>>> print(s[2:7])
ola m
>>> print(s[:3])
¡Ho
>>> print(s[3:])
la mundo!
```

- Búsquedas:

```
def find(c,s):
    i=0
    while i<len(s):
        if s[i]==c:
            return i
        i=i+1
    return None
```

Cadenas

- El operador `in` (comprobación de aparición)

```
>>> "red" in "perro verde"
False
>>> "erd" in "perro verde"
True
```

- Operador `in` (recorrido de iterables mediante for)

```
def letras_comunes(s1,s2):
    for c in s1:
        if c in s2:
            print(c)
```

- Ordenación *alfabética*:

```
def ordenar(s1,s2):
    if s1<s2:
        return [s1,s2]
    else:
        return [s2,s1]
```

```
>>> l=ordenar('pep','mau')
>>> print(l)
['mau','pep']

>>> l=ordenar('pep','Tom')
>>> print(l)
['Tom','pep']
```

Cadenas

- Operaciones con cadenas:
  - Concatenación: `s1+s2`
  - Repetición: `s*3`
  - Conversiones de datos a cadena:
    - **`str`**`(x)`
      imprimir
    - **`repr`**`(x)`
      representar para almacenar
    - `"{0:.4f}"`**`.format`**`(x)`
      imprimir con formato

- Métodos:
  - `s.`**`count`**`(a[,i,j])`
  - `s.`**`find`**`(a[,i,j])`
  - `s.`**`replace`**`(a,b[,maxreps])`
  - `s.`**`split`**`([sep[,maxsplits]])`
  - `s.`**`join`**`(t)`
  - `s.`**`splitlines`**`([keepends])`
  - `s.`**`strip`**`([charset])`
  - `s.`**`lower`**`()`
  - `s.`**`upper`**`()`
  - `s.`**`capitalize`**`()`

Cadenas
Constantes de grupos de caracteres

- `import string`
- Constantes (cadenas que contienen grupos de caracteres):

`string.ascii_letters`	lowercase + uppercase
`string.ascii_lowercase`	letras minúsculas (a-z)
`string.ascii_uppercase`	letras mayúsculas (A-Z)
`string.digits`	dígitos decimales (0-9)
`string.hexdigits`	dígitos hexadecimales (0-9a-fA-F)
`string.octdigits`	dígitos octales (0-7)
`string.punctuation`	signos de puntuación
`string.printable`	letras+dígitos+puntuación+blancos
`string.whitespace`	caracteres tomados como blancos

Cadenas: ejercicios propuestos

1. Escribir una función que retorne True si ninguno de los caracteres de una cadena s1 aparece 2 o más veces en otra cadena s2, y False en caso contrario.

2. Escribir una función swap_pos(s,l) que tome como entrada una cadena s y una lista l de 2*n índices enteros (n>0) tales que 0 <= l[i] < len(s) para todo i. La función deberá retornar la cadena que resulta al intercambiar los caracteres de las posiciones l[2*i] y l[2*i+1] para i=0,1,…,n-1 (estrictamente en ese orden). Por ejemplo, si s='tienes dos horas' y l=[4,0,5,8,11,4] (aquí n=3, es decir, hay 3 pares de índices), la función retornará: 'eienho dss toras'. ¿Cómo podría recuperarse la cadena original a partir de la cadena transformada?

3. Suponiendo que s es una cadena formada sólo por letras minúsculas, escribir dos funciones lrot(s,n) y rrot(s,n) que devuelvan la cadena resultante de rotar las letras de s n posiciones a izquierda o n posiciones a derecha en el alfabeto, respectivamente. Por ejemplo, *rotando* la letra 'b' 3 posiciones a izquierda se obtiene la letra 'y'. Y viceversa, rotando la letra 'y' 3 posiciones a derecha se obtiene la letra 'b'.

Cadenas: ejercicios propuestos

4. Escribir una función `find_all(t, s, overlaps=True)` que retorne una lista con todas las posiciones donde aparece la subcadena `t` dentro de la cadena `s`. Si `overlaps=True` (valor por defecto), se permiten solapamientos entre diferentes ocurrencias de la subcadena. Si `overlaps=False`, no se permiten solapamientos entre diferentes ocurrencias de la subcadena. Escribir un programa que ponga a prueba la función definida, usando la cadena `s="abbbbabbababababb"` y las subcadenas `t="abb"`, `t="bbb"` y `t="aba"`.

5. Escribir una función `my_split(s [, sep_seq [, max_splits]])` que tome como entrada una cadena `s` y, utilizando como separadores los elementos de la secuencia `sep_seq`, retorne la lista de cadenas resultante, haciendo como máximo `max_splits` separaciones. Los argumentos `sep_seq` y `max_splits` son opcionales. El argumento `sep_seq` podría ser una cadena de caracteres, tal que cada carácter de la cadena funcione como separador, o una lista de cadenas, tal que cada cadena funcione como separador. El valor por defecto de `sep_seq` será `' \n\t'` y el de `max_splits` será None (en cuyo caso se realizarán todas las separaciones posibles). En cualquier caso, si en la cadena `s` se encuentra una secuencia contigua de separadores, ésta se tratará como un único separador.