1. PYTHON: Introduction

Python is a programming language. It's used for many different applications. It's used in some high schools and colleges as an introductory programming language because Python is easy to learn, but it's also used by professional software developers at places such as Google, NASA, and Lucasfilm Ltd.

By the way, the language is named after the BBC show "Monty Python's Flying Circus" and has nothing to do with reptiles. Making references to Monty Python skits in documentation is not only allowed, it is encouraged!

Features:

It is compact and very easy to use OOP language.
It is more capable to express the purpose of the code.
It is interpreted line by line.
No need to download additional libraries.
It can run on variety of platform. Thus it is a portable language.
It is free and open source.
Variety of applications
Python allows you to split your program into modules that can be reused in other Python programs.
Programs written in Python are typically much shorter than equivalent C, C++, or Java programs, for several reasons:

- the high-level data types allow you to express complex operations in a single statement;
- statement grouping is done by indentation instead of beginning and ending brackets;
- no variable or argument declarations are necessary.

Disadvantages:

- Python is not the fastest language.
- Its library is not still competent with other language like C, Perl and JAVA.
- Python interpreter is not very strong on catching 'Type-Mismatch'.
- It is not easy convertible to other programming language.

Typing Python in a text Editor:
Install Sublime Text 3 editor in the machine where you work

PYTHON fundamentals

Character set:

Letters: alphabets like A-Z and a-z
Numbers: 0-9
Special symbols: + - , * / ** \ [] {} () // = != == < , > . ' "" ; : % !
Whitespaces: Blank spaces, tabs, carriage return, newline, formfeed etc.
Other characters: All ASCII and UNICODE characters.

Tokens:

The smallest individual unit of program is known as lexical unit or token.
Python has following tokens:

A. Keywords
B. Identifiers
C. Literals
D. Operators
E. Punctuators

A. **Keywords:** Keywords are special meaning and it is reserved by the programming language. Python has the following keywords.

false	Assert	del	for	in	or	while
none	Break	elif	from	is	pass	with
true	class	eElse	global	lamba	raise	yield
and	continue	except	if	nonlocal	return	

B. Identifiers: It is the name given by the user for declaring variables, constants, objects, classes, functions etc.

Naming rules of Identifiers are:

1. An identifier is an arbitrarily long sequence of letters and digits.
2. The first character must be a letter; the underscore count as a letter.
3. Python is a case sensitive.
4. Digits can be used in between.
5. It must not be a keyword.
6. No special character except underscore is allowed.

C. Literals/ Values

Literals are data items that have a fixed value. The types of literal are:

(i) String
(ii) Numeric
(iii) Boolean
(iv) Special Literal None
(v) Literal collection

(i) ***String***: String literals are formed using **single quote** or **double quote** in Python.
Python allows you to have certain non-graphic characters in string values. Non-graphic characters are those characters that cannot be typed directly from keyboard e.g. backspace, tab, spacebar, carriage return etc. These non graphic characters are represented by using escape sequences by using back slash (\) followed by one or more characters. Some escape sequences are; \a, \b, \\, \', \", \n, \f etc.

Types of String in Python:
(a) Single line String
(b) Multiple line String

(a) **Single Line String:** These strings are terminated in one line and enclosed within single quote or double quote. Example:
Str1 = 'Good
Morning'
The above example will show an error. To rectify it add a slash \ at the end of the first line
Str1 = 'Good \
Morning'

(b) **Multiple line string**: Multiple string can be created using two ways:
a. By adding a backslash
Str1 = 'Good \
Morning'

b. By typing the text in triple quotation mark
Example:
Str2= """ Hello
All of you
Good Morning."""

Size of String:
'\\' Size is 1
'xyz' Size is 3
"\ab" Size is 2
"Namrata\'s pen" Size is 13

(ii) **Numeric Literals**: The numeric literals in Python are:
a. int: it represents positive or negative whole numbers.
b. float: It represents real or the numbers with decimal point.
c. Complex: These numbers are in the form of a + bj where j is √-1 (Which is an imaginary numbers.)

(iii) **Boolean Literal:** The Boolean literal is used to represent one of the two Boolean values i.e. True or False.

(iv) **Special Literal None**: The special literal None is used to represent absence of the value.

D. Operators: Operators are symbols used for calculations. The following operators are used in Python:

a. Unary Operators
   i. + Unary Plus
   ii. -Unary Minus
   iii. ~ Bitwise complements
   iv. Not Logical operator
b. Binary Operators
   i. + Addition
   ii. - Subtraction
   iii. * Multiplication
   iv. / Division
   v. % Remainder
   vi. ** Exponent (raise to power)
   vii. // Floor division
c. Bitwise Operator
   i. & Bitwise AND
   ii. ^ Bitwise Exclusive OR
   iii. | Biwise OR
d. Shift Operators
   i. << shift Left
   ii. >>shift right
e. Identity operators
   i. Is is the identity same?
   ii. Is not is the identity not same?
f. Relational Operators
   i. <
   ii. >
   iii. <=
   iv. >=
   v. ==
   vi. !=
g. Logical Operators
   i. And Logical AND
   ii. Or Logical OR
h. Assignment operators
   i. =
   ii. /=
   iii. +=
   iv. -=
   v. %=
   vi. *=
   vii. **=
   viii. //=
i. Membership Operators
   i. In Whether variable in sequence
   ii. Not in whether variable is not in sequence.

E. Punctuators: Most commonly used punctuators in Python are:
‘ “ # \ () {} [] @ , : . =

Variable and Assignments:

A named memory location that refers to a value and whose value can be used and processed during program run is called variable.

Creating a variable:

```
Value = 50              # integer variable
Name = ‘AMIT’           # String variable
Amount = 123.35         # Floating point variable
```

Note: Variables are not Storage Containers in Python. The variable in Python does not have fixed locations unlike other programming languages. The location they refer to change every time their values change.

Lvalues and Rvalues:

Lvalue: it is the expression that can comes on the lhs of an assignment.
Rvalue: It is the expression that comes on the rhs of the assignment.

Multiple assignments:

You can assign the same value to the multiple variables in the following way:
x = y = z = 20
Assigning multiple values to multiple variables:
x,y,z = 40,50,60

Dynamic Typing

If you assign a value to a variable and later another value of different data type you assign to the same variable, it doesn't give any error.

Simple Input and Output:

In Python, to get input from the user, an inbuilt function input () is used as follows:

Sname= input(" Students name:")
The input function always return the data of type string.

Reading number; As we know that input() returns string value therefore for numeric operations we need to convert these values into its int or float form using int() or float() function. For example:

amount = input("Enter amount")

Amt=float(amount)

Output through print Statement:

The syntax of using print() function of PYTHON is:
print(*Object, [sep = '' or <separator string> end = '\n' or <end-string.])

*Object means it can be one or more multiple comma separated object to be printed.
Example: print(22+5, "Year Old")

Exercise: Write questions and answers of you text book in class notes copy.

2. Data Handling

This chapter includes more about data types, variables, operators and expressions used in Python.

Data Types

Data are used as integer, float and character or string type. Python offers following built in core data types:
(i)Number (ii) String (iii) List (iv) Tuple (v) Dictionary

(i) **Number:** The numbers in Python have following core data types:

a. *Integers*
- Integers Signed: Integers in Python can be of any length, it is only limited by the memory available. It is signed representation i.e. it can be positive as well as negative.
- Boolean: It represents false or true which behaves like 0 and 1. To get the Boolean equivalent of 0 or 1, you can type bool(0) or bool(1), Python will return false or true respectively.

b. *Floating Point:*
The Fractional numbers can be written in two forms:
- Fractional forms: Example: 5678.90, 34528.4532 etc.
- Exponent Form: 5.6789E03, 3.45284532E04

c. *Complex*: A complex number is a combination of real number and imaginary numbers. In complex number a+bi, a and b are real numbers whereas i is imaginary quantity which is represented by √-1. We can retrieve these two parts using attribute references. For complex number p: p.real will return real part of the complex number.
And p.imag will return the imaginary part as float, not as a complex number.

(ii) ***String***: A string data type lets you hold the string data which is any number of valid characters into a set of quotation mark.
Example: “pqrs”, ‘$$$’, “New Delhi”, “3456.89” etc.
Forward and backward indexing of any string value can be represented in the following manner:

0	1	2	3	4	5	6	7	8	9	**Forward Indexing**
I	N	F	O	R	M	T	I	C	S	
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	**Backward Indexing**

(iii) ***Lists and Tuples*:**
These are *compound data types* of Python therefore we have taken together to study. But there is one difference, ***list can be modified but tuples cannot be modifies***.
A list in the python represents a list of comma-separated values of any data type between square brackets.
The following are some examples of List:
[1,3,5,7,9]
[“Amit”, “Sumit”, “Rajiv”, “Gaurav”]
[‘Namrata’, 202,404,302]

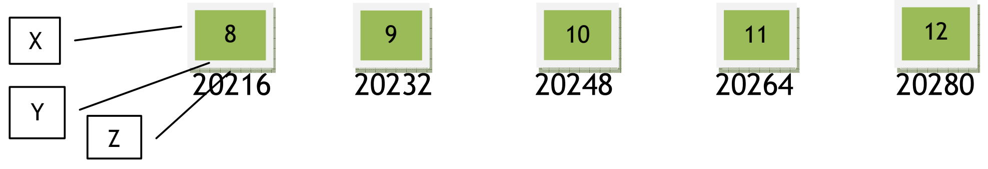
Tuples are represented as a list of comma separated values of any data types within parenthesis.
Example:
X= (1, 2, 3, 4, 5, 6, 7, 8)
Y= (‘a’, ‘e’, ‘i’, ‘o’, ‘u’)
List can be nested also. Example:
Head = [[‘A’, ‘B’, ‘C’, ‘D’, ‘E’,], [0,1,2,3,4,5,6,7]]
List = [‘x’, ‘y’, ‘z’]
List can be concatenated by using ‘+’ operator i.e.
>>> Head[1]+List would produce
[0,1,2,3,4,5,6,7,’x’, ‘y’, ‘z’]
To obtain the length of list following len command can be issue:
>>> print len(Head)
In addition, you can use the augmented addition assignment to add items to the list, but you must specify a list as the object to be added, as in the following example:
>>>List += [8]

(iv) ***Dictionary*:**
The dictionary is the list of unordered set of comma separated value with the combination of **key: value** pair. It is declared within parenthesis {}. For example:
V = {‘a’:1, ‘e’:2, ‘i’:3, ‘o’:4, ‘u’:5}

Mutable and Immutable Types:
The data types used in Python are broadly classified into two categories: Modifiable (Mutable) and Non-modifiable (immutable).

1. **Immutable Types:**
The immutable types are those that never change their values in place. Integer, float, Boolean, String and tuples come into this category.
Example:
X=8
Y=X
Z=8
It will produce 8,8,8
In python values to the variables are stored differently in a different manner. Each value is having a fixed defined memory location. When we define any variable for any given value, the variable is referred to the location where the value is stored and as we change the value of the variable, the reference of the memory location for the variable is changed. In Python it keeps a count internally to count that how many variables are referring a value.

(3)



You can check / confirm it yourself by using id(). It returns the memory address to which a variable is referencing.
In the above example

X=8
Y=X
Z=8

id(8)
id(X)
id(Y)
id(Z)
will give the same output because the values of all these identifiers are same.

2. **Mutable Types:**
   The mutable types are those whose values can be changed in place. These types are: **List, dictionaries and sets**.
   To change a member of a list, you may write:
   List1=[3,25,14]
   List1[1] = 80
   It will make the list namely List1 as [3,80,14]

Variable internal:
All data or values are referred to as object in Python.
In Python every object has three key attributes associated to it:
(i) The type of an object
(ii) The value of an object
(iii) The id of an object
Variable names are stored as reference to a value – object. Each time you change the value, the variable's reference memory address changes.

Operator
1. **Arithmetic Operators**
   a. Unary Operators
   b. Binary Operators
   c. Augmented Assignment Operators
      x+=y => x=x+y
      x**=y => x = x**y
2. **Relational Operators**
   <, >, <=, >=, ==, !=
3. **Identity Operators (is, is not)**
   Equality and Identity – Relation
   is => returns true if both its operands are pointing to the same object
   is not +> returns true if both operands are pointing to the different object.
4. **Logical Operators**
   a. OR operator
   b. AND operator
   c. NOT operator
5. **Bitwise Operators (&, |, ^, ~)**
   a. The AND operator and &
   b. The inclusive OR (|)
   c. The exclusive OR (^)
   d. The Complement Operator(~)

Expressions: Any valid combination of operators, literals and variables are known as expression. The expression in Python can be of any type:

1. Arithmetic Expressions
2. Logical Expressions
3. Relational Expressions
4. String Expressions

***Exercise*: Write questions and answers of you text book in class notes copy.**

3.Conditional Statements

Types of statements in Python:

1. Empty statement
2. Simple Statement
3. Compound statement

1. Empty Statement: A statement which does nothing is called empty statement in Python. Empty statement is pass statement. Whenever Python encounters a **pass** statement, Python does nothing and moves to the next statement in the flow of control.
2. Simple Statement: Any single executable statement is a simple statement.
   For example:
   Age = input (“Your Age?”)
   print(Age)
3. Compound Statement: It represents group of statements executed as a unit. It consists of **header** and **body.** The syntax is:
   <header>:
   <body containing multiple, simple or compound statement>

Statement Flow Control:
Programming statements are executed serially, selectively or iteratively.

Sequence: The execution of the statement follows the top to bottom approach.

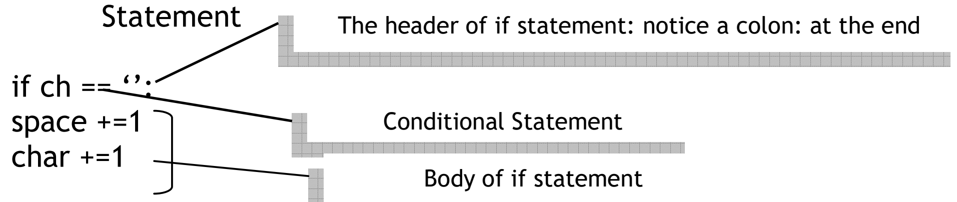
Selection: The execution of statement is based on the condition, if the condition evaluates to true, a group of statements are executed else other group of statements are executed.

Iteration (Looping): When the set of statements are executed repeatedly based on a condition, it is called iteration statement.

The *if* Statement of Python:

The if statement in Python constructs the selection statement. Its syntax is as shown below:

```
if<Conditional Expression>:
        Statement
```



```
if ch == ‘’:
space +=1
char +=1
```

Here in this case, if the conditional expression evaluates to true, the statements in the body of if are executed, otherwise ignored.

Example:

```
X = int(input(“Enter the value of x:”))
Y = int(input(“Enter the value of y:”))
 if A>20 and B<50:
        z=(x-y)*y;
        print(“The result is:”,z)
print(“Thank you”)
```

This statement is not the part of if as it is not indented at the same level as that of body of if statement.

<u>The if - else statement:</u>

This type of if statement test a condition and if the condition evaluates to true the body of if will be executed, otherwise else part will be executed.
Its Syntax is:

```
if <Condition>:
        statement
else
        statement
```

For example:

```
if age>=18:
print("You are eligible to get driving license")
else
print("You are not eligible")
print("Thank You")
```

Example: Program to check whether a given number is odd or even.

```
N1 = int(input("Enter an integer:")
if N1%2==0:
        print(N1, "is an Even Number")
else :
        print(N1, "is an Odd Number")
```

<u>Multiple if..elif</u>

Syntax:

```
if <condition1>:
        Statement1
elif<condition2>:
        Statement2
elif<condition3>:
        Statement3
else:
        Statement4
```

The nested if statement:

If an if condition is tested within another if condition then it is called nested if.
The syntax is(Nested within if):

```
if <conditional expression1>:
        If<conditional expression2>:
                Statements
        else:
                Statements
elif<conditional expression3>:
        statements
else:
        statements
```

Another Syntax(Nested within elif):

```
if <conditional expression1>:
        Statements
elif<conditional expression2>:
        If<conditional expression3>:
                Statements
        else:
                statements
else:
        statements
```

Related questions:
# Program to read three numbers and print them in ascending order.
#ABC shop deals in apparels and footwear. Write a program to calculate total selling price after levying the GDT. Do calculate central govt. GST and state govt. GST rates as applicable below:

Item	GST rate
Footwear<=500 (per pair)	5%
Footwear >500(per pair)	18%
Apparels <=1000(per piece)	5%
Apparels >=1000(per piece)	12%

Storing conditions:

The complex conditions can be stored under the name and used further, for example:

If percentage of marks is more than 90% and marks in maths is more than 80%, admission confirms in Maths stream.
If percentage of marks is more than 80% and marks in science is more than 80%, admission confirms in Bilogy stream.
If percentage of marks is more than 70% and marks in SocSc is more than 80%, admission confirms in Commerce stream.
Else admission closed for below 70% marks

The above statements can be written as follows:

```
MAths = per>=90 and Marks_Maths>80
Sc = per>=80 and Marks_Sc>80
Com=per>=70 and Marks_SSc>80
```

Now you can use these named conditionals in your coding as follows:

```
        if Maths:
                Stream="Mathematics"
        elif Sc:
                Srtream= "Science"
        elif Com:
                Stream = "Commerce"
        else :
                Stream = "No admission"
```

Repetition of Tasks:

Before learning the looping concept in PYTHON we should know the following range() function. The range() function of PYTHON generates a list which is a special sequence type. A sequence in PYTHON is a succession of values bound together by a single name. Some PYTHON sequence types are : string, list, tuples etc.

The syntax of range() function is:
range(<lower limit>,<upper limit>)   #both limit should be integers
Example: range(0,6) will produce a list as [0,1,2,3,4,5].
range(6,0) will return an empty list []
To produce a list with numbers having gap other than 1, then the syntax of range would be:
range(<lower limit>,<upper limit>,<step value>)
Example: range(0,8,2) will produce a list as [0,2,4,6]
range(6,0,-1) will produce a list as [6,5,4,3,2,1]

Another form of range is range(number) which produce a list from number 0 to the specified number in the range.
For example range(6) will produce a list as [0,1,2,3,4,5]

in and ***not in*** Operators in PYTHON:

These operators are used with range() function in for loop. To check whether a value is contained inside a list you can use in operator i.e.

3 in [0,1,2,3,4,5,6] will return True and 8 in [0,1,2,3,4,5,6] will return False.

Whereas 8 not in [0,1,2,3,4,5,6] will return True value.

These operators work with all sequence types i.e. string, tuples, list etc.

For example:
'S' in "Strength" will return True
Now consider the following code that uses in operator:

```
line = input("Enter a line:")
string=input("Enter a string:")

if string in line:
        print(string, "is part of", line)
else:
        print(string, "is not the part of", line)
```

Iterative Statements: These statements allow a set of instructions to be performed repeatedly until a certain condition is fulfilled. PYTHON provides two types of loops: for loop and while loop. These loops represents two categories of loops which are:

#Counting loops: The loop that repeat a certain number of times. Ex. *for* loop.

#conditional loop: The loop that repeat until a certain thing happens i.e. they keep repeating as long as some condition is true. Ex. *while* loop.

The ***for*** loop:

Syntax:

```
    for <variable> in <sequence>:
        statement to repeat
```

Example:

```
    for a in[1,4,7]:
        print(a)
        print(a*a)
```

Example: Printing multiplication table of a given number.

```
num=5
for a in range(1,11):
    print(num, ‘x’, a, ‘=’,num*a)
```

The above code will generate the multiplication table of 5.

Example: Program to print sum of natural numbers between 1 and 10.

```
sum=0
for n in range(1,11):
    sum+=n
    print(“The sum of first 10 natural number is”, sum)
```

The ***while*** loop:

Syntax:

```
    while <logical expression>:
        loop-body
```

Example:

```
    a=5
    while a>0:
        print(“Hello”,a)
        a=a-3
    print(“Loop Over!”)
```

Exercise: Write questions and answers of you text book in class notes copy.