LAB TITLE AND CODE : SOFTWARE DEFINED RADIO LAB 19CCE284
LAB SESSION : 1
DATE : 11/04/2022 (MONDAY)

FAMILIARIZATION WITH PYTHON, SIMPLE EXPERIMENTS

* **AIM:**
A brief introduction to python with the implementation of basic commands.

* **SOFTWARE REQUIRED :**
Spyder IDE - Python 3.9

* **THEORY - PYTHON BACKGROUND :**
Python is a cross-platform programming language, meaning it runs on multiple platforms like Windows, Mac OS X, Linux, Unix and has been ported to the Java and .NET virtual machines. It was created by Guido van Rossum, and released in 1991. Python is an open-source (free) programming language that is used in web programming, data science, artificial intelligence, and many scientific applications.

Python can be used on a server to create a web application, alongside software to create workflows. It can connect to database systems, read and modify files. It can also be used to handle big data and perform complex mathematics, for rapid prototyping, or production-ready software development.

Python has a simple syntax similar to the English language that allows developers to write programs with fewer lines than some other programming languages. It runs on an interpreter system, meaning the code can be executed as soon as it is written.

Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of a program maintenance. Python supports modules and packages, which encourages modularity and code usage. The Python interpreter and the executive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

* ALGORITHM – PYTHON GETTING STARTED AND COMMANDS :

→ PYTHON VARIOUS MODES :-

① Immediate Mode - Typing python in the command line will invoke the interpreter in immediate mode. We can directly type in Python expressions and press enter to get the output. For example, >>> is the python prompt. It tells us that the interpreter is ready for our output. Try typing in 1+1 and press enter. We get 2 as the output. This prompt can be used as a calculator. To exit this mode, type exit() or quit() and press enter.

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

② Script Mode - This mode is used to execute a python program written in a file. Such a file is called a script. Scripts can be saved to disk for future use. Python scripts have the extension .py, meaning that the filename ends with .py. For example, helloWorld.py

③ Integrated Development IDE - We just need to save it with the .py extension. But using an IDE can make our work a lot easier. 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. Using an IDE can get rid of redundant tasks and significantly decrease in the time required for application development.

① Immediate Mode -

```
In [1] : 1+1
Out [1] : 2

In [2] : print ("Hello, world!")
Hello, world!
```

→ PYTHON LINES AND INDENTATION :-

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced. For example,

```
if 5>2:
    print ("Five is greater than two!")
```

→ PYTHON COMMENTS :-

Comments can be used to explain python code, make the code more readable and prevent execution when testing code.

① Creating a comment - Comments start with a #, and python will ignore them. For example,

```
#This is a comment
print ("Hello, World!")
```

Comments can be placed at the end of a line and python will ignore the rest of the line. For example,

```
print ("Hello, World!") # This is a comment
```

A comment does not have to be text that explains the code, it can also be used to prevent python from executing code. For example,

```
# print ("Hello, World!")
print ("Cheers, Mate!")
```

② Multi-line comments - Python does not have a syntax for multi-line comments. To add a multi-line comment, you can insert a # for each line. For example,

```
#This is a comment
#written in
#more than just one line
print ("Hello, World!")
```

→ PYTHON LINES AND INDENTATION :-

Five is greater than two!

→ PYTHON COMMENTS :-

Hello, World!
Hello, World!
Cheers, Mate!
} Creating a comment

Hello, World!
Hello, World!
} Multi-line comments

Or not quite as intended, you can use a multiline string. Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it. For example,

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

As long as the string is not assigned to a variable, Python will read the code, but will ignore it, and you have made a multiline comment.

→ PYTHON VARIABLES :-

Variables are containers for storing data values.

① Creating variables - Python has no command for declaring a variable. A variable is created the moment you first assign a value to it. For example,

```
x = 5
y = "JOHN"
print(x)
print(y)
```

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

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

→ PYTHON VARIABLES :-

① Creating variables -

5
John
Sally

(2) Casting - If you want to specify the data type of a variable, this can be done with casting. For example,

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

(3) Get the Type - You can get the data type of a variable with the type() function. For example,

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

(4) Single or Double Quotes - String variables can be declared either by using single or double-quotes. For example,

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

(5) Case-sensitive - Variable names are case-sensitive. For example,

```
# This will create two variables
a = 4
A = "Sally"
#A will not overwrite a
```

→ PYTHON NUMBERS :-

There are three numeric types in python, int, float and complex. Variables of numeric types are created when you assign a value to them. For example,

```
x = 1       # int
y = 2.8     # float
z = 1j      # complex
```

③ Get the Type -

<class 'int'>

<class 'str'>

→ PYTHON NUMBERS :-

<class 'int'>

<class 'float'>

<class 'complex'>

To verify the type of any object in python, use the type() function.
For example,

```
print (type (x))
print (type (y))
print (type (z))
```

→ PYTHON ASSIGN VALUES TO MULTIPLE VARIABLES :-

Python allows us to assign values to multiple variables in one line. For example,

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

And you can assign the same value to multiple variables in one line. For example, x=y=z="orange"

```
print (x)
print (y)
print (z)
```

→ PYTHON USER INPUT :-

Python allows for user input. That means we can ask the user for input. The following example asks for the username, it gets printed on the screen.

```
username = input ("Enter username : ")
print ("Username is : " + username)
```

→ PYTHON OPERATORS :-

operators are used for performing operations on variables and values. Python divides the operators into the following groups :

→ PYTHON ASSIGN VALUES TO MULTIPLE VARIABLES :-

```
Orange
Banana
Cherry
```

```
Orange
Orange
Orange
```

→ PYTHON USER INPUT :-

```
Enter username : Santosh
Username is: Santosh
```

① Arithmetic operators -

Operator	Description	Syntax
+	Addition : adds two operands	x+y
-	Subtraction : subtracts two operands	x-y
*	Multiplication : Multiplies two operands	x*y
/	Division (float) : Divides the first operand by the second	x/y
//	Division (floor) : Divides the first operand by the second	x//y
%	Modulus : Returns the remainder when the first operand is divided by the second	x%y
**	Power : Returns first raised to power second	x**y

② Assignment operators -

Operator	Description	Syntax
=	Assign a value of right side of expression to left side operand	x = y+z
+=	Add and Assign : Add right-side operand with left side operand and then assign to left operand	a+=b
-=	Subtract AND : Subtract right operand from left operand and then assign to left operand : True if both operands are equal.	a-=b
*=	Multiply AND : Multiply right operand with left operand and then assign to left operand	a*=b

Operator	Description	Syntax
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b
%=	Modulus AND: Takes modulus using left and right operand and assign to left operand	a%=b
//=	Divide (floor) AND: Divide left operand with right operand and then assign the value (floor) to left operand	a//=b
**=	Exponent AND: Calculate exponent (raise power) value using operands and assign value to left operand	a**=b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b
^=	Performs xOR on operands and assign value to left operand	a^=b
>>=	Performs Bitwise right shift on operand and assign value to left operand	a>>=b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a<<=b

③ Comparison operators -

Operator	Description	Syntax
>	Greater than : True if the left operand is greater than the right	x>y
<	Less than : True if the left operand is less than the right	x<y
==	Equal to : True if both operands are equal	x==y
!=	Not equal to : True if both operands are not equal	x!=y

Operator	Description	Syntax
>=	Greater than or equal to: True if the left operand is greater than or equal to the right	x >= y
<=	Less than or equal to: True if the left operand is less than or equal to the right	x <= y

④ Logical operators -

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands are true	x or y
not	Logical NOT: True if the operand is false	not x

⑤ Identity operators -

operator	Description	Syntax
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. Here, it results in 1 if id(x) equals id(y).	x is y
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. Here, it results in 1 if x is not a member of sequence y.	x is not y

PTO →

⑥ Membership operators -

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise. Here, in results in a 1 if x is a member of sequence y.	x in y
not in	Evaluates to true if does not find a variable in the specified sequence and false otherwise. Here, not in results in a 1 if x is not a member of sequence y.	x not in y

⑦ Bitwise operators -

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands.	(a&b) (means 0000 1100)
| Binary OR	operator copies a bit if it exists in either operand.	(a|b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a^b) = 49 (means 0011 0001)
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a<<2 = 240 (means 1111 0000)
>> Binary Right Shift	The right operands value is moved right by the number of bits specified by the right operand.	a>>2 = 15 (means 0000 1111)

→ PYTHON COLLECTIONS (ARRAYS) :-

There are four collection data types in the Python programming language :

① List - List items are ordered, changeable, and allow duplicate values. They are indexed, the first item has index[0], the second item has index[1] etc. They are created using square brackets.

For example,

```
thislist = ["apple", "banana", "cherry"]
print (thislist)
```

② Tuple - Tuple items are ordered, unchangeable, and allow duplicate values. They are indexed, the first item has index[0], the second item has index[1] etc. They are written using round brackets.

For example,

```
thistuple = ("apple", "banana", "cherry")
print (thistuple)
```

③ Set - Set items are unordered, unchangeable, and do not allow duplicate values. Sets are unordered, so you cannot be sure in which order the items will appear. They are written using curly brackets. For example,

```
thisset = {"apple", "banana", "cherry"}
print (thisset)
```

④ Dictionaries - Dictionary items are ordered, changeable, and does not allow duplicates. They have been presented in key : value pairs, and can be referred to by using the key name. They are written using curly brackets and have keys and values.

For example,

```
thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
print (thisdict)
```

→ PYTHON COLLECTIONS (ARRAYS) :-

① List-

['apple', 'banana', 'cherry']

② Tuple-

('apple', 'banana', 'cherry')

③ Set-

{'banana', 'cherry', 'apple'}

④ Dictionaries-

{'brand': 'ford', 'model': 'mustang', 'year': 1964}

When choosing a collection type, it is useful to understand that properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

→ PYTHON IF ... ELSE :-

① Python conditions and If Statements - Python supports the usual logical conditions from mathematics-

(i) Equals : a == b
(ii) Not equals : a != b
(iii) Less than : a < b
(iv) Less than or equal to : a <= b
(v) Greater than : a > b
(vi) Greater than or equal to : a >= b

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

Syntax for if :-
if expression :
  statement(s)

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

② Elif - The elif keyword is python's way of saying "if the previous conditions were not true, then try this condition". For example,

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

→ PYTHON IF... ELSE :-

① Python conditions and If statement.

b is greater than a

② elif -

a and b are equal

③ Else - The else keyword catches anything which isn't caught by the preceding conditions. For example,

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else
  print("a is greater than b")
```

Syntax :-

```
if expression 1:
  statement (s)
elif expression 2:
  statement (s)
elif expression 3:
  statement (s)
  :       :
else :
  statement (s)
```

④ And - The and keyword is a logical operator and is used to combine conditional statements. For example,

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are true!")
```

⑤ Or - The or keyword is a logical operator and is used to combine conditional statements. For example,

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is true!")
```

⑥ Nested If - You can have if statements inside if statements, this is called nested if statements. For example,

→ PTO

③ Else -
a is greater than b

④ And -
Both conditions are true!

⑤ Or.
At least one of the conditions is TRUE

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

} Nested

⑦ The Pass statement - if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error. For example

```
a = 33
b = 200
if b > a:
  pass # No functionality implemented
```

→ PYTHON WHILE LOOPS :-

With the while loop, we can execute a set of statements as long as a condition is true. For example,

```
i = 1
while i < 6:
  print (i)
  i += 1
```

Syntax for while :-
while test_expression :
  statement(s)
  iteration

The while loop requires relevant variables to be ready, in this example, we need to define an indexing variable, i which we set to 1.

→ PYTHON FOR LOOPS :-

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). This is less like the for keyword in other programming languages and works more like an iterator method as found in other object-oriented programming languages. With the for loop, we can execute a set of set of statements, one for each item in a list, tuple, set, etc. The for loop does not require an indexing variable to set beforehand.

⑥ Nested If :

above ten,
and also above 20!

→ PYTHON WHILE LOOPS :-

1
2
3
4
5

For example,

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

Syntax for for-loop:-
for value in sequence
    body of loop
    statement(s)

→ PYTHON CONTROL STATEMENTS :-

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements -

① break - Terminates the loop statement and transfers execution to the statement immediately following the loop.

② continue - Causes the loop to skip the remainder of its body and immediately retest its condition before reiterating.

③ pass - Used when a statement is required syntactically but you do not want any command or code to execute.

→ PYTHON FUNCTIONS :-

A function is a block of code that only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

① Creating a function - In Python, a function is defined using the def keyword. For example,

```
def my_function1():
    print("Hello from a function!")
```

PTO

→ PYTHON FOR LOOPS :-

apple
banana
cherry

→ PYTHON FUNCTIONS :-

② Calling a function - To call a function, use the function name followed by parenthesis. For example,

```
def my_function1():
    print("Hello from a function!")
my_function1()
```

③ Arguments - Information can be passed into functions as arguments. Arguments are specified after the function name, inside the parenthesis. You can add as many arguments as you want, just seperate them with a comma. The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name -

```
def my_function2(fname):
    print(fname + "Refsnes")
my_function2("Amil")
my_function2("Tobias")
my_function2("Linus")
```

④ Number of Arguments - By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the functions with 2 arguments, not more, and not less. The following function expects 2 arguments, and gets 2 arguments -

```
def my_function3(fname, lname):
    print(fname + " " + lname)
my_function("Amil", "Refsnes")
```

If you try to call the function with 1 or 3 arguments, you will get an error.

PTO

② Calling a function -

Hello from a function!

③ Arguments -

Emil Refsnes

Tobias Refsnes

Linus Refsnes

④ Number of Arguments -

Emil Refsnes

⑤ Arbitrary Arguments, *args - If you do not know how many arguments will be passed into your function, add a * before the parameter name in the function definition. This way the function will receive a tuple of arguments and can access the items accordingly. For example,

```
def my_function4 (*kids)
    print ("The youngest child is " + kids[2])
my_function4 ("Amil", "Tobias", "Linus")
```

⑥ Keyword Arguments - You can also send arguments with the key = value syntax. This way the order of the arguments does not matter. For example,

```
def my_function5 (child3, child2, child1):
    print ("The youngest child is " + child3 )
my_function5 (child1 = "Amil", child2 = "Tobias", child3 = "Linus")
```

The phrase Keyword Arguments are often shortened to kwargs in python documentations.

⑦ Arbitrary Keyword Arguments, **kwargs - If you do not know how many keyword arguments will be passed into your function, add two asterisks : ** before the parameter name in the function definition. This way the function will receive a dictionary of arguments and can access the items accordingly. For example,

```
def my_function6 (**kid)
    print ("His last name is " + kid["lname"])
my_function6 (fname = "Tobias", lname = "Refsnes")
```

Arbitrary Keyword Arguments are often shortened to **kwargs in python documentations.

PTO

⑤ Arbitrary Arguments, * args -
the youngest child is Linus

⑥ Keyword Arguments -
the youngest child is Linus

⑦ Arbitrary Keyword Arguments, ** kwargs -
His last name is Refsnes

⑧ Default Parameter Value - The following example shows how to use a default parameter value. If we call the function without argument, it uses the default value

```
def my_function7 (country = "Norway"):
    print ("I am from " + country)
my_function7 ("Sweden")
my_function7 ("India")
my_function7 ()
my_function7 ("Brazil")
```

⑨ Passing a List as an Argument - You can send any data type of argument to a function (string, number, list, dictionary, etc), and it will be treated as the same data type inside the function. For example, if you send a list as an argument, it will still be a list when it reaches the function -

```
def my_function8 (food):
    for x in food:
        print(x)
fruits = ["apple", "banana", "cherry"]
my_function8 (fruits)
```

⑩ Return Values - To let a function return a value, use the return statement. For example,

```
def my_function9 (x)
    return 5*x
print (my_function9(3))
print (my_function9(5))
print (my_function9(9))
```

→ PYTHON IMPORT STATEMENTS :-

You can use any python source file as a module by executing an import statement in some other python source file. When the interpreter encounters an import statement, it imports the module

⑧ Default Parameter Value:

I am from Sweden
I am from India
I am from Norway
I am from Brazil

⑨ Passing a List as an Argument:

apple
banana
cherry

⑩ Return Values:

15
25
45

if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example,

```
import random # imports random source file
val1 = random.randrange(100);
val2 = random.randrange(100);
print("Following are the two generated random numbers under 100 :-");
print("First :", val1); # Gives a random number less than 100.
print("Second :", val2); # Gives a random number less than 100.
```

→ PYTHON CLASSES AND OBJECTS :-

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A class is like an object constructor, or a "blueprint" for creating objects.

① Create a class - To create a class, use the keyword class:

```
# Create a class named Myclass, with a property named x
class Myclass :
    x = 5
```

② Create object - Now, we can use the class named Myclass to create objects.

```
# Create an object named p1, and print the value of x
p1 = Myclass()
print(p1.x)
```

③ The __init__() Function - The examples above are classes and objects in their simplest form, and are not really useful in real life applications. To understand the meaning of classes, we have to understand the built-in __init__() function. All classes have a function called __init__(), which is always executed when the class is being initiated. Use the __init__() function to assign values to object

→ PYTHON IMPORT STATEMENTS :-

Following are the two generated random number under 100 :-

First : 82

Second : 15

→ PYTHON CLASSES AND OBJECTS :-

② Create Object -

5

properties, or other operations that are necessary to do when the object is being created:-

```
# Create a class named Person, use the __init__() function to assign values for name and age
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

(14) Object Methods - Object can also contain methods. Methods in objects are functions that belong to the object. Let us create a method in the Person class:-

```
# Insert a function that prints a greeting, and execute it on the p1 object
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self)
    print("Hello! My name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

PTO →

③ The __init__() Function -

John
36

④ Object Methods -

Hello! My name is John

⑤ The self Parameter - The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class. It does not have to be named self, you can call it whatever you like, but it has to be the first parameter of any function in the class :-

```
# Use the words mysillyobject and abc instead of self
class Person :
    def __init__(mysillyobject, name, age):
        mysillyobject.name = name
        mysillyobject.age = age

    def myfunc(abc):
        print("Hello! My name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

⑥ Modify Object Properties - You can modify properties on objects like this :-

```
# Set the age of p1 to 40
p1.age = 40
```

⑦ Delete Object Properties - You can delete properties of an object by using the del keyword :-

```
# Delete the age property from the p1 object
del p1.age
```

PTO →

⑥ The self Parameter -

Hello I my name is John

⑧ Delete Objects - You can delete objects by using the del keyword :-

```
# Delete the p1 object
del p1
```

⑨ The pass Statement - class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error:-

```
class Person:
    pass
```

→ PYTHON LIBRARIES :-

A Python library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs. It makes Python Programming simpler and convenient for the programmer. As we don't need to write the same code again and again for different programs. Python libraries play a very vital role in fields of machine learning, Data Science, Data Visualization, etc.

① NumPy - Numerical Python (NumPy) is a python library used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform and matrices. It was created in 2005 by Travis Oliphant, is an open-source project and we can use it freely. It aims to provide an array object that is up to 50x faster than traditional python lists.

② Matplotlib - It is a low-level graph plotting library in Python that serves as a visualization utility. It was created by John D. Hunter, is open source and we can use it freely. It is now a comprehensive library for creating static, animated and interactive visualizations in Python.

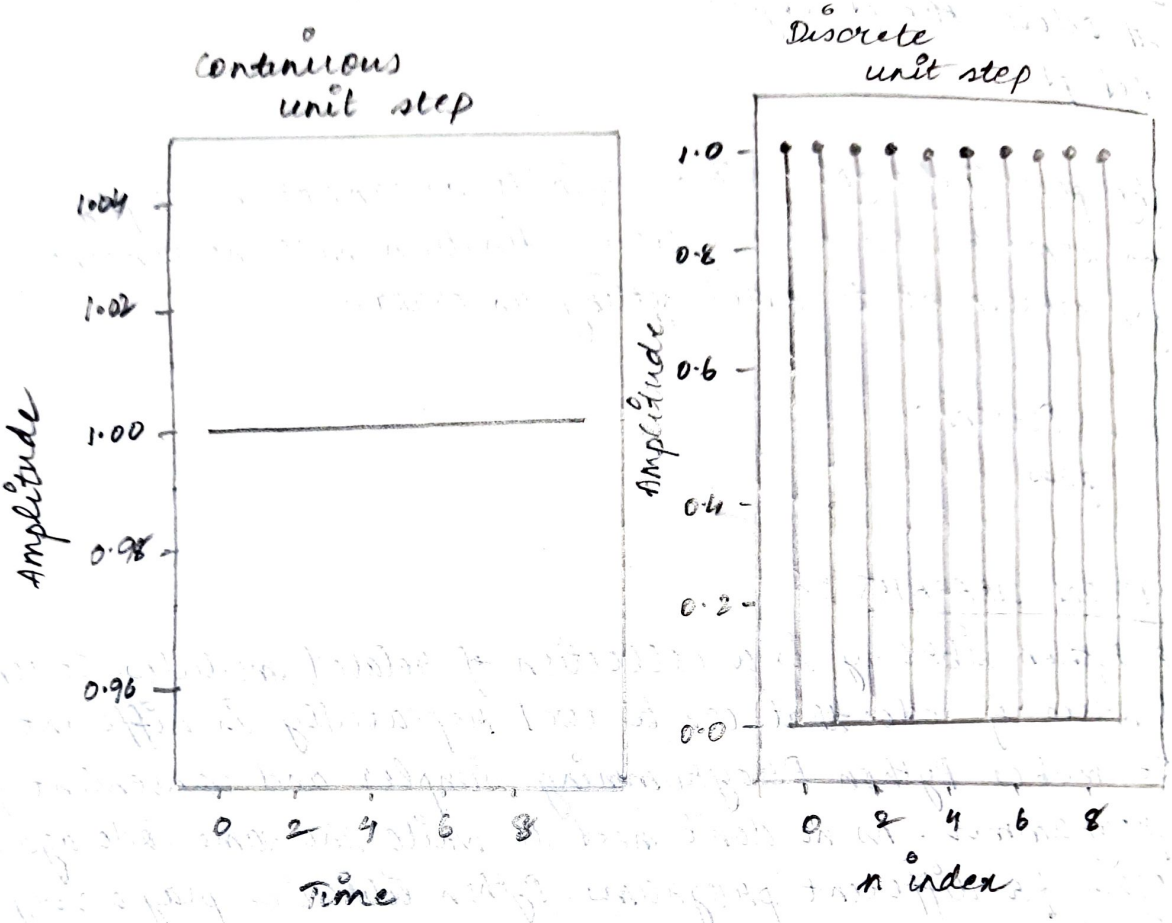
For example,

```
import numpy as np # Support for large, multi-dimensional arrays and matrices
import matplotlib.pyplot as plt # Provides an implicit way of plotting

x = np.ones(10) # Returns a new array of given shape and type, with ones

plt.subplot(121) # subplot(rows, columns, index) describes the figure layout
plt.plot(x) # Used to draw points (markers) in a diagram, draws a line from point to point
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Continuous unit step')

plt.subplot(122) # subplot(rows, columns, index) describes the figure layout
plt.stem(x) # stem plot plots vertical lines at x position covered under the graph from the baseline to y, and places a marker there
plt.xlabel('n index')
plt.ylabel('Amplitude')
plt.title('Discrete unit step')
```

(3) Pandas - It is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. The library is built on top of the Numpy library. Pandas is fast and it has high performance and productivity for users.

PTO →

→ PYTHON LIBRARIES :-



(b) Scipy -

(i) Fundamental Algorithms :- Scipy provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics and many other classes of problems.

(ii) Broadly Applicable :- The algorithms and data structures provided by Scipy are broadly applicable across domains.

(iii) Foundational :- Extends Numpy providing additional tools for array computing and provides specialized data structures, such as sparse matrices and k-dimensional trees.

(iv) Performant :- Scipy wraps highly-optimized implementations written in low-level languages like Fortan, C, and C++. enjoy the flexibility of Python with the speed of compiled code.

(v) Easy to use :- Scipy's high level syntax makes it accessible and productive for programmers from any background or experience level.

(vi) Open source :- Distributed under a liberal BSD license, Scipy is developed and maintained publicly on Github by a vibrant, responsive, and diverse community.

* INFERENCE :
A brief introduction to Python is documented along with the syntaxes and examples for basic concepts and commands.