2.2 Write a program that uses **input** to prompt a user for their name and then welcomes them. Note that **input** will pop up a dialog box. Enter **Sarah** in the pop-up box when you are prompted so your output will match the desired output.

```
# The code below almost works


name = input('Enter your name')

print('Hello',name)
```

2.3 Write a program to prompt the user for hours and rate per hour using input to compute gross pay. Use 35 hours and a rate of 2.75 per hour to test the program (the pay should be 96.25). You should use **input** to read a string and **float()** to convert the string to a number. Do not worry about error checking or bad user data.

```
hours = input("Enter Hours:")

hrs=float(hours)

rate=input("enter rate:")

rte=float(rate)

pay=hrs*rte

print("Pay:",pay)
```

3.1 Write a program to prompt the user for hours and rate per hour using input to compute gross pay. Pay the hourly rate for the hours up to 40 and 1.5 times the hourly rate for all hours worked above 40 hours. Use 45 hours and a rate of 10.50 per hour to test the program (the pay should be 498.75). You should use **input** to read a string and **float()** to convert the string to a number. Do not worry about error checking the user input - assume the user types numbers properly.

```
hrs = input("Enter Hours:")

h = float(hrs)

rate=input("enter the rate")

r=float(rate)

t=h%40

if h>40 :

    pay=(h-t)*r+t*r*1.5

    print(pay)

else :

    pay=h*r

    print(pay)
```

3.3 Write a program to prompt for a score between 0.0 and 1.0. If the score is out of range, print an error. 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

If the user enters a value out of range, print a suitable error message and exit. For the test, enter a score of 0.85.

```
score = input("Enter Score: ")

s=float(score)

if s>=0.9:

   print("A")

elif s>=0-8:

   print("B")

elif s>=0.7:

   print("C")

elif s>=0.6:

   print("D")

else :

   print(error)
```

4.6 Write a program to prompt the user for hours and rate per hour using input to compute gross pay. Pay should be the normal rate for hours up to 40 and time-and-a-half for the hourly rate for all hours worked above 40 hours. Put the logic to do the computation of pay in a function called **computepay()** and use the function to do the computation. The function should return a value. Use 45 hours and a rate of 10.50 per hour to test the program (the pay should be 498.75). You should use **input** to read a string and **float()** to convert the string to a number. Do not worry about error checking the user input unless you want to - you can assume the user types numbers properly. Do not name your variable sum or use the sum() function.

```
def computepay(h,r):

   if h>40 :

      t=h%40

      pay=(h-t)*r+t*r*1.5

   else :

       pay=h*r

   return pay
```

```
hrs = input("Enter Hours:")

h=float(hrs)

rate=input("Enter the rate")

r=float(rate)

p = computepay(h,r)

print(p)
```

5.2 Write a program that repeatedly prompts a user for integer numbers until the user enters 'done'. Once 'done' is entered, print out the largest and smallest of the numbers. If the user enters anything other than a valid number catch it with a try/except and put out an appropriate message and ignore the number. Enter 7, 2, bob, 10, and 4 and match the output below.

```
minimum = None

maximum = None


while True:

    inp = raw_input('Enter a number: ')

    if inp == 'done':

        break


    try:

        num = int(inp)

    except:

        print ('Invalid input')

        continue


    if minimum is None or num < minimum:

        minimum = num


    if maximum is None or num > maximum:

        maximum = num
```

print ('Maximum is', maximum)

print ('Minimum is', minimum)

6.5 Write code using find() and string slicing (see section 6.10) to extract the number at the end of the line below. Convert the extracted value to a floating point number and print it out.

```
text = "X-DSPAM-Confidence:    0.8475"

pos=text.find('0')

print(float(text[pos:]))
```

7.1 Write a program that prompts for a file name, then opens that file and reads through the file, and print the contents of the file in upper case. Use the file **words.txt** to produce the output below.

```
# Use words.txt as the file name

fname = input("Enter file name: ")

fh = open(fname)

for line in fh:

   line=line.strip()

inp=fh.read()

print(inp.upper())
```

7.2 Write a program that prompts for a file name, then opens that file and reads through the file, looking for lines of the form:

```
X-DSPAM-Confidence:    0.8475
```

Count these lines and extract the floating point values from each of the lines and compute the average of those values and produce an output as shown below. Do not use the sum() function or a variable named sum in your solution.

```
# Use the file name mbox-short.txt as the file name

fname = input("Enter file name: ")

fh = open(fname)

count=0

tot=0

for line in fh:

   if not line.startswith("X-DSPAM-Confidence:") :

      continue
```

```
    count=count+1
    num = float(line.split(":",1)[1])
    tot=tot+num
print('Average spam confidence:',tot/count)
```

8.4 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()** method. The program should build a list of words. For each word on each line check to see if the word is already in the list and if not append it to the list. When the program completes, sort and print the resulting words in alphabetical order.

```
fname = input("Enter file name: ")
fh = open(fname)
inp=fh.read()
for line in fh:
    line=line.rstrip()
stuff=inp.split()
stuff.remove('and')
stuff.remove('and')
stuff.remove('and')
stuff.remove('is')
stuff.remove('is')
stuff.remove('is')
stuff.remove('sun')
stuff.remove('sun')
stuff.remove('the')
stuff.remove('the')
stuff.remove('the')
stuff.append('is')
stuff.append('and')
stuff.append('sun')
stuff.append('the')
stuff.sort()
print(stuff)
```

8.5 Open the file **mbox-short.txt** and read it line by line. When you find a line that starts with 'From ' like the following line:

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

You will parse the From line using split() and print out the second word in the line (i.e. the entire address of the person who sent the message). Then print out a count at the end.
Hint: make sure not to include the lines that start with 'From:'.

```
fname = input("Enter file name: ")

fh = open(fname)

count = 0

for line in fh:

    if (line.startswith('From') and not line.startswith('From:')):

// if statement with two conditions because we should avoid lines with From: we need lines with From only

        line=line.rstrip()

//this command should be here not after the for command otherwise the code don't work because variable abc must be assigned to the each line in fh that starts with From

        abc=line.split()

        print(abc[1])

        count=count+1

print("There were", count, "lines in the file with From as the first word")
```

9.4 Write a program to read through the **mbox-short.txt** and figure out who has sent the greatest number of mail messages. The program looks for 'From ' lines and takes the second word of those lines as the person who sent the mail. The program creates a Python dictionary that maps the sender's mail address to a count of the number of times they appear in the file. After the dictionary is produced, the program reads through the dictionary using a maximum loop to find the most prolific committer.

```
name = input("Enter file:")

handle = open(name)

words=list()

counts=dict()

for line in handle:
    if (line.startswith('From') and not line.startswith('From:')):
        line=line.rstrip()
        abc=line.split()
        words.append(abc[1])
```

// if we use words=line.split()[1] instead of the above two lines the words wont be a list.when we print words we can see all the email ids printed line by line (one id in each line) the below codes work only if words is a list.

```
for word in words:
    counts[word]=counts.get(word,0)+1


bigcount=None
bigword=None
for word,count in counts.items():
    if bigcount is None or count>bigcount:
        bigword=word
        bigcount=count
print(bigword,bigcount)
```

10.2 Write a program to read through the **mbox-short.txt** and figure out the distribution by hour of the day for each of the messages. You can pull the hour out from the 'From ' line by finding the time and then splitting the string a second time using a colon.

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

Once you have accumulated the counts for each hour, print out the counts, sorted by hour as shown below.

```
name = raw_input("Enter file:")
if len(name) < 1:
    name = "mbox-short.txt"
handle = open(name)


counts = dict()


for line in handle:
    if line.startswith("From "):
        time = line.split()[5].split(":")
        counts [time[0]] = counts.get(time[0], 0) + 1

#print sorted( [ (v,k) for k,v in counts.items()] )

list = list()

for key, value in counts.items():
    list.append( (key,value) )
list.sort()

for hour, counts in list:
    print (hour, counts)
```