Solution for Model question papers

Important Instruction:

1. **Attempt questions from all module don't skip any module. Write what ever you understood**
2. **Here this solution is complete,but while writing in exam according to marks you can write what ever is required don't write more than expectation and less also.**
3. **Check question is for how much marks and accordingly you should write ans**
4. ***Kind request write properly indented code***
5. ***Write answers pointwise like 1,2,3… etc***
6. ***Here for 2nd model paper I have given to you wher you can find ans in text book only python program solutions I gave please refer given page numbers to find ans.***
7. ***Wish you All the best to all my dear students***

Model Paper1 solution

Module1

1a. 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

Ans:

The *order of operations* (also called *precedence*) of Python math operators is similar to that of mathematics as below:

The ** operator is evaluated first; the *, /, //, and % operators are evaluated next, from left to right; and the + and - operators are evaluated last (also from left to right).

You can use parentheses to override the usual precedence if you need to. Whitespace in between the operators and values doesn't matter for Python (except for the indentation at the beginning of the line), but a single space is convention.

Python uses PEMDAS acronym to evaluate any expression

Example:

Table 1-1: Math Operators from Highest to Lowest Precedence

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

Few more examples

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565878 * 578453
28093077826734
>>> 2 ** 8
256
```

```
>>> 23 / 7
3.2857142857142856
>>> 23 // 7
3
>>> 23 % 7
2
>>> 2     +         2
4
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

Steps to evaluate (5 - 1) * ((7 + 1) / (3 - 1)) and reduces it to a single value

(5 - 1) * ((7 + 1) / (3 - 1))

4*(8/2)

4*4

16

1b. 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 thes evalues to the function. Suitable error messages should be displayed if the conditions for input values are not followed.

```
start = int(input("enter starting interval"))
end = int(input("enter end interval"))
def prime(start,end):
  if start>end:
    print("start should be smaller than end")
  else:

    for i in range(start, end+1):
      if i>1:
```

```
        for j in range(2,i):
          if(i % j==0):
             break
       else:

         print(i)

prime(start,end)
```

1c. 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?

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*. A variable must be one or the other; it cannot be both local and global.

Code in the global scope, outside of all functions, cannot use any local variables.

However, code in a local scope can access global variables.

Code in a function's local scope cannot use variables in any other local scope.

You 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.

Global Variables Can Be Read from a Local Scope

Consider the following program:

```
def spam():
   print(eggs)
eggs = 42
spam()
print(eggs)
```

You can view the execution of this program at https://autbor.com/readglobal/. Since there is no parameter named eggs or any code that assigns eggs a value in the spam() function, when eggs is used in spam(), Python considers it a reference to the global variable eggs. This is why 42 is printed when the previous program is run.

2a 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

Comparison Operators

Comparison operators, also called *relational operators*, compare two values and evaluate down to a single Boolean value. Table 2-1 lists the comparison operators.

Table 2-1: Comparison Operators

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

These operators evaluate to True or False depending on the values you give them. Let's try some operators now, starting with == and !=.

```
>>> 42 == 42
True
>>> 42 == 99
False
>>> 2 != 3
True
>>> 2 != 2
False
```

As you might expect, == (equal to) evaluates to True when the values on both sides are the same, and != (not equal to) evaluates to True when the two values are different.
The == and != operators can actually work with values of any data type.

```
>>> 'hello' == 'hello'
True
>>> 'hello' == 'Hello'
False
>>> 'dog' != 'cat'
True
>>> True == True
True
>>> True != False
True
>>> 42 == 42.0
True
❶ >>> 42 == '42'
False
```

Note that an integer or floating-point value will always be unequal to a string value. The expression 42 == '42' ❶ evaluates to False because Python considers the integer 42 to be different from the string '42'.

The <, >, <=, and >= operators, on the other hand, work properly only with integer and floating-point values.

```
>>> 42 < 100
True
>>> 42 > 100
False
>>> 42 < 42
False
>>> eggCount = 42
❶ >>> eggCount <= 42
True
>>> myAge = 29
❷ >>> myAge >= 10
True
```

Boolean Operators

The three Boolean operators (and, or, and not) are used to compare Boolean values. Like comparison operators, they evaluate these expressions down to a Boolean value. Let’s explore these operators in detail, starting with the and operator.

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

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

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

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

Mixing Boolean and Comparison Operators

Since the comparison operators evaluate to Boolean values, you can use them in expressions with the Boolean operators.

Example

```
>>> (4 < 5) and (5 < 6)
True
```

>>> (4 < 5) and (9 < 6)
False
>>> (1 == 2) or (2 == 2)
True

2b. 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 Fn = Fn-1 + Fn-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

```
def fibo(num):
  number_1=0
  number_2=1
  count=0
  if num <= 0:
      print("error")
  elif num==1:
    print(number_1)
  else:
    print("fibo sequesnce")
    while count < num:
      print(number_1)
      number_3=number_1+number_2
      number_1=number_2
      number_2=number_3
      count += 1

num =int(input("enter terms"))
fibo(num)
```

2c 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

getting an error, or *exception*, in your Python program means the entire program will crash. You don't want this to happen in real-world programs. Instead, you want the program to detect errors, handle them, and then continue to run.

For example, consider the following program, which has a divide-by-zero error. Open a file editor window and enter the following code, saving it as *zeroDivide.py*:

```
def spam(divideBy):
    return 42 / divideBy
```

```
print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

We've defined a function called spam, given it a parameter, and then printed the value of that function with various parameters to see what happens. This is the output you get when you run the previous code:

```
21.0
3.5
Traceback (most recent call last):
  File "C:/zeroDivide.py", line 6, in <module>
    print(spam(0))
  File "C:/zeroDivide.py", line 2, in spam
    return 42 / divideBy
ZeroDivisionError: division by zero
```

You can put the previous divide-by-zero code in a `try` clause and have an `except` clause contain code to handle what happens when this error occurs.

```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

When code in a try clause causes an error, the program execution immediately moves to the code in the except clause. After running that code, the execution continues as normal. The output of the previous program is as follows:

```
21.0
3.5
Error: Invalid argument.
```

None
42.0

Note that any errors that occur in function calls in a try block will also be caught. Consider the following program, which instead has the spam() calls in the try block:

```
def spam(divideBy):
    return 42 / divideBy

try:
    print(spam(2))
    print(spam(12))
    print(spam(0))
    print(spam(1))
except ZeroDivisionError:
    print('Error: Invalid argument.')
```

When this program is run, the output looks like this:

```
21.0
3.5
Error: Invalid argument.
```

Module2:

3a. What is Dictionary in Python? How is it different from List data type? Explain how a forloop can be used to traverse the keys of the Dictionary with an example.

a *dictionary* is a mutable collection of many values. But unlike indexes for lists, 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*.

In code, a dictionary is typed with braces, {}. Enter the following into the interactive shell:

```
>>> myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
```

Dictionaries vs. Lists

Unlike lists, items in dictionaries are unordered. The first item in a list named spam would be spam[0]. But there is no “first” item in a dictionary. While 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. Enter the following into the interactive shell:

```
>>> spam = ['cats', 'dogs', 'moose']
>>> bacon = ['dogs', 'moose', 'cats']
>>> spam == bacon
False
>>> eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}
>>> ham = {'species': 'cat', 'age': '8', 'name': 'Zophie'}
>>> eggs == ham
True
```

Because dictionaries are not ordered, they can’t be sliced like lists.

Trying to access a key that does not exist in a dictionary will result in a KeyError error message, much like a list’s “out-of-range” IndexError error message. Enter the following into the interactive shell, and notice the error message that shows up because there is no 'color' key:

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> spam['color']
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    spam['color']
KeyError: 'color'
```

Though dictionaries are not ordered, the fact that you can have arbitrary values for the keys allows you to organize your data in powerful ways

Explain how a forloop can be used to traverse the keys of the Dictionary with an example.

Here, a for loop iterates over each of the values in the spam dictionary. A for loop can also iterate over the keys or both keys and values:

```
>>> for k in spam.keys():
...     print(k)

color
age
>>> for i in spam.items():
...     print(i)
```

('color', 'red')
('age', 42)

```
>>> 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

3b . Explain the methods of List data type in Python for the following operations with suitable code snippets for each. (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

Adding Values to Lists with the append() and insert() Methods

To add new values to a list, use the append() and insert() methods. Enter the following into the interactive shell to call the append() method on a list value stored in the variable spam:

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

The previous append() method call adds the argument to the end of the list. The insert() method can 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. Enter the following into the interactive shell:

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

Notice that the code is spam.append('moose') and spam.insert(1, 'chicken'), not spam = spam.append('moose') and spam = spam.insert(1, 'chicken'). Neither append() nor insert() gives the new value of spam as its return value. (In fact, the return value of append() and insert() is None, so you definitely wouldn't want to store this as the new variable

value.) Rather, the list is modified *in place*. Modifying a list in place is covered in more detail later in “Mutable and Immutable Data Types” on page 94.

Methods belong to a single data type. The append() and insert() methods are list methods and can be called only on list values, not on other values such as strings or integers. Enter the following into the interactive shell, and note the AttributeError error messages that show up:

```
>>> eggs = 'hello'
>>> eggs.append('world')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    eggs.append('world')
AttributeError: 'str' object has no attribute 'append'
>>> bacon = 42
>>> bacon.insert(1, 'world')
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    bacon.insert(1, 'world')
AttributeError: 'int' object has no attribute 'insert'
```

Removing Values from Lists with the remove() Method

The remove() method is passed the value to be removed from the list it is called on. Enter the following into the interactive shell:

```
>>> 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. For example, enter the following into the interactive shell and notice the error that is displayed:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('chicken')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    spam.remove('chicken')
ValueError: list.remove(x): x not in list
```

If the value appears multiple times in the list, only the first instance of the value will be removed. Enter the following into the interactive shell:

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

The del statement is good to use when you know the index of the value you want to remove from the list. The remove() method is useful when you know the value you want to remove from the list.

Finding a Value in a List with the index() Method

List values have an index() method that can be passed a value, and if that value exists in the list, the index of the value is returned. If the value isn't in the list, then Python produces a ValueError error. Enter the following into the interactive shell:

```
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> spam.index('hello')
0
>>> spam.index('heyas')
3
>>> spam.index('howdy howdy howdy')
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    spam.index('howdy howdy howdy')
ValueError: 'howdy howdy howdy' is not in list
```

When there are duplicates of the value in the list, the index of its first appearance is returned. Enter the following into the interactive shell, and notice that index() returns 1, not 3:

```
>>> spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
>>> spam.index('Pooka')
1
```

Sorting the Values in a List with the sort() Method

Lists of number values or lists of strings can be sorted with the sort() method. For example, enter the following into the interactive shell:

```
>>> 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']
```

You can also pass True for the reverse keyword argument to have sort() sort the values in reverse order. Enter the following into the interactive shell:

```
>>> spam.sort(reverse=True)
>>> spam
['elephants', 'dogs', 'cats', 'badgers', 'ants']
```

There are three things you should note about the sort() method. First, the sort() method sorts the list in place; don't try to capture the return value by writing code like spam = spam.sort().

Second, you cannot sort lists that have both number values *and* string values in them, since Python doesn't know how to compare these values. Enter the following into the interactive shell and notice the TypeError error:

```
>>> spam = [1, 3, 2, 4, 'Alice', 'Bob']
>>> spam.sort()
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    spam.sort()
TypeError: '<' not supported between instances of 'str' and 'int'
```

Third, sort() uses "ASCIIbetical order" rather than actual alphabetical order for sorting strings. This means uppercase letters come before lowercase letters. Therefore, the lowercase *a* is sorted so that it comes *after* the uppercase *Z*. For an example, enter the following into the interactive shell:

```
>>> spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
>>> spam.sort()
>>> spam
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

If you need to sort the values in regular alphabetical order, pass str.lower for the key keyword argument in the sort() method call.

```
>>> spam = ['a', 'z', 'A', 'Z']
>>> spam.sort(key=str.lower)
>>> spam
['a', 'A', 'z', 'Z']
```

This causes the sort() function to treat all the items in the list as if they were lowercase without actually changing the values in the list.

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

```
string=input("Enter string:")
count1=0
count2=0
C3=0
words=len(string.split())
print("the number of wors in sentences",words)
for i in string:
    if(i.islower()):
        count1=count1+1
    elif(i.isupper()):
        count2=count2+1
    elif i.isdigit():
      C3=C3+1
    else:
     pass

print("The number of lowercase characters is:")
print(count1)
print("The number of uppercase characters is:")
print(count2)
```

```
print("the number of digits")
print(C3)
```

output :

```
Enter string:i am ashwini123
the number of wors in sentences 3
The number of lowercase characters is:
10
The number of uppercase characters is:
0
the number of digits
3
```

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

The copy Module's copy() and deepcopy() Functions

passing around references is often the handiest way to deal with lists and dictionaries, if the function modifies the list or dictionary that is passed, you may not want these changes in the original list or dictionary value. For this, Python provides a module named copy that provides both the copy() and deepcopy() functions. The first of these, copy.copy(), can be used to make a duplicate copy of a mutable value like a list or dictionary, not just a

copy of a reference

for example

```
>>> import copy
>>> spam = ['A', 'B', 'C', 'D']
>>> cheese = copy.copy(spam)
>>> cheese[1] = 42
>>> spam
['A', 'B', 'C', 'D']
>>> cheese
['A', 42, 'C', 'D']
```

Now the spam and cheese variables refer to separate lists, which is why only the list in cheese is modified when you assign 42 at index 7. As you can see in Figure 4-7, the reference ID numbers are no longer the same for both variables because the variables refer to independent lists

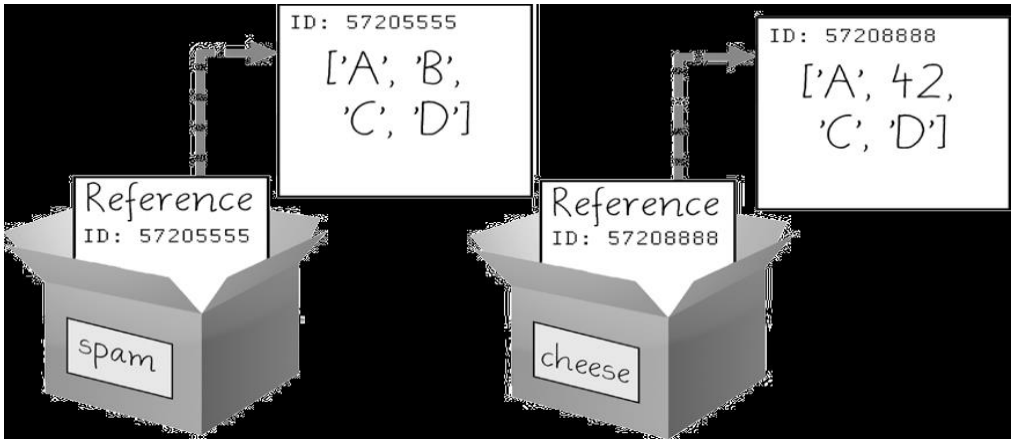

Figure 4-7: cheese = copy.copy(spam) creates a second list that can be modified independently of the first.

If the list you need to copy contains lists, then use the copy.deepcopy()

function instead of copy.copy(). The deepcopy() function will copy these inner lists as well

4b.

Discuss the following Dictionary methods in Python with examples.

(i) get() (ii) items() (iii) keys() (iv) values()

The get() Method

It's tedious to check whether a key exists in a dictionary before accessing that key's value. Fortunately, dictionaries have a get() method that takes two arguments: the key of the value to retrieve and a fallback value to return if that key does not exist.

Enter the following into the interactive shell:

```
>>> 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.'
```

Because there is no 'eggs' key in the picnicItems dictionary, the default value 0 is returned by the get() method. Without using get(), the code would have caused an error message, such as in the following example:

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
Traceback (most recent call last):
File "<pyshell#34>", line 1, in <module>
'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
KeyError: 'eggs
```

The keys(), values(), and items() Methods

There are three dictionary methods that will return list-like values of the dictionary's keys, values, or both keys and values: keys(), values(), and items(). The values returned by these methods are not true lists: They cannot be modified and do not have an append() method. But these data types (dict_keys,

dict_values, and dict_items, respectively) can be used in for loops. To see how these methods work, enter the following into the interactive shell:

```
>>> spam = {'color': 'red', 'age': 42}
>>> for v in spam.values():
print(v)
red
42
```

Here, a for loop iterates over each of the values in the spam dictionary.

A for loop can also iterate over the keys or both keys and values:

```
>>> for k in spam.keys():
print(k)
color
age
>>> for i in spam.items():
print(i)
('color', 'red')
('age', 42)
```

Using the keys(), values(), and items() methods, a for loop can iterate over the keys, values, or key-value pairs in a dictionary, respectively. Notice that the values in the dict_items value returned by the items() method are tuples of the key and value.

If you want a true list from one of these methods, pass its list-like return value to the list() function. Enter the following into the interactive shell:

```
>>> spam = {'color': 'red', 'age': 42}
>>> spam.keys()
dict_keys(['color', 'age'])
>>> list(spam.keys())
['color', 'age']
```

The list(spam.keys()) line takes the dict_keys value returned from keys() and passes it to list(), which then returns a list value of ['color', 'age'].

You can also use the multiple assignment trick in a for loop to assign the key and value to separate variables. Enter the following into the interactive shell:

```
>>> 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
```

4c.

Explain the various string methods for the following operations with examples.

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

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

Removing Whitespace with strip(), rstrip(), and lstrip()

Sometimes you may want to strip off whitespace characters (space, tab, and newline) from the left side, right side, 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. Enter the following into the interactive shell:

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

Optionally, a string argument will specify which characters on the ends should be stripped. Enter the following into the interactive shell:

```
>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
>>> spam.strip('ampS')
'BaconSpamEggs'
```

Passing strip() the argument 'ampS' will tell it to strip occurences of a, m, p, and capital S from the ends of the string stored in spam. The order of the characters in the string passed to strip() does not matter: strip('ampS') will do the same thing as strip('mapS') or strip('Spam').

Justifying Text with rjust(), ljust(), and center()

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. Enter the following into the interactive shell:

```
>>> 'Hello'.rjust(10)
```

' Hello'

>>> 'Hello'.rjust(20)

' Hello'

>>> 'Hello World'.rjust(20)

' Hello World'

>>> 'Hello'.ljust(10)

'Hello '

'Hello'.rjust(10) says that we want to right-justify 'Hello' in a string of total length 10. 'Hello' is five characters, so five spaces will be added to its left, giving us a string of 10 characters with 'Hello' justified right.

An optional second argument to rjust() and ljust() will specify a fill character other than a space character. Enter the following into the interactive shell:

>>> '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. Enter the following into the interactive shell:

>>> 'Hello'.center(20)

' Hello '

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

'=======Hello========'

These methods are especially useful when you need to print tabular

data that has the correct spacing. Open a new file editor window and enter the following code, saving it as picnicTable.py:

```
def printPicnic(itemsDict, leftWidth, rightWidth):
print('PICNIC ITEMS'.center(leftWidth + rightWidth, '-'))
for k, v in itemsDict.items():
print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))
picnicItems = {'sandwiches': 4, 'apples': 12, 'cups': 4, 'cookies': 8000}
printPicnic(picnicItems, 12, 5)
printPicnic(picnicItems, 20, 6)
```

output

```
---PICNIC ITEMS--
sandwiches.. 4
apples..... 12
cups..... 4
cookies..... 8000
-------PICNIC ITEMS-------
sandwiches..... 4
apples..... 12
cups..... 4
cookies..... 8000
```

Module3

5a. Write a Python Program to find an American phone number (example: 415-555-4242) in a given string using Regular Expressions.

A Regex object's search() method searches the string it is passed for any matches to the regex. The search() method will return None if the regex pattern is not found in the string. If the pattern is found, the search() method returns a Match object. Match objects have a group() method that will return the actual matched text from the searched string. (I'll explain groups shortly.) For example, enter the following into the interactive shell:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> print('Phone number found: ' + mo.group())
```

Phone number found: 415-555-4242

The mo variable name is just a generic name to use for Match objects.
This example might seem complicated at first, but it is much shorter than the earlier isPhoneNumber.py program and does the same thing.
Here, we pass our desired pattern to re.compile() and store the resulting Regex object in phoneNumRegex. Then we call search() on phoneNumRegex and pass search() the string we want to search for a match. The result of the search gets stored in the variable mo. In this example, we know that our pattern will be found in the string, so we know that a Match object will be returned.

Knowing that mo contains a Match object and not the null value None, we can call group() on mo to return the match. Writing mo.group() inside our print statement displays the whole match, 415-555-4242.

5b. Describe the difference between Python os and os.path modules. Also, discuss the following methods of os module

chdir() b) rmdir() c) walk() d) listdir() e) getcwd()

On Windows, paths are written using backslashes (\) as the separator between folder names. OS X and Linux, however, use the forward slash (/) as their path separator. If you want your programs to work on all operating systems, you will have to write your Python scripts to handle both cases. Fortunately, this is simple to do with the os.path.join() function. If you pass it the string values of individual file and folder names in your path, os.path.join() will return a string with a file path using the correct path separators. Enter the following into the interactive shell:

```
>>> import os
>>> os.path.join('usr', 'bin', 'spam')
'usr\\bin\\spam
```

The Current Working Directory

Every program that runs on your computer has a current working directory,

or cwd. Any filenames or paths that do not begin with the root folder are assumed to be under the current working directory. You can get the current working directory as a string value with the os.getcwd() function and change it with os.chdir(). Enter the following into the interactive shell:

```
>>> import os
>>> os.getcwd()
'C:\\Python34'
>>> os.chdir('C:\\Windows\\System32')


>>> os.getcwd()
'C:\\Windows\\System32'
```

Here, the current working directory is set to C:\Python34, so the filename project.docx refers to C:\Python34\project.docx. When we change the current working directory to C:\Windows, project.docx is interpreted as C:\Windows\project.docx.

Finding File Sizes and Folder Contents

Once you have ways of handling file paths, you can then start gathering information about specific files and folders. The os.path module provides functions for finding the size of a file in bytes and the files and folders inside a given folder.

- Calling os.path.getsize(path) will return the size in bytes of the file in the path argument.
- Calling os.listdir(path) will return a list of filename strings for each file

in the path argument. (Note that this function is in the os module, not os.path.)

Here's what I get when I try these functions in the interactive shell:

```
>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')
776192
>>> os.listdir('C:\\Windows\\System32')
['0409', '12520437.cpx', '12520850.cpx', '5U877.ax', 'aaclient.dll',
--snip--
'xwtpdui.dll', 'xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW', 'zipfldr.dll']
```

As you can see, the calc.exe program on my computer is 776,192 bytes in size, and I have a lot of files in C:\Windows\system32. If I want to find the total size of all the files in this directory, I can use os.path.getsize() and os.listdir() together.

Say you want to rename every file in some folder and also every file in every subfolder of that folder. That is, you want to walk through the directory tree, touching each file as you go. Writing a program to do this could get tricky; fortunately, Python provides a function to handle this process for you.

Here is an example program that uses the os.walk() function on the directory tree from Figure 9-1:

```
import os
for folderName, subfolders, filenames in os.walk('C:\\delicious'):
print('The current folder is ' + folderName)
```

```
for subfolder in subfolders:
print('SUBFOLDER OF ' + folderName + ': ' + subfolder)


for filename in filenames:
print('FILE INSIDE ' + folderName + ': '+ filename)
print('')
```

The os.walk() function is passed a single string value: the path of a folder. You can use os.walk() in a for loop statement to walk a directory tree, much like how you can use the range() function to walk over a range of numbers. Unlike range(), the os.walk() function will return three values on each iteration through the loop:

1. A string of the current folder's name
2. A list of strings of the folders in the current folder
3. A list of strings of the files in the current folder

5c Demonstrate the copy, move, rename and delete functions of shutil module with Python code snippet

shutil module

Copying Files and Folders

The shutil module provides functions for copying files, as well as entire folders.

Calling shutil.copy(source, destination) will copy the file at the path source to the folder at the path destination. (Both source and destination are strings.) If destination is a filename, it will be used as the new name of the copied file. This function returns a string of the path of the copied file.

Enter the following into the interactive shell to see how shutil.copy() works:

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

The first shutil.copy() call copies the file at C:\spam.txt to the folder C:\delicious. The return value is the path of the newly copied file. Note that since a folder was specified as the destination u, the original spam.txt filename is used for the new, copied file's filename. The second shutil.copy() call v also copies the file at C:\eggs.txt to the folder C:\delicious but gives the copied file the name eggs2.txt.

While shutil.copy() will copy a single file, shutil.copytree() will copy an entire folder and every folder and file contained in it. Calling shutil.copytree(source, destination) will copy the folder at the path source, along with all of its files and subfolders, to the folder at the path destination.

The source and destination parameters are both strings. The function returns a string of the path of the copied folder.

Enter the following into the interactive shell:

```
>>> import shutil, os
>>> os.chdir('C:\\')
>>> shutil.copytree('C:\\bacon', 'C:\\bacon_backup')
'C:\\bacon_backup'
```

The shutil.copytree() call creates a new folder named bacon_backup with the same content as the original bacon folder. You have now safely backed up your precious, precious bacon.

Moving and Renaming Files and Folders

Calling shutil.move(source, destination) will move the file or folder at the path source to the path destination and will return a string of the absolute path of the new location.

If destination points to a folder, the source file gets moved into destination and keeps its current filename. For example, enter the following into the interactive shell:

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

Assuming a folder named eggs already exists in the C:\ directory, this shutil.move() calls says, “Move C:\bacon.txt into the folder C:\eggs.”

If there had been a bacon.txt file already in C:\eggs, it would have been overwritten. Since it’s easy to accidentally overwrite files in this way, you should take some care when using move().

The destination path can also specify a filename. In the following

example,

the source file is moved and renamed.

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

This line says, “Move C:\bacon.txt into the folder C:\eggs, and while you’re at it, rename that bacon.txt file to new_bacon.txt.”

Both of the previous examples worked under the assumption that there was a folder eggs in the C:\ directory. But if there is no eggs folder, then move() will rename bacon.txt to a file named eggs.

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

Here, move() can’t find a folder named eggs in the C:\ directory and so assumes that destination must be specifying a filename, not a folder. So the bacon.txt text file is renamed to eggs (a text file without the .txt file extension)—probably not what you wanted! This can be a tough-to-spot bug in your programs since the move() call can happily do something that might be quite different from what you were expecting. This is yet another reason to be careful when using move().

Permanently Deleting Files and Folders

You 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, you use the shutil module.

- Calling os.unlink(path) will delete the file at path.
- Calling os.rmdir(path) will delete the folder at path. This folder must be

empty of any files or folders.

• Calling shutil.rmtree(path) will remove the folder at path, and all files and folders it contains will also be deleted.

a Python program that was intended to delete files that have the .txt file extension but has a typo (highlighted in bold) that causes it to delete .rxt files instead:

```
import os
for filename in os.listdir():
if filename.endswith('.rxt'):
os.unlink(filename)
```

If you had any important files ending with .rxt, they would have been accidentally, permanently deleted. Instead, you should have first run the program like this:

```
import os
for filename in os.listdir():
if filename.endswith('.rxt'):
#os.unlink(filename)
print(filename)
```

Now the os.unlink() call is commented, so Python ignores it. Instead, you will print the filename of the file that would have been deleted. Running this version of the program first will show you that you've accidentally told the program to delete .rxt files instead of .txt files.

Once you are certain the program works as intended, delete the print(filename) line and uncomment the os.unlink(filename) line. Then run the program again to actually delete the files.

Safe Deletes with the send2trash Module

Since Python's built-in shutil.rmtree() function irreversibly deletes files and folders, it can be dangerous to use. A much better way to delete files and folders is with the third-party send2trash module. You can install this module by running pip install send2trash from a Terminal window. (See Appendix A for a more in-depth explanation of how to install third-party modules.) Using send2trash is much safer than Python's regular delete functions, because it will send folders and files to your computer's trash or recycle bin instead of permanently deleting them. If a bug in your program deletes something with send2trash you didn't intend to delete, you can later restore it from the recycle bin.

After you have installed send2trash, enter the following into the interactive shell:

```
>>> import send2trash
>>> baconFile = open('bacon.txt', 'a') # creates the file
>>> baconFile.write('Bacon is not a vegetable.')
```



```
>>> baconFile.close()
>>> send2trash.send2trash('bacon.txt')
```

6a Describe the following with suitable Python code snippet.

(i) Greedy and Non Greedy Pattern Matching

(ii) findall() method of Regex object.

Greedy and Nongreedy Matching

Since (Ha){3,5} can match three, four, or five instances of Ha in the string 'HaHaHaHaHa', you may wonder why the Match object's call to group() in the

previous curly bracket example returns 'HaHaHaHaHa' instead of the shorter possibilities. After all, 'HaHaHa' and 'HaHaHaHa' are also valid matches of the regular expression (Ha){3,5}.

Python's regular expressions are greedy by default, which means that in ambiguous situations they will match the longest string possible. The nongreedy version of the curly brackets, which matches the shortest string possible, has the closing curly bracket followed by a question mark.

Enter the following into the interactive shell, and notice the difference between the greedy and nongreedy forms of the curly brackets searching the same string:

```
>>> greedyHaRegex = re.compile(r'(Ha){3,5}')
>>> mo1 = greedyHaRegex.search('HaHaHaHaHa')
>>> mo1.group()
'HaHaHaHaHa'
>>> nongreedyHaRegex = re.compile(r'(Ha){3,5}?')
>>> mo2 = nongreedyHaRegex.search('HaHaHaHaHa')
>>> mo2.group()
'HaHaHa'
```

Note that the question mark can have two meanings in regular expressions:

declaring a nongreedy match or flagging an optional group. These meanings are entirely unrelated.

The findall() Method

In addition to the search() method, Regex objects also have a findall() method. While search() will return a Match object of the first matched text in the searched string, the findall() method will return the strings of every match in the searched string. To see how search() returns a Match object only on the first instance of matching text, enter the following into the interactive shell:

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('Cell: 415-555-9999 Work: 212-555-0000')
>>> mo.group()
'415-555-9999'
```

On the other hand, findall() will not return a Match object but a list of strings—as long as there are no groups in the regular expression. Each string in the list is a piece of the searched text that matched the regular expression.

Enter the following into the interactive shell:

```
>>> 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']
```

If there are groups in the regular expression, then findall() will return

a list of tuples. Each tuple represents a found match, and its items are the matched strings for each group in the regex. To see findall() in action, enter

the following into the interactive shell (notice that the regular expression

being compiled now has groups in parentheses):

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)') # has groups
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
[('415', '555', '1122'), ('212', '555', '0000')]
```

To summarize what the findall() method returns, remember the

following:

1. When called on a regex with no groups, such as \d\d\d-\d\d\d-\d\d\d\d, the method findall() returns a list of string matches, such as ['415-555-9999', '212-555-0000'].

2. When called on a regex that has groups, such as (\d\d\d)-(\d\d\d)-(\d\d\d\d), the method findall() returns a list of tuples of strings (one string for each group), such as [('415', '555', '1122'), ('212', '555', '0000')].

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

There are three steps to reading or writing files in Python.

1. Call the open() function to return a File object.
2. Call the read() or write() method on the File object.

3. Close the file by calling the close() method on the File object.

Opening Files with the open() Function

To open a file with the open() function, you pass it a string path indicating the file you want to open; it can be either an absolute or relative path. The open() function returns a File object.

Try it by creating a text file named hello.txt using Notepad or TextEdit. Type Hello world! as the content of this text file and save it in your user home folder. Then, if you're using Windows, enter the following into the interactive shell:

```
>>> helloFile = open('C:\\Users\\your_home_folder\\hello.txt')
```

If you're using OS X, enter the following into the interactive shell instead:

```
>>> helloFile = open('/Users/your_home_folder/hello.txt')
```

Reading the Contents of Files

Now that you have a File object, you can start reading from it. If you want to read the entire contents of a file as a string value, use the File object's read() method. Let's continue with the hello.txt File object you stored in helloFile . Enter the following into the interactive shell:

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

Alternatively, you can use the readlines() method to get a list of string values from the file, one string for each line of text. For example, create a file named sonnet29.txt in the same directory as hello.txt and write the following text in it:

When, in disgrace with fortune and men's eyes,
I all alone beweep my outcast state,
And trouble deaf heaven with my bootless cries,
And look upon myself and curse my fate,

Make sure to separate the four lines with line breaks. Then enter the following into the interactive shell:

```
>>> sonnetFile = open('sonnet29.txt')
>>> sonnetFile.readlines()
[When, in disgrace with fortune and men's eyes,\n', ' I all alone beweep my outcast state,\n', And trouble deaf heaven with my bootless cries,\n', And look upon myself and curse my fate,']
```

Writing to Files

Python allows you to write content to a file in a way similar to how the print() function “writes” strings to the screen. You can’t write to a file you’ve opened in read mode, though. Instead, you need to open it in “write plaintext” mode or “append plaintext” mode, or write mode and append mode for short. Write mode will overwrite the existing file and start from scratch, just like when you overwrite a variable’s value with a new value. Pass 'w' as the second argument to open() to open the file in write mode. Append mode,

on the other hand, will append text to the end of the existing file. You can think of this as appending to a list in a variable, rather than overwriting the variable altogether. Pass 'a' as the second argument to open() to open the file in append mode.

If the filename passed to open() does not exist, both write and append mode will create a new, blank file. After reading or writing a file, call the close() method before opening the file again.

Let’s put these concepts together. Enter the following into the interactive Shell

```
>>> baconFile = open('bacon.txt', 'w')
>>> baconFile.write('Hello world!\n')
13
>>> baconFile.close()
>>> baconFile = open('bacon.txt', 'a')
>>> baconFile.write('Bacon is not a vegetable.')
25
>>> baconFile.close()
>>> baconFile = open('bacon.txt')
>>> content = baconFile.read()
>>> baconFile.close()
>>> print(content)
Hello world!
Bacon is not a vegetable.
```

6c . 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

Assertions

An assertion is a sanity check to make sure your code isn't doing something obviously wrong. These sanity checks are performed by assert statements. If the sanity check fails, then an AssertionError exception is raised. In code, an assert statement consists of the following:

• The assert keyword

• A condition (that is, an expression that evaluates to True or False)

• A comma

• A string to display when the condition is False

For example, enter the following into the interactive shell:

```
>>> podBayDoorStatus = 'open'
>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
>>> podBayDoorStatus = 'I\'m sorry, Dave. I\'m afraid I can't do that."
>>> assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>
assert podBayDoorStatus == 'open', 'The pod bay doors need to be "open".'
AssertionError: The pod bay doors need to be "open".
```

Using an Assertion in a Traffic Light Simulation

Say you’re building a traffic light simulation program. 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'. The code would look something like this:

```
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. To start the project, you want to write a switchLights() function, which will take an intersection dictionary as an argument and switch the lights.
At first, you might think that switchLights() should simply switch each light to the next color in the sequence: Any 'green' values should change to 'yellow', 'yellow' values should change to 'red', and 'red' values should change to 'green'. The code to implement this idea might look like this:

```
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)
```

You may already see the problem with this code, but let's pretend you wrote the rest of the simulation code, thousands of lines long, without noticing it. When you finally do run the simulation, the program doesn't crash—but your virtual cars do!

Since you've already written the rest of the program, you have no idea where the bug could be. Maybe it's in the code simulating the cars or in the code simulating the virtual drivers. It could take hours to trace the bug back to the switchLights() function.

But if while writing switchLights() you had added an assertion to check that at least one of the lights is always red, you might have included the following at the bottom of the function:

```
assert 'red' in stoplight.values(), 'Neither light is red! ' + str(stoplight)
```

With this assertion in place, your program would crash with this error message:

```
Traceback (most recent call last):
File "carSim.py", line 14, in <module>
switchLights(market_2nd)
File "carSim.py", line 13, in switchLights
```

assert 'red' in stoplight.values(), 'Neither light is red! ' + str(stoplight)

u AssertionError: Neither light is red! {'ns': 'yellow', 'ew': 'green'}

Module4

7a. 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.

Refer IAT3 Solution and ppts

7b. Illustrate the concept of modifier with Python code.

Refer solution for IAT3

7c Explain init and __str__ method with an example Python Program

Refer IAT3 solution and ppts

8a. 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.

Refer IAt3 solution

8b Illustrate the concept of pure function with Python code.

Refer IAT3 Solution

8c

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

Refer notes and ppts

Module5

9a. 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

Downloading Files from the Web with the requests Module

Downloading a Web Page with the requests.get() Function

The requests.get()function takes a string of a URL to download. By calling

type() on requests.get()'s return value, you can see that it returns a Response

object, which contains the response that the web server gave for your request.

I'll explain the Response object in more detail later, but for now, enter the

following into the interactive shell while your computer is connected to

the Internet:

>>> import requests

u >>> res = requests.get('http://www.gutenberg.org/cache/epub/1112/pg1112.txt')

>>> type(res)

<class 'requests.models.Response'>

v >>> 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
```

Saving Downloaded Files to the Hard Drive

From here, you can save the web page to a file on your hard drive with the standard open() function and write() method. There are some slight differences, though. First, you 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 (such as the Romeo and Juliet text you downloaded earlier), you need to write binary data instead of text data in order to maintain the Unicode encoding of the text.

To write the web page to a file, you can use a for loop with the Response object's iter_content() method.

```
>>> 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()
```

9b Write a note on the following by demonstrating with code snippet.

(i) Opening Excel documents with openpyxl.

(ii) Getting Sheets from the Workbook.

(iii)Getting Cells, Rows and Columns from the Sheets

Opening Excel Documents with OpenPyXL

Once you've imported the openpyxl module, you'll be able to use the openpyxl .load_workbook() function. Enter the following into the interactive shell:

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

The openpyxl.load_workbook() function takes in the filename and returns a value of the workbook data type. This Workbook object represents the Excel file, a bit like how a File object represents an opened text file.

Remember that example.xlsx needs to be in the current working directory in order for you to work with it. You can find out what the current working directory is by importing os and using os.getcwd(), and you can

change the current working directory using os.chdir().

Getting Sheets from the Workbook

You can get a list of all the sheet names in the workbook by calling the get_sheet_names() method. Enter the following into the interactive shell:

```
>>> 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">
```

Each sheet is represented by a Worksheet object, which you can obtain by passing the sheet name string to the get_sheet_by_name() workbook method. Finally, you can call the get_active_sheet() method of a Workbook object to get the workbook's active sheet. The active sheet is the sheet that's on top when the workbook is opened in Excel. Once you have the Worksheet object, you can get its name from the title attribute.

Getting Cells from the Sheets

Once you have a Worksheet object, you can access a Cell object by its name.

Enter the following into the interactive shell:

```
>>> 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
```

The Cell object has a value attribute that contains, unsurprisingly, the

value stored in that cell. Cell objects also have row, column, and coordinate attributes

that provide location information for the cell.

Here, accessing the value attribute of our Cell object for cell B1 gives

us the string 'Apples'. The row attribute gives us the integer 1, the column attribute gives us 'B', and the coordinate attribute gives us 'B1'.

OpenPyXL will automatically interpret the dates in column A and return them as datetime values rather than strings. The datetime data type is explained further in Chapter 16.

Specifying a column by letter can be tricky to program, especially because after column Z, the columns start by using two letters: AA, AB, AC, and so on. As an alternative, you can also get a cell using the sheet's cell() method and passing integers for its row and column keyword arguments. The first row or column integer is 1, not 0. Continue the interactive shell example by entering the following:

```
>>> sheet.cell(row=1, column=2)
<Cell Sheet1.B1>
>>> sheet.cell(row=1, column=2).value
'Apples'
>>> for i in range(1, 8, 2):
print(i, sheet.cell(row=i, column=2).value)
1 Apples
3 Pears
5 Apples
7 Strawberries
```

keyword argument. Note that the integer 2, not the string 'B', is passed.

You can determine the size of the sheet with the Worksheet object's get_highest_row() and get_highest_column() methods. Enter the following into the interactive shell:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Sheet1')
>>> sheet.get_highest_row()
7
>>> sheet.get_highest_column()
3
```

v Getting Rows and Columns from the Sheets

You can slice Worksheet objects to get all the Cell objects in a row, column, or rectangular area of the spreadsheet. Then you can loop over all the cells in the slice. Enter the following into the interactive shell:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Sheet1')
>>> tuple(sheet['A1':'C3'])
((<Cell Sheet1.A1>, <Cell Sheet1.B1>, <Cell Sheet1.C1>), (<Cell Sheet1.A2>,
<Cell Sheet1.B2>, <Cell Sheet1.C2>), (<Cell Sheet1.A3>, <Cell Sheet1.B3>,
<Cell Sheet1.C3>))
u >>> for rowOfCellObjects in sheet['A1':'C3']:
v for cellObj in rowOfCellObjects:
```

```
print(cellObj.coordinate, cellObj.value)
print('--- END OF ROW ---')
```

```
A1 2015-04-05 13:34:02
B1 Apples
C1 73
--- END OF ROW ---
A2 2015-04-05 03:41:23
B2 Cherries
C2 85
--- END OF ROW ---
A3 2015-04-06 12:46:51
B3 Pears
C3 14
--- END OF ROW ---
```

To access the values of cells in a particular row or column, you can also use a Worksheet object's rows and columns attribute. Enter the following into the interactive shell:

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_active_sheet()
>>> sheet.columns[1]
(<Cell Sheet1.B1>, <Cell Sheet1.B2>, <Cell Sheet1.B3>, <Cell Sheet1.B4>,
<Cell Sheet1.B5>, <Cell Sheet1.B6>, <Cell Sheet1.B7>)
```

```
>>> for cellObj in sheet.columns[1]:
print(cellObj.value)
Apples
Cherries
Pears
Oranges
Apples
Bananas
Strawberries
```

Workbooks, Sheets, Cells

As a quick review, here's a rundown of all the functions, methods, and data types involved in reading a cell out of a spreadsheet file:

1. Import the openpyxl module.
2. Call the openpyxl.load_workbook() function.
3. Get a Workbook object.
4. Call the get_active_sheet() or get_sheet_by_name() workbook method.
5. Get a Worksheet object.
6. Use indexing or the cell() sheet method with row and column keyword arguments.
7. Get a Cell object.
8. Read the Cell object's value attribute.

9c. Describe the getText() function used for getting full text from a .docx file with example code.

Getting the Full Text from a .docx File

If you care only about the text, not the styling information, in the Word document, you can use the getText() function. It accepts a filename of a .docx file and returns a single string value of its text. Open a new file editor window and enter the following code, saving it as readDocx.py:

```
#! python3
import docx
def getText(filename):
doc = docx.Document(filename)
fullText = []
for para in doc.paragraphs:
fullText.append(para.text)
return '\n'.join(fullText)
```

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. After the loop, the strings in fullText are joined together with newline characters.

The readDocx.py program can be imported like any other module.

Now if you just need the text from a Word document, you can enter the following:

```
>>> 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

10a. 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

Finding an Element with the select() Method

You 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. Selectors are like regular expressions: They specify a pattern to look for, in this case, in HTML pages instead of general text strings.

Table 11-2: Examples of CSS Selectors

Selector passed to the select() method Will match . . .

soup.select('div') All elements named

soup.select('#author') The element with an id attribute of author

soup.select('.notice') All elements that use a CSS class attribute named notice

soup.select('div span') All elements named that are within an element named

soup.select('div > span') All elements named that are

directly within an element named ,

with no other element in between

soup.select('input[name]') All elements named <input> that have a

name attribute with any value

soup.select('input[type="button"]') All elements named <input> that have an

attribute named type with value button

has an id attribute of author, as long as it is also inside a <p> element.

The select() method will return a list of Tag objects, which is how

Beautiful Soup represents an HTML element. The list will contain one

Tag object for every match in the BeautifulSoup object's HTML. Tag values

can be passed to the str() function to show the HTML tags they represent.

Tag values also have an attrs attribute that shows all the HTML attributes

of the tag as a dictionary. Using the example.html file from earlier, enter the

following into the interactive shell:

```
>>> 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'}
```

10b What is JSON? Briefly explain the json module of Python. Demonstrate with a Python program.

The 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. JSON can't store every kind of Python value. It can contain values of only the following data types: 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.

Reading JSON with the loads() Function

To translate a string containing JSON data into a Python value, pass it to the json.loads() function. (The name means "load string," not "loads.")

Enter the following into the interactive shell:

```
>>> stringOfJsonData = '{"name": "Zophie", "isCat": true, "miceCaught": 0,
"felineIQ": null}'
```

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

After you import the json module, you can call loads() and pass it a string of JSON data. Note that JSON strings always use double quotes. It will return that data as a Python dictionary. Python dictionaries are not ordered, so the key-value pairs may appear in a different order when you print jsonDataAsPythonValue.

Writing JSON with the dumps() Function

The json.dumps() function (which means “dump string,” not “dumps”) will translate a Python value into a string of JSON-formatted data. Enter the following into the interactive shell:

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

'{"isCat": true, "felineIQ": null, "miceCaught": 0, "name": "Zophie" }'
```

The value can only be one of the following basic Python data types: dictionary, list, integer, float, string, Boolean, or None.

10c Discuss the Creation, Encryption and Decryption of a PDF

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. Enter the following into the interactive shell with the PDF you downloaded, which has been encrypted with the password rosebud:

```
>>> import PyPDF2
>>> pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf', 'rb'))
u >>> pdfReader.isEncrypted
True

>>> pdfReader.getPage(0)
v Traceback (most recent call last):
File "<pyshell#173>", line 1, in <module>
pdfReader.getPage()
--snip--
File "C:\Python34\lib\site-packages\PyPDF2\pdf.py", line 1173, in getObject
raise utils.PdfReadError("file has not been decrypted")
PyPDF2.utils.PdfReadError: file has not been decrypted
w >>> pdfReader.decrypt('rosebud')
1
>>> pageObj = pdfReader.getPage(0)
```

All PdfFileReader objects have an isEncrypted attribute that is True if the PDF is encrypted and False if it isn't u. Any attempt to call a function that reads the file before it has been decrypted with the correct password will result in an error v.

To read an encrypted PDF, call the decrypt() function and pass the password as a string w. After you call decrypt() with the correct password, you'll see that calling getPage() no longer causes an error. If given the wrong password, the decrypt() function will return 0 and getPage() will continue to fail.

Note that the decrypt() method decrypts only the PdfFileReader object, not the actual PDF file. After your program terminates, the file on your hard drive remains encrypted. Your program will have to call decrypt() again the next time it is run.

Model Paper2 solution guidelines

1a. Write a python program to find the area of square, rectangle and circle. Print the results. Take input from user

```
op = input("For what shape are you trying to find the area of")

if op == "square":
  side = input("side of the square(must be just a number) : ")

  area = int(side) * int(side)

  print(f" the area is: {area}")

elif op == "rectangle":

  length = input("length of the rectangle (must be just a number): ")
```

```
  breadth = input(" breadth of the rectangle:")

  area = int(length) * int(breadth)

  print(f" the area is: {area}")

elif op == "circle":

  r = input("radius of the circle(must be just a number): ")

  r2 = int(r) * 2

  area = 3.14 * int(r2)

  print(f" the area is: {area}")
```

output:

```
For what shape are you trying to find the area of  square
side of the square(must be just a number) : 2
 the area is: 4
```

1b. List and explain the syntax of all flow control statements with example.

Ch2 . refer Flow Control Statements point page 40to 56 approx

1c. Illustrate the use of break and continue with a code snippet.

break Statements page 50 ch2

continue Statements page 50 to 54

draw flow char as well with example

2a. What are user defined functions? How can we pass parameters in user defined functions? Explain with suitable example.

Refer ch3 functions page no 42 onwards

2b. Explain global statement with example.

Ch3 page 70-72

2c. Write a function that computes and returns addition ,subtraction, multiplication , division of two integers.Take input from user.

```
def mathematicalop(a,b):
  addition=a+b
  print("addition is :=",addition)
  sub=a-b
  print("substraction :=",sub)
  mul=a*b
  print("multiplication",mul)
  div=a/b
  print("division ",div)
  return addition,sub,mul,div

l=int(input("enter 1st num"))
m=int(input("enter 2nd num:"))
mathematicalop(l,m)
```

Module2

3a. What is list?Explain the concept of list slicing with example
Refer ch 4 page 80 to 83

3b. Explain references with example
Ch4 page no 97 to 101

3c. What is dictionary?How it is different from list?Write a program to count the number of occurances of character in a string.
Page 106 to 107 ch5.

```
#count the number of occurances of character in a string
s="Ashwini"
d={}
for i in s:
  if i in d:
    d[i]+=1
  else:
    d[i]=1
print(d)
```

output

```
{'A': 1, 's': 1, 'h': 1, 'w': 1, 'i': 2, 'n': 1}
```

4a. 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

Ans

```
def displayInventory(d):

  count=0
  print("inventory")
  for i in d:
    print(str(d[i]) + ' ' + i)
    count += d[i]
  print("Total number of items: " + str(count))

s = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}

displayInventory(s)
```

output

```
inventory
1 rope
6 torch
42 gold coin
1 dagger
12 arrow
Total number of items: 62
```

4b. List any six methods associated with string and explain each of them with example
Refer page 127to 136 ch6

4c Write a Python program to swap cases of a given string.
Input: Java
Output: jAVA

```
#swap the cases of string
def swap_case_string(str1):
   result_str = ""
   for item in str1:
       if item.isupper():
           result_str += item.lower()
       else:
           result_str += item.upper()
   return result_str

print(swap_case_string("Java"))
```

output : jAVA

Module3

5a. List and explain Shorthand code for common character classes .Illustrate how do you define your own character class?

Ch7 page 158

Table 7-1: Shorthand Codes 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.

5b. Explain the usage of Caret and dollar sign characters in regular expression

Page 159 to 160 ch 7

5c. Write a python program to extract phone numbers and email addresses using regex
Program is on 165 to 169 in text book under ch7

6a. How do we specify and handle absolute ,relative paths?
Ch8 page no 177
6b. Explain saving of variables using shelve module
Ch 9 page no 184 and 185

6c. With code snippet, explain reading, extracting and creating ZIP files
Ch10 page no 203 to 205

Module 4
7a What is class? How do we define a class in python? How to instantiate the class and how class members are accessed?
Refer vtu practice questions and ans and given ppts ,IAT solutions

7b. 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.
Refer IAT 3 solution

7c. Explain __init__ and __str__ methods.
Refer IAT3 solutions

8a. Explain operator overloading with example
Refer textbook 2 or module4 notes from gcr
8b. What are polymorphic functions? Explain with code snippet
Refer IAT3 solution
8c Illustrate the concept of inheritance with example
Refer modiule4 ppts of inheritance

Module5:
9a. How do we download a file and save it to harddrive using request module?
Ch 11 page 237 to 239
9b. Write a python program to give search keyword from command line arguments and open the browser tab for each result page.
Page no 248 to 250 program is there fallow all steps

9c. Explain selenium's webdrive methods for finding elements
Ch 11 257 to259

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

You can do your own way also below program.

```
import openpyxl, sys

from openpyxl.styles import Font

times = int(sys.argv[1])
wb = openpyxl.Workbook()
sheet = wb['Sheet']
boldFont = Font(bold=True)

```

```
for i in range(1, times + 1):
      sheet.cell(row=i + 1, column=1).value = i
      sheet.cell(row=i + 1, column=1).font = boldFont
      sheet.cell(row=1, column=i+1).value = i
      sheet.cell(row=1, column=i+1).font = boldFont

for i in range(2, times + 2):
      for j in range(2, times + 2):
            x = sheet.cell(row=i, column=1).value
            y = sheet.cell(row=1, column=j).value
            sheet.cell(row=i, column=j).value = x * y

wb.save('multiplicationTable.xlsx')
```

10b. Write short notes on
i)Creating,copying and rotating pages with respect to pdf
ch 13 .page no 299 to 301

10c. 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 CSV file
Ch 14 page no 324 to 327