Module 1

Why should you learn to write programs

Exercise 1: What is the function of the secondary memory in a computer?
a) Execute all of the computation and logic of the program
b) Retrieve web pages over the Internet
c) Store information for the long term, even beyond a power cycle
d) Take input from the user

Ans: c) Store information for the long term, even beyond a power cycle

Exercise 2: What is a program?
Ans: A set of instructions that specifies a computation

Exercise 3: What is the difference between a compiler and an interpreter?
Ans: It is the nature of an *interpreter* to be able to have an interactive conversation and does code generation instantaneously. 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.

Exercise 4: Which of the following contains “machine code”?
a) The Python interpreter
b) The keyboard
c) Python source file
d) A word processing document

Ans: a) The Python interpreter

Exercise 5: What is wrong with the following code:

```
>>> primt 'Hello world!'
File "<stdin>", line 1
primt 'Hello world!'
^
SyntaxError: invalid syntax
>>>
```

Ans: Wrong function name, further in Python 3 onwards it needs parenthesis for function

Exercise 6: Where in the computer is a variable such as “x” stored after the following Python line finishes?

```
x = 123
```

a) Central processing unit
b) Main Memory
c) Secondary Memory
d) Input Devices

e) Output Devices

Ans: b) Main Memory

Exercise 7: What will the following program print out:

```
x = 43
x = x + 1
print(x)
```

a) 43
b) 44
c) x + 1
d) Error because x = x + 1 is not possible mathematically
Ans: c) 44

Exercise 8: Explain each of the following using an example of a human capability: (1) Central processing unit, (2) Main Memory, (3) Secondary Memory, (4) Input Device, and (5) Output Device. For example, "What is the human equivalent to a Central Processing Unit"?

Ans: As per human capability, (1) CPU does job of brain (2) Main Memory – human memory (3) Secondary memory- writing into a notebook/external device (4) Input – speech (5) Output Device – ears

Exercise 9: How do you fix a "Syntax Error"?

Ans: Going back to the file opened in an editor, checking line number shown by interpreter and fix as per the error like mismatch of quotes, wrong function prototype, missing modules etc.

Variables, expressions, and Statements

Exercise 2: Write a program that uses input to prompt a user for their name and then welcomes them.

```
Enter your name: Chuck
Hello Chuck
```

```
name = input('Enter your name: ')
print('Hello', name)
```

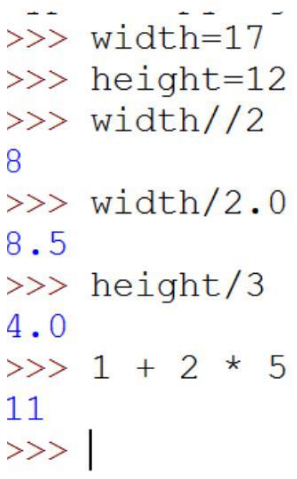
Exercise 3: Write a program to prompt the user for hours and rate per hour to compute gross pay.

```
Enter Hours: 35
Enter Rate: 2.75
```

Pay: 96.25

```
inp = input('Enter Hours: ')
hours = float(inp)
inp = input('Enter Rate: ')
rate = float(inp)
pay = hours * rate
print('Pay:', pay)
```

Exercise 4: Assume that we execute the following assignment statements:
width = 17
height = 12.0
For each of the following expressions, write the value of the expression and the type (of the value of the expression).
1. width//2
2. width/2.0
3. height/3
4. 1 + 2 * 5
Use the Python interpreter to check your answers.



```
>>> width=17
>>> height=12
>>> width//2
8
>>> width/2.0
8.5
>>> height/3
4.0
>>> 1 + 2 * 5
11
>>> |
```

Exercise 5: Write a program which prompts the user for a Celsius temperature, convert the temperature to Fahrenheit, and print out the converted temperature.

```
inp = input('Enter Celsius Temperature:')
cel = float(inp)
fahr = (cel * 9.0) / 5.0 + 32.0
print(fahr)
```

Conditional execution

Write a Python script to check an input number is positive or negative

```
num=int(input('Enter the number'))
if num==0:
    print("0 is neither positive nor negative")
elif num > 0:
    print('positive')
else:
    print('negative')
```

Write a Python script to check an input number is odd or even

```
num=int(input('Enter a number'))
if num%2==0:
    print('Even')
else:
   print('Odd')
```

Illustrate try-catch with a simple code snippet:

```
try:
    inp=int(input('Enter a number'))
    print(inp)
except:
    print('error')
```

Write a robust Python script that illustrates comparing three numbers using chained conditionals

```
try:
    x=int(input('Enter x'))
    y=int(input('Enter y'))
    if x==y:
        print('equal')
    elif x<y:
        print('Less')
    else:
        print('greater')
except:
    print('wrong conversion')
```

Python script to check single digit positive number

```
num=int(input('Enter a number'))
if 0<=num<10:
    print('true')
else:
    print('false')
```

Python script to check input day is weekday or weekend

```
day=input('Enter the day')
if day=='saturday' or day=='sunday':
    print('weekend')
else:
    print('weekday')
```

Exercise 1: Rewrite your pay computation to give the employee 1.5 times the hourly rate for hours worked above 40 hours.

```
Enter Hours: 45
Enter Rate: 10
Pay: 475.0
```

```
inp = input('Enter Hours: ')
hours = float(inp)
inp = input('Enter Rate: ')
rate = float(inp)
if hours > 40:
    pay = hours * rate + (hours - 40) * rate *
0.5 else:
    pay = hours * rate
print('Pay:', pay)
```

Exercise 2: Rewrite your pay program using try and except so that your program handles non-numeric input gracefully by printing a message and exiting the program. The following shows two executions of the program:

```
Enter Hours: 20
Enter Rate: nine
Error, please enter numeric input
Enter Hours: forty
Error, please enter numeric input
```

```
try:
    inp = input('Enter Hours: ')
    hours = float(inp)
    inp = input('Enter Rate: ')
    rate = float(inp)
    if hours > 40:
        pay = hours * rate + (hours - 40) * rate * 1.5
    else:
        pay = hours * rate
    print('Pay:', pay)
except:
    print('Error, please enter numeric input')
```

Exercise 3: Write a program to prompt for a score between 0.0 and 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

```
Enter score: 0.95
A
Enter score: perfect
Bad score
Enter score: 10.0
Bad score
Enter score: 0.75
C
Enter score: 0.5
F
```

Run the program repeatedly as shown above to test the various different values for input.

```
inp = input('Enter score: ')
try:
    score = float(inp)
except:
    score = -1

if score > 1.0 or score < 0.0:
    print('Bad score')
elif score > 0.9:
    print('A')
elif score > 0.8:
    print('B')
elif score > 0.7:
    print('C')
elif score > 0.6:
    print('D')
else:
    print('F')
```

Functions

Exercise 4: What is the purpose of the "def" keyword in Python?
a) It is slang that means "the following code is really cool"
b) It indicates the start of a function
c) It indicates that the following indented section of code is to be stored for later
d) b and c are both true
e) None of the above

Ans: **b)**

Exercise 5: What will the following Python program print out?

```
def fred():
print("Zap")
def jane():
print("ABC")
jane()
fred()
jane()
```

a) Zap ABC jane fred jane
b) Zap ABC Zap
c) ABC Zap jane
d) ABC Zap ABC
e) Zap Zap Zap

Ans: d)
ABC
Zap
ABC

Python script to convert degree to radians

```
import math
def deg2rad(x):
   return(x*(math.pi/180))

print(math.sin(deg2rad(180)),math.cos(deg2rad(180)))
```

Python script using recursion for computing factorial

```
def factorial(x):
   if x==1:
      return 1
   return x*factorial(x-1)
def scan_input():
   return int(input('Enter a number'))
def display(fact):
   print(fact)
```

```
num=scan_input()
fact=factorial(num)
display(fact)
```

Python script illustrating pseudo random number functions:

```
import random
print(random.random())
print(random.random())
x=random.randint(1,6)
print(x)
x=random.randint(1,6)
print(x)

x=random.randint(1,6)
print(x)

x=random.randint(1,6)
print(x)
t=[1,2,4,9]
print(random.choice(t))
```

Exercise 6: Rewrite your pay computation with time-and-a-half for overtime and create a function called computepay which takes two parameters (hours and rate).

```
Enter Hours: 45
Enter Rate: 10
Pay: 475.0
```

```
def computepay(hours,rate):
  try:
    if hours > 40:
      pay = hours * rate + (hours - 40) * rate * 1.5
    else:
      pay = hours * rate
    print('Pay:', pay)
  except:
    print('Error, please enter numeric input')

inp = input('Enter Hours: ')
hours = float(inp)
inp = input('Enter Rate: ')
rate = float(inp)
computepay(hours,rate)
```

Exercise 7: Rewrite the grade program from the previous chapter using a function called computegrade that takes a score as its parameter and returns a grade as a string.

Score Grade
> 0.9 A
> 0.8 B
> 0.7 C
> 0.6 D
<= 0.6 F
Program Execution:

```
Enter score: 0.95
A
Enter score: perfect
Bad score
Enter score: 10.0
Bad score
Enter score: 0.75
C
Enter score: 0.5
F
```

Run the program repeatedly to test the various different values for input.

```
def computegrade(inp):
    try:
        score = float(inp)
    except:
        score = -1

    if score > 1.0 or score < 0.0:
        print('Bad score')
    elif score > 0.9:
        print('A')
    elif score > 0.8:
        print('B')
    elif score > 0.7:
        print('C')
    elif score > 0.6:
        print('D')
    else:
        print('F')

inp = input('Enter score: ')
computegrade(inp)
```

Module 2

Iteration

Python script to simulate time bomb using while loop

```
import time
n=5
while n>=0:
   print(n)
   time.sleep(2)
   n=n-1
print('Blast..')
```

Python script to illustrate infinite loops:

```
while True:
   qn=input('Name')
   if qn=='sar':
      break
   if qn=='pass':
      continue
   print('keep trying')
```

Python program for implementing patter of counter, accumulator, max and min

```
seq=[-3,-5,-1,-6,-7,8.9]
cnt=0
sum=0
max=None
min=None
for i in seq:
   cnt=cnt+1
   sum+=i
   if max is None or i>max:
      max=i
   if min is None or i <min:
      min=i
print(cnt,sum,max,min)
```

Exercise 1: Write a program which repeatedly reads numbers until the user enters "done". Once "done" is entered, print out the total, count, and average of the numbers. If the user enters anything other than a number, detect their mistake using try and except and print an error message and skip to the next number. Enter a number: 4
Enter a number: 5
Enter a number: bad data
Invalid input
Enter a number: 7
Enter a number: done
16 3 5.333333333333333

```
total = 0
count = 0
while (True):
   inp = input('Enter a number: ')
   if inp == 'done':
       break
   try:
       value = float(inp)
   except:
       print('Invalid input')
       continue
   total = total + value
   count = count + 1

average = total / count
print('Average:', average)
```

Exercise 2: Write another program that prompts for a list of numbers as above and at the end prints out both the maximum and minimum of the numbers instead of the average.

```
max='A'
min='A'
while (True):
   inp = input('Enter a number: ')
   if inp == 'done':
       break
   try:
       value = float(inp)
       if max=='A':
           max=value
       else:
           if value>max:
            max=value
      if min=='A':
         min=value
      else:
         if value<min:
            min=value

   except:
      print('Invalid input')
      continue


print('Max',max,' Min:', min)
```

Strings

Python script to demonstrate string traversal using positive indexing

```
str='banana apple'
index=0
while index >=-len(str):
    print(str[index])
    index=index+1
```

Python script to demonstrate string traversal using negative indexing

```
str='banana apple'
index=-len(str)
while index <=-1:
    print(str[index],end='')
    index=index+1
```

Python script to demonstrate string traversal using for loop

```
str='banana apple'
for c in str:
   print(c)
```

Python script to find reverse of a string using positive indexing

```
str='banana apple'
index=len(str)-1
while index >=0:
    print(str[index])
    index=index-1
```

Python script to find reverse of a string using negative indexing

```
str='banana apple'
index=-1
while index >=-len(str):
      print(str[index])
      index=index-1
```

Python script to find reverse of a string using for loop

```
str='banana apple'
rev=''
for c in str:
    rev=c+rev
print(rev)
```

Python script to count number of 'a's in an input string

```
str=input('Enter a string')
count=0
for c in str:
    if(c=='a'):
        count+=1
print(count)
```

Python script illustrating string functions of Python

```
animal=input('Who is the forest king')
animal=animal.strip()
animal=animal.lower()
if animal.startswith('lion'):
    print('Yes')
else:
    print('improve ur GK')
```

Python script to illustrating parsing details from a given string

```
details='Name=Chetan K R, ID=10080, PhoneNo=9900923050,
Email=chetankr@jnnce.ac.in'
s1=details.find('Name=')
e1=details.find(',')
name=details[s1+len('Name='):e1]
print(name)

s1=details.find('ID=')
print(s1)
e1=details.find(',',e1+1)
ID=details[s1+len('ID='):e1]
print(ID)
```

```
s1=details.find('Email=')
print(s1)
e1=details.find(',',e1+1)
Email=details[s1+len('Email='):]
print(Email)
```

Python script to extract number of oranges and apples from a given string

```
str="There are 12:apples and 2:oranges"
s=str.find('and ')
s=s+len('and ')
e=str.find(':',s)
print(str[s:e])
```

Exercise 5: Take the following Python code that stores a string:‘

str = ’X-DSPAM-Confidence:**0.8475**’

Use find and string slicing to extract the portion of the string after the colon character and then use the float function to convert the extracted string into a floating point number.

```
str='X-DSPAM-Confidence:0.8475'
pos=str.find(':')
data=str[pos+1:]
fdata=float(data)
print(fdata)
```

Files

Python program to read file line by line

```
fh=open('poppins.txt')
cnt=0
for line in fh:
   line=line.strip()
   print(line)
   cnt=cnt+1
print('There are %d poppins'%(cnt))
fh.close()
```

Python program to demonstrate searching in a file and exception handling of files

```
try:
   fh=open('poppiins.txt')
   cnt=0
   for line in fh:
      line=line.strip()
      print(line)
      if line=='red':
         cnt=cnt+1
```

```
    print('There are %d red poppins'%(cnt))
    fh.close()
except:
    print('unable to open file')
    exit()
```

Python program to count number of emails received in a text file 'email.txt' from user specified domain

```
try:
    fh=open('emails.txt')
    domain=input('Enter the domain')
    for line in fh:
        line=line.strip()
        if line.find(domain)>-1:
            print(line)
    fh.close()
except:
    print('Error opening file')
    exit()
```

Python program to illustrate writing to file

```
fh=open('national.txt','w')
fh.write('National flower:Lotus\n')
fh.write('National animal:Tiger\n')
fh.write('National language:Hindi\n')
fh.close()
```

Exercise 1: Write a program to read through a file and print the contents of the file (line by line) all in upper case. Executing the program will look as follows:

```
fname=input('Enter file name:')
fhand = open(fname) try:
    fh=open('emails.txt')
    domain=input('Enter the domain')
    for line in fh:
        line=line.strip()
        if line.find(domain)>-1:
            print(line)
    fh.close()
except:
    print('Error opening file')
    exit()
count = 0
for line in fhand:
    print((line.upper()).strip())
```

Exercise 2: Write a program to prompt for a file name, and then read through the file and look for lines of the form:
X-DSPAM-Confidence:**0.8475**
When you encounter a line that starts with "X-DSPAM-Confidence:" pull apart the line to extract the floating-point number on the line. Count these lines and then compute the total of the spam confidence values from these lines. When you reach the end of the file, print out the average spam confidence

```
fname=input('Enter file name:')
fhand = open(fname)
count = 0
fsum=0
for line in fhand:
  if line.startswith('X-DSPAM-Confidence:'): pos=line.find('X-DSPAM-
    Confidence:')+len('X-DSPAM-Confidence:') fdata=float(line[pos:])

    fsum=fsum+fdata
    count=count+1
print('Total confidence',fsum,'Average confidence',fsum/count).
```

Exercise 3: Sometimes when programmers get bored or want to have a bit of fun, they add a harmless *Easter Egg* to their program Modify the program that prompts the user for the file name so that it prints a funny message when the user types in the exact file name "na na boo boo". The program should behave normally for all other files which exist and don't exist.

```
fname = input('Enter the file name: ')
if fname == 'na na boo boo':
  print('NA NA BOO BOO TO YOU - You have been punkd!')
  exit()

try:
  fhand = open(fname)
except:
  print('File cannot be opened:', fname)
  exit()
count = 0
for line in fhand:
  if line.startswith('Subject:'):
    count = count + 1
print('There were', count, 'subject lines in', fname)
```

Module 3

Lists, Dictionary, Tuples, Regular Expressions

Lists

1. Mutable list traversal

```
l=[1,2,3,4]
##print('Before',l)
##for ele in l:
##    print(ele*ele)
##print('After',l)
print('Before',l)
for i in range(len(l)):
    l[i]=l[i]*l[i]
print('After',l)
```

2. List and string conversion functions

```
ind_date='19/03/2019'
l=ind_date.split("/")
us_date=l[1]+"/"+l[0]+"/"+l[2]
print(us_date)
```

3. Parsing and searching lines in python

```
fh=open('email.txt')
for line in fh:
    line=line.strip()
    if(len(line)==0):
        continue
    words=line.split(" ")
    if words[1]=='ckr@jnnce.ac.in':
        print(words[3])
fh.close()
```

4. Python program demonstrating passing of lists to functions

```
def head(l):
    l.pop(0)

def chop(l):
    l.pop(0)
    l.pop(-1)

def nchop(l):
    return l[1:len(l)-1]


l=[1,2,3,4]
print('Before',l)
x=nchop(l)
```

```
    print('After',l)
    print(x)
```

Exercise 4: Write a program to open the file romeo.txt and read it line by line. For each line, split the line into a list of words using the split function. For each word, check to see if the word is already in a list. If the word is not in the list, add it to the list. When the program completes, sort and print the resulting words in alphabetical

```
fhand=open('romeo.txt')
words=fhand.read()
w=words.split()
l=list()
for s in w:
   if s in l:
      continue
   l.append(s)
l.sort()
print(l)
```

Exercise 5: Write a program to read through the mail box data and when you find line that starts with "From", you will split the line into words using the split function. We are interested in who sent the message, which is the second word on the From line. From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
fhand=open('mbox.txt')
count=0
for line in fhand:
   if line.startswith('From'):
      s=line.split()
      print(s[1])
      count=count+1
print('There were',count,' lines in the file with From as the first word')
```

Exercise 6: Rewrite the program that prompts the user for a list of numbers and prints out the maximum and minimum of the numbers at the end when the user enters "done". Write the program to store the numbers the user enters in a list and use the max() and min() functions to compute the maximum and minimum numbers after the loop completes.

```
l=list()
while True:
   num=input('Enter a number')
   if num=='done':
      break
   l.append(num)
print('Max',max(l),'Min',min(l))
```

Dictionaries

1. Frequency counter of each character in a string

```
str='sahasasimha'
d=dict()
for c in str:
    d[c]=d.get(c,0)+1
##    if c not in d:
##        d[c]=1
##    else:
##        d[c]=d[c]+1
print(d)
```

2. Find frequency of occurrence of words (histogram) for words in a file:

```
fh=open('nrj.txt')
d=dict()
for line in fh:
    line=line.strip()
    words=line.split()
    for word in words:
        d[word]=d.get(word,0)+1
fh.close()
print(d)
```

3. Find frequency of occurrence of words (histogram) for words in a file with ignoring punctuations:

```
import string
fh=open('nrj.txt')
d=dict()
for line in fh:
    line=line.strip()
    line=line.translate(line.maketrans('','',string.punctuation))
    line=line.lower()
    words=line.split()
    for word in words:
        d[word]=d.get(word,0)+1
fh.close()
print(d)
```

4. Sorting dictionary by key

```
student_marks={'4JN17CS017':23,'4JN17CS024':19,'4JN17CS011':20}
l=list(student_marks.keys())
l.sort()
for key in l:
   print(key,':',student_marks[key])
```

Exercise 2: Write a program that categorizes each mail message by which day of the week the commit was done. To do this look for lines that start with "From", then look for the third word and keep a running count of each of the days of the week. At the end of the program print out the contents of your dictionary (order does not matter). Sample Line:

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

Sample Execution:

```
python dow.py
Enter a file name: mbox-short.txt
{'Fri': 20, 'Thu': 6, 'Sat': 1}
```

```
fhand=open('mbox.txt')

d=dict()
for line in fhand:

   if line.startswith('From'):
      s=line.split()
      if len(s)>2:
         d[s[2]] = d.get(s[2],0) + 1

print(d)
```

Exercise 3: Write a program to read through a mail log, build a histogram using a dictionary to count how many messages have come from each email address, and print the dictionary.

```
Enter file name: mbox-short.txt
{'gopal.ramasammycook@gmail.com': 1, 'louis@media.berkeley.edu': 3,
'cwen@iupui.edu': 5, 'antranig@caret.cam.ac.uk': 1,
'rjlowe@iupui.edu': 2, 'gsilver@umich.edu': 3,
'david.horwitz@uct.ac.za': 4, 'wagnermr@iupui.edu': 1,
'zqian@umich.edu': 4, 'stephen.marquard@uct.ac.za': 2,
'ray@media.berkeley.edu': 1}
```

```
fhand=open('mbox.txt')

d=dict()
for line in fhand:

   if line.startswith('From'):
        s=line.split()
        d[s[1]] = d.get(s[1],0) + 1

print(d)
```

Exercise 4: Add code to the above program to figure out who has the most messages in the file.
Enter a file name: mbox-short.txt
cwen@iupui.edu 5
Enter a file name: mbox.txt
zqian@umich.edu 195

```
fhand=open('mbox.txt')

d=dict()
for line in fhand:

    if line.startswith('From'):
        s=line.split()
        d[s[1]] = d.get(s[1],0) + 1

l=list()
for key in d:
    l.append(d[key])
m=max(l)
r=list()
for key in d:
    if m==d[key]:
        r.append(key)
print(r)
```

Exercise 5: This program records the domain name (instead of the address) where the message was sent from instead of who the mail came from (i.e., the whole email address). At the end of the program, print out the contents of your dictionary.

```
python schoolcount.py
Enter a file name: mbox-short.txt
{'media.berkeley.edu': 4, 'uct.ac.za': 6, 'umich.edu': 7,
'gmail.com': 1, 'caret.cam.ac.uk': 1, 'iupui.edu': 8}
```

```
 fhand=open('mbox.txt')

d=dict()
for line in fhand:
    if line.startswith('From'):
        s=line.split()
        r=s[1].split("@")
        d[r[1]] = d.get(r[1],0) + 1

l=list()

for key in d:
    l.append(d[key])
m=max(l)
r=list()
for key in d:
    if m==d[key]:
        r.append(key)
print(r)
```

Tuples:

1. **Sort sentence by length of words in descending order**

```
txt="but soft what light in yonder window breaks"
words=txt.split()
l=list()
for word in words:
    l.append((len(word),word))
l.sort()
for (key,val) in l:
    print(val)
```

2. **Sorting and filtering dictionary using tuples**

```
stud_marks={'4JN17CS014':23,'4JN17CS009':24,'4JN17CS003':20}
l=list(stud_marks.items())
#l.sort()
res=list()
for (usn,marks) in l:
    res.append((marks,usn))
res.sort(reverse=True)
for (marks,usn) in res[:2]:
    print(usn,marks)
```

Exercise 1: Revise a previous program as follows: Read and parse the “From” lines and pull out the addresses from the line. Count the number of messages from each person using a dictionary. After all the data has been read, print the person with the most commits by creating a list of (count, email) tuples from the dictionary. Then

sort the list in reverse order and print out the person who has the most commits.
Sample Line:
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16
2008 Enter a file name: mbox-short.txt cwen@iupui.edu 5

Enter a file name: mbox.txt
zqian@umich.edu 195

```
fhand=open('mbox.txt')

d=dict()
for line in fhand:

   if line.startswith('From'):
      s=line.split()
      d[s[1]] = d.get(s[1],0) + 1
lst=list()
for key, val in list(d.items()):
   lst.append((val, key))
lst.sort(reverse=True)
print((lst[0])[1])
```

Exercise 2: This program counts the distribution of the hour of the day for each of the messages. You can pull the hour from the "From" line by finding the time string and then splitting that string into parts using the colon character. Once you have accumulated the counts for each hour, print out the counts, one per line, sorted by hour as shown below.

```
fhand=open('mbox.txt')

d=dict()
for line in fhand:
      if line.startswith('From'):
            s=line.split()
            if len(s)>4:
               t=s[5]
               (hr,min,sec)=t.split(":")
               d[hr] = d.get(hr,0) + 1
      lst=list()
for key, val in list(d.items()):
   lst.append((key, val))
lst.sort()
print(lst)
```

Regular Expressions

1. **Find lines containing email IDs**

```
import re
fh=open('mails.txt')
for line in fh:
   line=line.strip()
   if re.search('^From:\S*@\S+',line):
      print(line)
fh.close()
```

2. **Regular expression for ignoring case**

```
import re
tests="teSt1 Test2 TESt3"
lst=re.findall("(?i)test",tests)
print(lst)
```

3. **Demonstrating greedy and non-greedy regular expression parsing**

```
import re
html="<h1>JNNCE</h1>"
ans=re.match("<.*?>",html)
print(ans)
mes="The phone nums are 9900923050 89009889900 340 200007"
lst=re.findall("\d{10}",mes)
```

4. **Parse floating point numbers in each line of a file**

```
import re
fh=open('xval.txt')
for line in fh:
   line=line.strip()
   lst=re.findall('[0-9.]+',line)
   if(len(lst)>0):
      print(lst)
fh.close()
```

5. **Regular expression with escaping**

```
import re
mes="It costs $10.30 for book and 15.00 for pen"
lst=re.findall("\$[0-9.]+",mes)
print(lst)
```

Exercise 1: Write a simple program to simulate the operation of the grep command on
Unix. Ask the user to enter a regular expression and count the number of lines that matched the regular expression:

```
$ python grep.py
Enter a regular expression: ^Author
```

```
mbox.txt had 1798 lines that matched
^Author $ python grep.py
Enter a regular expression: ^Xmbox.
txt had 14368 lines that matched ^X-
$ python grep.py
Enter a regular expression: java$
mbox.txt had 4218 lines that matched java$
```

# Search for lines that start with From and have an at sign#

```
import re
hand = open('mbox.txt')
search = input('Enter a regular expression: )
count = 0
for line in hand:
    line =line.rstrip()
    if re.search(search, line):
        count = count + 1
print('mbox.txt had', count, 'lines that matched', search)
```

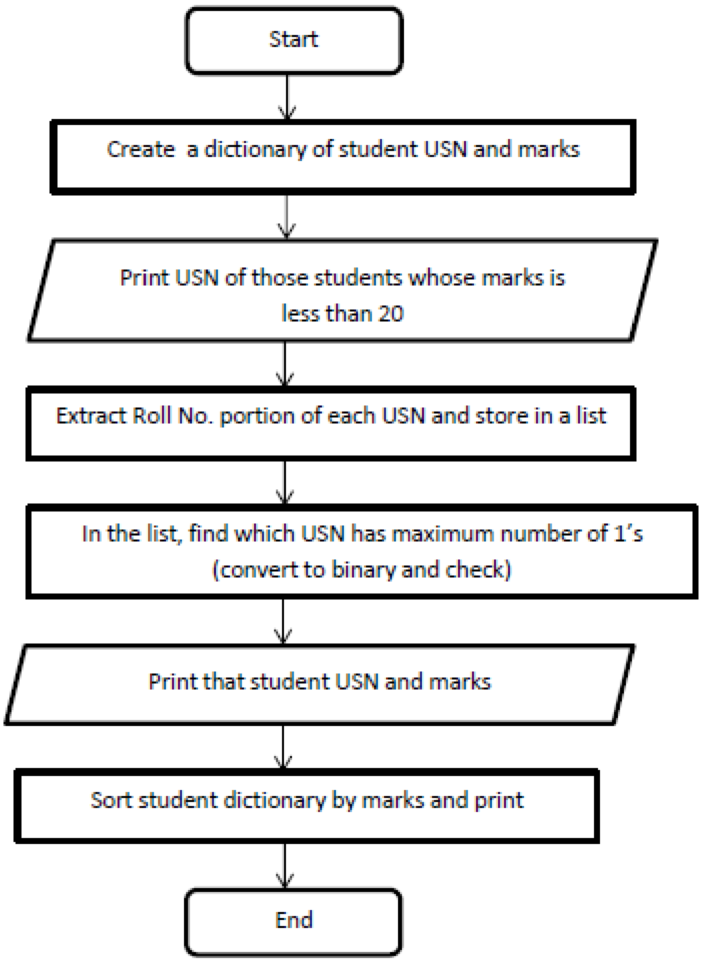
Exercise 2: Write a program to look for lines of the form `New Revision: 39772` and extract the number from each of the lines using a regular expression and the findall() method. Compute the average of the numbers and print out the average.

```
Enter file:mbox.txt
38549.7949721
Enter file:mbox-short.txt
39756.9259259
```

```
import re
fname = input('Enter file:')
hand = open(fname)
nums = list()
for line in hand:
    line = line.rstrip()
    x = re.findall('New Revision: ([0-9]+)', line)
    if len(x) == 1:
        val = float(x[0])
        nums.append(val)
print(len(nums))
print(sum(nums)/len(nums))
```

Assignment-II Solution:

Write a Python program to implement flow chart shown below:


Start
Create a dictionary of student USN and marks
Print USN of those students whose marks is less than 20
Extract Roll No. portion of each USN and store in a list
In the list, find which USN has maximum number of 1's (convert to binary and check)
Print that student USN and marks
Sort student dictionary by marks and print
End

```
#function to count number of ones in a decimal number
def find_ones(num):
    cnt=0
    while num!=0:
        b=num%2
        if b==1:
            cnt=cnt+1
        num=num//2
    return cnt

#Create a dictionary of student usn and marks
student_marks={"4JN17CS001":23,"4JN17CS052":9,"4JN17CS016":22,"4JN17CS033":20
,"4JN17CS011":13}
```

```
#print USN of students whose marks less than 20
for usns in list(student_marks.keys()):
    if student_marks[usns] < 20:
        print(usns)

#Extract roll no portion of each usn and store it in a list
l=list()
for usns in list(student_marks.keys()):
    l.append(usns[-3:])
print(l)

#In list find which USN has maximum number of 1s
max_ones=0
musn=''
for usns in l:
    cnt_ones=find_ones(int(usns))
    if cnt_ones>max_ones:
        max_ones=cnt_ones
        musn=usns

#print that student USN and marks
for usns in list(student_marks.keys()):
    if usns[-3:]==musn:
        print(usns,student_marks[usns])

#Sort student dictionary by marks and print
sl=list()
for usns in list(student_marks.keys()):
    sl.append((student_marks[usns],usns))
sl.sort()
for (marks,usn) in sl:
    print(usn,marks)
```

Compare and Contrast between Strings, Lists, Tuples and Dictionaries:

Point of Comparison	**Strings**	**Lists**	**Tuples**	**Dictionaries**
Definition	Sequence of characters	Sequence of values	Ordered sequence of values	Sequence of key-value pairs

Notation for representation	Single or double quotes
	Eg a="India"
		Notation:[ ]
		E.g. a=[1,2,3]
			Notation:( )
			E.g. a=(1,2,3)
				Notation: { }
				e.g. a={"first":1, "second":2}

Mutation	Strings are immutable	Lists are mutable	Tuples are immutable	Dictionaries are mutable

Accessing values	- Positive index starts from 0 - Negative index starts from -1 - Slicing with pair of indices Each indexing to access a character	- Positive index starts from 0 - Negative index starts from -1 - Slicing with pair of indices Each indexing to access a value	- Positive index starts from 0 - Negative index starts from -1 - Slicing with pair of indices Each indexing to access a value	- Positive index starts from 0 - Negative index starts from -1 - Slicing with pair of indices Each indexing to access a value
Index type	Integer	Integer	Integer	Non-integer indexing also possible
Adding and deleting of values	Not possible through indexing. Need to creat a newstring for the same	Possible directly through indexing.	Not possible through indexing. Need to create a new tuple for the same	Possible directly by giving same key
sort function	No direct method on strings	List on which sort applied gets sorted	sort on tuples returns a new sorted tuple	Sort cannot be applied on dictionary -sort by key by storing key in a list -sort by value by storing (key,value) pair in a tuple and then sorting
Impact on immutability on return values of the function	Need to return processed strings as its immutable (fruitful functions)	Functions return None as the passed list itself gets changed (void functions)	Need to return processed tuples as its immutable (fruitful functions)	Functions return None as the passed dictionary itself gets changed (void functions)
Some useful function examples	find() –search for occurrence of string and return its position lower()- convert a string to lower case strip()-Returns stripped version of string count()-Returns number of times a specified value occurs in a string	append()- add an element to end of list extend()- Add elements of any list to end of current list pop()- Removes element at specified position remove()-Remove first item with specified value	Same as list, along with: count()- Number of times a specified value occurs in a tuple index()- searches tuple for specified value and returns its position	get()- get value of specified key items()- returns a list containing tuple for each key-value pair keys()- Returns a list containing dictionary keys values()- Returns a list containing dictionary values

Module 4

Object Oriented Programming

1. Defining a class Point and adding functions using Points:

```
import math
class Point:
    """
    x,y
    """
def print_point(p):
    print('(%g,%g)'%(p.x,p.y))
def distance(p):
    return math.sqrt(p.x*p.x+p.y*p.y)

def distance_between(p1,p2):
    return math.sqrt((p1.x-p2.x)**2+(p1.y-p2.y)**2)
p=Point()
p.x=2
p.y=3
print_point(p)
d=distance(p)
p1=Point()
p1.x=2
p1.y=3
p2=Point()
p2.x=3
p2.y=4
d=distance_between(p1,p2)
print(d)
```

2. Design a class Rectangle and writing functions for doubling and moving of Rectangle object:

```
import copy
class Point:
    """
    attributes:x,y
    """

class Rectangle:
    """
     attributes:width,height,corner
    """
def center_of_rectangle(r):
    p=Point()
    p.x=r.corner.x+ r.width/2
    p.y=r.corner.y+ r.height/2
    return p

def double_rectangle(r):
    r.width*=2
    r.height*=2

def move_rectangle(r,x,y):
    r.corner.x+=x
    r.corner.y+=y

def print_point(p):
    print('(%g,%g)'%(p.x,p.y))


r=Rectangle()
r.width=15
r.height=5
r.corner=Point()
r.corner.x=5,
r.corner.y=10

c=center_of_rectangle(r)
print_point(c)

double_rectangle(r)
print(r.width,r.height)
move_rectangle(r,3,6)
print(r.corner.x,r.corner.y)
```

3. Demonstration of shallow and deep copy

```
p1=Point()
p1.x=5
p1.y=10
p2=copy.copy(p1)
p1.x=4
print_point(p1)
print_point(p2)

r1=Rectangle()
r1.width=15
r1.height=5
r1.corner=Point()
r1.corner.x=5
r1.corner.y=10
r2=copy.deepcopy(r1)
```

Exercise 15.1. Write a definition for a class named Circle with attributes center and radius, where center is a Point object and radius is a number. Instantiate a Circle object that represents a circle with its center at (150, 100) and radius 75. Write a function named point_in_circle that takes a Circle and a Point and returns True if the

Point lies in or on the boundary of the circle. Write a function named rect_in_circle that takes a Circle and a Rectangle and returns True if the Rectangle lies entirely in or on the boundary of the circle. Write a function named rect_circle_overlap that takes a Circle and a Rectangle and returns
True if any of the corners of the Rectangle fall inside the circle. Or as a more challenging version, return True if any part of the Rectangle falls inside the circle.

```
import math
import copy
class Point:
   """Represents a circle.

   Attributes: x, y

   """
class Rectangle:
   """  Represents a rectangle
   attributes:width,height,corner
"""

def distance_between_points(p1,p2):
      return math.sqrt((p1.x-p2.x)*(p1.x-p2.x) +(p1.y-p2.y)*(p1.y-p2.y))
class Circle:
   """Represents a circle.

   Attributes: center, radius
   """

def print_point(p):
   print('(%g,%g)'%(p.x, p.y))

def point_in_circle(point, circle):
   """Checks whether a point lies inside a circle (or on the boundary).

   point: Point object
   circle: Circle object
   """
   d = distance_between_points(point, circle.center)
   print(d)
   return d <= circle.radius
```

```
def rect_in_circle(rect, circle):
    """Checks whether the corners of a rect fall in/on a circle.
    rect: Rectangle object
    circle: Circle object
    """
    p = copy.copy(rect.corner)
    print_point(p)
    if not point_in_circle(p, circle):
        return False

    p.x += rect.width
    print_point(p)
    if not point_in_circle(p, circle):
        return False

    p.y -= rect.height
    print_point(p)
    if not point_in_circle(p, circle):
        return False

    p.x -= rect.width
    print_point(p)
    if not point_in_circle(p, circle):
        return False

    return True

def rect_circle_overlap(rect, circle):
    """Checks whether any corners of a rect fall in/on a circle.

    rect: Rectangle object
    circle: Circle object
    """
    p = copy.copy(rect.corner)
    print_point(p)
    if point_in_circle(p, circle):
        return True

    p.x += rect.width
    print_point(p)
    if point_in_circle(p, circle):
        return True

    p.y -= rect.height
    print_point(p)
    if point_in_circle(p, circle):
        return True

    p.x -= rect.width
    print_point(p)
    if point_in_circle(p, circle):
        return True

    return False

box = Rectangle()
box.width = 100.0
box.height = 200.0
box.corner = Point()
box.corner.x = 50.0
box.corner.y = 50.0

print(box.corner.x)
print(box.corner.y)

circle = Circle
circle.center = Point()
circle.center.x = 150.0
circle.center.y = 100.0
circle.radius = 75.0
print(circle.center.x)
print(circle.center.y)
print(circle.radius)
print(point_in_circle(box.corner, circle))
print(rect_in_circle(box, circle))
print(rect_circle_overlap(box, circle))
```

Difference between methods and functions

Point of Difference	Functions	Methods
Definition	Functions are defined outside the class	Methods are defined inside the class
Parameters	Atleast one object parameter is required to connect functions to classes	First parameter is self (this)
Invoking syntax	function_name(parameters)	object.method_name(parameters)
Constructors	No such functions	Special methods for initialization __init__(..)
Printing object	No such functions	Special methods for printing __str__(..)
Operator overloading	No such facility	Perform operator overloading by special method __add__()
Type based dispatch	Not possible	Possible through runtime instance checking

Classes and functions

1. Class time design with prototype and patches (add with each limitation found like validation, time updation etc) and demonstrating pure functions and modifiers

```
class Time:
    """
    attributes:hours,minutes,seconds
    """

#Pure function
def add_time(t1,t2):
    t=Time()
    t.seconds=t1.seconds+t2.seconds
    t.minutes=t1.minutes+t2.minutes
    t.hours=t1.hours+t2.hours
    if(t.seconds>=60):
        t.seconds-=60
        t.minutes+=1
    if(t.minutes>=60):
        t.minutes-=1
        t.hours+=1
    return t


def increment(t,sec):
    t.seconds+=sec
    while t.seconds>=60:
        t.seconds-=60
        t.minutes+=1

def valid_time(t):
    if t.hours <0 or t.minutes<0 or t.seconds<0:
        return False
    if t.minutes>=60 or t.seconds>=60:
        return False
    return True

def print_time(t):
    print(t.hours,':',t.minutes,':',t.seconds)

t1=Time()
t1.hours=10
t1.minutes=16
t1.seconds=23
assert valid_time(t1)
t2=Time()
t2.hours=1
```

```
t2.minutes=3
t2.seconds=50
assert valid_time(t2)
t=add_time(t1,t2)
print_time(t)

t=Time()
t.hours=11
t.minutes=20
t.seconds=40
increment(t,130)
print_time(t)
```

2. A more planned Time class avoiding checking of minutes and hours exceeding 59.

```
class Time:
    """
    attributes:hours,minutes,seconds
    """

#Pure function
def add_time(t1,t2):
    n=time_to_int(t1)+time_to_int(t2)
    t=int_to_time(n)
    return t


def increment(t,sec):
    n=time_to_int(t)+sec
    t=int_to_time(n)


def valid_time(t):
    if t.hours <0 or t.minutes<0 or t.seconds<0:
        return False
    if t.minutes>=60 or t.seconds>=60:
        return False
    return True

def time_to_int(t):
    seconds=t.minutes*60+t.seconds
    seconds+=t.hours*3600
    return seconds

def int_to_time(seconds):
    t=Time()
    (minutes,t.seconds)=divmod(seconds,60)
    (t.hours,t.minutes)=divmod(minutes,60)
    return t


def print_time(t):
    print(t.hours,':',t.minutes,':',t.seconds)

t1=Time()
t1.hours=10
t1.minutes=16
t1.seconds=23
assert valid_time(t1)
t2=Time()
t2.hours=1
t2.minutes=3
t2.seconds=50
assert valid_time(t2)
t=add_time(t1,t2)
print_time(t)

t=Time()
t.hours=11
t.minutes=20
t.seconds=40
increment(t,130)
print_time(t)
```

Exercise 16.1. Write a function called mul_time that takes a Time object and a number and returns a new Time object that contains the product of the original Time and the number. Then use mul_time to write a function that takes a Time object that represents the finishing time in a race, and a number that represents the distance, and returns a Time object that represents the average pace (time per mile).

```
class Time:
    """Represents the time of day.

    attributes: hour, minute, second
    """
def print_time(t):
    """Prints a string representation of the time.

    t: Time object
    """
    print('%.2d:%.2d:%.2d' % (t.hour, t.minute, t.second))

def valid_time(time):
    """Checks whether a Time object satisfies the invariants.

    time: Time
    returns: boolean
    """
    if time.hour < 0 or time.minute < 0 or time.second < 0:
        return False
    if time.minute >= 60 or time.second >= 60:
        return False
    return True

def int_to_time(seconds):
    """Makes a new Time object.

    seconds: int seconds since midnight.
    """
    time = Time()
    minutes, time.second = divmod(seconds, 60)
    time.hour, time.minute = divmod(minutes,
    60) return time
def time_to_int(time):
    """Computes the number of seconds since midnight.

    time: Time object.
    """
    minutes = time.hour * 60 + time.minute
    seconds = minutes * 60 + time.second
    return seconds

def is_after(t1, t2):
    """Returns True if t1 is after t2; false otherwise."""
    return (t1.hour, t1.minute, t1.second) > (t2.hour, t2.minute, t2.second)


def increment(t1, seconds):
    """Adds seconds to a Time object."""
    assert valid_time(t1)
    seconds += time_to_int(t1)
    return int_to_time(seconds)


def mul_time(t1, factor):
    """Multiplies a Time object by a factor."""
    assert valid_time(t1)
    seconds = time_to_int(t1) * factor
    return int_to_time(seconds)


race_time = Time()
race_time.hour = 1
race_time.minute = 34
race_time.second = 5

print('Half marathon time', end=' ')
print_time(race_time)

distance = 13.1      # miles
pace = mul_time(race_time, 1/distance)

print('Time per mile', end=' ')
print_time(pace)
```

Exercise 16.2. The datetime module provides time objects that are similar to the Time objects in this chapter, but they provide a rich set of methods and operators.
1. Use the datetime module to write a program that gets the current date and prints the day of the week.
2. Write a program that takes a birthday as input and prints the user's age and the number of days, hours, minutes and seconds until their next birthday.
3. For two people born on different days, there is a day when one is twice as old as the other. That's their Double Day. Write a program that takes two birthdays and computes their Double
Day.
4. For a little more challenge, write the more general version that computes the day when one person is n times older than the other.

```
from datetime import datetime
class Time:
    """Represents the time of day.

    attributes: hour, minute, second
    """
def print_time(t):
    print('%.2d:%.2d:%.2d' % (t.hour, t.minute, t.second))

def valid_time(time):
    """Checks whether a Time object satisfies the invariants.
    time: Time
    returns: boolean
    """
    if time.hour < 0 or time.minute < 0 or time.second < 0:
        return False
    if time.minute >= 60 or time.second >= 60:
        return False
    return True
```

```
def int_to_time(seconds):
    """Makes a new Time object.

    seconds: int seconds since midnight.
    """
    time = Time()
    minutes, time.second = divmod(seconds, 60)
    time.hour, time.minute = divmod(minutes,60)
    return time

def time_to_int(time):
    """Computes the number of seconds since midnight.
    tme: Time object.
    """
    minutes = time.hour * 60 + time.minute
    seconds = minutes * 60 + time.second
    return seconds

def is_after(t1, t2):
    """Returns True if t1 is after t2; false otherwise."""
    return (t1.hour, t1.minute, t1.second) > (t2.hour, t2.minute, t2.second)


def increment(t1, seconds):
    """Adds seconds to a Time object."""
    assert valid_time(t1)
    seconds += time_to_int(t1)
    return int_to_time(seconds)
```

Classes and methods

1. Converting all functions to methods for Time class

```
class Time:
    """
    attributes:hours,minutes,seconds
    """

    def __init__(self,h=0,m=0,s=0):
        self.hours=h
        self.minutes=m
        self.seconds=s
    def __add__(self,t2):
        if isinstance(t2,Time):
            return self.add_time(t2)
        else:
            return self.increment(t2)
    def add_time(self,t2):
        n=self.time_to_int()+t2.time_to_int()
        t=self.int_to_time(n)
        return t


    def increment(self,sec):
        n=self.time_to_int()+sec
        return self.int_to_time(n)



    def time_to_int(self):
        seconds=self.minutes*60+self.seconds
        seconds+=self.hours*3600
        return seconds

    def valid_time(self):
        if self.hours <0 or self.minutes<0 or self.seconds<0:
            return False
        if self.minutes>=60 or self.seconds>=60:
            return False
        return True

    def int_to_time(self,seconds):

        (minutes,self.seconds)=divmod(seconds,60)
        (self.hours,self.minutes)=divmod(minutes,60)
        return self


    def __str__(self):
        return ('%.2d:%.2d:%.2d'%(self.hours,self.minutes,self.seconds))

t1=Time(10,16)

assert t1.valid_time()
t2=Time(1,3,50)
t3=Time(1,1,10)

assert t2.valid_time()
t=t1+t2
print(t)

t=Time(11,20,40)

t=t+130
```

```
print(t)
for attr in vars(t):
    print(attr,getattr(t,attr))
#print(vars(t))
```

2. Adding overloading operators functionality to Time class and demonstrating type based dispatch:

```
class Time:
    """
    attributes:hours,minutes,seconds
    """

    def __init__(self,h=0,m=0,s=0):
        self.hours=h
        self.minutes=m
        self.seconds=s

    def __add__(self,t2):
        if isinstance(t2,Time):
            return self.add_time(t2)
        else:
            return self.increment(t2)

    def add_time(self,t2):
        n=self.time_to_int()+t2.time_to_int()
        t=self.int_to_time(n)
        return t


    def increment(self,sec):
        n=self.time_to_int()+sec
        return self.int_to_time(n)



    def time_to_int(self):
        seconds=self.minutes*60+self.seconds
        seconds+=self.hours*3600
        return seconds

    def valid_time(self):
        if self.hours <0 or self.minutes<0 or self.seconds<0:
            return False
        if self.minutes>=60 or self.seconds>=60:
            return False
        return True
    def int_to_time(self,seconds):

        (minutes,self.seconds)=divmod(seconds,60)
        (self.hours,self.minutes)=divmod(minutes,60)
        return self


    def __str__(self):
        return ('%.2d:%.2d:%.2d'%(self.hours,self.minutes,self.seconds))

t1=Time(10,16)

assert t1.valid_time()
t2=Time(1,3,50)

assert t2.valid_time()
t=t1+t2
print(t)
#t.print_time()

t=Time(11,20,40)

t=t+130
print(t)
```

Exercise 17.1. Change the attributes of Time to be a single integer representing seconds since midnight. Then modify the methods (and the function int_to_time) to work with the new implementation.

```
. class Time:
    """Represents the time of day.

    attributes: hour, minute, second
    """
    def __init__(self, hour=0, minute=0, second=0):
        """Initializes a time object.

        hour: int
        minute: int
        second: int or float
        """
        minutes = hour * 60 + minute
        self.seconds = minutes * 60 + second

    def __str__(self):
        """Returns a string representation of the time."""
        minutes, second = divmod(self.seconds, 60)
        hour, minute = divmod(minutes, 60)
        return '%.2d:%.2d:%.2d' % (hour, minute, second)

    def print_time(self):
        """Prints a string representation of the time."""
        print(str(self))

    def time_to_int(self):
        """Computes the number of seconds since midnight."""
        return self.seconds

    def is_after(self, other):
        """Returns True if t1 is after t2; false otherwise."""
        return self.seconds > other.seconds

def __add__(self, other):
        """Adds two Time objects or a Time object and a number.

        other: Time object or number of seconds
        """
        if isinstance(other, Time):
            return self.add_time(other)
        else:
            return self.increment(other)

    def __radd__(self, other):
        """Adds two Time objects or a Time object and a number."""
        return self.__add__(other)

    def add_time(self, other):
        """Adds two time objects."""
        assert self.is_valid() and other.is_valid()
        seconds = self.seconds + other.seconds
        return int_to_time(seconds)

    def increment(self, seconds):
        """Returns a new Time that is the sum of this time and seconds."""
        seconds += self.seconds
        return int_to_time(seconds)

    def is_valid(self):
        """Checks whether a Time object satisfies the invariants."""
        return self.seconds >= 0 and self.seconds < 24*60*60


def int_to_time(seconds):
    """Makes a new Time object.
    seconds: int seconds since midnight.
    """
    return Time(0, 0, seconds)


start = Time(9, 45, 00)
start.print_time()

end = start.increment(1337)
end.print_time()

print('Is end after start?')
print(end.is_after(start))


print('Using __str__')
print(start, end)

start = Time(9, 45)
duration = Time(1, 35)
print(start + duration)
print(start + 1337)
print(1337 + start)

print('Example of polymorphism')
t1 = Time(7, 43)
t2 = Time(7, 41)
t3 = Time(7, 37)
total = sum([t1, t2, t3])
print(total)
```

Exercise 17.2. This exercise is a cautionary tale about one of the most common, and difficult to find, errors in Python. Write a definition for a class named Kangaroo with the following methods:
1. An __init__ method that initializes an attribute named pouch_contents to an empty list.
2. A method named put_in_pouch that takes an object of any type and adds it to pouch_contents.
3. A __str__ method that returns a string representation of the Kangaroo object and the contents of the pouch.
Test your code by creating two Kangaroo objects, assigning them to variables named kanga and roo, and then adding roo to the contents of kanga's pouch.

```
class Kangaroo:
    """A Kangaroo is a marsupial."""

  def __init__(self, name, contents=[]):
        """Initialize the pouch contents.

        name: string
        contents: initial pouch contents.
        """
        self.name = name
        self.pouch_contents = contents

    def __init__(self, name, contents=None):
        """Initialize the pouch contents.

        name: string
        contents: initial pouch contents.
        """


        self.name = name
        if contents == None:
            contents = []
        self.pouch_contents = contents

    def __str__(self):
        """Return a string representaion of this Kangaroo.
        """
        t = [ self.name + ' has pouch contents:' ]
        for obj in self.pouch_contents:
            s = '    ' + object.__str__(obj)
            t.append(s)
        return '\n'.join(t)

    def put_in_pouch(self, item):
        """Adds a new item to the pouch contents.

        item: object to be added
        """
        self.pouch_contents.append(item)

kanga = Kangaroo('Kanga')
roo = Kangaroo('Roo')
kanga.put_in_pouch('wallet')
kanga.put_in_pouch('car keys')
kanga.put_in_pouch(roo)

print(kanga)
print(roo)


def mul_time(t1, factor):
    """Multiplies a Time object by a factor."""
    assert valid_time(t1)
    seconds = time_to_int(t1) * factor
    return int_to_time(seconds)


def days_until_birthday(birthday):
    """How long until my next birthday?"""
    today = datetime.today()
    # when is my birthday this year?
    next_birthday = datetime(today.year, birthday.month, birthday.day)

    # if it has gone by, when will it be next year
    if today > next_birthday:
        next_birthday = datetime(today.year+1, birthday.month, birthday.day)

    # subtraction on datetime objects returns a timedelta object
    delta = next_birthday - today
    return delta.days


def double_day(b1, b2):
    """Compute the day when one person is twice as old as the other.

    b1: datetime birthday of the younger person
    b2: datetime birthday of the older person
    """
    assert b1 > b2
    delta = b1 - b2
    dday = b1 + delta
    return dday


def datetime_exercises():
    """Exercise solutions."""

    # print today's day of the
    week today = datetime.today()
    print(today.weekday())
    print(today.strftime('%A'))

    # compute the number of days until the next birthday
    # (note that it usually gets rounded down)
    birthday = datetime(1967, 5, 2)
    print('Days until birthday', end=' ')
    print(days_until_birthday(birthday))
    # compute the day one person is twice as old as
    another b1 = datetime(2006, 12, 26)
    b2 = datetime(2003, 10, 11)
    print('Double Day', end=' ')
    print(double_day(b1, b2))

datetime_exercises()
```

Module 5

Network Programming, Using web services and Database Programming

1. **Socket program to get html data from web server**

```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(('localhost',80))
cmd='GET http://localhost/welcome.html HTTP/1.0\r\n\r\n'.encode()
s.send(cmd)

while True:
    d=s.recv(512)
    if len(d)<=0:
        break
    print(d.decode())
```

2. **Socket program to download an image from webserver**

```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(('localhost',80))
cmd='GET http://localhost/clouds.jpg HTTP/1.0\r\n\r\n'.encode()
s.sendall(cmd)

img=b''
while True:
    d=s.recv(5120)
    if len(d)<=0:
        break
    img=img+d

pos=img.find(b'\r\n\r\n')
fh=open('wrt.jpg','wb')
fh.write(img[pos+4:])
fh.close()
```

3. **Fetching html data from webserver using urllib**

```
import urllib.request
data=urllib.request.urlopen("http://localhost/welcome.html")
for line in data:
    print(line.decode())
```

4. **Fetching image from webserver using urllib**

```
import urllib.request
r=urllib.request.urlopen('http://localhost/zoom.jpg')
img=b''
while True:
    d=r.read(5120)
    if len(d)<=0:
        break
    img=img+d
```

```
fh=open('wrt.jpg','wb')
fh.write(img)
fh.close()
```

5. **Python program to fetch hyperlinks in a html file using regular expressions:**

```
import re
import urllib.request

a=urllib.request.urlopen("http://localhost/links.html").read()
data=re.findall(b'http[s]?://.+"',a)
for links in data:
    print(links.decode())
```

6. **Extracting information about all hyperlinks in a webpage using BeautifulSoup**

```
from bs4 import BeautifulSoup
import urllib.request

a=urllib.request.urlopen("http://localhost/links.html").read()
soup=BeautifulSoup(a,"html.parser")
links=soup("a")
for link in links:
    #print(link.get("href"))
    print(link.attrs)
```

7. **Counting number of paragraphs using BeautifulSoup:**

```
from bs4 import BeautifulSoup
import urllib.request

a=urllib.request.urlopen("http://localhost/prg.html").read()
soup=BeautifulSoup(a,"html.parser")
links=soup("p")
##cnt=0
##for link in links:
##    cnt=cnt+1
print('No of paragraphs',len(links))
```

8. **Fetching ratings and product names of amazon link using Beautiful Soup:**

```
from bs4 import BeautifulSoup
import urllib.request

a=urllib.request.urlopen("http://localhost/HotDeals.html").read()
soup=BeautifulSoup(a,"html.parser")
links=soup("a",attrs={"class":"a-link-normal"})
for link in links:
    if link.get("title") is None:
        continue
    print(link.get("title"))

ratings=soup("span",attrs={"class":"acs_product-rating__review-count"})
for link in ratings:
    if link.text is None:
        continue
    print(link.text)
```

9. **Parsing XML example data in python:**

```
import xml.etree.ElementTree as ET
data='''
<cover>
<person>
<name>Pandu</name>
<phone type="mobile">998899889</phone>
<email hide="yes"/>
</person>
<person>
<name>Punda</name>
<phone type="mobile">978899889</phone>
<email hide="no"/>
</person>
</cover>
'''

tree=ET.fromstring(data)
names=tree.findall("person/name")
for name in names:
    print(name.text)
emails=tree.findall("person/email")
for email in emails:
    print(email.get("hide"))
```

10. **Parsing JSON data in python**

```
import json
data='''
[
{
  "name":"Pandu",
  "phone":
        {
          "type":"mobile"
        },
  "email":{
          "hide":"yes"
        }
},
{
  "name":"Punda",
  "phone":
        {
           "type":"mobile"
        },
  "email":{
           "hide":"no"
        }
}
]
'''

jn=json.loads(data)
print(jn[0]["name"],jn[1]["name"])
print(jn[0]["email"]["hide"])
```

11. Performing Geocoding using Python and JSON

```
import json
import urllib.request

data=urllib.request.urlopen('https://maps.googleapis.com/maps/api/geoco
de/json?address=JNNCE&key=..').read()
jn=json.loads(data)
print(data)
lat = jn['results'][0]['geometry']['location']['lat']
lng = jn['results'][0]['geometry']['location']['lng']
print('lat', lat, 'lng', lng)
location = jn['results'][0]['formatted_address']
print(location)
```

12. Twitter interaction in Python with getting metadata, updating status, fetching friends and followers list and searching based on hash tags:

```
import tweepy
consumer_key = '..'
consumer_secret = '…'
access_token = '..'
access_token_secret = '…'


auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

api = tweepy.API(auth)

user = api.me()
print('Name: ' + user.name)
print('Location: ' + user.location)
print('Friends: ' + str(user.friends_count))
print('Followers: ' + str(user.followers_count))
print('Status Count:'+str(user.statuses_count))

#api.update_status('#SRHvsDC:Well played Panth and Prithvi')

msgs = []
msg =[]

for tweet in tweepy.Cursor(api.search, q='#MIvCSK', rpp=100).items(50):
    msg = [tweet.text.encode('unicode-escape').decode('utf-8'), tweet.source,
tweet.source_url]
    msgs.append(msg)

for msg in msgs:
    print(msg)

for user in tweepy.Cursor(api.friends, screen_name="Chetan").items():
    print('friend: ' + user.screen_name)

###Follower list
for user in tweepy.Cursor(api.followers, screen_name="Chetan").items():
   print('follower: ' + user.screen_name)
##
```

13. Interacting with SQLite in Python

```
import sqlite3

conn=sqlite3.connect('music.sql')
cur=conn.cursor()

cur.execute('DROP TABLE IF EXISTS tracks')
cur.execute('CREATE TABLE tracks(title TEXT, plays int)')

cur.execute('INSERT INTO tracks(title,plays) values(?,?)',('A',3))
cur.execute('INSERT INTO tracks(title,plays) values(?,?)',('B',2))
cur.execute('UPDATE tracks set plays=7 where title=?',('A',))
conn.commit()

cursor=cur.execute("SELECT * from tracks")
for row in cursor:
    print(row)

cur.close()
```

Exercise Solutions:

Networked programs

Exercise 1: Change the socket program socket1.py to prompt the user for the URL so it can read any web page. You can use split('/') to break the URL into its component parts so you can extract the host name for the socket connect call. Add error checking using try and except to handle the condition where the user enters an improperly formatted or non-existent URL.

```
import socket

url = input('Enter: ')
try:
    words = url.split('/')
    host = words[2]

    mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    mysock.connect((host, 80))
    mysock.send(('GET '+url+' HTTP/1.0\r\n\r\n').encode())

    while True:
        data = mysock.recv(512)
        if (len(data) < 1):
            break
        print(data.decode(), end='')

    mysock.close()
except:
    print('not formatted properly')
```

Exercise 2: Change your socket program so that it counts the number of characters it has received and stops displaying any text after it has shown 3000 characters. The program should retrieve the entire document and count the total number of characters and display the count of the number of characters at the end of the document.

```
import socket
url = input('Enter: ')
try:
    words = url.split('/')
    host = words[2]

    mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    mysock.connect((host, 80))
    mysock.send(('GET '+url+' HTTP/1.0\r\n\r\n').encode())

    n=''
    while True:
        data = mysock.recv(512)
        if (len(data) < 1):
            break
        n=n+(data.decode())

    print(n[:3001])
    print('Total number of characters',len(n))
    mysock.close()
except:
    print('not formatted properly')
```

Exercise 3: Use urllib to replicate the previous exercise of (1) retrieving the document from a URL, (2) displaying up to 3000 characters, and (3) counting the overall number of characters in the document. Don't worry about the headers for this exercise, simply show the first 3000 characters of the document contents.

```
import socket
import urllib.request

try:
    url = input('Enter: ')
    n = urllib.request.urlopen(url).read().decode()
    print(n[:3001])
    print('Total number of characters',len(n))

except:
    print('not formatted properly')
```

Exercise 4: Change the urllinks.py program to extract and count paragraph (p) tags from the retrieved HTML document and display the count of the paragraphs as the output of your program. Do not display the paragraph text, only count them. Test your program on several small web pages as well as some larger web pages.

```
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup


url = input('Enter - ')
html = urllib.request.urlopen(url).read()
soup = BeautifulSoup(html, 'html.parser')
# Retrieve all of the anchor
tags tags = soup('p')
count=0
for tag in tags:
    count=count+1
```

```
print('Number of paragraphs',count)
```

Exercise 5: (Advanced) Change the socket program so that it only shows data after the headers and a blank line have been received. Remember that recv is receiving characters (newlines and all), not lines.

```
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)

n=''
while True:
   data = mysock.recv(512)
   if len(data) < 1:
      break
   n=n+data.decode()
pos=n.find('\r\n\r\n')
print(n[pos+4:])

mysock.close()
```

Using Web Services

Exercise 1: Change either the www.py4e.com/code3/geojson.py or www.py4e.com/code3/geoxml.py to print out the two-character country code from the retrieved data. Add error checking so your program does not traceback if the country code is not there.

```
import urllib.request, urllib.parse, urllib.error
import xml.etree.ElementTree as ET

serviceurl="http://maps.googleapis.com/maps/api/geocode/json?"
address="address=JNNCE&key=AIzaSyCETOANWzQcYrPpQLo5mOLpYWaEXx
XxAys"
serviceurl = serviceurl + address

while True:
   address = input('Enter location: ')
   if len(address) < 1: break

   url = serviceurl + urllib.parse.urlencode({'address': address})
   print('Retrieving', url)
   uh = urllib.request.urlopen(url)
```

```
    data = uh.read()
    print('Retrieved', len(data), 'characters')
    print(data.decode())
    tree = ET.fromstring(data)

    results = tree.findall('result')
    lat = results[0].find('geometry').find('location').find('lat').text lng
    = results[0].find('geometry').find('location').find('lng').text
    location = results[0].find('formatted_address').text

    print('lat', lat, 'lng', lng)
    print(location)
```

Link to prezi presentation of various modules:
http://prezi.com/x_pb6ae4ae5f/?utm_campaign=share&utm_medium=copy