Model Question Papers Solution

MODULE 1

1 **Explain the math operators in Python from highest to lowest Precedence with an example for each. Write the steps how Python is evaluating the expression (5 - 1) * ((7 + 1) / (3 - 1)) and reduces it to a single value.** 6

- ✓ The operator is a symbol which tells the compiler to do specific mathematical or logical function.
- ✓ In python, a single value with no operators is also considered an expression, though it evaluates only to itself.
- ✓ The order of operations (also called precedence) of Python math operators is similar to that of mathematics i.e., **, *, /, //, %, +, -
- ✓ The associativity is also similar and followed from left to right.

Operator	Operation	Example	Evaluates to...
**	Exponent	2 ** 3	8
%	Modulus/remainder	22 % 8	6
//	Integer division/floored quotient	22 // 8	2
/	Division	22 / 8	2.75
*	Multiplication	3 * 5	15
-	Subtraction	5 - 2	3
+	Addition	2 + 2	4

- ✓ The steps for evaluating the expression in python is as follows:

```
(5 - 1) * ((7 + 1) / (3 - 1))
        ↓
      4 * ((7 + 1) / (3 - 1))
        ↓
      4 * (   8   ) / (3 - 1))
        ↓
      4 * (   8   ) / (  2   )
        ↓
      4 * 4.0
        ↓
      16.0
```

2 **Define a Python function with suitable parameters to generate prime numbers between two integer values. Write a Python program which accepts two integer values m and n (note: m>0, n>0 and m < n) as inputs and pass these values to the function. Suitable error messages should be displayed if the conditions for input values are not followed.** 8

```
def prime(m, n):
        for num in range(m, n + 1):
                if num > 1:
                        for i in range(2, num):
                                if (num % i) == 0:
                                        break
                        else:
                                print(num)

print("Enter the range of numbers to find prime numbers")
m = int(input())
n = int(input())
```

```
if m>0 and n>0 and m<n:
        prime(m, n)
else:
        print("Enter the correct values")
```

3 **Explain Local and Global Scope in Python programs. What are local and global variables? How can you force a variable in a function to refer to the global variable?** 6

- ✓ Parameters and variables that are assigned in a called function are said to exist in that function's *local scope.*
- ✓ Variables that are assigned outside all functions are said to exist in the *global scope*.
- ✓ A variable that exists in a local scope is called a *local variable*, while a variable that exists in the global scope is called a *global variable.*
- ✓ Scopes matter for several reasons:
  1. Code in the global scope cannot use any local variables.
  2. However, a local scope can access global variables.
  3. Code in a function's local scope cannot use variables in any other local scope.
  4. We can use the same name for different variables if they are in different scopes. That is, there can be a local variable named spam and a global variable also named spam.
- ✓ We can force a variable in a function to refer to the global variable using global statement as shown below:

Program

```
  def spam():
❶     global eggs
❷     eggs = 'spam'

  eggs = 'global'
  spam()
  print(eggs)
```

Output

```
spam
```

- ✓ Because eggs is declared global at the top of spam() ❶, when eggs is set to 'spam' ❷, this assignment is done to the globally scoped eggs. No local eggs variable is created.

4 **What are Comparison and Boolean operators? List all the Comparison and Boolean operators in Python and explain the use of these operators with suitable examples.** 6

- ✓ **Comparison operators:** These are used to compare two values and evaluate down to a single Boolean value.

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

- ✓ **Boolean Operators:** The three Boolean operators (and, or, and not) are used to compare Boolean values.

  i. *and operator:* The and operator evaluates an expression to True if both Boolean values are True; otherwise, it evaluates to False.

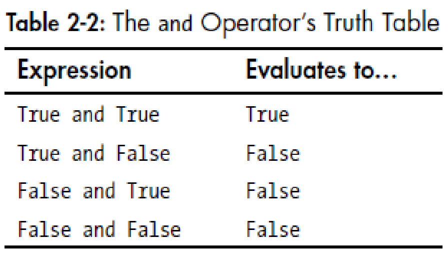
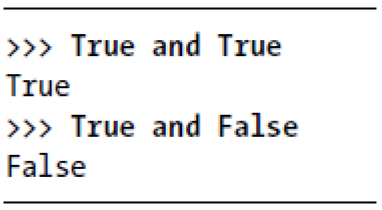


Table 2-2: The and Operator's Truth Table

Expression	Evaluates to...
True and True	True
True and False	False
False and True	False
False and False	False



```
>>> True and True
True
>>> True and False
False
```

ii. *<u>or operator:</u>* The or operator valuates an expression to True if either of the two Boolean values is True. If both are False, it evaluates to False.

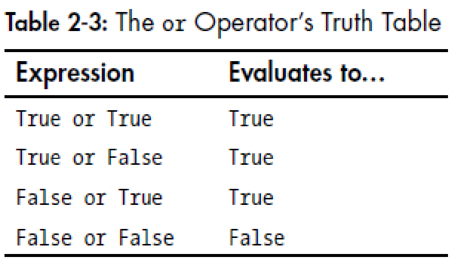
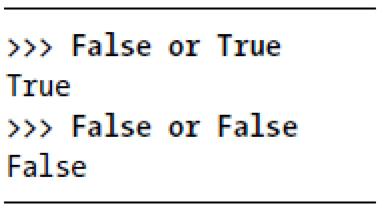


Table 2-3: The or Operator's Truth Table

Expression	Evaluates to...
True or True	True
True or False	True
False or True	True
False or False	False



```
>>> False or True
True
>>> False or False
False
```

iii. *<u>not operator:</u>* The not operator operates on only one Boolean value (or expression). The not operator simply evaluates to the opposite Boolean value. Much like using double negatives in speech and writing, you can nest not operators ❶, though there's never not no reason to do this in real programs.

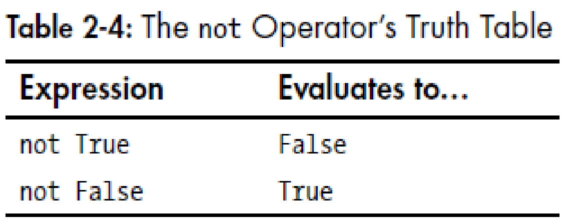
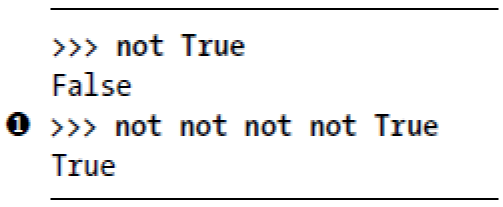


Table 2-4: The not Operator's Truth Table

Expression	Evaluates to...
not True	False
not False	True



```
  >>> not True
  False
❶ >>> not not not not True
  True
```

5 **Define a Python function with suitable parameters to generate first N Fibonacci numbers. The first two Fibonacci numbers are 0 and 1 and the Fibonacci sequence is defined as a function F as $F_n = F_{n-1} + F_{n-2}$. Write a Python program which accepts a value for N (where N >0) as input and pass this value to the function. Display suitable error message if the condition for input value is not followed.** 8

```
def fibonacci(n):
    if n < 0:
        print("incorrect input")
    elif n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

print("Enter the value of N")
n = int(input())
if n>0:
        x = fibonacci(n)
        print(x)
```

```
else:
        print("Enter the correct values")
```

6 **What is Exception Handling? How exceptions are handled in Python? Write a Python program with exception handling code to solve divide-by-zero error situation.** 6

- ✓ **Exception Handling**: If we don't want to crash the program due to errors instead we want the program to detect errors, handle them, and then continue to run is called exception handling.
- ✓ Exceptions can be handled with try and except statements.
- ✓ The code that could potentially have an error is put in a try clause. The program execution moves to the start of a following except clause if an error happens.
- ✓ **Program:**

```
def division(m, n):
        try:
                return m/n
        except ZeroDivisionError:
                print("Error: Divide-by-Zero Situation")

print("Enter two numbers")
x = int(input())
y = int(input())
print(division(x,y))
```

- ✓ In the above program if the second number given by the user as input is 0 then, it catches the error and the exception statement will be printed.

7 **Write a python program to find the area of square, rectangle and circle. Print the results. Take input from user.** 6

```
side = float(input('Enter side of square'))
sarea = side * side
print("The area of square is ",sarea)

length = float(input('Enter length of rectangle'))
breadth = float(input('Enter breadth of rectangle'))
rarea = length*breadth
print("The area of circle is ",rarea)

radius = float(input('Enter radius of circle'))
carea = 2*3.14*radius
print("The area of circle is ",carea)
```

8 **List and explain the syntax of all flow control statements with example.** 8

There are four types of flow control statements:

1. if Statement
2. else Statement
3. elif Statement

1. *if Statements:*

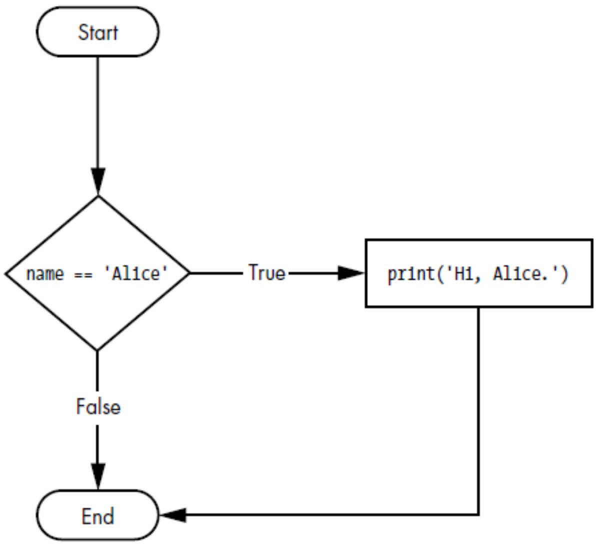
- An if statement's clause (that is, the block following the if statement) will execute if the statement's condition is True. The clause is skipped if the condition is False.
- In plain English, an if statement could be read as, "If this condition is true, execute the code in the clause."
- In Python, an if statement consists of the following:

1. The if keyword
2. A condition (that is, an expression that evaluates to True or False)
3. A colon
4. Starting on the next line, an indented block of code (called the if clause)

   - Example:

```
if name == 'Alice':
    print('Hi, Alice.')
```

   - Flowchart:



2. *else Statements:*

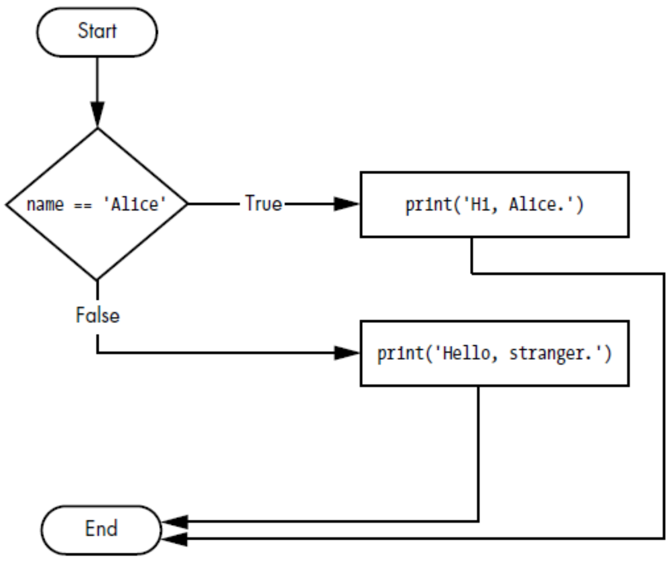
- An if clause can optionally be followed by an else statement. The else clause is executed only when the if statement's condition is False.
- In plain English, an else statement could be read as, "If this condition is true, execute this code. Or else, execute that code."
- An else statement doesn't have a condition, and in code, an else statement always consists of the following:

1. The else keyword
2. A colon
3. Starting on the next line, an indented block of code (called the else clause)

   - Example:

```
if name == 'Alice':
    print('Hi, Alice.')
else:
    print('Hello, stranger.')
```

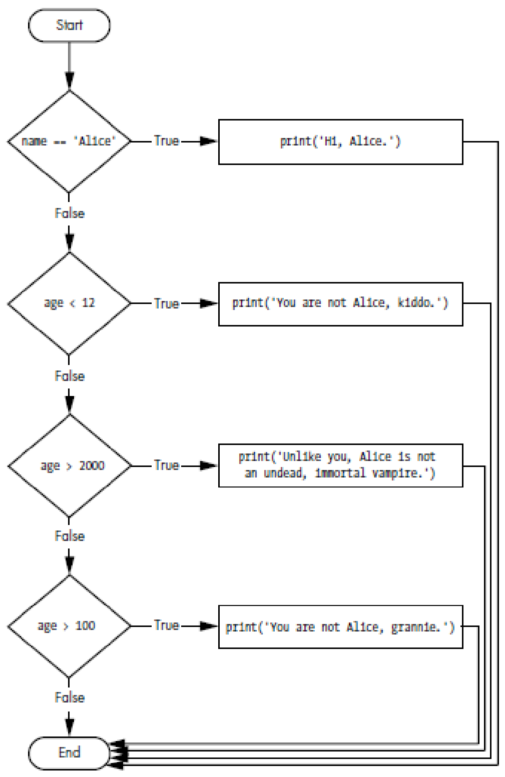
- Flowchart:



3. ***elif Statements:***

- If we have more than one condition then elif statement is used.
- The elif statement is an “else if” statement that always follows an if or another elif statement.
- It provides another condition that is checked only if all of the previous conditions were False.
- In code, an elif statement always consists of the following:
  1. The elif keyword
  2. A condition (that is, an expression that evaluates to True or False)
  3. A colon
  4. Starting on the next line, an indented block of code (called the elif clause)

- Flowchart:


Start
name == 'Alice'
True
print('Hi, Alice.')
False
age < 12
True
print('You are not Alice, kiddo.')
False
age > 2000
True
print('Unlike you, Alice is not an undead, immortal vampire.')
False
age > 100
True
print('You are not Alice, grannie.')
False
End

➢ Example:

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
elif age > 100:
    print('You are not Alice, grannie.')
```

9 **Illustrate the use of break and continue with a code snippet.** 6

<u>break Statement</u>

- ✓ The break statement is used to immediately exit from the loop.
- ✓ A break statement simply contains the break keyword.

```
❶ while True:
       print('Please type your name.')
❷      name = input()
❸      if name == 'your name':
❹          break
❺ print('Thank you!')
```

- ✓
- ✓ The first line ❶ creates an infinite loop; it is a while loop whose condition is always True.
- ✓ The program execution will always enter the loop and will exit it only when a break statement is executed.
- ✓ The program asks the user to type name ❷.
- ✓ While the execution is still inside the while loop, an if statement gets executed ❸ to check whether name is equal to entered name.
- ✓ If this condition is True, the break statement is run ❹, and the execution moves out of the loop to print('Thank you!') ❺.

<u>continue Statement</u>

- ✓ The continue statement is used to immediately jump back to the start of the loop and reevaluate the loop's condition.
- ✓ A continue statement simply contains continue keyword.

```
  while True:
      print('Who are you?')
      name = input()
❶     if name != 'Joe':
❷         continue
      print('Hello, Joe. What is the password? (It is a fish.)')
❸     password = input()
      if password == 'swordfish':
❹         break
❺ print('Access granted.')
```

- ✓ If the user enters any name besides Joe ❶, the continue statement ❷ causes the program execution to jump back to the start of the loop.
- ✓ When it reevaluates the condition, the execution will always enter the loop, since the condition is simply the value True. Once they make it past that if statement, the user is asked for a password ❸.
- ✓ If the password entered is swordfish, then the break statement ❹ is run, and the execution jumps out of the while loop to print Access granted ❺.

10 **What are user defined functions? How can we pass parameters in user defined functions? Explain with suitable example.** 7

- ✓ User defined functions are the mini-programs written by the user to do some specific task.
- ✓ The functions always start with the keyword “def”.
- ✓ We can pass parameters to the user defined functions as shown below:

```
❶ def hello(name):
❷     print('Hello ' + name)

❸ hello('Alice')
  hello('Bob')
```

```
Hello Alice
Hello Bob
```

- ✓ The definition of the hello() function in this program has a parameter called name ❶.
- ✓ A parameter is a variable that an argument is stored in when a function is called. The first time the hello() function is called, it’s with the argument 'Alice' ❸.
- ✓ The program execution enters the function, and the variable name is automatically set to 'Alice', which is what gets printed by the print() statement ❷.

11 **Explain global statement with example.** 8

- ✓ If we need to modify a global variable from within a function, use the global statement.
- ✓ If we have a line such as global eggs at the top of a function, it tells Python, “In this function, eggs refers to the global variable, so don’t create a local variable with this name.”
- ✓ For example:

<u>Program</u>

```
  def spam():
❶     global eggs
❷     eggs = 'spam'

  eggs = 'global'
  spam()
  print(eggs)
```

<u>Output</u>

```
spam
```

- ✓ Because eggs is declared global at the top of spam() ❶, when eggs is set to 'spam' ❷, this assignment is done to the globally scoped eggs. No local eggs variable is created.
- ✓ There are four rules to tell whether a variable is in a local scope or global scope:
  1. If a variable is being used in the global scope (that is, outside of all functions), then it is always a global variable.
  2. If there is a global statement for that variable in a function, it is a global variable.

3. Otherwise, if the variable is used in an assignment statement in the function, it is a local variable.
4. But if the variable is not used in an assignment statement, it is a global variable.

12 **Write a function that computes and returns addition, subtraction, multiplication, division of two integers. Take input from user.** 5

```
def calc(x, y):
        return x+y, x-y, x*y, x/y

print("Enter two numbers")
x = int(input())
y = int(input())
add, sub, mul, div = calc(x, y)
print("The sum is", add)
print("The difference is", sub)
print("The product is", mul)
print("The quotient is", div)
```

MODULE 2

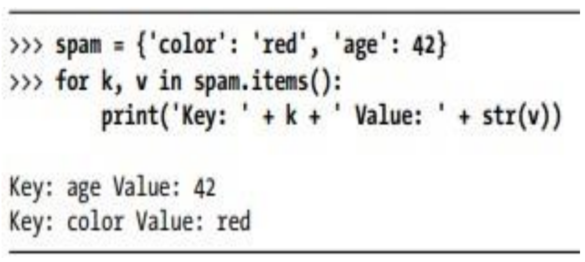
1 **What is Dictionary in Python? How is it different from List data type? Explain how a for loop can be used to traverse the keys of the Dictionary with an example.** **6**

Dictionary: A dictionary is a collection of many values. Indexes for dictionaries can use many different data types, not just integers. Indexes for dictionaries are called keys, and a key with its associated value is called a key-value pair. A dictionary is typed with braces, {}.

The dictionary is different from list data type:

- ✓ In lists the items are ordered and in dictionaries the items are unordered.
- ✓ The first item in a list exists. But there is no “first” item in a dictionary.
- ✓ The order of items matters for determining whether two lists are the same, it does not matter in what order the key-value pairs are typed in a dictionary.
- ✓ Trying to access a key that does not exist in a dictionary will result in a KeyError error message, in list “out-of-range” IndexError error message.

Traversing using for loop:



```
>>> spam = {'color': 'red', 'age': 42}
>>> for k, v in spam.items():
        print('Key: ' + k + ' Value: ' + str(v))

Key: age Value: 42
Key: color Value: red
```

2 **Explain the methods of List data type in Python for the following operations with suitable code snippets for each.** **8**

(i) Adding values to a list **ii) Removing values from a list**

(iii) Finding a value in a list **iv) Sorting the values in a list**

i) <u>Adding values to a list:</u>

- ✓ To add new values to a list, the append() and insert() methods can be used.
- ✓ The append() method adds the argument to the end of the list.

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.append('moose')
>>> spam
['cat', 'dog', 'bat', 'moose']
```

- ✓ The insert() method insert a value at any index in the list.
- ✓ The first argument to insert() is the index for the new value, and the second argument is the new value to be inserted.

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(1, 'chicken')
>>> spam
['cat', 'chicken', 'dog', 'bat']
```

ii) <u>Removing values from a list:</u>

- ✓ To remove values from a list, the remove( ) and del( ) methods can be used.
- ✓ The del statement is used when we know the index of the value we want to remove from the list.
- ✓ The remove() method is used when we know the value we want to remove from the list.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
>>> spam
['cat', 'rat', 'elephant']
```

- ✓ Attempting to delete a value that does not exist in the list will result in a ValueError error.
- ✓ If the value appears multiple times in the list, only the first instance of the value will be removed.

iii) <u>Finding a value in a list:</u>

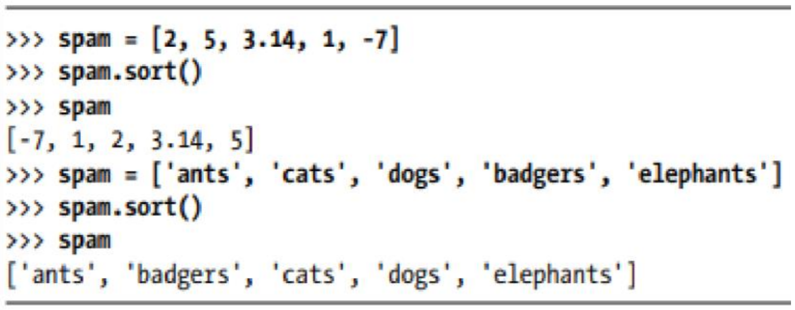
- ✓ To find a value in a list we can use index value.
- ✓ The first value in the list is at index 0, the second value is at index 1, and the third value is at index 2, and so on. The negative indexes can also be used.
- ✓ Example:

```
  >>> spam = ['cat', 'bat', 'rat', 'elephant']
  >>> spam[0]
  'cat'
  >>> spam[1]
  'bat'
  >>> spam[2]
  'rat'
  >>> spam[3]
  'elephant'
  >>> ['cat', 'bat', 'rat', 'elephant'][3]
  'elephant'
❶ >>> 'Hello ' + spam[0]
❷ 'Hello cat'
  >>> 'The ' + spam[1] + ' ate the ' + spam[0] + '.'
  'The bat ate the cat.'
```

- ✓ The expression 'Hello ' + spam[0] evaluates to 'Hello ' + 'cat' because spam[0] evaluates to the string 'cat'. This expression in turn evaluates to the string value 'Hello cat'.
- ✓ Negative index → spam[-1] → Retrieves last value.

iv) <u>Sorting the values in a list</u>

- ✓ Lists of number values or lists of strings can be sorted with the sort() method.



```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

3 **Write a Python program that accepts a sentence and find the number of words, digits, uppercase letters and lowercase letters.** **6**

```
def Count(str):
    up, low, num, special = 0, 0, 0, 0
    for i in range(len(str)):
        if str[i] >= 'A' and str[i] <= 'Z':
            up += 1
        elif str[i] >= 'a' and str[i] <= 'z':
            low += 1
        elif str[i] >= '0' and str[i] <= '9':
            num += 1
        else:
            special += 1
    print("LETTERS :",up+low,"DIGITS :",num,"UPPERCASE :",up,"LOWERCASE :",low)
print("Enter a sentence")
str = input()
Count(str)
```

4 **What is the difference between copy.copy( ) and copy.deepcopy( ) functions applicable to a List or Dictionary in Python? Give suitable examples for each.** **6**

Copy module must be imported and can be used to make a duplicate copy of a mutable value like a list or dictionary, not just a copy of a reference.

copy.copy( ): A shallow copy creates a new object which stores the reference of the original elements.So, a shallow copy doesn't create a copy of nested objects, instead it just copies the reference of nested objects. This means, a copy process does not create copies of nested objects itself.

<u>Example:</u>

```
import copy
old_list = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]
new_list = copy.copy(old_list)
```

```
old_list[1][1] = 'AA'
print("Old list:", old_list)
print("New list:", new_list)
```

Output:

```
Old list: [[1, 1, 1], [2, 'AA', 2], [3, 3, 3]]
New list: [[1, 1, 1], [2, 'AA', 2], [3, 3, 3]]
```

copy.deepcopy( ): A deep copy creates a new object and recursively adds the copies of nested objects present in the original elements.

Example:

```
import copy
old_list = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]
new_list = copy.deepcopy(old_list)
old_list[1][0] = 'BB'
print("Old list:", old_list)
print("New list:", new_list)
```

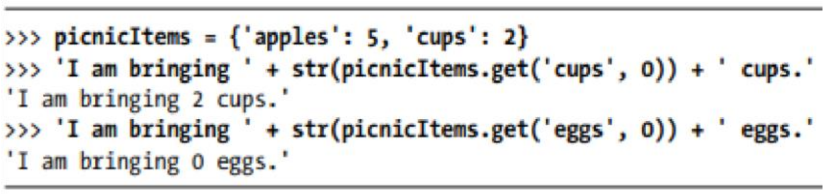
Output:

```
Old list: [[1, 1, 1], ['BB', 2, 2], [3, 3, 3]]
New list: [[1, 1, 1], [2, 2, 2], [3, 3, 3]]
```

5 **Discuss the following Dictionary methods in Python with examples.** 8
(i) get() (ii) items() (iii) keys() (iv) values()

(i) get( ): Dictionaries have a get() method that takes two arguments:

- ➢ The key of the value to retrieve
- ➢ A fallback value to return if that key does not exist.



```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.'
'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.'
'I am bringing 0 eggs.'
```

(ii) items( ): This method returns the dictionary values and keys in the form of tuples.

Ex:
```
spam = {‘color’ : ‘red’ , ‘age’ : 27}
for i in spam.items( ):
        print(i)
```
Output:
```
('color', 'red')
('age', 27)
```

(iii) keys( ): This method returns the dictionary keys.

Ex:
```
spam = {‘color’ : ‘red’ , ‘age’ : 27}
for i in spam.keys( ):
        print(i)
```

Output: color
age

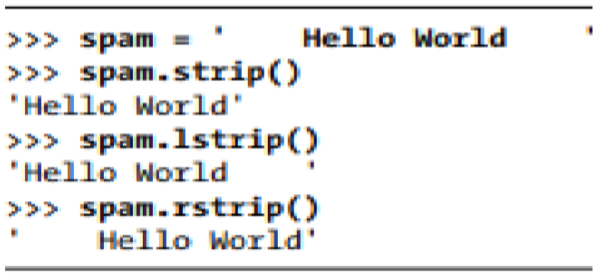
iv) values( ): This method returns the dictionary values.

Ex:
```
spam = {'color' : 'red' , 'age' : 27}
for i in spam.values( ):
        print(i)
```
Output:
```
red
27
```

6 **Explain the various string methods for the following operations with examples.** 6
(i) **Removing whitespace characters from the beginning, end or both sides of a string.**
(ii) **To right-justify, left-justify, and center a string.**

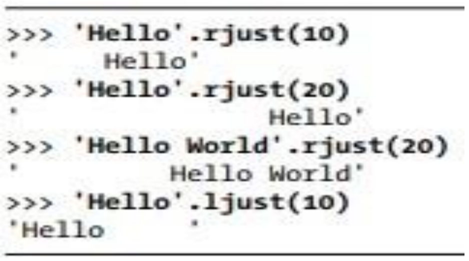
i) Removing whitespace characters from the beginning, end or both sides of a string.

- ✓ The strip() string method will return a new string without any whitespace characters at the beginning or end.
- ✓ The lstrip() and rstrip() methods will remove whitespace characters from the left and right ends, respectively.

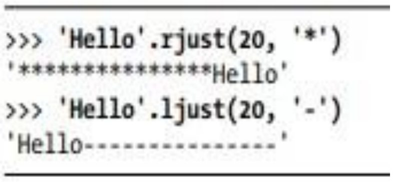

```
>>> spam = '    Hello World    '
>>> spam.strip()
'Hello World'
>>> spam.lstrip()
'Hello World    '
>>> spam.rstrip()
'    Hello World'
```

ii) To right-justify, left-justify, and center a string.

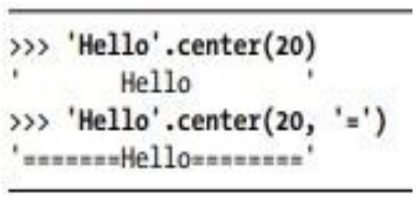
- ✓ The rjust() and ljust() string methods return a padded version of the string they are called on, with spaces inserted to justify the text.
- ✓ The **first** argument to both methods is an integer length for the justified string.


```
>>> 'Hello'.rjust(10)
'     Hello'
>>> 'Hello'.rjust(20)
'               Hello'
>>> 'Hello World'.rjust(20)
'         Hello World'
>>> 'Hello'.ljust(10)
'Hello     '
```

- ✓ An optional **second** argument to rjust() and ljust() will specify a fill character other than a space character.


```
>>> 'Hello'.rjust(20, '*')
'***************Hello'
>>> 'Hello'.ljust(20, '-')
'Hello---------------'
```

- ✓ The center() string method works like ljust() and rjust() but centers the text rather than justifying it to the left or right.



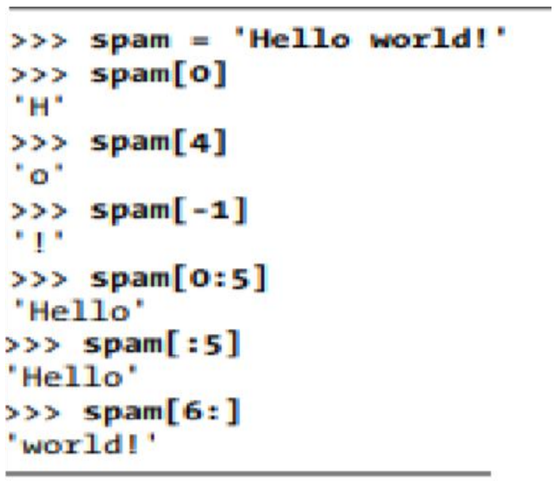
```
>>> 'Hello'.center(20)
'       Hello        '
>>> 'Hello'.center(20, '=')
'=======Hello========'
```

7 **What is list? Explain the concept of list slicing with example.** **6**

List: A list is a value that contains multiple values in an ordered sequence.

Slicing: Extracting a substring from a string is called substring.

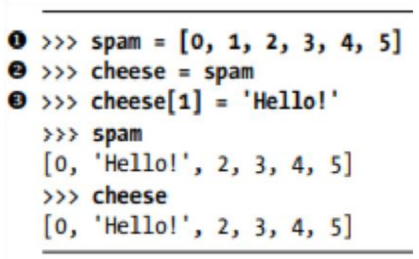
Example:



```
>>> spam = 'Hello world!'
>>> spam[0]
'H'
>>> spam[4]
'o'
>>> spam[-1]
'!'
>>> spam[0:5]
'Hello'
>>> spam[:5]
'Hello'
>>> spam[6:]
'world!'
```

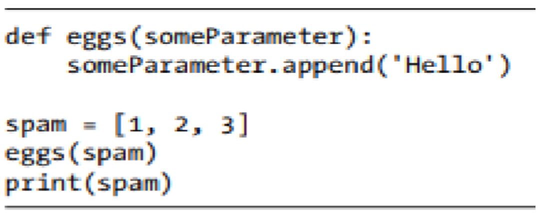
8 **Explain references with example.** **7**

- ✓ **Reference:** A reference is a value that points to some bit of data, and a list reference is a value that points to a list.

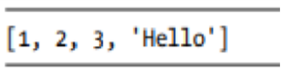


```
❶ >>> spam = [0, 1, 2, 3, 4, 5]
❷ >>> cheese = spam
❸ >>> cheese[1] = 'Hello!'
  >>> spam
  [0, 'Hello!', 2, 3, 4, 5]
  >>> cheese
  [0, 'Hello!', 2, 3, 4, 5]
```

- ✓ Here, the list ❶ is created, and assigned reference to it in the spam variable.
- ✓ In the next line copies only the list reference in spam to cheese, not the list value itself. This means the values stored in spam and cheese now both refer to the same list.
- ✓ When a function is called, the values of the arguments are copied to the parameter variables.



```
def eggs(someParameter):
    someParameter.append('Hello')

spam = [1, 2, 3]
eggs(spam)
print(spam)
```



```
[1, 2, 3, 'Hello']
```

	✓ when eggs() is called, a return value is not used to assign a new value to spam. ✓ Even though spam and someParameter contain separate references, they both refer to the same list. ✓ This is why the append('Hello') method call inside the function affects the list even after the function call has returned.	

9	**What is dictionary? How it is different from list? Write a program to count the number of occurrences of character in a string.**
	
	`import pprint`
	`x = input("Enter a String")`
	`count = {}`
	
	`for character in x:`
	`    count.setdefault(character, 0)`
	`    count[character] = count[character] + 1`
	
	`pprint.pprint(count)`
		7

10	**You are creating a fantasy video game. The data structure to model the player's inventory will be a dictionary where the keys are string values describing the item in the inventory and the value is an integer value detailing how many of that item the player has. For example, the dictionary value {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12} means the player has 1 rope, 6 torches, 42 gold coins, and so on. Write a function named displayInventory() that would take any possible "inventory" and display it like the following:**
	Inventory:
	12 arrow
	42 gold coin
	1 rope
	6 torch
	1 dagger
	Total number of items: 63
	
	`stuff = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}`
	`def displayInventory(inventory):`
	`    print("Inventory:")`
	`    item_total = 0`
	`    for k, v in inventory.items():`
	`        item_total = item_total + v`
	`        print(str(stuff.get(k, 0)) + ' ' + k)`
	`    print("Total number of items: " + str(item_total))`
	`displayInventory(stuff)`
		7

11	**List any six methods associated with string and explain each of them with example.** **i)** **upper( ):** This method is used to convert lower case characters into upper case characters. **Ex:** x = 'Python' **x =** x.upper( ) **PYTHON**	8

ii) **lower( ):** This method is used to convert upper case characters into lower case characters.

Ex: x = ‘Python’
x = x.lower( )
python

iii) **isupper( ):** This method is used to check whether a string has at least one letter or complete string is upper or not. It returns Boolean value.

Ex: x = ‘Python’
x = x.isupper( )
TRUE

Ex: y = ‘python’
y = y.isupper( )
FALSE

iv) **islower( ):** This method is used to check whether a string has at least one letter or complete string is lower or not. It returns Boolean value.

Ex: x = ‘Python’
x = x.islower( )
TRUE

Ex: y = ‘PYTHON’
y = y.isupper( )
FALSE

v) **isspace( ):** Returns True if the string consists only of spaces, tabs, and newlines and is not blank.

Ex: ‘   ‘.isspace( )

TRUE

vi) **isalnum( ):** Returns True if the string consists only of letters and numbers and is not blank.

Ex: ‘hello123’.isalnum( )

TRUE

Ex: **‘ ‘.**isalnum( )

FALSE

12 **Write a Python program to swap cases of a given string.** **5**
Input: Java
Output: jAVA

Solution 1: Using inbuilt function

```
print("Enter a String")
string = input()
print(string.swapcase())
```

Solution 2: Without using inbuilt function

```
def swapcase(string):
  result_str = ""
  for item in string:
     if item.isupper():
            result_str += item.lower()
        else:
            result_str += item.upper()
    return result_str
string = input("Enter a String")
print(swapcase(string))
```

MODULE 3

1 **Write a Python Program to find an American phone number (example: 415-555-4242) in a given string using Regular Expressions.** **6**

```
import re
print("Enter a string")
x = input()
phoneRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
phoneRegex.findall(x)
```

2 **Describe the difference between Python os and os.path modules. Also, discuss the following methods of os module**
a) chdir() b) rmdir() c) walk() d) listdir() e) getcwd() **7**

os module →It is from the standard library and it provides a portable way of accessing and using operating system dependent functionality.

os.path module → It is used to manipulate the file and directory paths and path names.

a) chdir( ) → This method is used to change from current working directory to the required directory.

Ex:
```
import os
os.chdir("C:\\Windows\\System")
```

b) rmdir( ) → This method is used to delete the folder at path. This folder must not contain any files or folders.

c) walk( ) → This method is used to access or do any operation on each file in a folder.

Ex:
```
import os
for foldername, subfolder, filenames in os.walk("C:\\Windows\\System")
        print("The current folder is "+foldername)
```

d) listdir( ) → This method returns a list containing the names of the entries in the directory given by path.

e) getcwd( ) → This method is used to get the current working directory.

Ex:
```
import os
os.getcwd( )
```

3 **Demonstrate the copy, move, rename and delete functions of shutil module with Python code snippet.** **7**

<u>Copy:</u> shutil.copy(source, destination) will copy the file from source path to destination path.

```
   >>> import shutil, os
   >>> os.chdir('C:\\')
❶ >>> shutil.copy('C:\\spam.txt', 'C:\\delicious')
   'C:\\delicious\\spam.txt'
❷ >>> shutil.copy('eggs.txt', 'C:\\delicious\\eggs2.txt')
   'C:\\delicious\\eggs2.txt'
```

In the above example:
1 → Destination is a folder, hence the source file is copied to destination file.
2 → Destination is a folder with a filename, hence the source file copied will be renamed to the given filename.

<u>Move:</u> shutil.move(source, destination) will move the file or folder from source path to the destination path and will return a string of the absolute path of the new location.

```
>>> import shutil
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs\\bacon.txt'
```

<u>Rename:</u> shutil.move(source, destination) can be used to move and rename also.

```
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs\\new_bacon.txt')
'C:\\eggs\\new_bacon.txt'
```

<u>Delete:</u> We can delete a single file or a single empty folder with functions in the os module, whereas to delete a folder and all of its contents, we use the shutil module.

1. Calling os.unlink(path) will delete the file at path.
2. Calling os.rmdir(path) will delete the folder at path. This folder must be empty of any files or folders.
3. Calling shutil.rmtree(path) will remove the folder at path, and all files and folders it contains will also be deleted.

<u>Ex:</u>
```
import os
for filename in in os.listdir( ):
        if filename.endswith('.txt'):
                os.unlink(filename)
```

4 **Describe the following with suitable Python code snippet.**
(i) Greedy and Non Greedy Pattern Matching (ii) findall() method of Regex object. **7**

(i) Greedy and Non Greedy Pattern Matching:

✓ Greedy pattern matching means matching with the longest possible string. By default, in python greedy matching is followed.
Ex:

```
>>> greedyHaRegex = re.compile(r'(Ha){3,5}')
>>> mo1 = greedyHaRegex.search('HaHaHaHaHa')
>>> mo1.group()
'HaHaHaHaHa'
```

- ✓ Non-greedy pattern matching means matching with the shortest possible string. It should be represented explicitly using a question mark after the curly brackets.
- ✓ Ex:

```
>>> nongreedyHaRegex = re.compile(r'(Ha){3,5}?')
>>> mo2 = nongreedyHaRegex.search('HaHaHaHaHa')
>>> mo2.group()
'HaHaHa'
```

(ii) findall() method of Regex object.

- ✓ search() method will return a Match object of the first matched text in the searched string.
- ✓ findall() method will return the strings of every match in the searched string in the form of list of strings—as long as there are no groups in the regular expression.
- ✓ Ex:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') # has no groups
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
['415-555-9999', '212-555-0000']
```

5 **Explain the file Reading/Writing process with suitable Python Program.** 6

<u>Reading File:</u> read( ) and readlines( ) methods are used to read the contents of a file. read( ) method reads the contents of a file as a single string value whereas readlines( ) method reads each line as a string value.
<u>Ex:</u>

```
>>> helloContent = helloFile.read()
>>> helloContent
'Hello world!'
```

<u>Writing File:</u> The file must be opened in order to write the contents to a file. It can be done in two ways: wither in write mode or append mode. In write mode the contents of the existing file will be overwritten whereas in append mode the contents will be added at the end of existing file.

<u>Ex:</u> In the below example bacon.txt file is opened in write mode hence, all the contents will be deleted and Hello world! Will be written.

```
>>> baconFile = open('bacon.txt', 'w')
>>> baconFile.write('Hello world!\n')
13
```

<u>Ex:</u> In the below example bacon.txt file is opened in append mode hence, the contents will be added after Hello world!

```
>>> baconFile.close()
>>> baconFile = open('bacon.txt', 'a')
>>> baconFile.write('Bacon is not a vegetable.')
25
```

- ✓ The files must be closed after once it is read or written.

6 **Define assertions. What does an assert statement in python consists of? Explain how assertions can be used in traffic light simulation with Python code snippet.** 7

	✓ **Assertion:** An assertion is a sanity check to make sure the code isn't doing something obviously wrong.
	✓ If the sanity check fails, then an AssertionError exception is raised.
	✓ In python, an assert statement consists of the following:
	1. The assert keyword
	2. A condition (that is, an expression that evaluates to True or False)
	3. A comma
	4. A string to display when the condition is False
	✓ In plain English, an assert statement meaning is, "I assert that this condition holds true, and if not, there is a bug somewhere in the program."
	
	✓ **Traffic Light Simulation:** The data structure representing the stoplights at an intersection is a dictionary with keys 'ns' and 'ew', for the stoplights facing north-south and east-west, respectively. The values at these keys will be one of the strings 'green', ' yellow', or 'red'.
	`market_2nd = {'ns': 'green', 'ew': 'red'}`
	`mission_16th = {'ns': 'red', 'ew': 'green'}`
	➢ These two variables will be for the intersections of Market Street and 2nd Street, and Mission Street and 16th Street. The switchLights() function, will take an intersection dictionary as an argument and switch the lights.
	(code below)
	➢ But, while writing switchLights() we add an assertion to check that at least one of the lights is always red.
	`assert 'red' in stoplight.values(), 'Neither light is red! ' + str(stoplight)`
	✓ If neither direction of traffic has a red light then assertion error occurs.	

```
def switchLights(stoplight):
    for key in stoplight.keys():
        if stoplight[key] == 'green':
            stoplight[key] = 'yellow'
        elif stoplight[key] == 'yellow':
            stoplight[key] = 'red'
        elif stoplight[key] == 'red':
            stoplight[key] = 'green'

switchLights(market_2nd)
```

7 **List and explain Shorthand code for common character classes .Illustrate how do you define your own character class?** 7

Character classes are used for shortening regular expression. The following are few of the shorthand code for common character classes:

Shorthand character class	Represents
\d	Any numeric digit from 0 to 9.
\D	Any character that is *not* a numeric digit from 0 to 9.
\w	Any letter, numeric digit, or the underscore character. (Think of this as matching "word" characters.)
\W	Any character that is *not* a letter, numeric digit, or the underscore character.
\s	Any space, tab, or newline character. (Think of this as matching "space" characters.)
\S	Any character that is *not* a space, tab, or newline.

<u>Defining own character class:</u> The own character class can be defined using square brackets [ ] and the necessary conditions can be given. Negative character class can also be defined using the symbol ^.

<u>Ex:</u>

[0-5] → Will match the numbers from 0 to 5
[0-5.] → Will match the number from 0 to 5 and a period
[a-zA-Z] → Will match all the upper case and lower case letters.
[^a-zA-Z] → Will match all the characters apart from upper case and lower case letters.

8 **Explain the usage of Caret and dollar sign characters in regular expression.** **6**

- ✓ The caret symbol (^) at the start of a regex is used to indicate that a match must occur at the beginning of the searched text.
- ✓ The dollar sign ($) at the end of the regex is used to indicate that a match must occur at the end of the searched text.
- ✓ Both the symbols can be used ^ and $ together to indicate that the entire string must match the regex.
- ✓ **<u>Ex:</u>** It checks whether the string starts with Hello.

```
x = re.compile(r'^Hello')
x.search('Hello world!')
```

- ✓ **<u>Ex:</u>** It checks whether the string ends with Hello.

```
x = re.compile(r'Hello$')
x.search('Hello world!')
```

- ✓ **<u>Ex:</u>** It checks whether the string starts and ends with Hello.

```
x = re.compile(r'^Hello$')
x.search('Hello Hello')
```

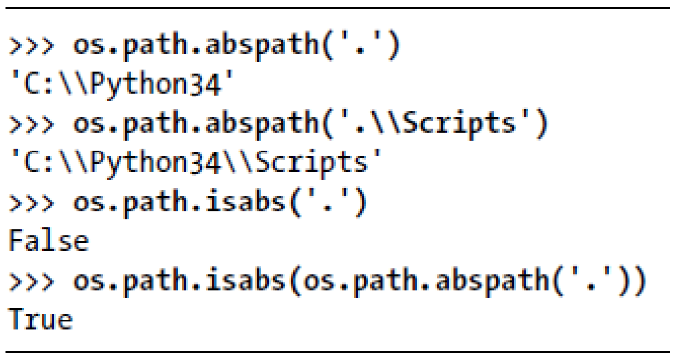
9 **Write a python program to extract phone numbers and email addresses using regex.** **7**

```
import pyperclip, re

phoneRegex = re.compile(r'''(
   (\d{3}|\(\d{3}\))? # area code
   (\s|-|\.)?        # separator
   (\d{3})            # first 3 digits
   (\s|-|\.)         # separator
   (\d{4})            # last 4 digits
   (\s*(ext|x|ext.)\s*(\d{2,5}))?  # extension
   )''', re.VERBOSE)

# Create email regex.
emailRegex = re.compile(r'''(
   [a-zA-Z0-9._%+-]+      # username
   @                      # @ symbol
   [a-zA-Z0-9.-]+         # domain name
```

```
    (\.[a-zA-Z]{2,4}){1,2} # dot-something
    )''', re.VERBOSE)

# Find matches in clipboard text.
text = str(pyperclip.paste())

matches = []
for groups in phoneRegex.findall(text):
    phoneNum = '-'.join([groups[1], groups[3], groups[5]])
    if groups[8] != '':
        phoneNum += ' x' + groups[8]
    matches.append(phoneNum)
for groups in emailRegex.findall(text):
    matches.append(groups[0])

# Copy results to the clipboard.
if len(matches) > 0:
    pyperclip.copy('\n'.join(matches))
    print('Copied to clipboard:')
    print('\n'.join(matches))
else:
    print('No phone numbers or email addresses found.')
```

10 **How do we specify and handle absolute, relative paths?** 10

- ✓ There are two ways to specify a file path.
  1. An absolute path, which always begins with the root folder
  2. A relative path, which is relative to the program's current working directory
- ✓ The os.path module provides functions for returning the absolute path of a relative path and for checking whether a given path is an absolute path.
  1. Calling os.path.abspath(path) will return a string of the absolute path of the argument. This is an easy way to convert a relative path into an absolute one.
  2. Calling os.path.isabs(path) will return True if the argument is an absolute path and False if it is a relative path.
  3. Calling os.path.relpath(path, start) will return a string of a relative path from the start path to path. If start is not provided, the current working directory is used as the start path.



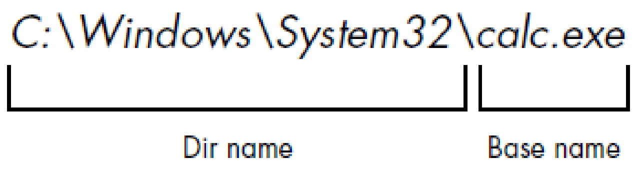
```
>>> os.path.abspath('.')
'C:\\Python34'
>>> os.path.abspath('.\\Scripts')
'C:\\Python34\\Scripts'
>>> os.path.isabs('.')
False
>>> os.path.isabs(os.path.abspath('.'))
True
```

○

- ✓ Since C:\Python34 was the working directory when os.path.abspath() was called, the "single-dot" folder represents the absolute path 'C:\\Python34'.

```
>>> os.path.relpath('C:\\Windows', 'C:\\')
'Windows'
>>> os.path.relpath('C:\\Windows', 'C:\\spam\\eggs')
'..\\..\\Windows'
>>> os.getcwd()
'C:\\Python34'
```

- ✓ Calling os.path.dirname(path) will return a string of everything that comes before the last slash in the path argument.
- ✓ Calling os.path.basename(path) will return a string of everything that comes after the last slash in the path argument.



- ➢ Example:

```
>>> path = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.basename(path)
'calc.exe'
>>> os.path.dirname(path)
'C:\\Windows\\System32'
```

11 **Explain saving of variables using shelve module.** **4**

- ✓ The variables can be saved in Python programs to binary shelf files using the shelve module.
- ✓ This helps the program to restore data to variables from the hard drive.
- ✓ The shelve module will let us add Save and Open features to your program.
- ✓ Example:

```
>>> import shelve
>>> shelfFile = shelve.open('mydata')
>>> cats = ['Zophie', 'Pooka', 'Simon']
>>> shelfFile['cats'] = cats
>>> shelfFile.close()
```

12 **With code snippet, explain reading, extracting and creating ZIP files.** **6**

<u>Reading ZIP Files:</u>

- ✓ To read the contents of a ZIP file, ZipFile object must be created.
- ✓ ZipFile objects are values through which the program interacts with the file.
- ✓ To create a ZipFile object, the zipfile.ZipFile() function is called, passing it a string of the .zip file's filename.

```
>>> import zipfile, os
>>> os.chdir('C:\\')    # move to the folder with example.zip
>>> exampleZip = zipfile.ZipFile('example.zip')
>>> exampleZip.namelist()
['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
```

Creating ZIP Files:

- ✓ To create our own compressed ZIP files, must open the ZipFile object in write mode by passing 'w' as the second argument.
- ✓ The write() method's first argument is a string of the filename to add.
- ✓ The second argument is the compression type parameter, which tells the computer what algorithm it should use to compress the files; we can always just set this value to zipfile.ZIP_DEFLATED

```
>>> import zipfile
>>> newZip = zipfile.ZipFile('new.zip', 'w')
>>> newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
>>> newZip.close()
```

Extracting ZIP Files:

- ✓ The extractall() method for ZipFile objects extracts all the files and folders from a ZIP file into the current working directory.
- ✓ After running the below code, the contents of example.zip will be extracted to C:\.

```
  >>> import zipfile, os
  >>> os.chdir('C:\\')    # move to the folder with example.zip
  >>> exampleZip = zipfile.ZipFile('example.zip')
❶ >>> exampleZip.extractall()
  >>> exampleZip.close()
```

- ✓ The extract() method for ZipFile objects will extract a single file from the ZIP file.

```
>>> exampleZip.extract('spam.txt')
'C:\\spam.txt'
>>> exampleZip.extract('spam.txt', 'C:\\some\\new\\folders')
'C:\\some\\new\\folders\\spam.txt'
>>> exampleZip.close()
```

MODULE 4

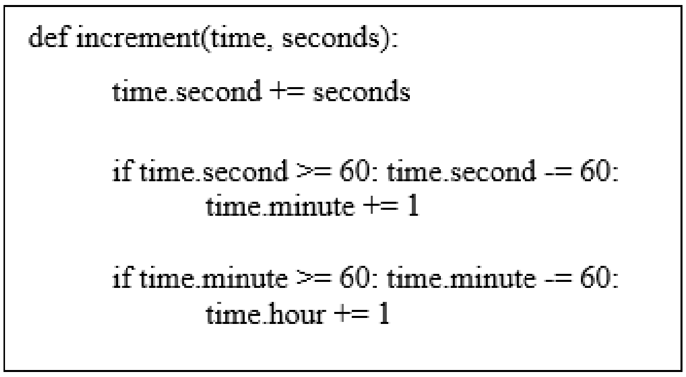
1 **Define classes and objects in Python. Create a class called Employee and initialize it with employee id and name. Design methods to:**
i) setAge_to assign age to employee.
ii) setSalary_to assign salary to the employee.
iii) Display_to display all information of the employee. 8

```
class student( ):

    def _init_(self, name, roll):
        self.name = name
        self.roll = roll

    def display(self):
        print(self.name)
        print(self.roll)
    def setAge(self, age):
        self.age = age

    def setMarks(self, marks):
        self.marks = marks
```

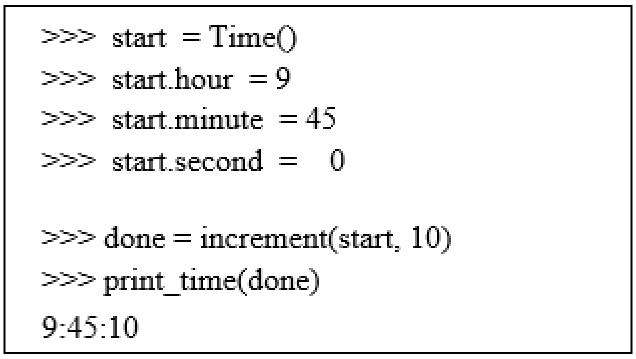
2 **Illustrate the concept of modifier with Python code.** **5**

__Modifiers:__ It modifies the objects passed to it as arguments and it has effect.

__Ex:__ In the below example the object start has been passed as parameters and it has been changed by adding seconds to it. Hence, it is a modifier.



```
def increment(time, seconds):
        time.second += seconds

        if time.second >= 60: time.second -= 60:
                time.minute += 1

        if time.minute >= 60: time.minute -= 60:
                time.hour += 1
```

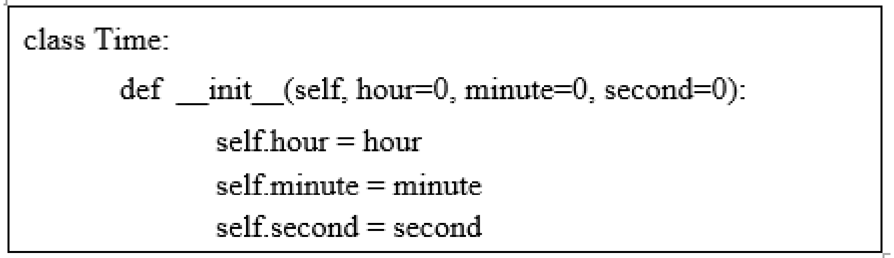


```
>>>  start  = Time()
>>>  start.hour  = 9
>>>  start.minute  = 45
>>>  start.second  =   0

>>> done = increment(start, 10)
>>> print_time(done)
9:45:10
```

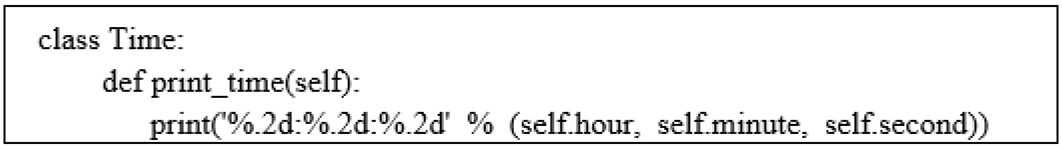
3 **Explain init and __str__ method with an example Python Program.** **7**

__init__: The init method (short for “initialization”) is a special method that gets invoked when an object is instantiated. The parameters of __init__ to have the same names as the attributes.

__Ex:__



```
class Time:
        def __init__(self, hour=0, minute=0, second=0):
                self.hour = hour
                self.minute = minute
                self.second = second
```



```
class Time:
    def print_time(self):
        print('%.2d:%.2d:%.2d'  %  (self.hour,  self.minute,  self.second))
```

If function is called with no arguments then, default values will be printed.	If function is called with one arguments then, it overrides hour.	If function is called with two arguments then, it overrides hour, minutes.

>>> time = Time()
>>> time.print_time()
00:00:00
	>>> time = Time (9)
	>>> time.print_time()
	09:00:00
		>>> time = Time(9, 45)
		>>> time.print_time()
		09:45:00

If function is called with three arguments then, all the values will be overridden.

__str__: It is a special method which returns a string representation of an object.

Ex:

```
class Time:
        def __str__(self):
                return '%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second)
```

When we print an object, Python invokes the str method:

```
>>> time = Time(9, 45)
>>> print(time)
09:45:00
```

4 **Define polymorphism? Demonstrate polymorphism with function to find histogram to count the number of times each letter appears in a word and in a sentence.** 7

Polymorphism: Functions that work with several types are called polymorphic. Polymorphism can facilitate code reuse.

Ex:

```
def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] = d[c]+1
    return d
```

This function also works for lists, tuples, and even dictionaries, as long as the elements of s are hashable, so they can be used as keys in d:

```
>>> t = ['spam', 'egg', 'spam', 'spam', 'bacon', 'spam']
>>> histogram(t)
{'bacon': 1, 'egg': 1, 'spam': 4}
```

Program: To count the frequency

```
def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] = d[c]+1
    return d
data = input("Enter the string: ")
dic = histogram(data)
print("The frequency of letters are: ")
for i in dic:
    print(i, dic[i])
```

Output

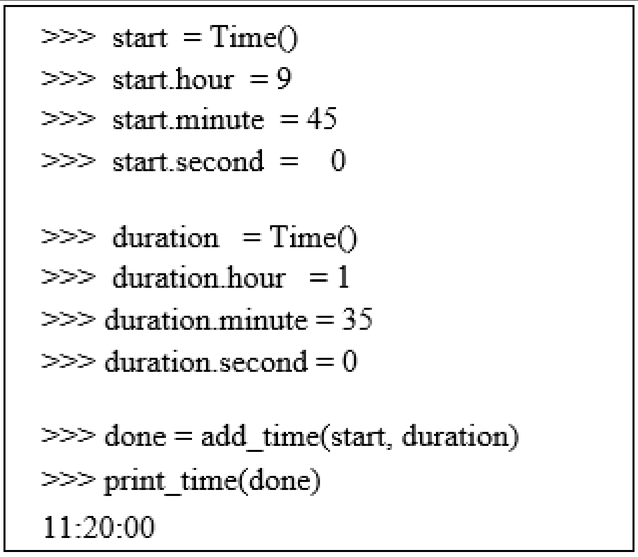
```
Enter the string: python python
The frequency of letters are:
p 2
y 2
t 2
h 2
o 2
n 2
  1
```

5 **Illustrate the concept of pure function with Python code.** 6

<u>Pure Function:</u> It does not modify any of the objects passed to it as arguments and it has no effect, like displaying a value or getting user input, other than returning a value.

<u>Ex:</u> In the below example the two objects start and duration have been passed as parameters to function add_time and it doesn't modify the objects received. Hence, it is a pure function.

```
def add_time(t1, t2):
sum = Time()
    sum.hour = t1.hour + t2.hour
    sum.minute = t1.minute + t2.minute
    sum.second = t1.second + t2.second

if sum.second >= 60: sum.second -= 60
    sum.minute += 1

if sum.minute >= 60: sum.minute -= 60
    sum.hour += 1

    return sum
```



```
>>> start = Time()
>>> start.hour = 9
>>> start.minute = 45
>>> start.second = 0

>>> duration = Time()
>>> duration.hour = 1
>>> duration.minute = 35
>>> duration.second = 0

>>> done = add_time(start, duration)
>>> print_time(done)
11:20:00
```

6 **Define Class Diagram. Discuss the need for representing class relationships using Class Diagram with suitable example.** 7

Stack diagram represents the state of a program, object diagram represents the attributes and their values and both are very detailed. Whereas, Class diagram represents the abstract representation of structure of program. There are several kinds of relationships between classes:

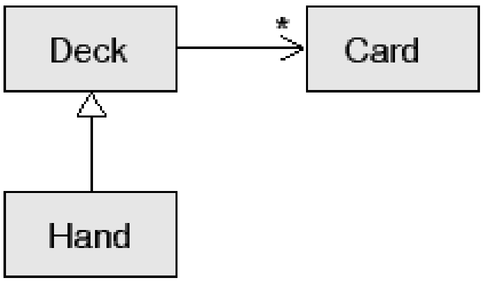
1. Objects in one class might contain references to objects in another class. For example, each Rectangle contains a reference to a Point, and each Deck contains references to many Cards. This kind of relationship is called **HAS-A**, as in, "a Rectangle has a Point."

2. One class might inherit from another. This relationship is called **IS-A**, as in, "a Hand is a kind of a Deck."

3. One class might depend on another in the sense that changes in one class would require changes in the other.

Class diagram: It is a graphical representation of the relationships.

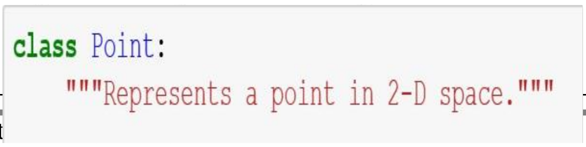
<u>Ex:</u>



- ➢ IS-A relationship → Arrow with a hallow triangle head → Hand inherits from Deck
- ➢ HAS-A relationship → Standard Arrow → Deck has reference to card objects.
- ➢ Star(*) → Multiplicity → Indicates how many cards a deck has.

7 **What is class? How do we define a class in python? How to instantiate the class and how class members are accessed?** 8

- ✓ <u>Class</u> is a abstract data type which can be defined as a template or blueprint that describes the behavior / state that the object of its type support.
- ✓ We define class as follows:



```
class Point:
    """Represents a point in 2-D space."""
```

	✓ The header indicates that the new class is called Point.
	✓ The body is a docstring that explains what the class is for. You can define variables and methods inside a class *definition*.
	✓ The process of creating this object is called instantiation
	✓ Class can be used to create new object instances (instantiation) of that class.
	✓ The class members are accessed using dot operator as shown below:
	`class Point:`
	`        " " " Represents a point in 2-D space " " "`
	`blank = Point( )`
	`blank.x = 10`
	`blank.y =20`	

8	**Write a python program that uses datetime module within a class , takes a birthday as input and prints user's age and the number of days, hours ,minutes and seconds until their next birthday.** (code below)	**7**
9	**Explain __init__ and __str__ methods.** Refer Module 4 Question 3 → Above	**5**
10	**Explain operator overloading with example.**	**7**

Code for question 8:

```
import datetime

def get_user_birthday():
    year = int(input('When is your birthday? [YY] '))
    month = int(input('When is your birthday? [MM] '))
    day = int(input('When is your birthday? [DD] '))
    birthday = datetime.datetime(year,month,day)
    return birthday

def calculate_dates(original_date, now):
    date1 = now
    date2 = datetime.datetime(now.year, original_date.month, original_date.day)
    delta = date2 - date1
    days = delta.total_seconds() / 60 /60 /24
    return days

def show_info(self):
    pass

bd = get_user_birthday()
now = datetime.datetime.now()
c = calculate_dates(bd,now)
print(c)
```

Operator Overloading: Ability of an existing to work on user defined data type. It is a polymorphic nature of any object oriented programming. Basic operators like +, -, * can b overloaded. Here, the behavior of an operator is changed like + so it works with a user defined type.

Ex: + → _ _ add _ _ - → _ _sub_ _ * → _ _ mul_ _

Program:

```
class A:
    def __init__(self, a):
        self.a = a

    def __add__(self, o):
        return self.a + o.a

    def __sub__(self, o):
        return self.a - o.a

    def __mul__(self, o):
        return self.a * o.a

ob1 = A(1)
ob2 = A(2)
ob3 = A("Python")
ob4 = A("Applications")

print(ob1 + ob2)
print(ob1 - ob2)
print(ob1 * ob2)
print(ob3 + ob4)
```

Output:

```
3

-1

2

PythonApplications
```

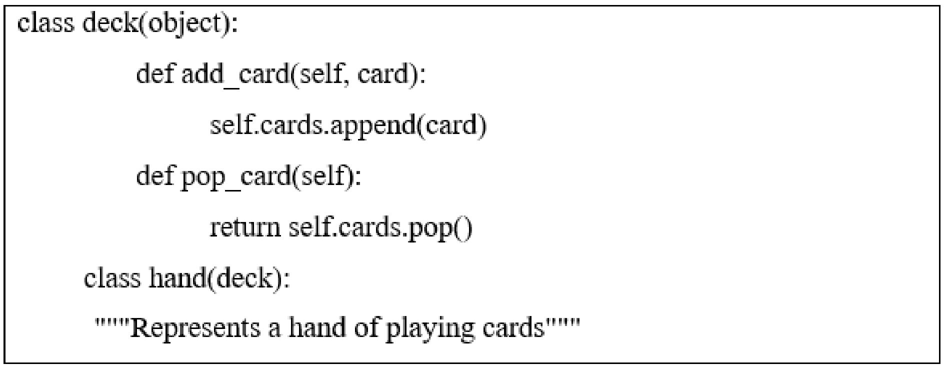
11 What are polymorphic functions? Explain with code snippet. **7**

Refer Module 4 Question 4 → Above

12 Illustrate the concept of inheritance with example **6**

Inheritance: The ability to define a new class that is a modified version of a previously defined class. The class from which a child class inherits is called as **Parent Class.** A new class created by inheriting from an existing class is called as **Child class** or **Sub Class.** The parent class will be represented in parenthesis while defining child class. Inheritance can facilitate code reuse and also behavior of parent classes can be customized without having to modify them.

Ex:



```
class deck(object):
        def add_card(self, card):
            self.cards.append(card)
        def pop_card(self):
            return self.cards.pop()
    class hand(deck):
     """Represents a hand of playing cards"""
```

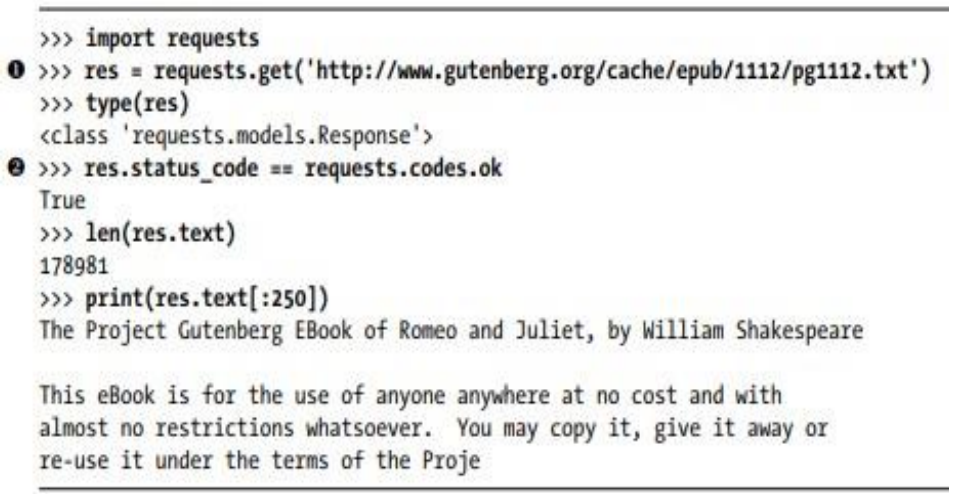
In the above example deck is parent class and hand is child class and the hand class can use add_card and pop_card.

MODULE 5

1 **Explain the process of downloading files from the Web with the requests module and also saving downloaded files to the hard drive with suitable example program.** **8**

<u>Downloading files from the Web with the requests module:</u>

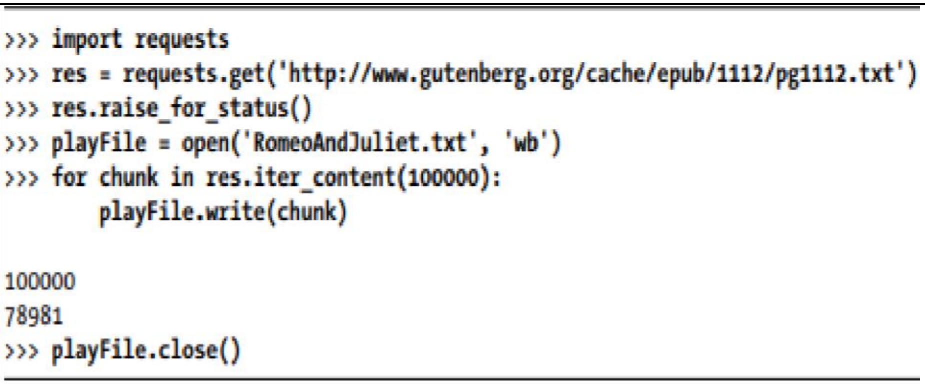
- The requests module lets us easily download files from the Web without having to worry about complicated issues such as network errors, connection problems, and data compression.
- **Ex:**



```
  >>> import requests
❶ >>> res = requests.get('http://www.gutenberg.org/cache/epub/1112/pg1112.txt')
  >>> type(res)
  <class 'requests.models.Response'>
❷ >>> res.status_code == requests.codes.ok
  True
  >>> len(res.text)
  178981
  >>> print(res.text[:250])
  The Project Gutenberg EBook of Romeo and Juliet, by William Shakespeare

  This eBook is for the use of anyone anywhere at no cost and with
  almost no restrictions whatsoever.  You may copy it, give it away or
  re-use it under the terms of the Proje
```

<u>Saving downloaded files to the hard drive</u>

- We can save the web page to a file on our hard drive with the standard open() function and write() method.
- We must open the file in write binary mode by passing the string 'wb' as the second argument to open().
- Even if the page is in plaintext, we need to write binary data instead of text data in order to maintain the Unicode encoding of the text.
- **Ex:**



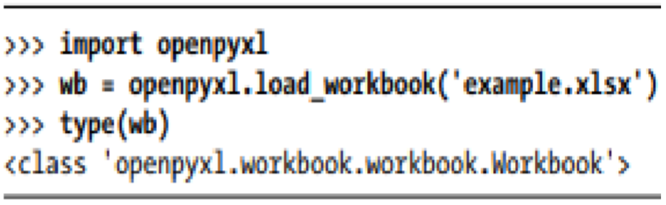
```
>>> import requests
>>> res = requests.get('http://www.gutenberg.org/cache/epub/1112/pg1112.txt')
>>> res.raise_for_status()
>>> playFile = open('RomeoAndJuliet.txt', 'wb')
>>> for chunk in res.iter_content(100000):
        playFile.write(chunk)

100000
78981
>>> playFile.close()
```

2 **Write a note on the following by demonstrating with code snippet.** **7**

i) **Opening Excel documents with openpyxl.**
ii) **Getting Sheets from the Workbook.**
iii) **Getting Cells, Rows and Columns from the Sheets.**

<u>Opening Excel documents with openpyxl.</u>

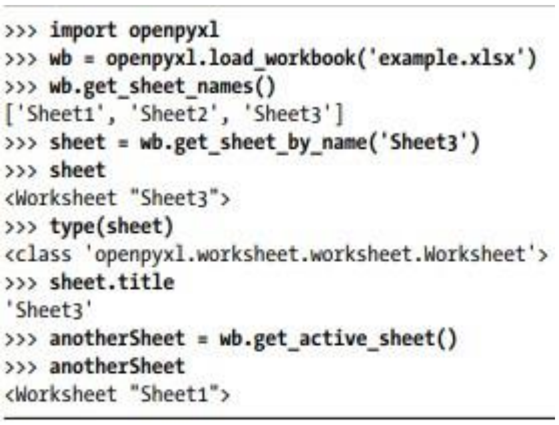
- ✓ Once we've imported the openpyxl module, we'll be able to use the openpyxl .load_workbook() function.
- ✓ The openpyxl.load_workbook() function takes in the filename and returns a value of the workbook data type.



```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> type(wb)
<class 'openpyxl.workbook.workbook.Workbook'>
```

<u>Getting Sheets from the Workbook.</u>

- ✓ We can get a list of all the sheet names in the workbook by calling the get_sheet_names() method.
- ✓ Each sheet is represented by a Worksheet object, which we can obtain by passing the sheet name string to the get_sheet_by_name() workbook method.
- ✓ We can call the get_active_sheet() method of a Workbook object to get the workbook's active sheet.



```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> wb.get_sheet_names()
['Sheet1', 'Sheet2', 'Sheet3']
>>> sheet = wb.get_sheet_by_name('Sheet3')
>>> sheet
<Worksheet "Sheet3">
>>> type(sheet)
<class 'openpyxl.worksheet.worksheet.Worksheet'>
>>> sheet.title
'Sheet3'
>>> anotherSheet = wb.get_active_sheet()
>>> anotherSheet
<Worksheet "Sheet1">
```

Getting Cells, Rows and Columns from the Sheets.

- ✓ Once we have a Worksheet object, we can access a Cell object by its name.
- ✓ Cell objects also have row, column, and coordinate attributes that provide location information for the cell.
- ✓ The row attribute gives us the integer 1, the column attribute gives us 'B', and the coordinate attribute gives us 'B1'.
- ✓ **Ex:**

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Sheet1')
>>> sheet['A1']
<Cell Sheet1.A1>
>>> sheet['A1'].value
datetime.datetime(2015, 4, 5, 13, 34, 2)
>>> c = sheet['B1']
>>> c.value
'Apples'
>>> 'Row ' + str(c.row) + ', Column ' + c.column + ' is ' + c.value
'Row 1, Column B is Apples'
>>> 'Cell ' + c.coordinate + ' is ' + c.value
'Cell B1 is Apples'
>>> sheet['C1'].value
73
```

3 **Describe the getText() function used for getting full text from a .docx file with example code.** **5**

- ✓ getText() function. It accepts a filename of a .docx file and returns a single string value of its text.
- ✓ getText() is used:
- ○ To modify the string before returning it.
- ○ To add a double space in between paragraphs, change the join() call code to this:
- ✓ **Ex**: In the below example the getText() function opens the Word document, loops over all the paragraph objects in the paragraphs list, and then appends their text to the list in fullText.

```
#! python3

import docx

def getText(filename):
    doc = docx.Document(filename)
    fullText = []
    for para in doc.paragraphs:
        fullText.append(para.text)
    return '\n'.join(fullText)
```

```
>>> import readDocx
>>> print(readDocx.getText('demo.docx'))
Document Title
A plain paragraph with some bold and some italic
Heading, level 1
Intense quote
first item in unordered list
first item in ordered list
```

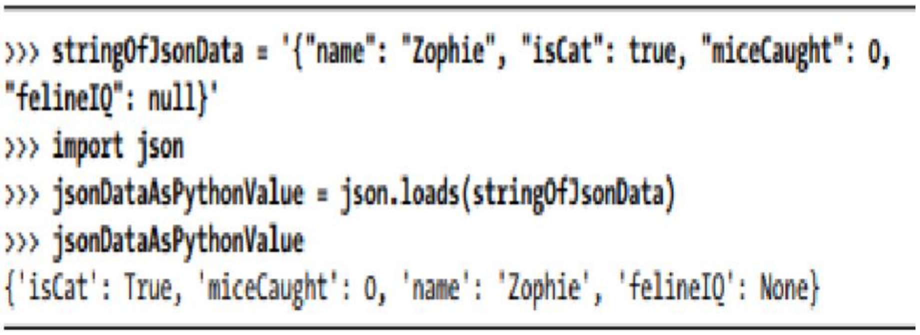
4 **Explain how to retrieve a web page element from a BeautifulSoup Object by calling the select method and passing a string of a CSS selector for the element you are looking for with an example program.** **8**

- ✓ Beautiful Soup is a module for extracting information from an HTML page.
- ✓ To install it, we will need to run pip install beautifulsoup4 from the command line.
- ✓ While beautifulsoup4 is the name used for installation, to import Beautiful Soup we run import bs4.
- ✓ The bs4.BeautifulSoup() function needs to be called with a string containing the HTML it will parse. The bs4.BeautifulSoup() function returns is a BeautifulSoup object.
- ✓ We can retrieve a web page element from a BeautifulSoup object by calling the select()method and passing a string of a CSS selector for the element you are looking for.
- ✓ **Ex:**

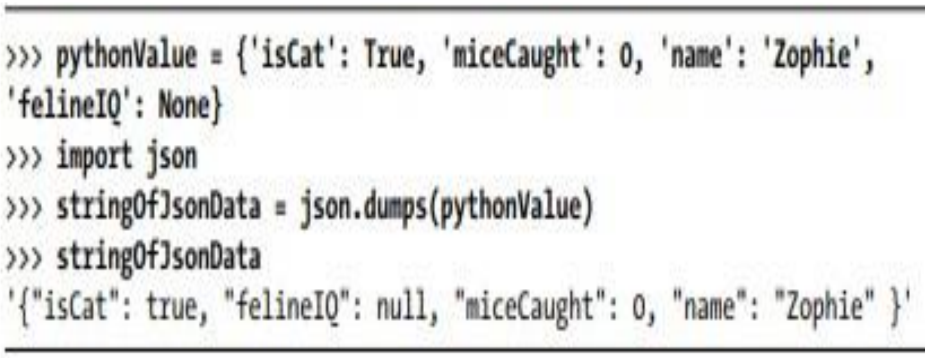
```
>>> import bs4
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile.read())
>>> elems = exampleSoup.select('#author')
>>> type(elems)
<class 'list'>
>>> len(elems)
1
>>> type(elems[0])
<class 'bs4.element.Tag'>
>>> elems[0].getText()
'Al Sweigart'
>>> str(elems[0])
'<span id="author">Al Sweigart</span>'
>>> elems[0].attrs
{'id': 'author'}
```

5 **What is JSON? Briefly explain the json module of Python. Demonstrate with a Python program.** 6

- ✓ JSON (pronounced "JAY-sawn" or "Jason") is a format that stores information as JavaScript source code in plaintext files.
- ✓ JSON is useful to know, because many websites offer JSON content as a way for programs to interact with the website. This is known as providing an application programming interface (API).
- ✓ Accessing an API is the same as accessing any other web page via a URL. **Json module**
- ✓ Python's json module handles all the details of translating between a string with JSON data and Python values for the json.loads() and json.dumps() functions.
- ✓ It can contain values of only the following data types: o strings, integers, floats, Booleans, lists, dictionaries, and NoneType.
- ✓ JSON cannot represent Python-specific objects, such as File objects, CSV Reader or Writer objects, Regex objects, or Selenium WebElement objects.
- ✓ **Example:**



```
>>> stringOfJsonData = '{"name": "Zophie", "isCat": true, "miceCaught": 0,
"felineIQ": null}'
>>> import json
>>> jsonDataAsPythonValue = json.loads(stringOfJsonData)
>>> jsonDataAsPythonValue
{'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}
```



```
>>> pythonValue = {'isCat': True, 'miceCaught': 0, 'name': 'Zophie',
'felineIQ': None}
>>> import json
>>> stringOfJsonData = json.dumps(pythonValue)
>>> stringOfJsonData
'{"isCat": true, "felineIQ": null, "miceCaught": 0, "name": "Zophie" }'
```

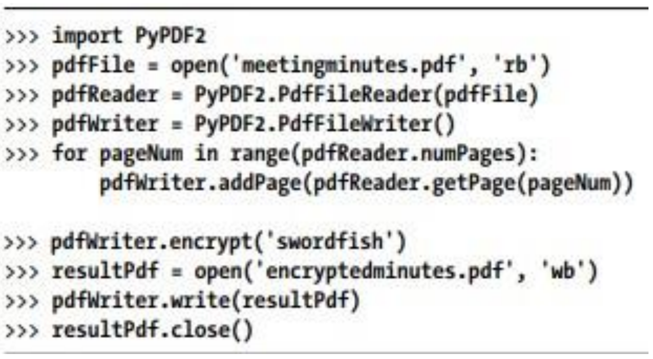
6 **Discuss the Creation, Encryption and Decryption of a PDF.** **6**

__Creating PDFs__

- ✓ PyPDF2’s counterpart to PdfFileReader objects is PdfFileWriter objects, which can create new PDF files.
- ✓ Creating a PdfFileWriter object creates only a value that represents a PDF document in Python. It doesn’t create the actual PDF file.
- ✓ The write() method takes a regular File object that has been opened in write-binary mode, which takes two arguments: the string of what we want the PDF’s filename to be and 'wb' to indicate the file should be opened in write-binary mode.

__Encrypting PDFs__

- ✓ A PdfFileWriter object can also add encryption to a PDF document.
- ✓ Before calling the write() method to save to a file, call the encrypt() method and pass it a password string .



```
>>> import PyPDF2
>>> pdfFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(pdfFile)
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> for pageNum in range(pdfReader.numPages):
        pdfWriter.addPage(pdfReader.getPage(pageNum))

>>> pdfWriter.encrypt('swordfish')
>>> resultPdf = open('encryptedminutes.pdf', 'wb')
>>> pdfWriter.write(resultPdf)
>>> resultPdf.close()
```

Decrypting PDFs

- ✓ Some PDF documents have an encryption feature that will keep them from being read until whoever is opening the document provides a password.
- ✓ All PdfFileReader objects have an isEncrypted attribute that is True if the PDF is encrypted and False if it isn't.
- ✓ To read an encrypted PDF, call the decrypt() function and pass the password as a string.
- ✓ After our program terminates, the file on our hard drive remains encrypted. our program will have to call decrypt() again the next time it is run.

7 How do we download a file and save it to harddrive using request module? **6**

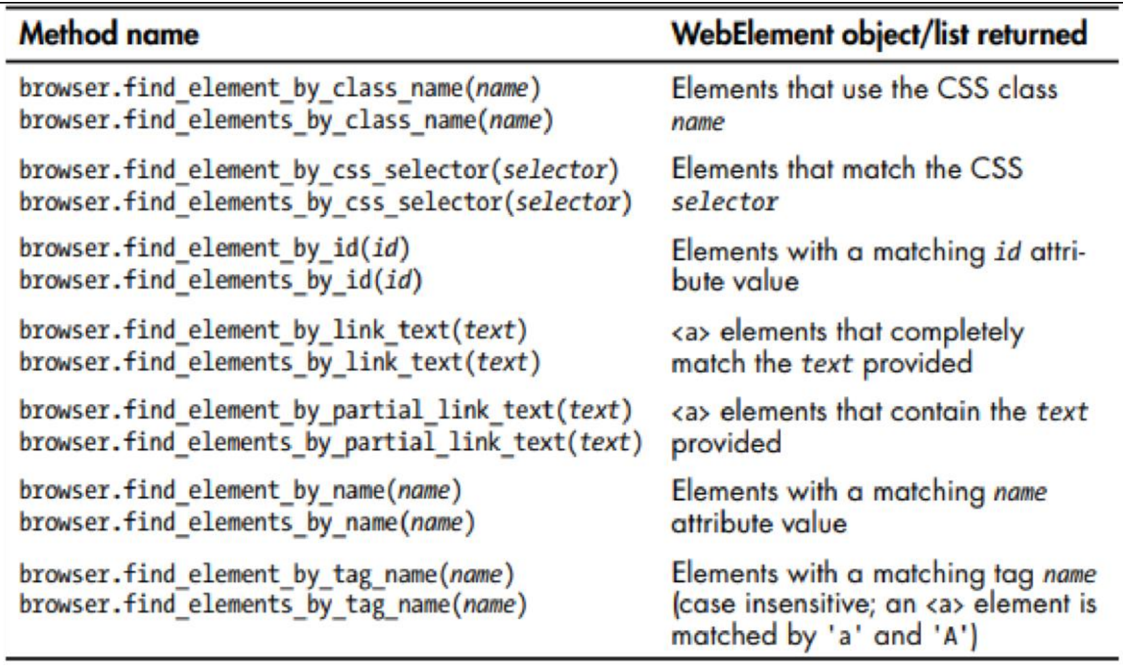
Refer Module 5 → Question 1

8 Write a python program to give search keyword from command line arguments and open the browser tab for each result page. **6**

```
Program: import requests, sys, webbrowser, bs4 print('Googling...')

# display text while downloading the Google page res =
requests.get('http://google.com/search?q=' + ' '.join(sys.argv[1:]))
res.raise_for_status()

# Retrieve top search result links. soup =
bs4.BeautifulSoup(res.text,"html.parser")

# Open a browser tab for each result.

linkElems = soup.select('.r a')
numOpen = min(5, len(linkElems))
for i in range(numOpen):
    webbrowser.open('http://google.com' + linkElems[i].get('href'))
```

9 Explain selenium's webdrive methods for finding elements. **8**

- ✓ WebDriver objects have quite a few methods for finding elements on a page.
- ✓ They are divided into the find_element_* and find_elements_* methods.
  - o The find_element_* methods return a single WebElement object, representing the first element on the page that matches your query.
  - o The find_elements_* methods return a list of WebElement_* objects for every matching element on the page.



Method name	WebElement object/list returned
browser.find_element_by_class_name(*name*) browser.find_elements_by_class_name(*name*)	Elements that use the CSS class *name*
browser.find_element_by_css_selector(*selector*) browser.find_elements_by_css_selector(*selector*)	Elements that match the CSS *selector*
browser.find_element_by_id(*id*) browser.find_elements_by_id(*id*)	Elements with a matching *id* attribute value
browser.find_element_by_link_text(*text*) browser.find_elements_by_link_text(*text*)	<a> elements that completely match the *text* provided
browser.find_element_by_partial_link_text(*text*) browser.find_elements_by_partial_link_text(*text*)	<a> elements that contain the *text* provided
browser.find_element_by_name(*name*) browser.find_elements_by_name(*name*)	Elements with a matching *name* attribute value
browser.find_element_by_tag_name(*name*) browser.find_elements_by_tag_name(*name*)	Elements with a matching tag *name* (case insensitive; an <a> element is matched by 'a' and 'A')

Table: Selenium's WebDriver Methods for Finding Elements

10 **Write a program that takes a number N from command line and creates an NxN multiplication table in excel spread sheet.** 8

```
#!/usr/bin/env python3

# This program takes a number N from the command line
# and creates an N by N multiplication table in an Excel spreadsheet.

import sys, openpyxl
from openpyxl.styles import Font
from openpyxl.utils import get_column_letter, column_index_from_string

# Default n is 6
if len(sys.argv) > 1:
n = int(sys.argv[1])
else:
   n = 6

# Open a new workbook and swtich the active worksheet
wb = openpyxl.Workbook()
sheet = wb.active

# Print the headers
boldFont = Font(bold=True)
for i in range(2, n+2):
multiplier = i - 1
leftCell = 'A' + str(i)
sheet[leftCell] = multiplier
sheet[leftCell].font = boldFont
rightCell = get_column_letter(i) + '1'
sheet[rightCell] = multiplier
sheet[rightCell].font = boldFont

# Print the multiplication table
```

```
for i in range(2, n+2):
leftMultiplier = i - 1
for j in range(2, n+2):
  rightMultiplier = j - 1
  cell = get_column_letter(j) + str(i)
  sheet[cell] = leftMultiplier*rightMultiplier

# Freeze the headers
sheet.freeze_panes = 'B2'

# Save to new Excel file in current working directory
wb.save('multiplicationTable.xlsx')
```

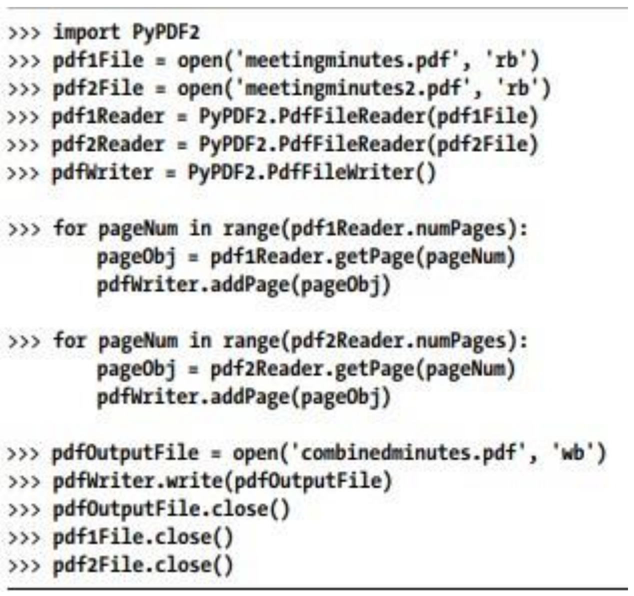
11 **Write short notes on: Creating, copying and rotating pages with respect to pdf.** **6**

<u>Creating PDFs</u>

- ✓ PyPDF2's counterpart to PdfFileReader objects is PdfFileWriter objects, which can create new PDF files.
- ✓ Creating a PdfFileWriter object creates only a value that represents a PDF document in Python. It doesn't create the actual PDF file.
- ✓ The write() method takes a regular File object that has been opened in write-binary mode, which takes two arguments: the string of what we want the PDF's filename to be and 'wb' to indicate the file should be opened in write-binary mode.

<u>Copying Pages</u>

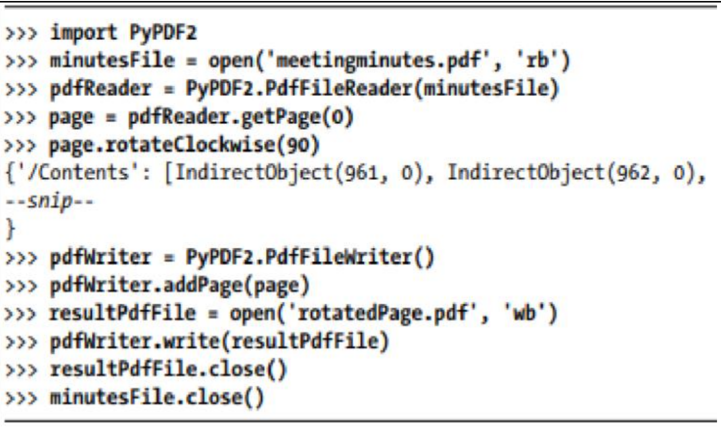
- ✓ We can use PyPDF2 to copy pages from one PDF document to another. This allows you to combine multiple PDF files, cut unwanted pages, or reorder pages.



```
>>> import PyPDF2
>>> pdf1File = open('meetingminutes.pdf', 'rb')
>>> pdf2File = open('meetingminutes2.pdf', 'rb')
>>> pdf1Reader = PyPDF2.PdfFileReader(pdf1File)
>>> pdf2Reader = PyPDF2.PdfFileReader(pdf2File)
>>> pdfWriter = PyPDF2.PdfFileWriter()

>>> for pageNum in range(pdf1Reader.numPages):
        pageObj = pdf1Reader.getPage(pageNum)
        pdfWriter.addPage(pageObj)

>>> for pageNum in range(pdf2Reader.numPages):
        pageObj = pdf2Reader.getPage(pageNum)
        pdfWriter.addPage(pageObj)

>>> pdfOutputFile = open('combinedminutes.pdf', 'wb')
>>> pdfWriter.write(pdfOutputFile)
>>> pdfOutputFile.close()
>>> pdf1File.close()
>>> pdf2File.close()
```

<u>Rotating Pages</u>

- ✓ The pages of a PDF can also be rotated in 90-degree increments with the rotateClockwise() and rotateCounterClockwise() methods.



```
>>> import PyPDF2
>>> minutesFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
>>> page = pdfReader.getPage(0)
>>> page.rotateClockwise(90)
{'/Contents': [IndirectObject(961, 0), IndirectObject(962, 0),
--snip--
}
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> pdfWriter.addPage(page)
>>> resultPdfFile = open('rotatedPage.pdf', 'wb')
>>> pdfWriter.write(resultPdfFile)
>>> resultPdfFile.close()
>>> minutesFile.close()
```

12 **Write a program that find all the CSV files in the current working directory, read in the full contents of each file, write out the contents, skipping the first line, to a new 6 CSV file.** **6**

```
#! python3
# removeCsvHeader.py - Removes the header from all CSV files in the current
# working directory.

import csv, os

os.makedirs('headerRemoved', exist_ok=True)

# Loop through every file in the current working directory.
for csvFilename in os.listdir('.'):
    if not csvFilename.endswith('.csv'):
        continue # skip non-csv files

    print('Removing header from ' + csvFilename + '...')

    # Read the CSV file in (skipping first row).
    csvRows = []
    csvFileObj = open(csvFilename)
    readerObj = csv.reader(csvFileObj)
    for row in readerObj:
        if readerObj.line_num == 1:
            continue # skip first row
        csvRows.append(row)
    csvFileObj.close()

    # Write out the CSV file.
    csvFileObj = open(os.path.join('headerRemoved', csvFilename), 'w', newline='')
    csvWriter = csv.writer(csvFileObj)
    for row in csvRows:
        csvWriter.writerow(row)
    csvFileObj.close()
```