# **CSC 103 Lab 1**

1. **zyBooks activities (PA :Participation activity)**
   1. PA 1.1.1
   2. PA 1.1.5
2. **Python interpreter**
   1. 1.2
   2. PA 1.3.1
3. **IDLE**
4. **Input and output**

Do the followings with **Python interpreter** and **IDLE**

Some of the functions like **input()** and **print()** are widely used for standard input and output operations respectively. Let us see the output section first.

**Python Output Using print() function**

We use the print() function to output data to the standard output device (screen).

print()

print("Python is fun.")

message= "Python is fun."

print(message)

print(‘Python is fun.’)

message= ‘Python is fun.’

print(message)

print("Python\n is fun.")

message= "Python\n is fun."

print(message)

print("Python\n is \nfun.")

print("’Python is fun.’")

print("’Python’ is fu\n.")

print("’Python’ is fu\\n.")

print("Python\t is fun.")

message= "Python\t is fun."

print(message)

print("Python\t is \tfun.")

a = 5

print(a)

print("a =", a)

print("a =")

print(a)

b = a

print('a =', a, '= b')

print('The value of a is', a)

The actual syntax of the print() function is

**print(\*objects, sep=' ', end='\n', file=sys.stdout, flush=False)**

* Here, **objects** is the value(s) to be printed.
* The **sep** separator is used between the values. It defaults into a space character.
* After all values are printed, **end** is printed. It defaults into a new line.
* The **file** is the object where the values are printed and its default value is s**ys.stdout (screen)**.

Here are examples to illustrate this.

print(1,2,3,4)

print(1,2,3,4,sep='\*')

print(1,2,3,4,sep='#',end='&')

a = 5

print("a =", a, sep='00000', end='\n\n\n')

print("a =", a, sep='0', end='')

**Output formatting**

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.

x = 5; y = 10

print('The value of x is {} and y is {}'.format(x,y))

**# Output: The value of x is 5 and y is 10**

Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index).

print('I love {0} and {1}'.format('python','java'))

**# Output: I love python and java**

print('I love {1} and {0}'.format('python','java'))

**# Output: I love java and python**

We can even use keyword arguments to format the string.

print('Hello {name}, {greeting}'.format(greeting = 'Good morning', name = 'John'))

**# Output: Hello John, Good morning**

We can even format strings like the old sprintf() style used in C programming language. We use the % operator to accomplish this.

x = 12.3456789

print('The value of x is %3.2f' %x)

**# Output: The value of x is 12.35**

print('The value of x is %3.4f' %x)

**# Output: The value of x is 12.3457**

Python Input

Up till now, our programs were static. The value of variables was defined or hard coded into the source code. To allow flexibility we might want to take the input from the user. In Python, we have the input() function to allow this. The syntax for input() is

**input([prompt])**

where prompt is the string we wish to display on the screen. It is optional.

input()

name = input()

name =input('Name: ')

print(name)

input('Enter a number: ')

Enter a number: 10

num = input('Enter a number: ')

Enter a number: 10

num

'10'

num = input('Enter a number: ')

Enter a number: 10

print(num+5)

num = int(input('Enter a number: '))

Enter a number: 10

num

10

Here, we can see that the entered value 10 is a string, not a number. To convert this into a number we can use int() or float() functions.

int('10')

10

float('10')

10.0

print(int(‘10’)+5)

This same operation can be performed using the **eval()** function. But it takes it further. It can evaluate even expressions, provided the input is a string

int('2+3')

Traceback (most recent call last):

File "<string>", line 301, in runcode

File "<interactive input>", line 1, in <module>

ValueError: invalid literal for int() with base 10: '2+3'

eval('2+3')

5

Python Import

When our program grows bigger, it is a good idea to break it into different modules.

A module is a file containing Python definitions and statements. Python modules have a filename and end with the extension .py.

Definitions inside a module can be imported to another module or the interactive interpreter in Python. We use the import keyword to do this.

For example, we can import the math module by typing in import math.

import math

print(math.pi)

from math import pi

pi

3.141592653589793