TUPLAS

La diferencia

Una lista no es lo mismo que una tupla. Ambas son un conjunto ordenado de valores, en donde este último puede ser cualquier objeto: un número, una cadena, una función, una clase, una instancia, etc. La diferencia es que las listas presentan una serie de funciones adicionales que permiten un amplio manejo de los valores que contienen. Basándonos en esta definición, puede decirse que las listas son dinámicas, mientras que las tuplas son estáticas.

Como las tuplas son estáticas, debes especificar sus elementos durante la creación:

```
a = (5, "Hola mundo!", True, -1.5)
```

Si se quiere crear una tupla con un único elemento, debe añadirse una coma (,) antes de cerrar el paréntesis:

```
a = (5,)
```

Acceso a los elementos

Puedes acceder a los distintos elementos de una lista o tupla indicando el índice (comenzando desde el 0) entre corchetes.

```
a = (5, "Hola mundo!", True, -1.5)
a[0] #5
a[2] #True
```

LISTAS

Las listas (o ‘List’) en Python son ***un tipo de estructuras de datos*** muy flexible que guardan de forma ordenada un conjunto de datos que no tiene porque ser del mismo tipo. En otros lenguajes de programación una lista equivaldría a un array, aunque Python no exige que los elementos de la lista tenga que ser del mismo tipo (‘int’, ‘float’, ‘chr’, ‘str’, ‘bool’, ‘object’).

```
# Declaramos una lista con diferente tipos de datos
```

```
lista = [1, 3.1416, 'j', 'global.komatsu',  True]

print 'Imprimimos los elementos de una lista y el tipo de dato de cada elemento'

for l in lista:

    print '%s - %s' %(l, type(l))
```

Métodos Generales de los diccionarios

Métodos para trabajar con diccionarios.

```
# Declaración de una lista

lista = list()

# Cuenta el número de elementos de la lista

len(lista)

# Agrega un elemento (x) al final de la lista.

lista.append(x)

# Inserta un elemento (x) en una posición determinada (pos)

lista.insert(pos, x)


# Une dos listas (une la lista2 (la que se pasa como parámetro) a la lista)

lista.extend(lista2)

# Borra el primer elemento de la lista cuyo valor sea x. Sino existe devuelve un error

lista.remove(x)

# Borra el elemento en la posición dada de la lista, y lo devuelve.

lista.pop(pos)

# Borra todos los elementos de la lista (lista.clear())
```

```
del lista[:]

# Devuelve el índice en la lista del primer elemento cuyo valor sea x.

lista.index(x)

# Devuelve el número de veces que x aparece en la lista.

lista.count(x)

# Ordena los ítems de la lista

lista.sort(cmp=None, key=None, reverse=False)

# Invierte los elementos de la lista.

lista.reverse()

# Devuelve una copia de la lista (lista.copy())

listaCopia = lista[:]
```

Métodos para ordenar listas.

```
nombreLista.sort(cmp=None, key=None, reverse=False)
```

el método sort tiene 3 parámetros. El primero 'cmp' lo vamos a obviar ya que a este habría que pasarle una función de comparación de dos elementos y esto lo haremos con el segundo parámetro '**key**' al que le indicaremos por que campo, valor, atributo, etc. queremos ordenar la lista. El tercer parámetro '**reverse**' le indicaremos con un *True* o *False* si queremos que la ordenación sea de forma creciente (reverse=False)

o decreciente (reverse=True). Por defecto sino definimos este parámetro la ordenación será de forma creciente.

Ejemplo

```
# Lista de Tuplas


futbolistasTup = [(1, "Bravo"), (15, "Jara"), (3, "Medel"), (5, "Aranguiz"), (11, "Vargas"), (14, "Valdivia"), (16, "Busquets"), (8, "Vidal"), (18, "Sanchez"), (6, "Dias"), (7, "Mena")]

futbolistasTup.sort(key=lambda futbolista: futbolista[0])

print "Imprimimos las lista ordenada por dorsal:"
print futbolistasTup
```

Si quisiéramos ordenar por el numero de camiseta en orden decreciente solo tendríamos que pasarle el parámetro 'reverse=True':

```
futbolistasTup.sort(key=lambda futbolista: futbolista[0], reverse=True)

futbolistasTup.sort(key=lambda futbolista: futbolista[1])

print "Imprimimos las lista ordenada por el nombre del jugador:"

print futbolistasTup
```

Veamos ahora como trabajaríamos con una lista de Strings que contiene el dorsal y el nombre de los jugadores separados por un guión:

```
# Lista de Tuplas


futbolista = [“1 - Bravo", “15 - Jara", “3 - Medel", “5 - Aranguiz"), “11 - Vargas", “14 - Valdivia",”16 - Busquets", “8 - Vidal",”18 - Sanchez", “6 - Dias", “7 - Mena"]
```

Para este ejemplo vamos a ordenar la lista por el nombre del jugador alfabéticamente de forma decreciente. Para ello tenemos que definir en la función lambda que le

pasamos al parámetro '**key**' que ordene la lista por la segunda parte del String (el nombre) tras hacer un "split" del mismo por el caracter "-":

```
futbolistas.sort(key=lambda futbolista: futbolista.split("-")[1],
reverse=True)
print "\nImprimimos las lista ordenada por el nombre del jugador:"
print futbolistas
```

DICCIONARIOS

Los diccionarios en Python son un tipo de estructuras de datos que permite guardar un conjunto no ordenado de pares clave-valor, siendo las claves únicas dentro de un mismo diccionario (es decir que no pueden existir dos elementos con una misma clave)

EJEMPLO 1:

```
# Defino la varible 'futbolistas' como un diccionario. No es necesario
declarar que tipo de dato es
futbolistas = dict()

futbolistas = {
    1 : "BRAVO", 2 : "MENA",
    4 : "ISLA", 17 : "Medel",
    18 : "Jara", 20 : "Aranguiz",
    21 : "Dias", 8 : "Vidal",
    7 : "Sanchez", 11 : "Vargas",
    10 : "Valdivia"
}
```

Como vemos el diccionario de declara entre los caracteres **'{}'** y cada uno de los elementos los separamos por comas **(',')**. Cada elemento lo definimos con su par clave:valor, pudiendo dar la clave y el valor de cualquier tipo de dato **('int', 'float', 'chr', 'str', 'bool', 'object').**

EJEMPLO 2:

```
# Recorrer un diccionario, imprimiendo su clave-valor
for k,v in futbolistas.items():
    print "%s -> %s" %(k,v)
```

Métodos Generales de los diccionarios

Métodos para trabajar con diccionarios.

```
# Declaración de un diccionario
diccionario = dict()

# Devuelve el numero de elementos que tiene el diccionario
len(dict)

# Compara el número de elementos distintos que tienen los dos
cmp (dict1,dict2)

# Devuelve una lista con las claves del diccionario
diccionario.keys()

# Devuelve una lista con los valores del diccionario
diccionario.values()

# Devuelve el valor del elemento con clave key. Sino devuelve default
diccionario.get(key, default=None)
```

```
# Inserta un elemento en el diccionario clave:valor. Si la clave existe no lo inserta
diccionario.setdefault(key, default=None)

# Insertamos un elemento en el diccionario con su clave:valor
dict['key'] = 'value'

# Eliminamos el elemento del diccionario con clave key
diccionario.pop('key',None)

# Devuleve la copia de un diccionario dict2 = dict.copy()
diccionario.copy()

# Elimina todos los elementos de un diccionario
diccionario.clear()

# Crea un nuevo diccionario poniendo como claves las que hay en la lista y los valores por defecto si se les pasa
diccionario.fromkeys(list, defaultValue)

# Devuelve true si existe la clave. Sino devuelve false
diccionario.has_key(key)

# devuelve un lista de tuplas formadas por los pares clave:valor
diccionario.items()


# Añade los elementos de un diccionario a otro
diccionario.update(dict2)
```

Creación de diccionarios basado en una llave dinámica

Ejemplo

```
lista = ["sdsd",1,2]
t={}
t= {lista[0]:lista[1]}
print (t)

name = 'rack.session'
val = 1
val2=2
cookies = dict([(name,[val,val2])])
print (cookies)
```

EJERCICIOS

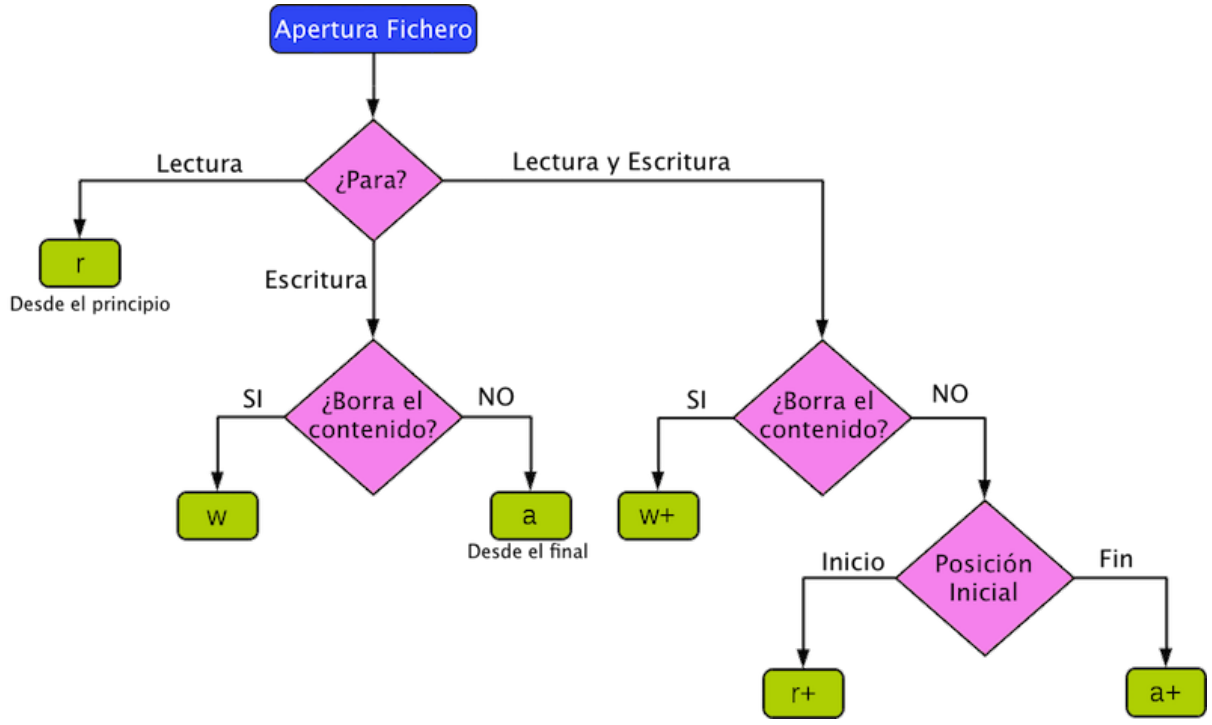
Ejecuta cada uno de estos códigos para ver los resultados

```
# -*- coding: utf-8 -*-
__author__ = 'Moises Bravo

# Defino la Variables  'futbolistas' como un diccionario. No es necesario declarar que tipo de dato es
futbolistas = dict()

futbolistas = {
    1 : "BRAVO", 2 : "MENA",
    4 : "ISLA", 17 : "Medel",
    18 : "Jara", 20 : "Aranguiz",
    21 : "Dias", 8 : "Vidal",
    7 : "Sanchez", 11 : "Vargas",
    10 : "Valdivia"
}


# Recorrer un diccionario, imprimiendo su clave-valor
for k,v in futbolistas.items():
    print "%s -> %s" %(k,v)

# Vemos cuantos elementos tiene nuestro diccionario
numElem = len(futbolistas)
print "\nNumero de elementos del diccionario len(futbolistas) = %i" %numElem

# Imprimimos una lista con las claves del diccionario
keys = futbolistas.keys();
print "\nClaves del diccionario futbolistas.keys(): \n%s" %keys

# Imprimimos en una lista los valores del diccionario
values = futbolistas.values()
print "\nValores del diccionario futbolistas.values(): \n%s" %values

# Obtenemos el valor de un elemento dada su clave
elem = futbolistas.get(6)
print "\nObtenemos el valor cuya clave es '6' futbolistas.get(6): %s" %elem

# Insertamos un elemento en el array. Si la clave ya existe no inserta el elemento
elem2 = futbolistas.setdefault(16,'Pizarro')
print "\nInsertamos un elemento en el diccionario. Si la clave existe no lo inserta" \
      "\nfutbolistas.setdefault(16,'Pizarro'): %s" %elem2
```

```
# Añadimos un nuevo elemento a la lista
futbolistas[22] = 'Paredes'
print "\nDiccionario tras añadir un elemento: futbolistas[22] = 'Paredes'
\n%s" %futbolistas

# Eliminamos un elemento del diccionario dada su clave
futbolistas.pop(22)
print "\nDiccionario tras eliminar un elemento: futbolistas.pop(22) \n%s"
%futbolistas

# Hacemos una copia del diccionario
futbolistasCopy = futbolistas.copy();
print "\nRealizamos una copia del diccionario
futbolistasCopy=futbolistas.copy(): \n%s" %futbolistas

# Eliminamos los elementos de un diccionario
futbolistasCopy.clear()
print "\nEliminamos los elementos de un diccionario futbolistasCopy.clear():
%s" %futbolistasCopy

# Creamos un diccionario a partir de una lista con las claves
keys = ['nombre', 'apellidos', 'edad']
dictList = dict.fromkeys(keys, 'nada')
print "\nCreamos un diccionario a partir de una lista dictList =
dict.fromkeys(keys, 'nada'): \n%s" %dictList

# Comprobamos si existe o no una clave
exit2 = futbolistas.has_key(2)
exit8 = futbolistas.has_key(8)
print "\nComprobamos si existen los elementos 2 y 8
\n\tfutbolistas.has_key(2) = %s " \
      "\n\tfutbolistas.has_key(8) = %s" %(exit2,exit8)

# Devolvemos los elementos del diccionario en tuplas
tuplas = futbolistas.items()
print "\nPasamos el diccionario a tuplas con clave-valor: tuplas =
futbolistas.items() \n%s" %tuplas

# Mergeamos dos diccionarios
suplentes = {
    4:'Parraguez', 9:'Torres', 12:'Valdes',
    13:'Figueroa' , 17:'Monson', 19:'Herrera',
    23:'Bravo'
}

futbolistas.update(suplentes)
print "\nAñadimos los elementos de un diccionario a otro
futbolistas.update(suplentes): \n%s" %futbolistas
```

ARCHIVOS

Antes de pasar a explicar y a mostrar ejemplos de lectura y escritura, mostramos en la siguiente imagen un diagrama para ver los modos de lectura y escritura en python:



Tipos de lectura y escritura de ficheros

la estructura general para trabajar con archivos es

```
with open(archivo, TIPO) as variable:
    for line in reader:

        print line
```

Archivo: Es el nombre del archivo, si el archivo esta en una ubicación distinta del archivo .py se debe especificar

TIPO: Es el modo como se va a abrir el archivo y puede ser

- **r:** Abrir fichero para lectura. El puntero se posiciona al principio del fichero

- **r+:** Abrir fichero para lectura y escritura. El puntero se posiciona al principio del fichero
- **w:** Trunca a cero la longitud o crea un fichero de texto para escritura. El puntero se posiciona al principio del fichero
- **w+:** Abrir fichero para lectura y escritura. Si el fichero no existe, se crea, de lo contrario se trunca. El puntero se posiciona al principio del fichero
- **a:** Abrir fichero para lectura. Se creará el fichero si no existe. El puntero se posiciona al final del fichero.
- **a+:** Abrir fichero para lectura y escritura. Se creará el fichero si no existe. El puntero se posiciona al final del fichero.

si la ubicación del archivo se encuentra en una carpeta distinta a la unicacion del archivo.py es recomendable utilizar lo siguiente

```
#define la ubicación del PATH , basado en la ubicacion del archivo.py
import os
dirname = os.path.dirname(__file__)
file = os.path.join(dirname,'archivos/miarchivo.txt')
```

Operaciones con archivos

Leer archivo: read, readline, readlines, with-as

Con el método **read**() es posible leer un número de bytes determinados. Si no se indica número se leerá todo lo que reste o si se alcanzó el final de fichero devolverá una cadena vacía.

```
# Abre archivo en modo lectura
archivo = open('archivo.txt','r')

# Lee los 9 primeros bytes
cadena1 = archivo.read(9)

# Lee la información restaste
cadena2 = archivo.read()

# Muestra la primera lectura
print(cadena1)

# Muestra la segunda lectura
print(cadena2)

# Cierra el archivo
archivo.close()
```

El método readline() lee de un archivo una línea completa

```
# Abre archivo en modo lectura
archivo = open('archivo.txt','r')

# inicia bucle infinito para leer línea a línea
while True:
    linea = archivo.readline()  # lee línea
    if not linea:
        break  # Si no hay más se rompe bucle
    print(linea)  # Muestra la línea leída
archivo.close()  # Cierra archivo
```

El método readlines() lee todas las líneas de un archivo como una lista. Si se indica el parámetro de tamaño leerá esa cantidad de bytes del archivo y lo necesario hasta completar la última linea.

```
# Abre archivo en modo lectura
archivo = open('archivo.txt','r')

# Lee todas la líneas y asigna a lista
lista = archivo.readlines()

# Inicializa un contador
numlin = 0

# Recorre todas los elementos de la lista
for linea in lista:
    # incrementa en 1 el contador
    numlin += 1
    # muestra contador y elemento (línea)
    print(numlin, linea)

archivo.close()  # cierra archivo


# abre archivo (y cierra cuando termine lectura)
with open("indice.txt") as fichero:
    # recorre línea a línea el archivo
    for linea in fichero:
        # muestra línea última leída
        print(linea)
```

Escribir en archivo: write, writelines

El método write() escribe una cadena y el método writelines() escribe una lista a un archivo. Si en el momento de escribir el archivo no existe se creará uno nuevo.

```
cadena1 = 'Datos'  # declara cadena1
cadena2 = 'Secretos'  # declara cadena2

# Abre archivo para escribir
archivo = open('datos1.txt','w')

# Escribe cadena1 añadiendo salto de línea
archivo.write(cadena1 + '\n')

# Escribe cadena2 en archivo
archivo.write(cadena2)

# cierra archivo
archivo.close()


# Declara lista
lista = ['lunes', 'martes', 'miercoles', 'jueves', 'viernes']

# Abre archivo en modo escritura
archivo = open('datos2.txt','w')

# Escribe toda la lista en el archivo
archivo.writelines(lista)

# Cierra archivo
archivo.close()
```

Mover el puntero: seek(), tell()

El método seek() desplaza el puntero a una posición del archivo y el método tell() devuelve la posición del puntero en un momento dado (en bytes).

```
# Abre archivo en modo lectura
archivo = open('datos2.txt','r')

# Mueve puntero al quinto byte
archivo.seek(5)

# lee los siguientes 5 bytes
cadena1 = archivo.read(5)

# Muestra cadena
print(cadena1)
```

```
# Muestra posición del puntero
print(archivo.tell())

# Cierra archivo
archivo.close()
```

EJERCICIOS

FUNCIONES

Sintaxis

```
def sumar(param1,param2=30):
    return param1 + param2


print sumar(10,15)
```

Para declarar una función solo se debe poner la palabra "**def**" seguido del nombre de la función, para el ejemplo le hemos puesto "sumar", en los paréntesis deben ir los **parámetros** que se van a usar dentro de la función, por último el contenido de la función . **return** indica el valor de respuesta del a función. Siempre tengan en cuenta la identación dentro de la función, si el parámetro tiene un = valor, es el valor por defecto del parámetro si este no se agrega Ej

tomando la función anterior

print sumar(10,15) : esta función retorna el valor 25 por que param1 toma el valor 10 y param2 toma el valor 15.
print sumar(10): esta función retorna el valor 40 esto es por que el param1 toma el valor 10, y como no se especifica el param2 toma el valor por defecto 30

Modulos

Math

```
import math
```

Redondeos

```
print(math.floor(3.99))  # Redondeo a la baja (suelo)
#resultado 3
print(math.ceil(3.01))   # Redondeo al alta (techo)
resultado 4
```

Sumatorio mejorado

```
numeros = [0.9999999, 1, 2, 3])
math.fsum(numeros)
resultado 6.9999999
```

Truncamiento

```
math.trunc(123.45)
resultado 123
```

Potencias y raíces

```
math.pow(2, 3)  # Potencia con flotante resultado 8
math.sqrt(9)    # Raíz cuadrada (square root) resultado 3
```

Constantes

```
print(math.pi)  # Constante pi
print(math.e)   # Constante e
```

RANDOM

```
import random

# Flotante aleatorio >= 0 y < 1.0
print(random.random())

# Flotante aleatorio >= 1 y <10.0
print(random.uniform(1,10))

# Entero aleatorio de 0 a 9, 10 excluído
print(random.randrange(10))

# Entero aleatorio de 0 a 100
print(random.randrange(0,101))

# Entero aleatorio de 0 a 100 cada 2 números, múltiples de 2
print(random.randrange(0,101,2))

# Entero aleatorio de 0 a 100 cada 5 números, múltiples de 5
print(random.randrange(0,101,5))
```

Muestras

```
# Letra aleatoria
print(random.choice('Hola mundo'))

# Elemento aleatorio
random.choice([1,2,3,4,5])

# Dos elementos aleatorios
random.sample([1,2,3,4,5], 2)
```

Mezclas

```
# Barajar una lista, queda guardado
lista = [1,2,3,4,5]
random.shuffle(lista)
print(lista)
```

EJERCICIOS

1. Crea un programa que permita guardar en un diccionario el archivo canción.txt usando como llave cada palabra distinta encontrada y contando la cantidad de veces que aparece en el archivo, antes de eso asegurate de dejar todas las palabras en mayúscula para que no se repitan
   Ej;

   dic = {“CACEROLAZO!”:17,”CUCHARA”:2}

2. Tomando los datos de población y superficie de cada región de chile (regiones.txt) calcule la densidad por kilómetro cuadrado para cada región de chile, muestrala en pantalla y crea un nuevo archivo agregando la densidad
3. toma el archivo regiones.txt y ordene los registros de menor a mayor por población y suma además el total de población de chile.
4. usa el archivo canción.txt y cuanta la cantidad de a , e, i, o, u presentes en el archivo.