Introducción a la Programación en Python

José Ortiz Bejar

job@correo.fie.umich.mx

Universidad Michoacana de San Nicolás de Hidalgo

26 de agosto de 2014

Introducción

- Tiempos importantes en el desarrollo de software:
  - Tiempo de desarrollo
  - Tiempo de ejecución
- Desempeno de lenguajes de alto nivel
  - desarrollo rápido
  - son generalmente lentos en tiempo de ejecución
- Objetivos del curso
  - Introducción al lenguaje de programación Python
  - Obtener habilidades básicas de desarrollo de software en python

Ventajas de Python

- Lenguaje interpretado tan flexible como shell pero con estructuras de datos reales
- Licenciado bajo GPL y disponible para varios sistemas operativos
- Ampliamente utilizado y bien documentado
- Más simple que otros lenguajes interpretado como perl

Desventajas de Python

- Los lenguajes interpretados como Python son más lentos que los compilados
- No tiene tantas librerias númericas como MATLAB

Ciclo de Ejecución

- Los lenguajes compilados utilizan dos ciclos de ejecución:
  - Compila el código fuente, se genera un archivo que contiene código de máquina ejecutable
  - El Sistema operativo(OS) o una máquina virtual ejecuta el contenido de archivo compilado
- Los lenguajes interpretados combinan los dos pasos
- El compilador y la máquina virtual son el mismo programa
- Carga el código fuente, lo transforma en código mas compacto si es necesario, y lo ejecuta

Ejecución de programas escritos en Python

- Se puede ejecutar mediante un interprete. Para salir del interprete presionando control-D.

```
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on d
Type "help", "copyright", "credits" or "license" for more i
>>> print 124/28
4
>>> print 124.0/28
4.42857142857
>>> ^D
```

Ejecución de programas escritos en Python

- Por convención se guarda el programa en un archivo con extensión *.py*

```
$ cat saved.py
print 124/28
print 124.0/28.0
$ python saved.py
4
4.42857142857
```

Ejecución de programas escritos en Python

- En Unix/Linux, puede ponerse *#!/usr/bin/python* como la primera línea del programa.
  - Con lo anterior le decimos al SO que ejecute */usr/bin/python* con el resto del archivo como su entrada
- Lo anterior no funciona si Python está instalado en una ubicación diferente a *#!/usr/bin/python*

Ejecución de programas escritos en Python

- Una mejor practica es poner *#!/usr/bin/env python* como la primera linea del programa
  - Esto le dice al SO que utilice */usr/bin/env* para encontrar donde se instalo el interprete y después ejecute el script con él

```
$ cat hashbang.py
#!/usr/bin/env python
print 124/28
print 124.0/28.0
$ ./hashbang.py
4
4.42857142857
```

- En windows, los archivos .py son asociados con Python
  - La asociación ocurre cuando se instala el interprete
  - Hacer doble click sobre un archivo .py ejecutará el interprete automáticamente

Variables

- No se necesitan declaraciones: Las variables son creadas al momento de la asignación

```
planeta = "Marte"
luna = "Deimos"
p = planeta
```

- Tipeado dinámico: Una variable puede referirse a diferentes tipos de valores en diferentes instantes

```
planeta = "Marte"
luna = "Deimos"
p = planeta
planeta = 9
```

Posibles errores

- Las variables deben tener un valor antes de ser utilizadas

```
planeta = "Tierra"
print plant      # note the misspelling

Traceback (most recent call last):
  File "lec/inc/py01/undefined_var.py", line 2, in ?
    print plant      # note the misspelling
NameError: name 'plant' is not defined
```

- A diferencia de otros lenguajes Python no asigna valores por defecto, debido a que esto puede ocultar posibles errores

Nota: Cualquier texto a partir de "#"hasta el final de linea es un comentario

Acerca de los tipos

- Las variables no tiene tipos, pero los valores si. Python lanzará un error si se pretende operar dos valores con tipos incompatibles

```
>>>x = "dos"     # "dos" es una cadena
>>>y = 2         # 2 es un entero
>>>print x * y  # multiplicar una cadena por un entero,
                # concatena la cadena
dosdos
```

Acerca de los tipos

```
>>>print x + y      # no es posible sumar una cadena
                   # y un entero

Traceback (most recent call last):
  File "lec/inc/py01/add_int_str.py", line 4, in ?
    print x + y              # but you can't add an integer a
TypeError: cannot concatenate 'str' and 'int' objects
```

Imprimir

- La sentencia print imprime cero o mas valores a la salida estándar
- Automáticamente agrega un salto de linea al final. Poniendo ",.al final de la linea el salto se omite.

```
planeta = "Marte"
numero_de_lunas = 2
luna1 = "Deimos"
luna2="Phobos"
print planeta, "tiene", numero_de_lunas, "satelites,",
print "sus nombres son:", luna1, "y", luna2

Marte tiene 2 satelites, sus nombres son: Deimos y Phobos
```

Comillas

- Se utilizan comillas simples o dobles para crear cadenas
  - Cada cadena debe comenzar y terminar con el mismo tipo de comillas
  - Diferentes cadenas en el mismo program pueden utilizar diferentes tipos de comillas

```
print "Cita: \"Ser o no ser esa es la cuestion..\""
Cita "Ser o no ser esa es la cuestion¨
```

Comillas

- Comillas triples, se sutilizan para crear cadenas de múltiples lineas

```
print '''Marte, apodado a veces como el Planeta Rojo,
         es el cuarto planeta del Sistema Solar. Es,
         en muchos aspectos, el m\'as parecido a la
         Tierra.'''
#---------Salida-----------
Marte, apodado a veces como el Planeta Rojo,
es el cuarto  planeta del Sistema Solar. Es,
en muchos aspectos, el mas parecido a
la Tierra.
```

Convertir valores a cadenas

- La función *str* convierte valores a cadenas

```
print "Diametro: " + str(1280) + "-" + str(1760) + " km"
Diameter: 1280-1760 km
```

- Se utilizan *int*, *float*, etc para convertir valores a otros tipos.

```
print int(12.3)
print float(4)
12
4.0
```

Secuencias de escape

- Las secuencias de escape se utilizan para poner caracteres especiales como parte de cadenas.

Expresión	Significado
\\	backslash
\'	comilla simple
\"	Comillas dobles
\b	backspace
\n	salto de linea
\r	retorno de carro
\t	tabulador

Números

- 14 entero de 32 bits
- 14.0 flotante de doble precisión (64 bits).
- 1+4j número complejo
  - mediante x.real y x.imag se accede a la parte real e imaginaria
- 123456789L entero largo
  - Longitud arbitraria
  - operaciones lentas

Aritmética

Nombre	Operador	Uso	Valor	Notas
Adición	+	35 + 22 'Py' + 'thon'	57 'Python'	
Resta	-	35 - 22	13	
Multiplicación	*	3 * 2 'Py' * 2	6 'PyPy'	
División hline	/	3.0 / 2 3 / 2	1.5 1	 División entera redondea hacia abajo : $-3/2$ es -2, no -1
Potencia	**	2 ** 0.5	1.4142135623730951	
Resto	%	13 % 5	3	

Operaciones lógicas

- **True** y **False** son los valores de verdad
- La cadena vacía y 0 son considerados falso
- Casi cualquier valor es verdadero
- Se combinan las expresiones lógicas utilizando **and**, **or** y **not**

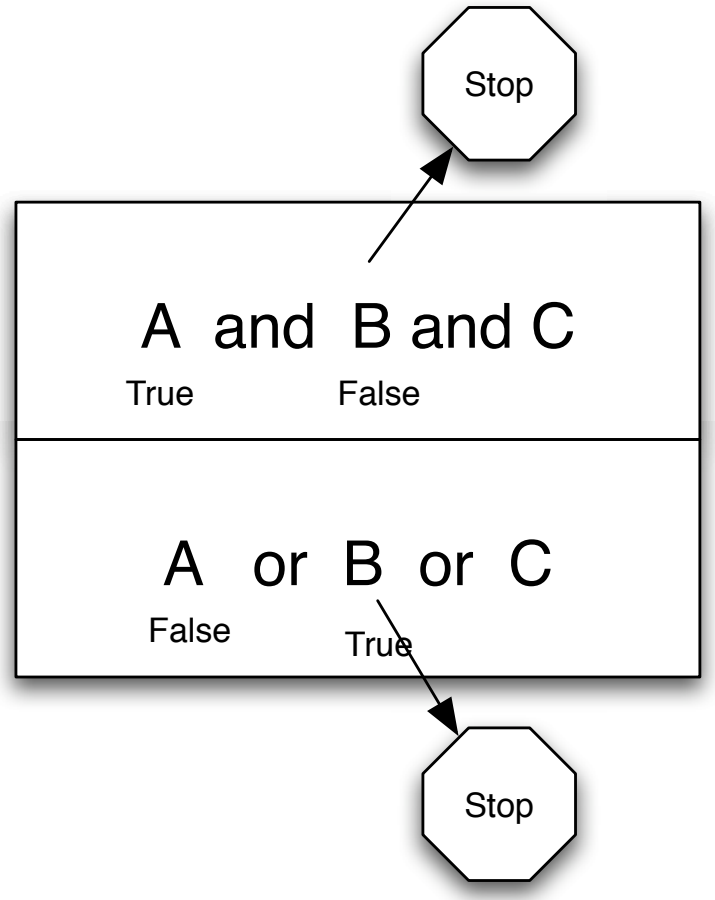
Operaciones lógicas

Expresión	Resultado	Notas
True or False	True	
True and False	False	
'a' or 'b'	'a'	Detiene la evaluación después de evaluar 'a'
'a' and 'b'	'b'	No se detiene hasta que evalúa 'b'
0 or 'b'	'b'	0 es falso, pero 'b' es verdadero
0 and 'b'	0	Como 0 es falso la evaluación se detiene
0 and (1/0)	0	1/0 es un error, pero nunca se evalúa
(x and 'a') or 'b'	Depende	Si x es verdadero, esta expresión es 'a', si x es falso 'b'

Evaluación de corto circuito

- and y or son operadores de corto circuito (short-circuit)
  - Evalúa expresiones de izquierda a derecha
  - Detiene la evaluación tan pronto como conoce la respuesta
  - El resultado es lo último en ser evaluado

Evaluación de corto circuito



Evaluación de corto-circuito

- Es posible crear expresiones como *val= cond and left or right*
  - Si cond es True, se asigna left a val
  - Si cond es False, se asigna right a val.
  - Es Difícil de leer
  - Algunas veces lo inteligente no es elegante

Comparación

- Utiliza los mismos que C, pero es posible encadenar comparaciones.

Expresión	Valor
$3 < 5$	True
$3{,}0 < 5$	True
$3 ! = 5$	True
$3 == 5$	False
$3 < 5 <= 7$	True
$3 < 5 >= 2$	True (difícil de leer)
$3 + 2j < 5$	Error: en número complejos solo se pueden utilizar == y !=

Comparación de cadenas

- Los caracteres son codificados como números: los menores son los números, después las letras mayúsculas y finalmente las minúsculas
- Las cadena son comparadas caracter a caracter desde el inicio hasta:
  - Un caracter sea menor que otro
  - Alcanzar el final de una de las cadenas

Comparación de cadenas

Expresión	Valor
$'abc' < 'def'$	True
$'abc' < 'Abc'$	False
$'ab' < 'abc'$	True
$'0' <' 9'$	True
$'100' <' 2'$	True

Condicionales

- Python utiliza *if*, *elif* (no *else if*), y *else*
- Utiliza dos puntos(:) para indicar niveles de anidamiento

```
a = 3
if a < 0:
    print 'less'
elif a == 0:
    print 'equal'
else:
    print 'greater'

greater
```

indentación

- Porqué Python no utiliza begin/end ...?
  - Estudios muestran que todos observan la indentación
  - Es muy común encontrar recomendaciones en los libros de programación
- No importa el tamaño de la indentación
  - Todo en el bloque debe tener la misma indentación
  - Normalmente se utilizan 4 espacios
- Es preferible no usar tabulador
  - El interprete de python lo interpreta como ocho espacios
  - Editores diferentes pueden interpretarlo diferente

While

- Repetir una operación, mientras una condición de cumpla

```
num_moons = 3
while num_moons > 0:
    print num_moons
    num_moons -= 1
```

```
3
2
1
```

While

- Repetir una operación cero veces si la condición es falsa

```
print 'before'
num_moons = -1
while num_moons > 0:
    print num_moons
    num_moons -=1
print 'after'

before
after
```

While

- Si la condición es siempre verdadera, el ciclo nunca termina

```
num_moons = 3
while num_moons > 0:
    print num_moons
    # oops --- olvide restar 1

3
3
3
...
```

Break

- Es posible interrumpir un ciclo antes de que termine, mediante el uso de la sentencia *break*

```
num_moons = 3
while True:                 # Parece un ciclo infinito...
    print num_moons
    num_moons -= 1
    if num_moons <= 1:
        break               # ... una salida
```

```
3
2
```

Continue

- Se puede saltar la ejecución de una sola iteración mediante *continue*

```
num_moons = 5
while num_moons > 0:
    print 'top:', num_moons
    num_moons -= 1
    if (num_moons % 2) == 0:
        continue
    print '...bottom:', num_moons
#------------Salida--------------------
top: 5
top: 4
...bottom: 3
top: 3
top: 2
```

Formato de cadenas

- El operador % permite aplicar formato a la salida
  - El lado izquierdo indica sobre cuantos elementos se aplica formato
  - El lado de derecho indica los valores de los elementos
    - Un solo valor se pone solo
    - Varios valores van entre parentesis, separados por comas

Formato de cadenas

```
planeta='Marte'
luna1='Deimos'
luna2='Phobos'
formato='%s es el planeta rojo'
print formato %planeta
print "%s tiene dos lunas: %s y %s" %(planeta,luna1,luna2)
#----------------------------Salida----------------------
Marte es el planeta rojo
Marte tiene dos lunas: Deimos y Phobos
```

Especificadores de formato

- 'Izquierda %d derecha %d' %(1,-1) crea 'Izquierda 1 derecha -1'
  - %d indica formato decimal entero
- ' %04d' % 13 crea '0013'
  - El 0 al inicio indica que se deben insertar 0 al inicio
  - EL 4 significa que el tamaño debe ser al menos de 4 caracteres
- ' %6.4f %%' % 37.2 crea "37.2000
  - Numero flotante de al menos 6 caracteres, rellenado a la izquierda con espacios, con 4 caracteres después del punto decimal
  - " %%.^{es} traducido como "

Formatos Soportados

Formato	Significado	Ejemplo	Salida
“d”	Entero decimal con signo	’%d %d’ %(13, 15)	”13 15”
“o”	Octal sin signo (base-8)	’%o %o’ %(13, 15)	”15 17”
“x”	Hexadecimal sin signo (base-16 minúsculas)	’%x %x’ %(13, 15)	”d f’
“X”	Hexadecimal sin signo (base-16 mayúsculas)	’%X %X’ %(13, 15)	”D F”
“e”	Exponencial punto flotante (minúsculas)	’%e’ %123.45	”1.234500e+02”
“E”	Exponencial punto flotante (mayúsculas)	’%E’ %123.45	”1.234500E+02”
“f”	Decimal de punto flotante	’%f’ %123.45	”123.4500”
“s”	Cadena (Convierte otros tipos utilizando str())	’%s %s %s’ %(’nickel’, 28, 58.69)	”nickel 28 58.69”

Ayuda en el interprete

- Para descubir las funciones que tiene un objeto
  - dir() Lista los atributos y métodos de un objeto
  - help() imprime la dicumentación

Ayuda

```
>>> cadena='hola'
>>> dir(cadena) #Informacion del objeto
['__add__',
 '__class__',
 salida omitida
 'upper',
 'zfill']
>>> help(cadena.upper)
Help on built-in function upper:

upper(...)
    S.upper() -> string

    Return a copy of the string S converted to uppercase.
```

Sumario

- Los lenguajes script están incrementando su popularidad debido a que optimizan el tiempo de desarrollo. El cual es ahora mas costoso que el tiempo de maquina en muchos casos
- Python es uno de los lenguajes script más claros. Es una excelente herramienta para construir software.

Ejercicios

- Hacer un programa que muestre "Hola mundo", y ejecutarlo para ver el mensaje en la pantalla. Cómo lo ejecutó? Qué otras maneras había de ejecutarlo?
- Abra el Intérprete Interactivo de Python, y realice algunas acciones simples. Ingrese la órden help().

Ejercicios

- Escriba un programa que muestre la siguiente figura:

```
  ccccc
o @ @ o
   ^
  \===/
```

Ejercicios

- Siendo a=”Qui hubo,”, b=”cuando”, c=28, imprimir las siguientes cadenas:

```
"Qui hubo, cuando"  (usando a y b)
"-Qui hubo,-cuando-"  (usando a y b)
"El resultado es: 28"  (usando c)
"El resultado es: 28 min (1680seg)"  (usando c ambas ve
"La temperatura es: 28"
```

Ejercicios

- Escribir un programa que dada una cadena la imprima a pantalla al revés. Por ejemplo, para el texto "Qui hubo", debería mostrar: "obuh iuQ"
- Escribir un programa que dado número muestre ese número menos dos, más dos, multiplicado por dos, dividido por dos de forma entera, dividido por dos de forma decimal, y elevado a la potencia de dos. Por ejemplo, para el número 5, debería mostrar: 3, 7, 10, 2, 2.5, 25
- Dado número, elevarlo al cuadrado, y mostrar los dígitos al reves y separados por espacio. Por ejemplo, si el usuario ingresa 25, la salida tiene que ser "5 2 6".