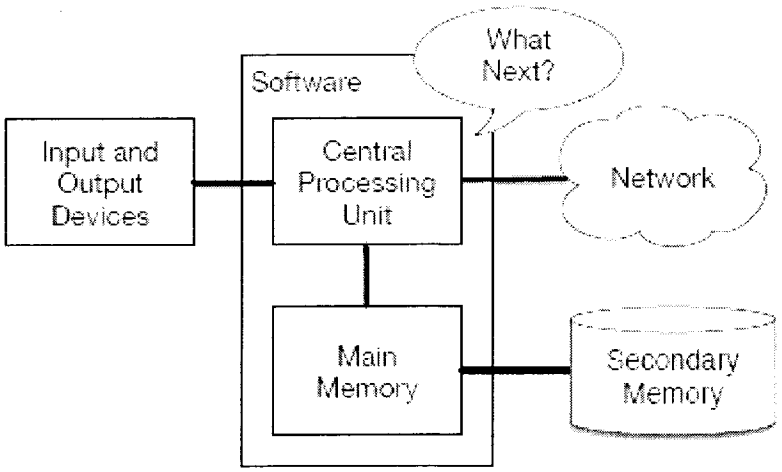
Python Application Programming

1a. Explain computer hardware Architecture with neat diagram



- The *Central Processing Unit* (or CPU) is the part of the computer that is built to be obsessed with "what is next?
- The *Main Memory* is used to store information that the CPU needs in a hurry. The main memory is nearly as fast as the CPU. But the information stored in the main memory vanishes when the computer is turned off.
- The *Secondary Memory* is also used to store information, but it is much slower than the main memory. The advantage of the secondary memory is that it can store information even when there is no power to the computer. Examples of secondary memory are disk drives or flash memory (typically found in USB sticks and portable music players).
- The *Input and Output Devices* are simply our screen, keyboard, mouse, microphone, speaker, touchpad, etc. They are all of the ways we interact with the computer.
- These days, most computers also have a *Network Connection* to retrieve information over a network. We can think of the network as a very slow place to store and retrieve data that might not always be "up". So in a sense, the network is a slower and at times unreliable form of *Secondary Memory*.

1b. Define High Level Language and Machine level Language. List out the difference between compiler and interpreter?

- A high-level language (HLL) is a programming language such as C, FORTRAN, Python or Pascal that enables a programmer to write programs that are more or less independent of a particular type of

computer. Such languages are considered high-level because they are closer to human languagesand further from machine languages.

- Programs written in high-level languages are translated into assembly language or machine language by a compiler
- An *interpreter* reads the source code of the program as written by the programmer, parses the source code, and interprets the instructions on the fly. Python is an interpreter and when we are running Python interactively, we can type a line of Python (a sentence) and Python processes it immediately and is ready for us to type another line of Python.
- A *compiler* needs to be handed the entire program in a file, and then it runs a process to translate the high-level source code into machine language and then the compiler puts the resulting machine language into a file for later execution.
- *inter-preters* and *compilers* that allow us to write in high-level languages like Python or C.

1c. write a function called is_palindrome that takes a string argument and returns True if it is a Palindrome and False otherwise. Use built-in function to check the length of a string. Prompt the user for input.

```
# function which return reverse of a string
def reverse(s):
    return s[::-1]
def isPalindrome(s):
    rev = reverse(s)
    # Checking if both string are equal or not
    if (s == rev):
        return True
    return False
# Driver code
s = raw_input("enter the string: ")
ans = isPalindrome(s)
if ans == 1:
    print("Yes")
else:
    print("No")
```

2a. Explain the concept of short circuit evaluation of logical expression in Python. Write a program to prompt for a score between 0.0 & 1.0. if the score is out of range,print an error message. If the score is between 0.0 and 1.0, print a grade using the following table

Score	grade
>=0.9	A
>=0.8	B
>=0.7	C
>=0.6	D
<0.6	F

- There are three logical operators: and, or, and not. The semantics (meaning) of these operators is similar to their meaning in English.

  example,

  x > 0 and x < 10

  is true only if x is greater than 0 and less than 10.

```
prompt1 = 'Please enter a score between 0.0 and 1.0?\n'
try:
 score = input (prompt1)
 score = float(score)
 if score <=1.0:
  if 0.9 <= score <= 1.0:
    print "Your grade is an A"
elif score >= 0.8:
    print "Your grade is a B"
elif score >= 0.7:
    print "Your grade is a C"
elif score >= 0.6:
    print "Your grade is a D"
elif score <  0.6:
    print "Your grade is an F"
 else:
  print ('Error, score cannot be greater than 1.0')
except:
  print ('Error, please enter a number')
```

2b. Explain in detail the building block of a program. State the need for function in Python?

- **INPUT** Get data from the "outside world". This might be reading data from a file, or even some kind of sensor like a microphone or GPS. In our initial programs, our input will come from the user typing data on the keyboard.
- **OUTPUT** Display the results of the program on a screen or store them in a file or perhaps write them to a device like a speaker to play music or speak text.
- **SEQUENTIAL EXECUTION** Perform statements one after another in the order they are encountered in the script.
- **CONDITIONAL EXECUTION** Check for certain conditions and then execute or skip a sequence of statements.
- **REPEATED EXECUTION** Perform some set of statements repeatedly, usually with some variation.
- **REUSE** Write a set of instructions once and give them a name and then reuse those instructions as needed throughout your program.
- In the context of programming, a function is a named sequence of statements that performs a computation.
- It is common to say that a function "takes" an argument and "returns" a result. The result is called the return value.

2c. Explain Syntax errors and Logic errors. Write a program which prompts the user for a Celsius temperature, convert the temperature to Fahrenheit and print out the converted temperature.

Syntax errors These are the first errors you will make and the easiest to fix. A syntax error means that you have violated the "grammar" rules of Python. Python does its best to point right at the line and character where it noticed it was confused. The only tricky bit of syntax errors is that sometimes the mistake that needs fixing is actually earlier in the program than where Python noticed it was confused. So the line and character that Python indicates in a syntax error may just be a starting point for your investigation.

Logic errors A logic error is when your program has good syntax but there is a mistake in the order of the statements or perhaps a mistake in how the statements relate to one another. A good example of a logic error might be, "take a drink from your water bottle, put it in your backpack, walk to the library, and then put the top back on the bottle."

Print('Celsius to Fahrenheit Conversion')

```
celsius = float(raw_input('Celsius: '))
fahrenheit = (celsius * 9 / 5) + 32
print ('Fahrenheit: ' + str(fahrenheit))
```

3a. Explain break and Continue statement with examples in Python. Write Pythonic code that iteratively prompts the user for input. It should continue until the user enters 'done' and then return the average value.

- Python break statement. It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C. The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both while and for loops.
- Python continue statement. It returns the control to the beginning of the while loop.. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop. The continue statement can be used in both while and for loops.

```
sum = 0
count = 0
average = 0
while True:
 try:
   x = input("Enter a number: ")
   if (x == "done"):
    break
   value = float(x)
   sum = value + sum
 count = count + 1
   average = sum / count
 except:
print("Invalid input.")
print (sum, count, average)
```

3b. Write a Python program to check whether a number is Prime or not using while loop and print appropriate messages.

```
import math
print ("Enter the a number")
number = int(input())
```

```
i = 2
prime = True

#if the number is not divisible by any number less than the square root of the number
#then it is prime
while i <= int(math.sqrt(number)):
    if number%i == 0:
        prime = False
        break
    i = i+1
if number < 2:
    prime = False
if prime:
    print (number,"is a prime number")
else:
    print (number,"is not a prime number")
```

3c. "Strings in python are Immutable". Explain this statement with example. Write pythonic code to find the factorial of any number entered through the keyboard.

It is tempting to use the operator on the left side of an assignment, with the intention of changing a character in a string. For example:

```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: 'str' object does not support item assignment
```

The "object" in this case is the string and the "item" is the character you tried to assign. For now, an object is the same thing as a value, but we will refine that definition later. An item is one of the values in a sequence.

The reason for the error is that strings are immutable, which means you can't change an existing string. The best you can do is create a new string that is a variation on the original:

```
>>> greeting = 'Hello, world!'
```

```
>>> new_greeting = 'J' + greeting[1:]
>>> print(new_greeting) Jello, world!
# Python program to find the factorial of a number provided by the user.
# change the value for a different result
#num = 7
# uncomment to take input from the user
num = int(input("Enter a number: "))
factorial = 1
# check if the number is negative, positive or zero
if num < 0:
  print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
  print("The factorial of 0 is 1")
else:
  for i in range(1,num + 1):
    factorial = factorial*i
  print("The factorial of",num,"is",factorial)
```

4a. Write a Python program to read the file and count and print the lines that start with word From. Prompt the user for the file name. Also use try/except to handle bad file names. Explain format operator with example in Python.

```
fname = input('Enter the file name: ')
try:
   fhand = open(fname)
except:
       print('File cannot be opened:', fname)
       exit() count = 0
for line in fhand:
        if line.startswith('From:'):
            count = count + 1
```

print('There were', count, 'From lines in', fname)

The format operator, % allows us to construct strings, replacing parts of the strings with the data stored in variables. When applied to integers, % is the modulus operator. But when the first operand is a string, % is the format operator.

The first operand is the format string, which contains one or more format sequences that specify how the second operand is formatted. The result is a string.

For example, the format sequence "%d" means that the second operand should be formatted as an integer (d stands for "decimal"):

>>> camels = 42

>>> '%d' % camels '42'

The result is the string "42", which is not to be confused with the integer value 42.

A format sequence can appear anywhere in the string, so you can embed a value in a sentence:

>>> camels = 42

>>> 'I have spotted %d camels.' % camels 'I have spotted 42 camels.'

If there is more than one format sequence in the string, the second argument has to be a tuple1. Each format sequence is matched with an element of the tuple, in order.

The following example uses "%d" to format an integer, "%g" to format a floatingpoint number (don't ask why), and "%s" to format a string:

>>> 'In %d years I have spotted %g %s.' % (3, 0.1, 'camels') 'In 3 years I have spotted 0.1 camels.'

The number of elements in the tuple must match the number of format sequences in the string. The types of the elements also must match the format sequences:

>>> '%d %d %d' % (1, 2) TypeError: not enough arguments for format string

>>> '%d' % 'dollars' TypeError: %d format: a number is required, not str

4b. Write Pythonic code to multiply two matrices using nested loops and print the result.

```
# Program to multiply two matrices using nested loops
# take a 3x3 matrix
A = [[12, 7, 3],
    [4, 5, 6],
    [7, 8, 9]]
```

```
# take a 3x4 matrix
B = [[5, 8, 1, 2],
     [6, 7, 3, 0],
     [4, 5, 9, 1]]

result = [[0, 0, 0, 0],
          [0, 0, 0, 0],
          [0, 0, 0, 0]]
# iterating by row of A
for i in range(len(A)):
    # iterating by coloum by B
    for j in range(len(B[0])):

        # iterating by rows of B
        for k in range(len(B)):
            result[i][j] += A[i][k] * B[k][j]
for r in result:
    print(r)
```

4c. Write Pythonic code to count and print the occurrence of each of the word in the file using dictionaries. Prompt the user for the file name. Also use try/except to handle bad file names.

```
fname = input('Enter the file name: ')
try:
   fhand = open(fname)
except:
    print('File cannot be opened:', fname)
exit()

counts = dict()
for line in fhand:
    words = line.split()
     for word in words:
         if word not in counts:
           counts[word] = 1
        else:
           counts[word] += 1
print(counts)
```

output:

python count1.py

Enter the file name: romeo.txt

{'and': 3, 'envious': 1, 'already': 1, 'fair': 1, 'is': 3, 'through': 1, 'pale': 1, 'yonder': 1, 'what': 1, 'sun': 2, 'Who': 1, 'But': 1, 'moon': 1, 'window': 1, 'sick': 1, 'east': 1, 'breaks': 1, 'grief': 1, 'with': 1, 'light': 1, 'It': 1, 'Arise': 1, 'kill': 1, 'the': 3, 'soft': 1, 'Juliet': 1}

5a. Write Pythonic code that implements and return the functionality of histogram using dictionaries. Also, write the function print_hist to print the keys and their values in alphabetical order from the values returned by the histogram function.

The name of the function is histogram, which is a statistical term for a set of counters (or frequencies)

```
>>> L = 'abracadabra'
>>> histogram(L)
{'a': 5, 'b': 2, 'c': 1, 'd': 1, 'r': 2}
```

```
def histogram(L):
    d = {}
    for x in L:
        if x in d:
            d[x] += 1
        else:
            d[x] = 1
    return d
```

5b. Explain join(), split() and append() methods in a list with examples. Write pythonic code to input information about 20 students as given below:

1. **Roll number.**
2. **Name.**
3. **Total marks.**

Get the input from the user for student name. The program should display the roll num and total marks for the given student name. Also find the average marks of all the students. Use dictionaries.

join(),

join is the inverse of split. It takes a list of strings and concatenates the elements. join is a string method, so you have to invoke it on the delimiter and pass the list as a parameter:

```
>>> t = ['pining', 'for', 'the', 'fjords']
>>> delimiter = ' '
>>> delimiter.join(t)
```

'pining for the fjords'

In this case the delimiter is a space character, so join puts a space between words. To concatenate strings without spaces, you can use the empty string, "", as a delimiter.

split()

The list function breaks a string into individual letters. If you want to break a string into words, you can use the split method:

```
>>> s = 'pining for the fjords'
>>> t = s.split()
>>> print(t) ['pining', 'for', 'the', 'fjords']
>>> print(t[2])
```

The

Once you have used split to break the string into a list of words, you can use the index operator (square bracket) to look at a particular word in the list.

Append()

Python provides methods that operate on lists. For example, append adds a new element to the end of a list:

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print(t)
['a', 'b', 'c', 'd']
```

```
n = int(raw_input("Please enter number of students:"))
student_data = ['stud_name', 'stud_rollno', 'mark1', 'mark2','mark3','total', 'average']
for i in range(0,n):
    stud_name=raw_input('Enter the name of student: ')
    print stud_name
    stud_rollno=input('Enter the roll number of student: ')
    print stud_rollno
    mark1=input('Enter the marks in subject 1: ')
    print mark1
    mark2=input('Enter the marks in subject 2: ')
```

```
print mark2

mark3=input('Enter the marks in subject 3: ')
print mark3
total=(mark1+mark2+mark3)
print"Total is: ", total
average=total/3
print "Average is :", average

dict = {'Name': stud_name, 'Rollno':stud_rollno, 'Mark1':mark1, 'Mark2':mark2,'Mark3':mark3,
'Total':total, 'Average':average}
print "dict['Name']: ", dict['Name']
print "dict['Rollno']: ", dict['Rollno']
print "dict['Mark1']: ", dict['Mark1']
print "dict['Mark2']: ", dict['Mark2']
print "dict['Mark3']: ", dict['Mark3']
print "dict['Total']: ", dict['Total']
print "dict['Average']: ", dict['Average']
```

6a. Define tuple. Explain DSU pattern. Write Pythonic code to demonstrate tuples by sorting a list of words from longest to shortest using loops.

A tuple1 is a sequence of values much like a list. The values stored in a tuple can be any type, and they are indexed by integers. The important difference is that tuples are immutable. Tuples are also comparable and hashable so we can sort lists of them and use tuples as key values in Python dictionaries.

```
txt = 'but soft what light in yonder window breaks'
words = txt.split()
t = list() for word in words:
 t.append((len(word), word))
t.sort(reverse=True)
res = list() for length, word in t: res.append(word)
```

```
print(res)
output:
['yonder', 'window', 'breaks', 'light', 'what', 'soft', 'but', 'in']
```

6b. Why do you need regular expression in python? Consider a line "From" Stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008" in the file email.txt. Write pythonic code to read the file and extract email address from the lines starting from the word "From". Use regular expressions to match email address.

This task of searching and extracting is so common that Python has a very powerful library called regular expressions that handles many of these tasks quite elegantly.

Regular expressions are almost their own little programming language for searching and parsing strings. As a matter of fact, entire books have been written on the topic of regular expressions.

The regular expression library re must be imported into your program before you can use it. The simplest use of the regular expression library is the search() function.

```
import re
hand = open('email.txt')
for line in hand:
    line = line.rstrip()
 if re.search('From:', line):
    print(line)
```

7a. What is operator overloading? Write Pythonic code to overload "+","-",and "*" operator by providing the methods _add_,_sub_,and _mul_.

Solution:

By defining other special methods, you can specify the behavior of operators on p # #inside class Time:

```
def __add__(self, other):
    seconds = self.time_to_int() + other.time_to_int()
    return int_to_time(seconds)
```

And here is how you could use it:

>>> start = Time(9, 45)

>>> duration = Time(1, 35)

>>> print(start + duration)

11:20:00programmer-defined types.

When you apply the + operator to Time objects, Python invokes __add__. When you print the result, Python invokes __str__.

So there is a lot happening behind the scenes! Changing the behavior of an operator so that it works with programmer-defined types is called operator overloading.

For every operator in Python there is a corresponding special method, like __add__.

7b. Consider a user defined class called Time that records the time of the day. Create a new time object and assign attributes for hours, minute and seconds. Write a function called print_time that takes a Time object and prints it in the form hour:minute:second. Write a Boolean function called is_after that takes two time object, t1 and t2, and returns True if t1 follows t2 chronologically and False otherwise. Write a function called increment which adds a given number of seconds to a Time object.

```
def add_time(t1, t2):
    sum = Time()
    sum.hours = t1.hours + t2.hours
    sum.minutes = t1.minutes + t2.minutes
    sum.seconds = t1.seconds + t2.seconds

    if sum.seconds >= 60:
        sum.seconds = sum.seconds - 60
        sum.minutes = sum.minutes + 1

    if sum.minutes >= 60:
        sum.minutes = sum.minutes - 60
        sum.hours = sum.hours + 1

    return sum
```

8a. Write pythonic code to create a function named move_rectangle that takes an object Rectangle and two number named dx and dy. It should change the location of the rectangle by adding dx to the x coordinate of corner and adding dy to the y coordinate of corner.

```
# Write a function named move_rectangle that takes a Rectangle and two numbers
# named dx and dy. It should change the location of the rectangle by adding
# dx to the x coordinate of corner and adding dy to the y coordinate of corner.
# Current Status: Complete
class Point(object):
    """Represents a point in 2d space"""
class Rectangle(object):
    """Represents a rectangle in 2d space"""
rectangle = Rectangle()
bottom_left = Point()
bottom_left.x = 3.0
bottom_left.y = 5.0

top_right = Point()
top_right.x = 5.0
top_right.y = 10.0
rectangle.corner1 = bottom_left
rectangle.corner2 = top_right
dx = 5.0
dy = 12.0
def move_rectangle(rectangle, dx, dy):
    """Takes a rectangle and moves it to the values of dx and dy."""
    print ("The rectangle started with bottom left corner at (%g,%g)"
        % (rectangle.corner1.x, rectangle.corner1.y)),
    print ("and top right corner at (%g,%g)."
        % (rectangle.corner2.x, rectangle.corner2.y)),
    print "dx is %g and dy is %g" % (dx, dy)
    rectangle.corner1.x = rectangle.corner1.x + dx
    rectangle.corner2.x = rectangle.corner2.x + dx
    rectangle.corner1.y = rectangle.corner1.y + dy
    rectangle.corner2.y = rectangle.corner2.y + dy
```

```
print ("It ended with a bottom left corner at (%g,%g)"
    % (rectangle.corner1.x, rectangle.corner1.y)),
print ("and a top right corner at (%g,%g)"
    % (rectangle.corner2.x, rectangle.corner2.y))
move_rectangle(rectangle, dx, dy)
```

8b. Explain Polymorphism in python in detail with example

Type-based dispatch is useful when it is necessary, but (fortunately) it is not always necessary. Often you can avoid it by writing functions that work correctly for arguments with different types. Many of the functions we wrote for strings also work for other sequence types.

For example, we used histogram to count the number of times each letter appears in a word.

```
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}
```

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

For example, the built-in function sum, which adds the elements of a sequence, works as long as the elements of the sequence support addition. Since Time objects provide an add method, they work with sum:

```
>>> t1 = Time(7, 43)
>>> t2 = Time(7, 41)
 >>> t3 = Time(7, 37)
>>> total = sum([t1, t2, t3])
```

>>> print(total) 23:01:00

In general, if all of the operations inside a function work with a given type, the function works with that type. The best kind of polymorphism is the unintentional kind, where you discover that a function you already wrote can be applied to a type you never planned for.

8 a. Define Socket. Write a python program to retrieve an image over HTTP.

The network protocol that powers the web is actually quite simple and there is builtin support in Python called sockets which makes it very easy to make network connections and retrieve data over those sockets in a Python program.

A socket is much like a file, except that a single socket provides a two-way connection between two programs. You can both read from and write to the same socket. If you write something to a socket, it is sent to the application at the other end of the socket. If you read from the socket, you are given the data which the other application has sent.

But if you try to read a socket when the program on the other end of the socket has not sent any data, you just sit and wait. If the programs on both ends of the socket simply wait for some data without sending anything, they will wait for a very long time.

So an important part of programs that communicate over the Internet is to have some sort of protocol. A protocol is a set of precise rules that determine who is to go first, what they are to do, and then what the responses are to that message, and who sends next, and so on. In a sense the two applications at either end of the socket are doing a dance and making sure not to step on each other's toes.

There are many documents which describe these network protocols. The HyperText Transport Protocol is described in the following document:

```
import socket
import time
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('www.py4inf.com', 80))
mysock.send('GET http://data.pr4e.org/cover.jpg HTTP/1.0\n\n')
count = 0
picture = "";
while True:
  data = mysock.recv(5120)
```

```
    if ( len(data) < 1 ) : break
    # time.sleep(0.25)
    count = count + len(data)
    print len(data),count
    picture = picture + data
mysock.close()
# Look for the end of the header (2 CRLF)
pos = picture.find("\r\n\r\n");
print 'Header length',pos
print picture[:pos]
# Skip past the header and save the picture data
picture = picture[pos+4:]
fhand = open("stuff.jpg","wb")
fhand.write(picture);
fhand.close()
```

9b. Write a python program that makes a connection to a web server requesting for a document and display what the server sends back. Your python program should follow the rules of the HTTP protocol. List the common header which the webserver sends to describe the document.

```
import socket
mysock=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org',80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode()
mysock.send(cmd)
while
True: data = mysock.recv(20)
if (len(data) < 1):
break print(data.decode(),end='')
mysock.close()
```

10a. State the need of urllib in python. Write pythonic code to retrieve the file "vtu.txt" by using the URL http://vtu.as.in/code/vtu.txt. also compute the frequency of each of the word in the retrived file.

While we can manually send and receive data over HTTP using the socket library, there is a much simpler way to perform this common task in Python by using the urllib library.
Using urllib, you can treat a web page much like a file. You simply indicate which web page you would like to retrieve and urllib handles all of the HTTP protocol and header details.

```
import urllib.request, urllib.parse, urllib.error
fhand = urllib.request.urlopen('http://vtu.as.in/code/vtu.txt')
counts = dict()
for line in fhand:
words = line.decode().split()
for word in words:
counts[word] = counts.get(word, 0) + 1
print(counts)
```

10b. Give an example to construct a simple web page using HTML. Write pythonic code to match and extract the various links found in a webpage using urllib

```
'''A simple program to create an html file froma given.string,
and call the default web browser to display the file.'''

contents = '''<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta content="text/html; charset=ISO-8859-1"
 http-equiv="content-type">
  <title>Hello</title>
</head>
<body>
Hello, World!
</body>
</html>
'''

def main():
```

```
    browseLocal(contents)

def strToFile(text, filename):
    """Write a file with the given name and the given text."""
    output = open(filename,"w")
    output.write(text)
    output.close()

def browseLocal(webpageText, filename='tempBrowseLocal.html'):
    '''Start your webbrowser on a local file containing the text
    with given filename.'''
    import webbrowser, os.path
    strToFile(webpageText, filename)
    webbrowser.open("file:///" + os.path.abspath(filename)) #elaborated for
Mac

main()
```

```
import BeautifulSoup, urllib2

address='http://www.419scam.org/emails/'
html = urllib2.urlopen(address).read()
f = open('test.txt', 'wb')
f.write(html)
f.close()
```