CSCI 1040

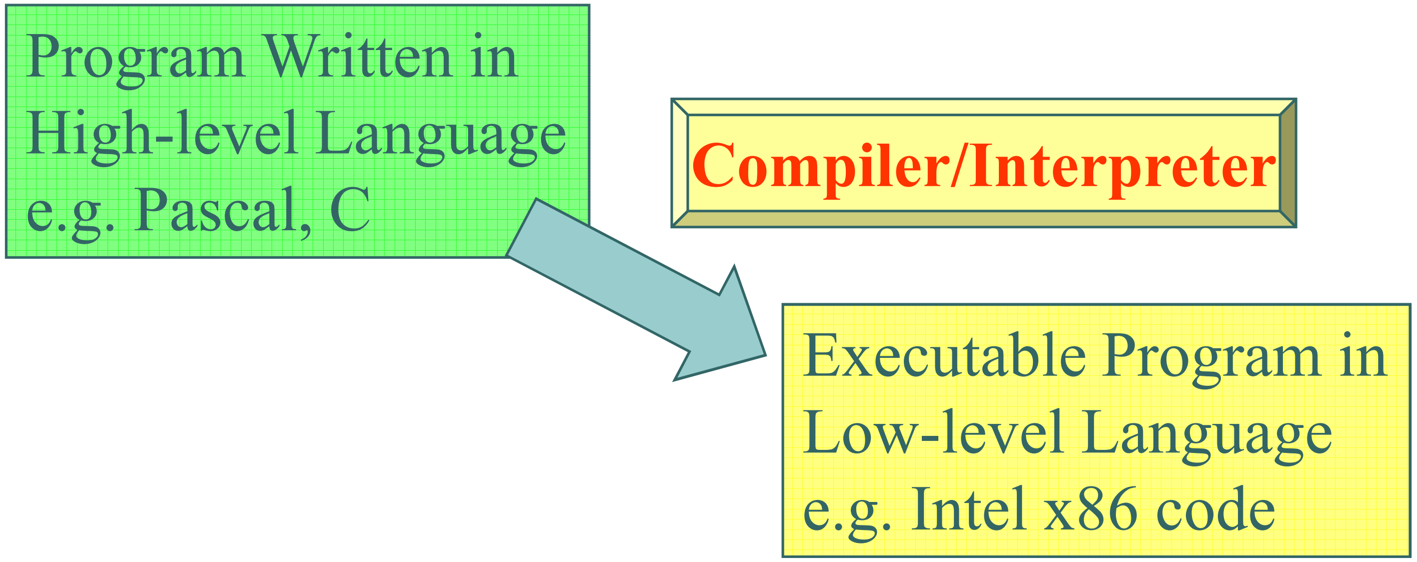
HANDS-ON INTRODUCTION TO PYTHON

Outline

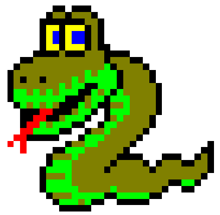
- Introduction to Python
- Python: a modern scripting language
- Python Syntax: Variables and Operations
- Simple control flow

Compiler / Interpreter

- Compiler: does the translation task for us by translating the whole program file into executable
- Interpreter only translate the current instruction of the high level program


Program Written in High-level Language e.g. Pascal, C
Compiler/Interpreter
Executable Program in Low-level Language e.g. Intel x86 code

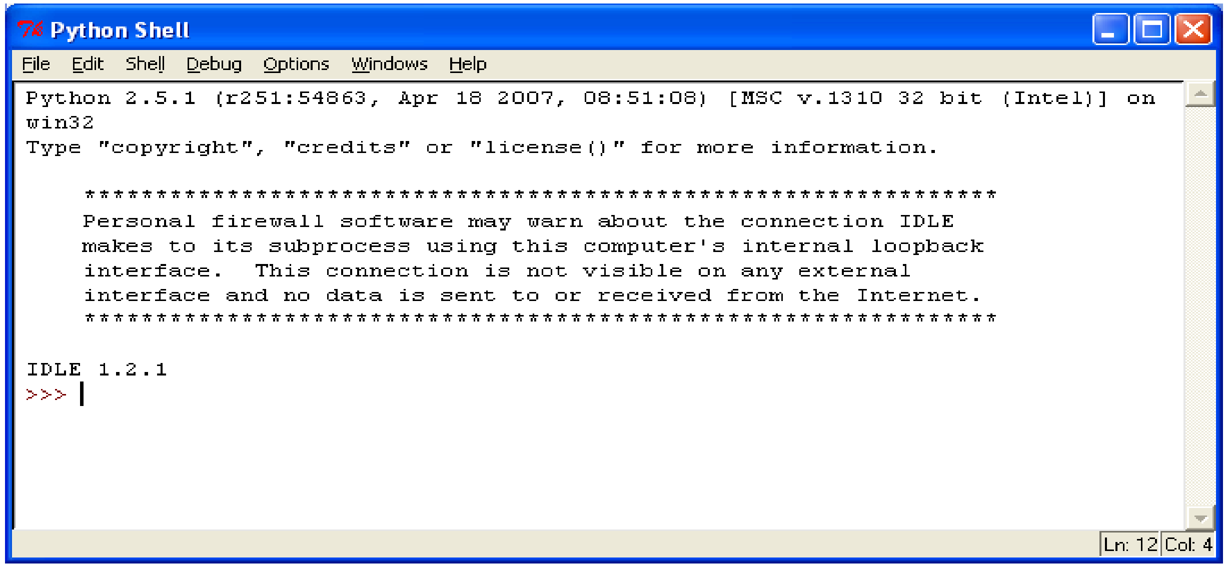
Python



- Object oriented high level language
- designed by Guido van Rossum in early 90’s
- Can work in both compiled/interpreted environment
- Available in many platforms – Windows, Mac, Linux etc
- Ideal for scripting, rapid prototyping applications
- Current version 3.3/2.7 – (3.3 backward incompatible with versions 2.7 or before)
- Syntax revamped with features enhancement from 3.0 on
- www.python.org for more info.

Programming Tools

- Python package
  - includes a graphical development environment (IDLE)
  - For *most* platforms (computer systems)
  - *Free* for download



```
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2.1
>>> 
```

The First Python picture

- Bring up your Python environment

```
>>>1+2
```

- press return gives

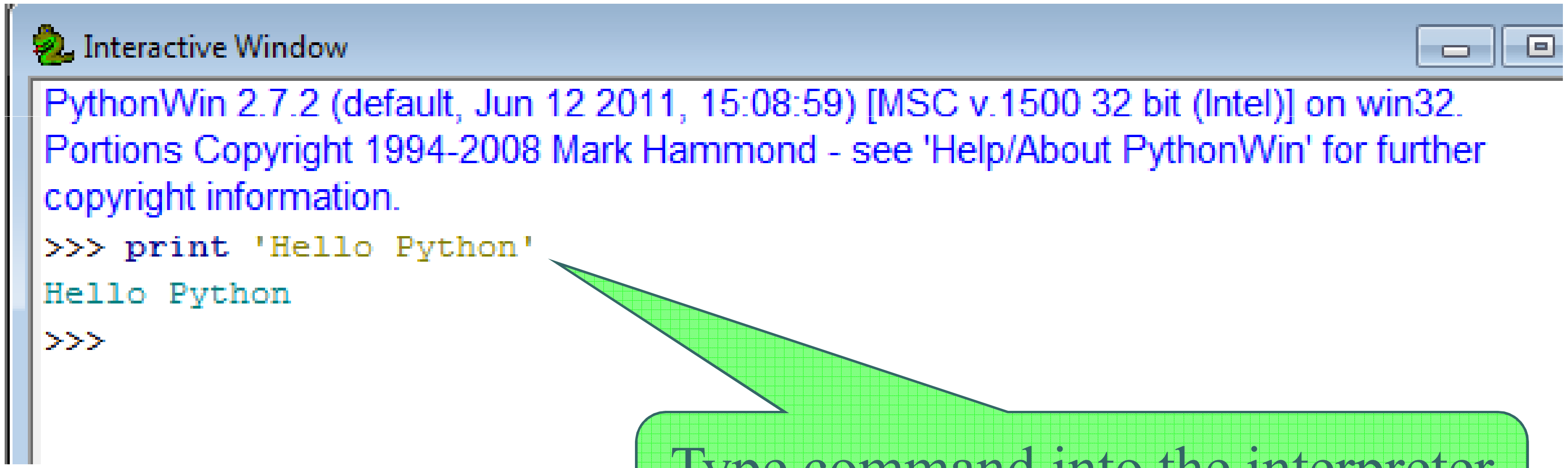
```
3
>>>7/-3
-3
```

Note : -2.333333333 for 3.0 or later

- Python can be used as a calculator!
- we recommend Python 2.7 for it has broader supports e.g. MS Windows extensions, 3rd party libraries ..

The First Python picture

- Bring up your Python environment

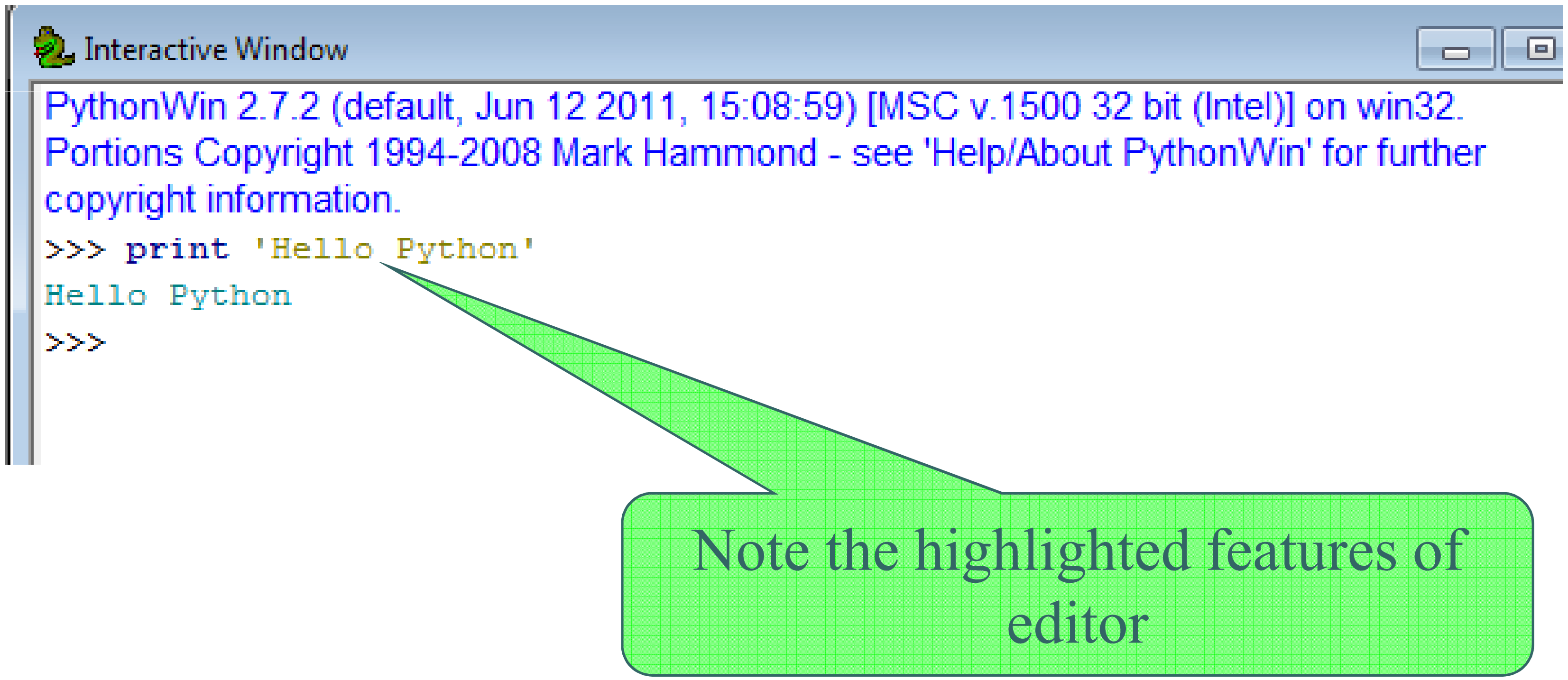


```
PythonWin 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further
copyright information.
>>> print 'Hello Python'
Hello Python
>>>
```

Type command into the interpreter window

The First Python picture

- Print send the message to the console output => interpreted execution



```
PythonWin 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further
copyright information.
>>> print 'Hello Python'
Hello Python
>>>
```

Note the highlighted features of editor

The First Python Program

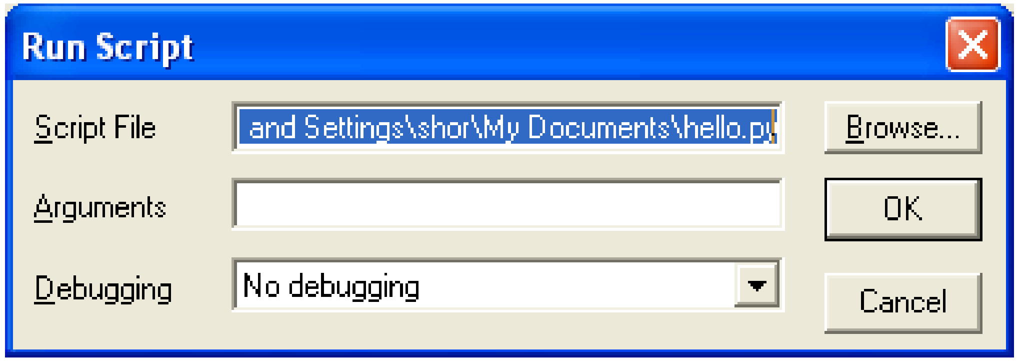
- Select *New/Python* script command
- Type the following statement into the window and save as "hello.py":

```
print 'Hello Python'
```

- `print('Hello Python')` **# for 3.0 or later**

most important difference between 3.X and 2.X

- Select '*Run*' and click *browse*, pick the file just saved and click *OK*


Run Script

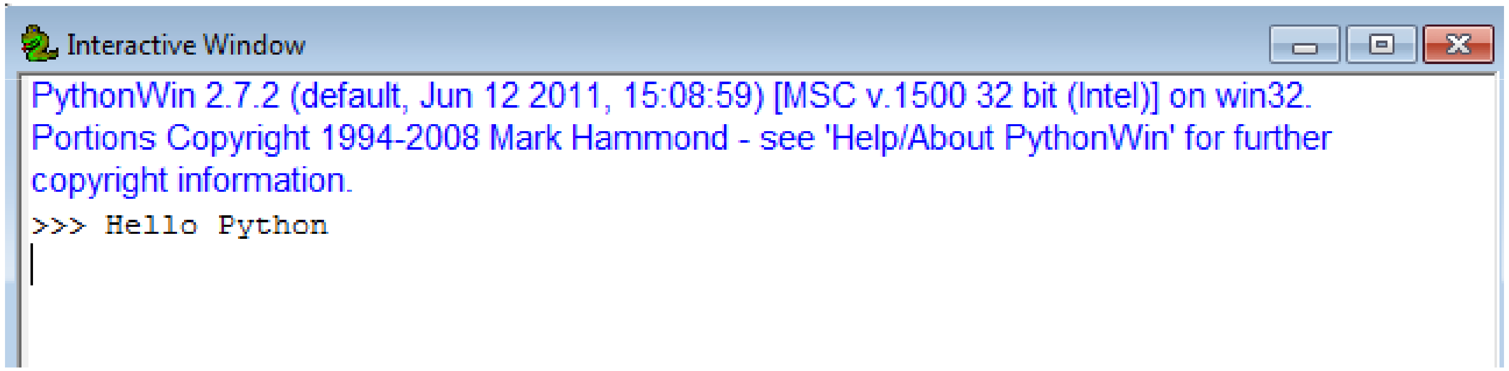
Script File: and Settings\shor\My Documents\hello.py

Arguments:

Debugging: No debugging

The First Python Program

- Watch the output in the console window=> compiled execution



```
PythonWin 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further
copyright information.
>>> Hello Python
```

- The python script file can also be executed in console by command "**`python hello.py`**"

Summary

- Python can run in both interpreted/compiled environment
- *print* statement can dump whatever you like to the console output
- You can test your commands first in interpreted environment and then save them to a python *script file* for complete execution later on
- Magic of scripting language – productivity increase of programmer as we will see later

Data types

- print can operate on many data types
- Literal constants – 5, 2.345, 9.8, ..
- Numbers: integers, floating points, even complex numbers
- Strings i.e. message like ‘Hello summer’
  - Delimited using either single quote ‘, double quote “ e.g.
    - ‘I said “conversation is good!”’
    - “I like my brother’s toy”

Variables

- A piece of scratchpad storage which you can change its content anytime during program execution
- You have to give a name to them first so that you can use it – **identifier**
  - The first character of the identifier must be a letter of the alphabet (upper or lowercase) or an underscore ('_').
  - rest of the name can consist of letters, underscores or digits (0-9).
  - Identifier names are case-sensitive. For example, myname and myName are **not** the same.
  - E.g. myHomework, python1, _interpreter

The Second Python Program

```
v = 5
print v
v = v * 3 - 12
print v
v = 'my new status'
print v
v = v + ' is okay!'
print v
```

Output

```
>>>
>>> 5
3
my new status
my new status is okay!
```

Assignment statement

```
v = 5
```

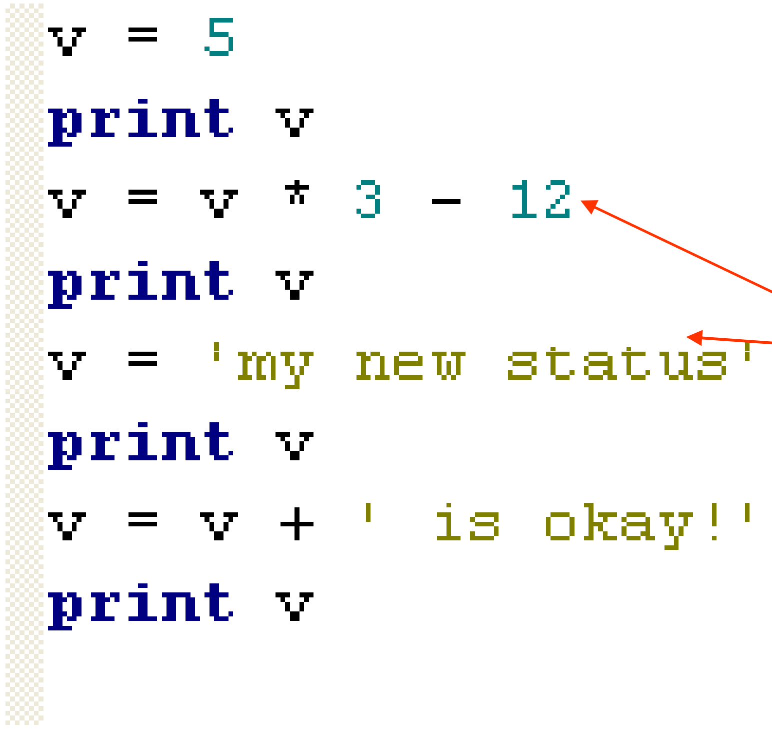
- store a value in a variable.

- The variable name appears on the Left Hand Side.
- The value to be stored appears on the Right Hand Side.

Common operators

Operator	Name	Examples
+	Plus	3 + 5 gives 8. 'a' + 'b' gives 'ab'.
-	Minus	-5.2 gives a negative number. 50 - 24 gives 26.
*	Multiply	2 * 3 gives 6. 'la' * 3 gives 'lalala'.
**	Power	3 ** 4 gives 81 (i.e. 3 * 3 * 3 * 3)
/	Divide	4/3 gives 1 or 1.3333333 (2.x and 3.x results). 4.0/3 or 4/3.0 gives 1.3333333333333333
//	Floor Division	4 // 3.0 gives 1.0
%	Modulo	8%3 gives 2. -25.5%2.25 gives 1.5 .

The Second Python Program



```
v = 5
print v
v = v * 3 - 12
print v
v = 'my new status'
print v
v = v + ' is okay!'
print v
```

Note : Python allows you to assign totally different data to the same variable – called *dynamic typing*

The Second Python Program

```
v = 5
print v
v = v * 3 - 12
print v
v = 'my new status'
print v
v = v + ' is okay!'
print v
```

Comparing the result, Python perform concatenation (glue together)

Output

```
>>>
>>> 5
3
my new status
my new status is okay!
```

Control Flow

- Used to control your program flow
- *if, while* & *for* statement

- In order to construct flow control statements, we have to construct a condition first

- A condition is a Boolean value to be determined (true or false) e.g. whether time is now 4'o clock or not?

Operator	Description	Examples
<	Less Than	5 < 3 gives 0 (i.e. False) and 3 < 5 gives 1 (i.e. True). Comparisons can be chained arbitrarily: 3 < 5 < 7 gives True.
>	Greater Than	5 < 3 returns True. If both operands are numbers, they are first converted to a common type. Otherwise, it always returns False.
<=	Less Than or Equal To	x = 3; y = 6; x <= y returns True.
>=	Greater Than or Equal To	x = 4; y = 3; x >= 3 returns True.
==	Equal To	x = 2; y = 2; x == y returns True. x = 'str'; y = 'stR'; x == y returns False. x = 'str'; y = 'str'; x == y returns True.
!=	Not Equal To	x = 2; y = 3; x != y returns True.
not	Boolean NOT	x = True; not y returns False.
and	Boolean AND	x = False; y = True; x and y returns False since x is False. In this case, Python will not evaluate y since it knows that the value of the expression will has to be false (since x is False). This is called short-circuit evaluation.
or	Boolean OR	x = True; y = False; x or y returns True. Short-circuit evaluation applies here as well.

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

Output

```
1
1
2
3
5
8
END
```

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

Multiple assignments
a = 0
b = 1

Assign at the same time

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print 'END'
```

Keep doing statements in red box as long as condition is true

Note the ':' a must

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print 'END'
```

Statements defined by indentation

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

First time : b = 1

Enter loop

print b

Output

```
1
```

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print 'END'
```

Using old values of a & b

First time : b = 1

a update to 1
b update to 0+1

Note: the two assignment update at the same time

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print 'END'
```

Program flow loop to while and check condition again

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

2nd time : b = 1
Enter loop
print b

Output

```
1
1
```

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print 'END'
```

2nd time : b = 1

a update to 1
b update to 1+1

Using old values of a & b

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

Program flow loop to while and check condition again

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

3rd time : b = 2
Enter loop
print b

Output

```
1
1
2
```

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

3rd time : b = 2

a update to 2

b update to 2+1

Using old values of a & b

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

4th time : b = 3
Enter loop
print b

Output

```
1
1
2
3
```

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

4th time : b = 3

a update to 3
b update to 3+2

Using old values of a & b

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

5th time : b = 5

Enter loop

print b

Output

```
1
1
2
3
5
```

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

5th time : b = 5

a update to 5
b update to 5+3

Using old values of a & b

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

6th time : b = 8

Enter loop

print b

Output

```
1
1
2
3
5
8
```

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

5th time : b = 8

a update to 8
b update to 8+3
(now 11)

The Third Python Program

```
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b
print ‘END’
```

Program flow exit loop as b< 10 is now false

Output

```
1
1
2
3
5
8
END
```

Nested While

- A while loop capped inside another

```
x = 0
while x < 3:
    y = 0
    while y < 3:
        print x, y
        y = y + 1
    x = x + 1
```

- What are the output?

If statement

- Selection statement for 1 or more choices

```
if choice == 1:
    print 'You choose Cola'
    out = 'coke'
elif choice == 2:
    print 'You choose Lemon tea'
    out = 'Lemon Tea'
elif choice == 3:
    print 'You choose Orange juice'
    out = 'Orange Juice'
else:
    print 'Invalid choice!'
    print 'choose again'
```

Only one outcome is selected

If statement

- Selection statement for 1 or more choices

```
if choice == 1:
    print 'You choose Cola'
    out = 'coke'
elif choice == 2:
    print 'You choose Lemon tea'
    out = 'Lemon Tea'
elif choice == 3:
    print 'You choose Orange juice'
    out = 'Orange Juice'
else:
    print 'Invalid choice!'
    print 'choose again'
```

Note the : after each Condition

Also statements indented are executed for each choice

If statement

- All alternatives i.e. elif & else are optional.

```
if LifeOfSun == 1000000000000000:
    print ‘Game over'
print ‘End‘
```

- Or

```
if OilRemain == 0:
    print ‘Game over‘
else
     print ‘Life is hard’
print ‘End‘
```

If statement

- Other possibilities

```
if LifeOfSun == 1000000000000000:
    print ‘Game over'
elif OilRemain == 0:
    print ‘Game over‘
print ‘End‘
```

Lists

- Another form of *array*, but can have different data types within
- Index start from 0
- Easy manipulation

```
>>> a = ['cuhk', 'us', 2010, 'maya']
>>> a[0]
'cuhk`
>>> a[1:-1]
['us', 2010]
>>> a[:2] + [2012, 'gogogo']
['cuhk', 'us', 2012, 'gogogo']
>>> 2*a[:3]
['cuhk', 'us', 2010, 'cuhk', 'us', 2010]
```

Counting 1 place from right to left

Slice 2 places

Lists

-

```
>>> a = ['cuhk', 'us', 2010, 'maya']
>>> a[2] = a[2] + 2
>>> a
['cuhk', 'us', 2012, 'maya']
```

- Built-in function len()

```
>>> len(a)
4
```

for statement

- Iterate over items in a sequence
- Best suit array or list

```
a = ['cat', 'window', 'defenestrate']
for x in a:
  print x, len(x)
```

Output:

```
cat 3
window 6
defenestrate 12
```

range() Function

- In for statement, to iterate over number sequence

```
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in (range(len(a))):
  print i, a[i]

Output
0 Mary
1 had
2 a
3 little
4 lamb
```

Further development

- Python is easy to use, and thus many useful tools are developed
- To use those tools in your own program is easy

  import yourModule
- *yourModule* is the tool you want to use
- Many tools are available e.g. multimedia, image processing, numerical library, etc.
- Check out our lab to have a taste

Differences between 3.X & 2.X

- print is a function, not command
- Better Unicode support – all text strings being Unicode by default
- exception chaining
- Iterators instead of lists
- syntax for keyword-only arguments
- extended tuple unpacking
- non-local variable declarations

- Details: http://docs.python.org/3/whatsnew/3.0.html

Summary

- Python can work in both compiled & interpreted environment
- Interpreted environment greatly help to enhance the productivity in many applications
- Using print, various operators and flow control, we can construct complicated programs
- Using developed module, we can built useful application in a very short period.