Programming Fundamentals 2 using Python

Prepared by: Hanan Hardan

Chapter 1:

Python Introduction

Introduction

- Python is an Open-source language that can be implemented using several Open-source editors:
  - Anaconda (Jupyter Notebook, Spider, VS code, ..etc.)
  - Google Colaboratory

Important Notes about Python

- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Indentation refers to the spaces at the beginning of a code line.
- Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:
  print("Five is greater than two!")
```

Note: Python will give you an error if you skip the indentation:

Python Comments

- Comments can be used to explain Python code.
- Comments starts with a #, and Python will ignore them:

```
#This is a comment

""" This is a comment written
in more than just one line """

'''This is a comment written
In more than just one line '''
```

Python Data Types

Built-in Data Types	
Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

Python Data Types

- In Python, the data type is set when you assign a value to a variable:
- Python has no command for declaring a variable.

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float

Python Variables

- You can get the data type of any object by using the type() function:
- Example: Print the data type of the variable x:

```
x = 5
print(type(x))
Print(x)
```

Python Variables

- Variables do not need to be declared with any particular *type*, and can even change type after they have been set.
- Example

```
x = 4            # x is of type int
x = ”Sally”      # x is now of type str
print(x)
```

Variable Names

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Variable Names

- Example

Legal variable names:	Illegal variable names:

myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
	2myvar = "John"
	my-var = "John"
	my var = "John"

Note: Remember that variable names are case-sensitive

Assign Multiple Values

- Python allows you to assign values to multiple variables in one line:

Example:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

- Assign one value to multiple variables

Example:

```
x = y = z = 10
```

Python Casting

- Specify a Variable Type

This can be done with casting. Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Python Casting

Example:

Integers:	Floats:	Strings:

x = int(1)
# x= 1 	
y = int(2.8)
# y= 2 	
z = int("3")
# z= 3 	
	x = float(1)
	# x = 1.0
	y = float(2.8)
	# y= 2.8
	z = float("3")
	# z= 3.0
	w = float("4.2")
	# w= 4.2
		x = str("s1")
		# x = 's1'
		y = str(2)
		# y= '2'
		z = str(3.0)
		# z= '3.0'

Input Statement

```
name = input ()
word = input('Enter a word: ')
```

- The value entered by the user is always considered string, unless it was casted (the type is changed)

```
x = int (input ())
y = int (input ())
```

Output Statement

```
print('Welcome:', end='')
print ('Sami')

x,y=4,5
print('sum=',end='')
print(x+y)
```

Output Statement

```
w, x, y, z = 10, 15, 20, 25
print(w, x, y, z)
print(w, x, y, z, sep=',')
print(w, x, y, z, sep='')
print(w, x, y, z, sep=':')
print(w, x, y, z, sep='-----')
print(w, x, y, z, sep='\t')
```

Output Variables

- The Python print statement is often used to output variables.
- To combine both text and a variable, Python uses the + character:

Example:

```
x = "awesome"
print("Python is " + x)
```

- You can also use the + character to add a variable to another variable:

Example:

```
x = "Python is "
y = "awesome"
z =  x + y
print(z)
```

Output Variables

- For numbers, the + character works as a mathematical operator:

Example:

```
x = 5
y = 10
print(x + y)
```

- If you try to combine a string and a number, Python will give you an error:

Example:

```
x = 5
y = "John"
print(x + y) -----> error
```

Python Operators

- Python divides the operators in the following groups:
  - Arithmetic operators
  - Comparison operators
  - Logical operators
  - Identity operators
  - Membership operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Example 1

Write a Python program that reads 3 numbers and calculates their average

Solution:

```
n1, n2, n3 = float (input ('Enter 3 numbers')), float (input ()), floa
    t (input ())
average = (n1 + n2 + n3) /3
print ('average = ', average)
```

Example 2

Write a Python program that reads the basic salary of an employee, calculates his tax (6% of his basic salary and calculates the Net salary by the following equation:

Note: Net Salary = Basic Salary – Tax

Solution:

```
basicSalary = float (input('Please, enter the basic salary'))
tax = 0.06 * basicSalary
NetSalary = basicSalary - tax
print ('Net Salary = ', NetSalary)
```

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

- Note:**Printing Boolean values**

  True and False

Python Identity Operators

- Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

- Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

If Statement

- One way selection

Example:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

- Indentation:

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Note: If statement, without indentation (will raise an error)

Python If ... Else

- Two-way selection

```
a = 33
b = 200
if a>b:
  print ('a is the greatest')
else:
  print (‘b is the greatest')
```

Python If … Else

- Multi-way selection

The else keyword catches anything which isn't caught by the preceding conditions.

Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Python If ... Else

- Nested If:

You can have if statements inside if statements, this is called *nested if* statements.

- Example:

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

Example 1

Write a Python program that reads two integers and divides the first over the second. Make sure that the second integer is not zero, otherwise, give an error message.

Solution:

```
n1, n2 = int (input('enter two integers')), int (input())
if  n2 != 0:
  print ('The result is: ', n1/n2)
else:
  print ('Error: Divide by Zero')
```

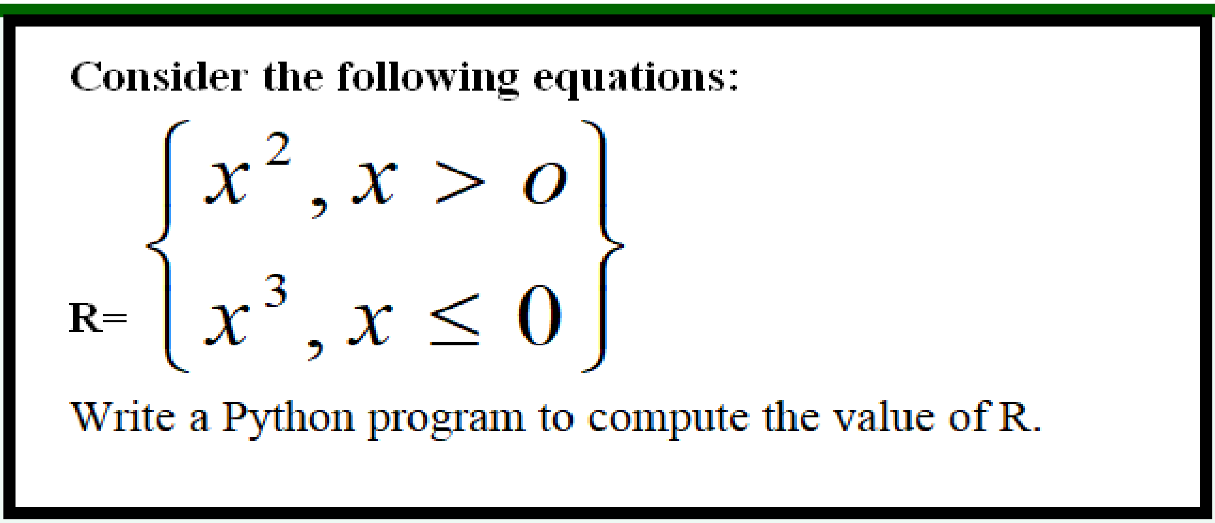
Example 2

- Write a Python program that reads an integer and decides wither it is divisible of 3 or not.

Solution:

```
num = int (input ('Enter an integer'))
if num%3 == 0:
  print (num, ' is divisible by 3')
else:
  print (num, ' is not divisible by 3')
```

Example 3



Consider the following equations:

$$R= \begin{cases} x^2, x > o \\ x^3, x \leq 0 \end{cases}$$

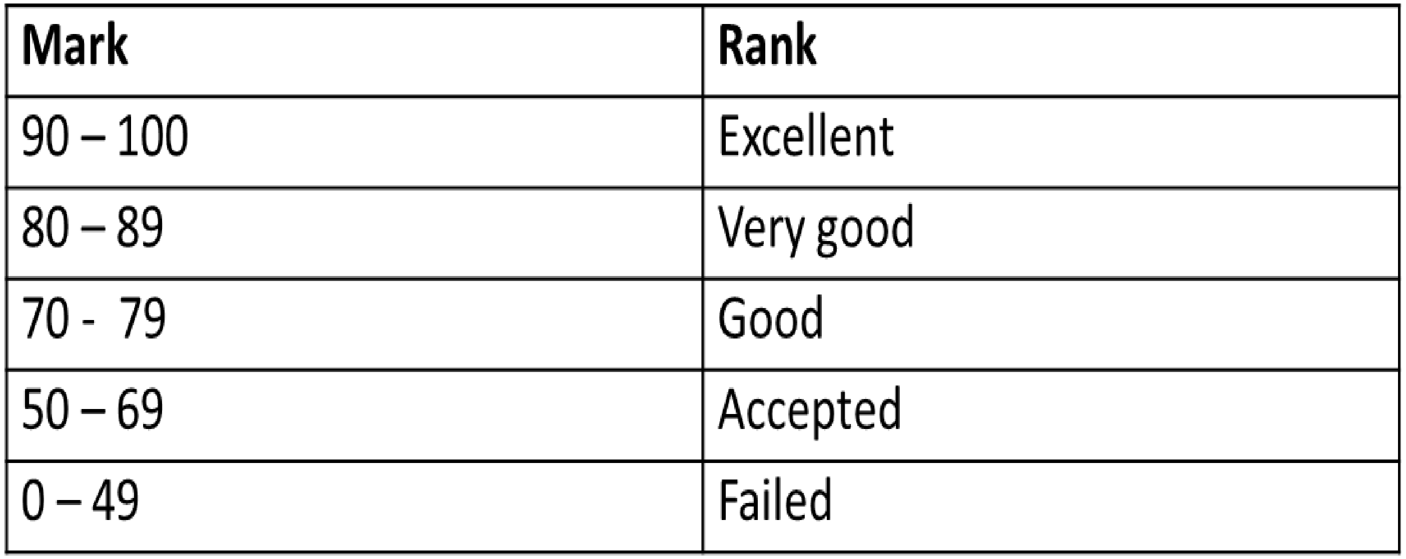
Write a Python program to compute the value of R.

```
x = int (input('Enter an integer'))
if x > 0:
  R = x ** 2
else:
  R = x ** 3
print ('R = ', R)
```

Example 4

- Write a Python program that reads a student mark, and outputs the corresponding rank, according to the table below.

  Hint: Make sure that the mark is in the correct range



Mark	Rank
90 – 100	Excellent
80 – 89	Very good
70 - 79	Good
50 – 69	Accepted
0 – 49	Failed

Example 4

```
mark = int (input ('Enter a student mark'))
if mark >=35 and mark <=100:
  if mark >= 90:
    print ('Excellent')
  elif mark >= 80:
    print ('Very Good')
  elif mark >= 70:
    print ('Good')
  elif mark >= 50:
    print ('Accepted')
  else:
    print ('Failed')
else:
  print ('Mark out of range')
```

Example 5

Write a Python program that reads a day in a week from the user, and prints the following:

1. Working day: if the user enters (Sunday, Monday, Tuesday, Wednesday or Thursday)
2. Weekend: if the user enters (Friday or Saturday)

Example 5

Solution:

```
day = input ('Please, enter a week day')
if day == 'Sunday' or day == 'Monday' or day == 'Tuesday' or day
    == 'Wednesday' or day == 'Thursday':
  print ('Working day')
elif day == 'Friday' or day == 'Saturday':
  print ('Weekend')
else:
  print ('Wrong day name or wrong spelling')
```

Python If ... Else

- The pass Statement

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

- Example:

```
x = int (input())
if x < 0:
  pass  #To Do: Add a proper error message
else:
  print (x + 10)
```

Python Loops

Python has two primitive loop commands:

- for loops
- while loops

The For Loop

```
for x in range(6):
  print(x)

for x in range(6):
  print(x)
  print()

for x in range(2, 6):
  print(x)

for x in range(2, 30, 3):
  print(x)
```

Example 1

- Write a Python program that prints the following sequence:

  2 4 6 8 10 12 14 16 18 20

```
for x in range (2,21, 2):
  print (x, end=' ' )
```

- Write a Python program that prints the following sequence:

  20 18 16 14 12 10 8 6 4 2

```
for x in range (20,1, -2):
  print (x, end=' ' )
```

Example 2

Write a Python program reads an integer from the user and prints its factorial:

```
n = int (input ('Please, Enter an integer'))
prod = 1
for i in range(n, 1, -1):
    prod *= i

print (i, '! = ', prod, sep = '')
```

The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

- Example: Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Note: remember to increment i, or else the loop will continue forever.

Example 3

Write a Python program that reads a sequence of positive integers and sums them. A negative integer should stop the input.

```
print ('Please enter sequence of positive integers, when finished, enter a negative integer')
sum = 0
num = int (input ())
while num >=0:
    sum += num
    num = int (input())
print ('Summation = ', sum)
```

Example 4

Write a Python program that reads integer numbers and finds their product, each time keep asking the user to continue or not.

```
prod = 1
finished = False
num = int (input ('Please, enter an integer'))
while not finished:
    prod *= num
    print (num)
    respond = input ('Do you want to continue? (Y/N)')
    if respond == 'n' or respond == 'N':
        finished = True
    else:
        num = int (input('Please, enter an integer'))
print ('Result = ', prod)
```

Break and Continue

- **Break**: when used in a loop, it skips the rest of the loop body and exits it.

- **Continue**: when used in a loop, it skips the rest of the loop body, but checks wither the loop condition is still true, if it still true, it returns to the start of the loop body and continues.

Example 5

Re-write the code in Example 4, by using Break command

```
prod = 1
while True:
    num = int (input ('Please, enter an integer'))
    prod *= num
    respond = input ('Do you want to continue? (Y/N)').lower()
    if respond == 'n':
        break

print ('Result = ', prod)
```

Example 6

- Re-write the code in Example 5, but by ignoring the negative numbers.
- Hint: you can use break and continue commands

```
prod = 1
while True:
    num = int (input ('Please, enter an integer'))
    if num<0:
        print ('Negative integers are ignored, try again')
        continue
    prod *= num
    respond = input ('Do you want to continue? (Y/N)').lower()
    if respond == 'n':
        break;

print ('Result = ', prod)
```

The For Loop

- Nested For Loop

```
for i in range (1,6):
  for j in range (2,4):
    print (i*j)
```