Introduccion a Python

Edgar Acuna
edgar.acuna@upr.edu

website:academic.uprm.edu/eacuna

Mayo del 2018

Compilacion e interpretacion

- Muchos lenguajes requieren compilacion (traduccion) del programa a una forma que la computadora entienda.

Python (al igual que R) es directamente *interpretado* en instrucciones de computadora.

4 versiones principales de Python

- “Python” or “CPython” is written in C/C++
  - Version 2.7.14. Septiembre 16, 2017
  - Version 3.6.4 Diciembre, 17 2017
- “Jython” esta escrita en Java. La ultima version salio en Mayo 2015.
- “IronPython” esta escrita en C# para el .Net entorno (version 2.7.7 salio en diciembre 2016).

Instalando Python en Windows

1. Descargar de Aliciaconda (www.continuum.io) ambas versions de python 2.7 y 3.6.

2- Instalar python2 siguiendo las instrucciones disponibles en https://docs.continuum.io/Aliciaconda/install/windows Usar su equivalente en MAC y Linux.

3- Dar el comando jupyter notebook para iniciarlo usando el kernel de python 2

3- Para Aliciadir el kernel python 3 al jupyter notebook, ejecutar los siguientes comandos:

conda create –n ipykernel_py3 python=3 ipykernel

activate ipykernel

python –m ipykernel install --user

Ambientes de Desarrollo (IDE)

1. Emacs (Linux)
2. Vim(Linux)
3. Idle (Windows)
4. NotePad++ (Windows)
5. Spyder
6. Jupyter Notebook (Windows, Linux, MacOS)

Python Interactive Shell

```
% python
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 204, 22:15:05)
[MSC v.1600 32 bit<intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Se puede correr cosas directamente en una session de Python

```
>>> 2+3*4
14
>>> name = “Camila"
>>> name
‘Camila'
>>> print ("Hola", name)
Hola Camila

>>>exit()
```

Ipython= Un Python Interactive Shell mejorado

```
(ipykernel_py2) C:\Users\edgar2017>ipython
Python 2.7.13 |Continuum Alicialytics, Inc.| (default, May 11 2017, 13:17:26)
    [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: 3+5
Out[1]: 8

In [2]: exit()

(ipykernel_py2) C:\Users\edgar2017>
```

Spyder

Spyder (Python 2.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\edgar2017

Editor

temp.py

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
```

Help

Source Console Object

Spyder - the Scientific PYthon Development EnviRonment

Spyder is an Integrated Development Environment (IDE) for scientific computing using the Python programming language. It comes with an Editor to write code, a Console to evaluate it and see its results at any time, a Variable Explorer to see what variables have been defined during evaluation, and several other facilities to help you to effectively develop the programs you need as a scientist.

This tutorial is authored by Hans Fangohr from the University of Southampton (UK) (see historical note for more detail).

Outline

First steps with Spyder

Recommended first steps for Python beginners

Variable explorer File explorer Help

IPython console

Console 1/A

```
bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:

In [1]:

In [1]:

In [1]: 3+5
Out[1]: 8

In [2]:
```

Python console History log IPython console

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 1 Column: 1 Memory: 84 %

Jupyter Notebook

Expresiones

- **Expresion**: Es el valor de un dato o un conjunto de operaciones para calcular un valor.

  Ejemplos: 1 + 4 * 3

  13

- Operadores aritmeticos:
  - + - * / suma, resta, multiplicacion, division
  - % module (o residuo)
  - ** exponenciacion

Operadores logicos

- Muchas expresiones logicas usan operadores relacionales:

Operador	Significado	Ejemplo	Resultado
==	Igual	`1 + 1 == 2`	`True`
!=	No es igual	`3.2 != 2.5`	`True`
<	Menor que	`10 < 5`	`False`
>	Mayor que	`10 > 5`	`True`
<=	Menor o igual a	`126 <= 100`	`False`
>=	Mayor o igual a	`5.0 >= 5.0`	`True`

Operador	Ejemplo	Resultado
`and`	`9 != 6 and 2 < 3`	`True`
`or`	`2 == 3 or -1 < 5`	`True`
`not`	`not 7 > 0`	`False`

print

- `print` : muestra un mensaje de texto o el valor de una expresion en la pantalla.

- Syntax:

  `print` ***"Message"***

  `print` ***Expression***

  Nota: En Python 3.4 se usa parentesis.

  `print (`***Item1 , Item2 , ... , ItemN)***

  - Imprime varios mensajes y/o expresiones en una misma linea.

input

- `input` :Lee un numero entrado por el usuario.
  - Se puede asignar (almacenar) el resultado de `input` a una variable.
  - Ejemplo(py2)

```
age = input("How old are you? ")
print "Your age is", age
print "You have", 65 - age, "years
until retirement"
```

Output:

```
How old are you? 53
Your age is 53
You have 12 years until retirement
```

Strings

- **string**: Una secuencia de caracteres textuales en un programa.
  - Strings empiezan y terminan en un character `"` o apostrophe `'` .
  - Ejemplos:

```
"hello"
"This is a string"
"This, too, is a string.   It can be very long!"
```

- Un string no puede escribirse en varias lineas ni contener un caracter `"` .

```
"This is not
a legal String."
"This is not a "legal" String either."
```

- Un string puede representar characteres precediendoles con un backslash.
  - `\t` tab character
  - `\n` new line character
  - `\"` quotation mark character
  - `\\` backslash character
  - Example: `"Hello\tthere\nHow are you?"`

Propiedades de Strings

- `len(`***string***`)` - numero de caracteres en un string (incluyendo espacios en blanco)
- `str.lower(`***string***`)` - lowercase version of a string
- `str.upper(`***string***`)` - uppercase version of a string

- Ejemplo:

```
name = "Edgar Acuna Fernandez"
length = len(name)
big_name = str.upper(name)
print big_name, "tiene", length,
"caracteres"
```

Listas

Es un objeto que puede contener distintos tipos de datos:

```
A=[0]
B=[2.3, 4.5]
C=[5, "Hello", "there", 9.8]
print type(C)
<type 'list'>
D=[]
```

Usar len() para obtener la longitud de una lista

```
>>> names = [“Alicia", “Elena", “Julia"]
>>> len(names)
3
```

Listas (cont)

#Aliciade un elemento al final de la lista

B.append(“Alicia”)

B

[2.3, 4.5, ‘Alicia’]

#Elimina el element 4.5 de la lista

B.remove(4.5)

[2.3, ‘Alicia’]

#Elimina el ultimo elemento de la lista

C.pop()

[5, "Hello", "there"]

Una tuple es una estructura de datos similar a lista pero cuyos elementos no pueden ser modificados.

Uso de [ ] para indexar items en una lista

```
>>> names[0]
'Alicia'
>>> names[1]
'Camila'
>>> names[2]
'Julia'
>>> names[3]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> names[-1]
'Julia'
>>> names[-2]
'Camila'
>>> names[-3]
'Alicia'
```

[0] es el primer item.
[1] es el segundo item
...

Valor Out of range da un exception

Valores negativos van hacia atras comenzando desde el ultimo elemento

Range()

- La funcion “range” crea una lista de numeros enteros en un rango especificado
- range([start], [stop], [step]) -> lista de enteros
- El valor de step especifica el incremento o decremento de la secuencia.

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 2)
[0, 2, 4, 6, 8]
```

Notar que ultimo elemento generado por range es n-1.

La funcion range funciona en forma distinta en la version 3 de Python

if

- **`Enunciado if`** : Ejecuta un grupo de enunciados solo si una condicion dada es cierta. De lo contrario los encunciados son omitidos.

  - Syntax:

    `if` ***condicion*** **:**
    enunciados

- Ejemplo:

```
gpa = 3.4
if gpa > 2.0:
    print "su solicitud es aceptada."
```

if/else

- **`Enunciado if/else.`** Ejecuta un bloque de enunciados si una cierta condicion se cumple, y un segundo bloque de enunciados si la condicion no se cumple .
  - Syntax:

```
if condition:
    statements
else:
    statements
```

- Ejemplo:

```
gpa = 1.4
if gpa >= 2.5:
    print "Bienvenido a la la UPR!"
else:
    print "Su solicitud es denegada."
```

- Si hay Multiple condiciones se usa `elif` ("else if"):

```
if condition:
    statements
elif condition:
    statements
else:
    statements
```

Logica BooleAlicia

Las expresiones en Python pueden tener "and"s y "or"s:

```
Alicia=3
Camila=25
if (Alicia <= 5 and Camila >= 10 or
Camila == 500 and Alicia != 5):
   print "Alicia and Camila"
```

El loop `for` [1]

- **`El loop for`**: Repite un conjunto de enunciados para un conjunto de valores.
  - Syntax:

    `for` ***variableName*** `in` ***groupOfValues*** `:`
    statements

    - Los enunciados a ser repetedidos son indentados con tabs or spaces.
    - ***variableName*** da un nombre a cada valor, para que puedan ser referidos en los enunciados
    - ***groupOfValues*** puede ser un rango de enteros especificados con la funcion ,`range` .

  - Ejemplo:

```
for x in range(1, 4):
    print x, "squared is", x * x
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
```

El loop for [2]

```
>>> names = ["Alicia", "Camila", "Julia"]
>>> for name in names:
...         print name
...
Alicia
Camila
Julia
```

Compresion de Listas

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Imprime [0, 1, 4, 9, 16]
```

Otra forma de hacer esto es usando la function map la cual sera vista mas adelante.

Hay otros tipos de containers (objetos para almacenar datos) tales como sets y dictionaries, pero que no seran discutidos aqui.

Break, continue

```
>>> for value in [3, 1, 4, 1, 5, 9, 2]:
...         print "Checking", value
...         if value > 8:
...                 print "Exiting for loop"
...                 break
...         elif value < 3:
...                 print "Ignoring"
...                 continue
...         print "The square is", value**2
...
```

```
Checking 3
The square is 9
Checking 1
Ignoring
Checking 4
The square is 16
Checking 1
Ighoring
Checking 5
The square is 25
Checking 9
Exiting for loop
>>>
```

Usar “break” para parar el for loop

Usar “continue” para parar el procesamiento del item

while

- **El loop while, ejecuta iterativamente un grupo de enunciados siempre que una condicion dada sea cierta.**
  - Bueno para loops *indefinidos* (repetir un enunciado un numero desconocido de veces)
- Sintaxis:

  while ***condicion***:

  enunciado
- Ejemplo:

```
number = 1
while number < 200:
    print number,
    number = number * 2
```

Imput de Files

- Muchos programas manipulan datos, que a menudo estan en files.
- Para leer el contenido de un file en python 2 usar:

variableName `= open("`***filename***`",'r')` o
variableName `= open("`***filename***`").read()`

Example:

```
File=open("c://esma3016/Animals2.csv").
 read()
```

Una forma mas rapida de leer

```
>>> lst= [ x for x in open("c://esma3016/Animals2.csv","r").readlines() ]
>>> lst
```

Tambien se puede leer datos directamente de la web usando el modulo urllib.

Pero la forma mas facil de leer un file es usando el modulo pandas (ver slide mas adelante)

File Output

```
input_file = open(“in.txt")
output_file = open(“out.txt", "w")
for line in input_file:
        output_file.write(line)
```

“w” = “write mode”
“a” = “append mode”
“wb” = “write in binary”
“r” = “read mode” (default)
“rb” = “read in binary”
“U” = “read files with Unix or Windows line endings”

Modulos

- Un programa en Python inicialmente solo tiene acceso a unas funciones basicas.

  (“int”, “dict”, “len”, “sum”, “range”, ...)
- dir(__builtins__) da una lista de las funciones disponibles.
- El uso de “Modulos” le Aliciade mas funcionalidad a Python. Para cargar un modulo se usa el comando “import” .
- Ejemplos

```
>>> import math   #Calculo de funciones matematicas
>>> import Numpy #Contiene calculos estadisticos y de matrices.
```

Mas modulos

```
>>>import  scipy # Para hacer calculos  cientificos tales como
    integracion numerica y optimizacion
>>> import  matplotlib # Para hacer graficas al estilo de matlab
>>> import  pandas #Para hacer Alicialisis estadistico
>>> import statsmodels #Para hacer regresion y series de tiempos
>>> import sklearn #Importa la libreria scikit-learn para hacer Machine
    learning
```

“import” y “from ... import ...”

```
>>> import numpy as np  #usando el alias np  para numpy
np.mean
>>> from math import cos, pi #importa solo cos y pi de math
cos  #se puede usar directamente cos
>>> from math import *  #importa todas las funciones de math
```

Algunos de estos modulos tienen sub-modulos

```
>>> import scipy.optimize
>>> import scipy.stats
>>> import matplotlib.pyplot as plt
```

Usando el modulo math

```
>>> import math
>>> math.pi
3.1415926535897931
>>> math.cos(0)
1.0
>>> math.cos(math.pi)
-1.0
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'exp', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp', 'log', 'log10',
'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
>>> help(math)
>>> help(math.cos)
```

Comandos del modulo math

- Python tiene varios commandos para hacer calculos.

Nombre del comando	Descripcion
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`,` ***value2***`)`	larger of two values
`min(`***value1***`,` ***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constante	Descripcion
`e`	2.7182818...
`pi`	3.1415926...

“import” y “from ... import ...”

```
>>> import math as m
m.cos
>>> from math import cos, pi
cos
>>> from math import *
```

Creando Funciones

Para crear funciones se usa el keyword def

```
def admision(igs):
    if igs> =300:
        return ‘Admitido'
    else:
        return ‘Denegado’

#Aplicando la funcion a 3 estudiantes
for x in [310, 290, 289]:
    print(admision(x))
#Una funcion puede tener mas de un argumentos
def admision1(nombre,igs):
    if igs>=350:
        print nombre, 'fue Admitido'
    elif igs<270:
        print nombre,'fue Denegado'
    else:
        print nombre, 'esta en la lista de espera'
```

Map

Funcion map() aplica una funcion dada a cada elemento de un objeto iterable (lista, tuple, etc.) y devuelve una lista de los resultados
La sintaxis es:
map(function,iterable)

```
def cuadrar(n):
    return n*n
x = (1, 2, 3, 4)
res= map(cuadrar, x)
print(res)
```

Lambdas

Una function lambda es una function sin nombre que se usa para definir operaciones sencillas.
map(function,iterable)

El ejemplo anterior puede ser ejecutado asi:

```
z = (1, 2, 3, 4)
res = map(lambda x: x*x, z)
print(res)
```

La funcion lambda puede tener mas de un argumento

```
x= [4, 5, 6]
y= [5, 6, 7]
result = map(lambda n1, n2: n1+n2, x, y)
print(list(result))
```

Creando Clases

Para crear funciones se usa el keword def

```
def admision(gpa):
    if gpa > =300:
        return ‘Admitido'
    else:
        return ‘Denegado’

def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)

hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True)  # Prints "HELLO, FRED!"
```

Modulo Numpy

Numpy es la libreria basica eh Python para llevar a cabo tareas de computacion cientifica . Trabaja con objetos que son considerados arreglos.
Un arreglo en Numpy es un conjunto de valores, todos ellos del mismo tipo, que es indexado por una tupla de vallores nonegativos. El numero de dimensiones es el rango del arreglo; el sghaoe de un arreglo es un tuple de enteros que da el tamnao del arreglo en cada una de las dimesnosne

```
import numpy as np
a = np.array([1, 2, 3])   # Crea un arreglo uni-dimensional
print(type(a))            # Imprime ‘<class 'numpy.ndarray'>"
print(a.shape)            # Imprime "(3,)"
a[0] = 5                  #Cambia el primer element del arreglo a al valor 5
print(a)                  # Prints "[5, 2, 3]"
b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
```

Numpy -1

```
a = np.zeros((2,2))   # Crea un arreglo 2x2  de ceros
print(a)
[[ 0.  0.]
 [ 0.  0.]]
b = np.ones((1,2))    # Crea un arreglo 1x2 de unos
print(b)
[[ 1.  1.]]
c = np.full((2,2), 5)  # Crea un arreglo 2 x 2  de 5's
print(c)
[[ 5.  5.]
 [ 5.  5.]]
d = np.eye(2)         # Crea una matriz identidad   2x2
print(d)
[[ 1.  0.]
 [ 0.  1.]]
```

Numpy-2

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Usando slicing para extraer un sub-arreglo consistentes de las dos  primeras
# filas y de las columnas 1 y 2
# b es un arreglo  2x2
b = a[:2, 1:3]
print b

x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

print(x + y)
print(np.add(x, y))
print(x * y)
print(np.sqrt(x))
print(x.T)
```

Pandas

Modulo para Alicialisis de datos, bastante similar a R.
Estructuras de datos:
Series: Arreglo uni-dimensional
DataFrames: Arreglp Bidimensional
Panel: Arreglo Tridimensional

```
import pandas as pd
import numpy as np
data = np.array([8,11,21,13])
s = pd.Series(data)
print s
s.mean()
s.median()
s.var()
s.std()
```

Pandas-1

```
import pandas as pd
data = [['Renato',19],['Paolo',16],['Camila',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print df
```

Data Science

Los componenets de un proceso en data science son:

1-Obtener los datos
2-Limpiar los datos
3-Explorar los datos
4-Modelar los datos
5-Interpretar los resultados

Leyendo datos con Pandas

```
# desde un file en su propia computadora
import pandas as pd
train=pd.read_csv('c:/Users/edgar2017/Downloads/titanic.
csv')
#directamente de la internet
#na_values representa los valores faltantes(missing
values)
breastdf=pd.read_csv("https://archive.ics.uci.edu/ml/mach
ine-learning-databases/breast-cancer-wisconsin/breast-
cancer-wisconsin.data",header=None,
sep=",",na_values=['?'])
```

Modulos para graficas

Matplotlib
Seaborn
Bokeh
Plotly
Plotnine

Modulos para Estadistica y Machine Learning

Scipy.stats
Statmodels
H20
Scikit-learn