CUSOL (Comunidad Universitaria de Software Libre)

Cadenas y Listas

Una **cadena** es una secuencia de caracteres. Se puede acceder a cada uno de sus componentes con [ ]. Ej:

```
>>>fruta = “fresa”
>>> fruta[2]
'e'
>>> fruta[-1]
'a'
>>>len(fruta)
```

De esta manera se puede acceder al último caracter.

El **método** len**,** nos dará la longitud de la cadena.

Las cadenas son **inmutables**.

Cadenas y Listas

Dos maneras de hacer lo mismo recorrido con `for` y `while`:

```
index = 0
while index < len(fruit):
    letter = fruit[index]
    print letter
    index = index + 1
```

```
for char in fruit:
    print char
```

Cadenas y Listas

También podemos segmentar nuestras cadenas usando [] similar a cuando obtenemos un solo carácter.

```
>>> s = 'Monty Python'
>>> print s[0:5]
Monty
>>> print s[6:12]
Python
```

```
>>> fruit = 'banana'
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
```

```
>>> fruit = 'banana'
>>> fruit[0:5:2]
'bnn'
```

Cadenas y Listas

El operador in nos permite saber si un elemento se encuentra en una lista o cadena. Ej:

```
>>>fruta = "fresa"
>>> "r" in fruta
True
>>> "z" in fruta
False
```

Ejercicio: Escriba una función que tome una letra y una palabra o frase y devuelve el número de ocurrencias de la letra.

Métodos para cadenas

Más …

```
>>> word = 'banana'
>>> new_word = word.upper()
>>> print new_word
BANANA
```

```
>>> word = 'banana'
>>> index = word.find('a')
>>> print index
1
```

```
>>> word.find('na')
2
```

Revisa la documentación para más métodos con cadenas !

https://docs.python.org/2/library/stdtypes.html#string-methods

Tarea

Escribir una función llamada “reves” que tome dos palabras y verifique si una es la otra al revés.

Ejemplo: pots, stop → Esto es cierto, retorna True

hodor, door → Esto no. Retorna False

Tip: Usa 2 índices para recorrer (para esto usas un while o for) una palabra de manera normal y la otra de atrás hacia adelante.

Tip: Puedes descartar automáticamente dos palabras si no tienen la misma longitud pues ahorras tiempo al no hacer ciclos ;)

Listas

Son secuencias, pero sus elementos pueden ser de diferentes tipos.

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> numbers = [17, 123]
>>> empty = []
>>> print cheeses, numbers, empty
['Cheddar', 'Edam', 'Gouda'] [17, 123] []
```

Incluso una lista puede ser elemento de otra lista!

```
['spam', 2.0, 5, [10, 20]]
```

Listas

A diferencia de las cadenas, las listas **sí son mutables.**

```
>>>lista = [“Camila”, 12, “estudiante”]
>>>lista[0]  #Se pueden acceder por indices
“Camila”
>>>lista[0] = “Monica” #los indices siempre son numeros enteros
>>>print lista
lista = [“Monica”, 12, “estudiante”]
```

Listas

El operador **in** se puede usar igual que en el caso de las cadenas para verificar la pertenencia de un valor a una lista.

```
>>>“Camila” in lista
False
```

Operaciones y métodos en listas

+ → Concatena listas

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 6]
```

***→ Repite una lista**

```
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Listas: Métodos

Segmentar

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3] = ['x', 'y']
>>> print t
['a', 'x', 'y', 'd', 'e', 'f']
```

Agregar elemento: append

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print t
['a', 'b', 'c', 'd']
```

Eliminar elemento

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> print t
['a', 'c']
>>> print x
b
```

Ejercicios

1)Escribe una función llamada "elimina_duplicados" que tome una lista y devuelva una nueva lista con los elementos únicos de la lista original. No tienen porque estar en el mismo orden.

2)Escriba una función que tome una lista de números y devuelva la suma acumulada, es decir, una nueva lista donde el primer elemento es el mismo, el segundo elemento es la suma del primero con el segundo, el tercer elemento es la suma del resultado anterior con el siguiente elemento y así sucesivamente. Por ejemplo, la suma acumulada de [1,2,3] es [1, 3, 6].

de http://www.pythondiario.com/2014/09/ejercicios-de-listas-en-python.html

Ejercicios

3) Para comprobar si una palabra está en una lista se puede utilizar el operador "in", pero sería una búsqueda lenta, ya que busca a través de las palabras en orden.

Debido a que las palabras están en orden alfabético, podemos acelerar las cosas con una búsqueda de bisección (también conocida como búsqueda binaria), que es similar a lo que haces cuando buscas una palabra en el diccionario. Comenzamos por el centro y comprobamos si la palabra que buscamos está antes o después del centro. Si está antes, se busca solo en la primera mitad, si está después se busca en la otra mitad de la lista. Con esto reduciremos el tiempo de búsqueda

Escribir una función llamada "bisect" que tome una lista ordenada y una palabra como objetivo, y nos devuelva el índice en el que se encuentra en la lista, en caso de no aparecer en la lista devuelve "No se encontró la palabra".

de http://www.pythondiario.com/2014/09/ejercicios-de-listas-en-python.html