Department of
Computer Science and Engineering

Title: Introduction to Basic Operations on Python (Part-1)

Artificial Intelligence Lab
CSE 404

Green University of Bangladesh

1 Objective(s)

- To acquire knowledge about python.
- To learn variables in python.
- To learn python operators.
- To learn conditional statements in python.
- To learn loops in python.
- To learn functions in python.

2 Introduction

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans, or Eclipse which are particularly useful when managing larger collections of Python files.

In this lab manual, we will learn the basic operations of python like variable declaration, data types, operators, conditional statements, loops, and functions.

3 Getting started with Python Variables

Variables are containers for storing data values. Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.

```python
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.

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

If you want to specify the data type of a variable, this can be done with casting.

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

You can get the data type of a variable with the type() function.

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

String variables can be declared either by using single or double quotes:

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

Variable names are case-sensitive.

```
a = 4
A = "Sally"
#A will not overwrite a'
```

3.1 Python - Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, totalvolume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha−numeric characters and underscores (A-z, 0-9, and)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Legal variable names:

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Illegal variable names:

```
2myvar = "John"
my-var = "John"
my var = "John
```

Variable names with more than one word can be difficult to read. There are several techniques you can use to make them more readable:

3.1.1 Camel Case

Each word, except the first, starts with a capital letter:

```
myVariableName = "John"
```

3.1.2 Pascal Case

Each word starts with a capital letter:

```
MyVariableName = "John"
```

3.1.3 Snake Case

Each word is separated by an underscore character:

```
my_variable_name = "John"
```

3.2 Assign Multiple Values

Python allows you to assign values to multiple variables in one line:

```
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:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

If you have a collection of values in a list, tuple, etc. Python allows you to extract the values into variables. This is called unpacking.

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

3.3 Output Variables

The Python print() function is often used to output variables.

```
x = "Python is awesome"
print(x)
```

In the print() function, you output multiple variables, separated by a comma:

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

You can also use the + operator to output multiple variables:

```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

For numbers, the + character works as a mathematical operator:

```
x = 5
y = 10
print(x + y)
```

In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

```
x = 5
y = "John"
print(x + y)
```

The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

```
x = 5
y = "John"
print(x, y)
```

3.4 Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.Global variables can be used by everyone, both inside of functions and outside.

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

3.4.1 The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Also, use the global keyword if you want to change a global variable inside a function.

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

4 Operator

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

```
print(10 + 5) # 15
```

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

4.1 Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

4.2 Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
$^{=}$	x $^{=}$3	x = x ^{3}
»=	x »= 3	x = x » 3
«=	x «= 3	x = x « 3

4.3 Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Same As
==	Equal	x == y
!=	Not Equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

4.4 Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

4.5 Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

4.6 Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

4.7 Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
	XOR	Sets each bit to 1 if only one of two bits is 1
	NOT	Inverts all the bits
«	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
»	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

5 Conditional Statement

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b

- Less than or equal to: a <= b
- a > b
- 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 keyword.

5.1 If Statement

If statement:

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

In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose. If statement, without indentation (will raise an error):

```python
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

5.2 Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

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

In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

5.3 Else

The else keyword catches anything which isn't caught by the preceding conditions.

```python
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")
```

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

You can also have an else without the elif:

```python
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

5.4 Nested If

You can have if statements inside if statements, this is called nested if statements.

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

6 Loop

Most languages have the concept of loops: If we want to repeat a task twenty times, we dont want to have to type in the code twenty times, with maybe a slight change each time. As a result, we have for and while loops in python.

6.1 For Loop

Iterating over a sequence is done with a **for** loop (that is either a list, a tuple, a dictionary, a set, or a string).

6.1.1 Implementation in Python

```python
# range(5) is not the values of 0 to 5, but the values 0 to 4.
for i in range(5):
  print(i)

------------------------

# range(2, 5), which means values from 2 to 5 (but not including 5)
for i in range(2, 5):
  print(i)

------------------------

# Increment the sequence with 3 (default is 1):
for i in range(2, 20, 3):
  print(i)

------------------------

# Here, else keyword indicates a block of code to be executed when the loop is
    finished:
for i in range(5):
  print(i)
else:
  print("Finally finished!")

```

```
-----------------------

fruits = ["orange", "apple", "cherry"]
for i in fruits:
  print(i)

-----------------------

# Loop through the letters in the word "orange":
for i in "orange":
  print(i)

-----------------------

# Loop through the letters in the word "orange":

fruits = ["orange", "apple", "cherry"]
for i in fruits:
  print(i)
    if i == "apple":
        break
```

6.1.2 Lab Task (Please implement yourself and show the output to the instructor)

- Write a python program to find the sum of odd and even numbers from a set of numbers.
- Write a python program to find the smallest number from a set of numbers.
- Write a python program to find the sum of all numbers between 50 and 100, which are divisible by 3 and not divisible by 5.
- Write a python program to find the second highest number from a set of numbers.
- Write a python program to find the factorial of a number using for loop.
- Write a python program to generate Fibonacci series.

6.1.3 While Loop

6.1.4 Implementation in Python

```
i = 1
while i < 5:
  print(i)
  i += 1

-----------------------

i = 1
while i < 5:
  print(i)
  if i == 2:
    break
  i += 1

-----------------------

i = 0
while i < 5:
  i += 1
  if i == 2:
    continue
  print(i)

-----------------------

i = 1
while i < 5:
  print(i)
  i += 1
else:
  print("i is no longer less than 5")
```

6.1.5 Lab Task (Please implement yourself and show the output to the instructor)

- Write a python program to find the sum of odd and even numbers from a set of numbers.
- Write a python program to find the smallest number from a set of numbers.
- Write a python program to find the sum of all numbers between 50 and 100, which are divisible by 3 and not divisible by 5.
- Write a python program to find the second highest number from a set of numbers.
- Write a python program to find the factorial of a number using for loop.
- Write a python program to generate Fibonacci series.

7 Function

Functions enable you to break down the overall functionality of a script into smaller, logical subsections, which can then be called upon to perform their individual tasks when needed. Using functions to perform repetitive tasks is an excellent way to create code reuse. This is an important part of modern object-oriented programming principles.
In Python a function is defined using the **def** keyword:

7.1 Implementation in Python

```
def my_function():
  print("Hello World")

my_function()   # Calling my_function()
-----------------------

# Passing Argument
def my_function(name):
  print("name " + name)

my_function("Mike")
my_function("Monica")
my_function("John")

-----------------------

# Passing multiple Arguments
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

```python

------------------------

# Arbitrary Arguments, *args
def my_function(*names):
  print("Name= " + names[2])

my_function("Mike", "Monica", "John")

------------------------

# Send arguments with the key = value syntax.
def my_function(name3, name2, name1):
  print("Name= " + names3)

my_function(name1 = "Mike", name2 = "Monica", name3 = "John")

------------------------

def my_function(**name):
  print("His last name is " + name["lname"])

my_function(fname = "Mike", lname = "Monica")

------------------------

# Passing a List as an Argument
def my_function(food):
  for i in food:
    print(i)

fruits = ["orange", "apple", "cherry"]

my_function(fruits)

------------------------

# Return Values
def my_function(a):
  return 5 * a

print(my_function(3))
print(my_function(5))
print(my_function(9))

------------------------

# Recursion Example
def my_recursion(k):
  if(k > 0):
    result = k + my_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
my_recursion(6)
```

7.2 Lab Task (Please implement yourself and show the output to the instructor)

- Write a python program to find the largest number between two numbers using function
- Write a python program to find the sum of the numbers passed as parameters.

8 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.

Department of
Computer Science and Engineering

Title: Introduction to Basic Operations on Python (Part-2)

Artificial Intelligence Lab
CSE 316/404

Green University of Bangladesh

1 Objectives

- To learn Basic Operations on Python such as Lists, Tuple, Dictionary, Numpy, Pandas, Matplotlib.

2 Problem Analysis

2.1 List

Lists are used to store multiple items in a single variable. Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage. Lists are created using square brackets:
thislist = ["apple", "banana", "cherry"]
print(thislist)

2.2 List Items

List items are ordered, changeable, and allow duplicate values. When we say that lists are ordered, it means that the items have a defined order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list. There are some list methods that will change the order, but in general, the order of the items will not change. The list is changeable, meaning that we can change, add, and remove items in a list after it has been created. Since lists are indexed, lists can have items with the same value.

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

2.3 List Items - Data Types

String, int and boolean data types
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]

A list can contain different data types:
list1 = ["abc", 34, True, 40, "male"]

2.4 The list() Constructor

```
thislist = list(("apple", "banana", "cherry"))
print(thislist)
```

Output: ['apple', 'banana', 'cherry']

2.5 Negative Indexing

Negative indexing means start from the end

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Output: cherry

2.6 Range of Indexes

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Output: ['cherry', 'orange', 'kiwi']

2.7 Append Items

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

Output: ['apple', 'banana', 'cherry', 'orange']

2.8 Insert Items

To insert a list item at a specified index, use the insert() method.

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Output: ['apple', 'orange', 'banana', 'cherry']

2.9 Remove Specified Item

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

Output: ['apple', 'cherry']

2.10 pop() method removes the specified index

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

Output: ['apple', 'cherry']

2.11 Loop Through a List

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

Output: apple
banana
cherry

2.12 List Length

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

Output: 3

2.13 Loop Through the Index Numbers

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

Output: apple
banana
cherry

2.14 Sort List

```python
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

Output: [23, 50, 65, 82, 100]

2.15 Copy List

```python
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

Output: ['apple', 'banana', 'cherry']

2.16 Join Lists: ('+' operator, extend, append)

```python
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
```

Output: ['a', 'b', 'c', 1, 2, 3]

```python
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
list1.extend(list2)
print(list1)
```

Output: ['a', 'b', 'c', 1, 2, 3]

```python
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
for x in list2:
  list1.append(x)
print(list1)
```

Output: ['a', 'b', 'c', 1, 2, 3]

2.17 List Methods

Python has a set of built-in methods that you can use on lists.

```python
append()
clear()
copy()
count()
extend()
index()
insert()
pop()
remove()
reverse()
sort()
```

2.18 Examples of list problem

- Write a Python program to sum all the items in a list.
- Write a Python program to get the largest number from a list.

3 Implementation in python programming language

```
// sum all the items in a list in python


lst = []

# number of elements as input
n = int(input("Enter number of elements : "))

# iterating till the range
for i in range(0, n):
    ele = int(input())

    lst.append(ele)

print(lst)
print("Sum of elements in given list is :", sum(lst))
```

4 Sample Input/Output (Compilation, Debugging & Testing)

enter number of elements

5

2
5
3
4
1

Output
[2, 5, 3, 4, 1]
Sum of elements in given list is : 15

```
// The largest number from a list
def max_num_in_list( list ):
    max = list[ 0 ]
    for a in list:
        if a > max:
            max = a
    return max
print(max_num_in_list([1, 2, -8, 0]))
```

5 Sample Input/Output (Compilation, Debugging & Testing)

Output
2

6 Tuple

Tuples are used to store multiple items in a single variable. A tuple is a collection which is ordered and unchangeable. Tuples are written with round brackets. Tuple items are ordered, unchangeable, and allow duplicate values.

6.1 Create Tuple

```
thistuple = ("CSE", "EEE", "BBA")
print(thistuple)
```

Output: ('CSE', 'BBA', 'EEE')

6.2 Tuple Length

```
thistuple = ("apple", "banana", "cherry", 1, 2, 3)
print(len(thistuple))
```

Output: 6

6.3 Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
thistuple = ("apple",)
print(type(thistuple))
```

Output: <class 'tuple'>

```
thistuple = ("apple")
print(type(thistuple))
```

Output: <class 'str'>

6.4 The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-
    brackets
print(thistuple)
```

Output: ('apple', 'banana', 'cherry')

6.5 Access Tuple Items

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Output: banana

6.6 Range of Indexes

By leaving out the start value, the range will start at the first item.

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

Output: ('apple', 'banana', 'cherry', 'orange')

6.7 Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called. But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

Output: ('apple', 'kiwi', 'cherry')

6.8 Add Items

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.

- You can convert it into a list, add your item(s), and convert it back into a tuple.

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
print(thistuple)
```

  Output: ('apple', 'banana', 'cherry', 'orange')

- You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y
print(thistuple)
```

  Output: ('apple', 'banana', 'cherry', 'orange')

6.9 Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple. But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking".

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

(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```

Output: apple
banana
cherry

6.10 Using a While Loop

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

Output: apple
banana
cherry

6.11 Tuple Methods

Python has two built-in methods that you can use on tuples.

```
count()
index()
```

7 Dictionary

Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

7.1 Create Dictionary

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

Output: 'brand': 'Ford', 'model': 'Mustang', 'year': 1964

7.2 Access elements

You can access the items of a dictionary by referring to its key name, inside square brackets.

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

output: Mustang

7.3 Change Values

You can change the value of a specific item by referring to its key name: Change the "year" to 2018:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

output: 'brand': 'Ford', 'model': 'Mustang', 'year': 2018

7.4 Add item

dding an item to the dictionary is done by using a new index key and assigning a value to it:

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

Output: ’brand’: ’Ford’, ’model’: ’Mustang’, ’year’: 1964, ’color’: ’red’

7.5 Removing Items

There are several methods to remove items from a dictionary:

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

output: ’brand’: ’Ford’, ’year’: 1964

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

output: ’brand’: ’Ford’, ’model’: ’Mustang’

7.6 Loop Dictionaries

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

output: Ford Mustang 1964

7.7 Copy Dictionary

ou cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method copy().

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

```
mydict = thisdict.copy()
print(mydict)
```

7.8 Nested Dictionaries

A dictionary can contain dictionaries, this is called nested dictionaries. Create a dictionary that contain three dictionaries:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}

print(myfamily)
```

Output: 'child1': 'name': 'Emil', 'year': 2004, 'child2': 'name': 'Tobias', 'year': 2007, 'child3': 'name': 'Linus', 'year': 2011

8 NumPy

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python. If you have Python and PIP already installed on a system, then installation of NumPy is very easy.

Install it using this command:

C:Name>pip install numpy

8.1 NumPy Creating Arrays

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() function.

```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

output: [1 2 3 4 5]
<class 'numpy.ndarray'>

8.2 Access Array Elements

```python
import numpy as np

arr = np.array([1, 2, 3, 4])
```

```python

print(arr[0])
```

output: 1

8.3 NumPy Array Iterating

Iterating means going through elements one by one.

As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python.

If we iterate on a 1-D array it will go through each element one by one.

```python
import numpy as np

arr = np.array([1, 2, 3])

for x in arr:
  print(x)
```

Output: 1 2 3

8.4 NumPy Joining Array

Joining means putting contents of two or more arrays in a single array.

In SQL we join tables based on a key, whereas in NumPy we join arrays by axes.

We pass a sequence of arrays that we want to join to the concatenate() function, along with the axis. If axis is not explicitly passed, it is taken as 0.

```python
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

output: [1 2 3 4 5 6]

8.5 NumPy Sorting Arrays

Sorting means putting elements in an ordered sequence.

Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a specified array.

```python
import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

8.6 Generate Random Number

NumPy offers the random module to work with random numbers.

```python
from numpy import random

x = random.randint(100)

print(x)
```

9 Pandas

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

Pandas allow us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

9.1 Installation of Pandas

If you have Python and PIP already installed on a system, then installation of Pandas is very easy.

Install it using this command:

```
  C:\Users\Your Name>pip install pandas
```

If this command fails, then use a python distribution that already has Pandas installed like Anaconda, Spyder, etc.

```python
    import pandas as pd

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pd.DataFrame(mydataset)

print(myvar)
```

9.2 Pandas Series

A Pandas Series is like a column in a table. It is a one-dimensional array holding data of any type. Create a simple Pandas Series from a list:

```python
    import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

Output: [0 1] [1 7] [2 2]

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

This label can be used to access a specified value. With the index argument, you can name your own labels. Example:

```python
    import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

Output: [x 1] [y 7] [z 2]

9.3 Pandas DataFrames

A Pandas DataFrame is a 2-dimensional data structure, like a 2-dimensional array, or a table with rows and columns.

10 Matplotlib

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Matplotlib was created by John D. Hunter.

Matplotlib is open source and we can use it freely.

Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Install it using this command:

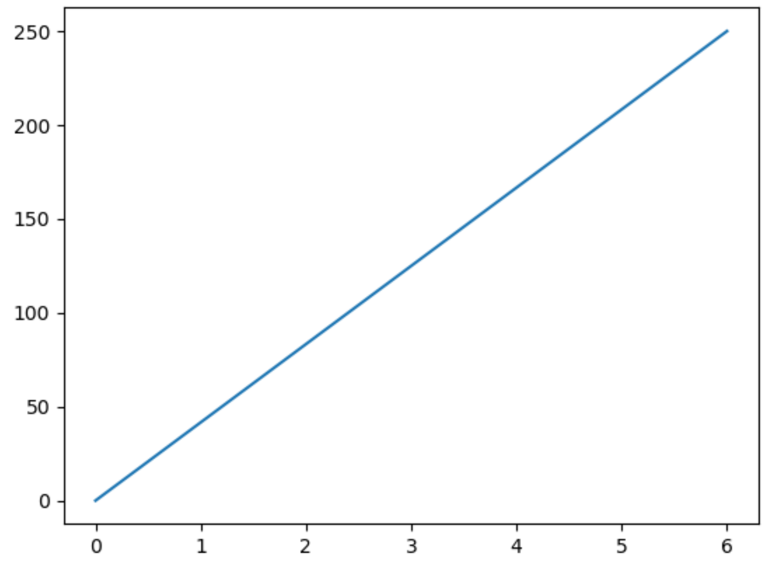
pip install matplotlib

10.1 Matplotlib Pyplot

Draw a line in a diagram from position (0,0) to position (6,250):

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```

output:



10.2 Matplotlib Plotting

The plot() function is used to draw points (markers) in a diagram.

By default, the plot() function draws a line from point to point.

The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the x-axis.

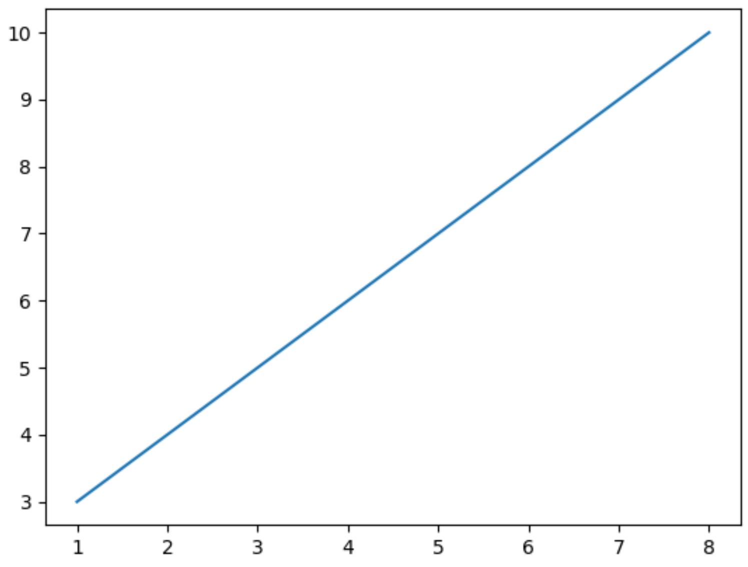
Parameter 2 is an array containing the points on the y-axis.

If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
```

```python
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

output:

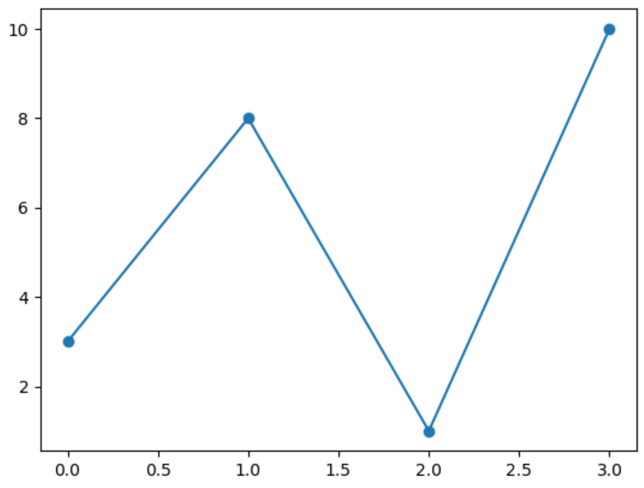


10.3 Matplotlib Markers

You can use the keyword argument marker to emphasize each point with a specified marker. Mark each point with a circle:

```python
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()
```

output:

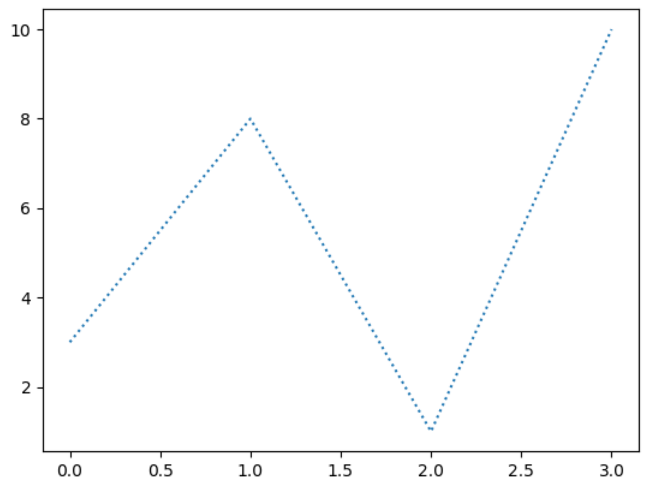


10.4 Matplotlib Line

You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:

```python
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```

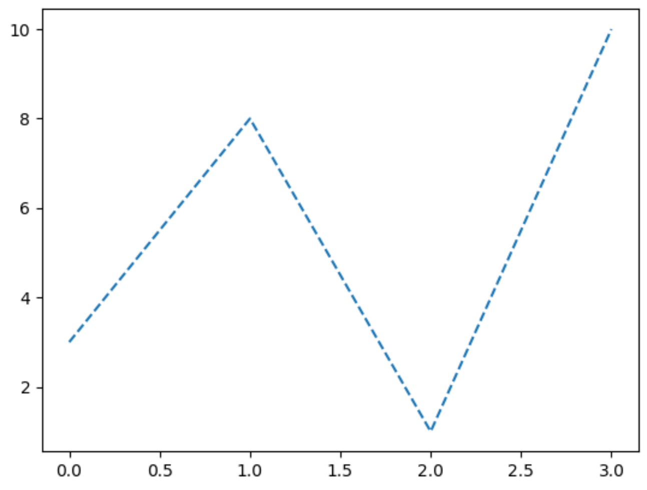
output:



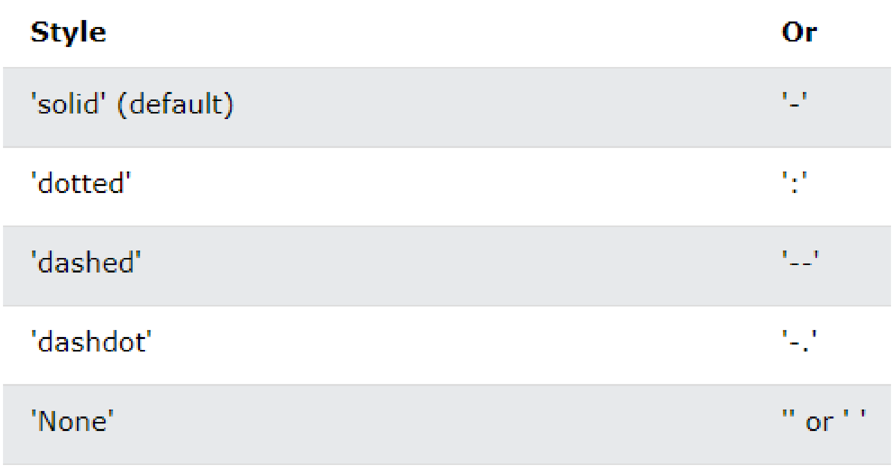
Use a dashed line:

```python
plt.plot(ypoints, linestyle = 'dashed')
```

output:



Other available Options:



Style	Or
'solid' (default)	'-'
'dotted'	':'
'dashed'	'--'
'dashdot'	'-.'
'None'	'' or ' '

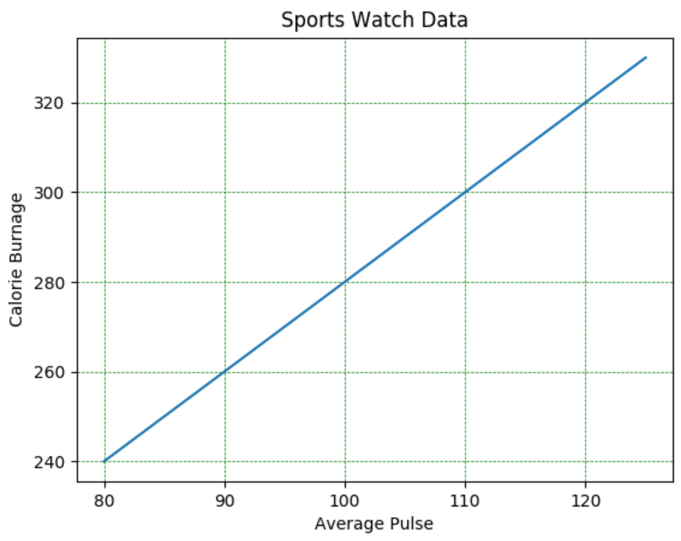
10.5 Matplotlib Adding Grid Lines

With Pyplot, you can use the grid() function to add grid lines to the plot.

You can also set the line properties of the grid, like this: grid(color = ’color’, linestyle = ’linestyle’, linewidth = number).

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(color = ’green’, linestyle = ’--’, linewidth = 0.5)

plt.show()
```

output:



10.6 Matplotlib Subplot

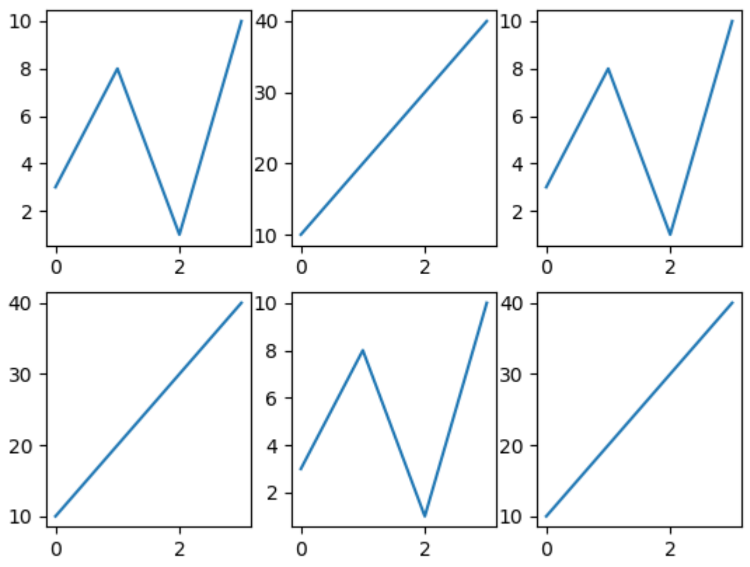
With the subplot() function you can draw multiple plots in one figure. The subplot() function takes three arguments that describes the layout of the figure.

The layout is organized in rows and columns, which are represented by the first and second argument.

The third argument represents the index of the current plot.

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```

output:



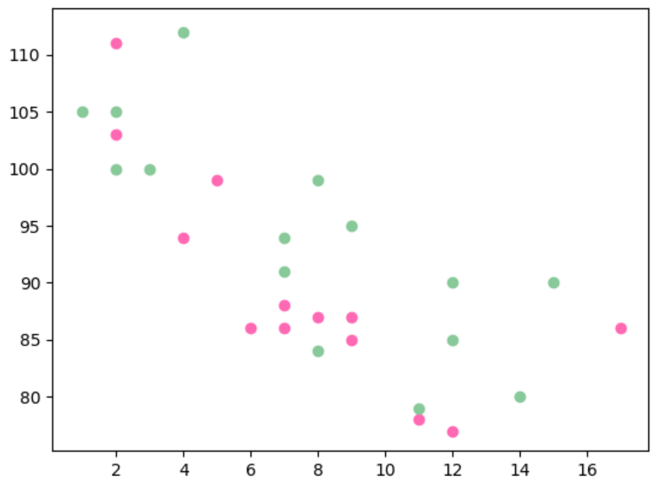
10.7 Matplotlib Scatter

With Pyplot, you can use the scatter() function to draw a scatter plot.

The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis.

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')

x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#88c999')

plt.show()
```

output:

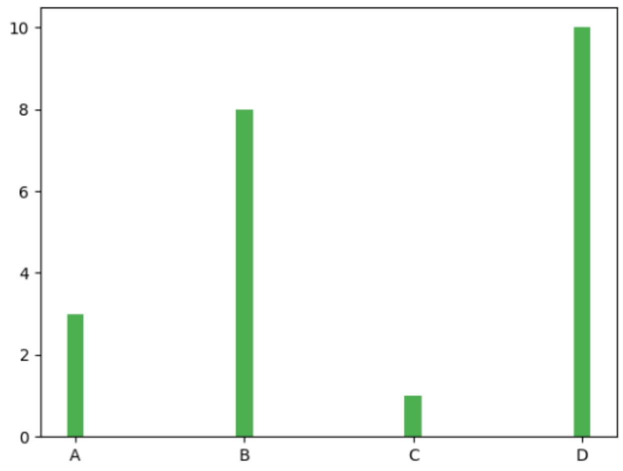


10.8 Matplotlib Bars

With Pyplot, you can use the bar() function to draw bar graphs:

```python
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "#4CAF50",width = 0.1)
plt.show()
```

output:



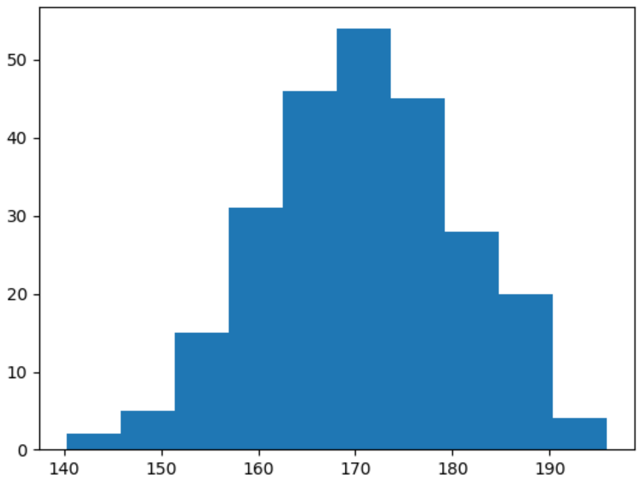
10.9 Matplotlib Histograms

In Matplotlib, we use the hist() function to create histograms.

The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```

output:

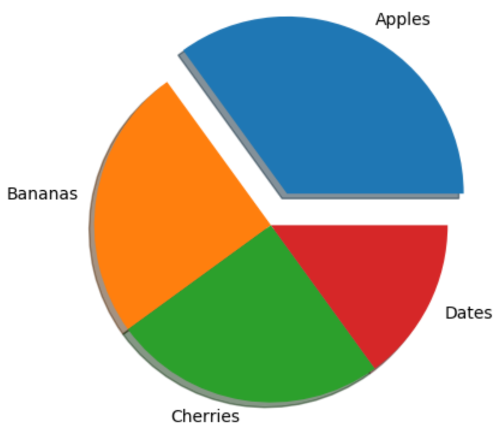


10.10 Matplotlib Pie Chart

With Pyplot, you can use the pie() function to draw pie charts.

```python
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
plt.show()
```

output:



11 Discussion & Conclusion

12 Lab Task (Please implement yourself and show the output to the instructor)

1. Write a Python program to reverse a tuple.

2. Write a python program to swap two tuples in Python.

```
Sample Input:
tuple1 = (11, 22)
tuple2 = (99, 88)
Sample Output:
tuple1: (99, 88)
tuple2: (11, 22)
```

13 Lab Exercise (Submit as a report)

- Write a Python program to get the 4th element from the beginning and the 4th element from the last of a tuple.

```
Sample Input:
tuplex = ("w", 3, "r", "e", "s", "o", "u", "r", "c", "e")
Sample Output:
e,u
```

- Write a Python Program to Count Even and Odd Numbers in a list.

14 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.