Question 1. [6 marks]

Beside each code fragment below, show the output that it would create. If it would generate an error say so, and give the reason why.

Part (a) [1 mark]

```
print 19 / 6
```

Part (b) [1 mark]

```
print 12 - 4.0*3 + 2
```

Part (c) [1 mark]

```
z = 3 * "Blah"
print z  + str(3) + "Blah"
```

Part (d) [1 mark]

```
ratio = 1.5433333
print "Ratio is %.4f", % (ratio)
```

Part (e) [1 mark]

```
s = "abc"
for c in s:
    piece = ""
    for i in range(4):
        piece = c * i + ":"
    print piece
```

Part (f) [1 mark]

```
s = "abc"
for i in range(2):
    piece = ""
    for c in s:
        piece = piece + str(i) + c * i
    print piece
```

Solution:

```
(a) 3
(b) 2.0 or 2 and some indication that it's a float.
(c) BlahBlahBlah3Blah
(d) Ratio is 1.453
(e)
aaa:
bbb:
ccc:
(f)
000
1a1b1c
2aa2bb2cc
```

Question 2. [8 MARKS]

Each of the code sequences below runs without error. Show the output for each. The next page is provided for your rough work.

Part (a) [2 MARKS]

```
morning = ["wake_up",["breakfast","drink"],"commute"]
afternoon = morning[:]
afternoon[2] = "work"
afternoon[1][0] = "lunch"
print morning
print afternoon
```

Solution:

```
['wake_up', ['breakfast', 'drink'], 'commute']
['wake_up', ['lunch', 'drink'], 'work']
```

Part (b) [2 MARKS]

```
morning = ["wake_up",["breakfast","drink"],"commute"]
afternoon = morning
afternoon[2] = "work"
afternoon[1][0] = "lunch"
print morning
print afternoon
```

Solution:

```
['wake_up', ['lunch', 'drink'], 'work']
['wake_up', ['lunch', 'drink'], 'work']
```

Part (c) [2 MARKS]

```
team = {1: "Anya", 23: "Yosuke", 34: ["Joseph"]}
new+team = {}
for key, value in team.items():
    new_team[key] = value
new_team[44] = "Gilbert"
team[34].append("Sahar")
print team
print new_team
```

Solution:

```
 {1: "Anya", 23: "Yosuke", 34: ["Joseph", "Sahar"]}
 {1: "Anya", 23: "Yosuke", 34: ["Joseph"], 44:"Gilbert"}
```

Part (d) [2 marks]

```
team = {1: "Anya", 23: "Yosuke", 34: ["Joseph"]}
new_team = team
new_team[44] = "Gilbert"
team[34].append("Sahar")
print team
print new_team
```

Solution:

```
{1: "Anya", 23: "Yosuke", 34: ["Joseph", "Sahar"], 44:"Gilbert"}
{1: "Anya", 23: "Yosuke", 34: ["Joseph", "Sahar"], 44:"Gilbert"}
```

Question 3. [12 marks]

8	1	6
3	5	7
4	9	2

Note that in the above magic square, each row, column and diagonal add to 15. We can use this to create a very basic algorithm to play tic tac toe. Tic tac toe is a game in which 1 player plays Xs in a 3 by 3 grid, and the other player plays Os. Each player can draw their symbol in one square of the grid, and when they do so the other player gets a turn. A player can only draw in empty squares. A player wins when they can have 3 of their symbols on either a row, column or diagonal. The magic square allows us to view this game differently. Each player chooses a a number from 1 to 9, and the first player that chooses 3 numbers that add exactly to 15 wins. When a player chooses a number, that number disappears.

An example of Tic tac toe were player 1 (the X player) wins:

X	O	
X	X	O
X		O

Part (a) [5 marks]

Write the following function according to its docstring:

```
def win(my_nums):
    '''Return True iff (list of ints) my_nums contains a set of 3 numbers that adds to 15.'''
```

Solution:

```
def win(my_nums):
    '''Return True iff my_nums contains a set of 3 numbers that adds to 15.'''
    for i in range(len(my_nums)):
        for j in range(i+1, len(my_nums)):
            for k in range(j+1, len(my_nums)):
                if (my_nums[i]+my_nums[j]+my_nums[k])==15:
                   return True
    return False
```

Part (b) [7 MARKS]

Given a correct implementation of wins, write the following function according to its docstring:

```
def chose_num(my_nums, rem_nums):
    '''Choose a number from (list of ints) rem_nums, remove that number
    from rem_nums, add that number to the (list of ints) my_nums, and
    return the number. If there is a number in rem_nums that can be added
    to my_nums so that my_nums contains three numbers that add to 15,
    then make sure that number is the one that is returned. Otherwise return
    an arbitrary number.'''
```

Solution:

```
def chose_num(my_nums, rem_nums):


    for i in range(len(rem_nums)):
        my_nums.append(rem_nums[i])
        if win(my_nums):
            return rem_nums.pop(i)
        my_nums.pop()
    my_nums.append(rem_nums[0])
    return rem_nums.pop(0)
```

Question 4. [5 marks]

Suppose you have a file `rainfalls.txt` that contains the rainfalls that occurred in a given town over the last years. Each day is represented by a separate line, and the rainfall is given in the format `11mm` where 11 can be replaced with any non-negative integer. Write code that will calculate the average amount of rainfall over the last year, and store it in a variable `rain_avg`.

Solution:

```
rain_file = open('rainfalls.txt','r')
rain_avg = 0
num_lines = 0
for line in file:
    rain_avg += int(line[:-2])
    num_lines+=1
rain_avg = float(rain_avg)/num_lines
```

Question 5. [5 marks]

Suppose you have a dictionary `sales_records` where they keys are employee names, and the values are lists whose elements are ints that represent the value of each individual sale made by an employee. Now, suppose we want quick access to the biggest sale made by each employee. To do this we create a new dictionary `biggest_sales` with the same keys, but whose values are ints that represent the biggest sale made by that employee. If an employee has made no sales, his biggest sale is considered to have a value of 0.

Write code to generate `biggest_sales`

Solution:

```
biggest_sales = {}
for key in sales_records:
    max = 0
    for i in sales_records[key]:
        if i>max:
           max = i
    biggest_sales[key]=max
```

Question 6. [7 marks]

Suppose that we are keeping track of test data for test subjects, and we bring them in for tests once a month. We don't want to keep our program consistently running, so we will use pickle to load our data. Suppose we have our data stored in a file called `data.pkl`. In the pickle file we have a dictionary called `study_results` where the keys are patient names, and the values are dictionaries where the keys are month names and the values are test results. Assume that cpickle has been imported.

Write the following function according to its docstring:

```
def update_data(month, test_results):
    '''Update study_results with the test results for the month
    (str) month. test_results is a dictionary where the keys are patient
    names, and the values are test results.'''
```

Solution:

```
def update_data(month, test_results):
    '''Update the pickle file with test results for the month
    (str) month. test_results is a dictionary where the keys are patient
    names, and the values are test results.'''

    pickle_file= open('data.pikl','r')
    study_results = cPickle.load(pickle_file)
    pickle_file.close()

    for key in test_results:
        study_results[key].update({month: test_results[key]})

    pickle_file= open('data.pikl','w')
    cPickle.dump(study_results,pickle_file)
```

Question 7. [11 MARKS]

Consider the case when we want to have mutable strings. One way to do this, is replace strings with lists, where every list element is a single character (that is, a one element string). Assume the following is a module called `mut_str.py`

Part (a) [4 MARKS]

Replace the pass statements with code that matches the docstring.

```
def create_mut_string(string):
    '''Given a regular string, return a list of individual characters that matches
    the contents of the string.'''
    pass
```

```
def mut_string_to_string(L):
    '''Given a list of chars L, returns a string which matches the contents
    of L.'''
    pass
```

```
def is_mutable_string(L):
    ''' Return True iff the contents of L are that of a mutable string.'''
    pass
```

Solution:

```
def create_mut_string(string):
    '''Given a regular string, return a list of individual characters that matches
    the contents of the string.'''
    L = []
    for char in string:
        L.append(char)
    return L

def mut_string_to_string(L):
    '''Given a list of chars L, returns a string which matches the contents
    of L.'''
    string = ''
    for i in L:
        string += i
    return L

def is_mutable_string(L):
    ''' Return True iff the contents of L are that of a mutable string.'''
    for i in L:
        if not ((type(i)==str) and (len(i)==1)):
            return False
    return True
```

Part (b) [7 MARKS]

Now say you wish to test your code using Nose: fill-in the following almost empty code of `test__mut_str.py` with your own tests so that when we run it, nose runs your tests successfully..

```
import nose
```

```
if __name__ == "__main__":
    nose.runmodule()
```

Question 8. [7 MARKS]

As we saw in class, sometimes we first write psuedocode before writing real code. Consider the following psuedocode for the following function foo, that takes a nested list L:

```
foo (L)
M=[]
for each sublist  i in L:
    append the maximum element in L[i] to M
return the median element of M.
```

Recall that the median means an element that has an equal number of elements above and below it. Anotherway to think about the median is the middle element in a sorted list. (You can assume that L has an odd number of sublists.)

Part (a) [5 MARKS]

Write python code to implement foo:

Solution:

```
def foo(L):
    M=[]
    for n in L:
        max=n[0]
        for i in n:
            if max < i:
                max = i
        M.append(max)
    M.sort()
    return M[len(M)/2]
```

Part (b) [2 MARKS]

Write a docstring for foo:

Solution:

```
"""Return the median element of a list of the maximum elements in each sublist of the nested list L.
Requires that all elements are comparable."""
```

Question 9. [13 MARKS]

Consider the following partially written class, that describes a painting. :

```
class Painting(object):
    '''An oil painting, with a name, a painter_name, an owner, a list of past owners, and a value.'''

    def __init__(self, name, painter):
        '''A new painting  with name s painted by the
        painter, no owner (or past owners) and a value of 0.''

        self.name = name
        self.painter = painter
        #Complete this.

    def get_value(self):
        return self.value()

    def add_owner(self, owner, value):
        '''Add str owner as the owner of this painting, and add the old owner to the list if past owners.
        Update the paintings value to the (int) value that the owner paid for it.'''
        #Write Me.
```

Part (a) [2 MARKS] Finish writing the constuctor (the `__init__` method).

Solution:

```
self.owner = ''
self.past_owners = []
self.value = 0
```

Part (b) [3 MARKS] Finish writing the `add_owner` method.

Solution:

```
   if self.owner != '':
      self.past_owners.append(self.owner)
   self.owner = owner
   self.value = value
```

Part (c) [2 MARKS] Create a `Painting` variable called `p1`. It's name is "The Milkmaid" and the painter is "Vermeer". Then call a method that represents what happens if "John" buys it for 100000000.

Solution:

```
    p1 = Painting("The Milkmaid", "Vermeer")
    p1.add_owner("John", 100000000)
```

Part (d) [2 MARKS] Write the following new method, to be added to the class.

```
def num_past_owners(self):
    '''Return the number of times the painting has been owned before.'''
```

Solution:

```
    return len(self.past_owners)
```

Part (e) [4 MARKS] Write the `__cmp__` method below, also to be added to the class. Define it so that if we call `sort` on a list of paintings, they will come out in order from cheapest (having the smallest value) to most expensive (having the largest value).

```
def __cmp__(self, other):
```

Solution:

```
def __cmp__(self, other):

    if self.value() < other.value():
        return -1
    elif self.value() == other.value():
        return 0
    else:
        return 1
```

Question 10. [6 marks]

Throughout this question, assume that we are sorting lists into non-desending order. **Do not guess** on the yes/no questions. There is a one-mark deduction for incorrect answers.

Part (a) [2 marks]

We are partly through sorting a list, and have completed 3 passes through the data. That is, we have called select 3 times. The list currently contains `[-1, 12, 17, 18, 17, 33, 98, 17]` Could we be doing selection sort? Circle one.

yes no

Solution: Yes.

Suppose it is insertion sort that we are doing. We have completed those 4 passes through, and the list contains: `[1, 64, 100, 110, 25, 234, 222, 109, 177]` Show the state of the list after the next (5th) pass through it.

Solution: `[1, 25 64, 100, 110, 234, 222, 109, 177]`

Part (b) [2 marks]

We are partly through sorting a different list, and have completed 4 passes through the data. The list currently contains `[-10, 0, 22, 27, 34, 64, 33, 28, 33]` Could we be doing insertion sort? Circle one.

yes no

Solution: Yes. The smallest three items are on the left.

Suppose it is selection sort that we are doing. We have completed those 4 passes through, and the list contains: `[1, 25, 64, 100, 110, 234, 222, 109, 177]` Show the state of the list after the next (5th) pass through it.

Solution: `[1, 25, 64, 100, 109, 110, 234, 222, 177]`

Part (c) [2 MARKS]

We are partly through sorting a different list, and have completed 4 passes through the data. The list currently contains `[2, 9, 12,19 98,65,21,65,46,2]` Could we be doing selection sort? Circle one.

yes no

Solution: No, if it were selection sort, the last element would not be 2.

Suppose it is bubble sort that we are doing. We have completed those 4 passes through, and the list contains: `[1, 23, 4, 56, 46, 45, 6, 65, 58, 70, 80, 90, 100]` Show the state of the list after the next (5^{th}) pass through it.

Solution: `[1, 4, 23, 46, 45, 6, 56, 58, 65, 70, 80, 90, 100]`

Question 11. [8 marks]

Don't guess. There is a 1-mark **deduction** for wrong answers to parts (a) through (d).

Part (a) [2 marks]

```
def f1(s):
    num_digits = 0
    for c1 in s:
        if c1.isdigit():
            num_digits = num_digits + 1
     return num_digits
```

Let n be the length of the string `s` passed to this function. Which of the following most accurately describes how the runtime of this function grow as n grows? Circle one.

(a) It grows linearly, like n does. (b) It grows quadratically, like n^2 does.

(c) It grows less than linearly. (d) It grows more than quadratically.

Part (b) [2 marks]

```
def f2(n):
    x = 0
    while n > 0:
        x = x + n * x
        for i in range(800):
            x = x + 1
        n = n - 1
    return x
```

Let n be the positive int passed to this function. Which of the following most accurately describes how the runtime of this function grow as n grows? Circle one.

(a) It grows linearly, like n does. (b) It grows quadratically, like n^2 does.

(c) It grows less than linearly. (d) It grows more than quadratically.

Part (c) [2 marks]

```
def f3(s):
    num = 0
    for c1 in s:
        for c2 in s:
            if c1 != c2:
                num = c1 - c2
    return num
```

Let n be the length of the string `s` passed to this function. Which of the following most accurately describes how the runtime of this function grow as n grows? Circle one.

(a) It grows linearly, like n does. (b) It grows quadratically, like n^2 does.

(c) It grows less than linearly. (d) It grows more than quadratically.

Part (d) [2 marks]

```
def f4(s):
    j=0
    L = []
    for i in range(len(s):
        L.append[0]
    while j < len(s):
        L[j] = L[j] + 1
        if L[j] == 3:
            L[j] = 0
            j = j + 1
        else:
            j = 0
```

Let n be the length of the string `s` passed to this function. Which of the following most accurately describes how the runtime of this function grow as n grows? Circle one.

(a) It grows linearly, like n does. (b) It grows quadratically, like n^2 does.

(c) It grows less than linearly. (d) It grows more than quadratically.

Solution:

```
(a) linear
(b) linear
(c) quadratic
(d) more than quadratic.
```

Short Python function/method descriptions:

```
__builtins__:
  len(x) -> integer
    Return the length of the list, tuple, dict, or string x.
  max(L) -> value
    Return the largest value in L.
  cmp(x, y) -> integer
    Return negative if x<y, zero if x==y, positive if x>y.
  min(L) -> value
    Return the smallest value in L.
  open(name[, mode]) -> file object
    Open a file.  Legal modes are "r" (read), "w" (write), and "a" (append).
  range([start], stop, [step]) -> list of integers
    Return a list containing the integers starting with start and ending with
    stop - 1 with step specifying the amount to increment (or decrement).
    If start is not specified, the list starts at 0.  If step is not specified,
    the values are incremented by 1.
  raw_input([prompt]) -> string
    Read a string from standard input.  The trailing newline is stripped.
cPickle:
  dump(obj, file)
    Write an object in pickle format to the given file.
  load(file) --> object
    Load a pickle from the given file
dict:
  D[k] --> value
    Return the value associated with the key k in D.
  k in d --> boolean
    Return True if k is a key in D and False otherwise.
  D.get(k) -> value
    Return D[k] if k in D, otherwise return None.
  D.keys() -> list of keys
    Return the keys of D.
  D.values() -> list of values
    Return the values associated with the keys of D.
  D.items() -> list of (key, value) pairs
    Return the (key, value) pairs of D, as 2-tuples.
file (also called a "reader"):
  F.close()
    Close the file.
  F.read([size]) -> read at most size bytes, returned as a string.
    If the size argument is negative or omitted, read until EOF (End
    of File) is reached.
  F.readline([size]) -> next line from the file, as a string. Retain newline.
    A non-negative size argument limits the maximum number of bytes to return (an incomplete
    line may be returned then).  Return an empty string at EOF.
float:
  float(x) -> floating point number
    Convert a string or number to a floating point number, if possible.
int:
  int(x) -> integer
    Convert a string or number to an integer, if possible.  A floating point
    argument will be truncated towards zero.
list:
  x in L --> boolean
    Return True if x is in L and False otherwise.
  L.append(x)
    Append x to the end of the list L.
  L.index(value) -> integer
    Returns the lowest index of value in L.
  L.insert(index, x)
    Insert x at position index.
  L.remove(value)
    Removes the first occurrence of value from L.
  L.reverse()
    Reverse *IN PLACE*
  L.sort()
    Sorts the list in ascending order.
random:
  randint(a, b)
    Return random integer in range [a, b], including both end points.
str:
  x in s --> boolean
    Return True if x is in s and False otherwise.
  str(x) -> string
    Convert an object into its string representation, if possible.
  S.count(sub[, start[, end]]) -> int
    Return the number of non-overlapping occurrences of substring sub in
    string S[start:end].  Optional arguments start and end are interpreted
    as in slice notation.
  S.find(sub[,i]) -> integer
    Return the lowest index in S (starting at S[i], if i is given) where the
    string sub is found or -1 if sub does not occur in S.
  S.index(sub) -> integer
    Like find but raises an exception if sub does not occur in S.
  S.isdigit() -> boolean
    Return True if all characters in S are digits and False otherwise.
  S.lower() -> string
    Return a copy of the string S converted to lowercase.
  S.lstrip([chars]) -> string
    Return a copy of the string S with leading whitespace removed.
    If chars is given and not None, remove characters in chars instead.
  S.replace(old, new) -> string
    Return a copy of string S with all occurrences of the string old replaced
    with the string new.
  S.rstrip([chars]) -> string
    Return a copy of the string S with trailing whitespace removed.
    If chars is given and not None, remove characters in chars instead.
  S.split([sep]) -> list of strings
    Return a list of the words in S, using string sep as the separator and
    any whitespace string if sep is not specified.
  S.strip() -> string
    Return a copy of S with leading and trailing whitespace removed.
  S.upper() -> string
    Return a copy of the string S converted to uppercase.
```