Data Type: Numeric values

I. Integer Values (Type: int)

The number five (5) is an example of a numeric value. In mathematics, 1, 2, 3, 4, 5….. are integers. Integers have no fractional parts, and they can be positive, negative, or zero. Python supports a number of numeric and non-numeric values. In particular, Python programs can use integer values. The Python statement
print(4)
prints the value 4.

II. Floating-point number (Type: float)

Many computational tasks require numbers that have fractional parts. For example, to compute the area of a circle given the circle's radius, the value pi, or approximately 3.14159 is used. Python supports such non-integer numbers, and they are called *floating point numbers*. The name implies that during mathematical calculations the decimal point can move or "float" to various positions within the number to maintain the proper number of significant digits.

Floating-point numbers are an approximation of mathematical real numbers. Floating-point numbers is limited, since each value must be stored in a fixed amount of memory. Because of the limited number of digits available, some numbers can be only approximated; for example,

```
a = 23.3123400654033989
print (a)
```

will store and print a as 23.312340065403397. *Try to enter any large real number and see how the number is stored and printed on interactive mode.*

Activity 1:

Instructions	Output of x	Data type of x	Remarks
x = 6 / 2 print (x)			What is the difference and why?
x = 6.0/2 print (x)			
x = 12/3 print (x)			What is the difference and why?
x = 12/3.0 print (x)			
x = 14/4 print (x)			

III. Data Type: Strings

A string is a sequence of characters. Strings most often contain non-numeric characters: for example

```
>>> print (“School”)
School
>>>print (‘School’)
School
```

Python recognizes both single quotes (’) and double quotes (") as valid ways to delimit a string value.

Variables and Assignment Statement

In algebra, variables represent numbers. The same is true in Python, except Python variables also can
represent values other than numbers. In variable.py, the program uses a variable to store an integer value and then prints the value of the variable.

variable.py

```
x = 3
print (x)
```

x = 3 is an assignment statement. An assignment statement associates a value with a variable. The key to an assignment statement is the symbol = which is known as the assignment operator. The statement assigns the integer value 3 to the variable x. [Do not say x is equal to 3. Assignment statement is NOT a mathematical equation!]

In variable.py, print(x) statement prints the variable x’s current value.

multi_assignment.py

```
x, y = 5, 15
print (x, y)
```

We can assign multiple variables in one statement. For example, The statement x, y = 5, 15 assigns 5 to x and 15 to y respectively.

Activity 1:
Variables can be re-assigned different values as needed. Enter the instructions of variable_reassign.py and observe its output. Enter its output in the table provided.

variable_reassign.py

```
x = 3
print (“x = “, x)
x = 13
print (“x = “, x)
x = 23
print (“x = “, x)
```

Output

Activity 2:
Compare print (x) and print (“x”)
Print (x) means ____

Print (“x”) means ____

In variable-reassign.py, print (“x = “, x) accepts two parameters. The first parameter is the string ’x =’, and the second parameter is the variable x which is an integer value.

Identifiers (eg variable names)

Reserved words:

and	del	from	None	try
as	elif	global	nonlocal	True
assert	else	if not while	break	except
import	or	with	class	false
in	pass	yield	continue	finally
is	raise	def	for	return
lambda				

While mathematicians are happy with giving their variables one-letter names like x, programmers should use longer, more descriptive variable names. Names such as sum, height, and sub_total are much better than s, h, and st. A variable’s name should be related to its purpose within the program. Good variable names make programs more readable by humans (for adding more user-friendliness to your program).

Python has strict rules for variable names. A variable name is one example of an identifier. An identifier

- must contain at least one character.
- is case sensitive
- must have the first character letter (upper or lower case) or the underscore
- may have the remaining characters (if any) alphabetic characters (upper or lower case), the underscore, or a digit
- No other characters or symbols (including spaces) are permitted in identifiers.
- A reserved word cannot be used as an identifier (refer to the table).

Activity 3:
Circle legal variable names from the following names. You may like to check your answer on Python interpreter by assigning a numeric value to each of the names:

class	class3A	num@	int-num
name_student	first.name	andName	sum_total
sumTotal	sum-total	sum total	sumtotal
2x	while	x2	private
$16	xTwo	_static	_4
___	10%	a27834	abc’s

Comment statement

As programs get bigger and more complicated, they get more difficult to read. For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing. These notes are called **comments**, and they start with the # symbol: for instance, the following shows a program fragment

```
# compute the percentage of the hour that has elapsed
```

```
percentage = (minute * 100) / 60
```

In this case, the comment appears on a line by itself. You can also put comments at the end of a line:

```
percentage = (minute * 100) / 60            # percentage of an hour
```

Everything from the # to the end of the line is ignored—it has no effect on the program.

Programming Practice 1(to be submitted):

Write a program, called rect_*xxyy*.py (xx is your name and yy is your class), which prints out the area and perimeter of a rectangle with specified integer dimensions (eg. 8cm by 10cm) . Pay attention to your choice of variable names and the descriptive output. Also add some comments to state the purpose of or explain the program.