

Sintaxis de Python

febrero de 2012

Parte del “Taller de Programación Científica: Simulaciones” – VERANO 2012
PCF-LIAP — IFT/CONFER, Universidad de Costa Rica.

Expositor: Guillermo André Oliva Mercado

Simbología:

△ Recordatorio de Principios de Informática. Puede ser útil para los que llevan tiempo sin programar, o no tienen mucha experiencia.

⊗ Comentario para los que han tenido más contacto con programación, o para los que saben programar en C o Java.

■ Ejercicio

1 Introducción a Python

 Python  interpretado  alto nivel  orientado a objetos programa de muestra: <pre>print 'Hola Mundo!'</pre>	Ficha técnica <i>Mantenido por:</i> Python Software Foundation www.python.org <i>Creado por:</i> Guido van Rossum, 1991 <i>Sistemas operativos</i> Multiplataforma (Unix, Linux, Windows, Mac OS, etc.)
---	---

- Python es un lenguaje de programación interpretado, de alto nivel y orientado a objetos¹, enfocado en la simplicidad de la sintaxis², creado por Guido van Rossum a principios de los años 90. El nombre de Python viene del grupo de cómicos ingleses *Monty Python*.
- △ *Lenguajes interpretados*. Un lenguaje interpretado es el que se ejecuta en la computadora mediante otro programa llamado intérprete, a diferencia de los lenguajes compilados, como C, que son convertidos directamente

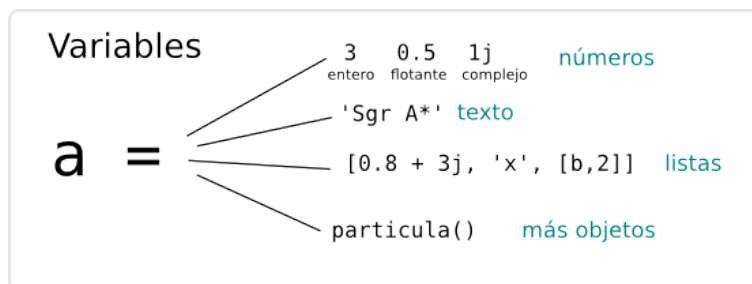
¹En esta sesión, la meta es que estos tres conceptos sean explicados y aplicados de forma muy natural.

²△ Sintaxis es como en el español, la forma en la que se programa (las palabras clave, puntuación, etc.). O sea, qué interpreta como qué la computadora.

a lenguaje de máquina para ejecutarse. Que Python sea de alto nivel significa que hay muchas operaciones básicas que no deben hacerse, como el manejo de la memoria (reservar y liberar memoria), etc., que se hacen en lenguajes como C.

- \otimes *Cpython*. El intérprete de Python que vamos a usar (porque existen más implementaciones) está escrito en C, y se llama CPython. Como en Java, el intérprete de Python está hecho para precompilar el código en *bytecode* antes de ejecutarlo por primera vez.
- Python es *multiplataforma*. Eso significa que el intérprete de Python está disponible para Unix, Linux, Solaris, Windows, DOS, Mac OS, etc. Las distribuciones de Linux corrientes traen ya Python por defecto. Varios sistemas móviles también usan Python en sus aplicaciones.
- **¿Por qué Python?** Porque es orientado a objetos, que es una forma natural de modelar el mundo real en el código (lo que hacen los físicos). La sintaxis es tan sencilla que nos permite concentrarnos en programar. El alto nivel de Python, las herramientas disponibles (librerías, etc.). Su rendimiento no es el óptimo, pero es aceptable para la mayoría de cálculos científicos. Bastan unas líneas de código para tener algo realmente sorprendente.

2 Variables (objetos primitivos)



```
$ python
```

- Python podría fácilmente usarse como una calculadora. Pruebe hacer varias operaciones. El intérprete automáticamente muestra los resultados. Observe que Python tiene predefinido el operador de elevación a potencias (**), y números complejos (2+5j).
- Asignación de variables:

```
a=2 # un entero
b = 0.5 # un decimal.
c = a+b # el resultado de una operación
```

```

palabras = 'hola mundo' # un texto. También con comillas dobles.
lista2_de_cosas=[2,"taller",0.1+3j] # una lista (de objetos)
tupla = (1,0,5) # una lista que no cambia nunca ("inmutable")
algo = [lista2_de_cosas,tupla, a, palabras] # lista que contiene
más listas y otras cosas

```

- Ejemplo. *Calculadora científica con el intérprete*

```

#Ahora un ejemplo de cálculos sencillos con python desde el intérprete
from math import cos,exp # hace que "cos" y "exp" estén disponibles
exp(2) # e^2. Es la función exponencial
amplitud=1.2
omega = 3
t=0.5
x = amplitud*cos(omega*t) # función coseno
a = 4
b=3
c = (a**2+b**2)**0.5
# Otras librerías muy útiles para cálculos rápidos son numpy y scipy.
www.scipy.org

```

- Los comentarios de una línea en Python se hacen con `#`. \triangle Los comentarios son parte del código ignorada por la computadora.
- \otimes *Nota para los que programan en C y Java*: Python es un lenguaje de tipado dinámico, o sea que como se observa, no hay que declarar el tipo de variable antes de crearla o modificarla. Y uno puede fácilmente cambiar el tipo de variable con sólo asignarle otra cosa a un símbolo. Es que para Python, "a", "b", "palabras", "tupla", son *símbolos* que tienen asociado un objeto en memoria; y este objeto puede ser de cualquier tipo. Recuerde tener cuidado con el tipo de objeto de una variable cuando esté trabajando con ella. Python es orientado a objetos, como Java y C++. Bash y C, en cambio, son lenguajes *imperativos*, es decir, que van orden por orden. Si usted está acostumbrado a programar en C, le sugiero que empiece a dejar a un lado la expresión "tipo de dato". En Python todo es un objeto. Hasta las clases son objetos. También las funciones son objetos.
- \blacksquare (Ejercicio): Un mismo operador tiene efectos diferentes con objetos diferentes. Ejecute lo siguiente y observe el resultado.

```

2+6.2
'fisica compu' + 'tacional'
2*'pa'
['a','b',2,4] + [6,'z']

```

2.1 Manipulación básica de listas

Las listas contienen objetos (no tienen por qué ser del mismo tipo)

- Acceder a los elementos de una lista

```
>>>a=[1,3,5,7,'impares']
>>>a[0]
1
>>>a[4]
'impares'
>>>a[-1] #Python es de tan alto nivel que reconoce
esto como el último elemento de la lista
'impares'
#otro ejemplo
>>>b = [1,2]
>>>c = [b,'a',0]
>>>c[0]
[1,2]
>>>c[0][0]
1
>>>c[0][1]
2
# una forma de construir matrices.
```

- ■ *Objetos mutables e inmutables.* Tome varios elementos de una lista: `a[0:1]`. Las listas son objetos *mutables*, es decir, que si ud hace `b=a`, y luego modifica `b`, también está modificando `a`. Para no modificar `a`, debe copiar la lista, o sea, `b = list(a)` o bien `b=a[:]`. Ahora sí, si ud modifica `b`, ya no se modifica `a`. Al contrario, los números, texto y tuplas son *inmutables*, o sea que si `x=2`, si yo hago `y=x` y modifico `y`, *no* se modifica `x`. Más adelante volveremos a usar estos conceptos.

- Añadir a la lista:

```
>>>x = 'nuevo elemento'
>>>a.append(x)
[1,3,5,7,'impares','nuevo elemento']
>>>a.append(5)
[1,3,5,7,'impares','nuevo elemento',5]
# Hay más de esto en Python para todos y en la
documentación oficial de Python
```

- Quitar elementos de la lista:

```
>>>a.remove(5) # quita el primer elemento
que tenga valor 5
[1,3,7,'impares','nuevo elemento',5]
# Más en la documentación de Python
```

- Unir dos listas. Con el operador +.

3 Scripts

Básicamente, un script es un archivo de texto con una lista de órdenes para el intérprete de Python.

```
#!/usr/bin/python
print "hola mundo"
```

El contenido de un archivo "holamundo.py" que imprima "hola mundo" en la pantalla. Se ejecuta con

```
$ python holamundo.py
```

4 Objetos y librerías

En el ejemplo de la calculadora utilizamos la librería math. La sintaxis es:

```
from <librería> import <función>
from math import cos
from numpy import array
from scipy import integrate
```

si deseamos importar todas las funciones de una librería,

```
from numpy import *
```

La librería *visual* de Python (vpython.org) es la que utilizaremos en estos talleres para hacer cálculos y visualizar las simulaciones al mismo tiempo.

```
>>>from visual import *
```

4.1 Objetos

- Los objetos son formas de modelar el mundo real en el código. Tienen propiedades, y también pueden hacer cosas.
- ¿Qué distingue a un objeto de otro? ¿A un carro de otro carro? ¿A una silla de una mesa? ¿A un carro de una mesa?

```
>>>a=sphere()
>>>a.color=color.red
>>>a.pos = vector(1,1,1)
```

- El color y la posición son atributos de la esfera.

- El color es un objeto también. El vector de posición es otro objeto también. Y las coordenadas del vector de posición (1,1,1) son los atributos del objeto vector.
- En el código, en la vida real, en la ejecución del programa (memoria, procesador), una esfera es una esfera, un vector es un vector y un color es un color. No son números, ni cadenas de texto, ni listas, ni nada por el estilo. Ese es el paradigma de orientación a objetos.
- Una *clase* es un *manual de construcción de objetos*. (En este caso, crearemos un manual de construcción “vacío”. Si desean saber más sobre clases y objetos, consulten [2], o ¡pidan más talleres!)

```
class perro:
    pass
```

- Cuando hago

```
neron = perro()
```

estoy creando a un perro en la memoria del sistema, y puedo interactuar con él. Con la misma clase, puedo crear cuantos perros necesite:

```
neron=perro()
fido=perro()
chata=perro()
```

e interactuar con ellos asignándoles atributos:

```
neron.pulgas=7
```

- *Un poco de mecánica clásica*. Ahora, vamos a programar una partícula.

```
from visual import *
class partícula:
    pass
bola = partícula()
bola.mass = 0.4
# Gracias a visual tengo definido qué es un vector tridimensional
bola.pos = vector(1,2,1)
bola.vel = vector(10,0,2)
bola.ke = 0.5*bola.mass*mag2(bola.vel) # mag2 es una función que
tengo disponible gracias a la librería visual. Calcula la magnitud
al cuadrado de un vector.
```

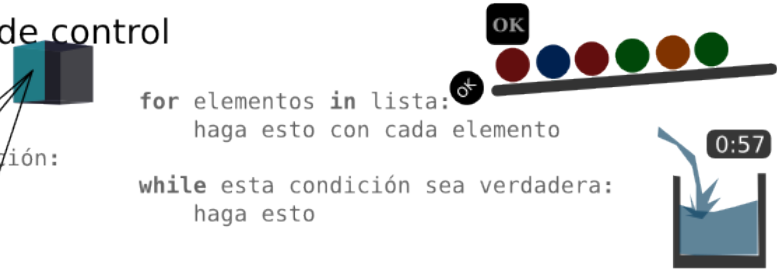
5 Estructuras de control

Estructuras de control

```
if condición:
    haga esto
elif otra condición:
    lo otro
else:
    lo último
```

```
for elementos in lista:
    haga esto con cada elemento
```

```
while esta condición sea verdadera:
    haga esto
```



5.1 if / else

Continuando con el ejemplo de la partícula...

```
if bola.ke > 20:
    print "se quebró el vidrio"
elif bola.ke == 20:
    print "suficiente energía para quebrar el vidrio"
else:
    print "la bola rebotó y no quebró el vidrio"
```

Es indispensable *indentar*³ el texto para separar los bloques dentro del *if*, pues es la única manera en la que el intérprete de Python “sabe” cuándo termina el *if*. Además, el código se ve mucho mejor, y es más legible (una de las metas de Python). El código puede indentarse usando espacios o tabulaciones, pero no ambos al mismo tiempo. *elif* significa *else if* («si no,...»); solo se ejecuta si la primera condición es falsa (puede reemplazarse por otro *if* anidado en un *else*).

△ *Los dos signos igual*. En programación, hay dos tipos de signos “igual”. Está el = que usamos en la sección 2, que sirve para *asignar* un valor. Este igual es para *comparar* dos valores, y por eso es diferente, y se escribe ==. O sea, es un operador que sirve para ver si una ecuación se cumple o no.

5.2 for / while

- △ Cuando necesitamos que se repita una instrucción muchas veces, usamos *bucles*. En Python existen dos tipos: el *for* y el *while*, y en teoría son equivalentes. A veces es más cómodo usar uno que otro en determinadas circunstancias.

- *For*:

```
for x in ["Mercurio", "Venus", "Tierra", "Marte"]:  
    print x + " es un planeta interior del Sistema Solar"
```

³La palabra correcta en español debería ser *sangrar* el texto, pero se usa muy frecuentemente en informática el anglicismo *indentación*.

- Python es un lenguaje de alto nivel, por lo que al usar

```
for <elemento> in <lista>
```

automáticamente recorre cada elemento de la lista.

- \otimes *Iteración en Python.* En Python se itera sobre los objetos, no sobre índices. Para los que programan en C o Java, tal vez les lleve un tiempo acostumbrarse a esto. En Python también uno puede iterar sobre índices, pero casi siempre eso es inútil.
- *Un ejemplo un poco más complicado.* Tal vez el ejemplo anterior no valía tanto la pena automatizarlo, pues eran 4 elementos en una lista. ¿Qué tal si son 100 iteraciones o más? Generemos una lista con 100 números complejos con un cierto patrón:

```
numeros = [] # creamos una lista vacía para que contenga
los números que vamos a generar
```

```
for i in range(100):
    numeros.append(i*3+(i/2)*1j)
```

La función `range(n)` viene por defecto en Python, y genera números enteros desde 0 hasta n . Hay otra función (`arange`) del módulo `numpy` que es muy útil también para este fin. Con este método sería muy fácil evaluar, por ejemplo, los coeficientes de una serie de Fourier hasta una n -ésima potencia.

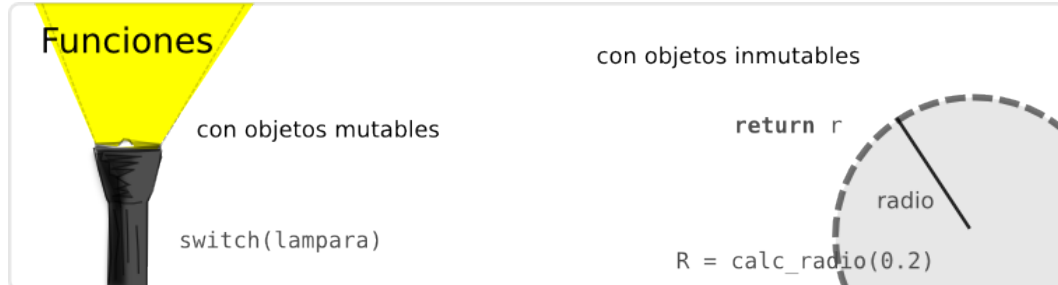
- *While.* Como ejemplo, llenemos un vaso con líquido, y hagamos que el programa pare cuando el líquido ya se esté derramando del vaso.

```
class vaso:
    pass
vaso.nivel = 0
vaso.h = 15
dy = 0.3
while vaso.nivel <= vaso.h:
    vaso.nivel = vaso.nivel + dy # [*]
print "vaso lleno"
```

* \triangle Recordar: esto no es una ecuación (sería incorrecta). Significa: “agarre el antiguo valor de `vaso.nivel` y súmele `dy`, y póngaselo al nuevo valor de `vaso.nivel`”.

- Este último ejemplo ya huele a simulación...
- \otimes En Python no existe el “do-while”.

6 Funciones



- En Python, las funciones se definen y usan así:

```
def f(x):  
    return 5*x+2  
y = f(0.3)
```

muy intuitivo.

- Cálculo del radio de una esfera con una densidad definida:

```
def calcular_radio(masa):  
    densidad = 2  
    volumen = masa/densidad  
    radio = (3*volumen/(4*3.14))**(1/3)  
    return radio  
R = 0.2  
print R  
R = calcular_radio(2.5)  
print R
```

Todo como se espera de cualquier otro lenguaje de programación.

- *La lámpara que se enciende y apaga.* Ahora, programemos una función que apague y encienda una lámpara. Si está encendida, que se apague y viceversa:

```
def switch(lampara):  
    if lampara.bombillo == "Encendido":  
        lampara.bombillo = "Apagado"  
    if lampara.bombillo == "Apagado":  
        lampara.bombillo = "Encendido"  
class lampara:  
    pass  
milampara = lampara()  
milampara.bombillo = "Apagado"  
switch(milampara)  
print milampara.bombillo
```

```

switch(milampara)
print milampara.bombillo
switch(milampara)
print milampara.bombillo

```

¿hay algo diferente aquí? Pues que la función no tiene un *return*. Los objetos en Python pueden ser de dos tipos: *mutables* e *inmutables*. Los números, cadenas de texto y tuplas son objetos inmutables. Se comportan como el ejemplo anterior a este, y requieren de un *return*. En cambio, la lámpara es un objeto *mutable*. Los objetos creados con *class*, y las listas, son objetos *mutables* en Python, y como en este caso, no fue necesario un *return*. En el primer ejercicio de la sección 2.1 se explica mejor la diferencia entre mutable e inmutable. Observe también que las maneras de llamar a la función son diferentes:

```

R = calcular_radio(2.5) # cambiar el radio...
switch(milampara) # ...cambiar el estado de la lámpara

```

- No hay nada de malo en poner un *return* en la función *switch*. Solo que para lo que la estamos usando, no es necesario. Uno no crea una lámpara nueva cada vez que la apaga y la enciende, sino que nada más cambia el estado a la misma lámpara.
- ⊗ Los objetos mutables se comportan como los punteros de C.

7 Comentarios finales

- Hay muchísimo más por aprender de Python. Lo que vimos es apenas lo indispensable para hacer una simulación. ¡Pidan más talleres!
- Los objetos pueden tener funciones definidas adentro. Estas funciones definidas se llaman *métodos* de un objeto. El ejemplo sería arrancar un carro. Podríamos definir una función *arrancar* dentro de un carro para que cambie sus atributos y de verdad arranque el motor... Hemos estado usando métodos, sin saber que eso son:

```

lista.append(elemento) # método del objeto 'lista' para
añadir un nuevo elemento
lista.remove(elemento)

```

además, podríamos haber programado la función *switch* de forma que al llamarla apagáramos o encendiéramos la lámpara:

```

milampara.switch()

```

- Insisto en que revise la página web de Scipy y Numpy: <http://www.scipy.org>. Son librerías extremadamente útiles para cálculos científicos. Podríamos hacer conferencias sobre ellas.

References

- [1] González Duque, Raúl. *Python para todos*. Descargable en formato PDF desde <http://mundogeek.net/tutorial-python/>
- [2] Documentación de Python. Puede leerse en línea desde <http://docs.python.org/>

Licencia

© 2011 Andrée Oliva – PCF-LIAP.

Este documento está disponible bajo una licencia Creative Commons Attribution-ShareAlike (atribución-compartir igual) 3.0 Unported. Para leer una copia de la licencia, visite <http://creativecommons.org/licenses/by-sa/3.0/>