Programming for Business Computing

Introduction

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Outline

- **Computer programming**
- Our first program: arithmetic and **`print`**
- Our second program: variable declaration and **`input`**
- Debugging

Computer programming

- What are **computer programs**?
  - The elements working in computers.
  - Also known as **software**.
  - A structured combination of data and instructions used to operate a computer to produce a specific result.
- Strength: High-speed computing, large memory, etc.
- Weakness: People (programmers) need to tell them what to do.
- How may a programmer tell a computer what to do?
  - Programmers use "**programming languages**" to write codes line by line and construct "computer programs".
- **Running a program** means executing the instructions line by line and (hopefully) achieve the programmer's goal.

Programming languages

- People and computers talk in programming languages.
- A programming language may be a **machine language**, an **assembly language**, or a **high-level language** (or something else).
  - Machine and assembly languages: Control the hardware directly, but hard to read and program.
  - High-level languages: Easy to read and program, but need a "translator."
- Most application software are developed in **high-level languages**.
  - The language we study in this course, Python, is a high-level language.
  - Some others: C, C++, Basic, Quick Basic, Visual Basic, Fortran, COBOL, Pascal, Perl, Java, C#, PHP, Matlab, Objective C, R, etc.

Python

- Python was invented by Guido van Rossum around 1996:
  - Was just something to do during the Christmas week.
  - The latest version (in August, 2017) is **3.6.2**.
- Python is very good for beginners.
  - It is simple.
  - It is easy to start.
  - It is powerful.

Interpreting a program

- An **interpreter** translates programs into assembly programs.
  - For other high-level programs, a **compiler** is used.
  - Python uses an interpreter.
- An interpreter interpret a program line by line.
- We may write Python in the **interactive mode**.
  - Input one line of program, then see the result.
  - Input the next line, then see the next result.
  - The statements should be entered after the **prompt**.

```
>>> 3 + 6
9
>>> 4 - 2
2
>>> a = 100
>>> b = 50
>>> c = a - b
>>> print(c)
50
```

Interpreting a program

- We may also write Python in the **script mode**.
  - Write several lines in a file (with the extension file name .py), and then interpret all the lines one by one at a single execution.
- A programming language using an interpreter is also called a **scripting language**.
  - E.g., R.

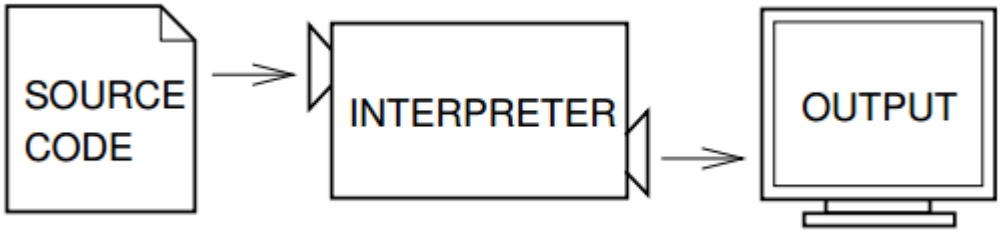
```
for i in xrange(0, bingo):
    a = random.randint(start, end) - 1
    temp = seqNo[a]
    seqNo[i] = temp

seqNoSorted = sorted(seqNo[0:bingo])
#print(seqNoSorted)

for i in xrange(0, bingo):
    print(seqNoSorted[i])
```

How to run Python

- To taste Python online:
  - https://www.python.org/ or other similar websites.
- To get the Python interpreter:
  - Go to https://www.python.org/downloads/, download, double click, and then click and then click… and then you are done.
- To try the interactive mode:
  - Open your console (the command line environment) and type **`python`** to initiate the interactive mode.
  - You may need to set up your "PATH" variables.

How to run Python

- To run Python on IDLE (Python GUI):
  - Click its icon and then play with the prompt.
  - Do "File → New File" to write and execute a script.
- To write Python on an **editor** and interpret a script with the interpreter:
  - Open a good text editor (e.g., Notepad++), write a script, save it (.py).
  - Open the **console**, locate your script file (.py), interpret it with the instruction **`python`**, and see the result.



(Figure 1.1, *Think Python*)

Outline

- Computer programming
- **Our first program: arithmetic and `print`**
- Our second program: variable declaration and **`input`**
- Debugging

Our first program

- As in most introductory computer programming courses, let's start from the "Hello World" example:

```
print("Hello World!")
```

- Let's try this in the interactive mode!

```
>>> print("Hello World!")
Hello World!
```

Our first program

```
print("Hello World!")
```

- The program has only one **statement**.
- In this statement, there is one single **operation**.
  - `print` is a **function**: Print out whatever after it on the screen.
  - `"Hello World!"` is an **operand**: A message to be printed out.
- In Python, each statement must be put in **a single line** in your editor.

Our first program

- We of course may print out other messages.

```
print("I love programming!")
```

- It does not matter whether to use single or double quotation marks here.
  - As long as they are paired.

Printing out more complicated messages

- What if we want to print out

長跪讀素書，書中竟何如。
上言加餐食，下言長相憶。

```
>>> print("長跪讀素書，書中竟何如。上言加餐食，下言長相憶。")
長跪讀素書，書中竟何如。上言加餐食，下言長相憶。
>>> print("長跪讀素書，書中竟何如。
上言加餐食，下言長相憶。")

SyntaxError: EOL while scanning string literal
```

- Something is wrong when we want to **create a new line**!

A newline character

- Inside a computer, everything is **encoded**.
  - In particular, each character has a corresponding number representing it.
  - "Creating a new line" actually means "printing out **a newline character**".
- A right way to do it is:

```
print("長跪讀素書，書中竟何如。\n上言加餐食，下言長相憶。")
```

```
>>> print("長跪讀素書，書中竟何如。\n上言加餐食，下言長相憶。")
長跪讀素書，書中竟何如。
上言加餐食，下言長相憶。
```

- That `\n` is the newline character.

Escape sequence

- In Python (and many modern language), the **slash** symbol "\" starts an **escape sequence** (character).
  - An escape sequence represents a "special character" that does not exist on the keyboard.

Escape sequence	Effect	Escape sequence	Effect
`\n`	A new line	`\\`	A slash: \
`\t`	A horizontal tab	`\'`	A single quotation: '
		`\"`	A double quotation: "

The escape sequence \n

- Try it:

```
print("《青青河畔草》：\"長跪讀素書，書中竟何如。\n上言加餐食，下言長相憶。\"")
```

```
print("《青青河畔草》：「長跪讀素書，書中竟何如。\n上言加餐食，下言長相憶。」")
```

```
print('《青青河畔草》：\"長跪讀素書，書中竟何如。\n上言加餐食，下言長相憶。\"')
```

- More details about **string operations** will be discussed later in this semester.

Basic arithmetic

- Computers are good at doing **computation**.
  - All computation starts from simple calculation, i.e., **arithmetic**.
- We may use the operators **+**, **−**, *****, **/**, and **//** to do addition, subtraction, multiplication, floating-point division, and floor division.
- We may use **(** and **)**, i.e., a pair of parentheses, to determine the calculation order.
- We may use the operator ****** to find the square of a number.

```
>>> 3 + 8
11
>>> 4 - 2 * 5
-6
>>> (4 - 2) * 5
10
>>> 3 ** (5 / 2)
15.588457268119896
>>> 3 ** (5 // 2)
9
```

Outline

- Computer programming
- Our first program: arithmetic and **`print`**
- **Our second program: variable declaration and `input`**
- Debugging

input()

- The **print** operator prints out data to the console output.
- A function **input** accepts data **input** (by the user or other programs) from the console input (typically the keyboard).
  - A function is a set of codes that together do a particular task. This will be explained in details later in this semester.
- In order to get input, we need to first prepare a "**container**" for the input data. The thing we need is a **variable**.
- When we use a single variable to receive the data, the syntax is

```
variable = input()
```

- Let's first learn how to **declare variables**.

Variables and data types

- A variable is a container that stores a value.
  - Once we declare a variable, the system allocates a **memory space** for it.
  - A value may then be stored in that space.
- A variable has its **data type**.
  - At this moment, three data types are important: **`int`** (for integer), **`float`** (for fractional numbers), and **`string`** (for strings).
- Three major attributes of a (typical) variable:
  - Type.
  - Name.
  - Value.

Variable declaration

- Before we use a variable, we must first **declare** it.
  - We need to specify its **name**.
  - We need to specify its **data type**, **initial value**, or both.
- Typically in Python we declare a variable with an initial value directly.

```
a = 689
b = 8.7
c = "Hi everyone, "
```

The interpreter will automatically set the type of a variable according to the assigned initial value.

- To see this, put a declared variable into the function **`type()`**.

Variable declaration

- Let's try to see the types of declared variables:

```
a = 689
b = 8.7
c = "Hi everyone, "
print(type(a))
print(type(b))
print(type(c))
```

- A variable may be overwritten:

```
a = 689
a = 8.7
print(type(a))
```

Variable declaration

- Sometimes we have no idea about an initial value.
- In this case, do:

```
a = int()
b = float()
c = ""
```

- Try to print them out to see their initial values!

Our second program (in progress)

- This is our second program (to be completed later):

```
num1 = 4
num2 = 13
print(num1 + num2)
```

- We first declare and initialize two integers.
- We then do

```
print(num1 + num2)
```

- There are two **operations** here:
  - **num1 + num2** is an addition operation. The sum will be **returned** to the program.
  - That returned value is then printed out.
- As a result, **17** is displayed on the screen.

Our second program (in progress)

- What will be displayed on the screen?

```
num1 = 4
num2 = 13

print(num1 - num2)
print(num1 * num2)
print(num1 // num2)
print(num1 / num2)
print(num1 % num2)
print(num1 ** num2)
```

Our second program

- Now we are ready to present our second program:

```
num1 = int()
num2 = int()
num1 = int(input())
num2 = int(input())
print(num1 + num2)
```

- In this example, we allow the user to enter two numbers.
- We declare two variables to receive the inputs.
- We then use the **`input`** function to read input values into the variables.
- We then sum them up and print out the sum.

Our second program

- Alternatively:

```
num1 = int(input())
num2 = int(input())
print(num1 + num2)
```

- The interpreter always stops when it execute the **`input`** function.
- It stops and waits for user input.
- After the user input something, it reads it into the program.

Our second program

- How about this?

```
num1 = input()
num2 = input()
print(num1 + num2)
```

- The **return type** of **`input`** is a string!
- The addition operator **+** will concatenate two strings.
- That is why the **`int`** function is required in the right implementation.

Outline

- Computer programming
- Our first program: arithmetic and **`print`**
- Our second program: variable declaration and **`input`**
- **Debugging**

Syntax errors vs. logic errors

- A **syntax error** occurs when the program does not follow the standard of the programming language.

```
num1 = int()
num2 = int()
num1 = int(inpnt())
num2 = int(input())
print(num1 + num2)
```

  - The interpreter detects syntax errors.

Syntax errors vs. logic errors

- A **logic error** occurs when the program does not run as the programmer expect.

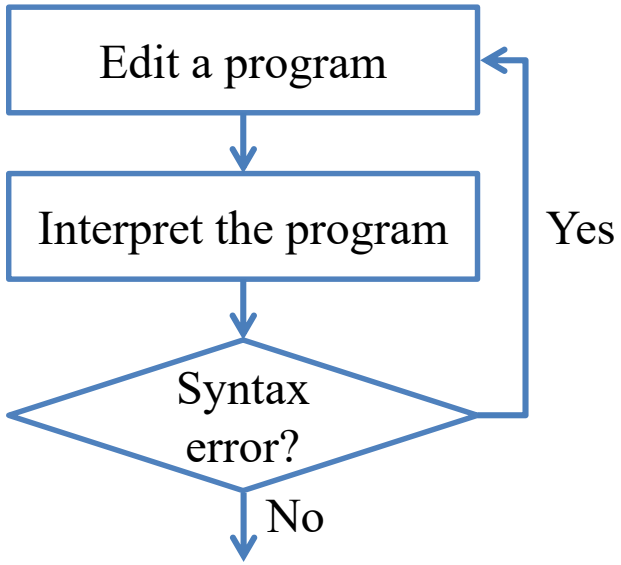
```
num1 = int()
num2 = int()
num1 = int(input())
num2 = int(input())
print(num1 + num1)
```

  - Programmers must detect logic errors by themselves.
  - The process is called **debugging**.

Steps to do computer programming

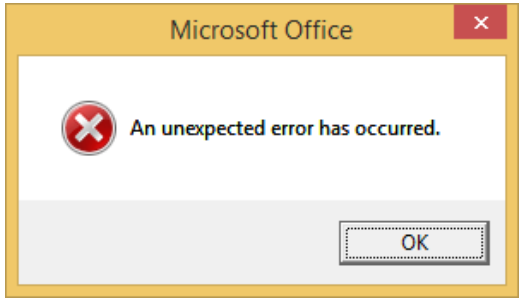
- (The following four pages of slides are modified from the lecture notes by Professor Pangfeng Liu in NTU CSIE.)

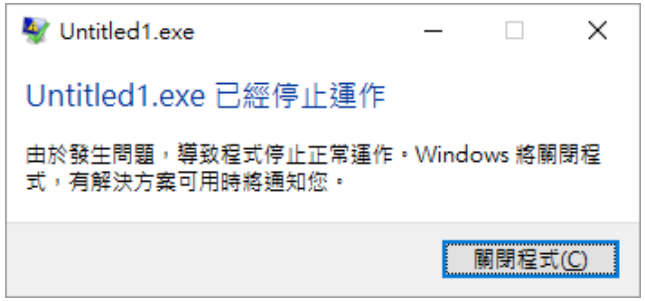
- First, **edit** a program.
- Second, **interpret** the program.
- If there is a **syntax error**, fix it.



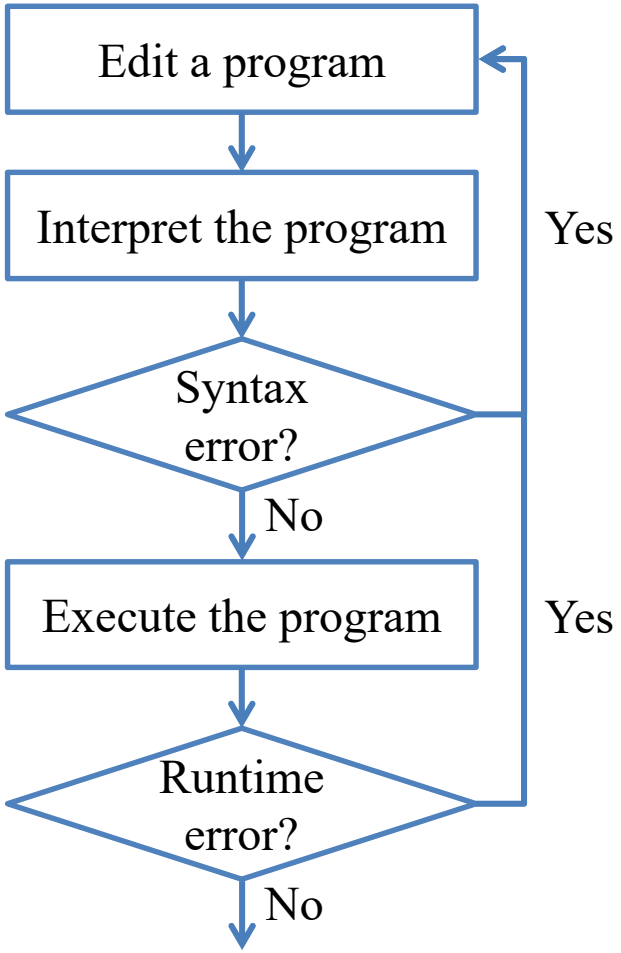
Steps to do computer programming

- Next, **execute** the program.
- Be aware of **runtime errors**:
  - A runtime error is one kind of logic error.
  - When it happens, the program **cannot terminate as we expect**.



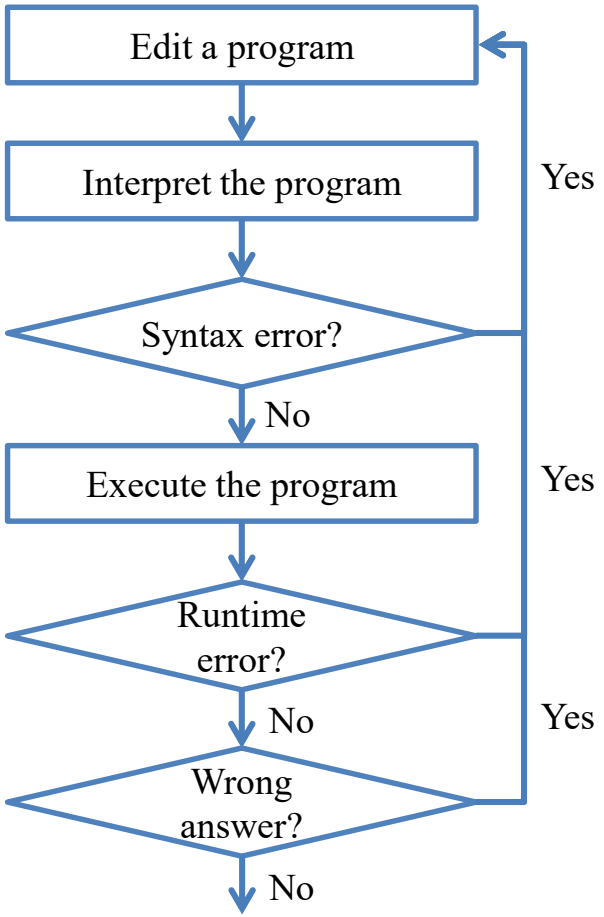


- If there is a runtime error, fix it.



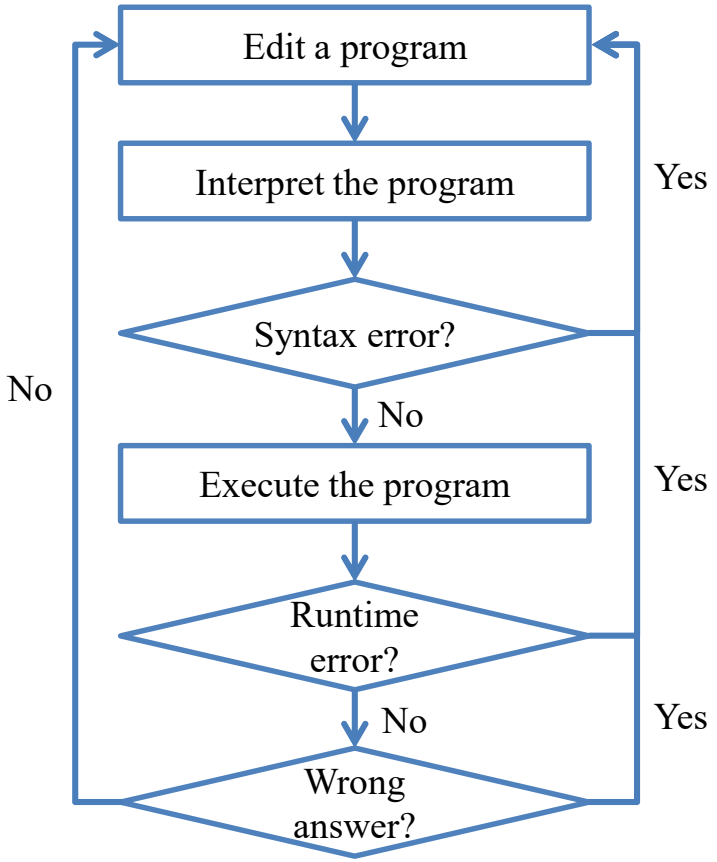
Steps to do computer programming

- Now your program terminates successfully.
- Next, check your answer.
  - You get a **wrong answer** if the outcome is incorrect.
  - Wrong answer is one kind of logic error.
- If there is a wrong answer, fix it.
  - Typically the most time consuming step.
  - **Logic**!



Steps to do computer programming

- Now the answer is correct. What is the **next step**?
- Write your **next program**!

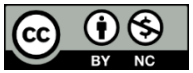
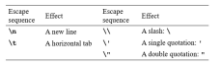
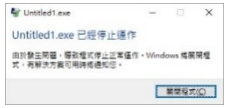


Using Notepad++ to run Python directly

- We may use Notepad++ (or many other editor) to run Python directly.
- To do so:
  - Select "Run" → "Run…"
  - Enter "cmd /k C:/Python36/python "$(FULL_CURRENT_PATH)" & PAUSE & EXIT"
  - Select "Save…" and choose a hotkey combination you like.

- Please replace the path in red by the path in your computer!

版權聲明

序	頁	作品	版權標章	作者 / 來源
1	8			Python Software Foundation, Guido van Rossum https://www.python.org/ 依據著作權法第46、52、65條合理使用 2017/7/24 visited
2	9			WordPress, Green Tea Press, Allen B. Downey, Think Python ver2.0.17,P2, CC BY-NC 3.0, 2012/8/3 http://www.greenteapress.com/thinkpython/thinkpython.pdf 2017/7/24 visited
3	16			台灣大學 孔令傑, CC BY-NC-ND 3.0
4	34			Microsoft Office軟體版權著作 https://products.office.com/zh-tw/home 依據著作權法第46、52、65條合理使用 2017/7/24 visited
5	34			Microsoft Windows軟體版權著作 https://www.microsoft.com/zh-tw 依據著作權法第46、52、65條合理使用 2017/7/24 visited
6	1-38			台灣大學 孔令傑, CC BY-NC-ND 3.0