

① NAME - SANDOSH (CB.EN.V4CLE 20053)

DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCE)

ACADEMIC YEAR - 2020-21 BATCH

LAB TITLE AND CODE : SIGNAL PROCESSING LAB IACCE 281

EXPERIMENT NUMBER : 1

DATE : 13/09/2021

INTRODUCTION TO PYTHON

* AIM :

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

* SOFTWARE REQUIRED :

Synder IDE (Anaconda 3) - 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 even 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

②

than some other programming languages. It runs on an interpreter system, meaning the code can be executed as soon as it is written. This means that prototyping can be very quick and can be treated procedurally, in an object-oriented way or a functional way.

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

+ H ALGORITHM - PYTHON GETTING STARTED AND COMMANDS :

→ P YTHON 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 Environment (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 the time required for application development.

→ 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!")

OUTPUT - 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.

④ Creating a comment - comments starts 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!")

(4)

(5)

- ② Multiline 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!")
```

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 then ignore it, and you have made a multiline comment.

→ PYTHON VARIABLES :-

Variables are containers for storing data values.

- ③ 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)
```

Output :
5
John

③

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)    # OUTPUT - 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))    # OUTPUT - <class 'int'>
print(type(y))    # OUTPUT - <class 'str'>
```

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

⑥

→ 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
```

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

```
print(type(x)) # OUTPUT - <class 'int'>
```

```
print(type(y)) # OUTPUT - <class 'float'>
```

```
print(type(z)) # OUTPUT - <class 'complex'>
```

→ PYTHON ASSIGN VALUES TO MULTIPLE VARIABLES :-

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

```
x, y, z = "orange", "Banana", "cherry"
```

```
print(x)
```

```
print(y)
```

```
print(z)
```

OUTPUT:
orange
Banana
cherry

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)
```

OUTPUT:
orange
orange
orange

→ 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)
```

①

→ Python Operators :-

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

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 operands and assign the result to left operand	$a \% = b$
$\lfloor =$	Divide (floor) AND : Divide left operand with right operand and then assign the value (floor) to left operand	$a \lfloor = b$
$** =$	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	$a \& = b$
$\mid =$	Performs Bitwise OR on operands and assign value to left operand	$a \mid = b$
$\wedge =$	Performs XOR on operands and assign value to left operand	$a \wedge = b$
$>> =$	Performs Bitwise right shift on operands 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	$x < y$

①

Operator	Description	Syntax
$=$	Equal to : True if both operands are equal	$x = y$
\neq	Not equal to : True if both operands are not equal	$x \neq y$
\geq	Greater than or equal to : True if the left operand is greater than or equal to the right	$x \geq y$
\leq	Less than or equal to : True if the left operand is less than or equal to the right	$x \leq y$

④ logical operators -

operator	Description	Syntax
and	Logical AND : True if both the operands are true.	$x \text{ and } y$
or	Logical OR : True if either one or both the operands is true.	$x \text{ or } y$
not	Logical NOT : True if the operand is false	$\text{not } a$

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 $\text{id}(x)$ equals $\text{id}(y)$	$x \text{ is } y$
$is not$	Evaluates to false if the variables on either side of the operator point to the same object and false otherwise. Here, not in results in a 1 if x is not a member of sequence y	$x \text{ is not } y$

⑥ Membership operators -

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	a in y, here in results in a 1 if n is a member of sequence y.
not in	Evaluates to true if does not find a variable in the specified sequence and false otherwise.	a not in y, here not in results in a 1 if a is not a member of sequence 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	It 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 Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a)=61 (means 1100 0011 In 2's complement form due to a signed binary number.)
<< Binary Left Shift	The left operand's value is moved left by the number of bits specified by the right operand.	a<<2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operand's 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) # OUTPUT - ["apple", "banana", "cherry"]
```

- 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 with round brackets. For example,

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

```
print(thistuple) # OUTPUT - ("apple", "banana", "cherry")
```

- 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 in curly brackets. For example,

```
thisset = {"apple", "banana", "cherry"}
```

```
print(thisset) # OUTPUT - {"banana", "cherry", "apple"}
```

- 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 with curly brackets and have keys and values. For example,

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

OUTPUT -
{'brand': 'Ford', 'model': 'Mustang',

'year': 1964}

```
print(thisdict)
```

When choosing a collection type, it is useful to understand the 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 : $a \leq b$

(v) Greater than : $a > b$

(vi) Greater than or equal to : $a \geq b$

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

```
a = 33
```

```
b = 200
```

```
if b > a:
```

```
    print("b is greater than a")
```

```
# OUTPUT - b is greater than a
```

Syntax for If :
if expression:
statement(s)

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")
```

```
# OUTPUT - a and b are equal
```

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

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

(13)

```

a = 200
b = 33
if b > a:
    # If
    print ("b is greater than a")
elif a == b:
    # Else if
    print ("a and b are equal")
else:
    # Else
    print ("a is greater than b")
    # 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: # AND
    print ("Both conditions are True")
    # 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: # OR
    print ("At least one of the conditions is True")
    # 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,

```

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

```

OUTPUT:
Above ten,
and also 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

OUTPUT :-
1
2
3
4
5

Syntax:

while test_expression:

Body of while

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 statements, once for each item in a list, tuple, set, etc. The for loop does not require an indexing variable to set beforehand. For example,

fruits = ["apple", "banana", "cherry"]
for x in fruits:

print(x)

OUTPUT:
apple
banana
cherry

Syntax:

for val in sequence:
loop body

→ 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_function():
 print("Hello from a function!")
```
- ② Calling a function - To call a function, use the function name followed by parenthesis. For example,
- ```
def my_function():
    print("Hello from a function!")
my_function()
# OUTPUT - Hello from a function!
```

- ③ Arguments - Information can be passed into functions as arguments. Arguments are specified after the function names inside the parentheses. You can add as many arguments as you want, just separate 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("Emil")
my_function2("Tobias")
my_function2("Linus")
```

OUTPUT:	Emil Refsnes
	Tobias Refsnes
	Linus Refsnes

- 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 function 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_function3("Emil", "Refsnes")
```

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

```
# OUTPUT - 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("Emil", "Tobias", "Linus")
```

```
# OUTPUT - The youngest child is 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_function_5(child3, child2, child1):
    print("The youngest child is " + child3)
```

```
my_function_5(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

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

OUTPUT - The youngest child is Linus

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_function_6(**kid):
```

```
    print("His last name is " + kid["name"])
```

```
my_function_6(name = "Tobias", lname = "Refsnes")
```

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

OUTPUT - 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 values.

```
def my_function_7(country = "Norway"):
```

```
    print("I am from " + country)
```

```
my_function_7("Sweden")
```

```
my_function_7("India")
```

```
my_function_7()
```

```
my_function_7("Brazil")
```

OUTPUTS

I am from Sweden

I am from India

I am from Norway

I am from Brazil

- ② 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 n in food:
        print(n)
```

```
fruits = ["apple", "banana", "cherry"]
my_function8(fruits)
```

OUTPUT:
apple
banana
cherry

- ③ 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))
```

OUTPUT:
15
25
45

→ 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 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 < 100.
print("Second: ", val2); # Gives a random number < 100.
```

(19)

→ PYTHON LIBRARIES :-

- 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 - Matplotlib 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.

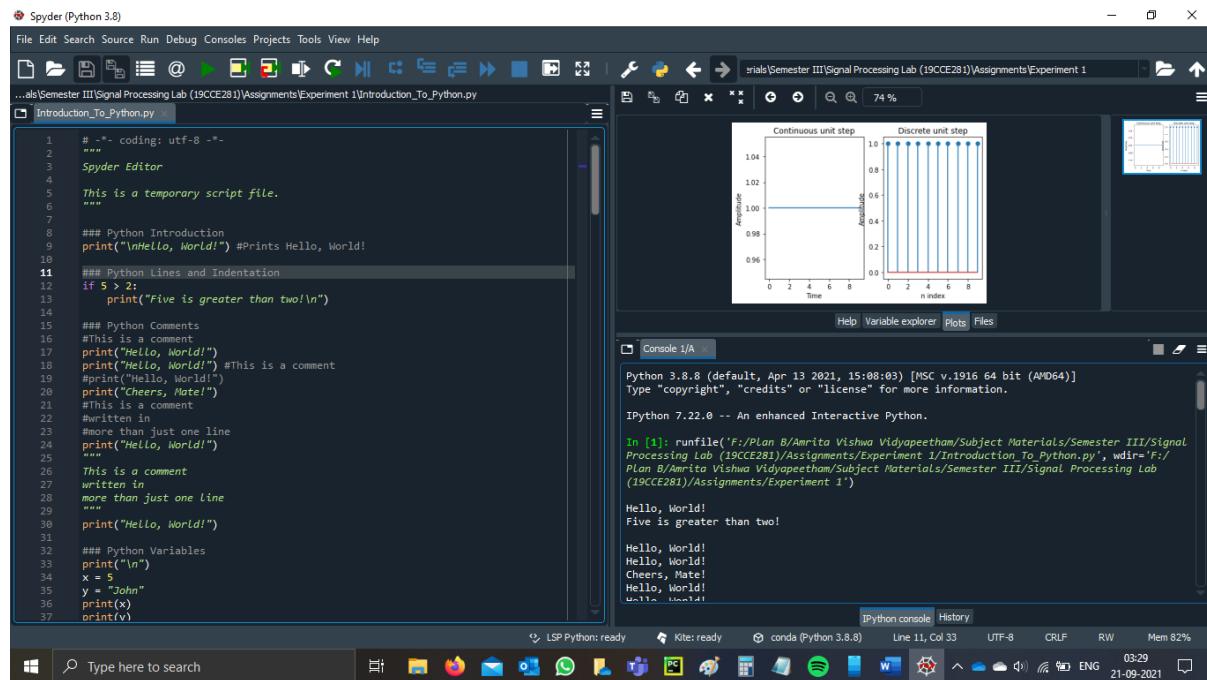
For example, [Sample code]

```
import numpy as np
import matplotlib.pyplot as plt
a = np.ones(10)
plt.subplot(121)
plt.plot(a)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('continuous unit step')
plt.subplot(122)
plt.stem(a)
plt.xlabel('n Index')
plt.ylabel('Amplitude')
plt.title('Discrete unit step')
```

+ INFERENCE :-

A brief introduction to python is documented along with syntaxes and examples for basic concepts and commands.

Python Lines and Indentation



Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

...als\Semester III\Signal Processing Lab (19CCE281)\Assignments\Experiment 1\Introduction_To_Python.py

Introduction_To_Python.py

```
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6
7
8 ### Python Introduction
9 print("\nHello, World!") #Prints Hello, World!
10
11 ### Python Lines and Indentation
12 if 5 > 2:
13     print("Five is greater than two!\n")
14
15 ### Python Comments
16 #This is a comment
17 print("Hello, World!")
18 print("Hello, World!") #This is a comment
19 #print("Hello, World!")
20 print("Cheers, Mate!")
21 #This is a comment
22 #written in
23 #more than just one line
24 print("Hello, World!")
25 """
26 This is a comment
27 written in
28 more than just one line
29 """
30 print("Hello, World!")
31
32 ### Python Variables
33 print("\n")
34 x = 5
35 y = "John"
36 print(x)
37 print(y)
```

Continuous unit step Discrete unit step

Time n index

Console I/A

```
In [1]: runfile('F:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 1/Introduction_To_Python.py', wdir='F:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 1')

Hello, World!
Five is greater than two!

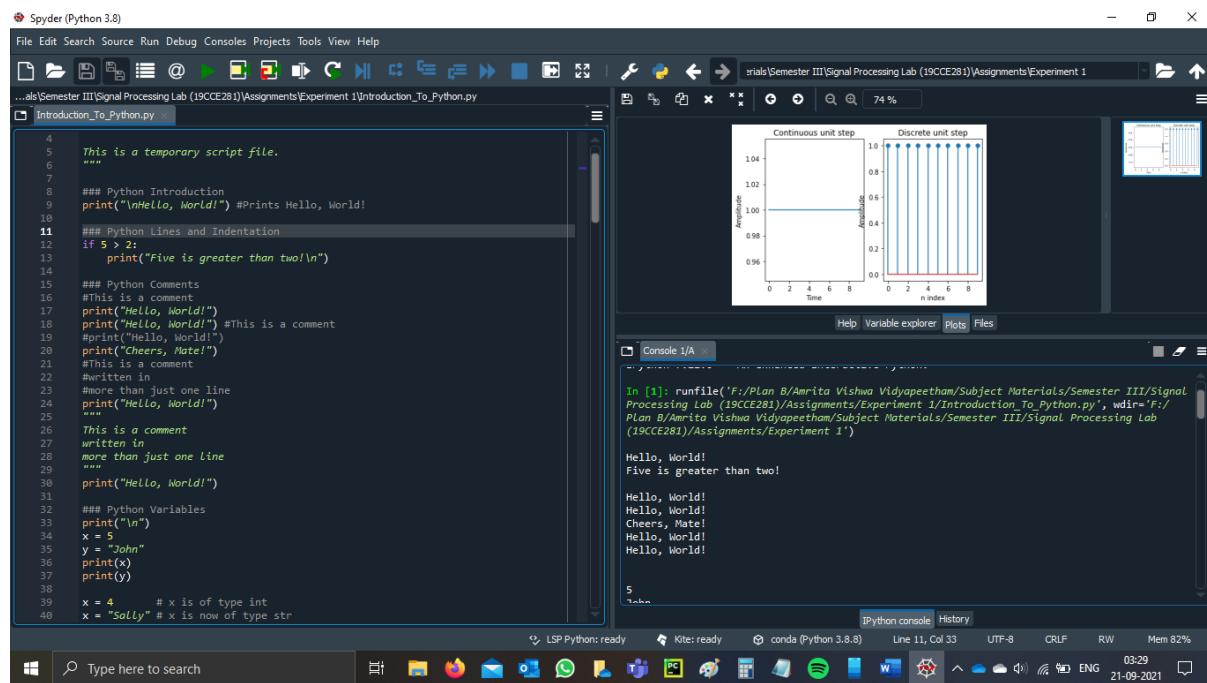
Hello, World!
Hello, World!
Cheers, Mate!
Hello, World!
```

LSP Python: ready Kite: ready conda (Python 3.8.8) Line 11, Col 33 UTF-8 CRLF RW Mem 82%

Type here to search

21-09-2021 03:29 ENG

Python Comments



Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

...als\Semester III\Signal Processing Lab (19CCE281)\Assignments\Experiment 1\Introduction_To_Python.py

Introduction_To_Python.py

```
4
5 This is a temporary script file.
6 """
7
8 ### Python Introduction
9 print("\nHello, World!") #Prints Hello, World!
10
11 ### Python Lines and Indentation
12 if 5 > 2:
13     print("Five is greater than two!\n")
14
15 ### Python Comments
16 #This is a comment
17 print("Hello, World!")
18 print("Hello, World!") #This is a comment
19 #print("Hello, World!")
20 print("Cheers, Mate!")
21 #This is a comment
22 #written in
23 #more than just one line
24 print("Hello, World!")
25 """
26 This is a comment
27 written in
28 more than just one line
29 """
30 print("Hello, World!")
31
32 ### Python Variables
33 print("\n")
34 x = 5
35 y = "John"
36 print(x)
37 print(y)
38
39 x = 4      # x is of type int
40 x = "Sally" # x is now of type str
```

Continuous unit step Discrete unit step

Time n index

Console I/A

```
In [1]: runfile('F:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 1/Introduction_To_Python.py', wdir='F:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab (19CCE281)/Assignments/Experiment 1')

Hello, World!
Five is greater than two!

Hello, World!
Hello, World!
Cheers, Mate!
Hello, World!
Hello, World!
```

LSP Python: ready Kite: ready conda (Python 3.8.8) Line 11, Col 33 UTF-8 CRLF RW Mem 82%

Type here to search

21-09-2021 03:29 ENG

Python Variables

Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

...als\Semester III\Signal Processing Lab (19CCE281)\Assignments\Experiment 1\Introduction_To_Python.py

Introduction_To_Python.py

```
25 """
26     This is a comment
27     written in
28     more than just one line
29 """
30
31 print("Hello, World!")
32
33 ### Python Variables
34 print("\n")
35 x = 5
36 y = "John"
37 print(x)
38 print(y)
39
40 x = 4 # x is of type int
41 x = "Sally" # x is now of type str
42 print(x)
43
44 x = str(3) # x will be '3'
45 y = int(3) # y will be 3
46 z = float(3) # z will be 3.0
47
48 x = 5
49 y = "John"
50 print(type(x))
51 print(type(y))
52
53 x = "John"
54 # is the same as
55 x = 'John'
56
57 # This will create two variables
58 a = 4
59 A = "Sally"
60 #A will not overwrite a
61
62 ### Python Numbers
```

Continuous unit step

Discrete unit step

Time

n index

Axes

Help Variable explorer Plots Files

Console I/A

```
Hello, World!
Hello, World!
Cheers, Mate!
Hello, World!
Hello, World!
```

```
5
John
Sally
<class 'int'>
<class 'str'>
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

LSP Python: ready Kite: ready conda (Python 3.8.8) Line 11, Col 33 UTF-8 CRLF RW Mem 77%

Type here to search

Windows taskbar: File Explorer, Edge, Firefox, Mail, WhatsApp, Task View, PC, Spotify, Word, Spyder, Taskbar icons, Date/Time: 21-09-2021 08:30, Language: ENG

Python Numbers

Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

...als\Semester III\Signal Processing Lab (19CCE281)\Assignments\Experiment 1\Introduction_To_Python.py

Introduction_To_Python.py

```
37 print(y)
38
39 x = 4 # x is of type int
40 x = "Sally" # x is now of type str
41 print(x)
42
43 x = str(3) # x will be '3'
44 y = int(3) # y will be 3
45 z = float(3) # z will be 3.0
46
47 x = 5
48 y = "John"
49 print(type(x))
50 print(type(y))
51
52 x = "John"
53 # is the same as
54 x = 'John'
55
56 # This will create two variables
57 a = 4
58 A = "Sally"
59 #A will not overwrite a
60
61
62 ### Python Numbers
63 print("\n")
64 x = 1 # int
65 y = 2.8 # float
66 z = 1j # complex
67
68 print(type(x))
69 print(type(y))
70 print(type(z))
71
72 ## Python Assign Values to Multiple Variables
73 print("\n")
74 x, y, z = "Orange", "Banana", "Cherry"
```

Continuous unit step

Discrete unit step

Time

n index

Axes

Help Variable explorer Plots Files

Console I/A

```
Hello, World!
Hello, World!
```

```
5
John
Sally
<class 'int'>
<class 'str'>
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

```
Orange
```

LSP Python: ready Kite: ready conda (Python 3.8.8) Line 11, Col 33 UTF-8 CRLF RW Mem 76%

Type here to search

Windows taskbar: File Explorer, Edge, Firefox, Mail, WhatsApp, Task View, PC, Spotify, Word, Spyder, Taskbar icons, Date/Time: 21-09-2021 03:31, Language: ENG

Python Assign Values to Multiple Variables

The screenshot shows the Spyder Python 3.8 IDE interface. The code editor displays a script named `Introduction_To_Python.py` with the following content:

```
49 print(type(x))
50 print(type(y))
51
52 x = "John"
53 # is the same as
54 x = "John"
55
56 # This will create two variables
57 a = 4
58 A = "Sally"
59 #A will not overwrite a
60
61 ### Python Numbers
62 print("\n")
63 x = 1 # int
64 y = 2.8 # float
65 z = 1j # complex
66
67 print(type(x))
68 print(type(y))
69 print(type(z))
70
71 ### Python Assign Values to Multiple Variables
72 print("\n")
73 x, y, z = "Orange", "Banana", "Cherry"
74 print(x)
75 print(y)
76 print(z)
77
78 x = y = z = "Orange"
79 print(x)
80 print(y)
81 print(z)
82
83 ### Python User Input
84 print("\n")
85 username = input("Enter username: ")
```

The IPython console shows the output of the print statements:

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'complex'>

Orange
Banana
Cherry
Orange
Orange
Orange
```

The plots pane displays two side-by-side plots: "Continuous unit step" and "Discrete unit step".

Python User Input

The screenshot shows the Spyder Python 3.8 IDE interface. The code editor displays the same script `Introduction_To_Python.py` as the previous screenshot, but with additional code at the bottom:

```
82
83 ### Python User Input
84 print("\n")
85 username = input("Enter username: ")
86 print("Username is: " + username)
87
88 ### Python Collections (Arrays)
```

The IPython console shows the user input and its output:

```
Enter username: Signal Processing Lab 19CCE281
Username is: Signal Processing Lab 19CCE281
```

The plots pane displays the same "Continuous unit step" and "Discrete unit step" plots as the first screenshot.

Python Collections (Arrays)

The screenshot shows the Spyder IDE interface with the following details:

- Code Editor:** The file `Introduction_To_Python.py` contains Python code demonstrating various data structures and user input. It includes:
 - Basic assignment: `x, y, z = "Orange", "Banana", "Cherry"`
 - Print statements: `print(x), print(y), print(z)`
 - Python User Input: `username = input("Enter username: ")`
 - Python Collections (Lists): `thislist = ["apple", "banana", "cherry"]`
 - Python Collections (Tuples): `thistuple = ("apple", "banana", "cherry")`
 - Python Collections (Sets): `thisset = {"apple", "banana", "cherry"}`
 - Python Dictionaries: `thisdict = { "brand": "Ford", "model": "Mustang", "year": 1964 }`
 - If statements: `a = 33 # If`, `b = 200`, `if b > a:`
 - Else statements: `a = 33 # Elif`, `b = 33`, `if b > a:`, `elif a == b:`, `print("a and b are equal")`
 - Else block: `a = 200 # Else`
 - Nested If: `a = 200 # Or`, `b = 33`, `c = 500`, `if a > b or a > c:`, `print("At least one of the conditions is True")`
 - Nested If: `x = 41 # Nested If`, `if x > 10:`, `print("Above ten,")`
- Plots:** Two plots are displayed in the plots pane:
 - Continuous unit step:** A plot of Amplitude vs Time showing a constant value of 1.0.
 - Discrete unit step:** A plot of Amplitude vs n index showing discrete steps at n=0, 2, 4, 6, 8.
- Console I/A:** The console output shows:
 - User input: `Enter username: Signal Processing Lab 19CCE281`
 - Username response: `Username is: Signal Processing Lab 19CCE281`
 - Printed lists: `['apple', 'banana', 'cherry']`, `('apple', 'banana', 'cherry')`, `{'apple', 'banana', 'cherry'}`, `{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}`
 - If condition check: `b is greater than a`, `a and b are equal`, `a is greater than b`
 - Printed dictionary: `{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}`
 - Final output: `b is greater than a`, `a and b are equal`, `a is greater than b`, `Both conditions are True`, `At least one of the conditions is True`, `Above ten,`, `and also above 20!`
- System Status:** The bottom status bar shows: LSP Python: ready, Kite: ready, conda (Python 3.8.8), Line 11, Col 33, UTF-8, CRLF, RW, Mem 78%, 03:33, 21-09-2021.

Python If ... Else

The screenshot shows the Spyder IDE interface with the following details:

- Code Editor:** The file `Introduction_To_Python.py` contains Python code demonstrating complex conditional logic and nested if statements. It includes:
 - Assignment: `a = 33 # If`, `b = 200`
 - Conditional blocks:
 - `if b > a:`, `print("b is greater than a")`
 - `a = 33 # Elif`, `b = 33`, `if b > a:`, `print("b is greater than a")`
 - `elif a == b:`, `print("a and b are equal")`
 - `a = 200 # Else`
 - `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")`
 - And operator: `a = 200 # And`, `b = 33`, `c = 500`, `if a > b and c > a:`, `print("Both conditions are True")`
 - Or operator: `a = 200 # Or`, `b = 33`, `c = 500`, `if a > b or a > c:`, `print("At least one of the conditions is True")`
 - Nested If: `x = 41 # Nested If`, `if x > 10:`, `print("Above ten,")`
- Plots:** The same two plots as in the first screenshot are displayed: Continuous unit step and Discrete unit step.
- Console I/A:** The console output shows:
 - Printed variables: `{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}`
 - Conditional checks:
 - `b is greater than a`
 - `a and b are equal`
 - `a is greater than b`
 - `Both conditions are True`
 - `At least one of the conditions is True`
 - `Above ten,`
 - `and also above 20!`
 - Final output: `1`, `2`, `3`, `4`
- System Status:** The bottom status bar shows: LSP Python: ready, Kite: ready, conda (Python 3.8.8), Line 11, Col 33, UTF-8, CRLF, RW, Mem 77%, 03:35, 21-09-2021.

Python While Loop

Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Introduction_To_Python.py

```
124     print("a and b are equal")
125 else:
126     print("a is greater than b")
127
128 a = 200 # And
129 b = 33
130 c = 500
131 if a > b and c > a:
132     print("Both conditions are True")
133
134 a = 200 # Or
135 b = 33
136 c = 500
137 if a > b or a > c:
138     print("At least one of the conditions is True")
139
140 x = 41 # Nested If
141 if x > 10:
142     print("Above ten,")
143     if x > 20:
144         print("and also above 20!")
145     else:
146         print("but not above 20.")
147
148 a = 33 # The Pass Statement
149 b = 200
150 if b > a:
151     pass
152
153 i = 1 # While Loop
154 while i < 6:
155     print(i)
156     i += 1
157
158 print("\n")
159 fruits = ["apple", "banana", "cherry"]
160 for x in fruits: # For Loop
```

Continuous unit step Discrete unit step

Absolute Time n index

Console I/A

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

b is greater than a
a and b are equal
a is greater than b
Both conditions are True
At least one of the conditions is True
Above ten,
and also above 20!
1
2
3
4
5

LSP Python: ready Kite: ready conda (Python 3.8.8) Line 11, Col 33 UTF-8 RW Mem 77% 03:36 21-09-2021

Type here to search

Python For Loop

Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Introduction_To_Python.py

```
127
128 a = 200 # And
129 b = 33
130 c = 500
131 if a > b and c > a:
132     print("Both conditions are True")
133
134 a = 200 # Or
135 b = 33
136 c = 500
137 if a > b or a > c:
138     print("At least one of the conditions is True")
139
140 x = 41 # Nested If
141 if x > 10:
142     print("Above ten,")
143     if x > 20:
144         print("and also above 20!")
145     else:
146         print("but not above 20.")
147
148 a = 33 # The Pass Statement
149 b = 200
150 if b > a:
151     pass
152
153 i = 1 # While Loop
154 while i < 6:
155     print(i)
156     i += 1
157
158 print("\n")
159 fruits = ["apple", "banana", "cherry"]
160 for x in fruits: # For Loop
161     print(x)
162
163 ### Python Functions
```

Continuous unit step Discrete unit step

Absolute Time n index

Console I/A

```
Above ten,
and also above 20!
1
2
3
4
5
```

apple
banana
cherry

Hello from a function!
Emil Refsnes
Tobias Refsnes
Linus Refsnes

LSP Python: ready Kite: ready conda (Python 3.8.8) Line 11, Col 33 UTF-8 RW Mem 77% 03:36 21-09-2021

Type here to search

Python Functions

Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Introduction_To_Python.py

```
163     ## Python Functions
164     print("\n")
165     def my_function(): # Creating a Function
166         print("Hello from a function!")
167         my_function() # Calling a Function
168
169     def my_function2(fname): # Arguments
170         print(fname + " Refsnes")
171         my_function2("Emil")
172         my_function2("Tobias")
173         my_function2("Linus")
174
175     def my_function3(fname, lname): # Number of Arguments
176         print(fname + " " + lname)
177         my_function3("Emil", "Refsnes")
178
179     def my_function4(*kids): # Arbitrary Arguments, *args
180         print("The youngest child is " + kids[2])
181         my_function4("Emil", "Tobias", "Linus")
182
183     def my_function5(child1, child2, child3): # Keyword Arguments
184         print("The youngest child is " + child3)
185         my_function5(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
186
187     def my_function6(**kid): # Arbitrary Keyword Arguments, **kwargs
188         print("His last name is " + kid["lname"])
189         my_function6(name = "Tobias", lname = "Refsnes")
190
191     def my_function7(country = "Norway"): # Default Parameter Value
192         print("I am from " + country)
193         my_function7("Sweden")
194         my_function7("India")
195         my_function7()
196         my_function7("Brazil")
197
198     def my_function8(food): # Passing a List as an Argument
199         for x in food:
200             print(x)
201
202         fruits = ["apple", "banana", "cherry"]
203         my_function8(fruits)
204
205         def my_function9(x): # Return Values
206             return 5 * x
207
208         print(my_function9(3))
209         print(my_function9(5))
210         print(my_function9(9))
211
212         ### Python Import Statements
213         print("\n")
214         import random
215         val1 = random.randrange(100);
216         val2 = random.randrange(100);
217         print("Following are the two generated random numbers under 100.")
218         print("First: ",val1);
219         print("Second: ",val2);
220
221         ### Python Libraries
222         import numpy as np
```

Continuous unit step Discrete unit step

Help Variable explorer Plots Files

Console I/A

```
Hello from a function!
Emil Refsnes
Tobias Refsnes
Linus Refsnes
Emil Refsnes
The youngest child is Linus
The youngest child is Linus
His last name is Refsnes
I am from Sweden
I am from India
I am from Norway
I am from Brazil
apple
banana
cherry
```

LSP Python: ready Kite: ready conda (Python 3.8.8) Line 11, Col 33 UTF-8 RW Mem 77% 03:37 21-09-2021

Python Import Statements

Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Introduction_To_Python.py

```
164     print("The youngest child is " + child3)
165     my_function5(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
166
167     def my_function6(**kid): # Arbitrary Keyword Arguments, **kwargs
168         print("His last name is " + kid["lname"])
169         my_function6(name = "Tobias", lname = "Refsnes")
170
171     def my_function7(country = "Norway"): # Default Parameter Value
172         print("I am from " + country)
173         my.function7("Sweden")
174         my.function7("India")
175         my.function7()
176         my.function7("Brazil")
177
178     def my.function8(food): # Passing a List as an Argument
179         for x in food:
180             print(x)
181
182         fruits = ["apple", "banana", "cherry"]
183         my.function8(fruits)
184
185         def my.function9(x): # Return Values
186             return 5 * x
187
188         print(my.function9(3))
189         print(my.function9(5))
190         print(my.function9(9))
191
192         ### Python Import Statements
193         print("\n")
194         import random
195         val1 = random.randrange(100);
196         val2 = random.randrange(100);
197         print("Following are the two generated random numbers under 100.")
198         print("First: ",val1);
199         print("Second: ",val2);
200
201         ### Python Libraries
202         import numpy as np
```

Continuous unit step Discrete unit step

Help Variable explorer Plots Files

Console I/A

```
I am from Norway
I am from Brazil
apple
banana
cherry
15
25
45

Following are the two generated random numbers under 100.
First: 85
Second: 59
```

Figures now render in the Plots pane by default. To make them also appear inline in the Console, uncheck "Auto-Render Plotting" under the Plots pane options menu.

LSP Python: ready Kite: ready conda (Python 3.8.8) Line 11, Col 33 UTF-8 RW Mem 77% 03:37 21-09-2021

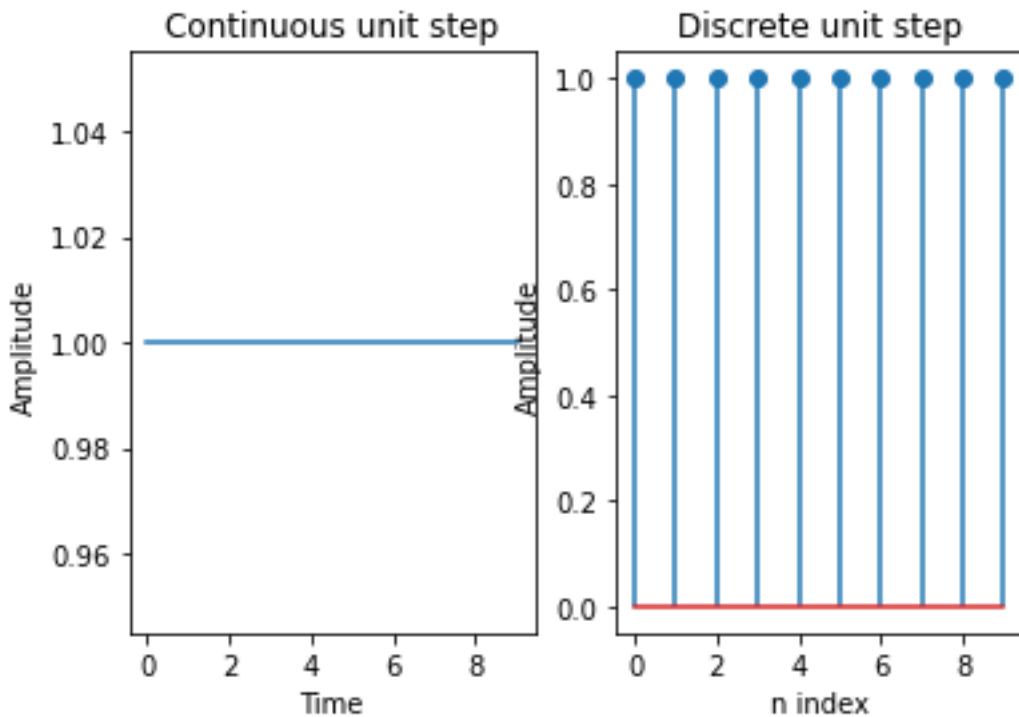
Python Libraries

The screenshot shows the Spyder Python IDE interface. The code editor displays a script named `Introduction_To_Python.py` with the following content:

```
198 def my_function8(food): # Passing a List as an Argument
199     for x in food:
200         print(x)
201     fruits = ["apple", "banana", "cherry"]
202     my_function8(fruits)
203
204 def my_function9(x): # Return Values
205     return 5 * x
206     print(my_function9(3))
207     print(my_function9(5))
208     print(my_function9(9))
209
210 ##### Python import Statements
211 print("\n")
212 import random
213 val1 = random.randrange(100);
214 val2 = random.randrange(100);
215 print("Following are the two generated random numbers under 100.")
216 print("First: ",val1);
217 print("Second: ",val2);
218
219 ##### Python Libraries
220 import numpy as np
221 import matplotlib.pyplot as plt
222 x=np.ones(10)
223 plt.subplot(121)
224 plt.plot(x)
225 plt.xlabel('Time')
226 plt.ylabel('Amplitude')
227 plt.title('Continuous unit step')
228 plt.subplot(122)
229 plt.step(x)
230 plt.xlabel('n index')
231 plt.ylabel('Amplitude')
232 plt.title('Discrete unit step')
```

The plots pane displays two plots: "Continuous unit step" and "Discrete unit step". The "Continuous unit step" plot shows a constant amplitude of 1.00 over time from 0 to 8. The "Discrete unit step" plot shows a discrete signal with amplitude 1.0 at n index values 0 through 9, and 0.0 at all other indices.

The console pane shows the output of the random number generation and the titles of the plots.



Thank You!

```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]  
Type "copyright", "credits" or "license" for more information.
```

```
IPython 7.29.0 -- An enhanced Interactive Python.
```

```
In [1]:      'E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/  
Signal Processing Lab (19CCE281)/Assignments/Experiment 1/Introduction_To_Python.py'  
= 'E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal  
Processing Lab (19CCE281)/Assignments/Experiment 1'
```

```
Hello, World!  
Five is greater than two!
```

```
Hello, World!  
Hello, World!  
Cheers, Mate!  
Hello, World!  
Hello, World!
```

```
5  
John  
Sally  
<class 'int'>  
<class 'str'>
```

```
<class 'int'>  
<class 'float'>  
<class 'complex'>
```

```
Orange  
Banana  
Cherry  
Orange  
Orange  
Orange
```

```
Enter username: Signal Processing Lab 19CCE281  
Username is: Signal Processing Lab 19CCE281
```

```
['apple', 'banana', 'cherry']  
('apple', 'banana', 'cherry')  
{'apple', 'cherry', 'banana'}  
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

```
b is greater than a  
a and b are equal  
a is greater than b  
Both conditions are True
```

```
At least one of the conditions is True
Above ten,
and also above 20!
```

```
1
2
3
4
5
```

```
apple
banana
cherry
```

```
Hello from a function!
Emil Refsnes
Tobias Refsnes
Linus Refsnes
Emil Refsnes
The youngest child is Linus
The youngest child is Linus
His last name is Refsnes
I am from Sweden
I am from India
I am from Norway
I am from Brazil
apple
banana
cherry
15
25
45
```

Following are the two generated random numbers under 100.

```
First: 34
Second: 86
```

Warning

Figures now render in the Plots pane by default. To make them also appear inline in the Console, uncheck "Mute Inline Plotting" under the Plots pane options menu.

In [2]: