Question 1

In [177]:

```
#(a)
def maxnum(alist):
    alist.sort(reverse=True)
    #This just uses the built-in sort function, but in reverse
    return alist[0]
    #I pick the integer at the 0th index because with reverse sort
    #that would be the largest integer in the list
```

In [176]:

```
def minnum(alist):
    alist.sort()
    #using the built-in sort function (smallest to largest)
    return alist[0]
    #integer at the 0th index is the smallest
```

In [179]:

```
#(b)
minnum([1,'a',2,3,4]) #No it does not work over other data types
```

```
---------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
<ipython-input-179-fc697e956fc0> in <module>()
      1 #(b)
----> 2 minnum([1,'a',2,3,4]) #No it does not work over other data typ
es

<ipython-input-176-5964e0cf7a75> in minnum(alist)
      1 def minnum(alist):
----> 2     alist.sort()
      3     #using the built-in sort function (smallest to largest)
      4     return alist[0]
      5     #integer at the 0th index is the smallest

TypeError: unorderable types: str() < int()
```

In [183]:

```
#(c)
#I made the function more robust by employing a
#list comprehension to consume the input list
#and recreate a new list that only contains integers
def maxnum(alist):
    newlist = [item for item in alist if type(item) == int]
    newlist.sort(reverse = True)
    return newlist[0]

def minnum(alist):
    newlist = [item for item in alist if type(item) == int]
    newlist.sort()
    return newlist[0]
```

In [184]:

```
minnum([1,5,2,7,3,5,9])
```

Out[184]:

1

In [185]:

```
minnum([1,2,3,4,"a",2,5,"hello",4,7])
```

Out[185]:

1

In [186]:

```
maxnum([1,2,3,4,"a",2,5,"hello",4,7])
```

Out[186]:

7

Question 2

In [189]:

```
data = [3,6,1,3,6,8,9,2,2,2,1]
```

In [190]:

```
mean_vals = 0
def mean(alist):
    mean_vals = sum(alist)/len(alist)
    return mean_vals
```

In [191]:

```
mean(data)
```

Out[191]:

```
3.909090909090909
```

In [192]:

```
a = 0
b = 0

def median(alist):
    alist.sort()
    if len(alist) % 2 == 0:
        a = len(alist)/2
        b = len(alist)/2 - 1
        c = .5*(test[int(a)]+test[int(b)])
        return c
    else:
        a = len(alist)//2
        return alist[a]
```

In [193]:

```
median(data)
```

Out[193]:

```
3
```

In [194]:

```
def rangefunc(alist):
    alist.sort()
    return alist[len(alist)-1]-alist[0]
```

In [195]:

```
rangefunc(data)
```

Out[195]:

```
8
```

In [196]:

```
def var(alist):
    print((1/len(alist)) * (sum([(item - mean(alist))**2
                                 for item in alist])))
```

In [197]:

```
var(data)
```

```
7.355371900826446
```

Question 3

(a)

$F_n = F_{n-1} + F_{n-2}$ for n < 2, and where $F_1 = 1$ and $F_2 = 2$

In [198]:

```
#I assume that the 0th Fibonacci number is 0, since we generally
#begin at the 1st Fibonacci number. Thus F1 and F2 = 1.
#We iterate through range(n-1) because the nth Fibonacci number
#must have n-1 loops through (because I started at 0,1 to account for
#the 0th Fibonacci number).
#i.e. fib_iter(3) must calculate two numbers, the 2nd and 3rd
#Fibonacci numbers (since the 1st number is given).

#(b)
def fib_iter(n, show_sequence = False):
    fprev1 = 1
    fprev2 = 1
    fnext1 = 0
    fiblist = []
    if n >= 1:
        fiblist.append(fprev1)
    if n >= 2:
        fiblist.append(fprev2)
    if n > 2:
        for value in range(n-2):
            fnext1 = fprev1 + fprev2
            fiblist.append(fnext1)
            fprev1 = fprev2
            fprev2 = fnext1
    if show_sequence == True:
        return fiblist
    else:
        return fiblist[n-1]
```

In [124]:

```
fib_iter(3)
```

Out[124]:

```
2
```

In [127]:

```
fib_iter(5, show_sequence = True)
```

Out[127]:

```
[1, 1, 2, 3, 5]
```

In [280]:

```
#(c)
def fib_recur(n, show_sequence = False):
    if show_sequence == False:
        if n == 1:
            return 1
        elif n == 2:
            return 1
        else:
            return fib(n-1) + fib(n-2)
    else:
        if n <= 1:
            return [1]
        elif n <= 2:
            return [1,1]
        else:
            a = fib(n-1, show_sequence = True)
            b = fib(n-2, show_sequence = True)
            return a + [a[-1] + b[-1]]
```

In [281]:

```
fib_recur(7, show_sequence = True)
```

Out[281]:

```
[1, 1, 2, 3, 5, 8, 13]
```

In [282]:

```
fib_recur(5)
```

Out[282]:

```
5
```

Question 4

(a)
If $f(x) := x^2 - r$ is our function
then $f'(x) = 2x$ is the derivative of the function.
Therefore, the Newton's method approximation is:

x_0 = a random guess.

$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$

$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - r}{2x_n}$ for $n \geq 2$

As $\lim_{n\to\infty}, x_n^2 - r \to 0$
Thus $(x_{n+1})^2 - r$ as $\lim_{n\to\infty} = x_n^2 - r = 0$ and x_n is the root of r.

In [269]:

```
#(b)
import numpy as np

def square_root(r):
    x0 = 0.1
    x = x0
    for item in np.arange(100):
        fx = x**2 - r
        dfx = 2*x
        if -0.1 < fx < 0.1:
            return x
        else:
            x = x - fx/dfx
```

In [271]:

```
square_root(1.4)
```

Out[271]:

```
1.1937187973877976
```

In [ ]: