Department of
Computer Science and Engineering

Title: Introduction to Basic Operations on Python (Part-1)

Artificial Intelligence Lab
CSE 404

Green University of Bangladesh

1 Objective(s)

- To acquire knowledge about python.
- To learn variables in python.
- To learn python operators.
- To learn conditional statements in python.
- To learn loops in python.
- To learn functions in python.

2 Introduction

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans, or Eclipse which are particularly useful when managing larger collections of Python files.

In this lab manual, we will learn the basic operations of python like variable declaration, data types, operators, conditional statements, loops, and functions.

3 Getting started with Python Variables

Variables are containers for storing data values. Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.

```python
x = 5
y = "John"
print(x)
print(y)
```

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

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

If you want to specify the data type of a variable, this can be done with casting.

```python
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

You can get the data type of a variable with the type() function.

```python
x = 5
y = "John"
print(type(x))
print(type(y))
```

String variables can be declared either by using single or double quotes:

```python
x = "John"
# is the same as
x = 'John'
```

Variable names are case-sensitive.

```python
a = 4
A = "Sally"
#A will not overwrite a'
```

3.1 Python - Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, totalvolume). 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)

Legal variable names:

```python
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Illegal variable names:

```python
2myvar = "John"
my-var = "John"
my var = "John
```

Variable names with more than one word can be difficult to read. There are several techniques you can use to make them more readable:

3.1.1 Camel Case

Each word, except the first, starts with a capital letter:

```python
myVariableName = "John"
```

3.1.2 Pascal Case

Each word starts with a capital letter:

```python
MyVariableName = "John"
```

3.1.3 Snake Case

Each word is separated by an underscore character:

```python
my_variable_name = "John"
```

3.2 Assign Multiple Values

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

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

And you can assign the same value to multiple variables in one line:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

If you have a collection of values in a list, tuple, etc. Python allows you to extract the values into variables. This is called unpacking.

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

3.3 Output Variables

The Python print() function is often used to output variables.

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

In the print() function, you output multiple variables, separated by a comma:

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

You can also use the + operator to output multiple variables:

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

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

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

In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

```
x = 5
y = "John"
print(x + y)
```

The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

```
x = 5
y = "John"
print(x, y)
```

3.4 Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.Global variables can be used by everyone, both inside of functions and outside.

```python
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

```python
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

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

3.4.1 The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

```python
def myfunc():
  global x
  x = "fantastic"

myfunc()

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

Also, use the global keyword if you want to change a global variable inside a function.

```python
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

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

4 Operator

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

```python
print(10 + 5) # 15
```

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

4.1 Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

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

4.2 Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
$^{=}$	x $^{=}$3	x = x ^{3}
»=	x »= 3	x = x » 3
«=	x «= 3	x = x « 3

4.3 Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Same As
==	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

4.4 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)

4.5 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

4.6 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

4.7 Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
	XOR	Sets each bit to 1 if only one of two bits is 1
	NOT	Inverts all the bits
«	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
»	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

5 Conditional Statement

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b

- Less than or equal to: a <= b
- a > b
- a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops. An "if statement" is written by using the if keyword.

5.1 If Statement

If statement:

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

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

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

```python
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

5.2 Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

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

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

5.3 Else

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

```python
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")
```

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

You can also have an else without the elif:

```python
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

5.4 Nested If

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

```python
x = 41
if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

6 Loop

Most languages have the concept of loops: If we want to repeat a task twenty times, we dont want to have to type in the code twenty times, with maybe a slight change each time. As a result, we have for and while loops in python.

6.1 For Loop

Iterating over a sequence is done with a **for** loop (that is either a list, a tuple, a dictionary, a set, or a string).

6.1.1 Implementation in Python

```python
# range(5) is not the values of 0 to 5, but the values 0 to 4.
for i in range(5):
  print(i)

-----------------------

# range(2, 5), which means values from 2 to 5 (but not including 5)
for i in range(2, 5):
  print(i)

-----------------------

# Increment the sequence with 3 (default is 1):
for i in range(2, 20, 3):
  print(i)

-----------------------

# Here, else keyword indicates a block of code to be executed when the loop is
    finished:
for i in range(5):
  print(i)
else:
  print("Finally finished!")

```

```
------------------------

fruits = ["orange", "apple", "cherry"]
for i in fruits:
  print(i)

------------------------

# Loop through the letters in the word "orange":
for i in "orange":
  print(i)

------------------------

# Loop through the letters in the word "orange":

fruits = ["orange", "apple", "cherry"]
for i in fruits:
  print(i)
    if i == "apple":
        break
```

6.1.2 Lab Task (Please implement yourself and show the output to the instructor)

- Write a python program to find the sum of odd and even numbers from a set of numbers.
- Write a python program to find the smallest number from a set of numbers.
- Write a python program to find the sum of all numbers between 50 and 100, which are divisible by 3 and not divisible by 5.
- Write a python program to find the second highest number from a set of numbers.
- Write a python program to find the factorial of a number using for loop.
- Write a python program to generate Fibonacci series.

6.1.3 While Loop

6.1.4 Implementation in Python

```
i = 1
while i < 5:
  print(i)
  i += 1

------------------------

i = 1
while i < 5:
  print(i)
  if i == 2:
    break
  i += 1

------------------------

i = 0
while i < 5:
  i += 1
  if i == 2:
    continue
  print(i)

-----------------------

i = 1
while i < 5:
  print(i)
  i += 1
else:
  print("i is no longer less than 5")
```

6.1.5 Lab Task (Please implement yourself and show the output to the instructor)

- Write a python program to find the sum of odd and even numbers from a set of numbers.
- Write a python program to find the smallest number from a set of numbers.
- Write a python program to find the sum of all numbers between 50 and 100, which are divisible by 3 and not divisible by 5.
- Write a python program to find the second highest number from a set of numbers.
- Write a python program to find the factorial of a number using for loop.
- Write a python program to generate Fibonacci series.

7 Function

Functions enable you to break down the overall functionality of a script into smaller, logical subsections, which can then be called upon to perform their individual tasks when needed. Using functions to perform repetitive tasks is an excellent way to create code reuse. This is an important part of modern object-oriented programming principles.
In Python a function is defined using the **def** keyword:

7.1 Implementation in Python

```
def my_function():
  print("Hello World")

my_function()   # Calling my_function()
-----------------------

# Passing Argument
def my_function(name):
  print("name " + name)

my_function("Mike")
my_function("Monica")
my_function("John")

-----------------------

# Passing multiple Arguments
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

```python

------------------------

# Arbitrary Arguments, *args
def my_function(*names):
  print("Name= " + names[2])

my_function("Mike", "Monica", "John")

------------------------

# Send arguments with the key = value syntax.
def my_function(name3, name2, name1):
  print("Name= " + names3)

my_function(name1 = "Mike", name2 = "Monica", name3 = "John")

------------------------

def my_function(**name):
  print("His last name is " + name["lname"])

my_function(fname = "Mike", lname = "Monica")

------------------------

# Passing a List as an Argument
def my_function(food):
  for i in food:
    print(i)

fruits = ["orange", "apple", "cherry"]

my_function(fruits)

------------------------

# Return Values
def my_function(a):
  return 5 * a

print(my_function(3))
print(my_function(5))
print(my_function(9))

------------------------

# Recursion Example
def my_recursion(k):
  if(k > 0):
    result = k + my_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
my_recursion(6)
```

7.2 Lab Task (Please implement yourself and show the output to the instructor)

- Write a python program to find the largest number between two numbers using function
- Write a python program to find the sum of the numbers passed as parameters.

8 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.