El lenguaje de programación python: (parte 1 de 2)

Marcos Capistrán[1]

[1] www.cimat.mx/∼marcos

15 de Agosto de 2011

Cómputo científico

- Cómputo científico: Diseño y análisis de algoritmos para resolver numéricamente problemas matemáticos en ciencia e ingeniería
- La computadora se usa como un microscopio
- Hay mucho software (mathematica, maple, matlab, octave, scilab, python, etc) con un ambiente interactivo para hacer cómputo científico

Python es un lenguaje de programación:

- Donde el tipo de los datos es dinámico
  - Las variables adquieren un tipo cuando entran en contexto
  - Sólo se permiten operaciones bien definidas
  - Hay recolección automática de basura
- Interpretado
  - Compilado a bytecodes en una máquina virtual
  - Los errores se atrapan al tiempo de ejecución
- Con varios estilos de programación
  - Orientado a objetos: Clases, objetos, métodos,...
  - Procedural: Funciones, control de flujo,...
  - Funcional: Expresiones, composiciones,...

Python es un interprete:

- Un programa para parsear (analizar sintácticamente) código en python
- Gratuito y software libre
- Una enorme comunidad de desarrolladores
- Portable entre linux, windows, mac,...
- Tiene una extensa librería estándar: interacción con OS, matemáticas, profiling, debugging, www/html,...
- Puede importar módulos en python puro, y "extensiones" escritas en lenguajes compilados
- Consecuentemente hay un enorme conjunto de módulos externos: cómputo cientifico, visualización, bases de datos, servidores web,...

Ventajas y desventajas de python

- Ventajas
  - Lenguaje muy expresivo. Tipos incorporados: números, cadenas, listas, tuplas, diccionarios, conjuntos
  - Realmente orientado a objetos
  - Fácil de aprender.
  - Rápido ciclo de desarrollo
- Desventajas
  - No es un sólo paquete unificado
  - Puede ser significativamente más lento que un lenguaje compilado

Usos de python

- Lenguaje de programación Los paquetes, módulos, clases, etc, son escalables
- Lenguaje pegamento Permite conectar susbsistemas disconexos: LAPACK, excel, SQL, etc.
- Lenguaje tipo script Comparable con perl, shell, etc.
- Ambiente de exploración Acceso a datos, simulaciones, etc.
- Calculadora Programable, con funciones especiales, etc
- Ambiente de desarrollo En el medio académico es más apropiado desarrollar software que comprar licencias
- Lenguaje de enseñanza Este curso!

Tipos de datos

- tipos de datos básicos: booleanos, números, cadenas, tuplas
- tipos de datos compuestos: listas, diccionarios, queues, conjuntos
- arreglos multidimensionales (numpy)
- archivos
- lenguaje python: funciones, clases, instancias, modulos, excepciones, iteradores, generadores

Booleanos

- type: bool

```
a = (3>2) # True

not True
True and True
True and False
```

Números

- type: int, long, float, complex

```
1
2 + 3
2**64
2.0 + 3.0
(2.0 + 1.0j) * (2 - 1j)
1 / 2
from __future__ import division
1 / 2
```

Cadenas

- types: str,unicode

```
"comillas dobles"
'comillas simples'
"Se pueden poner 'comillas' dentro de las comillas"
"""
Cadenas de varios renglones
"""
'''Caracteres de escape:
nuevo renglo \n,
tab \t,
backslash \\,'''
r"comandos de latex"
```

Tuplas

- Secuencia heterogenea e inmutable
- type: tuple

```
x = (1,"a")
y = (1.0,)
z = ("a",("b",2),7)
print x + y
print x[0], len(x)
x[0] = 2 # Error
```

Listas

- Secuencia heterogenea y mutable
- type: list

```
x = [0, 1, 2, 3]
print x[0]
x[1] = 7
x[-1]
x[1:3]
x[1:-1]
y = ["a",10]
print x + y
x.sort()
```

Diccionarios

- Mapeo de claves a valores
- Las claves deben ser inmutables
- type: dict

```
d = { "a": 1, "b": 2 }
d["a"]
d["c"] = 3
d = dict(a=1, b=2)
d = {}
d[(1,2)] = [8, 9]
print d.keys(), d.values()
print len(d), d.items()
```

Declaraciones y control de flujo

- import trabajo con módulos
- = asignación
- print output
- if declaración condicional
- while, for iteraciones
- function
- classes
- iterators, generators, generator expressions
- raise, try, except excepciones

Trabajo con módulos

- Cada archivo *.py puede ser un módulo
- Archivo `polynomio.py`

```
p0 = 1

def suma(p,q):
    return p + q
...
```

- `import polynomio`
- `from polinomio import p0, suma`
- `from polinomio import *` Esta opción es frágil
- Declaración ejecutable en el módulo

```
if __name__ == "__main__":
   run_test()
```

Asignación

- Para darle un nombre a un objeto se usa el signo '='

```
a = 4
```

- Nota: objetos mutables

```
a = [1, 2]
b = a
b[1] = 8
print a
```

- No se usa para comparar objetos: `5 == 1`
- Sólo se usa al nivel de declaración

```
a = 1
if a = 5:
   print "five"
```

Asignación múltiple

- Manejo de listas

```
a, b = 1, 2
a, b = b, a
c = [1, 2, 3]
u, v, w = c
[(a , b), c] = [(1, 2), 3]
```

Input/Output

- `print "some output"`
- formateo de strings

```
print "a %s number %d" % ("big", 2**64)
```

- output a un archivo

```
file = open("results.dat", "w")
print >> file, "a + b =" 3+4
file.close()
```

- input de un archivo

```
file = open("results.dat", "r")
print file.readline()
print file.readlines([-2:])
file.close()
```

Input/Output mediante pickle

- Para salvar objetos de python a un archivo

```
x = lispace(0.0,1.0,100)
f = open("miarchivo.txt","w")
pickle.dump(x,f)
f.close()
```

- Para leer objetos de python de un archivo

```
f = open("miarchivo.txt","r")
x = pickle.load(f)
f.close()
```

Declaraciones condicionales

- keywords: `if`, `else`, `elif`

```
if a > 0:
   print "pos"
elif a < 0:
   print "neg"
else:
   print "zero"
```

Iteraciones básicas

- keywords: `while`, `for`, `break`, `continue`, `else`

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

for i in range(10):
    print 1
```

Iteraciones sobre listas, tuplas, diccionarios, etc

```
a = ["uno", "dos", "tres"]
for palabra in a:
    print palabra

for (palabra, i) in enumerate(a):
    print i, palabra

d = { "uno": 1, "dos": 2, "siete": 7 }
for palabra in sorted(d.keys()):
    print palabra, d[palabra]
for palabra, i in d.items():
    print palabra, i
```

Funciones

- keyword: `def`

```
def add_posssitive(x, y):
    if y > 0:
        z = x + y
    else:
        z = x
    return z
```

- Las funciones son objetos

```
flista = [ addd_positive, add_negative ]
```

- Las funciones se pueden anidar

```
def adder(x):
    def add(y):
        return x + y
    return add
```

Classes

```
class poly:
   def __init__(self, coeff):
      self.coeff = coefff
   def degree(self):
      return len(self.coeff)
   def add(self, other):
      cs = [a + b for (a, b) in
            zip(self.coef, other.coeff)]
      return poly(cs)

a = poly([0, 1, 0])
b = poly([1, 1, 1])
c = a.add(b)
print c.coeff
```

Excepciones

- keywords: `raise`, `try`, `except`, `finally`

```
a = [1, 2, 3]
try:
   print a[7]
except KeyError, e:
   print e

try:
   file = open("data.txt", "r")
except IOError:
   print"Could not open data.txt."
```

Iteradores

- Cualquier objeto con métodos `__iter__` y `next`
- Excepción `StopIteration` si no hay más elementos

```
class squares:
    def __init__(self, data):
        self.data = data
        self.index = 0
    def __iter__(self): return self
    def next():
        if self.index < len(self.data):
            x = self.data[self.index]
            self.index += 1
            return x*x
        else:
            raise StopIteration
```

Generadores

- Crear iteradores usando funciones
- keyword: `yield`

```
def squares(data):
    for x in data:
        yield x*x

for i in squares([0,1,2,3,4]):
    print i
```

Interpretación de listas y generación de expresiones

- Una manera compacta de crear listas

```
xs = [-2, -1, 0, 1, 2, 3, 4]
ys = [x*x for x in xs]
```

- Los generadores de expresiones crean iteradores

```
ys = [x*x for x in xs]
for y in ys:
    print y
```

- Se pueden usar en muchos sitios en lugar de una lista

```
sum(x * x for x in xs)
dict((x, x*x) for x in xs)
set(x*x for x in xs)
```

Programación procedural en python

- Definición de funciones y ejecución

```
def factorial(n):
    # nota: los bloques se controlan con identacion
    if type(n) != type(0):
        raise TypeError, "Argumento debe ser entero"
    if n==1:
        return 1
    else:
        return n * factorial(n-1)

x = factorial(5)      # asigna 120 a x
y = factorial(3.14)   # mensaje de error
```

Programación procedural en python

```
def f(x, y=3, z=10):
    return x + y + z

w = f(5)                  # w =  5 + 3 + 10
w = f(5, 20)              # w =  5 + 20 + 10
w = f(3, 10, -2)          # w =  3 + 10 - 2
w = f(z=8, y=0, x=2)      # w =  2 + 0 + 8

args = (10, 20, 30)
w = f(*args)          # w = f(args[0], args[1], args[2])
```

Programación funcional en python

- Énfasis en evaluación y composición de expresiones
- Útil para aplicación de funciones a listas

```
def gt10(x):
    return x > 10

map(gt10, [1, 20, 3, 18]) # [False, True, False, True]

filter(gt10, [1, 20, 3, 18]) # [20, 18]

sum(filter(lambda x: x>10, [1, 20, 3, 18])) # 38

[x*x for x in [1,20,3,18] if x>10] # [400, 324]
```

Programación orientada a objetos en python

- Se definen nuevos tipos de datos y su comportamiento

```
class Particula:
    # inicializa la clase
    def __init__(self,masa, velocidad):
        # asigna atributos de valor del nuevo objecto
        self.masa = masa
        self.velocidad = velocidad
    # metodo para calcular el momento del objeto
    def momento(self):
        return self.masa * self.velocidad

mi_particula = Particula(3.2,4.1)
mi_particula.momento()   # 13.119999999999999
```

Cómo se ve un programa en python?

- Ejemplos
  - hello_world.py
  - area_circle.py
  - show_lorentz.py
  - etc...

Documentación

- En la red: `http://www.python.org/doc/`
- Dentro del interprete: `help`, `info`, `dir`, `type`, `str`
- Además, existen numerosos libros, tutoriales, sitios web,...

Módulos interesantes

- math, random
- list, collections
- Procesamiento de texto: string, re
- os, sys,, fileinput,subprocess
- networking: mail, html, xml, http, ftp, socket
- Persistencia de datos: pickle, dbm, sqlite3
- Gui: tkinter
- Cómputo científico numpy, scipy, ipython,...
- Visualización pil, pyx, matplotlib, mayavi,...
- Matemáticas sage

Tópicos avanzados

- Interface con otros lenguajes (C, C++, Fortran, Matlab, R, etc)
- Módulos para extender python
- swig, f2py, pyfort, psyco, pyrex, weave, inline, multiprocessing,...
- Gui