UNIT I - INTRODUCTION TO DATA, EXPRESSIONS, STATEMENTS

Introduction to Python and installation, Python interpreter and interactive mode, debugging data types: Integer, float, Boolean, string, and list; variables, expressions, statements, precedence of operators, comments, modules.

INTRODUCTION

Python

Python is a high-level, interpreted, general-purpose programming language known for its readability and ease of use. Created by Guido van Rossum and first released in 1991, Python emphasizes code readability with its clear and concise syntax. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python's versatility allows its use in various domains, such as web development (server-side), software development, data analysis, machine learning, artificial intelligence, and system scripting. Its extensive standard library and vast ecosystem of third-party libraries contribute to its power and applicability across diverse tasks.

It is a programming language like C, C++ designed to be easy to read, and less time to write. It is an open source and it is an interpreter which directly executes the program without creating any executable file. Python is portable which means python can run on different platform with less or no modifications.

"A program is a sequence of instructions that specifies how to perform a computation".

Basic Instruction in Language

Input: Get data from Keyboard.

***Output*:** Display data on the screen

Math: Perform the mathematical operations.

***Conditional Execution*:** Check condition and execute certain code.

Repetition: Perform some action repeatedly.

Features of Python:

- Python is publicly available open source software.
- Python is easy to learn.
- Python is easy to read.

- Python is easy to maintain.
- Python provides automatic memory management.
- Python is portable.
- Python support database and GUI(Graphical User Interface)

Installing Python on Windows

To install Python, follow these steps:

Download Python: Visit the official Python website at python.org/downloads.

Select Version: Choose the latest stable version of Python 3 for your operating system (Windows, macOS, or Linux).

Run Installer:

Windows: Execute the downloaded installer. Crucially, during the installation process, ensure you check the box that says "Add Python to PATH" before proceeding with the installation. This step allows you to run Python directly from your command prompt or terminal.

macOS: Python often comes pre-installed on macOS, but it might be an older version (Python 2). For Python 3, it's recommended to install it via Homebrew (brew install python3) or by using the official installer from python.org.

Linux: Python is typically pre-installed on most Linux distributions. If you need a specific version or it's not present, you can install it using your distribution's package manager (e.g., sudo apt install python3 on Debian/Ubuntu, sudo dnf install python3 on Fedora).

Verify Installation: Open your terminal or command prompt and type the following command:

```
python --version
```

or, for specific Python 3 installations:

```
python3 –version
```

If Python is installed correctly and added to your system's PATH, this command will display the installed Python version.

Installing Python on Linux

Linux distributions often have Python pre-installed or readily available through their package managers.

- **Check for Existing Installation:** Open a terminal and type python3 --version. If Python 3 is already installed, its version will be displayed.
- **Install via Package Manager (Recommended):**

Debian/Ubuntu-based systems:

```
sudo apt update
sudo apt install python3
sudo apt install python3-pip # For installing Python packages
```

Fedora/CentOS-based systems.

```
sudo dnf install python3
sudo dnf install python3-pip # For installing Python packages
```

Verify Installation: After installation, type python3 --version in the terminal to confirm.

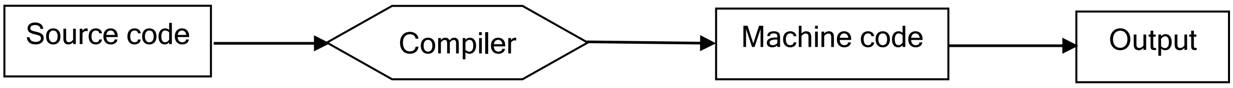
Note: While building Python from source is an option on Linux, using the package manager is generally simpler and recommended for most users.

PYTHON INTERPRETER AND INTERACTIVE MODE

Compilation in Python:

- When a Python script (.py file) is executed, the Python interpreter first compiles the source code into an intermediate format called bytecode. This bytecode has a .pyc extension and is platform-independent.
- This compilation step is performed by a component within the Python interpreter, which can be thought of as a compiler for Python.
- The purpose of compiling to bytecode is to optimize subsequent executions, as the bytecode can be loaded and executed more quickly than recompiling the source code each time.

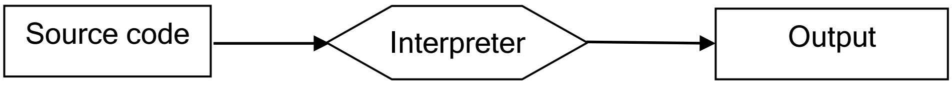
How Compiler Works



Interpretation in Python:

- The generated bytecode is then executed by the Python Virtual Machine (PVM), which acts as an interpreter.
- The PVM translates the bytecode instructions into machine code specific to the underlying hardware and executes them line by line.
- This real-time translation and execution of bytecode by the PVM is the "interpretation" aspect of Python's execution.

How Interpreter works

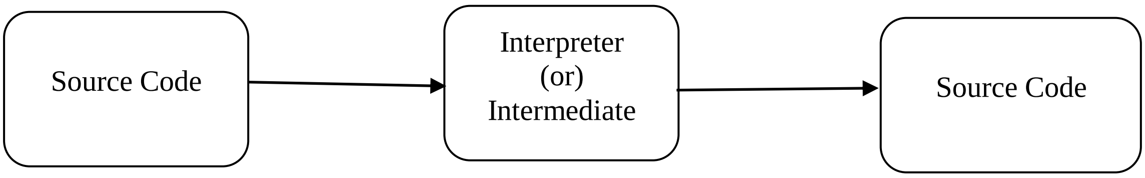


Difference Complier and Interpreter

S.No	Compiler	Interpreter
1	The compiler takes a program as a whole and translates it.	Interpreter translates a program statement by statement.
2	Intermediate code or machine code is generated in case of a compiler	Interpreter does not create intermediate code.
3	A compiler is comparatively faster than interpreter as the compiler take the whole program.	Interpreters compile each line of code after the other. So it is slow.
4	In compiler when an error occurs in the program, it stops its translation and after removing error whole program is translated again	When an error takes place in the interpreter it prevents its translation and after removing the error, translation resumes.
5	In a compiler the process requires two steps in which firstly source code is translated to target program then executed.	Interpreter is a one step process in which source code is compiled and executed at the same time.
6	The compiler is used in programming languages like C, C++, etc.,	Interpreter is used in languages like PHP, python etc.,

Python Interpreter

Python Interpreter translates high level instruction into an immediate form of machine level language. It executes instructions directly without compiling.



The Python interpreter is usually installed in C:/Program Files/Python3.6. In windows operating python can also be found in the start menu. All Programs→ Python 3.6 → Python 3.6 (Shell) and Python IDLE.

Python Interactive mode

- Interactive mode is a command line which gives immediate feedback for each statement. Python interactive mode can be start by two methods - Python 3.6 (Shell) and Python IDLE.
- Python 3.6 (Shell), A prompt will appear and it usually have 3 greater than signs (>>>). Each Statements can be enter after this (>>>) symbol. For continuous lines three dots (…) will represent the multiple lines.
- Python IDLE (Integrated Development for Learning Environment) which provides a user friendly console to the python users. Different colors are used to represent different keywords.
- IDLE starts by python version, after a line (>>>) three greater than symbols will be displayed. The statements will be executed in that line.

Example:

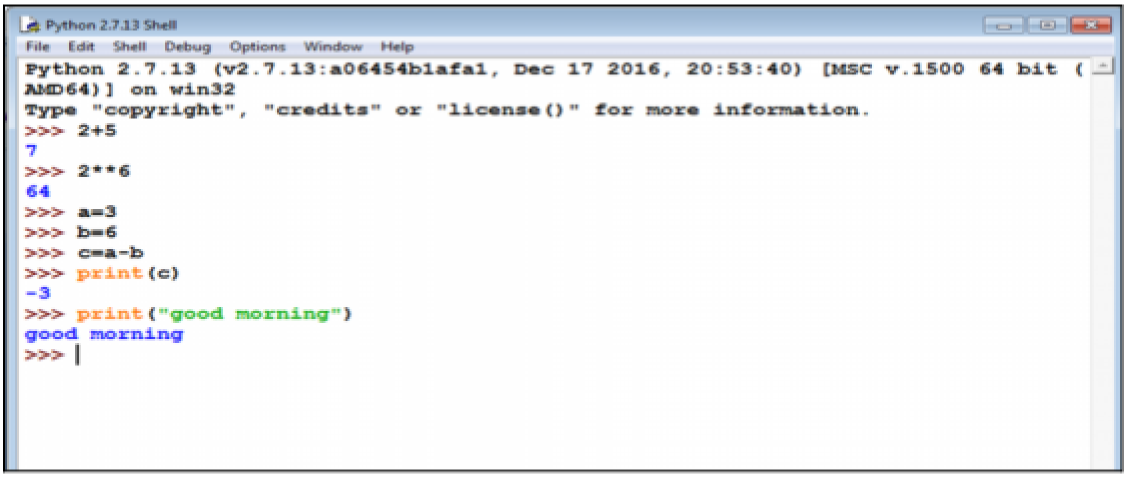
>>> 1+1

2

>>>5+10

15

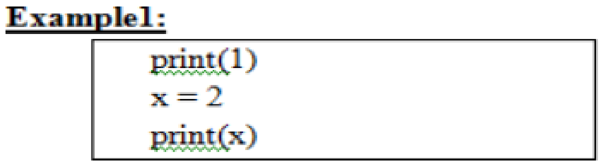
This is an example of a print statement. It displays a result on the screen. In this case, the result is the words.



```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:53:40) [MSC v.1500 64 bit (
AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+5
7
>>> 2**6
64
>>> a=3
>>> b=6
>>> c=a-b
>>> print(c)
-3
>>> print("good morning")
good morning
>>> 
```

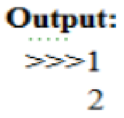
In Script mode

- In script mode, type python program in a file and store the file with .py extension and use the interpreter to execute the content of the file which is called a script.
- Working in script mode is convenient for testing small piece of code because you can type and execute them immediately, But for more than few line we can use script since we can modify and execute in future.



Example1:

```
print(1)
x = 2
print(x)
```



Output:

```
>>>1
    2
```

S.No	Interactive mode	Script mode
1	A way of using the python interpreter by typing commands and expressions at the prompt.	A way of using the python interpreter to rad and execute statements in a script.
2	Can't save and edit the code.	Can save and edit the code.
3	If we want to experiment with the code, we can use interactive mode.	If we are very clear about the code, we can use script mode.
4	We cannot save the statements for further use and we have to retype all the statements to re-run them	We can save the statements for further use and we no need to retype all the statements to re-run them.
5	We can see the results immediately	We can't see the code immediately

Debugging

Programming is error –prone, programming errors are called bugs and process of tracking them is called debugging.

Three kinds of error can occur in a program:

- Syntax Error
- Runtime Error
- Semantic error.

Syntax error:

- Syntax refers to the structure of a program and rules about the structure. Python can only execute a program if the syntax is correct otherwise the interpreter display an error message.

Runtime Error:

- The error that occurs during the program execution is called run time error.

Semantic Error:

- The computer will not generate any error message but it will not do the right thing since the meaning of the program is wrong.

Integrated Development Environment (IDE)

We can use any text editing software to write a Python script file. We just need to save it with the .py extension. IDE is a piece of software that provides useful features like code hinting, syntax highlighting and checking, file explorers etc. to the programmer for application development.

IDLE is a graphical user interface (GUI) that can be installed along with the Python programming language.

Example:

- Type the following code in any text editor or an IDE and save it as helloWorld.py print("Hello world!")
- To run the script, type python helloWorld.py in the command window and the output as follows:

```
Hello world!
```

DEBUGGING DATA TYPES: INTEGER, FLOAT, BOOLEAN, STRING, AND LIST

A datatype is a category for values and each value can be of different types. There are 7 data types mainly used in python interpreter.

a) Integer
b) Float
c) Boolean
d) String
e) List
f) Tuple
g) Dictionary

a) Integer

Let Integer be positive values, negative values and zero.

Example:

```
>>>2+2
4
>>>a=-20
print() → 20
>>> type(a) → <type ‘int’>
```

b) Float

A floating point value indicates number with decimal point.

Example:

```
>>> a=20.14
>>>type(a) → <type ‘float’>
```

c) Boolean

A Boolean variable can take only two values which are **True or False.** True and False are simply set of integer values of 1 and 0.The type of this object is bool.

Example:

```
>>>bool(1)
True
>>>bool(0)
False
>>>a=True
>>>type(a) → <type ‘bool’>
>>>b=false #Prints error
>>>c=’True’
>>>type(c) → <type ‘str’>
```

The boolean type is a subclass of the int class so that arithmetic using a boolean works.

```
>>>True + 1
2
>>>False * 85
0
```

# A Boolean variable should use Capital T in true & F in False and shouldn’t be enclosed within the quotes.

```
>>>d=10>45 → #which returns False
```

Boolean Operators

Boolean Operations are performed by ‘AND’, ‘OR’, ‘NOT’.

Example:

True and True → True

True and False → False

True or True → True

False or True →False

not False → True

d) String

A String is an ordered sequence of characters which can be created by enclosing characters in single quotes or double quotes.

Example:

```
>>>a=”Hello”
```

```
>>>type(a)
<type ‘str’>
```

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. The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Example:

```
>>>str = 'Python Programming'
>>>print(str)                # Prints complete string
>>>print(str[0])             # Prints first character of the string
>>>print(str[-1])            # Prints last 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 + " Course")    # Prints concatenated string
```

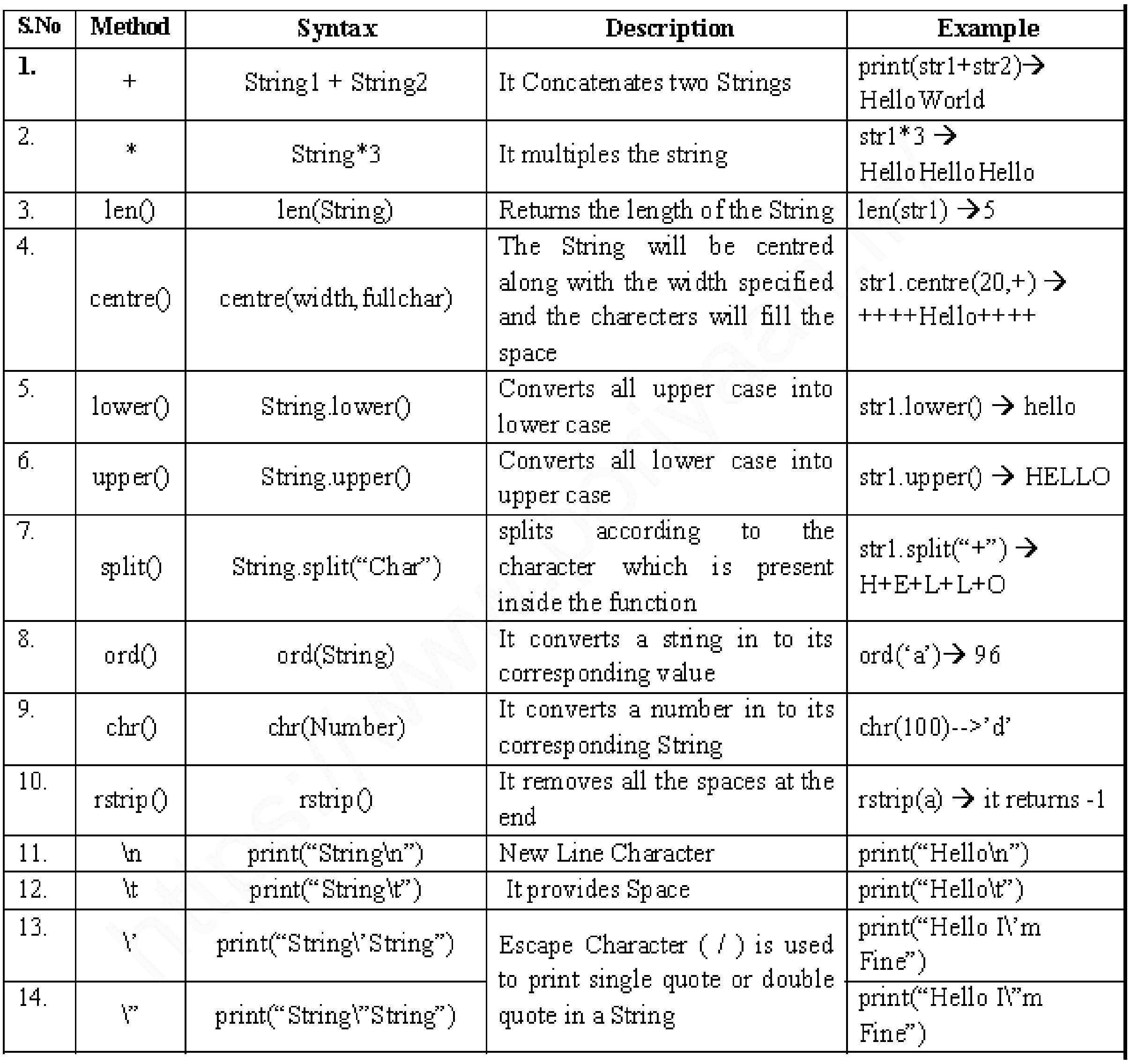
Output

```
Python Programming
P
g
tho
thon Programming
Python ProgrammingPython Programming
Python Programming Course
```

String Functions:

For the following string functions the value of ***str1*** and ***str2*** are as follows:

```
>>>str1=”Hello”
>>>str2=”World”
```



S.No	Method	Syntax	Description	Example
1.	+	String1 + String2	It Concatenates two Strings	print(str1+str2)→ HelloWorld
2.	*	String*3	It multiples the string	str1*3 → HelloHelloHello
3.	len()	len(String)	Returns the length of the String	len(str1) →5
4.	centre()	centre(width, fullchar)	The String will be centred along with the width specified and the charecters will fill the space	str1.centre(20,+) → ++++Hello++++
5.	lower()	String.lower()	Converts all upper case into lower case	str1.lower() → hello
6.	upper()	String.upper()	Converts all lower case into upper case	str1.upper() → HELLO
7.	split()	String.split("Char")	splits according to the character which is present inside the function	str1.split("+") → H+E+L+L+O
8.	ord()	ord(String)	It converts a string in to its corresponding value	ord('a')→ 96
9.	chr()	chr(Number)	It converts a number in to its corresponding String	chr(100)-->'d'
10.	rstrip()	rstrip()	It removes all the spaces at the end	rstrip(a) → it returns -1
11.	\n	print("String\n")	New Line Character	print("Hello\n")
12.	\t	print("String\t")	It provides Space	print("Hello\t")
13.	\'	print("String\'String")	Escape Character (/) is used to print single quote or double quote in a String	print("Hello I\'m Fine")
14.	\"	print("String\"String")		print("Hello I\"m Fine")

e) List

A list is an ordered set of values, where each value is identified by an index. The values that make up a list are called its elements. A list contains items separated by commas and enclosed within square brackets ([]). Lists are mutable which means the items in the list can be add or removed later.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

Example:

```
>>>list = [ 'Hai', 123 , 1.75, 'vinu', 100.25 ]
>>>smalllist = [251, 'vinu']
>>>print(list)          # Prints complete list
>>>print(list[0])       # Prints first element of the list
>>>print(list[-1])      # Prints last element of the list
>>>print(list[1:3])     # Prints elements starting from 2nd till 3rd
>>>print list([2:])     # Prints elements starting from 3rd element
>>>print(smalllist * 2) # Prints list two times
```

```
>>>print(list + smalllist) # Prints concatenated lists
```

Output

```
['Hai', 123, 1.75, 'vinu', 100.25]
Hai
100.25
[123, 1.75]
[1.75, 'vinu', 100.25]
[251, 'vinu', 251, 'vinu']
['Hai', 123, 1.75, 'vinu', 100.25, 251, 'vinu']
```

f) Tuple

Tuple are sequence of values much like the list. The values stored in the tuple can be of any type and they are indexed by integers. A tuple consists of a sequence of elements separated by commas. The main difference between list and tuples are:” List is enclosed in square bracket ([ ]) and their elements and size can be changed while tuples are enclosed in parenthesis (( )) and cannot be updated.

Syntax:

```
tuple_name = (items)
```

Example:

```
>>> tuple1=(‘1’,’2’,’3’,’5’)
>>>tuple2=(‘a’,’b’,’c’)
>>>tuple3=’3’,’apple’,’100’
>>>print(tuple2)              #print tuple2 elements
>>>print(tuple2[0])           #print the first element of tuple2
>>>print(tuple2 + tuple3)     #print the concatenation of tuple2 and tuple3
>>>print(tuple3[2])           #print the second element of tuple3
```

Output:

```
(‘a’,’b’,’c’)
(‘a’)
(‘1’,’2’,’3’,’5’,‘a’,’b’,’c’)
(’3’)
```

g) Dictionary

Dictionaries are an unordered collection of items. Dictionaries are enclosed by curly braces ‘{ }’ .The element in dictionary is a comma separated list of keys: value pairs where keys are usually numbers and strings and values can be any arbitrary python data types. The value of a dictionary can be accessed by a key. and values can be accessed using square braces ‘[ ]’

Syntax:

Dict_name = {key:value}

Example:

```
>>>dict1={}
>>>dict2={1:10,2:20,3:30}
>>>dict3={‘A’:’apple’,’B’:’200’}
>>>Dict={‘Name’:’john’,’SSN’:4576,’Designation’:’Manager’}
```

PYTHON KEYWORDS:

Keywords are reserved words that cannot be used as ordinary identifiers. All the keywords except True, False and None are in lowercase.

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

PYTHON IDENTIFIERS:

Identifiers are names for entities in a program such as class, variables and functions etc.

Rules for defining Identifiers:

Identifiers can be composed of uppercase, lowercase letters, underscore and digits bur should start only with an alphabet or an underscore.

- Identifiers can be a combination of lowercase letters (a to z) or uppercase letters (A to Z) or digits or an underscore.
- Identifiers cannot start with digit
- Keywords cannot be used as identifiers.
- Only (_) underscore special symbol can be used.

Valid Identifiers: sum total _ab_ add_1

Invalid Identifies: 1x x+y if

VARIABLES

A variable is nothing but a reserved memory location to store values. A variable in a program gives data to the computer.

Ex:

```
>>>b=20
>>>print(b)
```

PYTHON INDENTATION

Python uses indentation. Block of code starts with indentation and ends with the unintended line. Four whitespace character is used for indentation ans is preferred over tabs.

Ex:

```
x=1
if  x==1:
print("x is 1")
Result:
x is 1
```

EXPRESSIONS

An Expression is a combination of values, variables and operators.

Ex:

```
>>>10+20
```

STATEMENTS

A Statement is an instruction that a python interpreter can execute.IN python enf of a statement is marked by a newline character.

c=a+b

Multiline statement can be used in single line using semicolon(;)

```
>>a=1;b=10;c=a +b
```

Ex:

```
>>>b=20
>>>print(b)
>>>print("\"Hello\"")
```

Difference between a Statement and an Expression

A statement is a complete line of code that performs some action, while an expression is any section of the code that evaluates to a value. Expressions can be combined —horizontally into larger expressions using operators, while statements can only be combined vertically by writing one after another, or with block constructs. Every expression can be used as a statement, but most statements cannot be used as expressions

TUPLE ASSIGNMENTS

Tuple Assignment means assigning a tuple value into another tuple.

Ex:

```
t=('Hello','hi')
>>>m,n=t
>>>print(m) # Hello
>>>print(n) # hi
>>>print(t) # Hello,hi
```

In order to interchange the values of the two tuples the following method is used.

```
>>>a=('1','4')
>>>b=('10','15')
>>>a,b=b,a
>>>print(a,b) (('10','15'), ('1','4'))
```

COMMENTS

Comments are non-executable statements which explain what program does. There are two ways to represent a comment.

Single Line Comment

Begins with # hash symbol

Ex:

```
>>>print("Hello world")                # prints the string
```

Multi Line Comment

Multi line comment begins with a double quote and a single quote and ends with the same

Ex:

```
>>>"'This is a multi line comment'"
```

OPERATORS:

Operators are the construct which can manipulate the value of operands.

Eg: 4+5=9

Where 4, 5, 9 are operand

+ is Addition Operator

= is Assignment Operator

Types of Operator:

1. Arithmetic Operator
2. Comparison Operator (or) Relational Operator
3. Assignment Operator
4. Logical Operator
5. Bitwise Operator
6. Membership Operator
7. Identity Operator

1. Arithmetic Operator

It provides some Arithmetic operators which perform some arithmetic operations

Consider the values of a=10, b=20 for the following table.

Operator	Meaning	Syntax	Description
+	Addition	a+b	It adds and gives the value 30
-	Subtraction	a-b	It subtracts and gives the value -10
*	Multiplication	a*b	It multiplies and gives the value 200
/	Division	a/b	It divides and gives the value 0.5
%	Modulo	a%b	It divides and return the remainder 0
**	Exponent	a**b	It performs the power and return 10^{20}
//	Floor	a//b	It divides and returns the least quotient

Example Program:

1. ***Write a Python Program with all arithmetic operators***

```
>>>num1 = int(input('Enter First number: '))
>>>num2 = int(input('Enter Second number '))
>>>add = num1 + num2
>>>dif = num1 - num2
>>>mul = num1 * num2
>>>div = num1 / num2
>>>modulus = num1 % num2
>>>power = num1 ** num2
```

```
>>>floor_div = num1 // num2
>>>print('Sum of ',num1 ,'and' ,num2 ,'is :',add)
>>>print('Difference of ',num1 ,'and' ,num2 ,'is :',dif)
>>>print('Product of' ,num1 ,'and' ,num2 ,'is :',mul)
>>>print('Division of ',num1 ,'and' ,num2 ,'is :',div)
>>>print('Modulus of ',num1 ,'and' ,num2 ,'is :',modulus)
>>>print('Exponent of ',num1 ,'and' ,num2 ,'is :',power)
>>>print('Floor Division of ',num1 ,'and' ,num2 ,'is :',floor_div)
```

Output:

```
>>>
Enter First number: 10 Enter Second number 20 Sum of  10 and 20 is : 30
Difference of  10 and 20 is : -10
Product of 10 and 20 is : 200
Division of  10 and 20 is : 0.5
Modulus of  10 and 20 is : 10
Exponent of  10 and 20 is : 100000000000000000000
Floor Division of  10 and 20 is : 0
>>>
```

2. Comparison Operator (or) Relational Operator

These operators compare the values and it returns either True or False according to the condition. Consider the values of a=10, b=20 for the following table.

Operator	Syntax	Meaning	Description
==	a==b	Equal to	It returns false
!=	a!=b	Not Equal to	It returns true
>	a>b	Greater than	It returns false
<	a<b	Lesser than	It returns true
>=	a>=b	Greater than or Equal to	It returns false
<=	a<=b	Lesser than or Equal to	It returns true

3. Assignment Operator

Assignment operators are used to hold a value of an evaluated expression and used for assigning the value of right operand to the left operand.

Consider the values of a=10, b=20 for the following table.

Operator	Syntax	Meaning	Description
=	a=b	a=b	It assigns the value of b to a.
+=	a+=b	a=a+b	It adds the value of a and b and assign it to a.
- =	a-=b	a=a-b	It subtract the value of a and b and assign it to a.
*=	a*=b	a=a*b	It multiplies the value of a and b and assign it to

			a.
/=	a/=b	a=a/b	It divides the value of a and b and assign it to a.
%=	a%=b	a=a%b	It divides the value of a and b and assign the remainder to a.
**=	a**=b	a=a**b	It takes ‘a’ as base value and ‘b’ as its power and assign the answer to a.
//=	a//=b	a=a//b	It divides the value of a and b and takes the least quotient and assign it to a.

4. Logical Operator

Logical Operators are used to combine two or more condition and perform logical operations using Logical AND, Logical OR, Logical Not.

Consider the values of a=10, b=20 for the following table.

Operator	Example	Description
AND	if(a<b and a!=b)	Both Conditions are true
OR	if(a<b or a!=b)	Anyone of the condition should be true
NOT	not (a<b)	The condition returns true but not operator returns false

5. Bitwise Operator

Bitwise Operator works on bits and performs bit by bit operation.

Consider the values of a=60, b=13 for the following table.

Operator	Syntax	Example	Description
&	Binary AND	a&b= 12	It do the and operation between two operations
|	Binary OR	a|b= 61	It do the or operation between two operations
~	Binary Ones Complement	~a=61	It do the not operation between two operations
<<	Binary Left Shift	<<a	It do the left shift operation
>>	Binary Right Shift	>>a	It do the right shift operation

A	B	A&B	A|B	~A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0

1	1	1	1	0

1. Write a Python Program with all Bitwise Operator

```
a = 10                                          # 10 = 0000 1010
b = 20                                          # 20 = 0001 0100
c = 0                                           # 0 = 0000 0000
c = a & b;
print ("Line 1 - Value of c is ", c)            # 30 = 0001 1110
c = a | b;
print ("Line 2 - Value of c is ", c)            # -11 = 0000 1011
c = ~a;
print ("Line 3 - Value of c is ", c)            # 40 = 0011 1000
c = a << 2;
print ("Line 4 - Value of c is ", c)            # 2 = 0000 0010
c = a >> 2;
print ("Line 5 - Value of c is ", c)
```

<u>Output:</u>

```
Line 1 - Value of c is  12 Line 2 - Value of c is  61 Line 3 - Value of c is  -61 Line 4 - Value
of c is  240 Line 5 - Value of c is  15
```

6. <u>Membership Operator</u>

Membership Operator test for membership in a sequence such as strings, lists or tuples. Consider the values of a=10, b=[10,20,30,40,50] for the following table.

Operator	**Syntax**	**Example**	**Description**
in	value *in* String or List or Tuple	a in b returns True	If the value is '**in**' the list then it returns True, else False
not in	value *not in* String or List or Tuple	a not in b returns False	If the value is '**not in**' the list then it returns True, else False

<u>Example:</u>

x='python programming'

print('program' not in x) print('program' in x) print(' Program' in x) Output:

False True False

7. Identity Operator

Identity Operators compare the memory locations of two objects.

Consider the values of a=10, b=20 for the following table.

Operator	Syntax	Example	Description
is	variable 1 *is* variable 2	a is b returns False	If the variable 1 value **is** pointed to the same object of variable 2 value then it returns True, else False
is not	variable 1 *is not* variable 2	ot b returns False	If the variable 1 value **is not** pointed to the same object of variable 2 value then it returns True, else False

```
Example:
x1=7
y1=7
x2='welcome'
y2='Welcome'
print (x1 is y1)
print (x2 is y2)
print(x2 is not y2)

Output:
True
False
True
```

PRECEDENCE OF PYTHON OPERATORS

The combination of values, variables, operators and function calls is termed as an expression. Python interpreter can evaluate a valid expression. When an expression contains more than one operator, the order of evaluation depends on the Precedence of operations.

For example, Multiplication has higher precedence than Subtraction.

```
>>> 20 – 5*3 5
```

But we can change this order using Parentheses () as it has higher precedence.

```
>>> (20 - 5) *3
45
```

The operator precedence in Python are listed in the following table.

Table : Operator precedence rule in Python

S. No	Operators	Description
1.	()	Parentheses
2.	**	Exponent
3.	+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT

4.	*, /, //, %	Multiplication, Division, Floor division, Modulus
5.	+, -	Addition, Subtraction
6.	<<, >>	Bitwise shift operators
7.	&	Bitwise AND
8.	^	Bitwise XOR
9.	|	Bitwise OR
10.	==, !=, >, >=, <, <=,	mparison, Identity, Membership operators
11.	not	Logical NOT
12.	and	Logical AND
13.	or	Logical OR

ASSOCIATIVITY OF PYTHON OPERATORS

If more than one operator exists in the same group. These operators have the same precedence. When two operators have the same precedence, associativity helps to determine which the order of operations. Associativity is the order in which an expression is evaluated that has multiple operator of the same precedence. Almost all the operators have left-to-right

associativity. For example, multiplication and floor division have the same precedence. Hence, if both of them are present in an expression, left one evaluates first.

Example:

```
>>> 10 * 7 // 3
23
>>> 10 * (7//3) 20

>>> (10 * 7)//3 23
```

10 * 7 // 3 is equivalent to (10 * 7)//3.

Exponent operator ** has right-to-left associativity in Python.

```
>>> 5 ** 2 ** 3
390625
>>> (5** 2) **3 15625

>>> 5 **(2 **3) 390625
```

2 ** 3 ** 2 is equivalent to 2 ** (3 ** 2).

PRECEDANCE OF ARITHMETIC OPERATORS

Precedence	Operator	Description
1	**, ()	Exponent, Inside Parenthesis
2	/, *, %, //	Division, Multiplication, Modulo, Floor
3	+, -	Addition, Subtraction

FUNCTIONS

A **function** is a named sequence of statements that performs a specific task. Functions help programmers to break complex program into smaller manageable units. It avoids repetition and makes code reusable.

Types of Functions

Functions can be classified into

- BUILT-IN FUNCTIONS
- USER DEFINED FUNCTIONS

2.15.1 BUILT-IN FUNCTIONS

The Python interpreter has a number of functions that are always available for use. These functions are called built-in functions. The syntax is

function_name(parameter 1, parameter 2)

i) type()

```
>>>type(25)
<class 'int'>
```

The name of the function is type(). The expression in parentheses is called the argument of the function. The result, for this function, is the type of the argument. Function takes an argument and returns a result. The result is also called the return value.

Python provides functions that convert values from one type to another. The int() function takes any value and converts it to an integer, if it can, or it shows error otherwise:

ii) Casting:

```
>>>int('25') 25
>>>int('Python')
valueError: invalid literal for int(): Python
```

int() can convert floating-point values to integers, but it doesn’t round off; it chops off the fraction part:

```
>>>int(9.999999) 9
>>>int(-2.3)
-2
```

float() converts integers and strings to floating-point numbers:

```
>>>float(25) 25.0
>>>float('3.14159') 3.14159
```

Finally, str() converts its argument to a string:

```
>>>str(25)
'25'
```

i) *range()*

The range() constructor returns an immutable sequence object of integers between the given start integer to the stop integer.

Python's range() Parameters

The range() function has two sets of parameters, as follows:

ii) *range(stop)*

stop: Number of integers (whole numbers) to generate, starting from zero. eg. range(3) == [0, 1, 2].

iii) *range([start], stop[, step])*

start: Starting number of the sequence.

stop: Generate numbers up to, but not including this number. step: Difference between each number in the sequence.

Example:

```
>>>range(10) [0,1,2,3,4,5,6,7,8,9]

>>>range(5,10) [5,6,7,8,9]

>>>range[10,1,-2] [10,8,6,4,2]
```

iv) *Printing to the Screen*

print() function will prints as strings ,everything in a comma separated sequence of expressions, and it will separate the results with single blanks by default.

Example:

```
>>> x=10
>>> y=7
>>>print(‘The sum of’,x, ‘plus’, y, ‘is’, x+y)
```

Output:

```
The sum of 10 plus 7 is 17
```

print statement can pass zero or more expressions separated by commas.

v) *Reading Keyboard Input:*

Python provides a built-in function to read a line of text as a standard input, which by default comes from the keyboard. This function is:

- input()

The input() Function

The *input([prompt])* function print the string which is in the prompt and the cursor point will wait for an input.

```
>>>str = input("Enter your input: "); Enter your input: 10
```

```
>>> print(“The input is : ", str) 10
```

USER-DEFINED FUNCTIONS

If the user create own functions then these functions are called ***user-defined functions***.

Function Definition Syntax:

A function definition is a heart of the function where we will write main operation of that function.

Syntax:

```
def function_name(parameter 1, parameter 2): #Function Definition statements
```

Components of function definition

1. Keyword def marks the start of function header.
2. A function name to uniquely identify it.
3. Parameters (arguments) through which pass values to a function. They are optional.
4. A colon (:) to mark the end of function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have same indentation level.
7. An optional return statement to return a value from the function.

Example:

```
>>>def welcome(person_name):
"""This function welcome the person passed in as parameter""" print(" Welcome " , person_name , " to learn Python")
```

Using Function or Function Call

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

Syntax:

```
function_name(parameter 1, parameter 2)
```

Example:

```
>>> welcome('Students')
```

Output:

```
Welcome Students to learn Python.
```

The return statement:

The return statement is used to exit a function and go back to the place from where it was called.

Syntax:

```
return variable_name
```

This statement can contain expression which gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.

Example:

```
>>>def absolute_value(num):
"""This function returns the absolute value of the entered number""" if num >= 0:
return num
else:
return -num



>>>print(absolute_value(5))
>>>print(absolute_value(-7))
```

Output:

```
5
7
```

FLOW OF EXECUTION

The order in which statements run is **called the flow of execution**. Execution always begins at the first statement of the program. Statements are run one at a time, in order from top to bottom. Function definitions do not alter the flow of execution of the program, but when the function is called Instead of going to the next statement, the flow jumps to the body of the function, runs the statements there, and then comes back to pick up where it left off.

PARAMETERS AND ARGUMENTS

Inside the function, the a**rguments** are assigned to variables **called parameters**. Here is a definition for a function that takes an argument:

Function Arguments

Types of Formal arguments:

- Required arguments
- Default arguments
- Keyword arguments
- Variable-length arguments

Required Arguments

Required arguments are the arguments passed to a function in correct positional order.
Here, the number of arguments in the function call should match exactly with the function definition.

Example:

```
>>>def add(a,b):                # add() needs two arguments, if not it shows error return a+b

>>>a=10
>>>b=20
>>>print("Sum of ", a ,"and ", b, "is" , add(a,b))
```

Output:

```
Sum of 10 and 20 is 30
```

Default Arguments:

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

Example:

```
>>>def add(a,b=0):
print ("Sum of ", a ,"and ", b, "is" ,a+b)
>>>a=10
>>>b=20
>>>add(a,b)
>>>add(a)
```

Output:

```
Sum of 10 and 20 is 30
Sum of 10 and 0 is 10
```

Keyword Arguments:

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

Example:

```
>>>def add(a,b):
print ("Sum of ", a ,"and ", b, "is" ,a+b)
>>>a=10
>>>b=20
>>>add(b=a,a=b)
```

Output:

```
Sum of 20 and 10 is 30
```

Variable-Length Arguments:

The special syntax *args in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-keyworded, variable-length argument list.

- The syntax is to use the symbol * to take in a variable number of arguments; by convention, it is often used with the word args.
- What *args allows you to do is take in more arguments than the number of formal arguments that you previously defined. With *args, any number of extra arguments can be tacked on to your current formal parameters

Example:

```
>>>def myFun(*argv):
for arg in argv:
print (arg)
>>>myFun('Hello', 'Welcome', 'to', 'Learn Python')
```

Output:

```
Hello Welcome to
Learn Python
```

The Anonymous Functions or Lambda Functions

In Python, anonymous function is a function that is defined without a name. While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword. Hence, anonymous functions are also called lambda functions.

Syntax:

```
lambda arguments: expression
```

Example:

```
>>>double = lambda x: x * 2 print(double(5))
```

Output: 10

In the above program, lambda x: x * 2 is the lambda function. Here x is the argument and x * 2 is the expression that gets evaluated and returned.

The same Anonymous function can be written in normal function as

```
>>>def double(x):
return x * 2
>>>double(5)
```

MODULES

a) *Importing Modules*
b) *Built-in Modules*

Define Module

A module allows you to logically organize the python code. Grouping related code into a module makes the code easy to use. A module is a file consisting python code. A module can define functions, classes and variables. A module can also include runnable code.

Example:

The Python code for a module **add** normally resides in a file named **addition.py**.

support.py

```
>>>def add(a,b):
result=a+b return result
```

a) Importing Modules

We can invoke a module by two statements

i. import statement

ii. from…import statement

i) The import Statement

You can use any Python source file as a module by executing an import statement in some other Python source file.

Syntax:

```
import  module
```

Example:

```
import addition          # Import module addition
addition.add(22,33)    # Now we can call the function in that module as
```

Result:

```
Addition of two number is 55
```

ii) The from...import Statement

Python's from statement lets to import specific attributes from a module into the current namespace. Multiple function can be imported by separating by their names with commas .

Syntax:

```
from  module_name  import function_name
```

Example:

addition.py

```
>>>def add(a,b):
result=a+b
return result
```

subt.py

```
>>>def sub(a,b):
result=a-b
return result

>>>from example import add,sub
>>> print(addition.add(9,2)) 11
```

```
>>>print(subt.sub(5,2)) 3
```

b) **Built-in Modules**

Python have many built-in modules such as random, math, os, date, time, URLlib21.

i) *random module:*

This module is used to generate random numbers by using randint function.

Ex:

```
import random print(random.randint(0,5))

print(random.random())

print( random.random()*10)

my_data=[156,85,"john',4.82,True]

print(random.choice(my_data))
```

Output:

```
1
0.675788
5.2069
4.82
```

ii) *math module:*

This math module provides access to mathematical constants and functions.

Ex:

```
>>>import math
>>>math.pi                          #Pi, 3.14
>>>math.e                           #Euler's number
>>>math.degrees(2)                  #2 rads=114.59 degrees
>>>math.sin(2)                      #sin of 2
>>> math.cos(0.5)                   #cos of 0.5
>>> math.tan(0.23)                  #tan of 0.23
>>> math.sqrt(49)                   #sqrt of 49 is 7
>>>math.factorial(5)                #factorial of 5 is 1*2*3*4*5=120
```

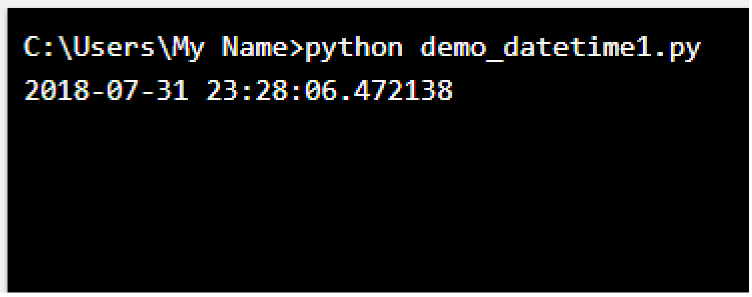
iii) *date & time module*

It is useful for web development. This is module is use to display date and time

Ex:

```
>>>import datetime
>>> x = datetime.datetime.now()
>>>print(x)
```

Output:



```
C:\Users\My Name>python demo_datetime1.py
2018-07-31 23:28:06.472138
```

iv) <u>Calender module:</u>

This module is used to display calendar

>>>import calendar

>>>cal=calendar.month(2019,5)

>>>print(cal)

<u>Output:</u>

Calendar is displayed

Part A:

1. What is interpreter?
2. What are the two modes of python?
3. List the features of python.
4. List the applications of python
5. List the difference between interactive and script mode
6. What is value in python?
7. What is identifier? and list the rules to name identifier.
8. What is keyword?
9. How to get data types in compile time and runtime?
10. What is indexing and types of indexing?
11. List out the operations on strings.
12. Explain slicing?
13. Explain below operations with the example

(i)Concatenation (ii)Repetition

14. Give the difference between list and tuple
15. Differentiate Membership and Identity operators.
16. Compose the importance of indentation in python.
17. Evaluate the expression and find the result

    (a+b)*c/d

    a+b*c/d

18. Write a python program to print ‘n’ numbers.
19. Define function and its uses
20. Give the various data types in Python
21. Assess a program to assign and access variables.
22. Select and assign how an input operation was done in python.
23. Discover the difference between logical and bitwise operator.

24. Give the reserved words in Python.
25. Give the operator precedence in python.
26. Define the scope and lifetime of a variable in python.
27. Point out the uses of default arguments in python
28. Generalize the uses of python module.

Part B

1. Explain in detail about various data types in Python with an example?
2. Explain the different types of operators in python with an example.
3. Discuss the need and importance of function in python.
4. Explain in details about function prototypes in python.
5. Discuss about the various type of arguments in python.
6. Explain the flow of execution in user defined function with example.
7. Illustrate a program to display different data types using variables and literal constants.
8. Show how an input and output function is performed in python with an example.
9. Explain in detail about the various operators in python with suitable examples.
10. Discuss the difference between tuples and list
11. Discuss the various operation that can be performed on a tuple and Lists (minimum 5)with an example program
12. What is membership and identity operators?
13. Write a program to perform addition, subtraction, multiplication, integer division, floor division and modulo division on two integer and float.
14. Write a program to convert degree Fahrenheit to Celsius
15. Analyze the difference between local and global variables.
16. Explain with an example program to circulate the values of n variables
17. Analyze with a program to find out the distance between two points using python.
18. Do the Case study and perform the following operation in tuples
    a) Maxima minima
    b) sum of two tuples
    c) duplicate a tuple
    d) Slicing operator
    e) obtaining a list from a tuple
    f) Compare two tuples
    g) printing two tuples of different data types
19. Write a program to find out the square root of two numbers.