Lenguaje y built-ins

Strings

Secuencia de caracteres encerrados entre comillas simples o dobles

Strings

```
>>> s = 'hola'
>>> print s[1]
o
>>> print len(s)
4
>>> print s + ' che'
hola che
```

Strings

Métdos de string

- s.lower(), s.upper()
- s.strip()
- s.isalpha()/s.isdigit()/s.isspace()...
- s.startswith('other'), s.endswith('other')
- s.find('other')
- s.replace('old', 'new')
- s.split('delim')
- s.join(list)

Strings: slices

Strings: slices

```
>>> s = "hola"
>>> s[1:2]
'o'
>>> s[1:4]
'ola'
>>> s[1:]
'ola'
>>> s[:]
'hola'
>>> s[1:100]
'ola'
>>> s[-1]
'a'
>>> s[-4]
'h'
>>> s[:-3]
'h'
>>> s[-3:]
'ola'
```

Strings: operador %

```
>>> "En %d tristes trastos de %s, %d tristes %s comían %s." % (3, "trigo", 3, "tigres", "trigo")
'En 3 tristes trastos de trigo, 3 tristes tigres comian trigo.'
>>>
>>> print "Today's stock price: %f" % 50.4625
Today's stock price: 50.462500
```

Lists

Una lista es un conjunto ordenado de elementos encerrados entre corchetes

Una lista de Python es como un array en Perl.

Una lista en Python sería similar a ArrayList en Java puesto que puede contener **objetos arbitrarios** y **expandirse de forma dinámica** según se añaden otros nuevos.

Implementación de listas:
http://www.laurentluce.com/posts/python-list-implementation/

Lists

```
>>> li = ["manzana", "pera", "naranja", 9, 1.2]
>>> li
["manzana", "pera", "naranja", 9, 1.2]
>>> li[0]
'manzana'
>>> li[3]
9
>>>  list = []
>>>  list.append('a')
>>>  list.append('b')
```

Lists

```
>>> li[:3]
['manzana', 'pera', 'naranja']
>>> li[3:]
[3, 1.2]
>>> li[:]
['manzana', 'pera', 'naranja', 3, 1.2]
```

Lists

```
>>> frutas = li
>>> frutas
['manzana', 'pera', 'naranja', 3, 1.2]
>>> li
['manzana', 'pera', 'naranja', 3, 1.2]
>>> li.pop()
1.2
>>> li
['manzana', 'pera', 'naranja', 3]
>>> frutas
['manzana', 'pera', 'naranja', 3]
```

Lists

```
>>>  squares = [1, 4, 9, 16]
>>>  sum = 0
>>>  for num in squares:
>>>      sum += num
>>>  print sum  ## 30
>>>
>>>  list = ['larry', 'curly', 'moe']
>>>  if 'curly' in list:
>>>    print 'yay'
```

Lists

```
>>> range(100)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
>>> xrange(100)
xrange(100)
>>> xrange(100)[10]
10
>>> range(100)[10]
10
```

Lists

```
>>> a = range(10)
>>> while i < len(a):
...     print a[i]
...     i = i + 3
...
0
3
6
9
```

Lists

Métdos de lists

- list.append(elem)
- list.insert(index, elem)
- list.extend(list2)
- list.index(elem)
- list.remove(elem)
- list.sort()
- list.reverse()
- list.pop(index)

List Comprehensions

```
>>>  nums = [1, 2, 3, 4]
>>>  squares = [ n * n for n in nums ]
[1, 4, 9, 16]

>>> strs = ['hello', 'and', 'goodbye']
>>> shouting = [ s.upper() + '!!!' for s in strs ]
['HELLO!!!', 'AND!!!', 'GOODBYE!!!']
```

List Comprehensions

```
>>>  nums = [2, 8, 1, 6]
>>>  small = [ n for n in nums if n <= 2 ]
>>>  small
[2, 1]

>>> ## Select fruits containing 'a', change to upper case
>>>  fruits = ['apple', 'cherry', 'bannana', 'lemon']
>>>  afruits = [ s.upper() for s in fruits if 'a' in s ]
>>> afruits
['APPLE', 'BANNANA']
```

Lists y Strings

```
>>> li = ['server=mpilgrim', 'uid=sa', 'database=master', 'pwd=secret']
 s = ";".join(li)
>>> s
'server=mpilgrim;uid=sa;database=master;pwd=secret'
>>> s.split(";")
['server=mpilgrim', 'uid=sa', 'database=master', 'pwd=secret']
>>> s
'server=mpilgrim;uid=sa;database=master;pwd=secret'
>>> s.split(";",1)
['server=mpilgrim', 'uid=sa;database=master;pwd=secret']
```

Ordenando Listas

```
>>> a = [5, 1, 4, 3]
>>> print sorted(a)
[1, 3, 4, 5]
>>> print a
[5, 1, 4, 3]

>>> strs = ['aa', 'BB', 'zz', 'CC']
>>> print sorted(strs)
['BB', 'CC', 'aa', 'zz']
>>> print sorted(strs, reverse=True)
['zz', 'aa', 'CC', 'BB']
```

Ordenando Listas

```
>>> strs = ['ccc', 'aaaa', 'd', 'bb']
>>> print sorted(strs, key=len)
['d', 'bb', 'ccc', 'aaaa']

>>> print sorted(strs, key=str.lower)
['aa', 'BB', 'CC', 'zz']
>>> strs = ['xc', 'zb', 'yd' ,'wa']

>>> def MyFn(s):
...         return s[-1]
>>>
>>> print sorted(strs, key=MyFn)
['wa', 'zb', 'xc', 'yd']
```

Tuples

Secuencia inmutable. Una tupla no puede ser modificada una vez creada.

Son mas eficientes que las listas

Protegidas contra escritura

Pueden ser usadas como claves en diccionarios

Pueden ser usadas en formateo de strings

Tuples

```
>>> t = ("manzana", "pera", "naranja", 3, 1.2)
>>> t
('manzana', 'pera', 'naranja', 3, 1.2)
>>> t.append
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> t.remove
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'remove'
```

Diccionarios

Definen una relación uno a uno entre claves y valores.

Como un hash en Perl

Como una instancia de clase Hashtable en Java

Como una instancia de Scripting.Dictionary en VB

Diccionarios

```
>>> d = {"server":"mpilgrim", "database":"master"}
>>> d
{'server': 'mpilgrim', 'database': 'master'}
>>> d["server"]
'mpilgrim'
>>> d["database"]
'master'
>>> d["mpilgrim"]
Traceback (innermost last):
  File "<interactive input>", line 1, in ?
KeyError: mpilgrim
```

Diccionarios

```
>>> d
{'server': 'mpilgrim', 'database': 'master'}
>>> d["database"] = "pubs"
>>> d
{'server': 'mpilgrim', 'database': 'pubs'}
>>> d["uid"] = "sa"
>>> d
{'server': 'mpilgrim', 'uid': 'sa', 'database': 'pubs'}
```

Diccionarios

```
>>> d = {}
>>> d["key"] = "value"
>>> d["key"] = "other value"
>>> d
{'key': 'other value'}
>>> d["Key"] = "third value"
>>> d
{'Key': 'third value', 'key': 'other value'}
```

Diccionarios

```
>>> d
{'server': 'mpilgrim', 'uid': 'sa', 'database': 'pubs'}
>>> d["retrycount"] = 3 1
>>> d
{'server': 'mpilgrim', 'uid': 'sa', 'database': 'master', 'retrycount': 3}
>>> d[42] = "douglas"   2
>>> d
{'server': 'mpilgrim', 'uid': 'sa', 'database': 'master',
42: 'douglas', 'retrycount': 3}
```

Diccionarios

```
>>> mydict
{}
>>> mydict[[1]] =2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> mydict[(1,)] =2
>>> mydict
{(1,): 2}
```

Diccionarios

```
>>> d
{'server': 'mpilgrim', 'uid': 'sa', 'database': 'master', 42: 'douglas', 'retrycount': 3}
>>> del d[42]
>>> d
{'server': 'mpilgrim', 'uid': 'sa', 'database': 'master', 'retrycount': 3}
>>> d.clear()
>>> d
{}
```

Set

Conjuntos donde cada valor puede aparecer una sola vez

Como un Set::Scalar en Perl

Como una instancia de clase HashSet en Java

Hay una version inmutable llamada frozenset()

Sets

```
>>> d = {1, 2, 3, 1}
>>> d
{1, 2, 3}
>>>  d.intersection([1,4,5])
{1}
>>>  d.union([6, 7, 8])
{1, 2, 3, 6, 7, 8}
>>> d.update([1, "hola"])
>>> d
{1, 2, 3, 'hola'}
```

Set

Tienen muchos más métodos de manipulación de conjuntos.

Para utilizar un frozenset tienen que utilizar

Directamente `frozenset({1, 3, 3})`

Valores de verdad

```
>>> if False:
...     print 'false'
...
>>> if True:
...     print 'true'
...
true

>>> if []:
...     print 'false'
...
>>> if not []:
...     print 'not false'
...
not false
```

If, elif

```
>>> def nro_palabra(a):
...     if a == 0:
...             return 'cero'
...     elif a == 1:
...             return 'uno'
...     elif b == 2:
...             return 'dos'
...     else:
...             return 'mucho'
```

If

```
>>> 'a' and 'b'
'b'
>>> '' and 'b'
''
>>> 'a' and 'b' and 'c'
c


>>> 'a' or 'b'
'a'
>>> '' or 'b'
'b'
>>> '' or [] or {}
{}
```