PYTHON INTERNSHIP MANUAL

Batch A - 2025 **Week#1 (Basic Python)**

Important Instructions:

- Read the manual carefully and understand every topic before starting the assignment.
- Follow the deadlines properly; marks will be deducted for late submissions.
- Avoid to use AI Tools and make your own logic.
- Completing all tasks in the assignment is compulsory to receive full marks.

- **Manual Posted Date**:

  5th March, 2025 (Wednesday)

- **Assignment-1 Deadline:**

  10th March, 2025 (Monday at 11:59pm)

1. Introduction to Python (Theory)

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.

Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to www browsers to games.

2. Python Program (Print Statement)

If you are running new version of Python, then you would need to use print statement with parenthesis. However, in Python version 2.4.x, you do not need the parenthesis. The above line produces the following result:

For Example,

Python print statement code

```
print("Hello World!")
```

Output

```
Hello World!
```

3. Quotations in Python

Python accepts *single, double and triple quotes* to denote string literals, as long as the same type of quote starts and ends the string. The triple quotes are used to span the string across multiple lines.

For example, all the following are legal:

Single Quote ('   ')

```
word = 'Ahmed'     # Single quote is use for a word ' '
```

Double Quote ("   ")

```
sentence = "My Name is Ahmed"     # Double quote is use for a sentence " "
```

Triple Quote ('''   ''') or ("""   """)

```
paragraph = '''My Name is Ahmed. I am doing Internship'''  # Triple quotes used for paragraph '''   '''

paragraph = """My Name is Ahmed. I am doing Internship"""  # Both can be use for paragraph """   """
```

- A single word is written in Single quote.
- A single sentence is written in Double quotes.
- A paragraph is written in Tripple quotes.

4. Comments in Python

A hash sign (#) begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

For example,

```
sentence = "My Name is Ahmed"       # sentence has defined

print("Hello World!")         # Hello world! is printing on screen
```

sentence has defined and # Hello world! is printing on screen (These are the comments above that can be used in the code)

5. Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variables.

```
a = 100      # Assigning a Integer value to the variable (a).

b = 100.2   #Assigning a Float value to the variable (b).

temp = 'Laptop'   #Assigning an String to the created variable (temp)
```

6. Python Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them. Python has five standard data types:

- Numbers
- String
- List
- Tuple
- Dictionary

1. Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them.

```
a = 100      # Assigning a Integer Number

b = 100.2   #Assigning a Float Number
```

2. Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

```
str = 'Hello World!'
print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])# Prints string starting from 3rd character
print (str* 2) # Prints string two times
print (str + "TEST") # Prints concatenated string
```

This will produce the following results.

```
Hello World!
H
llo
llo World!
Hello World!Hello
World!
```

3. Python Lists

The list is the most versatile data type available in Python, which can be written as a list of comma separated values (items) between square brackets. Important thing about a list is that the items in a list need not be of the same type. Creating a list is as simple as putting different comma-separated values between square brackets.

```
list1 = ['physics', 'chemistry', 1997, 2000]  # A list can have Integer, Float or String values (all data type values)

list2 = [1, 2, 3, 4, 5, 6, 7]

print("The element at index 0 = ", list1[0])

print("The elements from index 1 to 5 = ", list2[1:5])
```

This will produce the following results

```
The element at index 0 = 1


The elements from index 1 to 5 = [2,3,4,5]
```

Updating Lists

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() methods.

```
list = ['physics', 'chemistry', 1997, 2000]
print ("Value available at index 2 : ", list[2])

list[2] = 2001

print ("New value available at index 2 : ", list[2]) # Index2 of list has been overwrite with 2001
```

Deleting Element from List

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting. You can use the remove() method if you do not know exactly what is the index of the item to delete.

```
list = ['physics', 'chemistry', 1997, 2000]

del list[2]  # Deleting the index 2 (element 3 = 1997)

print ("After deleting value at index 2 : ", list) #Index 2 has been deleted and now list has total 3 values .
```

4. Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parenthesis. The main difference between lists and tuples are:
Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists.

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print (tuple)           # Prints complete tuple
print (tuple[0])        # Prints first element of the tuple
print (tuple[1:3])# Prints elements starting from 2nd till 3rd
print (tuple[2:]) # Prints elements starting from 3rd element
print (tinytuple * 2)     # Prints tuple two times
print (tuple + tinytuple) # Prints concatenated tuple
```

This will produce the result give below

```
('abcd', 786, 2.23, 'john', 70.200000000000003)
abcd (786, 2.23)
(2.23, 'john', 70.200000000000003)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')
```

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists.

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
list = [ 'abcd', 786 , 2.23, 'john', 70.2]
tuple[2] = 1000    # Invalid syntax with tuple
list[2] = 1000     # Valid syntax with list
```

5. Python Dictionary

Python's dictionaries are kind of hash-table type. They consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

```
dict = {}
dict['one'] = "This is one"
dict[2]= "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
print (dict['one'])           # Prints value for 'one'
key print (dict[2])           # Prints value for 2
key print (tinydict)          # Prints complete dictionary print
(tinydict.keys())             # Prints all the keys
print (tinydict.values())            # Prints all the values
```

7. Decision Making in Python

Decision-making is the anticipation of conditions occurring during the execution of a program and specified actions taken according to the conditions. Decision structures evaluate multiple expressions, which produce TRUE or FALSE as the outcome. You need to determine which action to take and which statements to execute if the outcome is TRUE or FALSE otherwise. Following is the general form of a typical decision making structure found in most of the programming languages.

IF Satements

The IF statement is similar to that of other languages. The if statement contains a logical expression using which the data is compared and a decision is made based on the result of the comparison.

Basic IF-ELSE Structure

```
if expression:
   statement(s)
else:
      statement(s)
```

IF - ELIF - ELSE Statements

An else statement can be combined with an if statement. An else statement contains a block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value. The else statement is an optional statement and there could be at the most only one else statement following if.

```
var = 100    #Initialize the variable with a value
if var < 100:                                    #Checking the Condition
     print("Variable value is less than 100")

elif var > 100:
    print("Variable value is greater than 100")

else:
   print("Value is 100")
```

The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
----------------------------------------------------------------------------------------------------------------------------------------------------
amount = int(input("Enter amount: "))

if amount<1000:
    discount = amount*0.05
    print ("Discount",discount)

elif amount<5000:
    discount = amount*0.10
    print ("Discount",discount)
```

8. Loops in Python

In general, statements are executed sequentially − The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement.

While Loop statement

In Python programming language repeatedly executes a target statement as long as a given condition is true.

```
count = 0
while (count < 9):                    #The count will start with 0 and increment till become 9
      print ("The count is ", count)
      count = count + 1

# When count = 9, condition false and loop exit or terminate.
# Output will be:

The count is 0    # Count start from 0
The count is 1
The count is 2
The count is 3
The count is 4
The count is 5
The count is 6
The count is 7
The count is 8    #Count should be less than 9
```

For Loop Statements

The for statement in Python has the ability to iterate over the items of any sequence, such as a list or a string. For Loop is used for iteration on any specific times or condition same as while Loop.

General For Loop Structure

```
for var in list(range(5)):

            print (var)   # Saving every index element starting from 0 in var variable and print on screen
```

```
str = 'Python'

for letter in str:
    print ("Current Letter :", letter)              # Traverse the every element of a single string and print on screen

# Output for the above code:

Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n

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

fruits = ['banana', 'apple', 'mango']

for fruit in fruits:
        print ('Current fruit :', fruit)              # Traverse the every element of list and print on screen
        print ("Good bye!")

# Output for the above code:

Current fruit : banana
Current fruit : apple
Current fruit : mango
```