Python Programming

John Dunnion

School of Computer Science
University College Dublin

Outline

Basic Elements of Python

- A Python program, or Python script, is a sequence of *definitions* and *commands*
- The definitions are evaluated and the commands are executed by the Python interpreter in the Python shell
- A command, or statement, instructs the interpreter to do something

Starting Python

- Start a Python shell
- Double-click on an icon
- Enter `python3` at a terminal window

My first program

```
print('Hello, world.')
```

This causes the interpreter to produce the following output on the screen:

```
Hello, world.
```

Comments

- Any text following the # character is ignored by the Python interpreter
- This is called a comment
- Comments are used to enhance the readability of code for human readers

```
# My first program
print('Hello, world.')
```

This causes the interpreter to produce the following output on the screen:

```
Hello, world.
```

Printing strings

- The `print` command takes a variable number of strings (text enclosed in single quotes) and prints them, separated by a space, in the order in which they appear

```
# My second program
print('Good morning!')
print('Vietnam!')
print('Good morning,', 'Vietnam!')
```

This causes the interpreter to produce the following output on the screen:

```
Good morning!
Vietnam!
Good morning, Vietnam!
```

IDE

- Typing programs into the shell is highly inconvenient
- Most prefer to use an integrated development environment (IDE)
- The IDE provides:
  - A text editor with syntax highlighting, auto-completion and smart indentation
  - A shell with syntax highlighting
  - An integrated debugger

IDE menus

- There are a number of IDEs that you can download. Each IDE provides the user with a number of menus such as
- The file menu
  - Create a new editing window
  - Open a file containing an existing Python program
  - Save the contents of the current editing window into a file (with a `.py` file extension)
- The edit menu
  - Standard text-editing commands
  - Copy, paste, find, . . .
  - Commands for editing Python code
  - Indent a region of code
  - Comment out a region of code
  - . . .

Quotes in strings

- You can use single quotes or double quotes to surround a string
- The same quotes must be used at either end of a particular string

```
'This is a string'
"This is also a string"
```

- If you use double quotes inside a string, you can use single quotes to surround the string

```
'She shouted "Hello" to the crowd.'
```

- Similarly, if you use single quotes inside a string, you can use double quotes to surround the string

```
"She shouted 'Hello' to the crowd."
```

Escape character

- If you need to include a quote inside a string, you can “escape” the quote
- The ’\’ character is the escape character
- When the escape character is encountered in a string, it and the next character(s) are interpreted as a code

```
'This␣is␣a␣\'special\'␣string'

"This␣is␣a␣string␣with␣a␣\"␣character"
```

Escape sequences

- The following are a few of the escape sequences in Python:

Escape Sequence	Output
\t	[horizontal] tab
\n	newline
\'	'
\"	"
\b	backspace
\v	vertical tab
\a	bell/beep
\\	\

Bigger strings

- Sometimes you need to write *very* long strings that span many lines
- For example, instructions to a user, a long menu, a logo,. . .
- You can do this using triple quotes:

```
"""
Anything here prints
as you
enter it ...
"""
```

or

```
'''You can
use triple
single quotes
too ... '''
```

Changing the behaviour of print

- By default, the **print** function does two things:
  1. It prints out the strings given to it, each separated by a space
  2. It adds a *newline* character at the end

```
print('String 1', 'String 2', 'String 3')
```

produces

```
String 1 String 2 String 3
```

Concatenating strings

- We can concatenate (join together) two strings by using the "+" operator

```
'Conca' + 'tenating␣strings'
```

- The "+" operator creates a new string that is the concatenation of two strings, without any spaces in between

```
print('Conca' + 'tenat' + 'ing␣strings')
```

produces

```
Concatenating strings
```

- Useful when using strings stored in *variables*

Repeating strings

- The "*" operator creates a string that is repeated the specified number of times:

```
print('String' * 5)
```

produces

```
StringStringStringStringString
```

Variables

- A variable is one of the most important elements of a programming language
- A variable can be thought of as a named/labelled storage location for data in memory
- It's called a *variable* because its contents can change during the execution of the program
- It is a **storage location**, ie Python will reserve some memory to store the data.
- The *value taken by the variable* will be stored at that location

Using variables (1)

- Running the following program:

```
# Greeting program, v1.0
# Demonstrates the use of a variable

print('Good morning!')

firstname = 'John'
print('Hi ' + firstname )
print('How are you?')
```

produces

```
Good morning!
Hi John
How are you?
```

Assignment

- An assignment statement gives a variable a value

```
firstname = 'John'
```

- In Python, the assignment operator is denoted by the "=" character
- A variable is just a name
- The assignment statement associates the name on the left of the assignment symbol with the value on the right of the assignment symbol
- We say that the variable is assigned a value
  - `firstname` is assigned the value `"John"`
  - `firstname` is given the value `"John"`
  - `firstname` becomes the value `"John"`
- NB The "=" character is not the equals we use in mathematics!

Using variables (2)

- In our program, the statement

  ```
  firstname  =  'John'
  ```

  creates a variable *firstname* containing the string "`John`"
- Note that when we use the variable in an expression or in a statement, the contents of the variable are used
- Recall that the output of our program has

  ```
  Hi John
  ```

  not

  ```
  Hi firstname
  ```

Using variables (3)

- The contents of a variable can be changed
- We simply have another assignment

```
# Greeting program, v2.0
# Demonstrates the use of a variable
name = 'John'
print('Hi ' + name + '!')
print('How are you?')

# Get a new value of name
name = 'Mary'
print('Oh! You\'re ' + name + 'now!')
```

produces

```
Hi John!
How are you?
Oh! You're Marynow!
```

Mind the gap!

- Correcting the output of our previous program:

```
# Greeting program, v2.1
# Demonstrates the use of a variable

name = 'John'
print('Hi ' + name + '!')
print('How are you?')

# Get a new value of name
name = 'Mary'
print('Oh! You\'re ' + name + ' now!')
```

produces

```
Hi John!
How are you?
Oh! You're Mary now!
```

Naming variables (1)

- A variable name can only contain the following:
  - letters (lowercase and uppercase, ie a–z and A–Z)
  - digits (0–9)
  - the "_" character
- A variable name cannot start with a digit
- Variable names in Python are case-sensitive
- `name` and `Name` are different variables
- There are a small number of reserved words or keywords that have built-in meanings in Python and cannot be used as variable names
- The different versions of Python have slightly different lists of reserved words

Naming variables (2)

- Choose *descriptive* names
- When you re-read your program in two weeks' time, or in a year's time, you will be grateful!
- When your team colleague reads your program in two years' time, after you've moved to a new section in the company, they (and you) will be extra grateful!
- For example, `tax_due` is a better name than `name` or `var3` or `x1234` or even `td`

Naming variables (3)

- Consider the following two programs:

```
# Greeting program, v3.0
# Demonstrates the use of variable names

name = 'John'
print('Hello ' + name + '!')
```

and

```
# Greeting program, v3.1
# Demonstrates the (bad) use of variable names

x = 'John'
print('Hello ' + x + '!')
```

- What is the difference in the output?
- None!

Don't rely on variable names. . . (1)

- The fact that a variable is called a particular name does not confer on it any particular properties
- For example, a variable called `name` does not necessarily hold names (although clearly that would be a good idea)
- If the name of a variable called `name` is changed everywhere in the program to `abcxyz`, the program will run in exactly the same way
- Recall that the Python interpreter (and the compilers/interpreters for other languages) translates the source code into code that the machine can execute
- So the variable names are for the **benefit/convenience** of the programmer or (human) reader of the program, not the computer

Don't rely on variable names. . . (2)

- Consider the following two programs:

```
# Greeting program, v4.0
# Demonstrates the further use of variable names

greeting = 'Hello'
name = 'John'
print(greeting, name)
```

and

```
# Greeting program, v4.1
# Demonstrates the further (bad) use of variable

name = 'Hello'
greeting = 'John'
print(greeting, name)
```