1. Python

Este lenguaje se está usando bastante para el desarrollo de aplicaciones Web, así como para el desarrollo de programas tipo MapReduce para su ejecución en paralelo en ambientes tipo Hadoop. En estos apuntes se presentan algunos aspectos destacados de Python y se dan algunos ejemplos de programas elaborados con el lenguaje.

1.1 Intérprete Python

Los programas escritos en Python normalmente son interpretados, esto es, se utiliza un intérprete del lenguaje para analizar y ejecutar cada instrucción de un programa. Las instrucciones pueden darse de manera interactiva o a través de un archivo que contendría un programa completo.

En la siguiente sección se dan algunos ejemplos de instrucciones dadas de manera interactiva las cuales también muestran algunos aspectos básicos del lenguaje.

Se utiliza un intérprete gratuito bajado desde el sitio de Python: http://www.python.org.

Instalación

1) Bajar desde el sitio el archivo: python-3.6.x (x puede ser la 1, 2, …); copiarlo en C:/Temp.
2) Ejecutar el archivo. Cuando aparece la pantalla:

Hay que marcar la opción: Add Python 3.6 to PATH, para agregar en la variable Path la referencia al ejecutable de Python.
Seleccionar: Install Now, con lo cual la instalación se hará en la carpeta default del usuario.

3) Continuar la instalación hasta que se termine todo el proceso.
4) Al final, en Iniciar, Todos los programas, debe verse la entrada: Python 3.6. En C:, en la carpeta default de usuario, debe haberse creado la carpeta: Python36, con todos los programas del software.

1.2 Python interactivo

En esta sección se muestran algunas instrucciones básicas para introducir ciertas características del uso de los números y las cadenas en el lenguaje[1].

Ejecución de Python interactivo

Simplemente abrir el ambiente GUI o el de línea de comandos desde la entrada del programa en <Iniciar>; aparece el prompt: >>>. Aquí ya se pueden dar las instrucciones de manera interactiva. Se usará el término *shell* para referirse a uno u otro tipo de ambiente.

Números

Los comentarios en un programa se establecen con **#**.

Aquí se dan algunos ejemplos de la especificación de números y lo que debería contestar el intérprete:

```
>>> #Esto es un comentario.
... 2+2
4
>>> 2+2   #Un comentario junto al código.
4
>>> (50-5*6)/4
5.0
>>>
```

Los operadores aritméticos básicos son: +, -, *, /, // (división entera), % (módulo), ** (potencia).
El módulo se define como: x % y = x – ((x // y) * y).
Ejemplos:

```
>>> 14 // 5
2
>>> 14 / 5
2.8
>>> 14 % 5
4
>>> 3.5 % 0.5
0.0
>>> 14 % -5
-1
>>> -14 % -5
-4
>>>
```

[1] Varios de los ejemplos y descripción de las instrucciones están tomados de: Guido van Rossum y Fred L. Drake, Jr. (editor), "Guía de aprendizaje de Python", Release 2.0, BeOpen PythonLabs, 2000.

Los puntos suspensivos (…) indican continuación de instrucción. Se pueden asignar valores a variables y usarlas en instrucciones posteriores. Las variables **no necesitan** declararse previamente:

```
>>> ancho = 20
>>> alto = 5*9
>>> ancho * alto
900
>>> x = y = z = 0   #Poner a cero 'x', 'y' y 'z'
>>> x
0
>>>
```

Cadenas

Se pueden especificar entre comillas o apóstrofos:

```
>>> "Éste es un ejemplo"
'Éste es un ejemplo'
>>> 'Éste es otro ejemplo'
'Éste es otro ejemplo'
```

Se pueden concatenar cadenas con el operador + y repetirlas con *:

```
>>> palabra = 'Ayuda' + 'Z'
>>> palabra
'AyudaZ'
>>> '<' + palabra*5 + '>'
'<AyudaZAyudaZAyudaZAyudaZAyudaZ>'
```

Se pueden manejar caracteres individuales de una cadena accediéndolos por medio de índices. Para el manejo de éstos se puede pensar que los índices están entre carácter y carácter de la siguiente manera:

```
+---+---+---+---+---+---+
| A | y | u | d | a | Z |
+---+---+---+---+---+---+
0   1   2   3   4   5   6
-6 -5  -4  -3  -2  -1
```

En base a esto, los siguientes son ejemplos válidos:

```
>>> palabra[2]
'u'
>>> palabra[2:4]   #Se puede dar un rango de índices.
'ud'
>>> palabra[:2]    #Los primeros dos caracteres.
'Ay'
>>> palabra[2:]    #Todos menos los primeros dos caracteres.
'udaZ'
>>> palabra[-2]    #Se pueden usar los índices negativos.
'a'
>>> palabra[-5:-2]
'yud'
>>> palabra[1:-2]  #Se pueden combinar índices positivo y negativo.
'yud'
```

Esto es, cuando se da un índice solo, positivo o negativo, se toma el carácter de la derecha, salvo en el caso del último índice positivo en que marca error. Cuando se da un rango, positivos y/o negativos, se toman los caracteres que están dentro del rango, exceptuando el que está después del segundo índice.

No se pueden cambiar caracteres dentro de una cadena:

```
>>> palabra[0] = 'x'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Sin embargo, se puede crear una nueva cadena usando las facilidades anteriores:

```
>>> 'x' + palabra[1:]
'xyudaZ'
```

Se pueden hacer concatenaciones usando los índices:

```
>>> palabra[:2] + palabra[2:]
'AyudaZ'
```

1.3 Instrucciones de control de flujo

El lenguaje brinda las instrucciones típicas para el control del flujo de ejecución de un programa. La sangría en las instrucciones es significativa y delimita al conjunto de instrucciones que están dentro de otra instrucción.

If

```
if condición:
      instrucciones
elif condición:
      instrucciones
elif condición:
      instrucciones
else:
      instrucciones
```

Ejemplo:

```
#! coding: latin-1

x = int(input("Introduce un número: "))
if x < 0:
      x = 0
      print('Negativo cambiado a cero')
elif x == 0:
      print('Cero')
elif x == 1:
      print('Uno')
else:
      print("Más")
```

Hay varias cuestiones que se pueden comentar acerca del código anterior:

1. La línea:
   ```
   #! coding: latin-1
   ```
   indica el tipo de codificación que se usará para el programa. El código especificado en este caso permite el uso de acentos.
2. La función `input` permite leer un valor desde teclado. La función `int` guarda al valor como entero.
3. Los dos puntos (:) después del if y la sangría de sus siguientes dos líneas serían equivalentes a { } o begin-end en otros lenguajes.
4. La comparación por igualdad se hace con ==.
5. La palabra `elif` es equivalente a else if.

While

```
while condición:
      instrucciones
```

Ejemplo:

```
#! coding: latin-1

# Serie de Fibonacci.
a, b = 0, 1     #Se puede hacer este tipo de asignación.
while b < 10:
      print("a= ", a, "b= ", b)
      a, b = b, a + b
print()
```

Obsérvese la asignación doble de valores en la primera y última líneas (de código).

For

```
For variable in secuencia:
      instrucciones
```

La instrucción `for` cambia un poco con respecto a otros lenguajes ya que se aplica sólo a secuencias y se usa para recorrerlas en el orden en que aparecen en la secuencia. Una secuencia puede ser una **lista** o una cadena. Las listas, descritas más adelante, son secuencias de valores y, en principio, pueden verse como arreglos unidimensionales.

Ejemplo:

```
#! coding: latin-1

a = ['gato', 'ventana', 'casa']
for x in a:
      print(x, len(x))
```

La segunda línea muestra cómo se crea la lista. La función `len(x)` obtiene la longitud de la cadena almacenada en `x`. La primera línea (de código) muestra cómo se crea la lista. La función `len(x)` obtiene la longitud de la cadena almacenada en `x`.

Nota: también se puede usar el operador *not in* dentro de un for.

Función range()

Sirve para generar una secuencia de números partiendo de un valor inicial hasta un valor final, sin incluirlo, con incrementos de uno en uno. Opcionalmente se pueden indicar incrementos/decrementos con valores diferentes de uno.

Ejemplos (los resultados se verían como se muestra aquí, si el código se ejecutara en la versión 2.7):

```
#! coding: latin-1

>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

Ejemplos de for con range:

```
>>> for x in range(1, 5):
...     print(x)
...
1
2
3
4

>>> for x in range(1, 11, 2):
...     print(x)
...
1
3
5
7
9
```

1.4 Estructuras de datos

En esta sección se describen características importantes de las diferentes estructuras de datos que pueden usarse en Python.

1.4.1 Listas

Simplemente son listas de valores separados por comas y encerrados entre corchetes. Los elementos de una lista no tienen que ser necesariamente del mismo tipo. Como en las cadenas, los índices de una lista inician en cero; asimismo, se pueden emplear las diversas notaciones empleadas en las cadenas para especificar los índices de los elementos de una lista.

Ejemplos:

```
>>> a = ['nombre', 'apellido', 100, 1234]
>>> a
['nombre', 'apellido', 100, 1234]
>>> a[0]
'nombre'
>>> a[3]
1234
>>> a[-2]
100
>>> a[1:-1]
['apellido', 100]
>>> a[:2] + ['alias', 2*2]
['nombre', 'apellido', 'alias', 4]
```

A diferencia de las cadenas, cuyos valores no se pueden cambiar, **es posible remplazar los valores de los elementos de una lista**:

```
>>> a
['nombre', 'apellido', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['nombre', 'apellido', 123, 1234]
>>> len(a)
4
```

Se pueden cambiar, quitar y agregar elementos a una lista:

```
>>> # Cambiar elementos:
... a[0:2] = [1, 12]
>>> a
[1, 12, 123, 1234]
>>> # Quitar elementos:
... a[0:2] = []
>>> a
[123, 1234]
>>> # Agregar elementos (pudiendo cambiar el tamaño inicial de la lista):
... a[1:1] = [321, 4321, 54321]
>>> a
[123, 321, 4321, 54321, 1234]
```

Es posible anidar listas:

```
>>> q = [2, 3]
>>> p = [1, q, 4]
>>> len(p)
3
>>> p
[1, [2, 3], 4]
>>> p[1]
[2, 3]
>>> p[1][0]
2
```

Con esta característica se puede crear el equivalente a arreglos de varias dimensiones:

```
>>> b= [[0,1], [2,3]]
>>> b
[[0, 1], [2, 3]]
>>> b[0]
[0, 1]
>>> b[1]
[2, 3]
>>> b[1][1]
3
```

El tipo "lista" tienen métodos asociados:

append(x) Añadir un elemento al final de una lista.

extend(L) Extender la lista concatenándole todos los elementos de la lista indicada.

insert(i, x) Inserta un elemento en un posición dada. El primer argumento es el índice del elemento antes del que se inserta, por lo que a.insert(0, x) inserta al principio de la lista y a.insert(len(a), x) equivale a a.append(x).

remove(x) Elimina el primer elemento de la lista cuyo valor es x. Provoca un error si no existe tal elemento.

index(x) Devuelve el índice del primer elemento de la lista cuyo valor sea x. Provoca un error si no existe tal elemento.

count(x) Devuelve el número de veces que aparece x en la lista.

sort() Ordena ascendentemente los elementos de la propia lista.

reverse() Invierte la propia lista.

Se puede usar `del` para eliminar elementos de una lista dado su índice o un rango de índices:

```
>>> a=[10,20,30,40,50]
>>> a
[10, 20, 30, 40, 50]
>>> del a[3]
>>> a
[10, 20, 30, 50]
>>> del a[1:3]
```

```
>>> a
[10, 50]
>>> del a
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

Al final marca error porque la lista ya no existe.

1.4.2 Tuplas

Una tupla es parecida a una lista, aunque tiene más el sentido de tupla en el modelo relacional: una serie de valores separados por comas. Ejemplos:

```
>>> t = 12345, 54321, '¡hola!'
>>> t[0]
12345
>>> t
(12345, 54321, '¡hola!')
>>> # Se pueden anidar tuplas:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, '¡hola!'), (1, 2, 3, 4, 5))
```

Las tuplas tienen las siguientes características:

- Al imprimirlas se muestran encerradas entre paréntesis.
- En la entrada los paréntesis son opcionales.
- Pueden anidarse.
- **No pueden cambiarse sus elementos**.
- **Pueden contener listas, las cuales sí pueden cambiarse**.

Tuplas de 0 o 1 elemento:

```
>>> vacio = ()
>>> singleton = 'hola', # <-- Obsérvese la coma final: necesaria.
>>> len(vacio)
0
>>> len(singleton)
1
>>> singleton
('hola',)
```

1.5 Funciones

Las funciones en Python son similares al concepto que se tiene de ellas en otros lenguajes de programación.

En Python se definen de la siguiente forma:

```
def  nombreFunción(parFormal_1, parFormal_2, …, parFormal_n):
    instrucción_1
    instrucción_2
         -----
    instrucción_m
    return resultado
```

donde:

parFormal_1, …, parFormal_n: pueden ser de cualquier tipo de datos. Opcionalmente se pueden escribir como: parFormal= *valor*, en cuyo caso el parámetro toma por default el *valor* indicado, si en el llamado a la función no se da valor para dicho parámetro.

instrucción_1, …, instrucción_m: pueden ser cualquier instrucción de Python.

return resultado: en el conjunto de instrucciones de la función debe haber al menos una instrucción *rteurn* que regresa el resultado del proceso que hace la función.

Ejemplo: definición de la función que encuentra el máximo de dos valores:

```
def maxVal(x, y= 15):
    if x > y:
        return x
    else:
        return y
```

Llamados a la función:

```
>>> maxVal(20,40)
40
>>> maxVal(40,20)
40
>>> maxVal(10)
15
>>> maxVal(x=10, y=50)
50
>>> maxVal(y=10, x=50)
50
>>>
>>> maxVal(y=10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: maxVal() missing 1 required positional argument: 'x'
>>> maxVal(x=10)
15
>>>
```

1.6 Diccionarios

Son conjunto no ordenados de pares clave:valor. Los pares se encierran entre {}, separando cada par con coma.

Ejemplo: `dic= {'a':10, 'b':20, 'c':30}`

Presentan las siguientes características:

- Las claves, en principio, pueden ser cadenas o números, aunque también pueden ser tipos inmutables como las tuplas.
- Las claves deben ser únicas dentro del mismo diccionario.
- El diccionario vacío se representa con {}.
- El acceso individual a elementos se hace como en una lista dando como índice la clave del elemento. Por ejemplo: `dic['a']` regresa el valor 10.
- Si se asigna un valor a un elemento cuya clave ya existe, entonces el nuevo valor sustituye al viejo. Por ejemplo: `dic['b']= 40`, pone 40 en el segundo elemento, perdiéndose el 20.
- Para agregar un nuevo elemento se escribe: `dic['d']= 50`, con lo que el diccionario quedaría como: `dic= {'a': 10, 'b': 40, 'c': 30, 'd': 50}`
- Aunque los valores anteriores se ven ordenados por clave, el conjunto no está ordenado.

Los métodos, principales, que se pueden usar con los diccionarios son:

- **len(dic)** regresa la cantidad de elementos de dic (4, según el último ejemplo).
- **list(dic.keys())** regresa una lista con las claves de dic.
- **list(dic.values())** regresa una lista con los valores de dic
- **k in dic** regresa True si k está en las claves de dic.
- **del(dic[k])** elimina el elemento (completo) con clave k.
- **for k in dic** itera sobre las claves de dic.
- **sorted(dic.keys())** regresa una lista con las claves ordenadas.

También:

- Con `dict()` se puede construir un diccionario:

```
dict([('sophie', 4139), ('guido', 4127), ('jack', 4098)])
```

- Se puede hacer:

```
{x: x**2 for x in (2, 4, 6)}
```

1.7 Entrada/salida

En esta sección se describen las instrucciones que se utilizan para leer/escribir datos, en su versión simple.

Lectura desde teclado

Se puede realizar usando el par de instrucciones indicadas en el siguiente código:

```
# Lee datos desde teclado, los guarda en una lista, los
# multiplica por 2 y los escribe en pantalla.

# Con estas dos instrucciones se lee desde teclado:
cad = input('Escribe números separados por comas: ')
arre = cad.split(',')

print('Los números de la lista multiplicados por 2 son:')
for num in arre:
      print(int(num) * 2)
```

Lectura y escritura con archivos

El manejo de los archivos en Python es parecido a como se hace en C. El siguiente programa muestra una lectura de datos y escritura de resultados empleando archivos:

```
# Lee datos desde un archivo, los multiplica por 2 y
# los escribe en otro archivo.
print('Inicié lectura y escritura de archivos')

# Apertura del archivo.
flec= open('Datos.txt','r')         #Con 'r' el arch. se abre para lectura.
fesc= open('Resultados.txt','w')    #Con 'w' se abre para escritura.

#Escritura en el archivo de salida.
fesc.write('Los números, y su multiplicación por 2, son: \n')

# Lectura desde archivo (cada línea se lee como cadena):
for línea in flec:
  arre = línea.split(',')     # Para separar cada número.

  for num in arre:
    num2= float(num) * 2
    salida= str(num) + '\t' + str(num2) + '\n'
    fesc.write(salida)    #Escritura en el archivo de salida.

# Cierra archivos.
flec.close()
fesc.close()
print('Terminé lectura y escritura de archivos')
```