1. Sintaxis ba?sica de Python

May 5, 2014

1 Aritmética

```
In [1]: 2
```

```
Out[1]: 2
```

```
In [3]: 2 + 4
```

```
Out[3]: 6
```

```
In [5]: x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-5-401b30e3b8b5> in <module>()
----> 1 x


NameError: name 'x' is not defined
```

```
In [6]: 2 / 3
```

```
Out[6]: 0
```

```
In [7]: 2. / 3
```

```
Out[7]: 0.6666666666666666
```

```
In [9]: 1 + 2. / 3
```

```
Out[9]: 1.6666666666666665
```

```
In [10]: (1+2.) / 3
```

```
Out[10]: 1.0
```

2 Variables

Una variable es un espacio reservado en la memoria, que contiene una información dada. En Python, tenemos *nombres* que son *referencias* a ubicaciones en la memoria.

Puedo poner ecuaciones en LaTeX: $y = \int_{x=0}^{\infty} \exp[-x^2]dx$

```
In [11]: x = 3

In [12]: x

Out[12]: 3
```

Automáticamente reconoce el *tipo* de la variable (qué tipo de contenido contiene):

```
In [13]: type(x)

Out[13]: int
```

El concepto está bastante escondido en Python.
Puedo hacer operaciones con `x`:

```
In [14]: x + 3

Out[14]: 6

In [15]: x * 2.5

Out[15]: 7.5

In [16]: x / 100.3

Out[16]: 0.029910269192422734

In [18]: x ** 17  # potencias como en Fortran

Out[18]: 129140163
```

Mi ejemplo favorito:

```
In [19]: 2 ** 2

Out[19]: 4

In [20]: 2 ** 2 ** 2

Out[20]: 16

In [21]: 2 ** 2 ** 2 ** 2

Out[21]: 65536

In [24]: 2 ** 2 ** 2 ** 2 ** 2

Out[24]: 200352993040684646497907235156025575044782547556975141926501697371089405955631145308950613088
```

La 'L' al final quiere decir que es un entero "largo"

```
In [25]: _ * 2  # _ es el resultado anterior

Out[25]: 400705986081369292995814470312051150089565095113950283853003394742178811911262290617901226176
```

¿Cuántos dígitos tiene este número?

```
In [26]: x = 2 ** 2 ** 2 ** 2 ** 2

In [28]: s = str(x)
```

```
In [29]: s
```

```
Out[29]: '200352993040684646497907235156025575044782547556975141926501697371089405955631145308950613088
```

Número de caracteres en una cadena es su *longitud* (*length*):

```
In [31]: len(s)
```

```
Out[31]: 19729
```

```
In [32]: 2**10
```

```
Out[32]: 1024
```

```
In [33]: 3*6550
```

```
Out[33]: 19650
```

```
In [34]: x = 3
         y = 10

         x * y
```

```
Out[34]: 30
```

Las variables pueden cambiar de tipo:

```
In [36]: x = 3
         x
```

```
Out[36]: 3
```

```
In [37]: x = 3.5
         x
```

```
Out[37]: 3.5
```

```
In [38]: type(x)
```

```
Out[38]: float
```

NB: `float` es un número flotante de *doble* precisión en Python (8 bytes)

```
In [40]: x = "hola"
         x
```

```
Out[40]: 'hola'
```

```
In [41]: print(x)
```

```
hola
```

```
In [42]: type(x)
```

```
Out[42]: str
```

3 Listas

Una lista es un *arreglo* (de otro lenguaje) (más o menos).

Una lista es un *contenedor* que *contiene* (guarda) varios datos; se escribe con corchetes, `[` y `]`

```
In [45]: l = [1, 2, 3]
```

Para accesar los elementos:

```
In [48]: l[1]  # elemento número 1 (el segundo elemento)

Out[48]: 2
```

Como en C, siempre empezamos a contar en 0

```
In [49]: l[0]

Out[49]: 1

In [50]: l[2]

Out[50]: 3

In [51]: l[4]


---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)

<ipython-input-51-23ef5daf5560> in <module>()
----> 1 l[4]


IndexError: list index out of range


In [55]: l[-1]  # ultimo elemento

Out[55]: 3

In [56]: l

Out[56]: [1, 2, 3]
```

Si tengo un dato nuevo, ¿qué hago?

```
In [57]: l

Out[57]: [1, 2, 3]

In [58]: type(l)

Out[58]: list
```

Todo en Python es un *objeto* con propiedades

`l.<TAB>` me da (en IPython) una lista de las propiedades internas de `l`:

```
In [59]: l.append

Out[59]: <function append>
```

Para mandar llamar a la función, uso paréntesis:

```
In [60]: l.append(17)

In [61]: l

Out[61]: [1, 2, 3, 17]

In [62]: l[3]

Out[62]: 17

In [79]: l = [3, -7.6, "hola", [3, 4]]
         l

Out[79]: [3, -7.6, 'hola', [3, 4]]

In [80]: l.append([18, 3])
         l

Out[80]: [3, -7.6, 'hola', [3, 4], [18, 3]]

In [81]: l.append(l)
         l

Out[81]: [3, -7.6, 'hola', [3, 4], [18, 3], [...]]

In [82]: l[-1]

Out[82]: [3, -7.6, 'hola', [3, 4], [18, 3], [...]]

In [84]: l[1:5]  # omite el ultimo

Out[84]: [-7.6, 'hola', [3, 4], [18, 3]]

In [85]: l[1:1]

Out[85]: []
```

Agregar una copia de `l`:

```
In [86]: l.append(l[:])
         l

Out[86]: [3,
          -7.6,
          'hola',
          [3, 4],
          [18, 3],
          [...],
          [3, -7.6, 'hola', [3, 4], [18, 3], [...]]]
```

Una lista sólo contiene *referencias* (*punteros*), no los objetos mismos.

3.1 Notación científica:

```
In [63]: 1.5e9

Out[63]: 1500000000.0

In [64]: 1.5e-9

Out[64]: 1.5e-09
```

3.2 Números complejos:

```
In [66]: 1j  # j = i, la raiz cuadrada de -1

Out[66]: 1j

In [69]: x = 3 + 4j
         x

Out[69]: (3+4j)
```

Las propiedades de `x` las averiguo con
`x.<TAB>`

```
In [73]: x.conjugate

Out[73]: <function conjugate>

In [74]: x.conjugate()

Out[74]: (3-4j)

In [75]: x * x.conjugate()

Out[75]: (25+0j)
```

4 Listas dos-dimensionales

```
In [87]: M = [ [1, 2], [3, 4] ]
         M

Out[87]: [[1, 2], [3, 4]]

In [88]: print(M)

[[1, 2], [3, 4]]

In [89]: l = [1, 2, 3]
```

Uno pensaría que `l` fuera como un vector

```
In [90]: l + l

Out[90]: [1, 2, 3, 1, 2, 3]
```

No lo es! Ya que Python *no* se diseñó como lenguaje para cómputo científico.