STRINGS

Tipo de datos cadea ou string (tipo 'str')

- Python ten soporte para manexar texto ou secuencias de caracteres ➔ string
  - Van entre comillas simples, dobles, ou triples-dobles

```
#p2.3.1.py
x, y, z = "hola", 'mundo', """do
python"""
t = "hola \
mundo \
do \
python"
print x,y,z,t
print "hola\nmundo\ndo\npython"
```

```
$ python p2.3.1.py
hola mundo do
python hola mundo do python
hola
mundo
do
python
```

Strings

- Relacionados cos strings, Python dispón de:
  - operadores (+, *,...): concatenar e repetición de cadeas
  - funcións:
    - int("123")
    - float("123.0")
    - len("hola")
    - cadea = str(123)
    - ord('a')
    - chr(97), chr (ord('a'))
  - Métodos:
    - 'HOLA'. capitalize()
    - 'hola'.upper()
    - 'hola'.count('ol')

iremos vendo estas e moitas máis!!

- Os strings poden conter secuencias de escape ou caracteres de control

Secuencia de escape para carácter de control	Resultado
\a	Carácter de «campana» (BEL)
\b	«Espacio atrás» (BS)
\f	Alimentación de formulario (FF)
\n	Salto de línea (LF)
\r	Retorno de carro (CR)
\t	Tabulador horizontal (TAB)
\v	Tabulador vertical (VT)
ooo	Carácter cuyo código ASCII en octal es *ooo*
\x*hh*	Carácter cuyo código ASCII en hexadecimal es *hh*

```
#px.py
print 'A\nB\t\012C\x0aD'
```

```
$ python px.py
A
B C
D
```

Otras secuencias de escape	Resultado
\\	Carácter barra invertida (\)
\'	Comilla simple (')
\"	Comilla doble (")
\ y salto de línea	Se ignora (para expresar una cadena en varias líneas).

- Formateo de strings (operador %)

Formato	Conversión
%c	caracter
%s	string
%i	enteiro con signo
%d	enteiro con signo
%u	enteiro sen signo
%o	enteiro en formato octal
%x	enteiro en hexadecimal (letras minúsculas)
%X	enteiro en hexadecimal (letras maiúsculas)
%e	real en notación exponencial ('e' minúscula)
%E	real en notación exponencial ('E' maiúscula)
%f	Número real
%g	O real no formato máis curto entre %f and %e
%G	O real no formato máis curto entre %f and %E

Símbolo	Función
numero	indicar ancho ou precisión
-	xustificar á esquerda
+	amosar o signo do número
espacio	deixar un espazo en branco antes dun num positivo
#	Engadir o cero ('0') antes dun número octal, ou o '0x' or '0X' antes dun número hexadecimal cando 'x' ou 'X' son usados.
0	pad from left with zeros (instead of spaces)
%	%%' escribe un único '%'
m.n.	m= ancho total, e n = número de díxito decimais

```
print ".%s." % 'hola'
print ".%8s." % 'hola'
print ".%-8s." % 'hola'
print ".%d." % 16
print ".%+d." % -16
print ".%+d." % 16
print ".% d." % 16
print ".%6.3f." % 16
print ".%Xf." % 16
print ".%#Xf." % 16
```

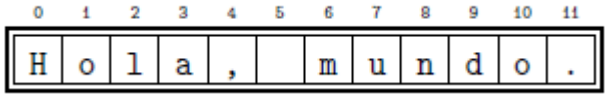
```
$ python strings.py
.hola.
.    hola.
.hola    .
.16.
.-16.
.+16.
. 16.
.16.000.
.10f.
.0X10f.
```

- Operadores de strings

Operador	Descripción	Exemplo (a="Hello" b="Python")
+	Concatenation - Adds values on either side of the operator	a + b ➔ HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 ➔ HelloHello
[ ]	Slice - Gives the character from the given index	a[1] ➔ e
[:]	Range Slice - Gives the characters from the given range	a[1:4] ➔ ell
in	Membership - Returns true if a character exists in the given string	'H' in a ➔ True
not in	Membership - Returns true if a character does not exist in the given string	'H' not in a ➔ False
r/R	Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.	print r'\n' ➔ imprime \n print R'\n' ➔ imprime \n
%	Format - Performs String formatting	-- formateo-- ¡xa o vimos!

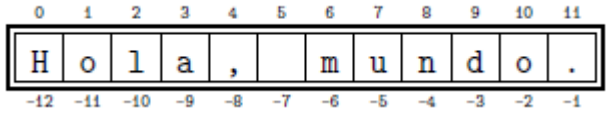
```
Escriba un programa que imprima
asdfasdfasdf =-=- =-=- =-=-????????asdfasdf
```

- Operadores de strings (II)
  - co operador [] de indexación.
    - podemos acceder a cada un dos caracteres que compón unha cadea.
    - pódese acceder desde a posición 0, á posición len(cadea)-1
      - Se ha cadea é "Hola, mundo."




```
s = "Hola, mundo."
print s[0],s[3]     ## imprime H e a
```

      - As posicións ou índices poden ser negativos ➔ son relativos á última posición do string (realmente á posición s[len(s)] que non sería válida)




```
s = "Hola, mundo."
print s[-1],s[-3], s[-12]     ## imprime . , d, e H
```

- Operadores de strings (II)
  - co operador s[i:j] obtemos un substring que inclúe os caracteres que van desde a posición **i** ata a **j-1**
    - De novo as posicións válidas van desde a 0 a len(s)-1

```
s = "Hola mundo"
print s[1:2]       ## imprime o
print s[0:4]       ## imprime Hola
print s[0:len(s)]  ## imprime Hola mundo
```

    - É opcional pór i ou j
      - s[: j] se non se pon i asúmese que interesa un substring desde o inicio de s
      - s[i :] se non se pon j asúmese que interesa un substring ata o final de s
      - s2= s[:], devolve unha copia do string (equivale a pór [0:len(s)])

```
s = "Hola mundo"
print s[:2]     ## imprime Ho
print s[3:]     ## imprime a mundo
print s[:]      ## imprime Hola mundo
```

      - s[inicio:fin:salto] pódese usar

```
s = "Hola mundo"
print s[::2]      ## imprime Hl ud
print s[::-1]                ## imprime odnum aloH
print s[-1:-len(s)-1:-1]     ## imprime odnum aloH
```

- Operadores de strings (III)
  - As cadeas son **INMUTABLES**!
    - Non podemos cambiala (aínda que sexa tentador) así s[0] = 'h'

```
#strings/p.inmutable.py
s = "Hola mundo"
s[0] = 'h'
```

```
$python p.inmutable.py
Traceback (most recent call last):
  File "p.inmutable.py", line 3, in <module>
    s[0] = 'h'
TypeError: 'str' object does not support item assignment
```

    - Como máximo podemos crear **outra cadea** usando operadores [ ] e +

```
#strings/p.inmutable.py
s = "Hola mundo"
s2= 'h'+s[1:]
print s,'---',s2
```

```
$python p.inmutable.py
Hola mundo --- hola mundo
```

- Percorrido de cadeas
  - Unha cadea é unha secuencia de caracteres e polo tanto é iterable cun FOR.
  - Pero tamén podemos iterar nós (pola nosa conta) por cada un dos seus elementos

```
s = "Hola, mundo."
for c in s:
    print c,
print ""
for i in range(len(s)):
    print s[i],
print ""

i=0
while (i<len(s)):
    print s[i],
    i+=1
```

```
$ python strings2.py
H o l a ,   m u n d o .
H o l a ,   m u n d o .
H o l a ,   m u n d o .
```

Strings. Métodos e funcións

http://www.tutorialspoint.com/python/python_strings.htm
https://librosweb.es/libro/python/capitulo_6/metodos_de_formato.html

- Algúns son métodos da *clase* string, outros son funcións aplicadas a strings
  - Métodos ➔"hola".capitalice()
  - Funcións ➔ len("hola")

- capitalize() : Capitalizes first letter of string

```
>>> print "adeus mundo cruel...".capitalize()
Adeus mundo cruel...
```

- lower() : Converts all uppercase letters in string to lowercase.

```
>>> print "ADEUS mundo CRUel...".lower()
adeus mundo cruel...
```

- swapcase() : Inverts case for all letters in string.
- upper(): Converts lowercase letters in string to uppercase.

```
>>> print "ADEUS mundo CRUel...".upper()
ADEUS MUNDO CRUEL...
```

- title() : Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.

```
>>> print "ADEUS mundo CRUel...".title()
Adeus Mundo Cruel...
```

Strings. Métodos

http://www.tutorialspoint.com/python/python_strings.htm
https://librosweb.es/libro/python/capitulo_6/metodos_de_formato.html

- isalnum() : Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.

```
>>> print "ADEUS mundo CRUel...".isalnum()
>>> print "ADEUSmundoCRUel123".isalnum()
False
True
```

- isalpha() : Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
- isdigit() : Returns true if string contains only digits and false otherwise.
- islower() : Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
- isnumeric() : Returns true if a unicode string contains only numeric characters and false otherwise.
- isspace() : Returns true if string contains only whitespace characters and false otherwise.

```
>>> print "\t   \n".isspace(), ;   print "   ".isspace()    ## \t e \n considéranse espazos!!
True True
```

- istitle() : Returns true if string is properly "titlecased" and false otherwise.
- isupper() : Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
- isdecimal() : Returns true if a unicode string contains only decimal characters and false otherwise.

Strings. Métodos

http://www.tutorialspoint.com/python/python_strings.htm
https://librosweb.es/libro/python/capitulo_6/metodos_de_formato.html

- max(str) : Returns the max alphabetical character from the string str.
- min(str) : Returns the min alphabetical character from the string str.
- ord(char): Returns ordinal position in ASCII table
- char(int): Returns char for a given position in ASCII table
- len(string) : Returns the length of the string
- lstrip() : Removes all leading whitespace in string.
- rstrip() : Removes all trailing whitespace of string.
- strip([chars]) : Performs both lstrip() and rstrip() on string

```
>>>#strings/p1.methodos.py
>>> cadea = "    quitar espazos\t \n   "
>>> print "--%s--" % (cadea.strip())
>>> print "--%s--" % (cadea.strip("\n"))
>>> print "--%s--" % (cadea.rstrip())
>>> print "--%s--" % (cadea.lstrip())
>>> print "len %d --> stripped %d" % (len(cadea), len(cadea.strip()) )
>>> print "min '%s' , max '%4s'" % (min(cadea), max(cadea) )
>>> print "min '%4d', max '%4s'" % (ord(min(cadea)), ord(max(cadea)) )
>>> cadea ="www.gmail.com"
>>> print "--%s--" % (cadea.strip("wmo."))
>>> print "--%s--" % (cadea.strip("w"))
--quitar espazos--
--    quitar espazos--
--quitar espazos
   --
len 24 --> stripped 14
min '   ' , max '   z'
min '   9', max ' 122'
--gmail.c--
--.gmail.com--
```

Strings. Métodos

http://www.tutorialspoint.com/python/python_strings.htm
https://librosweb.es/libro/python/capitulo_6/metodos_de_formato.html

- center(width[, fillchar]) : Returns a space-padded string with the original string centered to a total of width columns.
- ljust(width[, fillchar]) : Returns a space-padded string with the original string left-justified to a total of width columns.
- rjust(width,[, fillchar]) : Returns a space-padded string with the original string right-justified to a total of width columns.
- zfill (width) : Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).

```
>>>#strings/p1.methodos.py
>>>cadea = "o mellor do mundo mundial".capitalize()
>>>print cadea.center(50)
>>> print cadea.center(50, "-")
>>> print cadea.ljust(50, "-")
>>> print cadea.rjust(50, "-")
>>> print "hola".zfill(10)
>>> print "54321".zfill(10)

 O mellor do mundo mundial
------------O mellor do mundo mundial-------------
O mellor do mundo mundial------------------------
------------------------O mellor do mundo mundial
000000hola
0000054321
```

Strings. Métodos

http://www.tutorialspoint.com/python/python_strings.htm
https://librosweb.es/libro/python/capitulo_6/metodos_de_formato.html

- count(str, beg= 0,end=len(string)) : Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
- find(str, beg=0, end=len(string)) : Determines if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
- index(str, beg=0, end=len(string)) : Same as find(), but raises an exception if str not found.
- rfind(str, beg=0,end=len(string)) : Same as find(), but search backwards in string.
- rindex(str, beg=0, end=len(string)) : Same as index(), but search backwards in string.

```
>>> #strings/p1.methodos.count.find.py
>>> texto = "mississippi e un rio do estado Mississipii"
>>> patron_buscar = "ssi"
>>> numero   = texto.count(patron_buscar)
>>> posicion = texto.find(patron_buscar)

>>> print texto
>>> print "%s aparece %d veces" %(patron_buscar,numero)
>>> print "a primeira na posicion: %d" %(posicion)
>>> posicion = texto.find(patron_buscar,posicion+1)
>>> print "a segunda na posicion: %d" %(posicion)

mississippi e un rio do estado Mississipii
ssi aparece 4 veces
a primeira na posicion: 2
a segunda na posicion: 5
```

Strings **** FIND_ALL_OCCS **** (propio)

http://www.maestrosdelweb.com/guia-python-cadenas-de-texto/

BUSCAR TODAS AS OCCURRENCIAS DUN PATRÓN NUN TEXTO
crea unha lista baleira e vai engadindo as posicións onde se atopa

```
##p1.methodos.count.find.all.list.py
texto = "mississippi e un rio do estado Mississipii"
patron_buscar = "ssi"

posicions = []
pos = texto.find(patron_buscar, 0)  #buscamos a primeira ocurrencia se a hai

while pos != -1:
    posicions.append(pos)
    # cada vez buscamos desde un caracter máis adiante da última ocurrencia encontrada
    pos = texto.find(patron_buscar, pos+1)
else: # cando xa non se atope a letra
    num =len(posicions)
    #num = texto.count(patron_buscar)
    print "%d posicions de '%s' en '%s'" %(num, patron_buscar,texto)
    print posicions

$python p1.methodos.count.find.all.list.py
4 posicions de 'ssi' en 'mississippi e un rio do estado Mississipii'
[2, 5, 33, 36]

$python p1.methodos.count.find.all.list_index_excepcion.py
4 posicions de 'ssi' en 'mississippi e un rio do estado Mississipii'
[2, 5, 33, 36]
```

- format(args,**kwargs) : Gives format to a string, replacing text dinamically
  - é similar ao *print con formato*
  - args é a cadea de formato (onde se reemplaza)

    sustitúese o que vaia entre {0},{1},{2}... polos elementos da lista kwargs
  - **kwargs é unha lista de argumentos

```
>>> cadea = "Aprendo a usar format {0}"
>>> cadea = cadea.format("en Python")
>>> cadea = "Se tiña {0}eur e gasto {1}eur, quedanme {2}eur".format(100,45,100-45)
>>> print cadea
>>> print cadea
>>> cadea = "Se tiña {1}{0} e gasto {2}{0}, quedanme {3}{0}".format("eur",100,45,100-45)

Aprendo a usar format en Python
Se tiña 100eur e gasto 45eur, quedanme 55eur
Se tiña 100eur e gasto 45eur, quedanme 55eur
```

- replace(old, new [, max]) : Replaces all occurrences of old in string with new. Replaces at most max occurrences if max given.

Strings. Métodos

- format(args,**kwargs) : Gives format to a string, replacing text dinamically

- replace(old, new [, max]) : Replaces all occurrences of old in string with new. Replaces at most max occurrences if max given.

```
>>> cadea = "\
>>> En <cidade> a <data>,\n\
>>> Estimado <nome>,\n\
>>> En relacion ao escrito presentado \
>>> por <nome> <apelidos> quero expresar..."

>>> cadeaR=cadea.replace("<data>","1 de decembro de 2014")
>>> print cadeaR
>>> print "----------------------------"
>>> cadeaR=cadea.replace("<data>","1 de decembro de 2014").replace("<cidade>","A
   Corunha").replace("<nome>", "Juan").replace("<apelidos>","Fernandez")
>>> print cadeaR

En <cidade> a 1 de decembro de 2014,
Estimado <nome>,
En relacion ao escrito presentado por <nome> <apelidos> quero expresar...
----------------------------
En A Corunha a 1 de decembro de 2014,
Estimado Juan,
En relacion ao escrito presentado por Juan Fernandez quero expresar...
```

Strings. Métodos

http://www.tutorialspoint.com/python/python_strings.htm
https://librosweb.es/libro/python/capitulo_6/metodos_de_formato.html

- expandtabs(tabsize=8) : Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
- maketrans() : Returns a translation table to be used in translate function.
- translate(table, deletechars="") : Translates string according to translation table str(256 chars), removing those in the del string.
- endswith(suffix, beg=0, end=len(string)) : Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
- startswith(str, beg=0,end=len(string)) : Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise.

Strings. Métodos

http://www.tutorialspoint.com/python/python_strings.htm
https://librosweb.es/libro/python/capitulo_6/metodos_de_formato.html

- join(seq) : Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
- split(str="", num=string.count(str)) : Splits string according to delimiter str (space by default) and returns list of substrings; split into at most num substrings if given.
  - split () == split("") equivale a cortar por espacios. Es diferente a split(" ")!
- splitlines(num=string.count('\n')) : Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.
- decode(encoding='UTF-8',errors='strict') : Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.
- encode(encoding='UTF-8',errors='strict') : Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.

```
>>> fecha = '17/05/2012'
>>> datos = fecha.split('/') # separamos a cadea por /
>>> print datos
['17', '05', '2012']         # a lista contén os pedazos, sen o separador
>>> print 'día:', datos[0], 'mes:', datos[1], 'año:', datos[2]
día: 17 mes: 05 año: 2012
>>> print "-".join(lista)
17-05-2012
```

LISTAS

Listas []

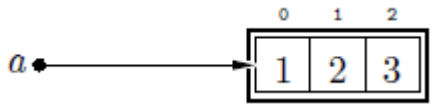
- Python ten soporte para secuencias de valores de calquera tipo ➔ listas de valores
  - Para crear unha lista, simplemente temos que pór eses valores entre corchetes, e separados por comas:

    - froitas= ['pera','laranxa','mazá','uva']
    - nome = ['Juan', 'Ana', 'Sergio', 'Jorge', 'Alberto','Elena']
    - numeros= [1,3,2,4,6,5]
    - varios = [1, 'Juan', 1.78]
    - varios2 = [1, 'Juan', ['pera','laranxa']]

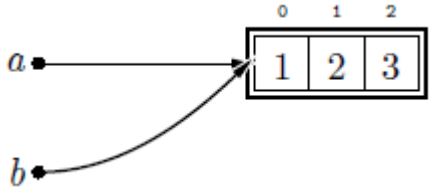
  - Unha lista baleira podémola inicializar así: lista = []

Listas []

- unha variable de tipo lista almacena unha referencia a un obxecto lista (como pasa sempre)
  - a= [1,2,3]



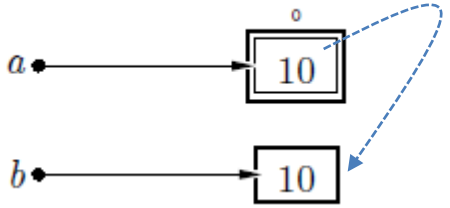
- Se asignamos a outra variable unha variable lista, simplemente se copia a referencia á lista: Non se duplica a lista!
  - a=[1,2,3]
  - b=a



  - así que se modifico un elemento da lista (unha lista é mutable) con b, ¡tamén a verá que foi modificado!

Listas []

- Unha lista que contén un elemento é diferente a ter simplemente ese dato
  - [10] != 10 ["hola"] != "hola"
    - a= [10]
    - b=10



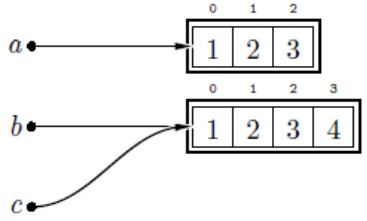

```
a = [10]
b =  10
print a
print b
print type(a)
print type(b)
print a==b
```

```
[10]
10
<type 'list'>
<type 'int'>
False
```

Listas []

- Operadores/ funcións (I): *(coñecidas de strings)*
  - Operador de concatenación de listas: +
  - Operador de replicado elementos dunha lista : *
    - Xeran **novas** listas a partir da orixinal (non modifican o orixinal)

```
a = [1,2,3]
b = a + [4]
c = b
```



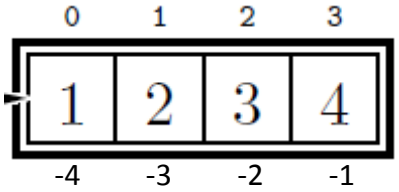
    - Exemplos:

```
a = [1,2,3]
print a       ➔ [1, 2, 3]
print a + [4] ➔ [1, 2, 3, 4]
print a * 3   ➔ [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

  - len(lista):
    - Indica o número de elementos da lista
      - len (a) ➔3 , len ([]) ➔ 0 , len([1,3]*4) ➔ 8

Listas []

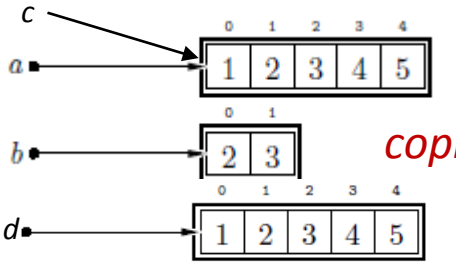
- Operadores/ funcións (II): *(coñecidas de strings)*
  - Operador de indexación [ ]
    - Acceder ao i-ésimo elemento da lista: lista[i]




```
a = [1,2,3,4]
print a[0]    ➔ 1
print a[1]    ➔ 2
print a[-1] ➔ 4
print [1,2,3,4][2] ➔ 3
print [][2] ➔ erro (lista baleira)
```

  - Operador *slice* ou corte [:], devolve unha nova lista que se obtén copiando os elementos da lista orixinal
    - b=a[:] → copia completa da lista
    - b= a[1:3] → elementos nas posicóns 1 e 2

```
a = [1,2,3,4,5]
c = a       ## orixinal
b = a[1:3] ## copia parcial
d=a[:]      ## copia completa
```



copia! non é a lista orixinal

Listas []

- Operadores/ funcións (III):
  - Comparación de listas:
    - Operador **==** (a == b), ... tamén podemos usar o **!=** (desigualdade)
      - Compara se as listas teñen os mesmos elementos
        - » Se as listas son de diferente lonxitude ➔ false
        - » Se as listas teñen a mesma lonxitude:
          - Se todos os elementos son iguais ➔ True
          - Se algún elemento é diferente ➔ False
    - Operador **is / is not** : (a is b)
      - compara se as variables a e b fan referencia ao mesmo obxecto <lista>

- Exemplos:

```
a = [1,2,3]
b = [1,2,3]
c=a
print a==b   ##imprime True
print a is b ##imprime False
print a is c ##imprime True
```

- Operadores/ funcións (IV):
  - Comparación de listas:
    - Operadores **<, >, <=, >= , !=**
      - Compara os elemenos un a un (ex. a<b)
        - » se a[0] < b[0] ➔ True
        - » se a[0] > b[0] ➔ False
        - » se a[0]== b[0]
          - se a[1] < b[1] ➔ True
          - se a[1] > b[1] ➔ False
          - se a[1] == b[1]
            - se a[2] < b[2] ➔ True
            - ...

- Exemplos:

```
a = [1,2,3]
b = [1,3,1]
c = [1,2]
print a<b   ##imprime True
print a<c   ##imprime False
print a>c   ##imprime True
```

Listas []

- Operadores/ funcións (V):
  - Pertenencia ou non a unha lista
    - Operadores **in / not in**
      - Indica se un elemento aparece nunha lista ou non
    - Exemplos:

```
a = [1,2,3]
print 1 in a ##imprime True
print 4 in a ##imprime false
```

Listas []

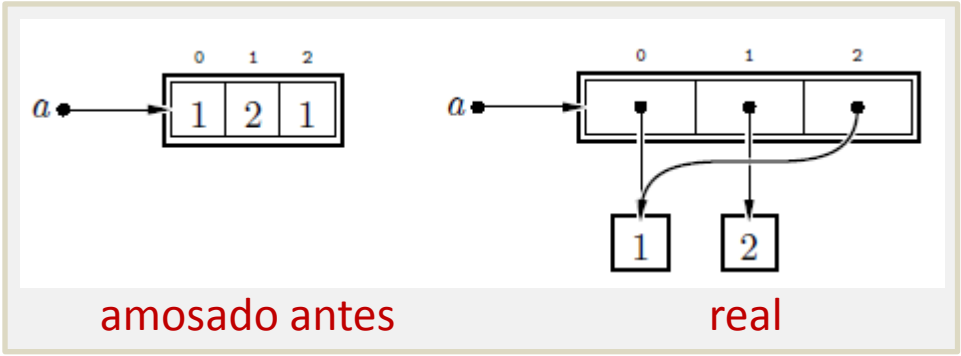
- Percorrido de listas *(como xa vimos en strings...)*
  - Unha listas é unha secuencia de valores e polo tanto é iterable cun FOR
  - Tamén podemos iterar directamente polos seus elementos usando o operador [i]

```
lista = [1,"hola",2.85]
for v in lista:
    print v,
print ""

for i in range(0,len(lista)):
    print lista[i],
print ""

i=0
while (i<len(lista)):
    print lista[i],
    i = i+1
```

```
1 hola 2.85
1 hola 2.85
1 hola 2.85
```

- As listas son mutables
  - podemos modificar os seus elementos (lista[i]=valor), e tamén engadir/borrar elementos da lista
    - Aínda que por agora representamos unha lista como se ve no debuxo esquerdo,

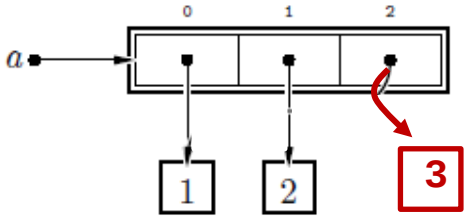
      a= [1,2,1]



... realmente Python garda en cada posición da lista unha referencia ao obxecto correspondente (ver dereita), nótese que a[0] e a[2] fan referencia a un obxecto que contén o enteiro **1**.

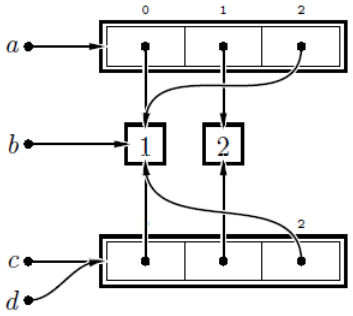
- Modificar un elemento da lista significa que esa referencia se modifica

  a[2] = 3

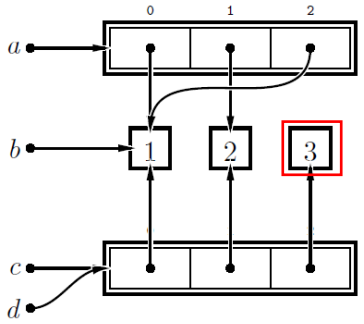


- As listas son mutables
  - Outro exemplo:

```
a = [1,2,1]
b = 1
c = [1,2,1]
d=c
print a[0] == b     ## True
print a[0] is b     ## True
print c[-1] is a[0] ## True
```



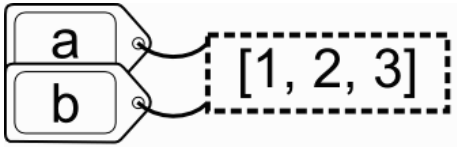

```
a = [1,2,1]
b = 1
c = [1,2,1]
d = c
d[2] = 3
print d ## imprime [1,2,3]
print c ## imprime [1,2,3]  ¡c foi modificada!
```

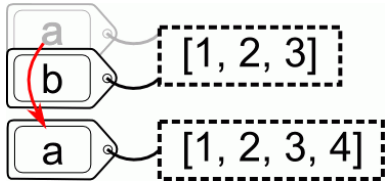


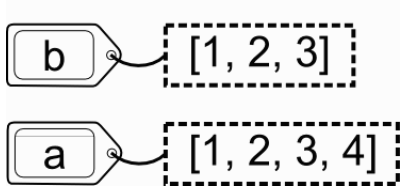
Listas []: Modificación de listas ("mutables")

- x = x+ [4] ##crea nova lista

```
a= [1,2,3]
b=a
print a,b   #[1, 2, 3]  [1, 2, 3]
a = a + [4] ## crea unha nova lista [1,2,3,4] asígnaselle a a
print a,b   #[1, 2, 3,4]  [1, 2, 3]
```



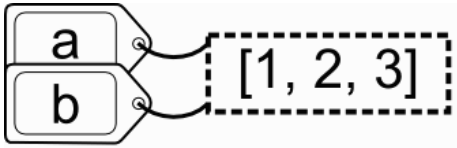


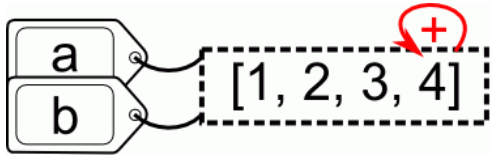


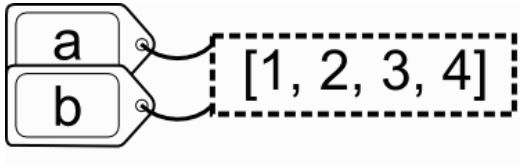
OLLO!!!!

- x += [4] Ou x.append (4) ##modifica lista actual

```
a= [1,2,3]
b=a
print a,b   #[1, 2, 3]  [1, 2, 3]
a += [4]  ou tamén a.append(4)  ## modifícase a lista
print a,b   #[1, 2, 3, 4]  [1, 2, 3, 4]
```





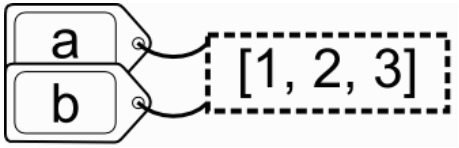


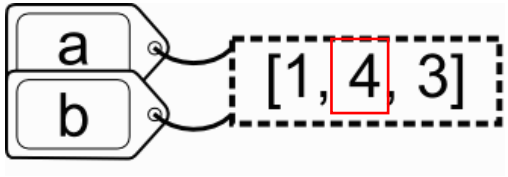
http://www.mclibre.org/consultar/python/lecciones/python_variables_2.html

Listas []: Modificación de listas ("mutables")

- Modificar un valor dunha lista modifica o obxecto: x[1]=k

```
a= [1,2,3]
b=a
print a,b   #[1, 2, 3] [1, 2, 3]
a[1] = 4    ## modifícase o obxecto
print a,b   #[1, 4, 3] [1, 4, 3]
```

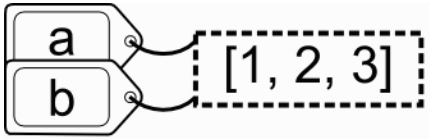


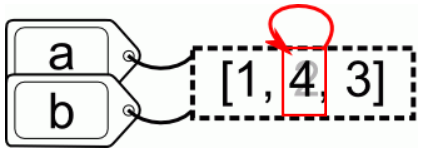


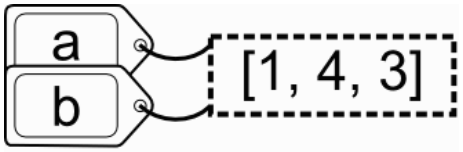
- Modificando o valor dunha sublista… tamén se modifica o obxecto

```
a= [1,2,3]
b=a
print a,b   #[1, 2, 3] [1, 2, 3]
a[1:2] = [4]  ## modifícase a lista
print a,b   #[1, 4, 3] [1, 4, 3]
```

OLLO isto é válido





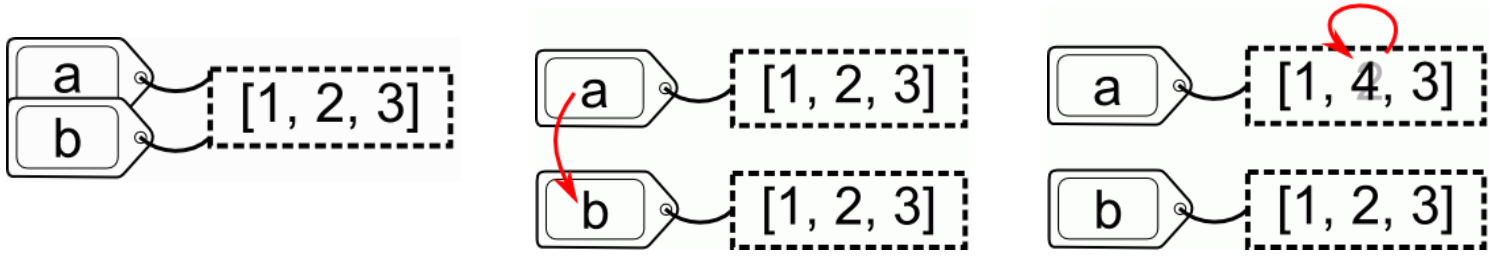


http://www.mclibre.org/consultar/python/lecciones/python_variables_2.html

Listas []: Facer unha copia dunha lista

- Para obter unha copia dunha lista cómpre usar a notación de sublistas
  - a=[1,2,3]
  - b=a[:]

```
a= [1,2,3]
b=a[:]
print a,b   #([1, 2, 3], [1, 2, 3])
a[1] = 4    ## modifícase o obxecto asociado a a
print a,b   #([1, 4, 3], [1, 2, 3])
```



http://www.mclibre.org/consultar/python/lecciones/python_variables_2.html

Listas: Funcións e métodos

http://www.tutorialspoint.com/python/python_lists.htm
https://librosweb.es/libro/python/capitulo_7.html

- Funcións:
  - cmp(list1, list2): Compara os elementos das dúas listas
  - **len(list):** Devolve a lonxitude da lista

```
>>> print len([1,2,3,4])
##imprime 4
```

  - max(list):Devolve o elemento con maior valor
  - min(list): Devolve o elemento con menor valor
  - list(seq): convirte unha *tupla* nunha lista

```
>>> tupla1 = (123, 'xyz', 'zara', 'abc')
>>> lista= list(tupla1)
>>> print lista
##imprime [123, 'xyz', 'zara', 'abc']
```

  - del lista[i] : elimina o i-esimo elemento da lista

```
list1 = ['physics', 'chemistry', 1997, 2000];
print list1
del list1[2];
print "After deleting value at index 2 : "
print list1
##imprime ['physics', 'chemistry', 1997, 2000]
##imprime After deleting value at index 2 :
##imprime['physics', 'chemistry', 2000]
```

véxase tamén método list1.remove(1997)

Listas: Funcións e métodos

http://www.tutorialspoint.com/python/python_lists.htm
https://librosweb.es/libro/python/capitulo_7.html

- Métodos do obxecto list:
  - list.append(obj): Appends object obj to list

```
>>> aList = [123, 'xyz', 'zara', 'abc'];
>>> aList.append( 2009 );
>>> print "Updated List : ", aList
##imprime [123, 'xyz', 'zara', 'abc', 2009]
```

  - list.extend(seq): Appends the contents of seq to list

```
>>> aList = [123, 'xyz', 'zara', 'abc', 123];
>>> bList = [2009, 'manni'];
>>> aList.extend(bList)
>>> print "Extended List : ", aList
##imprime [123, 'xyz', 'zara', 'abc', 123, 2009, 'manni']
```

  - list.insert(index, obj): Inserts object obj into list at offset index

```
>>> aList = [123, 'xyz', 'zara', 'abc']
>>> aList.insert( 3, 2009)
>>> print "Final List : ", aList
##imprime  [123, 'xyz', 'zara', 2009, 'abc']
```

  - list.pop(index=-1): Removes object by index. Default index = -1. It also returns the removed object

```
>>> aList = [123, 'xyz', 'zara', 'abc'];
>>> print "A List : ", aList.pop()   ##imprime 'abc'
>>> print "B List : ", aList.pop(2)  ##imprime 'zara'
```

  - list.remove(obj): Removes object obj from list (only 1st occurrence)

```
>>> aList =[123, 'xyz', 'zara', 'abc', 'xyz'];
>>> aList.remove('xyz');
>>> print "List : ", aList     ##imprime [123, 'zara', 'abc', 'xyz']
>>> aList.remove('abc');
>>> print "List : ", aList     ##imprime [123, 'zara', 'xyz']
```

Listas: Funcións e métodos

http://www.tutorialspoint.com/python/python_lists.htm
https://librosweb.es/libro/python/capitulo_7.html

- Métodos do obxecto list:
  - list.count(obj): Returns count of how many times obj occurs in list

```
>>> aList = [123, 'xyz', 'zara', 'abc', 123];
>>> print "Count for 123 : ", aList.count(123)     ##imprime 2
>>> print "Count for zara : ", aList.count('zara') ##imprime 1
```

  - list.index(obj): Returns the lowest index in list that obj appears

```
>>> aList = [123, 'xyz', 'zara', 'abc'];
>>> print "Index for xyz : ", aList.index( 'xyz' )   ##imprime 1
>>> print "Index for zara : ", aList.index( 'zara' ) ##imprime 2
```

  - list.reverse():Reverses objects of list in place

```
>>> aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
>>> aList.reverse();
>>> print "List : ", aList
##imprime  ['xyz', 'abc', 'zara', 'xyz', 123]
```

  - list.sort([func]):Sorts objects of list, use compare func if given (optional, otherwise af-num sorting)

```
>>> aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
>>> aList.sort();
>>> print "List : ", aList
##imprime [123, 'abc', 'xyz', 'xyz', 'zara']
```

Listas [] e matrices

- Unha lista pode conter outra lista ➔ podemos crear matrices

```
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Acceso usando []

```
print matriz[1]  ➔ [4, 5, 6]  ## é unha lista
print matriz[1][1] == 5  ## é o segundo elemento da
                         ## lista [4,5,6]
                         ## equivale a [4,5,6][1]
```

Listas [] e iteradores

- métodos range() e xrange()
  - range(10)
  - range(1,5)
  - range (1,100,2) ... l,r,step
  - devolven unha lista de valores no rango desexado

```
total=0
for i in xrange(10):
   total +=i
print total
```

```
$ python px.py
45
```

```
# crea a lista (en memoria) e despois súmase
sum_of_first_n = sum(range(1000000))

# usa un "xerador", que crea a lista "elemento a elemento" e xa se
vai sumando. Usa menos memoria, e é máis rápido.
sum_of_first_n = sum(xrange(1000000))
```