Python for Scientific Computing

Lecture 1: The Python Calculator

Albert DeFusco
Center for Simulation and Modeling

September 23, 2013

Section 1

Computer Programming

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

Assembler

```
1   global _mult3
2   sum equ 16
3   section .text
4   _mult3:
5     push ebp
6     mov ebp,esp
7     push esi
8     push edi
9     sub esp, 4
10    mov esi, [ebp+12]
11    mov edi, [ebp+8]
12    mov dword [ebp-sum], 0
13    mov ecx, 3
14  .forloop:
15    mov eax, [edi]
16    imul dword [esi]
17    add edi, 4
18    add esi, 4
19    add [ebp-sum], eax
20    loop .forloop
21    mov eax, [ebp-sum]
22    add esp, 4
23    pop edi
24    pop esi
25    pop ebp
26    ret
```

C source[1]

```
1   int mult3( int *dst, int *src)
2   {
3     int sum = 0, i;
4
5     for (i = 0; i < 3; i++)
6       sum += dst[i] * src[i];
7
8     return sum;
9   }
10
11  int main(void)
12  {
13    int d[3] = { 1, 2, 3};
14    int s[3] = {8, 9, 10 };
15
16    printf("answer is %i\n", mult3(d, s) );
17    return 0;
18  }
```

[1] The first compiler was written by Grace Hopper in 1952 for the A-0 language

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

Python

```python
import numpy
dist = numpy.array([1,2,3])
src  = numpy.array([8,9,10])

print dist.dot(src)
```

Choices

Computer Time

Programmer Time

History of Python

- December 1989
  - Implementation by Guido van Rossum as successor of ABC to provide exception handling
- February 1991
  - Python 0.9.0 had classes with inheritance, exception handling, functions, and the core datatypes
- January 1994
  - Version 1.0 has functional programming features
- October 2000
  - Python 2.0 brings garbage collections
  - Python 2.2 improves Python's types to be purely object oriented
- December 2008
  - Python 3
  - *Reduce feature duplication by removing old ways of doing things*
  - Not backwards compatible with Python 2.x

What's so great about Python?

- Portable
  - Your program will run if the interpreter works and the modules are installed

What's so great about Python?

- Efficient
  - Rich language features built in
  - Simple syntax for ease of reading
  - Focus shifts to "algorithm" and away from implementation

What's so great about Python?

- Flexible
  - Imperative
  - Object-oriented
  - Functional

What's so great about Python?

- Extendible
  - Plenty of easy-to-use modules
    - *If you can imagine it, then someone has probably developed a module for it*
  - Your program can adjust the way the interpreter works

Why use python for science?

- A highly programmable calculator
- Fast proto-typing new algorithms or program design
- Use C or Fortran routines directly
- Many science and mathematics modules already exist

Development Environment

- Frank
  1. Launch Putty
  2. Connect to `login0a.frank.sam.pitt.edu`

Read the documentation

```
> pydoc

> python
>>> help()
```

Disclaimer

Python 2.7 ! = 3.0

`http://wiki.python.org/moin/Python2orPython3`

Python syntax

- Extremely simplified syntax
  - Nearly devoid of special characters
  - Intended to be nearly English
- Dynamic types
  - It is the responsibility of the programmer
  - Still “strongly typed”

Python objects

- All data in Python is represented by objects
- Objects have
  - an identity
  - a type
  - a value
  - a name ("variable")[2]
- Variables are names assigned to `objects`

[2] `http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html#other-languages-have-variables`

Section 2

Hands-on Python

Numerical objects

- Integers
  - a = 1
- Floats
  - a = 1.0
- Complex numbers
  - a = 1.5 + 0.5j
  - a.real and a.imag return each component
- Type casts
  - myFloat = float(myInteger)
  - myInt = int(myFloat)
- Operators
  - Addition +
  - Subtraction −
  - Multiplication *
  - Exponentiation **
  - Division /
  - Modulus %

Mathematical functions

- many functions exist with the `math` module

```
1  import math
2
3  help(math)
4
5  print math.cos(0)
6  print math.pi
```

Logicals

- Boolean type
  - a $= (3 > 4)$
- Comparisons
  - ==
  - !=,<>
  - >
  - <
  - <=
  - >=

Strings

- a = 'single quoted'
- a = "double quoted"
- Triple quotes preserve whitespace and newlines

```
1       a = """triple
2       quoted
3       string"""
```

- "\" is the escape character

String operations

- Typecasting
  - doubleA = float(a)
  - intA = int(a)
- Comparisons return a Boolean
  - A == B
- concatenated = a + b

Formatted strings

- width placeholders

\%s	string
\%d	integer
\%f	float with 6 decimals
\%E	scientific notation
\%\%	the % sign

- modifiers
  - integers after % adjust with width to print
  - decimal points are specified after the width as ".x"
  - use a hyphen after % to left justify
  - printing long strings or large numbers will skew columns

```
1  from math import pi,e
2
3  print "pi is %d and e is %d" % (pi,e)
4  print "pi is %6.4f and e is %6.4f" % (pi,e)
5  print "pi is %f and e is %f" % (pi,e)
6  print "pi is %10.8e and e is %10.8e" % (pi,e)
7  print "pi is %15.8e and e is %15.8e" % (pi,e)
```

Writing a script file

- save the file to `hello.py`

```
1  #/usr/bin/env python
2
3  #This is my comment about the following script
4   print 'Hello, world!'
```

- run the file

```
> module load python/epd-7.2
> python hello.py
Hello, world!
```

Structured blocks

```
1  if (a>b):
2    print 'a is bigger'
3  elif (b>a):
4    print 'big is bigger'
5  else:
6    print 'a must equal b'
```

Structured blocks

```
1  if (a>b):
2    print 'a is bigger'
3  elif (b>a):
4    print 'big is bigger'
5  else:
6    print 'a must equal b'
```

- Why is it indented?

Logical operations

- **not**, **and**, **or**
- be careful with assignments
  - Just use if statements

Structured blocks

```
1   i=0
2   while (i<10):
3       print i
4       i=i+1
```

Structured blocks

```
1  for i in range(4):
2    print i
```

- `for` is not limited to arithmetic

Simple loops

- range(start , stop , step)
  - list of integers from start to stop-1 in step increments
  - End point is always omitted

```
1  for i in range(4):
2    print i
```

Simple Functions

- Simplify your program
- Easier debugging
- Improved elegance

```
1  import math
2  def area(radius):
3    return math.pi*radius**2
```

Simple Functions

- Functions must be defined before being used
- Arguments are passed by *reference to the object*[3]
  - Variables do not have values, objects do
- Functions are first class citizens

```
1
2  def y(x):
3    x=x+1
4    return x**2
5
6  x=5
7  y(x)
8  print x
```

[3] http://me.veekun.com/blog/2012/05/23/python-faq-passing/

Mathematical Exercises

- Print a table of temperature conversion

$$C = \frac{5}{9}(F - 32)$$

- Compute Pi using the Wallis formula

$$\pi = 2 \prod_{i=1}^{\infty} \frac{4i^2}{4i^2 - 1}$$