Chapter 1 – Introduction to Python Programming

Outline

1.1 Introduction

- Introduction to Python programming
- Introduction to programming techniques
  - Structured programming
  - Object-oriented programming

1.2 First Program in Python: Printing a Line of Text

- Python
  - The **#** symbol
    - Used to denote a single line comment
  - The **`print()`** function
    - Used to send a stream of text to be output to the user
- Executing
  - Saving as a file
    - Type code into a .py file and save it
    - To run it type **`python`** ***`fileName`*****`.py`**
  - Executing code
    - Type **`python`** in the command line
    - Runs the python interpreter

```python
# Example 01_01: example01_01.py
# Printing a line of text in Python.

print("Welcome to Python Programming!")
```

This is a comment

Prints out the line of text

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\cenit python courses\ch01>python example01_01.py
Welcome to Python Programming!

C:\cenit python courses\ch01>_
```

Fig 1.2 Example 1 Code and Output

1.2 First Program in Python: Printing a Line of Text in Interative Mode

```
C:\Windows\System32\cmd.exe - python

C:\cenit python courses\ch01>python
Python 3.5.1 |Anaconda 4.0.0 (64-bit)| (default, Feb 16 2016,
 09:49:46) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more inf
ormation.
>>> print("Welcome to Python Programming")
Welcome to Python Programming
>>>
```

Fig. 1.2 Interactive mode.

1.3 Modifying our First Python Program

- Text outputs
  - How to display the text on one line through multiple statements, 1.3.1
  - How to display the text on several lines with only one code statement, 1.3.2

1.3.1 Displaying a Single Line of Text with Multiple Statements

- Printing Lines of Text
  - Python displays each print function on a new line
  - The comma can be used to tell the compiler to make a space rather than a white line

1.3.1 Displaying a Single Line of Text with Multiple Statements

Computer system	Keyboard combination
UNIX/Linux systems	*Ctrl-D* (on a line by itself)
Windows	*Ctrl-Z* (sometimes followed by pressing *Enter*)
Macintosh	*Ctrl-D*

Fig. 1.3 End-of-file key combinations for various popular computer systems.

```python
# Example 01_02: example01_02.py
# Printing a line with multiple st

print("Welcome to", end=" ")
print("Python Programming!")

```

The comma followed by end = “ “ tells the compiler to insert a space rather than go to the next line

```
C:\Windows\System32\cmd.exe

C:\cenit python courses\ch01>python example01_02.py
Welcome to Python Programming!

C:\cenit python courses\ch01>
```

Fig 1.4: Example 2 Printing a line with multiple statements

1.3.2 Displaying Multiple Lines of Text with a Single Statement

- Escape characters
  - Used to perform a different task that normally intended
  - **`\n`** – insert a new line
  - **`\"`** – insert double quotes
  - **`\'`** – insert a single quote
  - **`\\`** – inserts a backslash
  - More are listed in Fig. 1.6

C:\cenit python courses\ch01\example01_03.py - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

example01_01.py | example01_02.py | example01_03.py | example01_04.py | example01_05.py | example01_06.py | example01_07.py | example01_08.py | example01_09.py

```python
# Example 01_03: example01_03.py
# Printing multiple lines with a single statement.

print("Welcome\nto\nPython\nProgramming!")
```

C:\Windows\System32\cmd.exe

```
C:\cenit python courses\ch01>python example01_03.py
Welcome
to
Python
Programming!

C:\cenit python courses\ch01>
```

Fig 1.5 Printing Multiple lines with a single statement

1.3.2 Displaying Multiple Lines of Text with a Single Statement

Escape Sequence	Description
`\n`	Newline. Move the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Move the screen cursor to the beginning of the current line; do not advance to the next line.
`\b`	Backspace. Move the screen cursor back one space.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Print a backslash character.
`\"`	Double quote. Print a double quote character.
`\'`	Single quote. Print a single quote character.

Fig. 1.6 Escape sequences.

1.4 Another Program: Adding Integers

- Functions
  - The **`input()`** function
    - Used to retrieve data from the user
  - The **`int()`** function
    - Used to convert strings to integers

C:\cenit python courses\ch01\example01_04.py - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

example01_01.py | example01_02.py | example01_03.py | example01_04.py | example01_05.py | example01_06.py | example01_07.py | example01_08.py | example01_09.py

```python
# Example 01_04: example01_04.py
# Simple integer addition program.

# prompt the user for input
integer1 = input( "Enter first integer:\n" )  # read a string
integer1 = int( integer1 )    # convert the string to an integer

integer2 = input( "Enter second integer:\n" ) # read a string
integer2 = int( integer2 )    # convert the string to an integer

sum = integer1 + integer2     # assignment of su

print ("Sum is", sum)         # print sum
```

Prompts the user to enter and integer value

Converts the string value into an integer value

Adds up and then prints out the sum of the two numbers

Python file | length : 457 lines : 15 | Ln : 1 Col : 24 Sel : 0 | 0 | Dos\Windows | ANSI as UTF-8 | INS

```
C:\cenit python courses\ch01>python example01_04.py
Enter first integer:
4
Enter second integer:
6
Sum is 10
```

Fig 1.7 Simple Integer Addition Program introducing variables

1.4 Another Program: Adding Integers

C:\Windows\System32\cmd.exe - python

```
>>> integer1 = input( "Enter first integer:\n" )
Enter first integer:
2
>>> integer2 = input( "Enter second integer:\n" )
Enter second integer:
4
>>> sum = integer1 + integer2
>>> print ("Sum is", sum)
Sum is 24
>>>
>>>
>>>
>>>
```

Fig. 1.8 Adding values from `input` (incorrectly) without converting to integers (the result should be 6).

1.5 Memory Concepts

- Objects
  - Every object has a type, size, value, and location
    - Stored in computers memory
    - Type and location cannot be changed
  - When a variable is made the name is binded to the value
  - Values are not modified as a computer performs the calculation

1.5 Memory Concepts

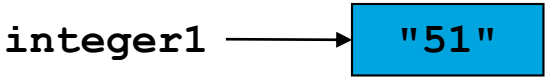

Fig. 1.9 Memory location showing value of a variable and the name bound to the value.

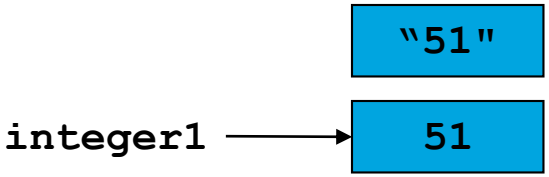

Fig. 1.10 Memory location showing the name and value of a variable.

1.5 Memory Concepts

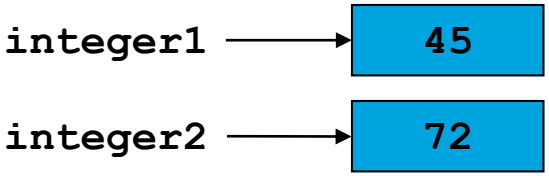

Fig. 1.11 Memory locations after values for two variables have been input.

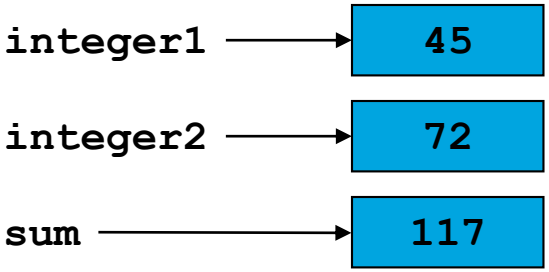

Fig. 1.12 Memory locations after a calculation.

```python
# Example 01_05: example01_05.py
# Displaying an object's location, type and value.

# prompt the user for input
integer1 = input( "Enter first integer:\n" )  # read a string
print("integer1: ", id( integer1 ), type( integer1 ), integer1 )
integer1 = int( integer1 )   # convert the string to an integer
print("integer1: ", id( integer1 ), type( integer1 ), integer1)

integer2 = input( "Enter second integer:\n" ) # read a string
print("integer2: ", id( integer2 ), type( integer2 ), integer2)
integer2 = int( integer2 )   # convert the string to an integer
print ("integer2: ", id( integer2 ), type( integer2 ), integer2)

sum = integer1 + integer2    # assignment of sum
print ("sum: ", id( sum ), type( sum ), sum)
```

```
C:\cenit python courses\ch01>python example01_05.py
Enter first integer:
23
integer1:  2557971708984 <class 'str'> 23
integer1:  1974071760 <class 'int'> 23
Enter second integer:
12
integer2:  2557971708984 <class 'str'> 12
integer2:  1974071408 <class 'int'> 12
sum:  1974072144 <class 'int'> 35
```

Fig 1.13: Obtaining objects location and type

1.6 Arithmetic

- Symbols
  - * = multiply
  - / = divide
  - % = modulus
  - ** = exponential
  - // = floor division
- Order
  - Operators are done in order of parenthesis, exponents, multiple and divide (left to right), and lastly add and subtract (left to right)

1.6 Arithmetic

Python operation	Arithmetic operator	Algebraic expression	Python expression
Addition	+	*f + 7*	*f + 7*
Subtraction	–	*p – c*	*p - c*
Multiplication	*	*bm*	*b * m*
Exponentiation	**	x^y	*x ** y*
Division	/ //	*x / y*	*x / y* *x // y*
Modulus	%	*r mod s*	*r % s*

Fig. 1.14 Arithmetic operators.

1.6 Arithmetic

```
C:\Windows\System32\cmd.exe - python
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\cenit python courses\ch01>python
Python 3.5.1 |Anaconda 4.0.0 (64-bit)| (default, Feb 16 2016, 09:49:46
900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 3 / 4        # floor division (default behavior)
0.75
>>> 3.0 / 4.0    # true division (floating-point operands)
0.75
>>> 3 // 4       # floor division (only behavior)
0
>>> 4 // 3       # floor division (only behavior)
1
```

Fig. 1.15 Difference in behavior of the `/` and `//` operators

1.6 Arithmetic

Operator(s)	Operation(s)	Order of Evaluation (Precedence)
`( )`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
`**`	Exponentiation	Evaluated second. If there are several, they are evaluated right to left.
`* / // %`	Multiplication Division Modulus	Evaluated third. If there are several, they are evaluated left to right
`+ -`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Fig. 1.16 Precedence of arithmetic operators.

1.6 Arithmetic

Step 1.

```
y = 2 * 5 ** 2 + 3 * 5 + 7
        5 ** 2 = 25
```

Exponentiation

Step 1.

```
y = 2 * 25 + 3 * 5 + 7
    2 * 25 = 50
```

Leftmost multiplication

Step 3.

```
y = 50 + 3 * 5 + 7
        3 * 5 = 15
```

Multiplication before addition

Step 4.

```
y = 50 + 15 + 7
    50 + 15 = 65
```

Leftmost addition

Step 5.

```
y = 65 + 7 is 72
```

Last edition

Step 6.

```
y = 72
```

Python assigns **72** to **y**

Fig. 1.17 Order in which a second-degree polynomial is evaluated.

1.7 String Formatting

- Strings
  - Unlike other languages strings are a built in data type
    - Allows for easy string manipulation
  - Double quote strings
    - Single quotes need not be escaped
  - Single quote strings
    - Double quotes need not be escaped
  - Triple quoted strings
    - Do not need any escape sequence
    - Used for large blocks of text

```python
# Example 01_06: example01_06
# Creating strings and using  trings.

print( "This is a string with \"double quotes
print( 'This is another string with "double quotes."')
print( 'This is a string with \'single quotes.\'')
print( "This is another string with 'single quotes.'")
print("""This string has "double quotes" and 'single quotes'.
   You can even do multiple lines.""
print( '''This string also has "doubl  tes.''')
```

Strings in single quotes need not have double quotes escaped

Strings with double quotes need not escape single quotes

Strings in triple quotes do not have to escape anything and can span many lines

C:\Windows\System32\cmd.exe

```
C:\cenit python courses\ch01>python example01_06.py
This is a string with "double quotes."
This is another string with "double quotes."
This is a string with 'single quotes.'
This is another string with 'single quotes.'
This string has "double quotes" and 'single quotes'.
   You can even do multiple lines.
This string also has "double" and 'single' quotes.

C:\cenit python courses\ch01>
```

Fig 1.18: Creating Strings

C:\cenit python courses\ch01\example01_07.py - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

example01_01.py | example01_02.py | example01_03.py | example01_04.py | example01_05.py

```python
# Example 01_07: example01_07.py
# String formatting.

integerValue = 4237
print("Integer ", integerValue)
print("Decimal integer %d" % integerValue)
print("Hexadecimal integer %x\n" % integerValue)

floatValue = 123456.789
print( "Float", floatValue)
print( "Default float %f" % floatValue)
print( "Default exponential %e\n" % floatValue)

print( "Right justify integer (%8d)" % integerValue)
print( "Left justify integer  (%-8d)\n" % integerValue)

stringValue = "String formatting"
print("Force eight digits in integer %.8d" % integerValue)
print( "Five digits after decimal in float %.5f" % floatValue)
print( "Fifteen and five characters allowed in string:")
print( "(%.15s) (%.5s)" % ( stringValue, stringValue ))
```

The %e is used to format the string to scientific notation number

Formats the string to contain exactly a specified amount of letters

Formats the string to only allow so many characters

Fig 1.19: Formatting Strings

`Example01_.py`
Program Output

```
C:\Windows\System32\cmd.exe

C:\cenit python courses\ch01>python example01_07.py
Integer  4237
Decimal integer 4237
Hexadecimal integer 108d

Float 123456.789
Default float 123456.789000
Default exponential 1.234568e+05

Right justify integer (    4237)
Left justify integer  (4237    )

Force eight digits in integer 00004237
Five digits after decimal in float 123456.78900
Fifteen and five characters allowed in string:
(String formatti) (Strin)

C:\cenit python courses\ch01>
```

Fig 1.19

1.7 String Formatting

	Conversion Specifier Symbol and Meaning
c	Single character (i.e., a string of length one) or the integer representation of an ASCII character.
s	String or a value to be converted to a string.
d	Signed decimal integer.
u	Unsigned decimal integer.
o	Unsigned octal integer.
x	Unsigned hexadecimal integer (with hexadecimal digits **a** through **f** in lowercase letters).
X	Unsigned hexadecimal integer (with hexadecimal digits **A** through **F** in uppercase letters).
f	Floating-point number.
e, E	Floating-point number (using scientific notation).
g, G	Floating-point number (using least-significant digits).

Fig. 1.20 String-formatting characters.

1.8 Decision Making: Equality and Relational Operators

- The if structure
  - Can be formed with equality and relational operators
    - <, >, ==, …

1.8 Decision Making: Equality and Relational Operators

Standard algebraic equality operator or relational operator	Python equality or relational operator	Example of Python condition	Meaning of Python condition
Relational operators			
>	`>`	`x > y`	`x` is greater than `y`
<	`<`	`x < y`	`x` is less than `y`
>=	`>=`	`x >= y`	`x` is greater than or equal to `y`
<=	`<=`	`x <= y`	`x` is less than or equal to `y`
Equality operators			
=	`==`	`x == y`	`x` is equal to `y`
	`!=`, `<>`	`x != y`, `x <> y`	`x` is not equal to `y`

Fig. 1.21 Equality and relational operators.

C:\cenit python courses\ch01\example01_08.py - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

example01_01.py | example01_02.py | example01_03.py | example01_04.py | example01_05.py | example01_06.py

```python
#  Example 01_08: example01_08.py
# Compare integers using if structures, relational operators
# and equality operators.

print("Enter two integers, and I will tell you")
print("the relationships they satisfy.")

# read first string and convert to integer
number1 = input( "Please enter first integer: " )
number1 = int( number1 )

# read second string and convert to integer
number2 = input( "Please enter second integer: " )
number2 = int( number2 )

if number1 == number2:
    print( "%d is equal to %d" % ( number1, number2 ))

if number1 != number2:
    print("%d is not equal to %d" % ( number1, number2 ))

if number1 < number2:
    print("%d is less than %d" % ( number1, number2 ))

if number1 > number2:
    print ("%d is greater than %d" % ( number1, number2 ))

if number1 <= number2:
    print("%d is less than or equal to %d" % ( number1, number2 ))

if number1 >= number2:
    print ("%d is greater than or equal to %d" % ( number1, number2 ))
```

Gets two values from the user and converts them to strings

Checks each of the rational operators or the numbers using if statements

Fig 1.22: Using Relational Operators

```
Enter two integers, and I will tell you
the relationships they satisfy.
Please enter first integer: 37
Please enter second integer: 42
37 is not equal to 42
37 is less than 42
37 is less than or equal to 42

Enter two integers, and I will tell you
the relationships they satisfy.
Please enter first integer: 7
Please enter second integer: 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7

Enter two integers, and I will tell you
the relationships they satisfy.
Please enter first integer: 54
Please enter second integer: 17
54 is not equal to 17
54 is greater than 17
54 is greater than or equal to 17
```

1.8 Decision Making: Equality and Relational Operators

```
C:\Windows\System32\cmd.exe - python
C:\cenit python courses\ch01>python
Python 3.5.1 |Anaconda 4.0.0 (64-bit)| (default, Feb 16 20
16, 09:49:46) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>> print(1 + ')
  File "<stdin>", line 1
    print(1 + ')
              ^
SyntaxError: EOL while scanning string literal
>>> print(1 +
... 2)
3
>>>
```

Fig 1.23: Syntax Errors

1.8 Decision Making: Equality and Relational Operators

Operators				Associativity	Type
`()`				left to right	parentheses
`**`				right to left	exponential
`*`	`/`	`//`	`%`	left to right	multiplicative
`+`	`-`			left to right	additive
`<`	`<=`	`>`	`>=`	left to right	relational
`==`	`!=`	`<>`		left to right	equality

Fig. 1.24 Precedence and associativity of operators discussed so far.

1.9 Indentation

- Indenting
  - Used to delimit code
  - Python uses no end of statement character
  - Therefore a new line of code is determined by return space
  - Indenting is the same way
    - Python does not use {} to enclose a multi-line statement
    - The indentation must be exactly the same same
  - There is no exact rule for the number of spaces but they are generally in groups of four spaces by convention

```python
# Example 01_09: example01_09.py
# Using if statements, relational operators and equality
# operators to show improper indentation.

print( "Enter two integers, and I will tell you")
print( "the relationships they satisfy.")

# read first string and convert to integer
number1 = input( "Please enter first integer: " )
number1 = int( number1 )

# read second string and convert to integer
number2 = input( "Please enter second integer: " )
number2 = int( number2 )

if number1 == number2:
    print("%d is equal to %d" % ( number1, number2 ))

    # improper indentaion causes this if statement to execute only
    # when the above if statement executes
    if number1 != number2:
        print( "%d is not equal to %d" % ( number1, number2 ))

if number1 < number2:
    print("%d is less than %d" % ( number1, number2 ))

if number1 > number2:
    print( "%d is greater than %d" % ( number1, number2 ))

if number1 <= number2:
    print( "%d is less than or equal to %d" % ( number1, number2 ))

if number1 >= number2:
    print("%d is greater than or equal to %d" % ( number1, number2 ))
```

Since this if statement is indented it is considered part of the other if statement

on file | length : 1118 lines : 36 | Ln : 4 Col : 1 Sel

Fig 1.25: Problems with improper indentation

```
Enter two integers, and I will tell you
the relationships they satisfy.
Please enter first integer: 1
Please enter second integer: 2
1 is less than 2
1 is less than or equal to 2
```

Fig 1.26: Output caused by incorrect indentation

1.10 Thinking about Objects: Introduction to Object Technology

- Objects
  - Everything in the real world is made of objects
  - Each object has attributes
    - Shape, size, color, weight
  - Each object has behaviors
    - Roll, bounce, inflate, deflate
- **O**bject **O**riented **P**rogramming (OOP)
  - Modes real world objects with programming counterparts
  - Information hiding
    - Know how to communicate with one another
    - Don’t know the specifics of other objects
  - Encapsulate, to prevent code repetition