

Curso Python en 8 clases

Clase 2: Tipos de datos y estructuras de control

Autor: Sebastián Bassi



Versión: 1.1

Licencia: Creative Commons BY-NC-SA 2.5. ([ver texto completo](#))



Introducción

Una característica de Python es la cantidad y versatilidad de sus tipos de datos. Conocerlos en profundidad sirve para tener un buen criterio a la hora de elegir como modelar nuestros datos.

En líneas generales los tipos de datos se dividen en *primarios* y *derivados*:

- Primarios (o primitivos): No están basados en otro tipo de datos, como numericos (int, float, decimal, complex) y str (cadenas).
- Derivados: Agrupan a alguno de los anteriores, como listas, diccionarios, tuplas, etc. Se pueden subclasificar según distintos parámetros: Ordenados (o secuenciales) / Desordenados y Mutables / Inmutables.

Algunos ejemplos de datos primarios (enteros, flotantes, complejos y cadenas):

```
>>> type(5)
<type 'int'>
>>> type(5.0)
<type 'float'>
>>> type(5 + 5.0)
<type 'float'>
>>> 5 + 5.0
10.0
>>> type(2+3j)
<type 'complex'>
>>> (2+3j).real
2.0
>>> (2+3j).imag
3.0
>>> type('Hola!')
<type 'str'>
>>> 'hola' + ' mundo!'
'hola mundo!'
>>> 'hela' + 2
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    'hela' + 2
TypeError: cannot concatenate 'str' and 'int' objects
>>> 'hela' + str(2)
'hela2'
```

Este error se debe a que Python es un lenguaje de tipado dinámico pero fuerte.

El "problema" de los números flotantes

El resultado de las operaciones con números flotantes puede ser inesperado:

```
>>> 0.1 + 0.1 + 0.1 - 0.3
5.5511151231257827e-17
```

Una manera de evitar esto:

```
>>> round(0.1 + 0.1 + 0.1 - 0.3,1)
0.0
```

Alternativamente, para no perder precisión:

```
>>> from decimal import Decimal
>>> Decimal('0.1') + Decimal('0.1') + Decimal('0.1') - Decimal('0.3')
Decimal('0.0')
```

Mas información: <http://docs.python.org/library/decimal.html> y <http://floating-point-gui.de>

str (String o Cadenas)

Los string o cadenas son secuencias ordenadas de simbolos. El manejo de cadenas en Python es bastante intuitivo. Hay que tener en cuenta que las cadenas son un tipo de datos inmutables. Veamos algunas operaciones asociadas a las cadenas:

```
>>> 'Hola mundo!'
'Hola mundo!'
>>> a='Hola mundo!'
>>> len(a)
11
>>> a.lower()
'hola mundo!'
>>> a.count('o')
2
>>> a.find('H')
0
>>> a.find('mundo')
5
>>> a.find('e')
-1
>>> a.index(' ')
4
>>> a.index('e')
Traceback (most recent call last):
  File "<pyshell#52>", line 1, in <module>
    a.index('e')
ValueError: substring not found
>>> a.split(' ')
['Hola', 'mundo!']
```

Listas

Las listas, al igual que las cadenas, son datos ordenados. Una diferencia importante es que es un tipo de dato derivado, también llamado "contenedor". Una lista es una secuencia ordenada de datos.

Algunas operaciones elementales con listas:

```
>>> mi_lista = [1,2,3]
>>> mi_lista.append(5)
>>> mi_lista
[1, 2, 3, 5]
>>> mi_lista.pop()
5
>>> mi_lista
[1, 2, 3]
>>> mi_lista + [4]
[1, 2, 3, 4]
>>> mi_lista
[1, 2, 3]
>>> mi_lista = mi_lista + [4]
>>> mi_lista
[1, 2, 3, 4]
>>> mi_lista.extend([5,6])
>>> mi_lista
[1, 2, 3, 4, 5, 6]
>>> mi_lista[0]
1
>>> mi_lista[3]
4
>>> mi_lista[3:5]
[4, 5]
>>> mi_lista[-2]
5
```

Las listas comparten con los str la "notación de rebanadas" (*slice notation*) donde se usan índices que marcan las posiciones de inicio y fin de una porción. En lugar de indicar el número de elemento, se indica el espacio entre los elementos:

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
0   1   2   3   4   5   6
```

Esta lista se accede usando slice notation:

```
>>> mi_lista = "Python"
>>> mi_lista[0:2]
'Py'
>>> mi_lista[:2]
'Py'
>>> mi_lista[4:6]
'on'
>>> mi_lista[4:]
'on'
```

Mas operaciones relacionadas a listas:

```

>>> variada = ['boga', 'cornalito', 'tararira']
>>> variada[2]
'tararira'
>>> variada[2][2:8]
'rarira'
>>> variada[2][2:]
'rarira'
>>> variada.append('pulpo')
>>> variada
['boga', 'cornalito', 'tararira', 'pulpo']
>>> variada.remove('cornalito')
>>> variada
['boga', 'tararira', 'pulpo']
>>> variada.sort()
>>> variada
['boga', 'pulpo', 'tararira']
>>> variada.index('pulpo')
1
>>> variada.index('pulpa')
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    variada.index('pulpa')
ValueError: list.index(x): x not in list
>>> 'pulpo' in variada
True
>>> 'pulpa' in variada
False

```

List comprehension

Como en teoria de conjuntos, las listas puede ser descriptas por extensión o por definición. En este último caso damos la regla para crearla sin especificar todos los elementos de manera taxativa:

```

>>> vec = [2, 4, 6]
>>> [3*x for x in vec]
[6, 12, 18]
>>> [3*x for x in vec if x > 3]
[12, 18]
>>> [3*x for x in vec if x < 2]
[]
>>> [[x,x**2] for x in vec]
[[2, 4], [4, 16], [6, 36]]

```

Tuplas

Las tuplas son elementos ordenados como las listas, con la diferencia que son inmutables. Veamos algunos ejemplos:

```

>>> t1 = ('sgn1545',5,45)
>>> t1[0]
'sgn1545'
>>> 5 in t1
True
>>> t1.index(45)

```

```

2
>>> t1.count(45)
1
>>> t1.append(4)
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    t1.append(4)
AttributeError: 'tuple' object has no attribute 'append'
>>> t1.pop()
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    t1.pop()
AttributeError: 'tuple' object has no attribute 'pop'
>>> t1.remove(45)
Traceback (most recent call last):
  File "<pyshell#44>", line 1, in <module>
    t1.remove(45)
AttributeError: 'tuple' object has no attribute 'remove'

```

La decisión de usar una lista o un tupla a veces tiene mas que ver con la naturaleza de los datos que con las operaciones que querramos hacer con ellos. En principio las listas parecen mas útiles porque permiten ser modificadas a diferencia de las tuplas, por su parte las tuplas pueden ser índices de otros tipos de datos. Las listas deben ser usadas para datos del mismo tipo mientras que las tuplas se adecuan a datos de distinto tipo donde la posición indicaría el "campo".

Diccionarios

Es una estructura contenedora y sin orden. Se lo usa para almacenar datos indexados por claves (pares claves/valor). Definición y propiedades:

```

>>> en2es = {'blue':'azul','red':'rojo','black':'negro'}
>>> en2es['blue']
'azul'
>>> en2es['azul']
Traceback (most recent call last):
  File "<pyshell#47>", line 1, in <module>
    en2es['azul']
KeyError: 'azul'
>>> 'blue' in en2es #verifico que una clave exista en 1 dict.
True
>>> en2es.keys()
['blue', 'black', 'red']
>>> en2es.values()
['azul', 'negro', 'rojo']
>>> en2es.items()
[('blue', 'azul'), ('black', 'negro'), ('red', 'rojo')]
>>> en2es.get('green','N/D')
'N/D'
>>> es2en = {} #Diccionario vacio
>>> es2en['azul'] = 'blue' #Cargo un par clave-valor
>>> es2en
{'azul': 'blue'}

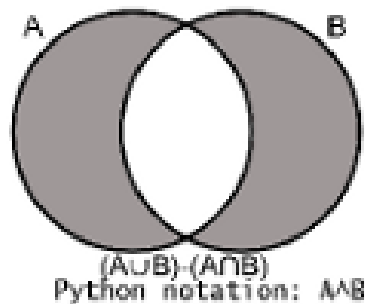
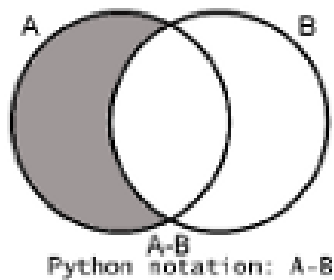
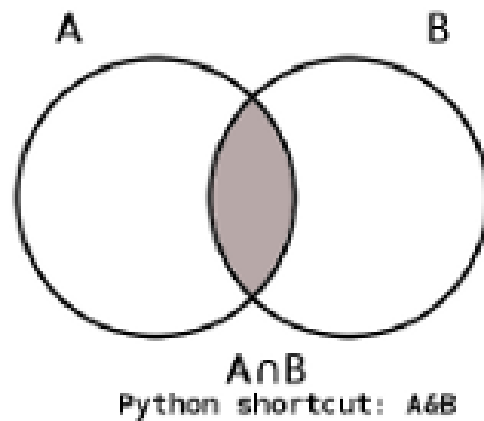
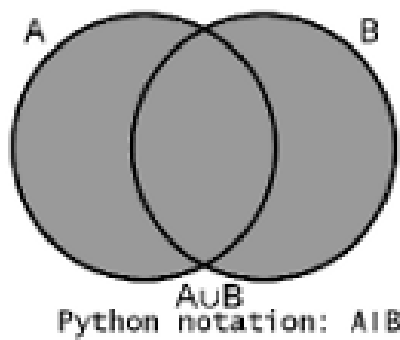
```

Sets (conjuntos)

Al igual que los diccionarios los sets son contenedores sin orden. La característica principal es que sus elementos son únicos y permiten operaciones asociadas a conjuntos. Uso:

```
>>> mi_set = set()
>>> mi_set.add('jose')
>>> mi_set.add('juan')
>>> mi_set.add('natalia')
>>> mi_set.add('viki')
>>> mi_set
set(['jose', 'juan', 'viki', 'natalia'])
>>> mi_set.pop()
'jose'
>>> mi_set
set(['juan', 'viki', 'natalia'])
>>> mi_set.add('jose')
>>> mi_set
set(['jose', 'juan', 'viki', 'natalia'])
>>> mi_set.add('jose')
>>> mi_set
set(['jose', 'juan', 'viki', 'natalia'])
>>> otro_set = set(['juan', 'karina', 'diana'])
>>> otro_set
set(['diana', 'juan', 'karina'])
>>> mi_set.intersection(otro_set)
set(['juan'])
>>> mi_set.union(otro_set)
set(['jose', 'viki', 'natalia', 'diana', 'juan', 'karina'])
>>> mi_set.difference(otro_set)
set(['jose', 'viki', 'natalia'])
```

Representaciones de algunas operaciones de conjuntos:



Referencias tipos de datos

Estructuras de datos:
http://en.wikibooks.org/wiki/Python_Programming/Sets
<http://docs.python.org/tutorial/datastructures.html>

Strings

- <http://docs.python.org/library/string.html>
- <http://www.network-theory.co.uk/docs/pytut/Strings.html>

Listas

- <http://effbot.org/zone/python-list.htm>

Diccionarios

- http://diveintopython.org/getting_to_know_python/dictionaries.html

Sets

- PEP-218 <http://www.python.org/dev/peps/pep-0218/>
- <http://www.devshed.com/c/a/Python/Python-Sets/>
- http://en.wikibooks.org/wiki/Python_Programming/Sets

Estructuras de control

Python tiene 3 estructuras de control. Una de decisión (*if*) y 2 de ciclos (*for* y *while*).

if

Modo general

```
if <expresion1>:
    <Instrucciones1>
elif <expresion2>:
    <Instrucciones2>
else:
    <Instrucciones3>
```

En caso que *<expresion1>* sea verdadera, se ejecuta *<Instrucciones1>*, si es falsa, se evalúa *<expresion2>*, que en caso de ser cierta, se ejecutan el bloque de código designado *<Instrucciones2>*. Puede haber tantos *elif* como sea necesario. En caso que ninguna expresión sea verdadera, el flujo del programa entra en el bloque del *else*, esto es en este esquema, *<Instrucciones3>*.

Ejemplo de uso

Si el objeto llamado *coord* es distinto al str 'N/A', llamar *year* al entero del primer elemento de la lista *coord* sin los 4 primeros elementos:

```
if coord != 'N/A':
    year = int(coord[0][-4:])
```

for

for es usado para "recorrer" un objeto iterable.

Modo general

```
for <var> in <iterable>:
    <instrucciones>
```

Ejemplo de uso

El siguiente código sirve para recorrer la lista [1,3,4], y en cada iteración *x* toma el valor de cada elemento de la lista.

```
for x in [1, 3, 4]:
    print x
```


¿Qué es un iterable?

Un iterable puede ser: Lista, string, tupla, diccionario, set o cualquier objeto que tenga el método `next()`.

while

Es para ejecutar un bloque de código repetidas veces, siempre y cuando *<expresion>* sea verdadera. Normalmente dentro del bloque de código hay alguna instrucción que cambia un valor para que esto ocurra y el ciclo termine. Otra alternativa para terminar el ciclo es usando **break** (ver mas adelante).

Modo general

```
while <expresion>:  
    <instrucciones>
```

Ejemplo de uso

Imprimir elementos de *mi_set* hasta que este objeto contenedor (el set) se consuma:

```
mi_set = set([1,2,3,4])  
while mi_set:  
    print mi_set.pop()
```

Break

Break se usa para salir del ciclo que lo contiene. No confundir con saltos incondicionales de otros lenguajes.