



Taller de Tecnologías de Producción de Software

*Técnicas y Estrategias para la Resolución de
Problemas*



Python Estructuras de datos

List

Las listas en Python son indexadas, mutables, y pueden contener elementos de cualquier tipo.

```
>>> l = [1, 2, 3]
```

```
>>> l[0]
```

```
1
```

```
>>> l.append(1)
```

```
>>> l
```

```
[1, 2, 3, 1]
```

```
>>> l.append(['d', 'e', 'f'])
```

```
>>> l
```

```
[1, 2, 3, 1, ['d', 'e', 'f']]
```

```
>>> l.extend(['d', 'e', 'f'])
```

```
>>> l
```

```
[1, 2, 3, 1, ['d', 'e', 'f'], 'd', 'e', 'f']
```

Más: <https://docs.python.org/2/tutorial/datastructures.html#more-on-lists>



Tuple

Las tuplas son muy parecidas a las listas, pero su manipulación es más eficiente, ya que son inmutables.

```
>>> t = (1,2,3,1)
```

```
>>> t.count(1)
```

```
2
```

```
>>> t.index(2)
```

```
1
```

```
>>> data = (1,2,3)
```

```
>>> x, y, z = data
```

```
>>> x
```

```
1
```

Más: <https://docs.python.org/2/tutorial/datastructures.html#tuples-and-sequences>



Dict

El diccionario es una secuencia no ordenada de elementos, del tipo clave valor.

```
>>> d = {'a':'value', 'b':[1,2]}
```

```
>>> d.keys()
```

```
['a', 'b']
```

```
>>> d.values()
```

```
['value', [1, 2]]
```

```
>>> d['a']
```

```
'value'
```

```
>>> d.items()
```

```
[('a', 'value'), ('b', [1, 2])]
```

```
>>> d.has_key('a')
```

```
True
```

Más: <https://docs.python.org/2/library/stdtypes.html#typesmapping>



Set

Conjunto no ordenado de valores, que no permite duplicados.

```
>>> a = set([1, 2, 3, 4, 4])
```

```
>>> b = set([3, 4, 5, 6, 6])
```

```
>>> a | b # Union
```

```
{1, 2, 3, 4, 5, 6}
```

```
>>> a & b # Intersection
```

```
{3, 4}
```

```
>>> a < b # Subset
```

```
False
```

```
>>> a - b # Difference
```

```
{1, 2}
```

```
>>> a ^ b # Symmetric Difference
```

```
{1, 2, 5, 6}
```

Más: <https://docs.python.org/2/library/stdtypes.html#set-types-set-frozenset>



String

Los strings son secuencias de caracteres inmutables. Python tiene un amplio grupo de métodos para manipulación y creación de strings.

```
>>> text = "special \", \' inside"
```

```
>>> text[-1]
```

```
e
```

```
>>> print("%s" % "some text")
```

```
"some text"
```

```
>>> "{a}!={b}".format(a=2, b=1)
```

```
2!=10
```

```
>>> "44".isdigit()
```

```
True
```

Más: <https://docs.python.org/2/library/stdtypes.html#string-methods>




I/O

Métodos normales de lectura / escritura:

`raw_input()`: lee una línea de la entrada y la convierte a string, eliminando el salto de línea.
Cuando llega al EOF, tira una excepción.

`print()`: puede recibir múltiples parámetros, e imprime cada uno de ellos, con un espacio entre medio, agregando un salto de línea al final.

Referencia: <http://www.geeksforgeeks.org/python-input-methods-competitive-programming/>



I/O (fast)

Para una lectura / escritura más eficiente:

`sys.stdin.readline()`: utiliza un buffer para la lectura de datos.

2. `stdout.write()`: imprime el string que se le pase por parámetros. Es más rápido que `print()`, el cual por defecto es sólo un wrapper a `stdout.write()`.

Referencia: <http://www.geeksforgeeks.org/python-input-methods-competitive-programming/>