Introduction to Python

Lecture 0

Dr. Felix P. Muga II
Mathematics Department
School of Science and Engineering
Ateneo de Manila University

Second Semester, SY 2006 – 2007

Outline

Quick Overview

PYTHON

- an object-oriented language
- developed in late the 1980s as a scripting language
- not compiled into machine code
- run by an *interpreter*
- no stand-alone application.

Advantages

- an open-source software
- available for all major operating systems
- can be tested and debugged quickly
- easier to learn

Obtaining Python

Python interpreter can be downloaded from `www.python.org`

it comes with a nice code editor called *Idle*.

on-line books:

How to Think Like a Computer Scientist Learning with Python
by Downey, Elkner and Meyers. April 2002
pp. 288

Dive into Python
by Downey, Elkner and Meyers. April 2002
pp. 327

Outline

Variables

- represent a value of a given type stored in a fixed memory location.
- are *typed dynamically*.

Example of Variables

Notation: (>>> is the Python prompt)

```
>>> c = 5    # c is integer type
>>> print c
5
>>> c = c * 3.0    # Now, c is a float type
>>> print c
15.0
```

Outline

Strings

- a sequence of characters
- enclosed in a single or double quotes
- concatenated with the plus (+) operator
- slicing (:) is used to extract a portion of the string

Example of Strings

```
>>> string1 = 'Press return to exit '

>>> string2 = 'the program'

>>> print string1 + string2   # concatenate
Press return to exit the program

>>> print string1[0:12]   # slicing
Press return
```

String as an Immutable Object

```
>>> s = 'Press return to exit'

>>> s[0] = 'p'
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in -toplevel-
    s[0]='p'
TypeError:  object does not support item assignment
```

Outline

Tuples

- a sequence of *arbitrary objects*
- separated by *commas* and enclosed in *parentheses*

Example of Tuples

```
>>> rec = ('Tamad','Juan',(6,28,1972))
>>> lastName,firstName,birthDate = rec
>>> print firstName
Juan
>>> birthYear = birthDate[2]
>>> print birthYear
1972
>>> name = rec[1] + ' ' + rec[0]
>>> print name
Juan Tamad
```

Outline

Lists

- similar to a tuple but is *mutable*
- its element and length can be changed
- enclosed in *brackets*

Example of Lists

```
>>> a = [2.0, 3.0, 4.0]     # create a list
>>> a.append(5.0)     # append 5.0 to the list
>>> print a
[2.0, 3.0, 4.0, 5.0]
>>> a.index(5.0)     # find position of 5.0
3

>>> a.insert(0,1.0)     # insert 1.0 in position 0
>>> print a
[1.0, 2.0, 3.0, 4.0, 5.0]
>>> print len(a)
5
```

Independent Copy of a Mutable Object

- the assignment b = a does result in new object
- any changes made to a will be reflected in b
- to create an independent copy: c = a[:]

Example

```
>>> a = [2.0, 3.0, 4.0]    # create a list
>>> b = a    # b is an alias of a
>>> c = a[:]
>>> del a[0]
>>> print a
[3.0, 4.0]
>>> print b
[3.0, 4.0]
>>> print c
[2.0, 3.0, 4.0]
```

Matrices

```
>>> A = [[1,2,3],\
 [4,5,6],\
 [7,8,9]]
```

The backslash (\) is Python's continuation character.

```
>>> print A
A = [[1,2,3],[4,5,6],[7,8,9]]
>>> print A[1]
[4,5,6]
>>> print A[0][2]
3
```

Outline

TAble of Arithmetic Operators

$+$	addition
$-$	subtraction
$*$	multiplication
$/$	division
$**$	exponentiation
$\%$	modular division

Example

```
>>> r = 'Hello '
>>> s = 'to you'
>>> a = [4,5,6]
>>> 3*r
Hello Hello Hello
>>> r+s
Hello to you
>>> 2*a
[4, 5, 6, 4, 5, 6]
>>> [1,2,3]+a
[1, 2, 3, 4, 5, 6]
```

Table of Augmented Assignment Operators

a += b	a = a+b
a -= b	a = a-b
a *= b	a = a*b
a /= b	a = a/b
a **= b	a = a**b
a %= b	a = a%b

Table of Comparison Operators

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to

Examples of Comparison Operators

```
>>> a = 10
>>> b = 19.99
>>> c = '20'

>>> print a < b
True

>>> print a == c
False

>>> print (a > b) and (a != c)
False

>>> print (a > b) or (a != c)
True
```

Outline

`if` Construct

`if` *condition*:
block

executes a block of statements (which must be indented) if the condition returns true.

If the condition returns false, the block skipped.

`elif` Construct

The `if` conditional can be followed by any number of `elif` constructs

```
elif condition:
    block
```

which work in the same manner.

`else` Clause

```
else:
    block
```

can be used to define the block of statements which are to be executed if none of the `if–elif` clauses are true.

Example: Function `sign_of_(a)`

```
def sign_of_(a):
  if a < 0.0:
    sign = 'negative'
  elif a > 0.0:
    sign = 'positive'
  else:
    sign = 'zero'
  return sign

>>> sign_of_(1.25)
'positive'
>>> sign_of_(-32.25)
'negative'
>>> sign_of_(0.0)
'zero'
```

Outline

The `while` Construct

`while` *condition*:
block

executes a block of (indented) statements if the condition is *true*.

After execution of the block, the condition is evaluated again.
If it is still *true*, the block is executed again.

This process is continued until the condition becomes *false*.

Example

```
def squares_of_(n):        # squares from 0 to n
    k = 0
    squares = [ ]
    while k <= n:
        squares.append(k**2)
        k += 1
    return squares
```

The `for` Construct

`for` *condition* in *sequence*:
block

Example

```
def squares_of_(n):        # squares from 0 to n
    squares = [ ]
    for k in range(n):
        squares.append(k**2)
    return squares
```

Outline

Types Conversion

`int(a)`	Converts a to an integer
`long(a)`	Converts a to a long integer
`float(a)`	Converts a to floating point
`complex(a)`	Converts to complex $a + 0j$
`complex(a,b)`	Converts to an complex $a + bj$

Example

```
>>> a = 4
>>> b = -5.3
>>> c = '4.0'
>>> print a + b
-1.3
>>> print int(b)
-5
>>> print complex(a,b)
(4-5.3j)
>>> print float(c)
4.0
```

Mathematical Functions

Core Python supports only a few mathematical functions.

`abs(a)`	Absolute value of a
`max(`*sequence*`)`	Largest element of *sequence*
`min(`*sequence*`)`	Smallest element of *sequence*
`round(a,n)`	Round a to n decimal places
`cmp(a,b)`	Returns $\begin{cases} -1 & \text{if } a < b \\ 0 & \text{if } a = b \\ 1 & \text{if } a > b \end{cases}$

The majority of mathematical functions are available in the `math` module.

Reading Input

The intrinsic function for accepting user input is

```
raw_input(prompt)
```

It displays the prompt and then reads a line of input which is converted to a string.

To convert the string into a numerical value use the function

```
eval(string)
```

Example

```
a = raw_input('Input a:  ')
print a, type(a)
b = eval(a)
print b, type(b)
```

```
Input a:  2**123
2**123 <type 'str'>
10633823966279326983230456482242756608 <type 'long'>
```

More on Reading Input

To input a number and assign it to the variable *a* is

```
a = eval(raw_input(string))
```

Printing Output

Output can be displayed with the print statement

```
print object1, object2, ...
```

which converts *object1, object2, . . .* to strings and prints them on the same line, separated by spaces.

The *newline* character '\n' can be used to force a new line.

Example

```
>>> a = 1234.56789
>>> b = [1, 3, 5, 7, 9]
>>> print a,b
1234.56789 [1, 3, 5, 7, 9]
>>> print 'a = ', a, '\n b = ',b
a = 1234.56789
b = [1, 3, 5, 7, 9]
```

More on Printing Output

The *modulo operator* (%) can be used to format a tuple.
The form of the conversion is

```
'%format1 %format2 ...' % tuple
```

w.d	Integer
*w.d*f	Floating point notation
*w.d*e	Exponential notation

where *w* is the width of the field and *d* is thenumber of digits after the decimal point.

Example

```
>>> a = 1234.56789
>>> n = 987
>>> print '%7.2f' % a
1234.57
>>> print 'n = %6d' % n   # Pad with 2 spaces
n =   9876
>>> 'n = %06d' % n   # Pad with 2 zeros
n = 009876
>>> print '%12.4e %6d' % (a,n)
1.2346e+003  9876
```

Functions

The structure of a Python function is

```
def func_name(param1,param2,...):
    statements
    return return_values
```

Example

```
def main():
    print "This program finds the real solution to a quadratic"
    print
    a, b, c = input('Please enter the coefficients (a, b, c): ')
    discriminant = (b * b - 4 * a * c)**(0.5)
    root1 = (-b + discriminant) / (2 * a)
    root2 = (-b - discriminant) / (2 * a)
    print
    print 'The solutions are: %12.8f %12.8f' %(root1, root2)
main()
```

Another Example

```
def main():
    n = input("How many numbers do you have? ")
    sum = 0.0
    for i in range(n):
    x = input('Enter a number » ')
    sum = sum + x
    print '\n The average of the numbers is %4.2f' %(sum/n)
main()
```

Example

```
def factorial(n):
    if n == 0:
        return 1
    else:
        k = 1
        fact = 1
        while k <= n:
            fact = fact*k
            k += 1
        return fact
```

Module

- A module is simply a file where the functions reside.
- The name of the module is the name of the file.
- A module can be loaded into a program by the statement

```
from module_name import *
```

```
from factorial import *
def permutation(n,k):
    return factorial(n)/factorial(n-k)
```

Exercises

1. Show the output from the following fragments:

   (a)
   ```
   for i in range(10):
       print i * i
   ```

   (b)
   ```
   for d in [3,1,4,1,5]:
       print d,
   ```

   (c)
   ```
   for i in range(4):
       print "Hello"
   ```

   (c)
   ```
   for i in range(7):
       i,2**i
   ```

Exercises

2. Write a program that approximates the value of π by summing the terms of this series:

$$\frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \cdots$$

The program should prompt the user for *n*, the number of terms to sum, and then output the sum of the first *n* terms of this series. Have your program subtract the approximation from the value of the `math.pi` to see how accurate it is.

3. A Fibonacci sequence is a sequence of numbers where each successive number is the sum of the previous two. The classic Fibonacci sequence begins: $1, 1, 2, 3, 5, 8, 13, \ldots$. Write a program that computes the *n*th Fibonacci number where *n* is the value input by the user. For example, if $n = 6$ then the result is 8.

Exercises

2. Write a program that approximates the value of π by summing the terms of this series:

$$\frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \cdots$$

The program should prompt the user for n, the number of terms to sum, and then output the sum of the first n terms of this series. Have your program subtract the approximation from the value of the `math.pi` to see how accurate it is.

3. A Fibonacci sequence is a sequence of numbers where each successive number is the sum of the previous two. The classic Fibonacci sequence begins: $1, 1, 2, 3, 5, 8, 13, \ldots$. Write a program that computes the nth Fibonacci number where n is the value input by the user. For example, if $n = 6$ then the result is 8.

Exercises

4. Numerologists claim to be able to determine a person's character traits based on the 'numeric value' of a name. The value of a name is determined by summing up the values of the letters of the name where 'a' is 1, 'b' is 2, 'c' is 3 etc., up to 'z' being 26. For example, the name "Felix" would have the value $6 + 5 + 12 + 9 + 24 = 56$. Write a program that calculates the numeric value of a single name provided as input.

Exercises

5. A Caesar cipher is a simple substitution cipher based on the idea of shifting each letter of the plaintext message a fixed number (called the key) of positions in the alphabet.
   For example, if the key value is 2, the word “SOURPUSS” would be encoded as “UQWTRWUU.” The original message can be recovered by “reencoding” it using the negative of the key.
   Write a program that can encode and decode Caesar ciphers. The input to the program will be string of plaintext and the value of the key. The output will be and encoded message where each character in the original message is replaced by shifting it *key* characters in the English alphabet. For example, if `A` is a character in the string and 3 is the amount to shift, then the character that replaces `A` is `D`.