Lab #1: Understand Python Basics

1.1 Objectives

Understand the python syntax and execute some basic python programs.

1.2 Pre-Lab

- Introduction to python and its applications.
- Installing Anaconda for Python, set basic environment and install libraries and packages using Anaconda command prompt.
- Understanding the Spyder interface for executing the python codes.

1.2.1 Python

Python is a very popular programming language with many great featuresfor developing Artificial Intelligence. Many Artificial Intelligence developers allaround the world use Python. This lab experiment will provide an introduction to the PythonProgramming Language.

Why the python is so popular Artificial Intelligence developers.

3. Python is a general-purpose programming language conceived in 1989 by Dutchprogrammer Guido van Rossum.
   - Python is free and open source, with development coordinated through thePython Software Foundation.
   - Python has experienced rapid adoption in the last decade and is now one of themost commonly used programming languages.

1.2.2 Common Applications

Python is a general-purpose language used in almost all application domains such as

- Communications
- Web development (Flask and Django covered in the future chapters)
- CGI and graphical user interfaces
- Game development
- AI and data science (very popular)

1.2.3 Setting up Your Python Environment

The core Python package is easy to install but **not** what you should choose for these labs.

We require certain libraries and the scientific programming eco system, which;

- The core installation (e.g., Python 3.7) does not provide.
- Is painful to install one piece at a time (concept of a package manager)

And, we are using Anaconda.

1.2.3.1 Anaconda

To install Anaconda, download the binary and follow the instructions.

Important points:

4. Install the latest version.
5. If you are asked during the installation process whether you would like to make Anaconda your default Python installation, say yes.

Anaconda downloading, installation and setting up environment video tutorial is available at the following link: https://youtu.be/4kpJYuup3lg

1.2.3.2 Test Your Python Installation and Environment

Pre-Lab Task 1

Execute a simple python program to check the python installation and environment setup.

```
# Import two packages first
import numpy as np
import matplotlib.pyplot as plt
# The program starts here.
values = np.random.randn(100)
plt.plot(values)

plt.title('Random Noise using Test Program')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```

After executing the program, you should see an output similar to the following image.

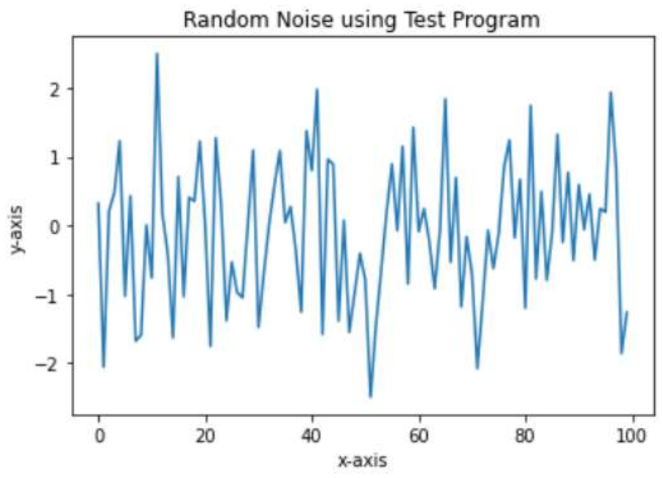
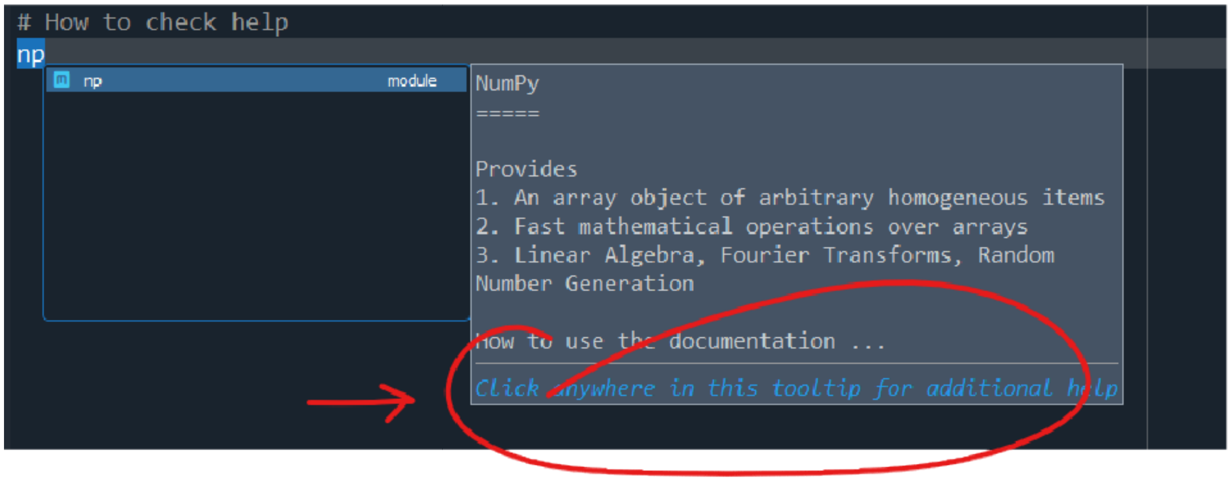


Figure 1: Image displayed by the sample program.

1.2.3.3 Using Help and Documentation in Python

To use help on any function or command, simply type in the prompt and a small window will appear for help.



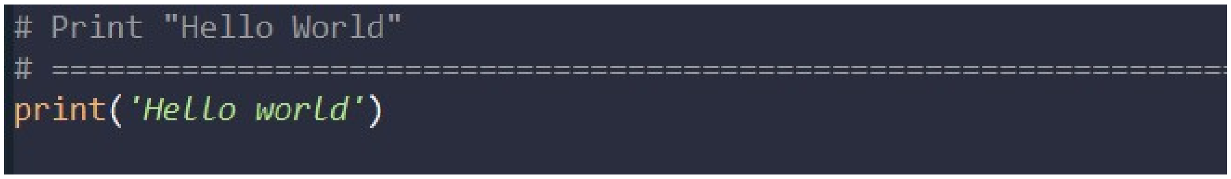
```
# How to check help
np
```

```
NumPy
=====

Provides
1. An array object of arbitrary homogeneous items
2. Fast mathematical operations over arrays
3. Linear Algebra, Fourier Transforms, Random
Number Generation

How to use the documentation ...
Click anywhere in this tooltip for additional help
```

Figure 2: Using Help in Python

1.3 1.2 In-Lab

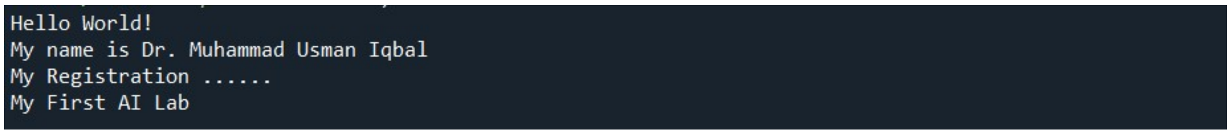
Write the following code in Spyder IDE and learn about printing a string in python.



```
# Print "Hello World"
# ==========================================================
print('Hello world')
```

In-Lab Task 1

Print the following lines using one print command.



```
Hello World!
My name is Dr. Muhammad Usman Iqbal
My Registration ......
My First AI Lab
```

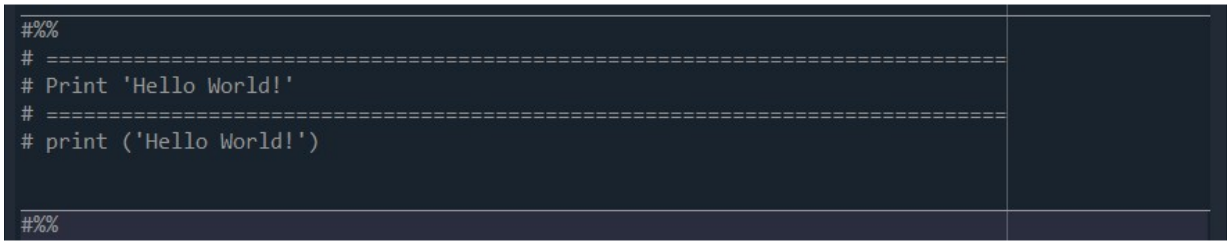
Use the line breaks and a new line in printing the string.

6. \ (Learn to use)
7. \n (Learn to use)
8. You can also add helpful comments to your code with the # symbol. Any linestarting with a # is not executed by the interpreter.

1.3.1.1 Learn to use a Running a Single Cell of Code

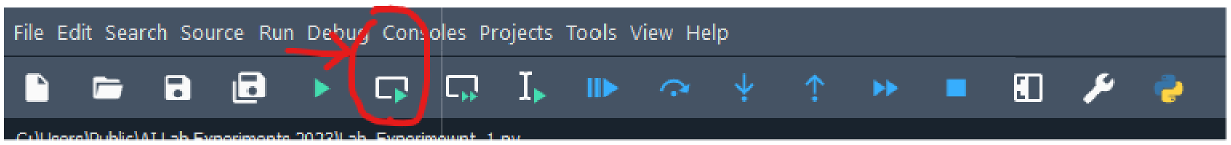
The python code can be sectioned into cells which we can execute separately. The cells can be created by using the following syntax " #%% "

In-Lab Task 2



```
#%%
# ==============================================================================
# Print 'Hello World!'
# ==============================================================================
# print ('Hello World!')

#%%
```

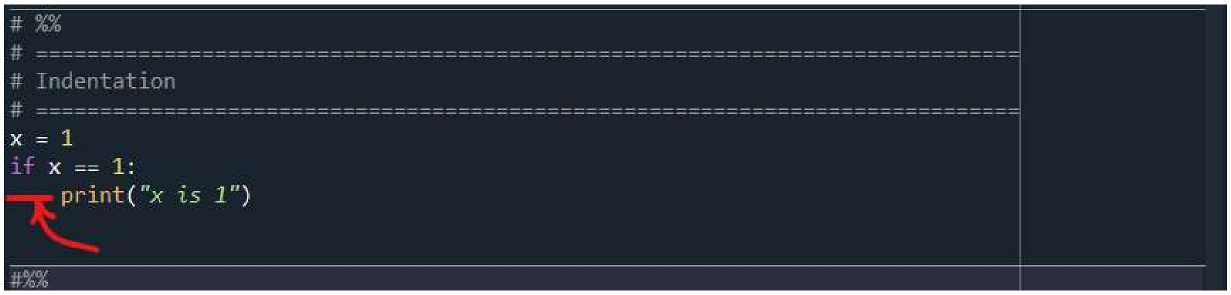
Now using the run only cell in Spyder IDE, we can execute a single cell instead of complete code.


File Edit Search Source Run Debug Consoles Projects Tools View Help

1.3.1.2 Indentation

Python uses indentation to indicate parts of the code that need to be executedtogether. Both tabs and spaces (usually four per level) are supported, and mypreference is for tabs.

In-Lab Task 3



```
# %%
# ==============================================================================
# Indentation
# ==============================================================================
x = 1
if x == 1:
    print("x is 1")

#%%
```

To assign a variable with a specific value, use =.

To test whether a variable has a specific value, use the Boolean operators:

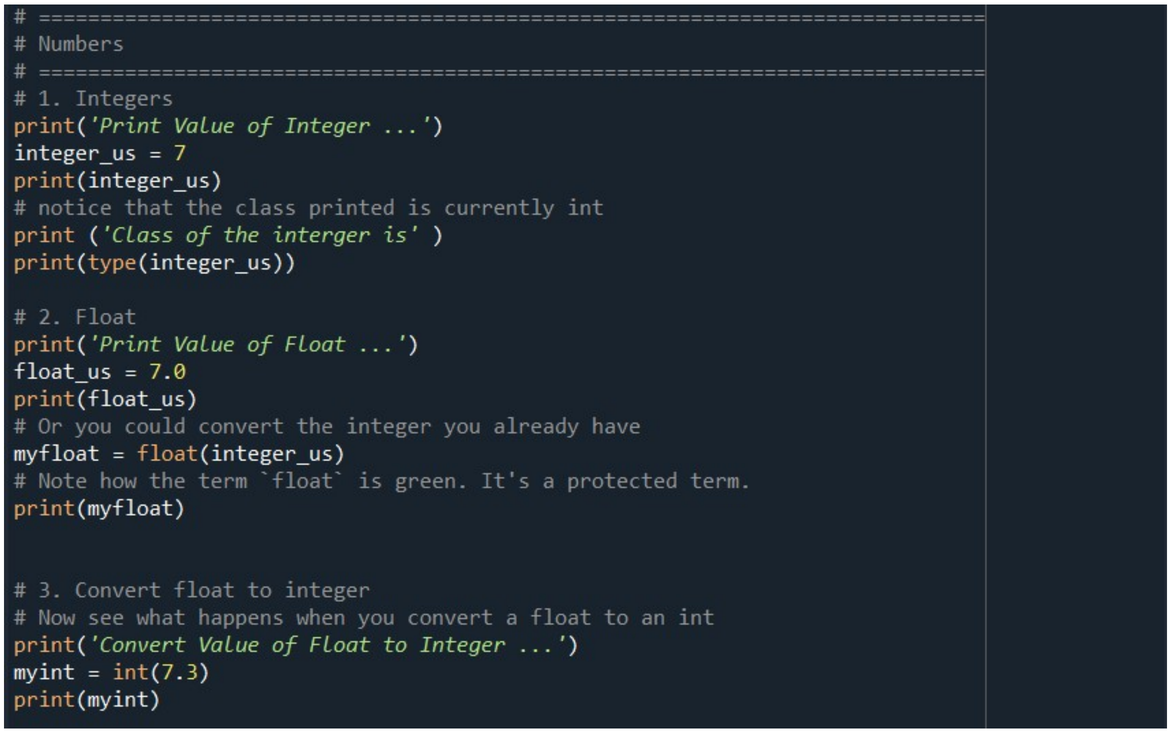
- equal: ==
- not equal: !=
- greater than: >
- less than: <

1.3.1.3 Variables and Types

Python is not “statically-typed.” This means you do not have to declare all yourvariables before you can use them. You can create new variables whenever you want. Python is also “object-oriented,” which means that every variable is an object.

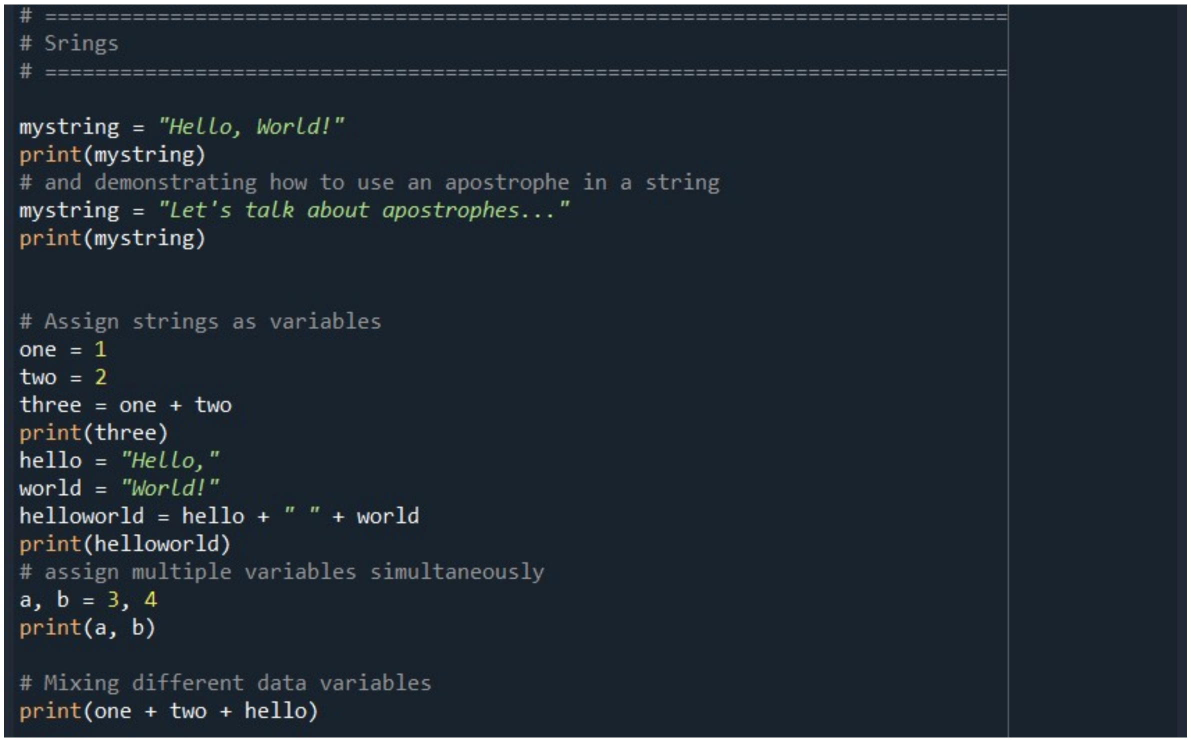
In-Lab Task 4

Numbers



```python
# ================================================================================
# Numbers
# ================================================================================
# 1. Integers
print('Print Value of Integer ...')
integer_us = 7
print(integer_us)
# notice that the class printed is currently int
print ('Class of the interger is' )
print(type(integer_us))

# 2. Float
print('Print Value of Float ...')
float_us = 7.0
print(float_us)
# Or you could convert the integer you already have
myfloat = float(integer_us)
# Note how the term `float` is green. It's a protected term.
print(myfloat)


# 3. Convert float to integer
# Now see what happens when you convert a float to an int
print('Convert Value of Float to Integer ...')
myint = int(7.3)
print(myint)
```

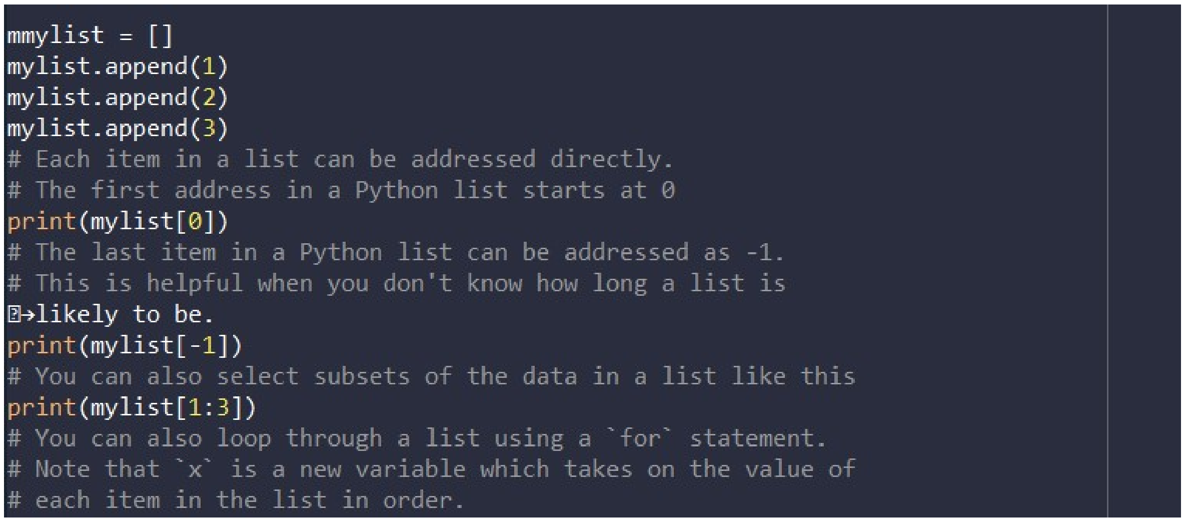
Strings



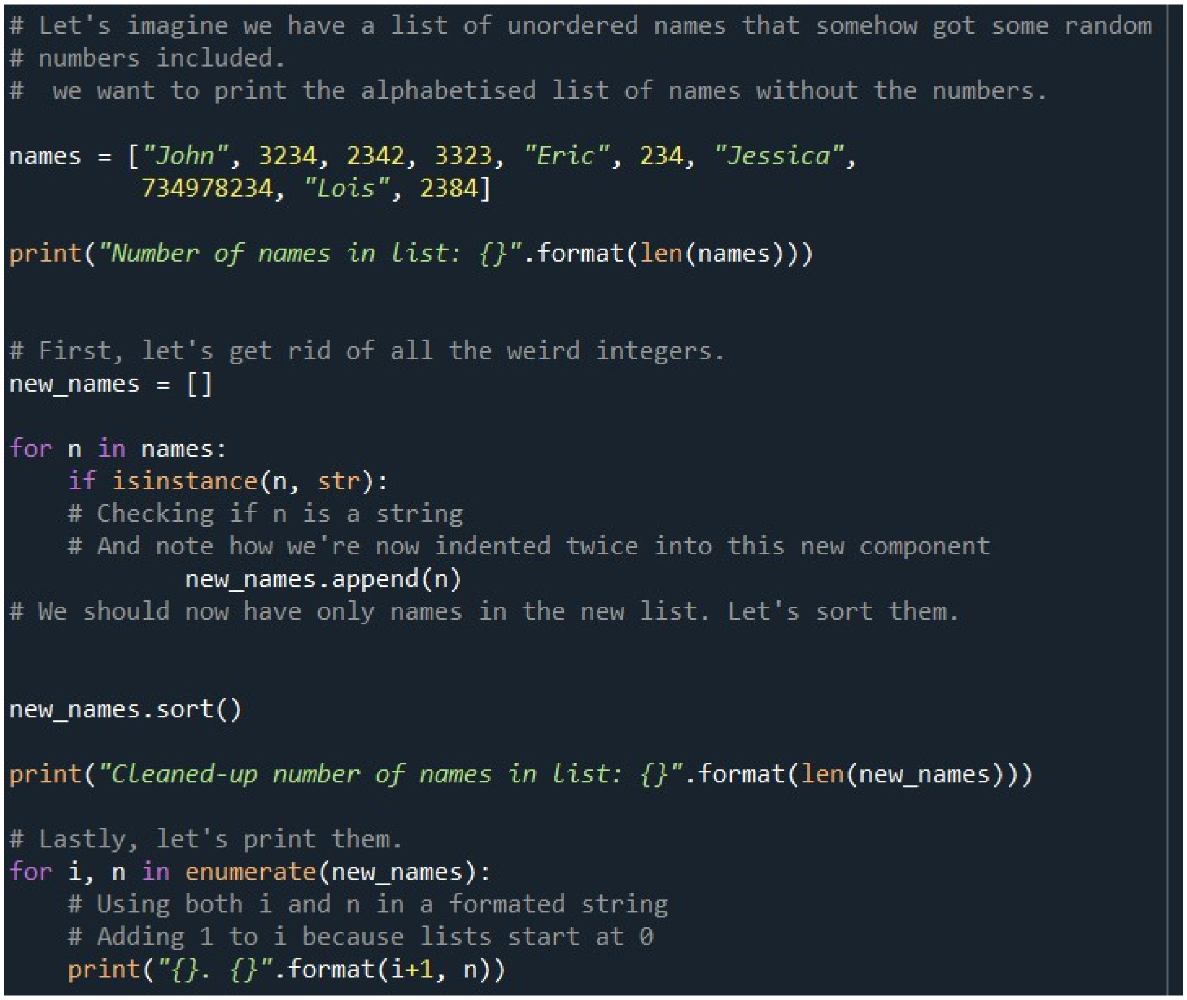
```python
# ================================================================================
# Srings
# ================================================================================

mystring = "Hello, World!"
print(mystring)
# and demonstrating how to use an apostrophe in a string
mystring = "Let's talk about apostrophes..."
print(mystring)


# Assign strings as variables
one = 1
two = 2
three = one + two
print(three)
hello = "Hello,"
world = "World!"
helloworld = hello + " " + world
print(helloworld)
# assign multiple variables simultaneously
a, b = 3, 4
print(a, b)

# Mixing different data variables
print(one + two + hello)
```

Lists

`Lists` are an ordered list of any type of variable. You can combine as manyvariables as you like, and they could even be of multiple data types. Ordinarily,unless you have a specific reason to do so, lists will contain variables of one type.You can also iterate over a list (use each item in a list in sequence).A list is placed between square brackets: `[]`.



```
mmylist = []
mylist.append(1)
mylist.append(2)
mylist.append(3)
# Each item in a list can be addressed directly.
# The first address in a Python list starts at 0
print(mylist[0])
# The last item in a Python list can be addressed as -1.
# This is helpful when you don't know how long a list is
⍰→likely to be.
print(mylist[-1])
# You can also select subsets of the data in a list like this
print(mylist[1:3])
# You can also loop through a list using a `for` statement.
# Note that `x` is a new variable which takes on the value of
# each item in the list in order.
```



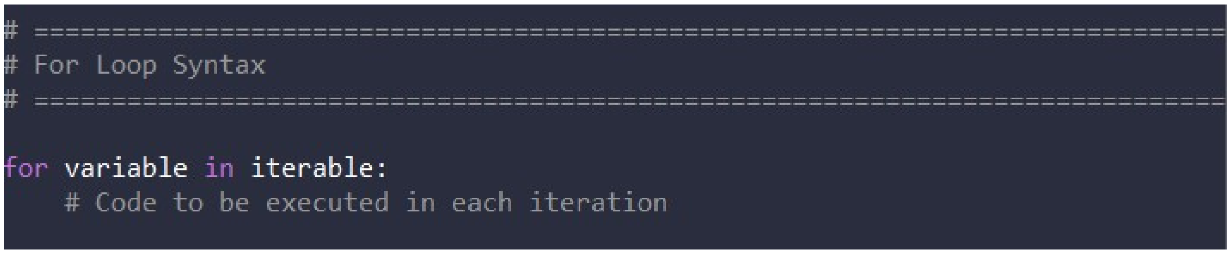
```
# Let's imagine we have a list of unordered names that somehow got some random
# numbers included.
#  we want to print the alphabetised list of names without the numbers.

names = ["John", 3234, 2342, 3323, "Eric", 234, "Jessica",
         734978234, "Lois", 2384]

print("Number of names in list: {}".format(len(names)))


# First, let's get rid of all the weird integers.
new_names = []

for n in names:
    if isinstance(n, str):
    # Checking if n is a string
    # And note how we're now indented twice into this new component
            new_names.append(n)
# We should now have only names in the new list. Let's sort them.


new_names.sort()

print("Cleaned-up number of names in list: {}".format(len(new_names)))

# Lastly, let's print them.
for i, n in enumerate(new_names):
    # Using both i and n in a formated string
    # Adding 1 to i because lists start at 0
    print("{}. {}".format(i+1, n))
```

1.4 1.3 Post-Lab

Post-Lab Task

Understand the following syntax for the for loop, while loop and if condition in python and sample program for each.

for Loop



```
# ==============================================================================
# For Loop Syntax
# ==============================================================================

for variable in iterable:
    # Code to be executed in each iteration
```

for: *This is the keyword that indicates the start of a for loop.*

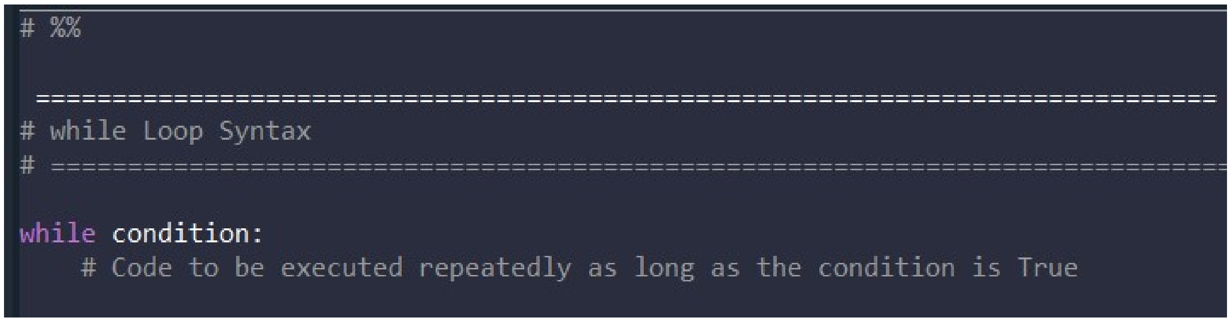
variable: *This is a variable that takes on the value of each item in the iterable during each iteration of the loop. You can choose any valid variable name.*

in: *This is the keyword used to specify that you are iterating over the elements of the iterable.*

iterable: *This is an object that you want to iterate over. It can be a list, tuple, string, dictionary, range, or any other iterable object.*

: (colon): *A colon is used to denote the beginning of the indented block of code that will be executed in each iteration.*

while Loop



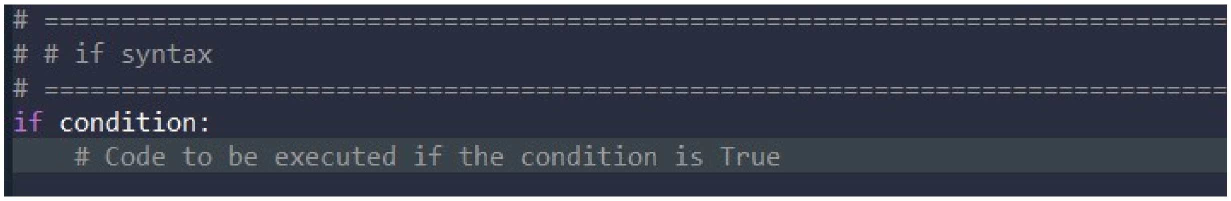
```
# %%

 ==============================================================================
# while Loop Syntax
# ==============================================================================

while condition:
    # Code to be executed repeatedly as long as the condition is True
```

while:This is the keyword that indicates the start of a while loop.

condition:This is a Boolean expression that is evaluated before each iteration of the loop. If the condition is True, the loop continues to execute; if it's False, the loop terminates.

: (colon): *A colon is used to denote the beginning of the indented block of code that will be executed in each iteration.*

if Syntax



```
# ================================================================================
# # if syntax
# ================================================================================
if condition:
    # Code to be executed if the condition is True
```

if: This is the keyword that indicates the start of an if statement.

condition: This is a Boolean expression that is evaluated. If the condition is True, the code block indented below the if statement is executed; if it's False, the code block is skipped.

: (colon): A colon is used to denote the beginning of the indented block of code that will be executed if the condition specified in the if statement is True.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: ________________________________ Date: _____________________

Lab # 2: Python Data Types, Basic Operators, Logical Conditions and Loops

2.1 Objectives

Understand the Python Data Types, Basic Operators, Logical Conditions and Loops

2.2 Pre-Lab

- Review the python data types studied in Lab Experiment 1
- Review basic loops syntax studied in Lab Experiment 1

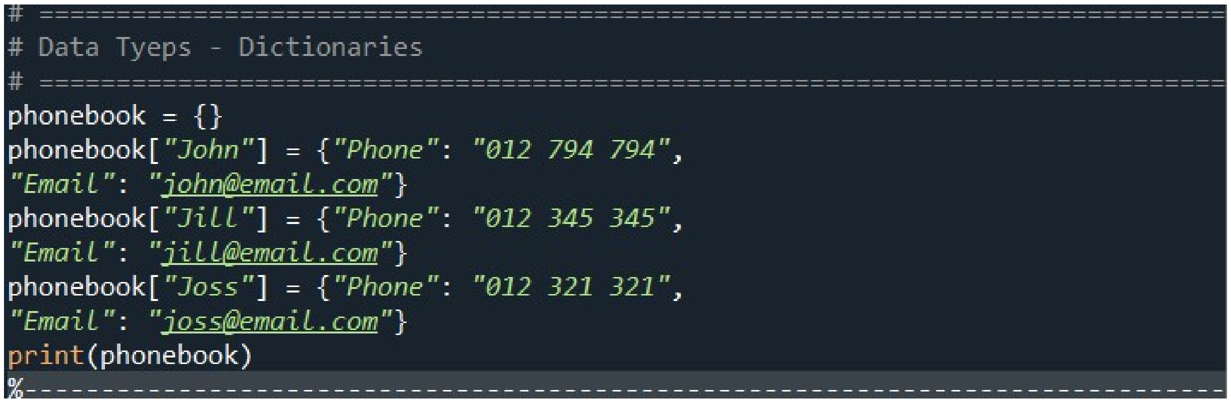
1.3 In-Lab

2.2.1 Dictionaries

Dictionaries are one of the most useful and versatile data types in Python. They aresimilar to arrays but consist of ***key:value*** pairs. Each value stored in a dictionaryis accessed by its key, and the value can be any sort of object (string, number, list,etc.).
This allows you to create structured records. Dictionaries are placed within {}.

In-Lab Task 1

Create the dictionaries according to the following code and display results. You should use names and phone number of your friends.

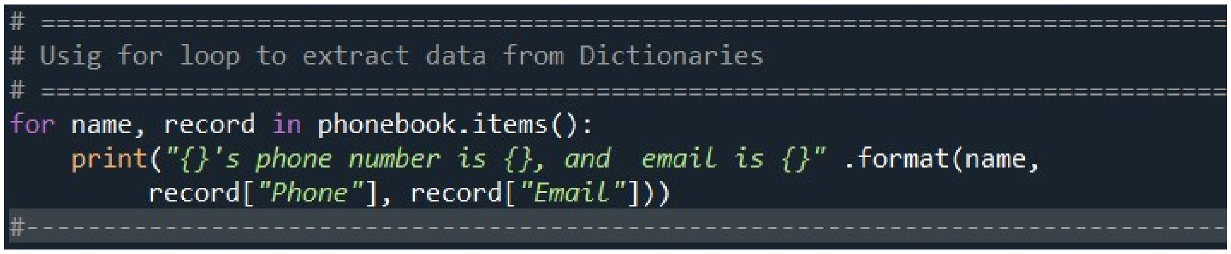


```
# ==========================================================================
# Data Tyeps - Dictionaries
# ==========================================================================
phonebook = {}
phonebook["John"] = {"Phone": "012 794 794",
"Email": "john@email.com"}
phonebook["Jill"] = {"Phone": "012 345 345",
"Email": "jill@email.com"}
phonebook["Joss"] = {"Phone": "012 321 321",
"Email": "joss@email.com"}
print(phonebook)
%---------------------------------------------------------------------------
```

You can iterate over a dictionary just like a list, using the dot term***.items().***

In-Lab Task 2

Use for loop to print data from the dictionary.



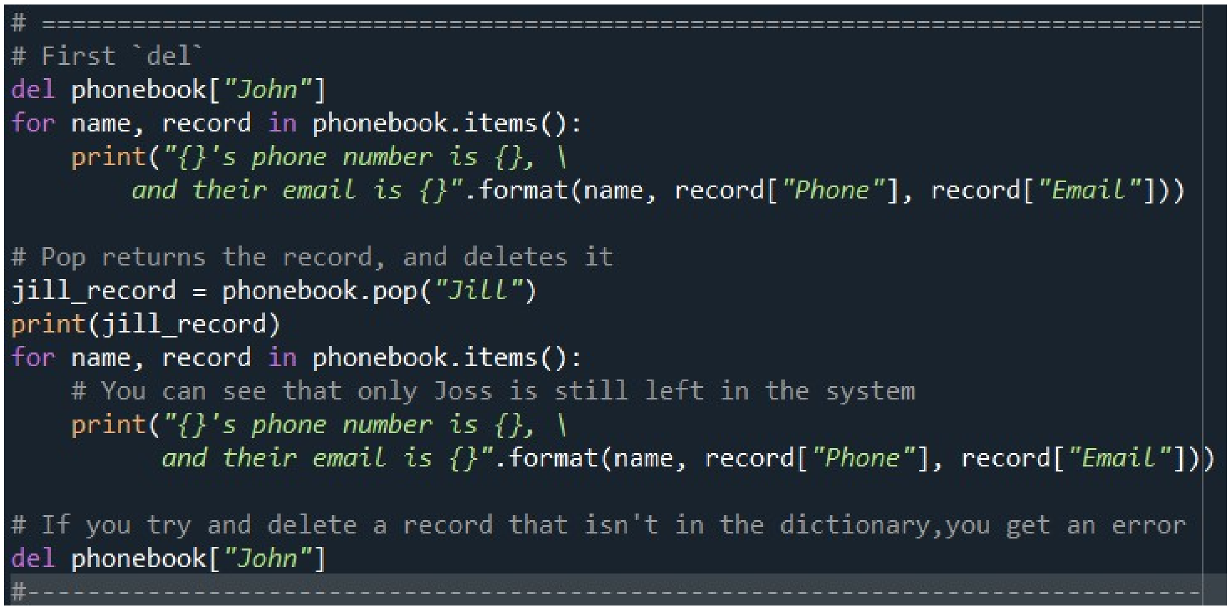
```
# ==========================================================================
# Usig for loop to extract data from Dictionaries
# ==========================================================================
for name, record in phonebook.items():
    print("{}'s phone number is {}, and  email is {}" .format(name,
          record["Phone"], record["Email"]))
#---------------------------------------------------------------------------
```

Similarly, you add new records as shown above, and you remove records with del or pop.

They each have a different effect.

In-Lab Task 3

Learn and practice the commands ‘del’ and ‘pop’ for dictionaries.



```
# ============================================================================
# First `del`
del phonebook["John"]
for name, record in phonebook.items():
    print("{}'s phone number is {}, \
        and their email is {}".format(name, record["Phone"], record["Email"]))

# Pop returns the record, and deletes it
jill_record = phonebook.pop("Jill")
print(jill_record)
for name, record in phonebook.items():
    # You can see that only Joss is still left in the system
    print("{}'s phone number is {}, \
          and their email is {}".format(name, record["Phone"], record["Email"]))

# If you try and delete a record that isn't in the dictionary,you get an error
del phonebook["John"]
#-----------------------------------------------------------------------------
```

2.2.2 Basic Operators

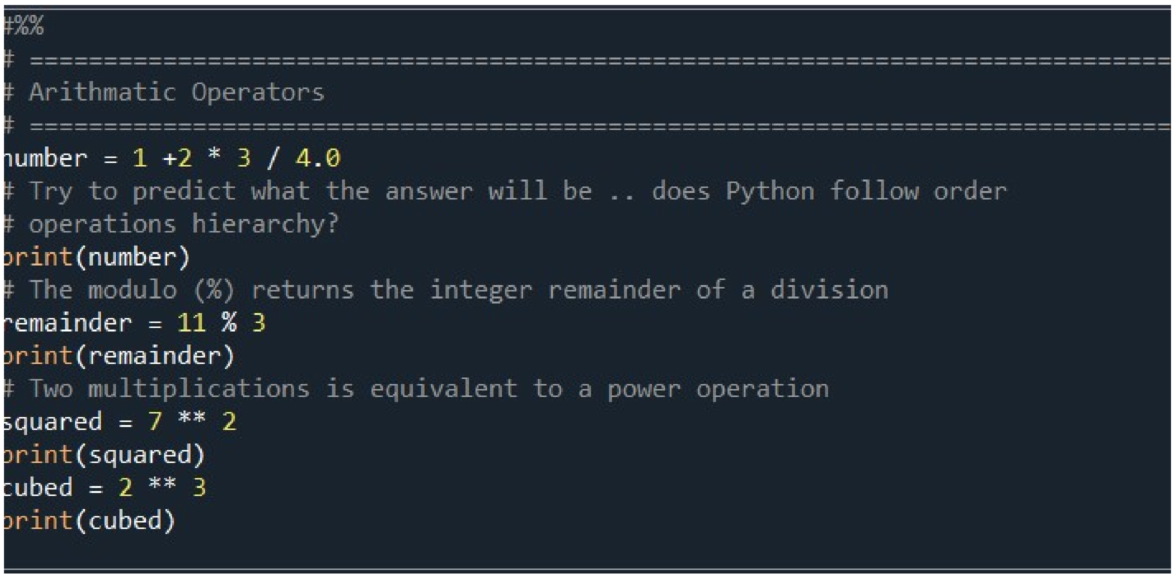
Operators are the various algebraic symbols (such as +, -, *, /, %, etc.). Once ’youhave learned the syntax, programming is mostly mathematics.

Arithmetic Operators

As you would expect, you can use the various mathematical operators with numbers(both integers and floats).

In-Lab Task 4

Learn and practice the arithmetic operators for python programming.

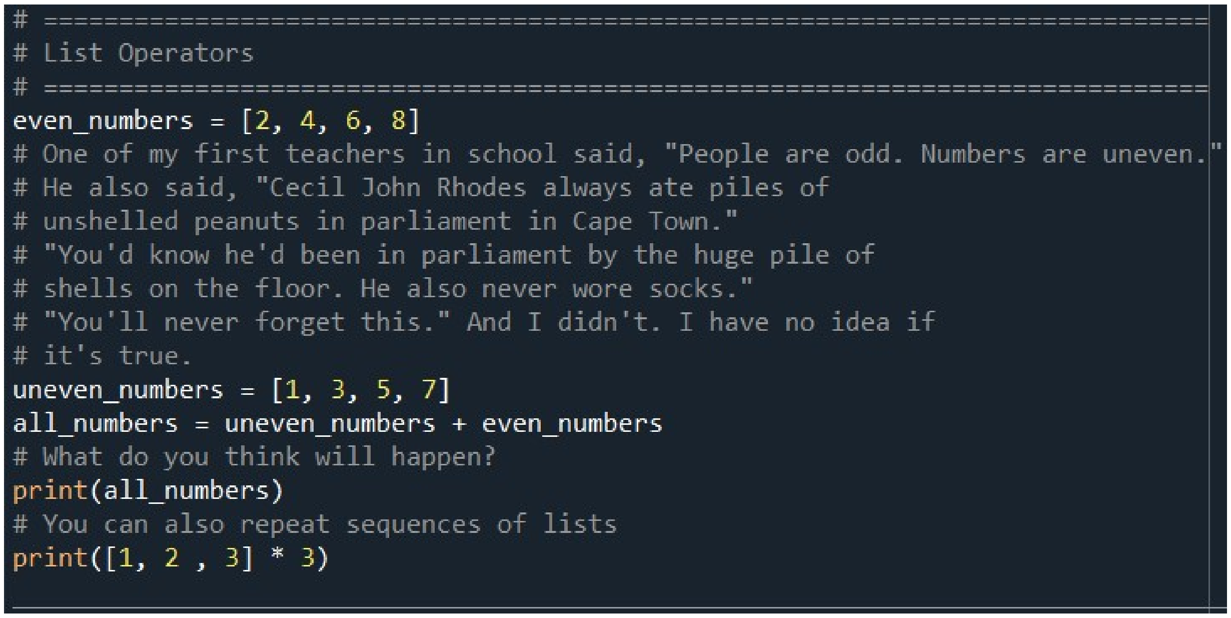


```
#%%
# ============================================================================
# Arithmatic Operators
# ============================================================================
umber = 1 +2 * 3 / 4.0
# Try to predict what the answer will be .. does Python follow order
# operations hierarchy?
rint(number)
# The modulo (%) returns the integer remainder of a division
emainder = 11 % 3
rint(remainder)
# Two multiplications is equivalent to a power operation
quared = 7 ** 2
rint(squared)
ubed = 2 ** 3
rint(cubed)
```

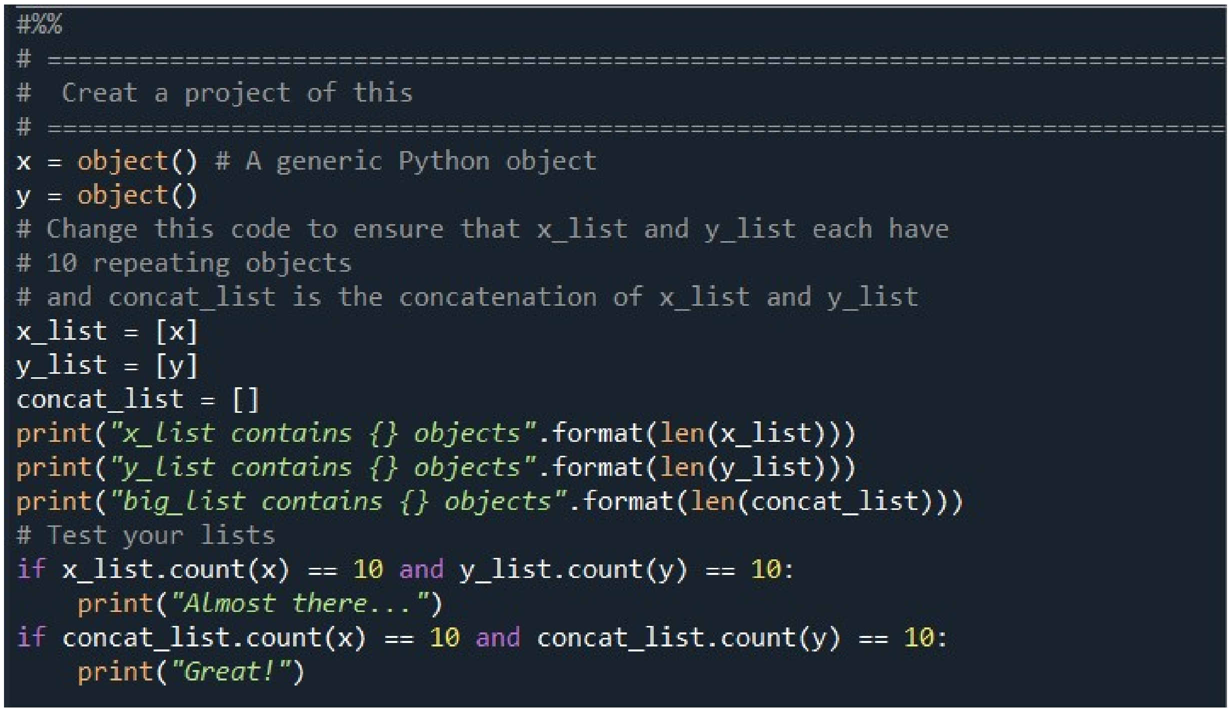
List Operators

In-Lab Task 4

Learn and practice the list operators for python programming.



```python
# ==============================================================================
# List Operators
# ==============================================================================
even_numbers = [2, 4, 6, 8]
# One of my first teachers in school said, "People are odd. Numbers are uneven."
# He also said, "Cecil John Rhodes always ate piles of
# unshelled peanuts in parliament in Cape Town."
# "You'd know he'd been in parliament by the huge pile of
# shells on the floor. He also never wore socks."
# "You'll never forget this." And I didn't. I have no idea if
# it's true.
uneven_numbers = [1, 3, 5, 7]
all_numbers = uneven_numbers + even_numbers
# What do you think will happen?
print(all_numbers)
# You can also repeat sequences of lists
print([1, 2 , 3] * 3)
```



```python
#%%
# ==============================================================================
#  Creat a project of this
# ==============================================================================
x = object() # A generic Python object
y = object()
# Change this code to ensure that x_list and y_list each have
# 10 repeating objects
# and concat_list is the concatenation of x_list and y_list
x_list = [x]
y_list = [y]
concat_list = []
print("x_list contains {} objects".format(len(x_list)))
print("y_list contains {} objects".format(len(y_list)))
print("big_list contains {} objects".format(len(concat_list)))
# Test your lists
if x_list.count(x) == 10 and y_list.count(y) == 10:
    print("Almost there...")
if concat_list.count(x) == 10 and concat_list.count(y) == 10:
    print("Great!")
```

String Operators

You can do a surprising amount with operators on strings.

.

In-Lab Task 5

Print the following using string operators. This is a self-learning task.

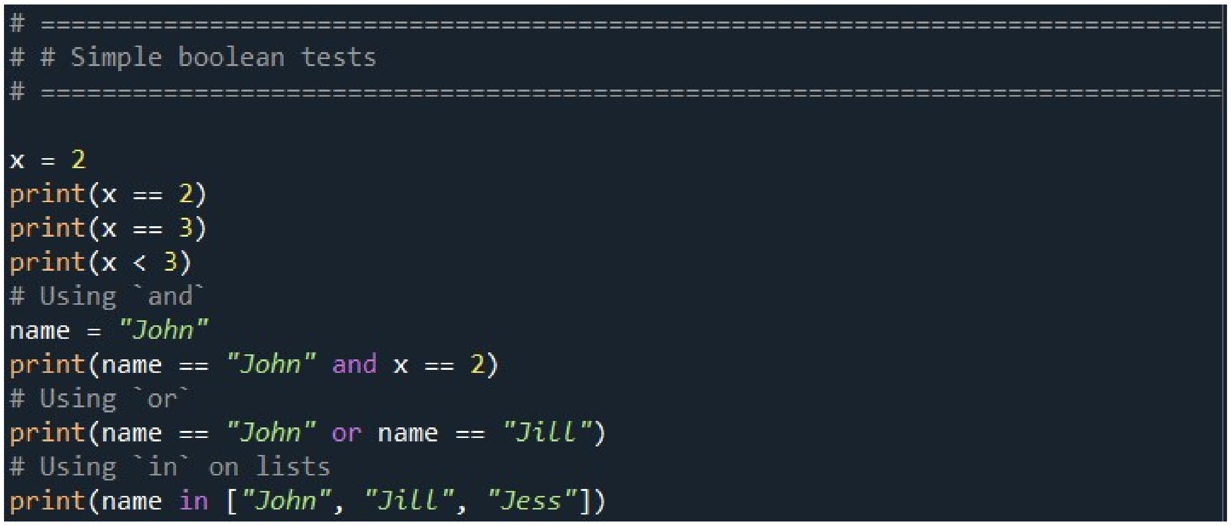
```
Hello, World!
Hello Hello Hello Hello Hello Hello Hello
```

Logical Conditions

In the section on Indentation, you were introduced to the if statement and the setof boolean operators that allow you to test different variables against each other.To that list of Boolean operators are added a new set of comparisons: and, or,and in.

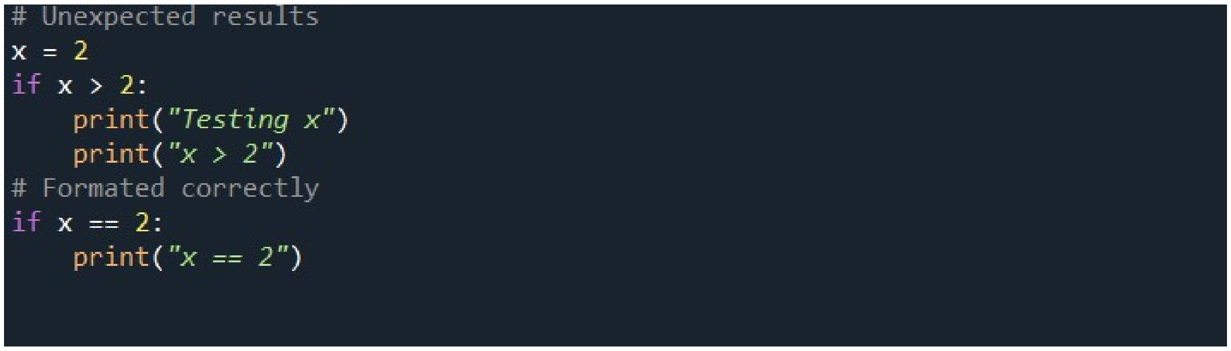
In-Lab Task 6

Learn and practice the Boolean operators for python programming.



```python
# ===============================================================================
# # Simple boolean tests
# ===============================================================================

x = 2
print(x == 2)
print(x == 3)
print(x < 3)
# Using `and`
name = "John"
print(name == "John" and x == 2)
# Using `or`
print(name == "John" or name == "Jill")
# Using `in` on lists
print(name in ["John", "Jill", "Jess"])
```

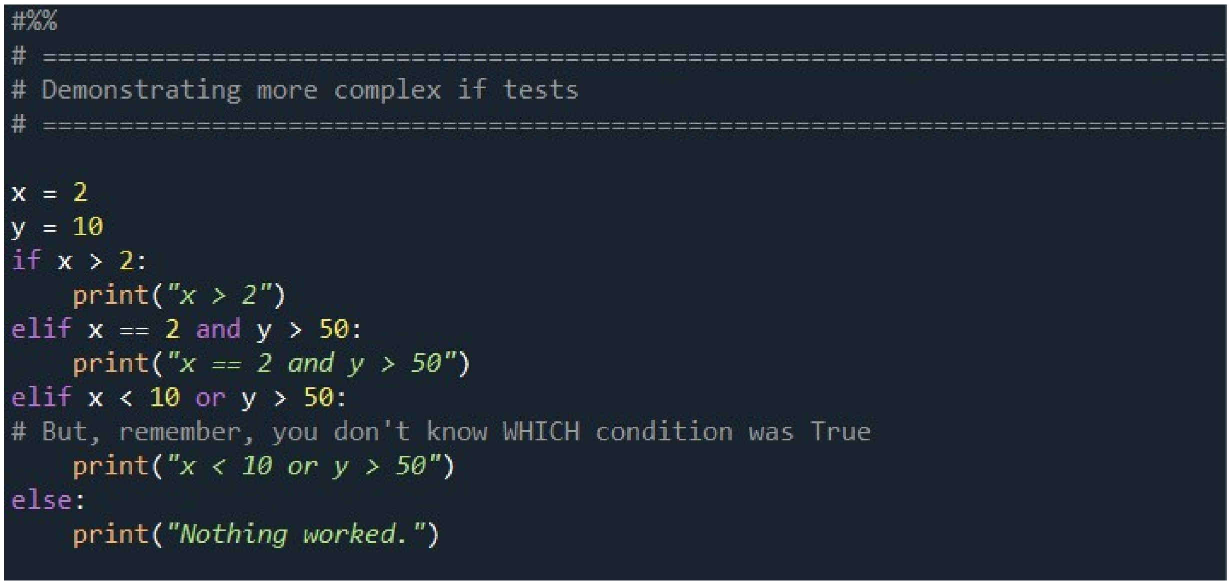
These can be used to create nuanced comparisons using if. You can use a seriesof comparisons with if, elseif, and else.Remember that code must be indented correctly, or you will get unexpectedbehavior.



```python
# Unexpected results
x = 2
if x > 2:
    print("Testing x")
    print("x > 2")
# Formated correctly
if x == 2:
    print("x == 2")
```

In-Lab Task 7

Demonstrate complex if-else conditions for python programming and also explore the ‘not’ and ‘is’.

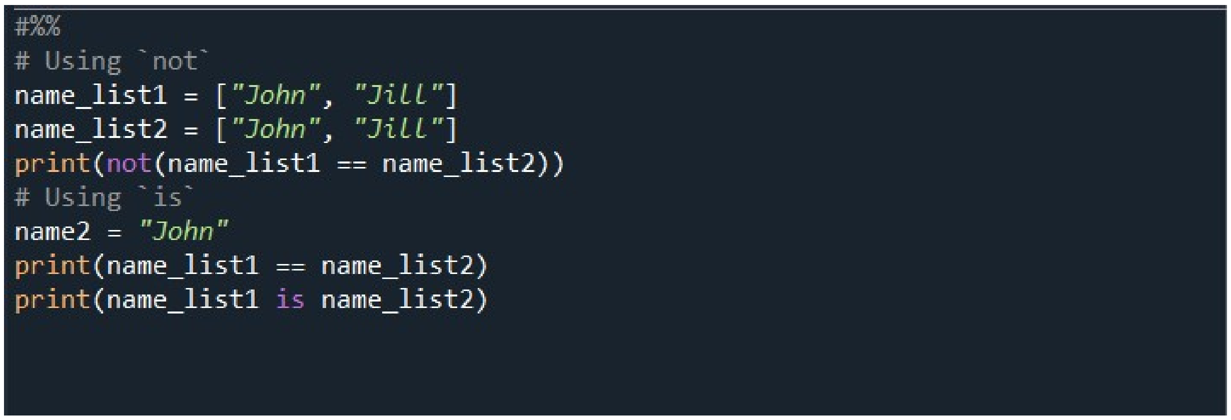


```
#%%
# ==============================================================================
# Demonstrating more complex if tests
# ==============================================================================

x = 2
y = 10
if x > 2:
    print("x > 2")
elif x == 2 and y > 50:
    print("x == 2 and y > 50")
elif x < 10 or y > 50:
# But, remember, you don't know WHICH condition was True
    print("x < 10 or y > 50")
else:
    print("Nothing worked.")
```

Two special cases are not and is.

9. not is used to get the opposite of a particular Boolean test, e.g., not(False) returns True.
10. is would seem, superficially, to be like = =, but it tests whether the actual objects are the same, not whether the values which the objects reflect are equal.

A quick demonstration:



```
#%%
# Using `not`
name_list1 = ["John", "Jill"]
name_list2 = ["John", "Jill"]
print(not(name_list1 == name_list2))
# Using `is`
name2 = "John"
print(name_list1 == name_list2)
print(name_list1 is name_list2)
```

2.2.3 1.3.3 Loops

Loops iterate over a given sequence, and—here—it is critical to ensure yourindentation is correct or 'you will get unexpected results for what is consideredinside or outside the loop.

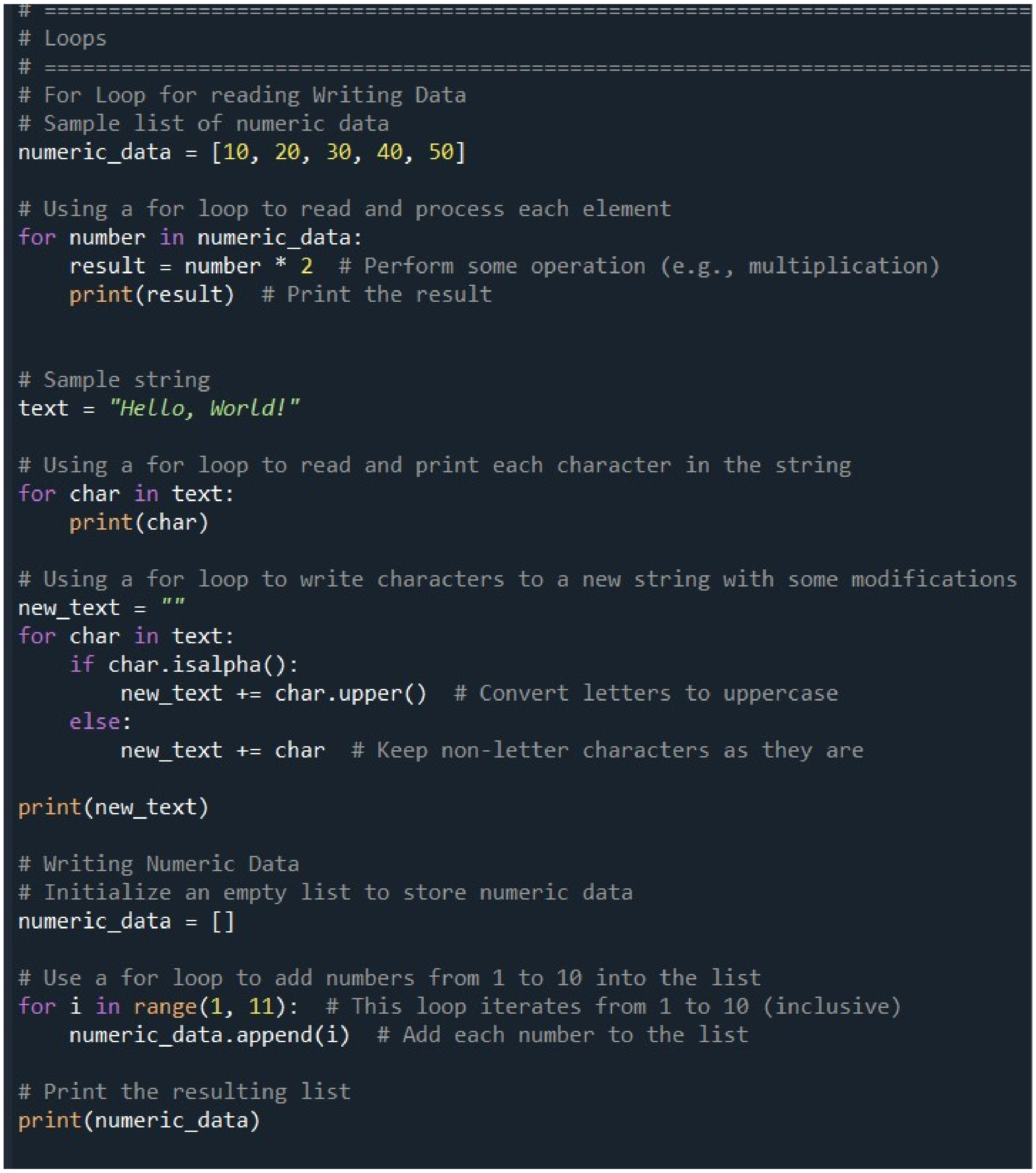
11. For loops, for, which loop through a list. There is also some new syntax to use in for loops:
    – In Lists you saw enumerate, which allows you to count the loop number.

    – Range creates a list of integers to loop, range(start, stop) creates alist of integers between start and stop, or range(num) creates a zero-basedlist up to num, or range(start, stop, step) steps through a list inincrements of step.

12. While loops, while, which execute while a particular condition is True. And some new syntax for while is:
    – while is a conditional statement (it requires a test to return True), whichmeans we can use else in a while loop (but not for)

In-Lab Task 8

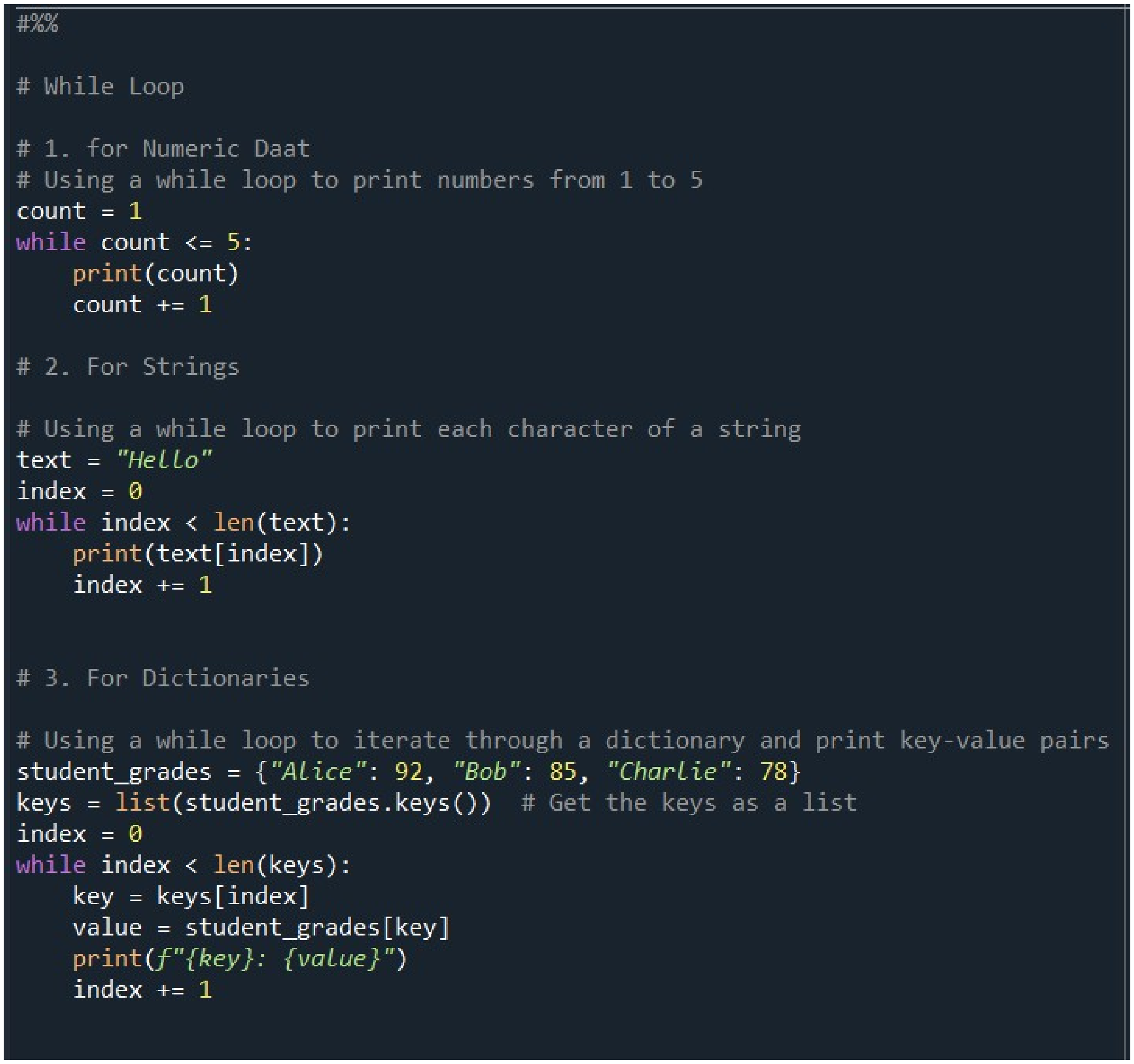
Demonstrate the reading and writing different types of data in python using For Loop.



```python
# ===============================================================================
# Loops
# ===============================================================================
# For Loop for reading Writing Data
# Sample list of numeric data
numeric_data = [10, 20, 30, 40, 50]

# Using a for loop to read and process each element
for number in numeric_data:
    result = number * 2  # Perform some operation (e.g., multiplication)
    print(result)  # Print the result


# Sample string
text = "Hello, World!"

# Using a for loop to read and print each character in the string
for char in text:
    print(char)

# Using a for loop to write characters to a new string with some modifications
new_text = ""
for char in text:
    if char.isalpha():
        new_text += char.upper()  # Convert letters to uppercase
    else:
        new_text += char  # Keep non-letter characters as they are

print(new_text)

# Writing Numeric Data
# Initialize an empty list to store numeric data
numeric_data = []

# Use a for loop to add numbers from 1 to 10 into the list
for i in range(1, 11):  # This loop iterates from 1 to 10 (inclusive)
    numeric_data.append(i)  # Add each number to the list

# Print the resulting list
print(numeric_data)
```

In-Lab Task 9

Demonstrate the application of While Loop.



```
#%%

# While Loop

# 1. for Numeric Daat
# Using a while loop to print numbers from 1 to 5
count = 1
while count <= 5:
    print(count)
    count += 1

# 2. For Strings

# Using a while loop to print each character of a string
text = "Hello"
index = 0
while index < len(text):
    print(text[index])
    index += 1


# 3. For Dictionaries

# Using a while loop to iterate through a dictionary and print key-value pairs
student_grades = {"Alice": 92, "Bob": 85, "Charlie": 78}
keys = list(student_grades.keys())  # Get the keys as a list
index = 0
while index < len(keys):
    key = keys[index]
    value = student_grades[key]
    print(f"{key}: {value}")
    index += 1
```

2.3 1.3 Post-Lab

Post-Lab Task

Your task is to create a Python program that manages student grades of at least seven (07) students and performs several operations. Your program should do the following:

1. Calculate and display the average grade for a list of students along with the list students and corresponding grades.
2. Categorize each student's grade into one of the following categories: "Excellent," "Very Good," "Good," or "Needs Improvement" based on the grade ranges (Your own assumptions).

3. Allow the user to search for a specific student's grade by entering the student's name. The program should repeatedly prompt the user for a name until a valid student name is entered, and then display the corresponding grade.

Provide the Python program that accomplishes these tasks, ensuring that it includes relevant data**, if-else conditions**, and both **for** and **while** loops.

Implement the program and demonstrate its functionality.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: ________________________________ Date: _____________________

Lab # 3: Learning Functions and Classes in Python

3.1 Objectives

13. Develop and use reusable code by encapsulating tasks in functions.
14. Package functions into flexible and extensible classes.

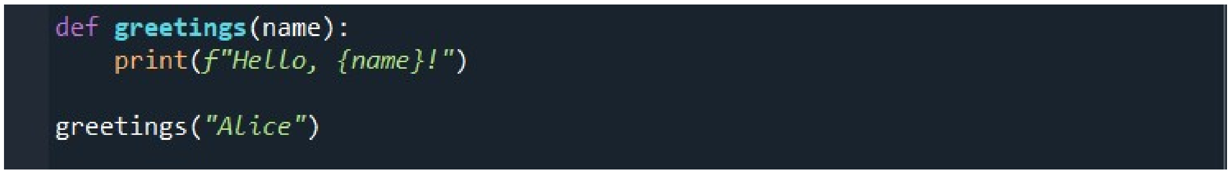
3.2 Pre-Lab

The code from the previous lab experiments was limited to shortsnippets. Solving more complex problems means more complex code - stretchingover hundreds, to thousands, of lines, and—if you want to reuse that code—it is notconvenient to copy and paste it multiple times. Worse, any error is magnified, andany changes become tedious to manage.

In Python, “functions” are complete suite of functions grouped around a set of related tasks is called a library or module. Libraries permit you to inherit a wide variety of powerful software solutions developed and maintained by other people.

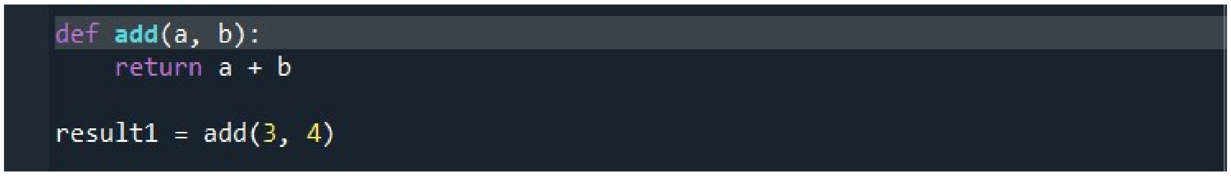
Functions in Python are defined using the def keyword, and they play a crucial role in organizing code, promoting reusability, and improving readability. Here are key aspects and concepts related to functions in Python:

15. Function Definition:
    Functions are defined using the “def” keyword, followed by the function name and a pair of parentheses. Any parameters the function takes are listed inside the parentheses.



```python
def greetings(name):
    print(f"Hello, {name}!")

greetings("Alice")
```

16. Function Call:
To execute a function, you "call" it by using its name followed by parentheses. If the function takes parameters, you provide the values inside the parentheses.



```python
def add(a, b):
    return a + b

result1 = add(3, 4)
```

17. Return Statement:
Functions can return a value using the **return** keyword. The returned value can be assigned to a variable or used directly.

```python
def multiply(x, y):
    return x * y

result2 = multiply(5, 6)
```

18. Parameters and Arguments:
Parameters are variables listed in a function's definition. Arguments are the values passed to the function when it is called.

19. Default Parameters:
You can provide default values for parameters. If a value is not passed for a parameter, the default value is used.

```python
def power(base, exponent=2):
    return base ** exponent

result4 = power(3)  # Uses default exponent of 2
```

20. Variable-Length Arguments:
Functions can accept a variable number of arguments using ***args** and ****kwargs**.

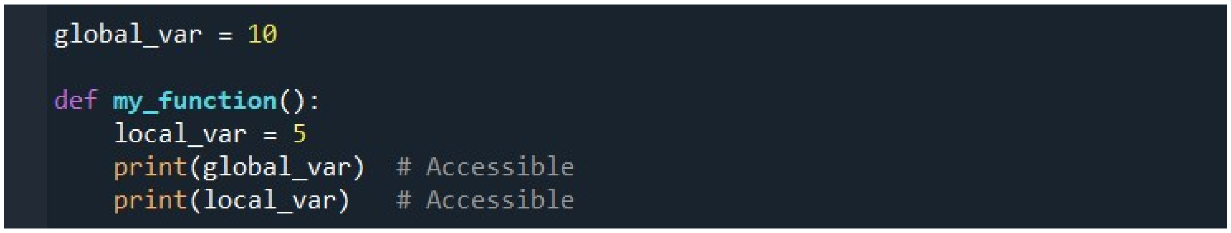
```python
def sum_all(*numbers):
    return sum(numbers)

result5 = sum_all(1, 2, 3, 4)
```

21. Lambda Functions:
Also known as anonymous functions, these are small, one-line functions defined using the **lambda** keyword.

```python
square = lambda x: x ** 2
result6 = square(5)
```

- **Scope:**

  Variables defined inside a function have local scope, meaning they are only accessible within that function. Variables defined outside functions have global scope.



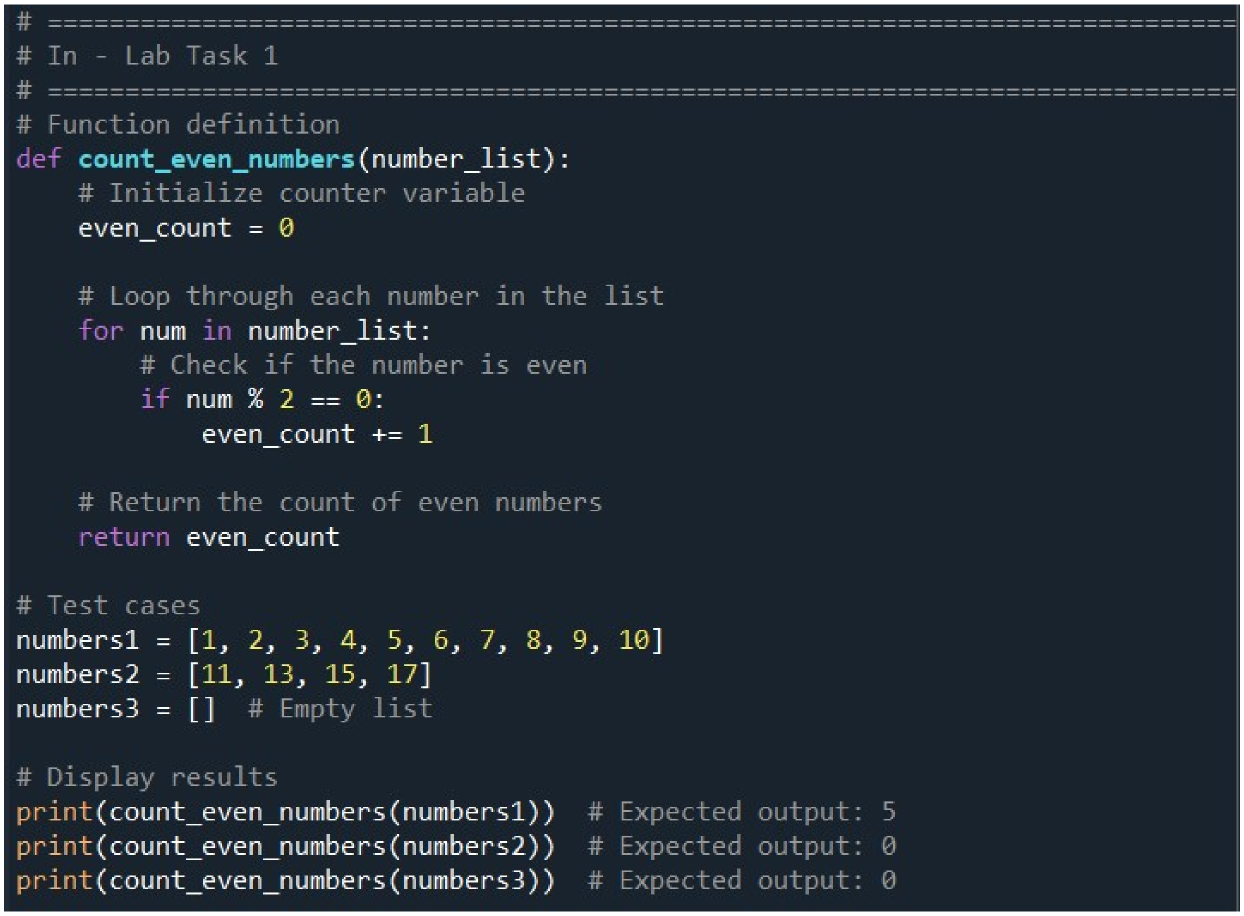
```python
global_var = 10

def my_function():
    local_var = 5
    print(global_var)  # Accessible
    print(local_var)   # Accessible
```

3.3 In-Lab

In-Lab Task 1

Write a Python function called "count_even_numbers" that takes a list of integers as input and returns the count of even numbers in the list. Additionally, ensure that the function handles the case where the input list is empty. Follow the following instructions:

22. Define a function named "count_even_numbers" that takes one parameter: number_list (a list of integers).
23. Inside the function, use a loop to iterate through each number in the list.
24. Use a conditional statement to check if each number is even.
25. If a number is even, increment a counter variable.
26. After the loop, return the counter variable.



```python
# ==============================================================================
# In - Lab Task 1
# ==============================================================================
# Function definition
def count_even_numbers(number_list):
    # Initialize counter variable
    even_count = 0

    # Loop through each number in the list
    for num in number_list:
        # Check if the number is even
        if num % 2 == 0:
            even_count += 1

    # Return the count of even numbers
    return even_count

# Test cases
numbers1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
numbers2 = [11, 13, 15, 17]
numbers3 = []  # Empty list

# Display results
print(count_even_numbers(numbers1))  # Expected output: 5
print(count_even_numbers(numbers2))  # Expected output: 0
print(count_even_numbers(numbers3))  # Expected output: 0
```

In-Lab Task 2

Write a Python function calculate_gpa that takes a list of dictionaries, where each dictionaryrepresents the marks of a student in 5 different courses (out of 100). Each dictionary should have keys 'name' and 'marks' (a list of 5 marks)

* Percentage Obtained in a Semester System	Grade	Grade Points
85 and above	A	4.00
80 84	A-	3.66
75 79	B+	3.33
71 - 74	B	3.00
68 70	B-	2.66
64 67	C+	2.33
61 63	C	2.00
58 60	C-	1.66
54 - 57	D+	1.30
50 53	D	1.00
Below 50	F	0.00

Calculate the Grade Point Average (GPA) for each student by averaging their Grade Points.

The function should return a list of dictionaries where each dictionary has keys 'name', 'grades' (a list of course grades), 'grade_points', and 'gpa'.

Create a dictionary which have at least record to five students (Name and Scores in 5 Courses)

3.3.1 Class

In Python, a class is a blueprint for creating objects. Objects are instances of a class, and each object can have attributes (characteristics) and methods (functions) associated with it. Classes provide a way to bundle data and functionality together. Here's a basic explanation of how classes work in Python:

27. Class Declaration: To create a class, you use the class keyword. Here's a simple example of a class:

```python
class Car:
    pass  # The 'pass' keyword is a placeholder for the class body
```

28. Attributes: You can define attributes (characteristics) inside the class. These attributes represent the data associated with objects of the class.

```python
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model
```

In this example, the **__init__** method is a special method called a constructor. It is called when an object is created. **self** refers to the instance of the class (the object), and you can set attributes like **make** and **model** for each object.

29. Methods: You can also define methods inside the class. Methods are functions associated with the class.

```python
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def display_info(self):
        print(f"{self.make} {self.model}")
```

The **display_info** method can be called on an instance of the **Car** class to display information about the car.

30. Creating Objects (Instances): Once a class is defined, you can create objects (instances) of that class.

```python
my_car = Car("Toyota", "Camry")
```

This creates an instance of the **Car** class called **my_car** with the specified make and model.

31. Accessing Attributes and Calling Methods: You can access attributes and call methods using the dot notation.

```python
my_car = Car("Toyota", "Camry")


print(my_car.make)  # Output: Toyota
my_car.display_info()  # Output: Toyota Camry
```

Here, **my_car.make** accesses the **make** attribute, and **my_car.display_info()** calls the **display_info** method.

In-Lab Task 3

Create a Python class named **student** to manage student information. The class should have the following attributes:

1. **name**: Name of the student.
2. **roll_number**: Roll number of the student.
3. **marks**: A list to store the marks of the student in different subjects.

The class should have the following methods:

1. **__init__**: A constructor to initialize the attributes.
2. **add_marks**: A method to add marks to the **marks** list.
3. **calculate_average**: A method to calculate and return the average marks of the student.

Create an instance of the **student** class, add some marks, and calculate the average.

3.4 Post-Lab

Post-Lab Task

Create a simple library management system using Python. Define a class **Book** to represent a book with attributes such as title, author, and availability status. Implement functions to borrow and return books.

Rubric for Lab Assessment