Python

Python es un lenguaje de programación de propósito general de alto nivel:

- Dado que el código se compila automáticamente y es ejecutado, Python es adecuado para su uso como lenguaje de implementación de aplicaciones web, o programación matemática, etc.
- Porque Python puede ampliarse en C y C++, Python puede proporcionar la velocidad necesaria, incluso para computar tareas intensivas.
- Debido a sus buenas construcciones de estructura (bloques de código anidados, funciones, clases, módulos y paquetes) y su uso consistente de los objetos y la programación orientada a objetos, Python nos permite escribir, aplicaciones lógicas claras para tareas pequeñas y grandes.

Importantes características de Python

- Tipos de datos incorporados: cadenas, listas, diccionarios, etc.
- Las estructuras de control habituales: if, if-else, if-elif-else, while, además del potente iterador (for).
- Varios niveles de estructura organizativa: funciones, clases, módulos y paquetes. Estos ayudan en la organización de código. Un ejemplo excelente es la librería estándar de Python.
- Compilar sobre la marcha el código de bytes. El código fuente es compilado a código de bytes sin un paso de compilación independiente.

Importantes características de Python

- Python proporciona una forma consistente para utilizar objetos: todo es un objeto. Y en Python es fácil de poner en práctica nuevos tipos de objetos (llamados clases).
- Extensiones en C y C++. Módulos de extensión usando herramientas como Swig.
- Jython es una versión de Python que funciona ’bien’ con Java. The Jython Project.

Algunos enlaces importantes

- Todo el mundo Python esta aquí, Web de Python.
- El conjunto de documentación de Python estándar, Python documentation.
- Página de Ipython, un entorno interactivo de computación científica, Ipython interactive computing.
- Una serie de artículos, sobre la parte interna de Python, Python Internal.
- Artículos traducidos sobre el estilo de Python y mucho mas, Python idiomático.
- Una colección de lecturas de Python, sobre muchos temas, Lecturas de Python.

Corriendo Python

```
% python
Python 2.7.5+ (default, Sep 19 2013, 13:49:51)
[GCC 4.8.1] on linux2
Type "help", "copyright", "license" for more information.
>>> print "Python"
Python
```

Sobre Windows y Macintosh

- Python es lanzado como una aplicación.
- Una ventana con un interprete aparece y tu puedes ver el prompt.

Corriendo Python

El programa se termina al

- Tipear Control-D o Control-Z en el prompt interactivo.
- El programa corre hasta que EOF es alcanzado.
- o si escribimos

```
raise SystemExit
```

Convención-PEP8

Python utiliza indentación para mostrar la estructura de bloque. Indentación en el primer nivel muestra el inicio de un bloque. ’La proxima indentación fuera ’ de la anterior muestra el final de un bloque. Por ejemplo:

```
if x:
    if y:
        f1()
    f2()
```

Y, la convención es usar cuatro espacios para cada nivel de indentación. En realidad, es más que una convención, es prácticamente un requisito.

El primer programa

Hello World

```
>>> print "Hola a todos"
Hola a todos
```

Podemos poner algo de código en un archivo hello.py, como esto

```
print "Hola a todos"
```

y ejecutarlo de la siguiente forma

```
% python hello.py
```

Variables y Expresiones

Expresiones

Las operaciones éstandar de matemáticas trabajan como otros lenguajes.

```
3+4
3 ** 4
'Hello' +'Python'
3 * ( 4 + 5)
```

Asignamiento de Variables

```
a = 4 << 3
b = a * 4.5
c = (a + b)/2.5
d = "Hello World"
```

Variables y Expresiones

- Las variables son tipeadas dinámicamente (No tipificación explícita, los tipos pueden cambiar durante la execución.)
- Las variables son sólo nombres para un objeto. No están atados a una localización de memoria como en C.
- El operador de asignación crea una asociación entre el nombre y el valor.

Variables y Expresiones

Escribir en un archivo de texto con la extensión .py lo siguiente y ejecutarlo

```
principal = 1200     # Cantidad inicial
rate = 0.05          # Taza de interes
numyears = 5
year =1
while year <= numyears:
    principal = principal * (1 + rate)
    print year, principal
    year += 1
```

Y el resultado es.....

Variable0.py

Condicionales

if-else

```
# Calcular el maximo de a y b
if a < b:
    z = b
else:
    z =a
```

La declaración pass

```
if a < b :
    pass      # No se hace nada
else:
    z = a
```

Condicionales

Notas:

- La indentación es usada para denotar los cuerpos de if-else.
- pass es usado para denotar un cuerpo vacío.
- Python no tiene una especial declaración *switch* o *case* para testear valores. Para manejar múltiples casos de test, usamos la declaración *elif*.

Condicionales

Declaración elif

```
if a == 'Python':
    op = 1
elif a == 'C++':
    op = 2
elif a = 'Node':
    op = 3
else:
    op = 4
```

Expresiones booleanas: and, or, not

```
if b >=a and b <=c :
    print "b esta entre a y c"
if not (b < a or b > c):
    print "b esta todavia entre a y c"
```

Tipos básicos (Números y cadenas)

Números

```
a = 3                  # Entero
b = 4.9                # Punto flotante
c = 57484442227L    # Entero grande (Precision arbitraria)
d = 3 + 8j             # Numero Complejo
```

Cadenas

```
a = 'Python'           # Una coma
b = "Optimizacion"     # Doble coma
c =  "Python tiene un modulo llamado 'Numpy'"
d ='''Una cadena de triple coma
puede escribirse  en multiples
lineas '''
e ="""Tambien trabaja para
comas dobles y todo funciona bien"""
```

Tipos básicos (Listas)

Lista de arbitrarios objetos

```
a = [2,3,4]                    # Una lista de enteros
b = [2, 7, 3.5, "Python"]      # Una mix de objetos
c = []                         # Una lista vacia
d = [3, [a,b]]                 # Una lista de lista
e = a + b                      # Unir una lista
```

Manipulación de lista

```
x = a[1]                       # 2–elemento de la lista
y = b[1:3]                     # Retorna una sublista
z = d[1][0][2]                 # Lista anidada
b[0] = 42                      # Cambia un elemento
```

Tipos básicos (Tuplas)

Tuplas

```
f = (3, 6, 7)          # Una tupla de enteros
g = (,)                # Una tupla vacia
h =(2, [3,4], (10,11,12))
```

Manipulación de tuplas

```
x = f[1]               # acceder al elemento, x=3
y = f[1:3]             # y = (3,4)
z = h[1][1]            # z = 4
```

- Las tuplas son como las listas, pero el tamaño es fijado en el tiempo de creación.
- No podemos reemplazar los miembros de una lista (se dice que es ’inmutable’).

Tipos básicos (Diccionarios)

Diccionarios

```
a = {}
b = {'Django': 1, 'Node.js' :2, 'R':3, }
c = {'uid' : 105,
 'login' : 'Lara',
 'name':  'Cesar Lara Avila'
 }
```

Acceso a diccionarios

```
u = c['iud']
c['shell'] = "/bin/zsh"  # Colocamos un nuevo elemento
if c.has_key("login"):   # Vemos si esta este elemento
    d = c['login']
else:
    d = None

d = c.get("login", None) # Lo mismo, pero mas compacto
```

Bucles

La declaración While

```
while a < b:
    # Hacer algo
    a = a +1
```

La declaración for (Iterar sobre los miembros de una secuencia)

```
for n in range (1,10):
    print "2 a la potencia %d is %d" %(n, 2**n)
```

Nota:

range(i, j, [, paso]) es una función que crea un objeto que representa un rango de enteros con valores desde *i* a *j - 1*.

Bucles

- La declaración *for* no está limitada a secuencias de números y puede ser usada para iterar sobre muchos tipos de objetos incluyendo listas, diccionarios y archivos. ejemplo3.py
- El bucle for es una de las características más poderosas de Python muy relacionadas a funciones generadores y programación funcional.

Basico-Python.py

Funciones

La declaración def

```
#Retorna el resto de a/b
def remainder (a,b):
    q = a//b        # truncamiento de la division
    r = a - q*b
    return r

#Ahora como usarlo
a = remainder (42,5)      # a = 2
```

Retornando múltiples valores

```
def divide(a,b):
    q = a//b
    r = a - q*b
    return r
x ,y =  divide(42, 5)  # x = 8, y = 2
```

Funciones

```
def test(msg, count):
    for idx in range(count):
        print '%s_%d' % (msg, idx)

test('Test:', 3)
```

Al igual que con otros objetos de Python, puedes rellenar un objeto función en otras estructuras como tuplas, listas y diccionarios. He aquí un ejemplo:

```
# Creamos una tupla:
val = (test, 'A_label:', 5)

# Llamamos a la funcion:
val[0](val[1], val[2])
```

Funciones

Una función con argumentos por defecto

```
def testDefaultArgs(arg1='default1', arg2='default2'):
    print 'arg1:', arg1
    print 'arg2:', arg2

testDefaultArgs('Otro valor')
```

La salida debe ser algo así

```
arg1: Otro valor
arg2: default2
```

funcion0.py

Funciones

Las listas de argumentos y listas de argumentos de palabra clave (keywords)

```
def testArgLists_1(*args, **kwargs):
    print 'args:', args
    print 'kwargs:', kwargs


testArgLists_1('aaa', 'bbb', arg1='ccc', arg2='ddd')


def testArgLists_2(arg0, *args, **kwargs):
    print 'arg0: "%s"' % arg0
    print 'args:', args
    print 'kwargs:', kwargs
```

Funciones

```
def test():
    testArgLists_1('aaa', 'bbb', arg1='ccc', arg2='ddd')
    print '-' * 40
    testArgLists_2('Primer argumento', 'aaa', 'bbb',
                   arg1='ccc', arg2='ddd')
test()
```

Como resultado de ejecutar esto, tenemos:

```
args: ('aaa', 'bbb')
kwargs: {'arg1': 'ccc', 'arg2': 'ddd'}
args: ('aaa', 'bbb')
kwargs: {'arg1': 'ccc', 'arg2': 'ddd'}
----------------------------------------
arg0:" Primer argumento"
args: ('aaa', 'bbb')
kwargs: {'arg1': 'ccc', 'ag2': 'ddd'}
```

funcion1.py

Funciones

Llamar a una función con argumentos de palabra clave

funcion2.py

```
#Llamando a una funcion con argumentos
#de palabras claves (keyword)

def test_keyword_args(color_1 ='verde',
        color_2 = 'azul',
     color_3 = 'red',
        color_4 = 'marron'):
        print 'color_1: "%s"'   %color_1
        print 'color_2: "%s"'   %color_2
        print 'color_3: "%s"'   %color_3
        print 'color_4: "%s"'   %color_4
```

Funciones

```
def test():

    test_keyword_args()
    print '-'*40
    test_keyword_args(color_2 ='amarillo')
    print '-' *40
    test_keyword_args(color_3 = 'naranja',
            color_4 = 'violeta')

test()
```

Comprobar el resultado.

Ejercicio

Ejecutar el archivo funcion3.py y en un archivo con la extensión .py escribir el siguiene código,

```
def print_two(*args):
    arg1, arg2 = args
    print "arg1: %r, arg2: %r" %(arg1, arg2)
def print_two_again(arg1, arg2):
    print "arg1: %r, arg2: %r" %(arg1, arg2)
def print_one(arg1):
    print "arg1: %r" %arg1
def print_none():
    print "Nada"

print_two("_____", "_____")
print_two_again("_____", "_____")
print_one("_____")
print_no
```