Cookbook Python

September 8, 2014

En este documento encontrarán paso a paso lo que realizaremos en clase, durante el documento habrán ejercicios los cuales no se deberán de entregar, pero se recomienda al estudiante hacerlos para reforzar lo visto en clase.

1 Empezando

Para abrir Python escribimos en la terminal `python` lo que arrojará lo siguiente:

```
python
Python 2.7.4 (default, Sep 26 2013, 03:20:26)
[GCC 4.7.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Las 3 flechas `>>>` indican que estamos dentro de python y podemos empezar a escribir. Por ejemplo:

```
>>> 1
>>> 1
```

Es muy para utíl escribir aritmetica e.g:

```
>>> 3*5
>>> 15
```

Si quiero dividir dos numeros tengo que especificar que son decimales de lo contrario el resultado sera aproximado al entero mas cercano. Para especificar que un número es un decimal utilizamos el punto así:

```
>>> 40/25.0
>>> 1.6
```

otra operación importante es el operador modulo (%) que se define como el sobrante de la division del numero a entre b veces.

```
>>> 10%3
>>> 1
```

ya que 3 puede estar 3 veces en 10 y hay un sobrante de 1.

Con el fin de escribir programas lo mejor será abrir un editor de texto:

```
emacs holamundo.py
```

Acá es donde escribiremos todos nuestros programas.

Nuestro primer programa será imprimir `Hola mundo`, para eso escribimos lo siguiente:

```
print 'hola mundo'
```

Las comillas sencillas o dobles se pueden usar para indicar que imprimimos strings (cadenas de caracteres), Para ejecutar el programa escribimos en la terminal:

```
   python holamundo.py
Hola mundo
```

2 Variables:

Que tipo de variables existen en `python`?, Para esto abramos un documento `emacs variables.py &` y escribamos:

```
a = 1
b = 1.0
c = "hola"

print a, b, c
print type(a), type(b), type(c)
```

Esto arrojaría el siguiente resultado:

```
1 1.0 hola
<type 'int'> <type 'float'> <type 'str'>
```

Las variables tipo `int` guardan números enteros, mientras que las tipo `float` guardan flotantes que son numeros relaes y las tipo `str` guardan strings (Cadenas de caracteres).

3 Condicionales (if y while)

Podemos controlar los programas poniendo condiciones en el. Por ejemplo si queremos imprimir algo solo si alguna condicion cumple, por ejemplo si $a == b$, podemos utilizar el codicional `if`.

```
a = 2.0
b = 3.0
if (a==b):
   print a, "es igual que", b
elif (a>b):
   print a, "es mayor que ". b
else:
   print b, "es mayor que", a
```

En este caso el resultado de este programa será:

```
   3.0 es mayor que 2.0
```

La sintaxis de `if` es: entre parentesis la condición (En programación == es igual, >= es mayor o igual, <= es menor o igual) seguido por dos puntos (:) y luego indexado (espaciado) lo que se quiera que vaya dentro de la condicion. `elif` funciona como la contraccion de `else if`.

Tambien podemos poner condiciones usando `while`

```
i = 0
while (i<10):
   print i
   i += 1
```

Esto va a producir la siguiente cuenta regresiva:

```
0
1
2
3
4
5
6
7
8
9
```

Ejercicio: Escriba un programa en python que pida al usuario entrar un número entero y si este es impar que pare de pedir número, pero si es par que siga pidiendo números.

Ayuda: Para recibir número del usuario se puede usar `raw_input("Escriba un n\'umero: " )`

4 Iteración

Otra forma de controlar el programa es mediante iteraciones esto lo hacemos mediante un `for`:

```
for i in range(5):
  print "hola"
```

Lo que arrojara lo siguiente:

```
hola
hola
hola
hola
hola
```

5 Listas

En muchos casos es utĺ, crear listas con diferentes tipos de variables por ejemplo:

```
Lista = [1.0, 3, "hola"]
```

Si queremos imprimir toda la lista hacemos:
`print Lista`

Podemos llamar a un elemento en partícular de la lista solo conociendo su posición, En python las posiciones empiezan desde cero, es decir que en este caso el primer elemento de la lista 1.0 lo podemos llamar así:

`print Lista[0]`
Y así los siguientes elementos:

```
Lista[1]
Lista[2]
```

imprimiran respectivamente `3` y `'hola'`

Si queremos imprimir los elementos 1 y 2 podemos usar:

```
print Lista[0:2]
```

Esta notación significa imprima todos los elementos desde el [0] hasta antes del elemento [2]

Si queremos agregar mas elementos a la lista usamos `append` así:

`Lista.append(10)` Esto agregara un elemento nuevo al final de la lista.

Para agregar elementos en cualquier posición utilizamos `insert`.

`Lista.insert(2, "avion")`, donde el primer argumento entre parentesis indica la posición en la cual queremos insertar el elemento, minetras que el segundo es lo que queremos insertar.

Si queremos borrar un elemento de la lista usamos `pop`:

`Lista.pop(1)` done el argumento entre parentesis es la **posición** del elemento que quiero eliminar.

Mientras que en `Lista.remove(2)` el argumento es el **elemento** que queremos eliminar.

Si queremos ordenar la lista en orden acendente podemos usar `sort`:

`Lista.sort()`

Finalmente si queremos saber la longitud de la lista podemos usar `len(Lista)`
Hacer una lista con todos los números impares del 1 al 100 Explicar y dar ejemplos del uso de extend, index, reverse y count

6 NumPy :

NumPy es la la librería númerica de Python, dentro de esta encontraremos funciones trigonometricas, funciones especiales, alegra lineal (numpy.linalg) y mucho más. .

Para usar la librería de NumPy hay que llamarla al inicio de nuestro programa, para esto la importamos usando `import`

```
import numpy as np
```

Ahora cualquier funcion que usemos de NumPy debe de ir con el alias `np.` antes del nombre función.

```
>>>np.cos(np.pi)
-1.0
```

Ejercicio: Investigar como funciona la libreria de algebra lineal de NumPy, Dar ejemplos de como se hace la multiplicacion matricial, suma/resta de matrices, como sacar determinantes y autovalores.

7 Arreglos

A diferencia de las listas los arreglos nos permiten hacer operaciones matematicas. Estos hacen parte de NumPy, los arregos se definen así:

`x = np.array([])`, en este caso hemos creado un arreglo vacio. Si queremos crear un arreglo de zeros podemos usar:

`np.zeros(5)` Lo que arrojará:

```
array([ 0.,  0.,  0.,  0.,  0.])
```

O si quiero hacer un arreglo de unos:

```
ones(5)

array([ 1.,  1.,  1.,  1.,  1.])
```

Si queremos hacer una matriz de 3×5 podemos usar:

```
zeros([3, 5])

array([[ 0.,  0.,  0.,  0.,  0.],[ 0.,  0.,  0.,  0.,  0.],[ 0.,  0.,  0.,  0.,  0.]])
```

Si queremos borrar un elemento de un arreglo `x` usamos la función delete.

`np.delete(x, 10)` Donde el primer argumento es el arreglo y el segundo la **posición** que quiero eleminar. De manera similar funciona `append`

Ejercicio: Hacer dos arreglos con numeros diferentes usando `if` y hacer operaciones matematicas entre ellos.

Si quieres aprender mas sobre arreglos puedes ir a:
`http://docs.scipy.org/doc/numpy/reference/arrays.html`

8 Números aleatorios

Para genera número aleatorios usamos el función de NumPy `np.random` dentro de esta función hay varias funciones que generan numeros aleatorios segun varias distribuciones. Si queremos números aleatorios entre $[0, 1)$ podemos usar `random`:

```
   np.random.random(5)
array([ 0.85099041,  0.63817084,  0.79495685,  0.22457286,  0.68746815])
```

Ejercicio: Hacer una matriz de 3xn donde n sea un numero cualquiera, con números aleatorios entre -1 y 1

Ejercicio: Ilustrar 5 ejemplos mas de funciones generadoras de números aleaotrios dentro de np.random y explicar como se usan.

Ejercicio: Ir a `codeacademy.com` y realizar los modulos de "Condicionales y Control de Flujo" y "Listas y Diccionarios".

9 Leyendo y escribiendo archivos:

9.1 Leyendo archivos:

Dentro de `numpy` esta la función `loadtxt` esta nos permite leer archivos de datos.

```
data=np.loadtxt("data.txt")
```

En `https://github.com/jngaravitoc/HerramientasComputacionales/tree/master/Lectures/5.Python-2/data` esta el archivo `data.ascii` leerlo y responder las siguientes preguntas:

1. Que tipo de variables es el archivo `data.txt` cuando es llamado por `loadtxt`?
2. Con lo visto de arreglos, seleccionar las coumnas de los datos.
3. Sacar la desviacion standard y promedio de las primeras 3 columnas

9.2 index:

Imaginemos que tenemos dos columnas, una con edades y la otra con alturas y queremos relacionar datos de edad y altura. Como selecionaria las edades que corresponen a alturas mayores a $1.70cm$?

10 Funciones:

Las funciones nos permiten estructurar los programas y organizarlos. Una funcion realize una parte del programa y se ejecuta cuando la llamemos.

Definiendo la función:

```
def square(x): # entre el parentesis van los argumentos que necesitan la funci\'on.
   return x**2
```

`def` es el comando que indica al programa que vamos a definir una funcion.
`square` es el nombre que le di a esta funcion en especifico, pero puedo utilizar el nombre que yo quiera.
`return` retorna el resultado de la función.

Llamando a mi función:
`square(10)` que arrojará como resultado `100`

Hacer una función que haga los siguiente cambios de cordenadas:

1. **De Millas a Km**
2. **De Pulgadas a cm**
3. **De Litros a cm^3**

11 Recurrencia:

Cuando queremos hacer un programa que haga operaciones recurrentes, podemos llamar una funcion dentro de la misma función. Ejemplo:

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```