TUPLAS

Tuplas ( )

- Ao igual que as listas, unha tupla é unha secuencia de obxectos Python. As diferencias están en que:
  - As tuplas son inmutables (non poden ser modificadas tras a súa creación)
    - Non teñen métodos pop/remove
    - Non teñen método append(), nin se poden engadir elementos co operador +=
  - **Ventaxas**: usan menos memoria que as listas (ok para gardar lista constante)

- Creación: Especifícanse como unha secuencia de elementos entre parénteses e separados por comas (aínda que os parénteses son opcionais se non hai conflictos)
  - tupla1 = ('fisica', 'quimica', 128.9,256.0);
  - tupla2 = (1, 2, 3, 4, 5)
  - tupla_baleira = ()
  - tupla_cun_unico_elemento = ('hola',) ## coma ao final !!
  - tupla2b = 1,2,3,4,5

```
t1 = ('a')   ##un string 'a' entre parénteses
t2 = ('a',)  ##unha tupla cun só elemento 'a'
print type(t1)  ## imprime <type 'str'>
print type(t2)  ## imprime <type 'tuple'>
```

Tuplas ( )

- Operadores +, *, [], [:], in, not in, son aplicables. Tamén len(tupla)

```
tupla = ('a','b','c')    #
tupla2= tupla + tupla    #
tupla3=tupla*3           #
tupla4=(5,)              #
tupla5=()                #
print tupla              # ('a', 'b', 'c')
print tupla2             # ('a', 'b', 'c', 'a', 'b', 'c')
print tupla3             # ('a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c')
print tupla[0]           # a
print tupla[1:]          # ('b', 'c')
print tupla[::-1]        # ('c', 'b', 'a')
print 'c' in tupla       # True
print 'd' in tupla       # False
print len(tupla)         # 3
print len(tupla4)        # 1
print len(tupla5)        # 0
```

- Non se poden modificar, pero...

....podemos crear novas tuplas a partir dunha/s existente/s

```
tupla  = ('a','b','c')
tupla  = tupla + ('d',)          ## tupla contería ('a','b','c','d')
tupla  = tupla[0:2] + tupla[3:]  ## tupla contería ('a','b','d')
tupla  = tupla[1:]               ## tupla contería ('b','d')
tupla2 = ('5',)+ tupla + ('6',)  ## tupla2 contería ('5','b','d','6')
print tupla                      ## ('b','d')
print tupla2                     ## ('5','b','d','6')
del tupla2                       ## elimina a tupla completa (obxecto)
```

Tuplas ( )

- Recordemos, que os ( ) non son obrigatorios.
  - calquera conxunto de obxectos separados por comas considéranse por defecto unha tupla

```
print 'abc', -4.22, 'xyz'  ## imprime abc -4.22 xyz
x, y = 1, 2;
(a,b) = 'hola','mundo'
print "x , y : ", x,y      ## imprime x , y :  1 2
print "a , b : ", a,b      ## imprime hola mundo
```

- Xa temos visto tuplas...
  - ao facer asignación simultánea (de tuplas)

```
a,b,c = 8, 9, 10
x,y = y,x
```

  - cando unha función devolve varios valores

```
def  f()
    ...
    return 'si', 198

(a,b) = f()
```

- As tuplas son iterables (podémolas iterar cun for)

```
for item in tupla:
    print item
```

Tuplas(): Funcións

http://www.tutorialspoint.com/python/python_tuples.htm

- Funcións aplicables a tuplas:
  - cmp(t1, t2) . compara valores das 2 tuplas
    - devolve -1 se t1<t2; 0 se son iguais, +1 se t1>t2
  - len(tupla) . Lonxitude da tupla
  - max(tupla) . Maior valor da tupla
  - min(tupla) . Menor valor da tupla
  - tuple(sec) . Convirte unha secuencia en tupla

```
t1, t2 = (123, 'xyz'), (456, 'abc')
t3 = t1[::-1]
t4 = 456, 'abc'
print cmp(t1, t2)     ## -1
print cmp(t2, t1)     ##  1
print cmp(t1, t3)     ## -1
t3 = t2 + (786,);
print cmp(t2, t3)     ## -1
print cmp(t2, t4)     ##  0

print len(t2)         ##  2
print max(t2)         ## abc
print max(t2)         ## 456

t4=tuple ('abc')      ## ('a','b','c')
t5=tuple(['a','5'])   ## ('a',5)
```

DICCIONARIOS

Diccionarios { k1:v1 , k2:v2, }

- Podemos ver un *diccionario* como un conxunto de pares *clave:valor* de forma que será posible buscar/acceder a elementos por clave.

  {'r':'red', 'g':'green', 'b':'blue'} ➔ claves ['r', 'g', 'b'], valores ['red','green','blue']

- Creación:
  - Secuencia de pares clave:valor separados por comas e entre chaves

    ```
    cores = {'r':'red', 'g':'green', 'b':'blue'}
    ```

  - Partir dun diccionario baleiro, e ir engadindo valores usando operador de indexación

    ```
    cores = {}              ##crea diccionario baleiro
    cores['r'] = 'red'
    cores['g'] = 'green'
    cores['b'] = 'blue'
    ```

Diccionarios { k1:v1 , k2:v2, }

- Propiedades das claves:
  - Mentres nas secuencias string/listas/tuplas os índices utilizables eran enteiros
    - `l = ['a','b', 'c']  ou  l="abc"  ou  l = ('a','b', 'c'])`
    - `print l[1]  ##imprime 'b'`
  - Os diccionarios permiten indexar por calquera tipo de elemento inmutable (clave), recuperando así o valor asociado a dita clave. A clave pode ser un número, string, ou unha tupla!
    - A clave debe ser inmutable
    - A clave debe ser única
- Propiedades dos valores:
  - Ningunha en particular: poden ser calquera tipo de obxecto
    - Un enteiro, un string, unha lista, outro diccionario,...
- A orde dos elementos no diccionario non importa: Táboa hash.
  - Non accederemos aos elementos por un índice "enteiro" se non por clave.

Diccionarios { k1:v1 , k2:v2, }

- Operacións (I):
  - Engadir /modificar un elemento ao diccionario: d[key] = v

```
cores = {}
cores['r'] = 'red'
cores['g'] = 'green'
cores['b'] = 'BLUE'     #inserción
cores['b'] = 'blue'     #modificación
```

  - Obter un elemento do diccionario d[key] ou d.get(key)
    - dic[clave] ou dic.get(clave, default=None)

```
 cor  = cores['r']
#cor1 = cores['a']            ##provocará unha excepción KeyError
 cor1 = cores.get('a')
 cor2 = cores.get('a', None)
 print (type(None))               ## imprime <type 'NoneType'>
 cor3 = cores.get('a', 'N.d')
 cor4 = cores.get('r','N.d')
print cor, cor1, cor2,\
            cor3, cor4       ## imprime red None None N.d red
```

Diccionarios { k1:v1 , k2:v2, }

- Operacións (II):
  - Engadir /actualizar dict cos elementos doutro diccionario

```
cores = {'r':'red',  'g':'green', 'b':'blue'}
cores2= {'y':'yellow', 'w':'white', 'r':'rojo', }
cores2.update(cores)
print cores2
   ## imprime {'y':'yellow', 'r':'red', 'b':'blue', 'g':'green', 'w':'white'}
```

  - Obter un elemento do diccionario, e se non está meter un valor (o por defecto, ou o indicado) d.setdefault(key[,val])
    - dic.setdefault(clave, default=None)

```
 cores = {'r':'red',  'g':'green', 'b':'blue'}
cores.setdefault('w','white')
print cores
## imprime {'r': 'red', 'b': 'blue', 'w': 'white', 'g': 'green'}
```

Diccionarios { k1:v1 , k2:v2, }

- Operacións (III):
  - Eliminar un elemento do diccionario: del d[key]

```
cores = {'r':'red',  'g':'green', 'b':'blue'}

del cores['r'] ## borra elemento 'r':'red'
               ## provoca excepción keyError, se a clave 'r' non existe!
print cores    ## imprime {'b': 'blue', 'g': 'green'}
```

  - Chequear pertenencia ao diccionario: in/not in d.haskey(k)
    - mira se o valore indicado está entre as ***claves*** do diccionario (non chequea pertenencia de valores, só de claves!!

```
cores = {'r':'red', 'g':'green', 'b':'blue'}

print 'r' in cores    ## imprime True
print 'red' in cores  ## imprime False
print 'a' in cores    ## imprime False
```

Diccionarios { k1:v1 , k2:v2, }

- Operacións (IV):
  - Iterar polos elementos do

```
cores = {'r':'red',  'g':'green', 'b':'blue'}
for key in cores:
        print key, ":", d[key]

#Imprime:
r : red
b : blue
g : green
```

  - len(d): Número de elementos no diccionario

```
cores = {'r':'red', 'g':'green', 'b':'blue'}

print len(cores)  ## imprime 3
```

Diccionarios { k1:v1 , k2:v2, }

- Operacións (V):
  - Comparación de diccionarios : ==, !=
    - Mírase se teñen os mesmos elementos (==) ou distintos (!=)
    - Operadores <, >, <=, >= NON son aplicables

```
cores1 = {'g':'green', 'b':'blue', 'r':'red'}
cores2 = {'r':'red', 'g':'green', 'b':'blue'}  ##mesmo, diferente orde
cores3 = {'r':'red', 'g':'green'}
cores4 = {'g':'green', 'b':'blue'}

print cores1 == cores2  #True
print cores1 == cores3  #False
print cores1 == cores4  #False
```

Diccionarios { k1:v1 , k2:v2, } : Funcións

http://www.tutorialspoint.com/python/python_tuples.htm

- Funcións aplicables a diccionarios:
  - cmp(d1,d2) . compara 2 diccionarios
    - Primeiro mira se as lonxitudes coinciden se non ➔ -1 (d1<d2) ou +1 (d1>d2)
    - Se igual lonxitude
      - Compara a clave máis pequena
        - » se non coinciden ➔ -1 ou +1
        - » se son iguais
          - compara a segunda clave máis pequena...

```
cores1 = {'g':'green', 'b':'blue', 'r':'red'}
cores2 = {'r':'red', 'g':'green', 'b':'blue'}
cores3 = {'r':'red', 'rrrr':'green'}
cores4 = {'g':'green', 'b':'blue'}
print "cmp--> ", cmp (cores1, cores2) ## 0
print "cmp--> ", cmp (cores1, cores3) ## 1
print "cmp--> ", cmp (cores4, cores1) ## -1
print "cmp--> ", cmp (cores3, cores4) ## 1
print "cmp--> ", cmp (cores4, cores3) ## -1
```

  - len(dic) . Lonxitude do diccionario
  - str(dic). Xera un string ("para ver o seu contido en forma { k:v , k:v, ...}
  - type(dic) . Indica o tipo.

```
print type(cores) ## <type 'dict'>
```

Diccionarios { k1:v1 , k2:v2, } : Métodos

http://www.tutorialspoint.com/python/python_tuples.htm

- Métodos:
  - d.clear(). Borra todos os elementos do diccionario
  - d.copy(). Devolve un duplicado do diccionario
  - **dict.**fromkeys(seq [,valor]). Crea un NOVO diccionario coas claves da secuencia *seq. Establece os valores a None, ou a* valor se incluido. (Este é un método estático da clase dict. Invóquese como: **d = dict.fromkeys((1,2,3),[])**
  - d.items(). Devolve unha lista de tuplas (clave,valor)
  - d.keys(). Devolve unha lista coas claves do diccionario
  - d.values(). Devolve a lista de valores do diccionario
  - d.has_key(key). Devolve True/False se o d contén/non contén a clave key
  - d.get(key, default=None). Devolve un valor para a key indicada. Se non existe devolve None (defecto), ou o valor indicado.
  - d.setdefault(key, default=None). Similar a get(), pero se non existe a clave key no diccionario, insértaa facendo dict[key] = default. Por defecto establécese o valor a None, pero pódese indicar o valor a establecer.
  - d.update(dict2). Engade os pares clave:valor do dict2 a dict, e para as claves que xa existían son actualizados cos valores de dict2

Diccionarios { k1:v1 , k2:v2, } : Métodos

http://www.tutorialspoint.com/python/python_tuples.htm

- Métodos:
  - Exemplos

```
cores = {'r':'red',  'g':'green', 'b':'blue'}
print d.items()
print d.keys()
print d.values()
print d.has_key('r')
print d.get('r')
print d.setdefault('w','blanco')
print d.get('w')
print d
d2= {'w':'white'}
d.update(d2)
print d
```

```
## Esta é a saída do programa
[('r', 'red'), ('b', 'blue'), ('g', 'green')]
['r', 'b', 'g']
['red', 'blue', 'green']
True
red
blanco
blanco
{'r': 'red', 'b': 'blue', 'w': 'blanco', 'g': 'green'}
{'r': 'red', 'b': 'blue', 'w': 'white', 'g': 'green'}
```

MÓDULOS

Módulos

- Os módulos permiten organizar o código Python.
  - Agrupar código relacionado dentro dun módulo facilita o seu uso e compresión.
- Un módulo é un ficheiro que contén código Python e onde están definidas funcións, clases e variables (aínda que tamén pode conter código executable)
- Desde un programa Python podemos **importar** 1+ módulos e desta forma ter acceso ao seu contido

Módulos

- Exemplo

```
#main.py
##importo todo o de numeros
from numeros import *

## programa principal
amosa_primos(1,30)
amosa_pares(1,30)
```

main.py

```
#módulo numeros.py  ## MANEXO DE NUMEROS ##
def e_primo(n):
   for i in range(2,n):
      if n%i == 0:
         return False
   return True

def amosa_primos(ini,n):
    print "amosa primos entre %d e %d" % (ini,n)
    for i in range (ini,n+1):
        if e_primo(i):
            print i,
    print ""

def amosa_pares(ini,n):
    print "amosa pares entre %d e %d" % (ini,n)
    for i in range (ini,n+1):
        if i%2==0:
            print i,
    print""
```

numeros.py

```
$python main.py
amosa primos entre 1 e 30
1 2 3 5 7 11 13 17 19 23 29
amosa pares entre 1 e 30
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
```

Módulos

- Sentencia **import**
  - from numeros import *
    - importa todos os obxectos do módulo numeros.py
    - podémolos usar como se estivesen incluidos no módulo actual
      - ex: amosa_primos(1,30)
  - import numeros.obxecto
    - import numeros.amosa_primos
    - import numeros.amosa_pares
      - Importa únicamente os obxectos seleccionados do módulo numeros.py
        - » ex: amosa_primos(1,30)
  - import numeros
    - Importa o paquete, pero non engade os obxectos ao espacio de nome do módulo actual, polo que haberá que ser referir aos obxectos como

      numeros.<nome_obxecto>
    - ex: numeros.amosa_primos(1,30)
    - ex: numeros.amosa_pares(1,30)

ENTRADA/SAÍDA: FICHEIROS

E/S: FICHEIROS

- Un ficheiro pódese ver como unha secuencia de datos que se almacena en disco nunha determinada ruta (d:\ex\axenda.dat)
  - Ao residir en disco, é un almacenamento permanente de datos: ¡non se perde o seu contido cando o noso programa en Python remata!

d:\ex\axenda.dat

```
1,Antonio,666999215\n2,Pepe,670931688\n3,Juan,699874563\n...
```

- Desde os nosos programas en Python poderemos abrir un ficheiro, ler ou modificar o seu contido, e finalmente pechar dito ficheiro.

E/S: FICHEIROS

http://www.tutorialspoint.com/python/python_files_io.htm

- Principais operacións con ficheiros
  - f = open(ruta_ficheiro [,modo][, buffering])
    - ruta_ficheiro
      - relativa "axenda.dat", ou absoluta "D:\ex\axenda.dat"
    - modo
      - "r" (lectura), "w"(escritura), "a" (append) ... e outros
    - Atributos do **obxecto f** (devolto por open)

f.closed	Devolve True se o ficheiro foi pechado
f.mode	Indica o modo no que foi aberto o ficheiro (w, r, a, ...)
f.name	Devolve o nome do ficheiro (ruta_ficheiro)

  - close(f)
    - Débese invocar cando xa non queiramos seguir usando o ficheiro f

E/S: FICHEIROS

http://www.tutorialspoint.com/python/python_files_io.htm

- Principais operacións con ficheiros
  - f.write(string)
    - Escribe os datos contidos en *string* no ficheiro f.name
  - string = f.read([count])
    - Le todo o contido do ficheiro f.name, ou só *count* bytes (se se indica)
- Outras funcións con ficheiros
  - f.tell()
    - Indica a posición (offset en bytes respecto ao principio do ficheiro) na que se fará a seguinte lectura de datos do ficheiro
      - recórdese que se en f.read lle pasamos o parámetro count, non se le todo de golpe
  - f.seek(offset, desde)
    - Móvese á posición offset do ficheiro
      - se desde = 0, o offset é relativo ao principio
      - se desde = 1, o offset é relativo á posición actual
      - se desde = 2, o offset é relativo á última posición do ficheiro.

E/S: FICHEIROS

http://www.tutorialspoint.com/python/python_files_io.htm

- Máis funcións de interese
  - O módulo *os*
    - import os
      - os.remove(ruta) → elimina ficheiro
      - os.rename(orixe, destino) → cambia nome "orixe" a "destino"
  - O módulo *os.path*
    - import os.path
      - os.path.exists(ruta) → ver se unha ruta existe
      - os.path.isfile(ruta) → ver se un path é un ficheiro

E/S: FICHEIROS

http://www.tutorialspoint.com/python/python_files_io.htm

- Exemplos
  - Escribir nun ficheiro

```
# Open file
f = open("datos.dat", "wb")
f.write( "Vou escribir isto a disco\n\ dentro do ficheiro datos.dat\n");

# Close opend file
f.close()
```

  - Ler dun ficheiro

```
# Open file
f = open(" datos.dat ", "r")
str = f.read(10);
print "Lin cadea : ", str
# Close file
f.close()
```

E/S: FICHEIROS

http://www.tutorialspoint.com/python/python_files_io.htm

- Exemplos
  - Ler o contido dun ficheiro e amosalo por pantalla

```
import os.path

# Define a filename.
filename = "p13.dat"

if not os.path.isfile(filename):
    print 'File does not exist.'
else:
    # Open the file as f.
    f =open(filename)
    content = f.read()
    content.splitlines()

    # Show the file contents line by line.
    # We added the comma to print single newlines and not double newlines.
    # This is because the lines contain the newline character '\n'.
    for line in content:
        print line...
```

ARGUMENTOS DESDE A LIÑA DE COMANDOS

Argumentos desde a liña de comandos

- O noso programa pode recibir argumentos desde a liña de comandos cando é invocado:

```
$python prog.py argumento1 argumento2
```

  - Uso de **sys.argv**
    - argv é unha lista que contén len(argv) strings que contén o nome do programa e os elementos que veñan tras el desde a liña de comandos

```
argv=[ "prog.py", "argumento1", "argumento2" ]
```

    - podemos acceder aos seu elementos usando a sintaxe propia de listas []
      - Recordemos que cada elemento é un string, pero en caso de que os datos dese string sexan "convertibles" a outro tipo (p.ex. int), podemos facer unha conversión explícita de tipos (p.ex `int(argv[1])`)

```
#pargv.py
from sys import argv
if len(argv) != 3:
    print "sintaxe incorrecta: debe invocar ao programa"
    print "asi: %s  <nome> <apelidos>" % argv[0]
    exit(0)

prog, nome, apelidos = argv
print "O programa recibiu %d parametros" % len (argv)
print "%s %s %s" % (prog, nome, apelidos)
print "%s %s %s" % (argv[0], argv[1], argv[2])
```

```
$ python pargv.py Juan Fernandez
O programa recibiu 3 parametros
pargv.py Juan Fernandez
pargv.py Juan Fernandez
```

Argumentos desde a liña de comandos

- Uso de **sys.getopt()**
  - é posible recibir os argumentos como soen pasarse aos programas linux
    - ex: python -n juan -a fernandez
    - ex: python --nome=juan --apelidos=fernandez
    - ex: python -h
  - Básicamente, hai que redefinir a función __main__ para que se execute un main que recibe como argumentos a lista "sys.argv", e despois se usa a función sys.getopt() para procesar ditos argumentos.
  - Ver exemplo ➔

Argumentos desde a liña de comandos

– Exemplo de uso de **sys.getopt()**

```
import sys, getopt

def main(argv):
   nome = ''
   apelidos = ''
   try:
      opts, args =
getopt.getopt(argv,"hn:a:",["nome=","apelidos="])
   except getopt.GetoptError:
      print 'pgetopt.py -n <nome> -a <apelidos>'
      sys.exit(2)
   for opt, arg in opts:
      if opt == '-h':
         print '%s -n <nome> -a <apelidos>' % ("pgetopt.py")
         sys.exit()
      elif opt in ("-n", "--nome"):
         nome = arg
      elif opt in ("-a", "--apelidos"):
         apelidos = arg
   print 'Nome = %s' % (nome)
   print 'Apelidos = %s' % (apelidos)

if __name__ == "__main__":
   main(sys.argv[1:])
```

```
$python pgetopt.py -h
pgetopt.py -n <nome> -a <apelidos>

$python pgetopt.py --nome=juan –a fernandez
Nome = juan
Apelidos = fernandez
```

Máis información: http://www.tutorialspoint.com/python/python_command_line_arguments.htm