Introducció a Python: Sessió3:Dades(2) Seqüències i maps vistos com a Objectes

1. Introducció

En Python tot és un objecte. Les estructures de dades complexes també ho són i per poder-hi treballar cal conèixer els mètodes predefinits.

2. Mètodes de les cadenes o strings

<Text pendent d'el·laboració i revisió>

capitalize()
Retorna otra cadena con el primer caracter en mayúsculas.

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

Es importante notar que **no se modifica el contenido** de la variable 'cadena'. El método capitalize() **retorna otra cadena** con el primero en mayúsculas.

upper()
Retorna otra cadena con todos los caracteres convertidos a mayúscula.

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

lower()
Retorna otra cadena con todos los caracteres convertidos a minúsculas.

```
cadena1='Ana'
cadena2=cadena1.lower()
print cadena2                #ana
```

isupper()
Retorna True si todos los caracteres de la cadena están en mayúsculas.

```
cadena='ANA'
if cadena.isupper():
    print 'La cadena '+cadena+' esta toda en mayusculas'
```

islower()
Retorna True si todos los caracteres de la cadena están en minúsculas.

```
cadena1='ana'
if cadena.islower():
    print 'La cadena '+cadena+' esta toda en minusculas'
```

isdigit()
Retorna verdadero si todos los caracteres de la cadena son dígitos.

```
cadena='120'
if cadena.isdigit():
    print 'Todos los caracteres de la cadena son numeros'
```

Si al menos uno de los caracteres es distinto a un dígito retorna False. Inclusive si tiene el caracter punto.

isalpha()
Retorna True si todos los caracteres son alfabéticos.

```
cadena='Hola Mundo'
if cadena.isalpha():
    print 'Todos los caracteres de la cadena son del alfabeticos'
else:
    print 'No todos los caracteres de la cadena son del alfabeticos'
```

En el ejemplo se ejecuta el bloque del else ya que la cadena contiene un caracter de espacio.

isspace()
Retorna verdadero si todos los caracteres de la cadena son espacios en blanco

```
cadena='   '
if cadena.isspace():
    print 'Todos los caracteres de la cadena son espacios en blanco'
```

isalnum()
Retorna True si todos los caracteres de la cadena son números o caracteres alfabéticos.

```
cadena='cordoba2008'
if cadena.isalnum():
    print 'Todos los caracteres son numeros o alfabeticos'
```

find('cadena',[inicio],[fin])
Retorna la posición donde se encuentra el valor del primer parámetro en el string. Si no se encuentra retorna -1. Opcionalmente, se puede indicar como segundo y tercer parámetro las posiciones de inicio y final de búsqueda.

```
cadena='esto es una prueba y es solo eso'
pos=cadena.find('es')
print pos   #0
```

Retorna 0 ya que los dos primeros caracteres son la cadena 'es', es decir retorna la primera aparición del string.

```
cadena='esto es una prueba y es solo eso'
pos=cadena.find('es',5)
print pos   #5
```

En este otro ejemplo comenzamos la búsqueda a partir de la posición 5. Si no indicamos el tercer parámetro la búsqueda la hace hasta el final de la cadena.

rfind('cadena',[inicio],[fin])
Igual al método find con la diferencia que la búsqueda comienza desde el final.

```
cadena='esto es una prueba y es solo eso'
pos=cadena.rfind('es')
print pos  #29
```

count('cadena',[inicio],[fin])
Retorna la cantidad de veces que la cadena se repite en el string.

```
cadena='esto es una prueba y es solo eso'
cant=cadena.count('es')
print cant   #4
```

replace('cadena1','cadena2',[maximo])
Retorna un string remplazando todas las ocurrencias de cadena1 por cadena2. Podemos eventuamente indicar la cantidad máxima de remplazos.

```
cadena1='esto es una prueba y es solo eso'
cadena2=cadena1.replace('es','ES')
print cadena2  #ESto ES una prueba y ES solo ESo
```

split('caracter separador',[maximo])
El método split retorna una lista dividiendo el string por el caracter indicado en el primer parámetro. Podemos pasar un segundo parámetro indicando la cantidad de trozos a general, el último elemento de la lista almacena el resto del string.

```
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
```

rsplit('caracter separador',[maximo])
Semejante a split pero procesando desde el final del string. En caso de indicar maximo el primer elemento de la lista almacena el trozo restante.

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

splitlines()
Retorna una lista dividiendo el string con los retornos de carro contenidos en el mismo.

```
mensaje="""Primer linea
           Segunda linea
           Tercer linea
           Cuarta linea"""
lista=mensaje.splitlines()
print lista #['Primer linea', ' Segunda linea', ' Tercer linea', ' Cuarta
linea']
```

swapcase()
Retorna un string transformando los caracteres minúsculas a mayúsculas y los mayúsculas a minúsculas.

```
cadena1='Sistema de Facturacion'
cadena2=cadena1.swapcase()
print cadena2        #sISTEMA DE fACTURACION
```

rjust(ancho,caracter de relleno)
Retorna un string justificado a derecha y rellenando el lado izquierdo con caracteres según el segundo parámetro. La nueva cadena toma un largo indicado según el valor del primer parámetro.

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

ljust(ancho,caracter de relleno)
Similar a rjust con la salvedad que se justifica a derecha el string.

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

center(ancho,caracter de relleno)
El string original se centra.

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

3. Mètodes de les llistes

append(elemento)
El método append añade un elemento al final de la lista.

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

extend(elementos)
El método extend procesa la secuencia de elementos del parámetro y los añade uno a uno a la lista. La diferencia con el método append es que append siempre añade un único elemento a la lista y extend añade tantos elementos como tenga la secuencia.

```
lista=['juan','ana','luis']
lista.extend(['uno','dos'])
print lista  #['juan', 'ana', 'luis', 'uno', 'dos']
```

Ahora la lista tiene 5 elementos, es decir se añadieron 2 nuevas componentes a la lista.En cambio si utilizamos append el resultado es:

```
lista=['juan','ana','luis']
lista.append(['uno','dos'])
print lista  #['juan', 'ana', 'luis', ['uno', 'dos']]
Ahora la lista tiene cuatro elementos y el último elemento es una lista
también.
```

insert(posición,elemento)
El método insert añade un elemento en la posición que le indicamos en el primer parámetro.

```
lista=['juan','ana','luis']
lista.insert(0,'carlos')
print lista  #['carlos', 'juan', 'ana', 'luis']
```

En este ejemplo insertamos la cadena 'carlos' al principio de la lista, ya que pasamos la posición 0, no se borra el elemento que se encuentra en la posición 0 sino se desplaza a la segunda posición.
Si indicamos una posición que no existe porque supera a la posición del último elemento se inserta al final de la lista.

pop([posicion]
El método pop sin parámetro retorna y borra la información del último nodo de la lista. Se puede indicando la posición del nodo que se debe extraer.

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

remove(elemento)
Borra el primer nodo que coincide con la información que le pasamos como parámetro.

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

count(elemento)
Retorna la cantidad de veces que se repite la información que le pasamos como parámetro.

```
lista=['juan','ana','luis','marcos','ana']
print lista.count('ana')  #2
```

index(elemento,[inicio],[fin])
Retorna la primera posición donde se encuentra el primer parámetro en la lista. Podemos eventualmente indicarle a partir de que posición empezar a buscar y en que posición terminar la búsqueda. Si no lo encuentra en la lista genera un error: ValueError: list.index(x): x not in list

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

sort()
Ordena la lista de menor a mayor.

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

reverse()
Invierte el orden de los elementos de la lista.

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

Borrado de elementos de la lista
Si necesitamos borrar un nodo de la lista debemos utilizar el comando del que provee Python:

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

Podemos utilizar el concepto de porciones para borrar un conjunto de elementos de la lista:

Si queremos borrar los elementos de la posición 2 hasta la 3:

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

Si queremos borrar desde la 2 hasta el final:

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

Si queremos borrar todos desde el principio hasta la posición 3 sin incluirla:

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

Si queremos ir borrando de a uno de por medio:

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

Si necesitamos modificar el contenido de un nodo de la lista debemos utilizar el operador de asignación:

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

Conocer la cantidad de elementos actual

Se utiliza la función len():

```
lista=['juan','ana','luis','marcos']
print len(lista)  # 4
```

4. Mètodes dels diccionaris

keys()
Retorna una lista con todas las claves del diccionario.

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

values()
Retorna una lista con todos los valores almacenados en el diccionario.

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

items()
Retorna una lista que contiene en cada nodo una tupla con la clave y valor del diccionario.

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

pop(clave,[valor])
Extrae el valor de la clave que pasamos como parámetro y borra el elemento del diccionario. Genera un error si no se encuentra dicha clave, salvo que se inicialice un segundo parámetro que será el dato que retornará.

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

Si no encuentra la clave en el diccionario y hemos indicado un segundo parámetro al método pop será dicho valor el que retorne:

```
diccionario={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
valor=diccionario.pop('love','clave no encontrada')
print valor  # clave no encontrada
```

has_key(clave)
Retorna True si la clave se encuentra en el diccionario, False en caso contrario.

```
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'
```

clear()
Elimina todos los elementos del diccionario.

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

copy()
Se genera una copia idéntica del diccionario actual en otra parte de memoria.

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

Es importante hacer notar que no es lo mismo:

```
diccionario2=diccionario1
```

Con la asignación anterior no se esta creando un segundo diccionario sino que se tiene dos variables que referencian al mismo objeto.

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

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

update(diccionario2)
Modifica el diccionario actual agregando los elementos del diccionario2, si una clave está repetida se modifica su valor.

```
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'}
```

Borrado de elementos del diccionario
Si necesitamos borrar un elemento del diccionario, debemos utilizar el comando del que provee Python:

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

Modificación y creación de elementos del diccionario

Se hace mediante la asignación:

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

Conocer la cantidad de elementos actual

Se utiliza la función len():

```
diccionario={'house':'casa','red':'rojo','bed':'cama','window':'ventana'}
print len(diccionario)  # 4
```

4. Activitats de la Sessió 3

1. Llistes.

Crear un fitxer d'script amb el següent contingut d'exemple:

```
import random

#Crear una lista con 30 valores aleatorios comprendidos entre 1 y 300
lista=[]
for x in range(1,50):
    valor=random.randint(1,300)
    lista.append(valor)
print lista
print '\n'

#Borrar el primer y último elemento de la lista
del lista[0]
del lista[-1]
print lista
print '\n'

#Insertar un elemento al final con la suma de todos los elementos actuales
suma=0
for x in range(1,len(lista)):
    suma=suma+lista[x]
lista.append(suma)
print lista
print '\n'

#Insertar un elemento entre el primero y el segundo elemento de la lista con el valor 125.
lista.insert(1,125)
print lista
```

Executar-lo i analitzar com funciona.

Programar una variació del programa que

- crei una llista amb 10 valors aleatoris entre 1 i 5.
- inserti un element al final amb el màxim valor de la llista
- inserti un element al principi, amb la suma dels primers 5 elements
- imprimeixi les vegades que apareix a la llista el valor de segon element

2. Diccionaris.

Crear un fitxer d'script amb el següent contingut d'exemple:

```
# (COMENTAR)
frutas={'manzanas':1.60,'peras':1.90,'bananas':0.95}
print frutas
print '\n'

# (COMENTAR)
frutas['naranjas']=2.50
print len(frutas)
print '\n'

# (COMENTAR)
del frutas['naranjas']
for x in frutas.keys():
    print x
print '\n'

# (COMENTAR)
for x in frutas.values():
    print x
print '\n'

# (COMENTAR)
for (clave,valor) in frutas.items():
    print clave+' '+str(valor)+' - '
print '\n'

# (COMENTAR)
frutas.clear()
print frutas
```

Executar i comentar què es el que fa cada fragment de codi.

3. Llistes.

Crear un fitxer d'script amb el següent contingut d'exemple:

```
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] # whoa! fancy
print stuff.pop()
print ' '.join(stuff) # what? cool!
print '#'.join(stuff[3:5]) # super stellar!
```

Executar-lo i comprendre com funciona el programa.
Comentar l'efecte dels mètodes emprats.

Descriure l'efecte del mètode “join”.
(Veure http://www.tutorialspoint.com/python/string_join.htm)

4. Diccionaris.

Crear un fitxer d'script amb el següent contingut d'exemple:

```
# 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']

```

```
31  # do it by using the state then cities dict
32  print '-' * 10
33  print "Michigan has: ", cities[states['Michigan']]
34  print "Florida has: ", cities[states['Florida']]
35
36  # print every state abbreviation
37  print '-' * 10
38  for state, abbrev in states.items():
39      print "%s is abbreviated %s" % (state, abbrev)
40
41  # print every city in state
42  print '-' * 10
43  for abbrev, city in cities.items():
44      print "%s has the city %s" % (abbrev, city)
45
46  # now do both at the same time
47  print '-' * 10
48  for state, abbrev in states.items():
49      print "%s state is abbreviated %s and has city %s" % (
50          state, abbrev, cities[abbrev])
51
52  print '-' * 10
53  # safely get a abbreviation by state that might not be there
54  state = states.get('Texas')
55
56  if not state:
57      print "Sorry, no Texas."
58
59  # get a city with a default value
60  city = cities.get('TX', 'Does Not Exist')
61  print "The city for the state 'TX' is: %s" % city
```

Comentaris a la línia 56: En Python tots els objectes es poden avaluar com un valor booleà.

Avaluen a **True** qualsevol numèric no 0, qualsevol seqüència, set, mapping etc no buit i el valor **None** (el valor **None**, és el que té una variable a la qual no s'ha assignat cap valor).

Avaluen a **False** un valor numèric igual a 0 i qualsevol seqüència, set, mapping buit.

Executar-lo i comprendre com funciona el programa.
Comentar l'efecte dels mètodes emprats.

Prova d'assignar una ciutat a un estat que ja en té una d'assignada. Què passa?

5. Diccionaris.

Adaptar el codi de l'activitat anterior, de manera que es puguin guardar diverses ciutats per cada estat.