

# Introducció a Python: Sessió 3: Dades (2) Seqüències i maps vistos com a Objectes

Capitalize

```
>>> cadena='ana'
>>> print cadena.capitalize()
Ana
>>> print cadena
ana
>>>
```

Upper

```
>>> cadena1='ana'
>>> cadena2=cadena1.upper()
>>> print cadena1
ana
>>> print cadena2
ANA
>>>
```

Lower

```
>>> cadena1='ANA'
>>> print cadena1
ANA
>>> cadena2=cadena1.lower()
>>> print cadena2
ana
>>>
```

Isupper



```
1 #!/usr/bin/python
2 #-*- coding: utf-8 -*-
3 # > <
4
5 cadena='ANA'
6 if cadena.isupper():
7     print 'La cadena '+cadena+' esta toda en mayusculas'
8
```

En la primera llamada al ejercicio no printa nada, porque no he puesto el nombre todo en mayúsculas para probar.

Islower



```
1 #-*- coding: utf-8 -*-
2 # > <
3
4 cadena='ana'
5 if cadena.islower():
6     print 'La cadena '+cadena+' esta toda en minuscula'
7
8
```

Te printa la cadena cuando esta toda en minúscula

Isdigit()



Te devuelve la cadena en caso de que todos los dígitos sean números

Isalpha()



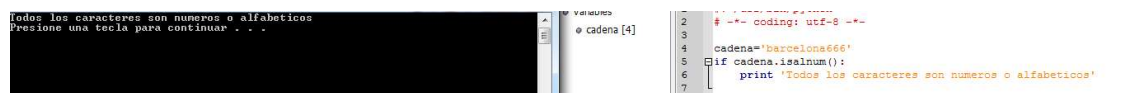
Sale que no todos los caracteres son alfabéticos porque hay un espacio. (en el isalpha hace falta el else)

Isspace()



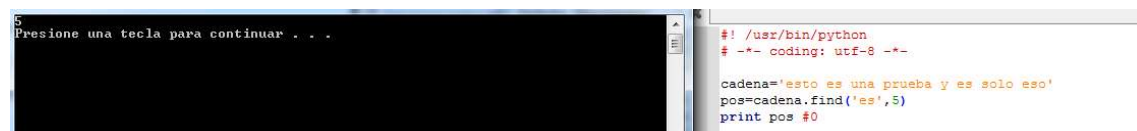
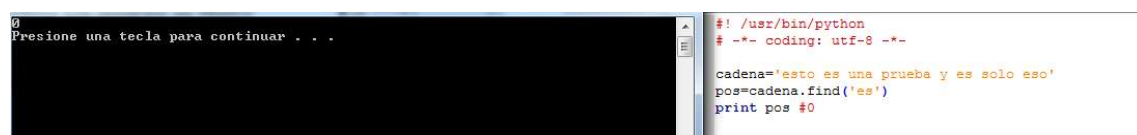
Imprime la frase si la cadena son solo espacios

Isalnum()

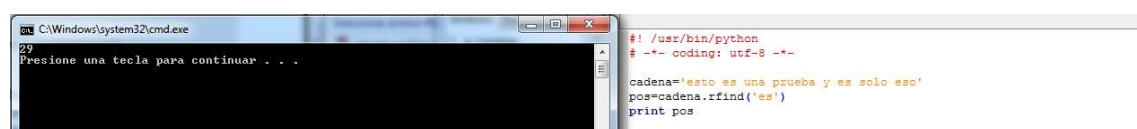


Imprime la frase si la cadena son números o letras.

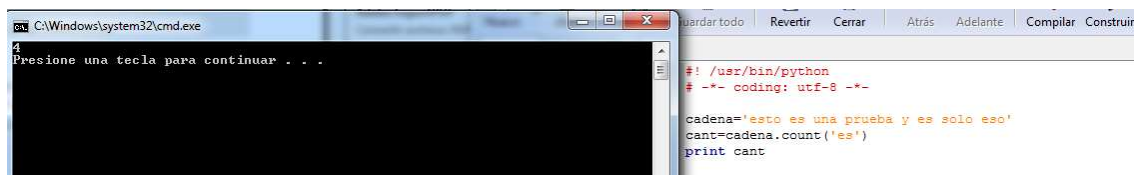
Probando la función find



Probando la función rfind



## Probando count



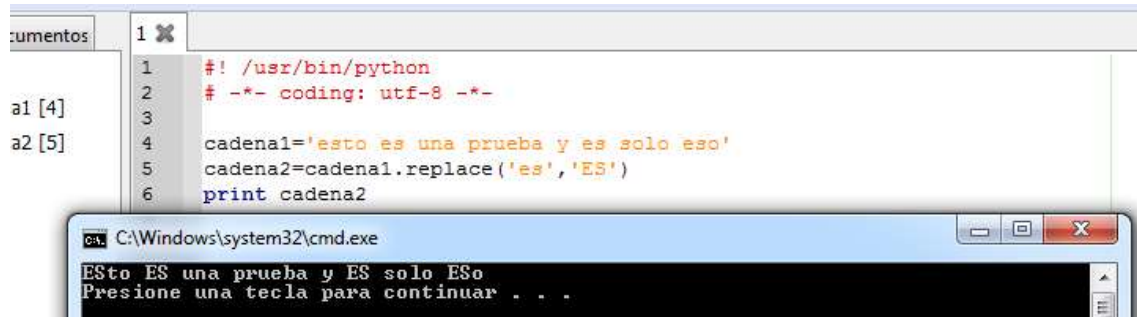
```
C:\Windows\system32\cmd.exe
4
Presione una tecla para continuar . . .

#! /usr/bin/python
# -*- coding: utf-8 -*-

cadena='esto es una prueba y es solo eso'
cant=cadena.count('es')
print cant
```

Sale 4 porque busca por las letras 'e-s' no por la palabra es.

## Probando replace



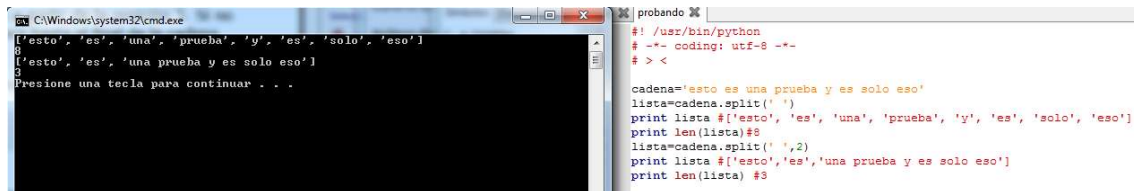
```
documentos 1 X
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  cadena1='esto es una prueba y es solo eso'
5  cadena2=cadena1.replace('es','ES')
6  print cadena2

C:\Windows\system32\cmd.exe
Esto ES una prueba y ES solo ESo
Presione una tecla para continuar . . .
```

## Probando Split (lo hace de izq a derecha)

El primero indica que divida la cadena en ' ', y el segundo indica que las dos primeras palabras la separe individualmente y lo que sobre en otra parte.

El print len(lista) imprime las partes que se han creado de la cadena.



```
C:\Windows\system32\cmd.exe
['esto', 'es', 'una', 'prueba', 'y', 'es', 'solo', 'eso']
['esto', 'es', 'una prueba y es solo eso']
Presione una tecla para continuar . . .

probando X
#! /usr/bin/python
# -*- coding: utf-8 -*-
# > <

cadena='esto es una prueba y es solo eso'
lista=cadena.split(' ')
print lista #['esto', 'es', 'una', 'prueba', 'y', 'es', 'solo', 'eso']
print len(lista)#8
lista=cadena.split(' ',2)
print lista #['esto','es','una prueba y es solo eso']
print len(lista) #3
```

## Probando rsplit (lo hace de derecha a izquierda)

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  # > <
4
5  cadena='esto es una prueba y es solo eso'
6  lista=cadena.rsplit(' ')
7  print lista
8  print len(lista)
9  lista=cadena.rsplit(' ',2)
10 print lista
11 print len(lista)
```

```
C:\Windows\system32\cmd.exe
['esto', 'es', 'una', 'prueba', 'y', 'es', 'solo', 'eso']
8
['esto es una prueba y es', 'solo', 'eso']
3
Presione una tecla para continuar . . .
```

Probando splitlines()

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

mensaje="""Primer linea
          Segunda linea
          Tercera linea
          Cuarta linea"""
lista=mensaje.splitlines()
print lista
```

```
C:\Windows\system32\cmd.exe
['Primer linea', '\t\t\t Segunda linea', '\t\t\t Tercera linea', '\t\t\t Cua
rta linea']
Presione una tecla para continuar . . .
```

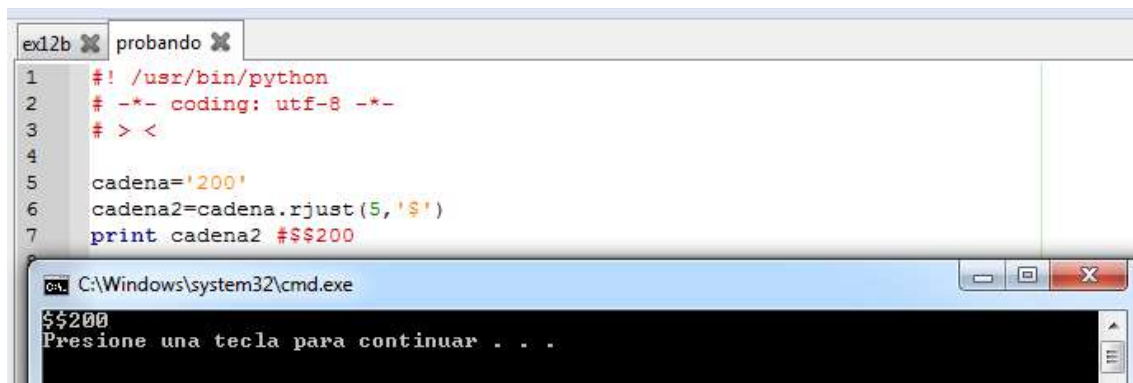
Probando swapcase

Cambia mayúsculas por minúsculas y viceversa

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  # > <
4
5  cadena1='Sistema de Facturacion'
6  cadena2=cadena1.swapcase()
7  print cadena2
```

```
C:\Windows\system32\cmd.exe
sistema de facturacion
Presione una tecla para continuar . . .
```

Rjust



The image shows a code editor window with a file named 'probando.py' and a terminal window below it. The Python script defines a string 'cadena' with the value '200', right-aligns it by 5 characters using 'rjust', and prints the result. The terminal shows the output '\$\$200' and a prompt to press a key to continue.

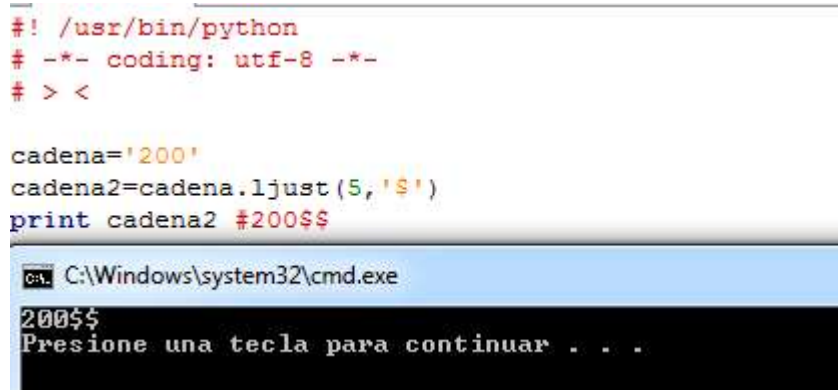
```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  # > <
4
5  cadena='200'
6  cadena2=cadena.rjust(5,'$')
7  print cadena2 ##$200
```

```
C:\Windows\system32\cmd.exe
$$200
Presione una tecla para continuar . . .
```

Llena el lado izq con caracteres según el segundo parámetro indicado.

Ljust

Llena el lado derecho con caracteres según el parámetro indicado



The image shows a code editor window with a file named 'probando.py' and a terminal window below it. The Python script defines a string 'cadena' with the value '200', left-aligns it by 5 characters using 'ljust', and prints the result. The terminal shows the output '200\$\$' and a prompt to press a key to continue.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

cadena='200'
cadena2=cadena.ljust(5,'$')
print cadena2 ##200$$
```

```
C:\Windows\system32\cmd.exe
200$$
Presione una tecla para continuar . . .
```

Center


Centramos el string

```

#! /usr/bin/python
# -*- coding: utf-8 -*-
# > <

cadena='200'
cadena2=cadena.center(5,'$')
print cadena2

```



C:\Windows\system32\cmd.exe

\$200\$

Presione una tecla para continuar . . .

Append(solo añade un elemento)


Añade un elemento al final de la lista

```

#! /usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','luis']
lista.append('carlos')
print lista

```



C:\Windows\system32\cmd.exe

['juan', 'ana', 'luis', 'carlos']

Presione una tecla para continuar . . .

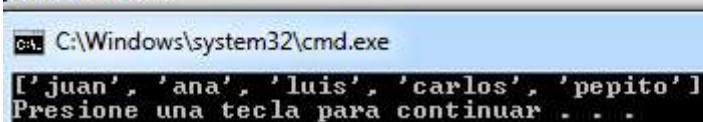
Extend (añade todos los elementos que hay uno a uno)

```

#! /usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','luis']
lista.extend(['carlos','pepito'])
print lista

```



C:\Windows\system32\cmd.exe

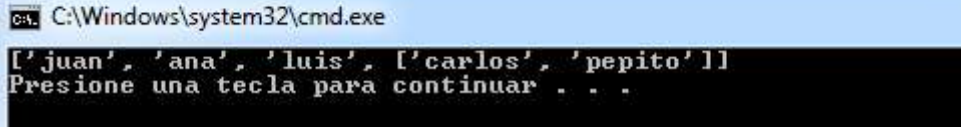
['juan', 'ana', 'luis', 'carlos', 'pepito']

Presione una tecla para continuar . . .

Lista append con mas de un elemento

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','luis']
lista.append(['carlos','pepito'])
print lista
```




```
C:\Windows\system32\cmd.exe
['juan', 'ana', 'luis', ['carlos', 'pepito']]
Presione una tecla para continuar . . .
```

Los dos últimos elementos los añado como una lista, con lo cual tenemos una lista con 3 elementos mas otra lista dentro de la lista principal.

Insert posición, elemento

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','luis']
lista.insert(1,'carlos')
print lista
```




```
C:\Windows\system32\cmd.exe
['juan', 'carlos', 'ana', 'luis']
Presione una tecla para continuar . . .
```

Pop([posición])

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','luis','marcos']
elemento=lista.pop()
print elemento
print lista
print lista.pop(1)
print lista
```



```
C:\Windows\system32\cmd.exe
marcos
['juan', 'ana', 'luis']
ana
['juan', 'luis']
Presione una tecla para continuar . . .
```

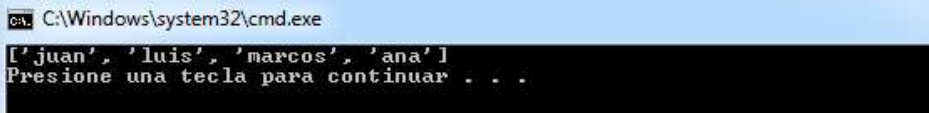


El elemento=lista.pop() guarda la ultima cadena de la lista, la printa, luego el siguiente print imprime lo que queda en la lista, el lista.pop(1) imprime y a la vez saca la cadena en posición 1 que es ana y después printa la lista una vez lista.pop ha hecho su función de sacar la cadena ana.

Remove(elemento)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','luis','marcos','ana']
lista.remove('ana')
print lista
```



C:\Windows\system32\cmd.exe

```
['juan', 'luis', 'marcos', 'ana']
Presione una tecla para continuar . . .
```

Borra el primer nodo que coincide con la información que le pasamos como parámetro.

Count(elemento)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','luis','marcos','ana']
print lista.count('ana')
```



C:\Windows\system32\cmd.exe

```
2
Presione una tecla para continuar . . .
```

Retorna la cantidad de veces que se repite la información que pasamos por parámetro.

Index(elemento,[inicio],[fin])

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','luis','marcos','ana']
print lista.index('ana')
```



C:\Windows\system32\cmd.exe

```
1
Presione una tecla para continuar . . .
```

Retorna la primera posición donde se encuentra el primer parámetro en la lista

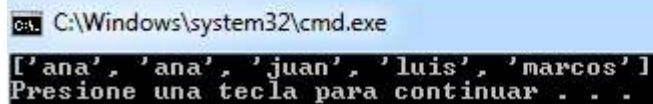


Sort()

Ordena de menor a mayor

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','luis','marcos','ana']
lista.sort()
print lista
```



```
C:\Windows\system32\cmd.exe
['ana', 'ana', 'juan', 'luis', 'marcos']
Presione una tecla para continuar . . .
```

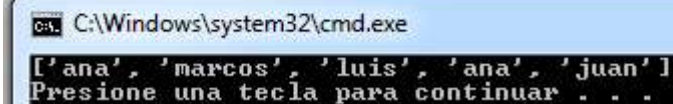
En este caso ordena alfabéticamente

Reverse()

Invierte el orden de los elementos de la lista

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','luis','marcos','ana']
lista.reverse()
print lista
```

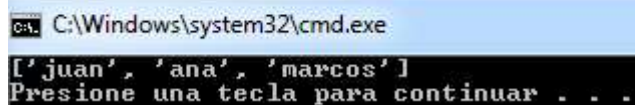


```
C:\Windows\system32\cmd.exe
['ana', 'marcos', 'luis', 'ana', 'juan']
Presione una tecla para continuar . . .
```

Borrar elementos de la lista

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','luis','marcos']
del lista[2]
print lista
```



```
C:\Windows\system32\cmd.exe
['juan', 'ana', 'marcos']
Presione una tecla para continuar . . .
```

Borrar elementos desde la pos 2 hasta la 4.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','carlos','maria','pedro']
del lista[2:4]
print lista
```

```
CA. C:\Windows\system32\cmd.exe
['juan', 'ana', 'pedro']
Presione una tecla para continuar . . .
```

Borrar desde la 2 hasta el final

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','carlos','maria','pedro']
del lista[2:]
print lista
```

```
CA. C:\Windows\system32\cmd.exe
['juan', 'ana']
Presione una tecla para continuar . . .
```

Borrar todos desde el principio hasta la posición 3 sin incluirla

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

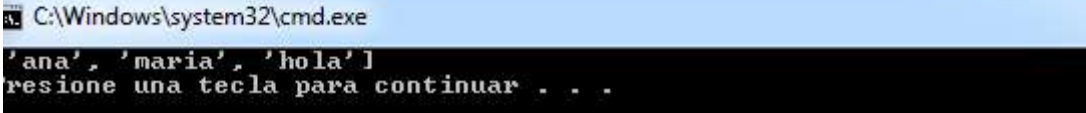
lista=['juan','ana','carlos','maria','pedro']
del lista[:3]
print lista
```

```
CA. C:\Windows\system32\cmd.exe
['maria', 'pedro']
Presione una tecla para continuar . . .
```

El primero no lo imprime, el siguiente si, el siguiente no, el siguiente si y asi todo el rato,

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

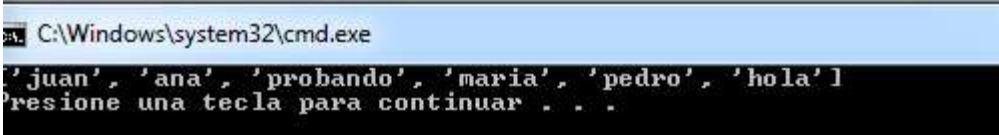
lista=['juan','ana','carlos','maria','pedro','hola']
del lista[::2]
print lista
```



Cambiar de valor de algún nodo

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','carlos','maria','pedro','hola']
lista[2]='probando'
print lista
```



Conoce la cantidad de elementos actuales

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

lista=['juan','ana','carlos','maria','pedro','hola']
print len(lista)
```




Keys()

Devolverá lo que este a la izquierda de los dos puntos(las claves del diccionario)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

diccionario={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
lista=diccionario.keys()
print lista
```

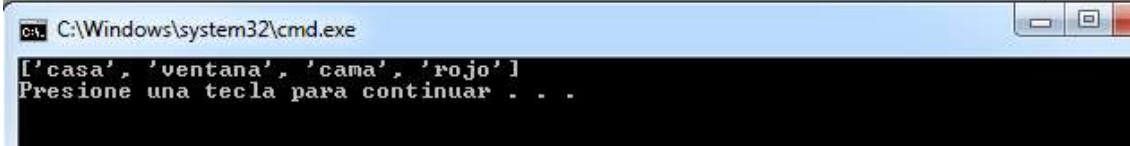


## Values()

Retorna una lista con todos los valores almacenados en el diccionario

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

diccionario={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
lista=diccionario.values()
print lista
```



## Items()

Retorna una lista que contiene en cada nodo la clave y el valor que tiene en el diccionario.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

diccionario={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
lista=diccionario.items()
print lista
```



## Pop(clave,[valor])

Extraer el valor de la clave que pasamos como parámetro y borra el elemento del diccionario.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

diccionario={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
valor=diccionario.pop('window')
print valor
print diccionario
```

C:\Windows\system32\cmd.exe

ventana  
{'house': 'casa', 'bed': 'cama', 'red': 'rojo'}  
Presione una tecla para continuar . . .

Has\_key retorna true si la clave se encuentra en el diccionario y falso en caso contrario

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

diccionario={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
if diccionario.has_key('love'):
    print 'Si tiene la clave buscada'
else:
    print 'No existe la clave buscada'
```

C:\Windows\system32\cmd.exe

No existe la clave buscada  
Presione una tecla para continuar . . .

Clear()

Elimina todos los elementos del diccionario

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

diccionario={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
diccionario.clear()
print diccionario
```

C:\Windows\system32\cmd.exe

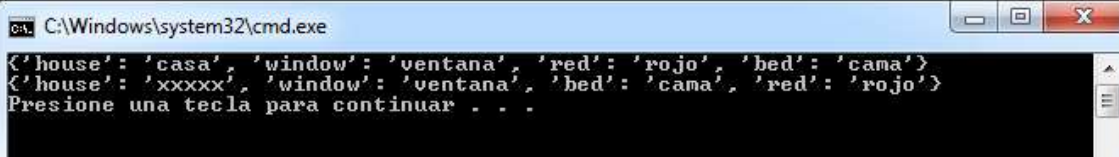
{}  
Presione una tecla para continuar . . .

Copy()

Generamos una copia idéntica del diccionario actual en otra parte de memoria

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

diccionario1={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
diccionario2=diccionario1.copy()
print diccionario2
diccionario1['house']='xxxxx'
print diccionario1
```



```
<'house': 'casa', 'window': 'ventana', 'red': 'rojo', 'bed': 'cama'>
<'house': 'xxxxx', 'window': 'ventana', 'bed': 'cama', 'red': 'rojo'>
Presione una tecla para continuar . . .
```

### Popitem()

Retorna un elemento del diccionario y lo elimina. Como no hay un sentido de orden en el diccionario se extrae uno al azar.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

diccionario1={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
elemento=diccionario1.popitem()
print elemento
print diccionario1
```




```
<'house', 'casa'>
<'window': 'ventana', 'bed': 'cama', 'red': 'rojo'>
Presione una tecla para continuar . . .
```

### Update(diccionario2)

Modifica el diccionario principal y si una clave esta repetida se modifica su valor

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

diccionario1={'uno':'1','dos':'2','tres':'3333'}
diccionario2={'tres':'3','cuatro':'4','cinco':'5'}
diccionario1.update(diccionario2)
print diccionario1
```




```
'cuatro': '4', 'cinco': '5', 'dos': '2', 'tres': '3', 'uno': '1'>
Presione una tecla para continuar . . .
```



## Borrado de elementos del diccionario

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

diccionario={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
del diccionario['house']
print diccionario
```



```
<'window': 'ventana', 'bed': 'cama', 'red': 'rojo'>
Presione una tecla para continuar . . .
```

## Modificación y creación de elementos del diccionario

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <


diccionario={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
diccionario['red']='colorado'
diccionario['blue']='azul'
print diccionario
```



```
<'blue': 'azul', 'house': 'casa', 'window': 'ventana', 'bed': 'cama', 'red': 'colorado'>
Presione una tecla para continuar . . .
```

## Conocer la cantidad de elementos actual

```
diccionario={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
print len(diccionario)
```



```
4
Presione una tecla para continuar . . .
```



## 1. LLISTES

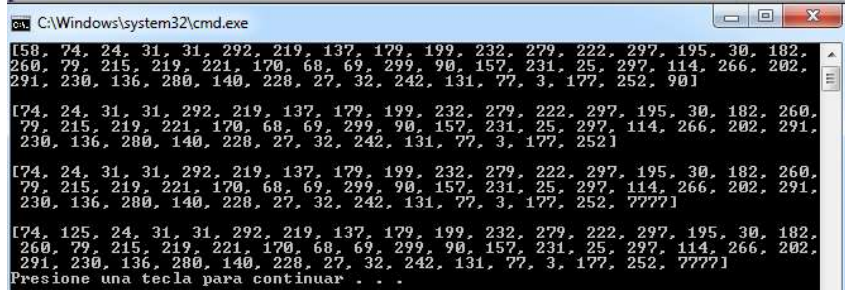
```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <
import random

lista=[]
for x in range(1,50):
    valor=random.randint(1,300)
    lista.append(valor)
print lista
print

del lista[0]
del lista[-1]
print lista
print

suma=0
for x in range(1,len(lista)):
    suma=suma+lista[x]
lista.append(suma)
print lista
print

lista.insert(1,125)
print lista
```



```
C:\Windows\system32\cmd.exe
[58, 74, 24, 31, 31, 292, 219, 137, 179, 199, 232, 279, 222, 297, 195, 30, 182, 260, 79, 215, 219, 221, 170, 68, 69, 299, 90, 157, 231, 25, 297, 114, 266, 202, 291, 230, 136, 280, 140, 228, 27, 32, 242, 131, 77, 3, 177, 252, 90]
[74, 24, 31, 31, 292, 219, 137, 179, 199, 232, 279, 222, 297, 195, 30, 182, 260, 79, 215, 219, 221, 170, 68, 69, 299, 90, 157, 231, 25, 297, 114, 266, 202, 291, 230, 136, 280, 140, 228, 27, 32, 242, 131, 77, 3, 177, 252]
[74, 24, 31, 31, 292, 219, 137, 179, 199, 232, 279, 222, 297, 195, 30, 182, 260, 79, 215, 219, 221, 170, 68, 69, 299, 90, 157, 231, 25, 297, 114, 266, 202, 291, 230, 136, 280, 140, 228, 27, 32, 242, 131, 77, 3, 177, 252, 7777]
[74, 125, 24, 31, 31, 292, 219, 137, 179, 199, 232, 279, 222, 297, 195, 30, 182, 260, 79, 215, 219, 221, 170, 68, 69, 299, 90, 157, 231, 25, 297, 114, 266, 202, 291, 230, 136, 280, 140, 228, 27, 32, 242, 131, 77, 3, 177, 252, 7777]
Presione una tecla para continuar . . .
```

Tras realizar los cambios

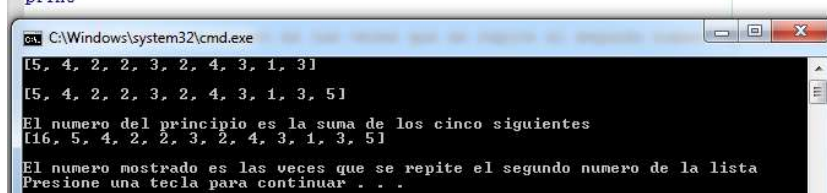
```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <
import random

lista=[]
for x in range(1,11):
    valor=random.randint(1,5)
    lista.append(valor)
print lista
print

lista.insert(11,5)
print lista
print

suma=0

print "El numero del principio es la suma de los cinco siguientes"
for x in range(0,5):
    suma=suma+lista[x]
lista.insert(0,suma)
print lista
print
```



```
C:\Windows\system32\cmd.exe
[5, 4, 2, 2, 3, 2, 4, 3, 1, 3]
[5, 4, 2, 2, 3, 2, 4, 3, 1, 3, 5]
El numero del principio es la suma de los cinco siguientes
[16, 5, 4, 2, 2, 3, 2, 4, 3, 1, 3, 5]
El numero mostrado es las veces que se repite el segundo numero de la lista
Presione una tecla para continuar . . .
```

## Ejercicio 2

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

#asigna un valor a las frutas
frutas={'manzanas':1.60,'peras':1.90,'bananas':0.95}
print frutas
print


#añade una fruta mas y el len indicara la cantidad de frutas que hay
frutas['naranjas']=2.50
print len(frutas)
print

#el del borra la fruta naranjas y el for con el keys devuelve una lista con todas la:
del frutas['naranjas']
for x in frutas.keys():
    print x
print

#escribe el valor de las frutas
for x in frutas.values():
    print x
print

#escribira el nombre de la clave y el valor de la lista frutas. Acompañado de un - a:
for (clave,valor) in frutas.items():
    print clave+' '+str(valor)+' - '
print

#borra el diccionario frutas
frutas.clear()
print frutas
```



```
C:\Windows\system32\cmd.exe
{'peras': 1.9, 'bananas': 0.95, 'manzanas': 1.6}
4
peras
bananas
manzanas
1.9
0.95
1.6
peras 1.9 -
bananas 0.95 -
manzanas 1.6 -
{}
Presione una tecla para continuar . . .
```

### Ejercicio 3

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

ten_things = "Apples Oranges Crows Telephone Light Sugar"

print "Wait there are not 10 things in that list. Let's fix that."


stuff = ten_things.split(' ')
more_stuff = ["Day", "Night", "Song", "Frisbee", "Corn", "Banana", "Girl", "Boy"]

while len(stuff) != 10:
    next_one = more_stuff.pop()
    print "Adding: ", next_one
    stuff.append(next_one)
    print "There are %d items now." % len(stuff)

print "There we go: ", stuff

print "Let's do some things with stuff."

print stuff[1]
print stuff[-1]
print stuff.pop()
print ' '.join(stuff)
print '#'.join(stuff[3:5])
```



```
C:\Windows\system32\cmd.exe
Wait there are not 10 things in that list. Let's fix that.
Adding: Boy
There are 7 items now.
Adding: Girl
There are 8 items now.
Adding: Banana
There are 9 items now.
Adding: Corn
There are 10 items now.
There we go: ['Apples', 'Oranges', 'Crows', 'Telephone', 'Light', 'Sugar', 'Boy', 'Girl', 'Banana', 'Corn']
Let's do some things with stuff.
Oranges
Corn
Corn
Apples Oranges Crows Telephone Light Sugar Boy Girl Banana
Telephone#Light
Presione una tecla para continuar . . .
```

Primero hacemos el print normal, el `stuff = ten_things.split` va a separar la lista `stuff` con ' ', el `more_stuff` es una lista de palabras que podremos añadir a `ten_things`, en el `while` lo que hace es mientras no haya 10 strings en la lista `stuff`, hara el `pop`(coger el ultimo) de la lista `more_stuff` y lo añadirá, volverá a imprimir la cantidad de strings que hay. Una vez tengamos 10 pondra `there we go` printara la lista `stuff`(10 strings).

En los últimos print imprimirá los que estén en esa posición, la uno es oranges, la -1 es la ultima es decir corn, `pop` elige el ultimo es decir también `pop`, el `join` imprime todos separados por un espacio que es lo indicado, y en el otro `join` nos separa con # las cadenas indicadas.

El método `join` devuelve una cadena en la que los elementos de la cadena son unidos mediante un separador.

## Ejercicio 4

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# > <

# create a mapping of state to abbreviation

states = {
    'Oregon': 'OR',
    'Florida': 'FL',
    'California': 'CA',
    'New York': 'NY',
    'Michigan': 'MI'
}

# create a basic set of states and some cities in them
cities = {
    'CA': 'San Francisco',
    'MI': 'Detroit',
    'FL': 'Jacksonville'
}

# add some more cities
cities['NY'] = 'New York'
cities['OR'] = 'Portland'
# print out some cities
print '-' * 10
print "NY State has: ", cities['NY']
print "OR State has: ", cities['OR']
# print some states
print '-' * 10
print "Michigan's abbreviation is: ", states['Michigan']
print "Florida's abbreviation is: ", states['Florida']
# do it by using the state then cities dict
print '-' * 10
print "Michigan has: ", cities[states['Michigan']]
print "Florida has: ", cities[states['Florida']]
# print every state abbreviation
print '-' * 10
for state, abbrev in states.items():
    print "%s is abbreviated %s" % (state, abbrev)
# print every city in state
print '-' * 10
for abbrev, city in cities.items():
    print "%s has the city %s" % (abbrev, city)
# now do both at the same time
print '-' * 10
for state, abbrev in states.items():
    print "%s state is abbreviated %s and has city %s" % (state, abbrev, cities[abbrev])
print '-' * 10
# safely get a abbreviation by state that might not be there
state = states.get('Texas')
if not state:
    print "Sorry, no Texas."

# get a city with a default value
city = cities.get('TX', 'Does not exist')
print "The city for the state 'TX' is :%s" % city
```

```
-----
NY State has: New York
OR State has: Portland

-----
Michigan's abbreviation is: MI
Florida's abbreviation is: FL

-----
Michigan has: Detroit
Florida has: Jacksonville

-----
California is abbreviated CA
Michigan is abbreviated MI
New York is abbreviated NY
Florida is abbreviated FL
Oregon is abbreviated OR
-----
FL has the city Jacksonville
CA has the city San Francisco
MI has the city Detroit
OR has the city Portland
NY has the city New York
-----
California state is abbreviated CA and has city San Francisco
Michigan state is abbreviated MI and has city Detroit
New York state is abbreviated NY and has city New York
Florida state is abbreviated FL and has city Jacksonville
Oregon state is abbreviated OR and has city Portland
-----
Sorry, no Texas.
The city for the state 'TX' is :Does not exist
Presione una tecla para continuar . . .
```

Hay una serie de diccionarios en los que se van guardando información.

En los print se llama a los valores de los diccionarios.

Hay un for que printa todas las abreviaciones y las ciudades que tienen.

También hay un for que printa los estados, abreviaciones y las ciudades que tienen.

También hay un print “fail” que es porque Texas no está en el diccionario.

La prueba de asignar una ciudad a un estado que ya tiene ciudades es esta.

```
#create a mapping of state to abbreviation
states = {
    'Oregon': 'OR',
    'Florida': 'FL',
    'California': 'CA',
    'New York': 'NY',
    'Michigan': 'MI'
}

#create a basic set of states and some cities
cities = {
    'CA': 'San Francisco',
    'MI': 'Detroit',
    'FL': 'Jacksonville',
    'FL': 'Lluisville'
}

#add some more cities
Michigan has: Detroit
Florida has: Lluisville

California is abbreviated CA
Michigan is abbreviated MI
New York is abbreviated NY
Florida is abbreviated FL
Oregon is abbreviated OR

FL has the city Lluisville
CA has the city San Francisco
MI has the city Detroit
OR has the city Portland
NY has the city New York

California state is abbreviated CA and has city San Francisco
Michigan state is abbreviated MI and has city Detroit
New York state is abbreviated NY and has city New York
Florida state is abbreviated FL and has city Lluisville
Oregon state is abbreviated OR and has city Portland
```

Salí Lluisville porque es la última añadida en “FL”, se pone encima de la información que hubiera anteriormente en FL.