Unit-1

Topics to be covered

Introduction: Introduction to Python, Program Development Cycle, Input, Processing, and Output, Displaying Output with the Print Function, Comments, Variables, Reading Input from the Keyboard, Performing Calculations, Operators. Type conversions, Expressions, More about Data Output.
Data Types, and Expression: Strings Assignment, and Comment, Numeric Data Types and Character Sets, Using functions and Modules.
Decision Structures and Boolean Logic: if, if-else, if-elif-else Statements, Nested Decision Structures, Comparing Strings, Logical Operators, Boolean Variables.
Repetition Structures: Introduction, while loop, for loop, Calculating a Running Total, Input Validation Loops, Nested Loops.

I. Introduction

Introduction to Python

Python was created by ***Guido Van Rossum***, who was a Dutch person from Netherlands in the Year 1991. Guido Van Rossum created it when he was working in National Research Institute of Mathematics and Computer Science in Netherlands. He thought to create a scripting language as a "Hobby" in Christmas break in 1980. He studied all the languages like ABC (All Basic Code), C, C++, Modula-3, Smalltalk, Algol-68 and Unix Shell and collected best features. He stared implementing it from 1989 and released first working version of Python in 1991. He named it as "Python", being a big fan of "***Monty Python's Flying Circus***" comedy show broadcasted in BBC from 1969 to 1974.

Python is a high-level, general-purpose programming language for solving problems on modern computer systems. The language and many supporting tools are free, and Python programs can run on any operating system. You can download Python, its documentation, and related materials from www.python.org.

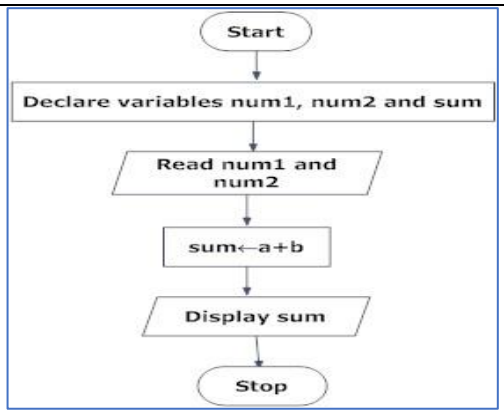
Program Development Cycle

The Program Development Cycle (PDC) has various states as follow:

a. **Problem Definition:** Here the formal definition o the problem is stated. This stage gives thorough understanding of the problem, and

all the related factors such as input, output, processing requirements, and memory requirements etc.

b. **Program Design:** Once the problem and its requirements have been identified, then the design of the program will be carried out with tolls like algorithms and flowcharts. An ***Algorithm*** is a step-by-step process to be followed to solve the problem. It is formally written using the English language. The ***flowchart*** is a visual representation of the steps mentioned in the algorithm. This flowchart has some set of symbols that are connected to perform the intended task. The symbols such as Square, Dimond, Lines, and Circles etc. are used.

Adding two integer numbers	
Algorithm	**Flowchart**
i. Declare variables a,b,c ii. Read a, and b from keyboard iii. Add a, b and store result in c iv. Display result	

c. **Coding:** Once the design is completed, the program is written using any programming language such as C, C++, Python, and Java etc. The Coding usually takes very less time, and syntax rules of the language are used to write the program.

adding two integers numbers in Python
a=int(input('Enter a')) b=int(input('Enter b')) c=a+b print('The sum is',c)

d. **Debugging:** At this stage the errors such as syntax errors in the programs are detected and corrected. This stage of program development is an important process. Debugging is also known as program validation.
e. **Testing:** The program is tested on a number of suitable test cases. The most insignificant and the most special cases should be identified and tested.
f. **Documentation:** Documentation is a very essential step in the program development. Documentation helps the users and the who actually maintain the software.
g. **Maintenance:** Even after the software is completed, it needs to be maintained and evaluated regularly. In software maintenance, the programming team fixes program errors and updates the software when new features are introduced.

Input, Processing, and Output

The data that is given to the programs is called input. This input is accepted from some source, and it is processed inside the program, and then finally output is sent to some destination. In terminal-based interactive programs, the input source is the keyboard, and the output destination is the terminal display. The Python shell itself is such a program; its inputs are Python expressions or statements. This input is processed and out is displayed in the shell itself.

The programmer can also force the output of a value by using the **print** function. The simplest form for using this function looks like the following:

```
print(values or expression, sep=‘ ’, end=‘ ’, file=sys.stdout, flush=true )
```

Parameters:

value(s) or expression(s): Any value, or expression that gives a value. This will be converted to string before printed.

sep='separator': (Optional) Specify how to separate the objects, if there is more than one. Default: ' '

end='end': (Optional) Specify what to print at the end. Default : '\n'

file: (Optional) An object with a write method. Default :sys.stdout

flush: (Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

```
>>> print('Hello, CSE Students')
Hello, CSE Students
>>> print(12*3/4)
9.0
>>> print(1,2,3,4,sep='-',end='$')
1-2-3-4$
```

When we are really working with python programs, they often require input from the user. The input can be received from the user using the **input** function. When you are working with **input ()** function it causes the program to stop until the user enters the input and presses the enter button. The program later uses the input given by the user and processes it to display output using print () function directly to the console or saves it to the specified file with file parameter. The syntax of the input function will be as follow:

```
var_name=input('Enter value for the variable')
```

The input function does the following:

- Displays a prompt for the input. In this example, the prompt is "Enter value for the variable".
- Receives a string of keystrokes, called characters, entered at the keyboard and returns the string to the shell.

Displaying Output with the Print Function

Even though there are different ways to print values in Python, we discuss two major string formats which are used inside the print() function to display the contents onto the console.

- str.format() method
- f-strings

str.format() –this function is used to insert value of a variable into another string and display it as a single string.

Syntax: str.format(p0,p1,..k0=val1,k1=val1..), where p0,p1 are called ***positional***, and k0,k1 are called ***keyword arguments***.

Positional arguments are accessed using the index, and keyword arguments are accessed using the name of the argument.

f-strings -Formatted strings or f-strings were introduced in Python 3.6. A f-string is a string literal that is prefixed with "f".

Using str.format() with positional arguments

dataoutput.py	**Output**

#more about data output
country=input('Enter your country')
print('I Love my {}'.**format**(country))
	Enter your country india
	I Love my india

Where {} are called ***placeholder***. The value of the variable is placed inside the placeholder according to the position. The arguments are placed according to their position.

dataout.py	Output
branch=input('Enter branch name') year=int(input('Enter the year of study')) print('The branch name is {0} and the year is {1}'.format(branch,year))	Enter branch name CSE Enter the year of study 2 The branch name is CSE and the year is 2

Ex: print('The branch name is {1} and the year is {0}'.format(year,branch))

Note: Position is important here. You need to remember the position, otherwise wrong result is expected

Using str.format() with keyword arguments

It may be difficult for us to remember the order or positions of arguments. Keyword arguments are suitable when you are not sure of position, but know the names of the arguments.

dataout1.py

```
branch=input('Enter branch name')
year=int(input('Enter the year of study'))
print('The branch is {b} and year is {y}'.format(b=branch,y=year))
```

Output:

```
Enter branch name cse
Enter the year of study 2
The branch is cse and year is 2
```

Using f-string

Formatted strings or f-strings were introduced in Python 3.6. A f-string is a string literal that is prefixed with **"f"**. These strings may contain replacement fields, which are expressions enclosed within

curly braces { }. The expressions are replaced with their values. An **f** at the beginning of the string tells Python to allow any valid variable names within the string.

Dataout2.py

```
branch=input('Enter branch name')
year=int(input('Enter the year of study'))
print( f 'The branch name is {branch} and the year is {year}')
```

Output

```
Enter branch name cse
Enter the year of study 2
The branch name is cse and the year is 2
```

Comments

Comments are non-executable statements in Python. It means neither the python interpreter nor the PVM will execute them. Comments are intended for human understanding. Therefore, they are called non-executable statements.

There are two types of commenting features available in Python: These are single-line comments and multi-line comments.

A single-line comment – It begins with a hash (#) symbol, all the characters until end of the line will be treated as part of comment. Example: ***#read input from the keyboard***

Multi-line comment – It is useful when we need to comment on many lines. In Python Triple double quote (""") and triple single quote (''') are used for Multi-line commenting. It is used at the beginning and end of the block to comment on the entire block. Hence it is also called block comments. Example:

Triple double quotes	*Triple single quotes*
“”” python supports multiple comment using Triple double quotes”””	‘‘‘python supports multiple comment using Triple double quotes’’’

Variables

Variable is the name given to the value that is stored in the memory of the computer.

Ex: X=5, f=12.34, name=‘Python’, c=2+4j

Value is either string, numeric etc. Example: "Sara", 120, 25.36. Variables are created when values are first assigned. Variables must be assigned before being referenced. The value stored in a variable can be accessed or updated later. No declaration required. The type (string, int, float etc.) of the variable is determined based on value assigned. The interpreter allocates memory on the basis of the data type of a variable.

Naming rules of variable

- Must begin with a letter (a - z, A - B) or underscore (_)
- Other characters can be letters, numbers or _ (alphanumeric Characters)
- Variables are Case Sensitive (‘age’ , ‘Age’, ‘AGE’ are three different variables)
- Can be any (reasonable) length (var.bit_length() returns the number of bits required)
- There are some reserved words which you cannot use as a variable name because Python uses them for other things.

Assignment Operator

It is used to create a variable and make it reference to a value in the memory. Variable name is written to the left of =, and value is written to the right of =

Syntax

```
<variable> = <expr>
```

Examples

x=5

Y=10.25

C=3+4J

Multiple Assignment

The basic assignment statement works for a single variable and a single expression. You can also assign a single value to more than one variable simultaneously.

Syntax

var1=var2=var3...varn= <expr>

Example :

x = y = z = 1

Example:

x, y, z = 1, 2, "abcd"

In the above example x, y and z simultaneously get the new values 1, 2 and "abcd".

Local and global variables in python

In Python, variables that are only referenced inside a function are implicitly global. If a variable is assigned a value anywhere within the function's body, it's assumed to be a local variable unless explicitly declared as global. Variables that are created outside of a function are known as global variables. Global variables can be used by everyone, both inside of functions and outside.

```
var1='Python'
#function definition
def fun1():
    var1='Pyt'
```

```
    print('Variable value is :',var1)

def fun2():

    global var1 #here global is the keyword that refers global variable

    print('Variable value is:',var1)

#main function

#calling the functions

fun1()

fun2()
```

Reading Input from the Keyboard

When we are really working with python programs, they often require input from the user. The input can be received from the user using the input function. When you are working with input () function it causes the program to stop until the user enters the input and presses the enter button. The program later uses the input given by the user and processes it to display output using print () function directly to the console or saves it to the specified file with file parameter. The syntax of the input function will be as follow:

```
var_name=input('Enter value for the variable')
```

The input function does the following:

- Displays a prompt for the input. In this example, the prompt is "Enter value for the variable".

- This function will automatically convert the input into string. If we want the input in the integer format we need to convert it using int() function which is known as type conversion. Similarly, we have float(), complex(), hex(),oct(), ord(),chr(), and str() functions to convert one type of data into another type.

Write a python program to read your name and display it to console.

readinput.py	***Output***
name=input('Enter your name:') print('My name is:',name)	Enter your name: guido vas rossum My name is: guido vas rossum

Performing Calculations

Performing calculations involving both integers and floating-point numbers is called mixed-mode arithmetic. For instance, if a circle has radius 3, we compute the area as follows:

```
>>> 3.14*3*3
28.259999999999998
```

In the binary operation the less general type (int) will be automatically converted into more general type (float) before operation is performed. For example:

```
>>> 9*5.0
45.0
```

Here, 9 is integer, and 5.0 is float, then the less general type that is int will be converted into more general type that is float and the entire expression will result in float value.

The eval () function

We can even use eval () function to perform calculation at the interpreter. The expression is written inside the single quotes. For example:

```
>>> eval('45/9*2')
10.0
```

Operators

Operators are symbols, such as +, –, =, >, and <, that perform certain mathematical or logical operation to manipulate data values and produce a result based on some rules. An operator manipulates the data values called operands.

Consider the expression,

>>> 4 + 6

where 4 and 6 are operands and + is the operator.

Python language supports a wide range of operators. They are

1. Arithmetic Operators

2. Bitwise Operators

3. Comparison Operators

4. Logical Operators

5. Assignment Operators

6. Membership Operators

7. Identity Operators

Arithmetic Operators

Arithmetic operators are used to execute arithmetic operations such as addition, subtraction, division, multiplication etc. The following table shows all the arithmetic operators.

Operator	Meaning	Example
+	Addition-Used to perform arithmetic addition	x+y, results in 10
-	Subtraction-Used to perform arithmetic subtraction	x-y, results in 4
*	Multiplication-Used to perform multiplication	x*y, results in 21
/	Division-Used to perform division	x/y , results in 2
%	Modulus-Used to perform modulus operation (remainder)	x%y, results in 1
//	Used to perform floor division (floor value)	x//y, results in 2
**	Exponent- Used to raise operand on left to the power of operand on right	x**y, 343

Write a program that asks the user for a weight in kilograms and converts it to pounds. There are 2.2 pounds in a kilogram.

Prg1.py	Output

k=float (input ('Enter kilograms'))
#Convert kgs to pounds
p=k*2.2;
#Display result
print('The equivalent pounds is',p)
	Enter kilograms100
	The equivalent pounds is 220.00

Write a program that asks the user to enter three numbers (use three separate input statements). Create variables called total and average that hold the sum and average of the three numbers and print out the values of total and average.

Prg2.py	Output

x=float(input('Enter x value'))
y=float(input('Enter y value'))
z=float(input('Enter z value'))
#create total and avg variable
total=x+y+z
print('The total is:',total)
#average
average=total/3
print('The average is:',average)
	Enter x value12
	Enter y value13
	Enter z value14
	The total is: 39.0
	The average is: 13.0

Bitwise Operators

Bitwise operators treat their operands as a sequence of bits (zeroes and ones) and perform bit by bit operation. For example, the decimal

number ten has a binary representation of 1010. Bitwise operators perform their operations on such binary representations, but they return standard Python numerical values. The Following table lists all the Bitwise operators:

Operator	Meaning	Example
&	Bitwise AND	x&y=0 x=010 y=111 x&y= 010 (2)
|	Bitwise OR	x|y=7 x=010 y=111 x|y=111 (7)
~	Bitwise Not	~x is ,-3
^	Exclusive OR (XOR)	X^y=5 x=010 y=111 x|y=101 (5)
>>	Shift Right (operand >>no. of bit positions)	x>>1, results 1
<<	Shift Left (operand <<no. of bit positions)	X<<2, 1000 (8)

Write a python program to demonstrate all the bitwise operators.

Exp1.py	Out put
x=input("Enter value of x :") y=input("Enter value of y :") print("-----Bitwise Operations--------") print (" AND (&) is:",(x&y)) print (" OR (|) is:",(x|y)) print (" XOR (^) is:",(x^y)) print (" Not (~) is:",(~x)) print (" Shift Right(>>) is:",(x>>1)) print (" Shift Left (<<)is:",(x<<2))	Enter value of x :2 Enter value of y :7 -----Bitwise Operations-------- AND (&) is: 2 OR (|) is: 7 XOR (^) is: 5 Not (~) is: -3 Shift Right(>>) is: 1 Shift Left (<<)is: 8

Comparison Operators

When the values of two operands are to be compared then comparison operators are used. The output of these comparison operators is always a ***Boolean value***, either ***True or False***. The operands can be Numbers or Strings or Boolean values. Strings are compared letter by letter using their ASCII values; thus, "M" is less than "N", and "Guido" is greater than "Bob". The following table shows all the comparison operators.

Operator	Meaning	Example
>	Greater Than-Returns True if the left operand is greater than the right, otherwise returns False	x>y, results in True
<	Less Than-Returns True if the left operand is less than the right, otherwise returns False	X<y, results in False
= =	Equal to-Returns True if both are equal, otherwise False	x==y, returns False
!=	Not Equal to- Returns True if both are not equal, otherwise False	x!=y, return True
>=	Greater than or Equal- Returns True if the left operand is greater than or equal to the right, otherwise returns False	x>y, returns True
<=	Less than or Equal- Returns True if the left operand is Less than or equal to the right, otherwise returns False	X<y, returns False

Write a python program to determine the biggest number among three numbers.

Exp2.py	Output

a=int(input('Enter a'))
b=int(input('Enter b'))
c=int(input('Enter c'))
if a>b and a>c:
  print(a, 'is biggest')
elif b>a and b>c:
  print(b,' is biggest')
else:
  print(c, 'is biggest')
	Enter a5
	Enter b3
	Enter c7
	7 is biggest

logical operators

The logical operators are used for comparing the logical values of their operands and to return the resulting logical value. The values of the operands on which the logical operators operate evaluate to either True or False. There are three logical operators: and, or, and not.

Operator	Meaning	Example
and	True if both the operands are True	x **and** y
Or	True, if either of the operands is True	x **or** y
not	True if operand false	**not** x

Write a python program to demonstrate the Logical operators.

Exp3.py	Out Put
x=True y=False print (" x and y is :",x and y) print (" x or y is :",x or y) print (" not x is:",not x)	x and y is : False x or y is : True not x is: False

Assignment Operator

Assignment operator is used to assign values to the variable. For example, x=5 is simple assignment operator, that assigns value 5 to the to the variable x. There are various compound operators in python like a+=5, which adds value 5 to the variable and later assigns it to variable a. This expression is equivalent to a=a+5. The same assignment operator can be applied to all expressions that contain arithmetic operators such as, *=, /=, -=, **=,%= etc.

```
x=4
x+=5
print (“The value of x is:”, x)
```

Output:

The value of x 9

Membership Operators

These operators are used to test whether a value or operand is there in the sequence such as list, string, set, or dictionary. There are two membership operators in python: in and not in. In the dictionary we can use to find the presence of the key, not the value. If x is a list containing the elements [1,2,3,4] then following example demonstrates the use of these operators.

Operator	Meaning	Example
in	True if value or operand is present in the sequence	5 in x
not in	True if value or operand is not present in the sequence	5 not in x

Write a program that asks the user to enter a word and prints out whether that word contains any vowels. (Lab prg 8)

Prg8.py	Output

word=input ('Enter any word:')
vowels=['a','e','i','o','u']
for i in vowels:
  if i in word:
  print (i,'is present')
	Enter any word: python
	
	o is present

Identity Operators

These are used to check if two values (variable) are located on the same part of the memory. If the x is a variable contain some value, it is assigned to variable y. Now both variables are pointing (***referring***) to the same location on the memory as shown in the example program.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same memory)	X=5 Y=X X is Y , returns True

is not	True if the operands are not identical (refer to the same memory)	X=5 #int
		Y=5.0 # float
		X is not Y, returns True

Type conversions

You can explicitly cast, or convert, a variable from one type to another type.

- To explicitly convert a float number or a string to an integer, cast the number using **int()** function. Ex: f=3.2, i=int(f), then variable i contains 3, decimal part is truncated.

- The **float()** function returns a floating point number constructed from a number or string. Ex: a=3, f=**float(a)**, then f contains 3.0
- The **str()** function returns a string which is fairly human readable. Ex: a=3, s=**str(a),** then s contains '3'
- Convert an integer to a string of one character whose ASCII code is same as the integer using **chr()** function. The integer value should be in the range of 0–255.
- char_A=**chr(65)**, here char_A contains ascii character capital A
- char_a=**chr(97)**, here char_a contains ascii character small a
- Use **complex()** function to print a complex number with the value ***real + imag*j*** or convert a string or number to a complex number. c1=complex(3,4), then if we print the value of c1, it gives out output as follow: (3+4j)
- The **ord()** function returns an integer representing Unicode code point for the given Unicode character.
- alpha_Z=**ord('Z'**), where alpha contains ascii value 90.
- Convert an integer number (of any size) to a lowercase hexadecimal string prefixed with "0x" using **hex()** function. For example: i_to_h=hex(255), i_to_h contains '0xff' and i_to_h=hex(16), i_to_h contains '0x10'

- Convert an integer number (of any size) to an octal string prefixed with “0o” using **oct()** function. For example, o=oct(8), where o contains ‘0o10’ and o=oct(16), where o contains ‘0o20’
- The **type()** function returns the data type of the given object. If we pass 20 to the type() function as follow type(20) then its type will be displayed as <class 'int'>

Note: Python is a dynamically typed, high level programming language.

Expressions

An Expression is a combination of operators and operands that computes a value when executed by the Python interpreter. In python, an expression is formed using the mathematical operators and operands (sometimes can be values also).

Precedence of operator determines the way in which operators are parsed with respect to each other. Operators with higher precedence will be considered first for execution than the operators with lower precedence. ***Associativity*** determines the way in which operators of the same precedence are parsed. Almost all the operators have left-to-right associativity. The exponent and assignment operators have right-to-left associativity. The acronym ***PEMDAS*** is a useful way to remember the order of operations:

Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2*(3-1) is 4, and (1+1)**(5-2) is 8.

Exponentiation has the next highest precedence, so 2 **1+1 is 3 and not 4, and 3*1** 3 is 3 and not 27.

Multiplication and Division have the same precedence, which is higher than ***Addition and Subtraction***, which also have the same precedence. So 2 *3-1 yields 5 rather than 4, and 2/3-1 is -1, not 1 (re-member that in integer division, 2/3=0).

Precedence of the Operators

Operator Precedence in Python

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisons, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

More about Data Output

There are several ways to present the output of a program, data can be printed to the console.

The print function will print everything as strings.

Syntax: **print(value(s), sep= ' ', end = '\n', file=file, flush=flush)**

Parameters:

value(s): Any value, and as many as you like. Will be converted to string before printed.

sep='separator': (Optional) Specify how to separate the objects, if there is more than one.Default :' '

end='end': (Optional) Specify what to print at the end. Default : '\n'

file: (Optional) An object with a write method. Default :sys.stdout

flush: (Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

Example:

>>>print(10,20,30,40,sep='-',end='&')

Output:

10-20-30-40&

>>>print('apple',1,'mango',2,'orange',3,sep='@',end='#')

Output:

apple@1@mango@2@orange@3#

Even though there are different ways to print values in Python, we discuss two major string formats which are used inside the print() function to display the contents onto the console.

- str.format() method
- f-strings

str.format() –this function is used to insert value of a variable into another string and display it as a single string.

Syntax: str.format(p0,p1,..k0=val1,k1=val1..), where p0,p1 are called ***positional***, and k0,k1 are called ***keyword arguments***.

Positional arguments are accessed using the index, and keyword arguments are accessed using the name of the argument.

f-strings -Formatted strings or f-strings were introduced in Python 3.6. A f-string is a string literal that is prefixed with "f".

Using str.format() with positional arguments

dataoutput.py	Output
#more about data output country=input('Enter your country') print('I Love my {}'.**format**(country))	Enter your country india I Love my india

Where {} are called ***placeholder***. The value of the variable is placed inside the placeholder according to the position. The arguments are placed according to their position.

dataout.py	Output
branch=input('Enter branch name') year=int(input('Enter the year of study')) print('The branch name is {0} and the year is {1}'.format(branch,year))	Enter branch name CSE Enter the year of study 2 The branch name is CSE and the year is 2

Ex: print('The branch name is {1} and the year is {0}'.format(year,branch))

Note: Position is important here. You need to remember the position, otherwise wrong result is expected

Using str.format() with keyword arguments

It may be difficult for us to remember the order or positions of arguments. Keyword arguments are suitable when you are not sure of position, but know the names of the arguments.

dataout1.py

```
branch=input('Enter branch name')

year=int(input('Enter the year of study'))

print('The branch is {b} and year is {y}'.format(b=branch,y=year))
```

Output:

Enter branch name cse

Enter the year of study 2

The branch is cse and year is 2

Using f-string

Formatted strings or f-strings were introduced in Python 3.6. A f-string is a string literal that is prefixed with **"f"**. These strings may contain replacement fields, which are expressions enclosed within curly braces { }. The expressions are replaced with their values. An **f** at the beginning of the string tells Python to allow any valid variable names within the string.

Dataout2.py

```
branch=input('Enter branch name')
year=int(input('Enter the year of study'))
print( f 'The branch name is {branch} and the year is {year}')
```

Output

Enter branch name cse

Enter the year of study 2

The branch name is cse and the year is 2

II. Data types, and expressions

Strings, Assignment, and Comment

In Python, a string literal is a sequence of characters enclosed in ***single*** or ***double*** quotation marks. The following program statements with the Python shell shows some example strings:

```
>>> s2="Hello, python"
>>> s1='Hello,Python'
>>> s1
'Hello,Python'
>>> s2
'Hello, python'
```

Double-quoted strings are suitable for composing strings that contain single quotation marks or apostrophes. Here is a self-justifying example:

```
>>> pyt="I'am interested in learning Python"
>>> pyt
"I'am interested in learning Python"
```

Escape Sequence

The newline character \n is called an ***escape sequence***. Escape sequences are the way Python expresses special characters, such as the tab, the newline, and the backspace (delete key), as literals.

ESCAPE SEQUENCE	*MEANING*
\b	Backspace
\n	Newline
\t	Horizontals tab
\\	The \ Character
\’	Single Quotation mark
\”	Double quotation mark

Example:

```
>>> print("Hello \t Python")
Hello    Python
>>> print("Hello \n Python")
Hello
 Python
>>> print("Hello \" Python")
Hello " Python
>>> print("Hello \\ Python")
Hello \ Python
```

String Concatenation

We can join two or more strings to form a new string using the concatenation operator **+**. Here is an example:

```
>>> 'My name is' + 'Guido'

'My name isGuido'
```

Assignment Statement

Programmers use all uppercase letters for the names of variables that contain values that the program never changes. Such variables are known as ***symbolic constants***. Examples of symbolic constants in the tax calculator case study are: **TAX_RATE and STANDARD_DEDUCTION.**

Variables receive their initial values and can be reset to new values with an assignment statement. The form of an assignment statement is the following:

<variable_name> = <expression>

The Python interpreter first evaluates the expression on the ***right side*** of the assignment symbol and then binds the variable name on the left side to this value.

When this happens to the variable name for the first time, it is called ***defining or initializing*** the variable. Note that the = symbol means assignment, not equality. After you initialize a variable, subsequent uses of the variable name in expressions are known as *variable references*.

There are two important ***purposes*** of variables: ***First***, it helps to keep track of the variable changing inside the program. ***Second***, it helps to refer the complex information with simple name, which is also called abstraction.

Comment

A comment is a piece of program text that the interpreter ignores but that provides useful documentation to programmers. At the very least, the author of a program can include his or her name and a brief statement about the purpose of the program at the beginning of the program file. This type of comment, called a ***docstring,*** is a multi-line string. This can be written inside Triple double quotes or Triple single quotes. In addition to docstrings, ***end-of-line*** comments can document a program. These comments begin with the **#** symbol and extend to the end of a line.

Docstring	***End-of-line***
""" Program: VowelTest.py Author : KSR Purpose: Testing whether a given word contains any vowels or not """	# read word from keyboard

Numeric Data Types

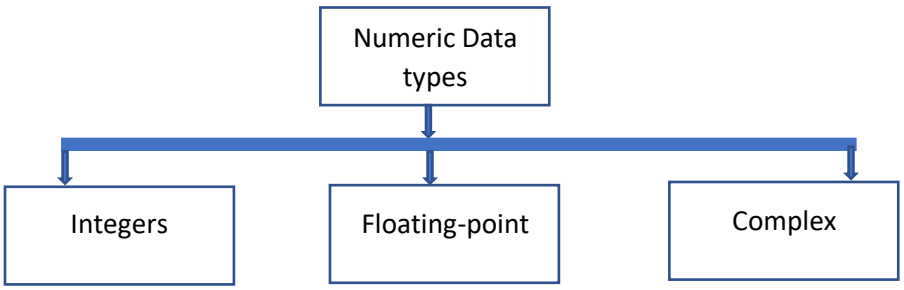


Figure 1 Numerical Data types

Under the numeric data types python has three different types: integers, floating-point, complex numbers. These are defined as int, float, complex in python. ***Integers*** can be of any length; it is only limited by the memory available. Python uses floating-point

numbers to represent real numbers. A ***floating-point*** number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is an integer, 1.0 is floating point number. A floating-point number can be written using either ordinary decimal notation or scientific notation. Example, 37.8 can be represented in scientific notation as 3.78e1. ***Complex numbers*** are written in the form, x + yj, where x is the real part and y is the imaginary part. Example, (3+4j).

Character Sets

Some programming languages use different data types for strings and individual characters. In Python, character literals look just like string literals and are of the string type. But they also belong to several different character sets, among them the ***ASCII set*** and the ***Unicode set***.

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DCI	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	"	#	$	%	&	`
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

Using functions and Modules

A function is a chunk of code that can be called by name to perform a task. Functions often require *arguments*, that is, specific data values, to perform their tasks. Arguments are also known as

parameters. When a function completes its task, the function may send a result back to the part of the program that called that function, which is also known as *caller*. The process of sending a result back to another part of a program is known as returning a value.

For example, the argument in the function call **round(6.6**) is the value **6.6**, and the value returned is **7**. When an argument is an expression, it is first evaluated, and then its value is passed to the function for further processing. For instance, the function call **abs(4 – 5)** first evaluates the expression 4 - 5 and then passes the result, -1, to abs. Finally, abs returns 1.

The values returned by function calls can be used in expressions and statements. For example, the function call **print (abs(4 - 5) + 3)** prints the value 4. Some functions have only optional arguments, some have required arguments, and some have both required and optional arguments. For example, **round (4.34234,2),** returns 4.34 with 2 decimal points. In this function call second argument 2 is optional argument, first argument is required argument.

The math module

Functions and other resources are placed in components called modules. Functions like **abs()** and **round()** from the __builtin__ module are always available for use, whereas the programmer must explicitly **import** other functions from the modules where they are defined. The math module includes several functions that perform basic mathematical operations. If we want to know all the function present inside the math module, first we have import it, and then use the dir() function to list all the functions at the prompt.

```
>>> import math
>>> dir(math)
```

['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf',

'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', '**sqrt**', 'tan', 'tanh', 'tau', 'trunc']

To use a resource from a module, you write the name of a module as a qualifier, followed by a **dot (.)** and the name of the resource. For example, to use the function **sqrt()**, we have to write it as math.sqrt(25).

III. Decision Structures and Boolean Logic

In all most all programming languages the control flow statements have been classified as Selection/Decision Statements, Loop/Repetition/Iterative Statements, and Jump Statements. Under the Decision statements in Python we have ***if, elif and else*** statement. Under the Repetition statements we have ***for*** and ***while*** statements. Under the Jump statements we have ***break***, ***continue*** and ***pass*** statements.

The if Decision statement:

The simplest form of decision/selection is the if statement. This type of control statement is also called a ***one-way selection*** statement, because it consists of a condition and just a single sequence of statements. If the condition is True, the sequence of statements is run. Otherwise, control proceeds to the next statement following the entire selection statement.

The Syntax of the of if statement will be as follow:

```
if Boolean_ Expression:
        Statement 1
        Statement 2
        :

        Statement N
```

The if decision control flow statement starts with **if** keyword and ends with a colon. The expression in an if statement should be a Boolean expression which will be evaluated to True or False. The Boolean_Expression is evaluated to True, then if block will be

executed, otherwise the first statement after the if block will be executed. The statements inside the if block must be properly indented with spaces or tab.

Write python program to test whether a given number is Even or not using if decision statement.

```
#Even or Odd
n=int(input("Enter the number"))
if n%2==0:
    print(n,'is even number')
    print('End of if block')
print('End of the program')
print('Execution is completed')
```

The if-else decision statement

An **if** statement can also be followed by an **else** statement which is optional. An else statement does not have any condition. The statements in the if block are executed if the Boolean_Expression is **True**, otherwise the else block will be executed. The if...else statement used for two-way decision, that means when we have only two alternatives. The syntax of if-else will be as follow:

```
if Boolean_Expression:
        statements 1
else:
        statements 2
```

Here, Statements 1 and Statement 2 can be single statement or multiple statements. The statements inside if block and else block must be properly indented. Both the indentation need not be same.

Above program with if-else decision statements.

```
#Even or Odd
n=int(input("Enter the number"))
```

```
if n%2==0:
    print(n,'is even number')
    print('End of if block')
else:
 print(n,'is odd')
 print('else block')
 print('End of the program')
print('Execution is completed')
```

Note: if you observe above if and else blocks. The else block indentation is single space, whereas the if bloc has used tab for indentation. Hence both Block can use different indentation, but all the statements within the block should have same indentation.

Generate a random number between 1 and 10. Ask the user to guess the number and print a message based on whether they get it right or not. (Lab prg 6)

```
import random
while True:
    n=int(input('Enter any number between 1 and 10'))
    print(f' You are trying to guess number {n}')
    rn=random.randint(1,10)
    print('The random number generated is:',rn)
    if rn==n:
        print('Your guessing is right')
    else:
        print('Badluck your guessing is wrong')
```

Output:

```
Enter any number between 1 and 108
 You are trying to guess number 8
The random number generated is: 1
Bad luck your guessing is wrong
Enter any number between 1 and 105
```

```
You are trying to guess number 5
The random number generated is: 5
Your guessing is right
```

The if-elif-else decision statement

The if…elif…else is also called multi-way decision statement. This multi-way decision statement is preferred whenever we need to select one choice among multiple alternatives. The keyword '**elif**' is short for 'else if'. The else statement will be written at the end and will be executed when no if or elif blocks are executed. The syntax of will be as follow:

```
If Boolean_expression1:
        Statements
elif Boolean_expression2:
        Statements
elif Boolean_exxpression3:
        Statements
else:
        Statements
```

Write a Program to Prompt for a Score between 0.0 and 1.0. If the Score Is Out of Range, Print an Error. If the Score Is between 0.0 and 1.0, Print a Grade Using the Following Table.

Score	>=0.9	>=0.8	>=0.7	>=0.6	<0.6
Grade	A	B	C	D	F

```
score=float(input('Enter your score:'))
if score<0 or score > 1:
   print('Wrong input is given')
elif score>=0.9:
   print('Your  Grade is A')
elif score>=0.8:
   print('Your  Grade is B')
```

```
elif score>=0.7:
    print('Your  Grade is C')
elif score>=0.6:
    print('Your  Grade is D')
else:
    print('Your Grade is F')
```

Output:

Enter your score **0.75**

Your Grade is **C**

Nested if statements

Sometimes it may be need to write an if statement inside another if block then such if statements are called nested if statements. The syntax would be as follow:

```
if Boolean_expression1:
        if Boolean_expression2:
                if Boolean_expression3:
                        Statements
                else:
                        Statements
```

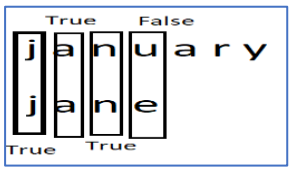
Program to Check If a Given Year Is a Leap Year

```
year=int(input('Enter year:'))
if year%4==0:
    if year%100==0:
        if year%400==0:  #nested if statements
            print(f'{year} is a leap year')
        else:
            print(f'{year} is not leap year')

    else:
        print(f'{year} is a leap year')
else:
        print(f'{year} is not a leap year')
```

Comparing Strings

We can use comparison operators such as >, <, <=, >=, ==, and != to compare two strings. This expression can return a Boolean value either True or False. Python compares strings using ASCII value of the characters. For example,




```
>>> "january" == "jane"
False
```

String equality is compared using == (double equal sign). This comparison process is carried out as follow:

- First two characters (j and j) from the both the strings are compared using the ASCII values.
- Since both ASCII values are same then next characters (a and a) are compared. Here they are also equal, and hence next characters (n and n) from both strings are compared.
- This comparison also returns True, and comparison is continued with next characters (u and e). Since the ASCII value of the 'u' is greater than the ASCII value of 'e' this time it returns False. Finally, the comparison operation returns False.

Boolean Variables

The variables that store Boolean value either True or False called Boolean variables. If the expression is returning a Boolean value then it is called Boolean expression.

Example:

```
>>>X=True
>>>Y=False
>>> a>5 and a<10  #Boolean expression
```

IV. Repetition Structures

Introduction

Whenever if we want to execute a block of statements repeatedly for some finite number of times or until some condition is satisfied then repetition structures are used. The repetition structures also known

as loops, which repeat an action. Each repetition of the action is known as a ***pass*** or an ***iteration***. There are two types of loops—those that repeat an action a predefined number of times (***definite iteration***) and those that perform the action until the program determines that it needs to stop (***indefinite iteration***). There are two loop statements in Python, for and while.

The while loop

In many situations, however, the number of times that the block should execute is not known in advance. The program's loop continues to execute as long as valid input is entered and terminates at special input. This type of process is called conditional iteration. In this section, we explore the use of the ***while loop*** to describe conditional iteration

While loop repeats as long as a certain Boolean condition is met. The block of statements is repeatedly executed as long as the condition is evaluated to True. The general form of while will be as follow:

```
while condition: #Loop Header
        statement 1
        statement 2
        ……
        Statement N
```

The first line is called loop header which contains a keyword while, and condition which return a Boolean value and colon at the end. The body of the loop contains statement1, statements2 , and so on. Thus is repeated until the condition is evaluated to True otherwise loop will be terminated.

Write a python program to demonstrate wile loop for computing the sum of numbers entered by the user. Terminates when user enters special character 'space'.

```
data=input('Enter any number')
sum=0
while data!=" ":
    sum=sum+int(data)
    data=input('Enter any number or space to quit')
print('The sum is :',sum)
```

The count control with while loop

We can also use while loop for count-controlled loops. The body of the while is repeatedly executed until the condition which returns a Boolean value True. Otherwise body of the loop is terminated and the first statement after the while loop will be executed.

Write a program that asks the user for their name and how many times to print it. The program should print out the user's name the specified number of times. (Lab Prg 4)

```
n=int(input('Enter the number that you want to display your name:'))
name=input('Enter name:')
while n>=0:
    print(name)
    n=n-1
print('End of the program')
```

Output:

```
Enter the number that you want to display your name:3
Enter name: Guido
Guido
Guido
Guido
End of the program
```

The for loop

For loop iterates over a given sequence or list. It is helpful in running a loop on each item in the list. The general form of "for" loop in Python will be as follow:

```
for variable in [value1, value2, etc.]:  # Loop Header
    statement1
    statement2
    ……
    Statement N
```

Here variable is the name of the variable. And **for** and **in** are the keywords. Inside the square brackets a sequence of values is separated by comma. In Python, a comma-separated sequence of data items that are enclosed in a set of square brackets is called a ***list***. The list is created with help of [] square brackets. The list also can be created with help of tuple. We can also use range() function to create the list. The general form of the range() function will be as follow:

- range(number) –ex: **range (10)** –It takes all the values from 0 to 9
- range (start,stop, interval_size) –ex: **range(2,10,2**)-It lists all the numbers such as 2,4,6,8.
- range(start,stop)-ex: **range(1,6**), lists all the numbers from 1to 5, but not 6. Here, by default the interval size is 1.

Write a Python Program to find the sum of all the items in the list using for loop.

fortest.py	**Output**
#sum of all items in the list s=0 for x in [1,2,3,4,5]: # **list** s=s+x print ("The sum of all items in the list is:",s)	The sum of all items in the list is: 15

Write a program that uses a *for* loop to print the numbers 8, 11, 14, 17, 20, . . . , 83, 86, 89. (Lab Prg 3)

```
for i in range(8,90,3):
    print(i)
```

Use a for loop to print a triangle like the one below. Allow the user to specify how high the triangle should be. (lab Prg 5)

```
for i in range(0,4):
    for j in range(0,i+1):
        print('*',end=" ")
    print("\r")
```

Calculating a Running Total

We can calculate the sum of input numbers while entering from the keyboard as demonstrated in the following example.

```
n=int(input('Enter n:'))
sum=0
for i in range(n):
    data=float(input('Enter value'))
    sum=sum+data
#display sum
print('Sum is:',sum)
```

Output:

Enter n:4

Enter value12

Enter value13

Enter value21

Enter value22

Sum is: 68.0

Input Validation Loops

Loops can be used to validate user input. For instance, a program may require the user to enter a positive integer. Many of us have seen a "yes/no" prompt at some point, although probably in the form of a dialog box with buttons rather than text.

```
import random
action = "Y"
while action == "Y":
   print("Generating random number...")
   randomNumber = random.randint(1,10)
   print("Random number is", randomNumber)
   action = input("Another (Y/N)? ")
   while action != "Y" and action != "N":
      action = input("Invalid input! Enter Y or N: ")
print("All done!")
```

Nested loops

Writing a loop statement inside another is called Nested loops. The "***inner loop***" will be executed one time for each iteration of the "***outer loop***". We can put any type of loop inside any other type of loop. For example, a for loop can be inside a while loop or vice versa.

Syntax of nested loops:

```
#Nested for loops
for iterating_var in sequence: #outer for loop
   for iterating_var in sequence: #inner for loop
      statements(s)
   statements(s)
```

```
# Nested loop with for & while
for iterating_var in sequence: #outer for loop
    while expression: #inner for loop
        statement(s)
    Statements
```

Use a for loop to print a triangle like the one below. Allow the user to specify how high the triangle should be. (Lab Prg 5)

Using for nested loops	Using for and while nested loops

for i in range(4):
  for j in range(0,i+1):
  print('*',end=' ')
  print('\r')
	for i in range(4):
	  j=0
	  while j < (i+1):
	  print('*',end=' ')
	  j=j+1
	  print('\r')

* * * * * * * * * *	

Else with loops

Loop statements may have an else clause

- It is executed when the loop terminates through exhaustion of the list (with for loop).
- It is executed when the condition becomes false (with while loop), but not when the loop is terminated by a break statement.

Example**: Printing all primes numbers up to 100**

```
print('2')
for i in range(3,101,2):
    for j in range(2,i):
        if i%j==0:
            break
    else:
        print(i)
```

Using the while inner loop

```
print('2')
for i in range(3,101,2):
    j=2
    while j<i:
        if i%j==0:
            break
        j=j+1
    else:
        print(i)
```

Jump Statements

- we have three jump statements: **break, continue and pass.**
- **Break statement:**
  - It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C.
  - The break statement can be used in both while and for loops.

Write a python program to search for a given number whether it is present in the list or not.

```
n=int(input('Enter number to search :'))
for x in range(1,11):#[1,2,3,4,5,6,7,8,9,10]
    if x==n:
        print(n,' is found')
        break #terminates the loop
print('end of program')
```

Output:

```
======= RESTART: C:\Users\91
Enter number to search :5
5  is found
end of program
>>>
```

- **The continue statement**
  - It returns the control to the beginning of the loop.
  - The continue statement skips all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
  - The continue statement can be used in both while and for loops.

```
1
2
3
4
6
7
8
9
10
end of program
>>>
```

- **The pass statement**
- The pass statement does nothing
- It can be used when a statement is required syntactically but the program requires no action
- Example: creating an infinite loop that does nothing

```
while True:
        pass
```

Example program to demonstrate the pass statement

```
f=True
while f:
    pass
    print('This line will be printed')
    f=False
print('End of the program')
```

Output:

```
This line will be printed
End of the program
```

********* End of the Unit 1********