30/10/2025 21:17 1/5 think python

# think python

<string>.capitalize()<string>.isupper()

list(cadena): separa en caracteres<string>.split(): separa en palabras

se puede pasar por parámetro el delimitador

## cadenas

```
    <delimitador>.join(<lista>): junta los elementos de la lista, poniendo en medio el delimitador (!!)

list
    • por referencia (aliasing, más de una referencia al mismo objeto)

    OJO al usar funciones o métodos al trabajar con listas, (referencia VS new lista): página 96

    • empty = [], cadenas = ['uno', 'dos', 'tres', 'cuatro', 'cinco'], numeros = [1,2,3],
      mixta = ['uno',2,['tres',3],'cuatro']
    • mutables : cadenas[1] = 'DOS'
    • búquedas: 'tres' in cadenas → true
    recorridos:
          ∘ for i in numeros:
          ∘ for i in range(len(mixta)):

    concatenación: otra mixta = cadenas + numeros → ['uno','dos','tres','cuatro','cinco',1,2,3]

    • multiplicativo: numeros * 2 → [1,2,3,1,2,3]
    slices:

    de los índices pasados, incluye el primero, excluye el segundo.

          o cadenas [1:] → ['dos','tres','cuatro','cinco']
          o cadenas[:1] → ['uno','dos']

    cadenas [2:4] → ['tres','cuatro']

          asignación:
                ■ cadenas[2:3]=['aaa','bbb'] → ['uno','aaa','bbb','cuatro','cinco']
    • métodos:
          o <list>.append(): añade elemento al final

    extend(<list>): añade al final de la lista otra lista, cambia la primera.

          o list>.pop(#elemento) : extrae elemento de la lista (lo devuelve) en función del índice del

    del <list>[#elemento] : borra sin devolverlo en función del índice del mismo. permite

                  slices
                <re>remove(elemento) : borra el elemento si sabemos cual es
```

## diccionarios

- pareja clave-valor
- no mantiene el orden de entrada dentro del diccionario

o print list('cadena'):['c','a','d','e','n','a']

- empty = dict()
- diccio = {}
- asignación: diccio[<clave>] = 'valor'
- **len**: len(<diccionario>)

```
    get: recupera un valor de una clave y permite establecer un valor por defecto (si no existe la clave)

            <diccionario>.get('<clave>',<default>)

    keys: devuelve como lista las keys de un diccionario
    in:
```

- busca en las claves, devuelve cierto/falso → '<clave>' in diccio
  - o con el uso de <diccionario>.values() se pueden hacer búsquedas en los valores
  - búsqueda por hashtable
  - o not in
- INVERSE pagina 127 → inverse[val] = [key] ????

## **Tuplas**

```
    secuencia de valores, separados por coma, inmutables

tupla = 'a', 'b', 'c', 'd'
• tupla = ('a','b','c','d')
• tupla = 'a', ← nótese la coma final
      o type(tupla)
tupla = tuple('cadena')
      o print tupa → ('c','a','d','e','n','a')

    acceso a elementos:

      tupla[0]
      tupla[1:3] ← primero inclusive, último no
• no se puede reasignar un valor a través del índice, por su caracter inmutable, pero si:
      o tuplaNueva = ('C',) + tupla[1:] → ('C','a','d','e','n','a')
usando tuplas:
      o para intercambiar valores: a,b = b,a
      ∘ separar una cadena: add='nombre@dominio.com → nombre,dominio = addr.split('@')
en funciones:
      o parámetros variables a una función (gathers): def printall(*args):
      o pasar parámtros a través de tuplas, cuando la función espera los valores por separado: t =
        (7,3); divmod(*t) \leftarrow sin * da error
• zip: combina dos tuplas, generando una lista de tuplas con un elemento de cada (hasta la más corta)
      \circ s = 'abc'; t = [0,1,2]; mizip = zip(s,t) \rightarrow [('a', 0), ('b', 1), ('c', 2)]
      for letra, numero in mizip:print numero, letra
```

- enumerate: for indice, elemento in enumerate('abc'):
- diccionarios:
  - uso de items() para pasar el diccionario a tuplas
  - o uso de dict() para pasar una lista de tuplas a diccionario
- en combinación: d = dict(zip('abc', range(3))) → {'a': 0, 'c': 2, 'b': 1}
- es de uso común usar tuplas como índices de diccionarios
- se pueden usar operadores lógicos de comparación con tuplas, se van comparando elemento a elemento hasta que satisface la operación
- sort se usa de una manera similar (ejemplo SORT en página 119)

### **13**

## ficheros

#### escritura fichero

30/10/2025 21:17 3/5 think python

```
    escritura: fout = open('fichero.txt','w'); fout.write('una linea'); fout.close()
    fout requiere string
    str()
    format operator % → palabras=5; fout.write('Una cadena de %d palabras' % palabras)
    si hay más de 1 variables, ha de ser una tupla:'La cadena %s %d palabras' % ('tiene',5)
    más info: https://docs.python.org/2/library/stdtypes.html#string-formatting
```

## nombres fichero y paths

```
    módulo os → import os
    .getcwd()
    .path.abspath('file') : devuelve path absoluto al fichero
    .path.exists('file')
    .path.isdir('file')
    .path.isfile('file')
    .path.join(dirname, name)
    .listdir(cwd) : devuelve una lista de ficheros y directorios de cwd
```

### catch exceptions

```
try:
    fin = open('fichero_no_existente')
    for line in fin:
        print line
    fin.close()
except:
    print "Error"
```

#### databases

```
    import anydbm
        o solo soporta strings → uso de pickling
    db = anydbm.open('captions.db','c')
        o 'c': crear si no existe aún
    uso como diccionario
        o asignación
        o recorridos: key,items
    db.close()
```

## pickling

- para superar la limtación de anydbm de trabajar solo con strings, usamos el módulo pickle
- convierte casi cualquier cosa a una representación en string y viceversa
- import pickle
- pickle.dumps(<anything>)
- pickle.loads(<string>)
- módulo shelve ??

## **Pipes**

- cualquier comando que se pueda lanzar desde la shell se puede lanzar como un pipe en python
- os.popen('ls -l'): popen está deprecado a favor de subprocess

```
filename = 'book.tex'
cmd = 'md5sum ' + filename
fp = os.popen(cmd)
res = fp.read()
stat = fp.close() // devuelve None si todo correcto
// res contiene el resultado del comando
```

#### **Modulos**

- cualquier fichero que contenga código python puede ser importado como un módulo
- normalmente solo se definen funciones, otro código sería ejecutado
- para evitar la ejecución de ese código cuando el programa se usa como módulo, se añade:

```
if __name__ == '__main__':
codigo
```

- name es una variable interna que se setea al arrancar el programa. Contiene main si se ejecuta como script
- un módulo que se importa una vez no puede ser reimportado, aunque haya cambiado
- para eso, usar reload, pero parece no ser muy estable o deseado

## Clases y objetos

```
class Point(object):
    attributes: x,y

class Rectangle(object):
    attributes: width,height,corner

box = Rectangle()
box.width = 100.0
box.height = 200.0
box.corner = Point()
box.corner.x = 0.0
box.corner.y = 0.0
```

• import copy: copia objetos (aunque no objetos dentro del objeto si los hubiese, así que ese objeto embebido sería el mismo para los 2 objetos creados a través de **copy.copy**) → usar **copy.deepcopy** 

## **Clases y funciones**

- funciones puras: no modifican los objetos
- funciones modificadoras : lo contrario

30/10/2025 21:17 5/5 think python

# Clases y métodos (17)

## otros

- raise : provoca/lanza una excepción
- variables globales:
  - para ser manipuladas en una función, se han de declarar previamente global <var> si son inmutables
  - si son mutables (listas, diccionarios), se puede añadir, borrar, modificar sin problema. Solo se tendrían que declarar en caso de reasignación

From:

https://miguelangel.torresegea.es/wiki/ - miguel angel torres egea

Permanent link:

https://miguelangel.torresegea.es/wiki/development:python:thinkpython?rev=1601552421

Last update: 01/10/2020 04:40

