1. Write a function, named get\_median(), that allows a user to enter as many numbers as desired, and then returns the \textit{median} number. That is, for a sequence of \textit{n} values, return the one that, when those values are placed in order, is in the middle position. The user should be allowed to enter as many values as desired; the user entry of \textquote{done} should indicate that the sequence of values is complete; non-numeric and non-\textquote{done} values should be skipped.

\textbf{Note:} You may use reasonable \textit{list} methods such as \texttt{sort()} and \texttt{append()}.

2. Provide short answers to the following questions:

(a) What do we mean when we say, \textquote{Binary search is a $O(lgn)$ algorithm}?
(b) What is the basic structure of a \textit{divide and conquer} algorithm?
(c) What is the \textit{Turing Test} meant to determine?
3. Consider the following module...

def some_func (a, b, x, y):
    try:
        x[a], x[b] = x[b], x[a]
    except:
        a = (a + 1) % len(x)
        b = (b + 1) % len(x)
        x[a] = x[b]
        x[b] = x[a]
    answer = 'baz'
    if x is y:
        answer += 'foo'
    if x == y:
        answer += 'quux'
    return answer

def main ():
    lstA = [-3, -5, -7, -9]
    lstB = [2, 4, 6, 8]
    msg = some_func(1, 3, lstA, lstB)
    print(lstA)
    print(lstB)
    print(msg)

    lstC = [10, 11, 9, 8]
    lstD = lstC
    msg = some_func(7, 5, lstC, lstD)
    print(lstC)
    print(lstD)
    print(msg)

    lstE = [3, 3, 8, 1]
    lstF = [3, 3, 8, 1]
    msg = some_func(1, 0, lstE, lstF)
    print(lstE)
    print(lstF)
    print(msg)

    if __name__ == '__main__':
        main()

What does it print?
Consider the following `main()` function, noting that it uses `Temp` objects that represent temperatures in Kelvin, Celsius, and Fahrenheit...

```python
def main():
    t = Temp(283, 'K')
    s = Temp(15, 'C')
    r = Temp(32, 'F')

    if t < s and t < r:
        coldest = t
    elif s < t and s < r:
        coldest = s
    elif r < t and r < s:
        coldest = r
    else:
        coldest = None

    print('The coldest is ' + str(coldest))
```

Write the methods to complete the `Temp` class, using the code above as an indicator of how those methods should behave. (Note that special method name for the less than operator is `__lt__`.)

```python
class Temp(object):
    k = None
    metric = None

    def __init__(self, temp, metric):
        self.metric = metric
        if metric == 'K':
            self.k = temp
        elif metric == 'C':
            self.k = temp + 273
        elif metric == 'F':
            self.k = ((temp - 32) * 5/9) + 273

    def get_K(self):
        return self.k
    def get_C(self):
        return self.k - 273
    def get_F(self):
        return (self.get_C() * 9/5) + 32
```
5. Write a function that checks if a square matrix of integers is a *partial magic square*. The function should return `True` if the values in each row and column of the matrix add up to the same total, and it should return `False` otherwise. This essentially checks if the matrix is a *magic square*, but it disregards the diagonals.

6. **Write a function**, named `find_substring()`, that takes two strings as parameters, and searching for instances of the first string (a *substring*) within the second string. For example, consider the following substring and string:

- **substring**: 'jump'
- **string**: 'The quick brown fox jumped over the lazy frog'

The function should return the starting index of the first instance of the substring within the string; the value `False` should be returned if the substring can be found. In the example above, the substring occurs at position 20 within the string, and so that value should be returned.