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 -10 / 4
```

Part (b) [1 MARK]

```
print 24 - 12 / 4 - 2
```

Part (c) [1 MARK]

```
z = "over" + "and"
print "..." + z * 3 + "..."
```

Part (d) [1 MARK]

```
avg = 87.28196
print "Average is %.2f percent", % (avg)
```

Part (e) [1 MARK]

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

Part (f) [1 MARK]

```
s = "no"
for i in range(4):
    piece = ""
    for c in s:
        piece = piece + c * i + ":"
    print piece
```

Solution:

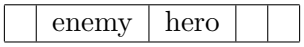
```
(a) -3
(b) 19
(c) ...overandoverandoverand...
(d) Average is 87.28 percent
(e)
:y:yy:yyy:
:a:aa:aaa:
(f)
::
n:o:
nn:oo:
nnn:ooo:
```

Question 2. [4 MARKS]

Assume we have a game board consisting of n positions, and two objects: the hero and the enemy. (The game board has only one row.) At any point in time, the hero and the enemy will be a certain number of positions apart. Consider the number of spaces that the hero must move to land on the enemy's position, as a fraction of the most the hero might have to move, $n - 1$. That fraction is the **distance** between the hero and the enemy. It is positive if the enemy is to the right of the hero, negative if the enemy is to the left of the hero, and exactly 0 if the hero and enemy are on the same position.

For example, consider the following game board where $n = 6$:

The distance here is 3/5, which is 0.6. For this game board with $n = 5$:



The distance is -0.25.

Write the following function according to its docstring.

```
def distance (game_board_size, hero, enemy):
  '''game_board_size is a positive int giving the number of positions on a
  game board. The positions are numbered from 0 to game_board_size-1. hero and
  enemy are ints giving valid hero and enemy positions on the game board,
  i.e., they are between 0 and game_board_size-1. Return a float between -1.0
  and 1.0 giving the distance between hero and enemy.'''
```

Solution:

```
def distance (game_board_size, hero, enemy):
  '''game_board_size is a positive int giving the number of positions on a
  game board. The positions are numbered from 0 to game_board_size-1. hero and
  enemy are ints giving valid hero and enemy positions on the game board,
  i.e., they are between 0 and game_board_size-1. Return a float between -1.0
  and 1.0 giving the distance between hero and enemy.'''

  return (enemy - hero) / float(game_board_size - 1)
```

Question 3. [14 MARKS]

Consider strings where each character is one of the following three characters:

- `g` for "good"
- `b` for "bad"
- `u` for "unusable"

The **goodness** of a string follows these two rules:

- The goodness of a string containing one or more **u**'s is 0.
- Otherwise, the goodness of a string is equal to the number of **g**'s in the string.

For example, the goodness of `"gbbgb"` is 2, and the goodness of `"gubgb"` is 0. Remember that a slice of a string is a string, and so the goodness of a slice is defined by these same rules.

Any string has more than one slice you can take from it (unless it is a very short string). Each slice may have a different goodness value. Suppose we are only interested in slices of a particular length, and we want to find the one with the highest goodness. In part (c), you will write the following function according to its docstring.

```
def best_slice (s, k):
  '''s is a str, k is an integer such that 0 <= k <= len(s). Return the
  starting index of the length-k slice of s with the highest goodness.
  If k is zero, return -1.'''
```

Part (a) [3 MARKS]

Assume that we want to test `best_slice`. Describe three test cases that each test different "categories" of inputs. To describe each test case, give the `str` and `int` that you would pass to `best_slice`, the return value you expect, and the purpose of the test case. Do **not** write any code. We have given you one test case as an example; add three more.

Value of `s`	Value of `k`	Return value	Purpose of this test case
`""`	`0`	`-1`	empty string

Solution:

```
Lots of options:
+String with a u
+String with all b's
+String with one b and one u
+String with one g and one u
```

Part (b) [3 MARKS]

Identify a function that would be a useful helper for `best_slice`. Define it here, including a docstring.

Part (c) [8 MARKS]

Now write function `best_slice`. You do not need to repeat the docstring.

```
def best_slice (s, k):
```

Solution:

```
def goodness(s):
  if s.count ('u') > 0:
    return 0
  else:
    return s.count('g')

def best_slice (s, k):
  '''s is a str, k is an integer such that 0 <= k <= len(s). Return the
  starting index of the length-k slice of s with the highest goodness.
  If k is zero, return -1.'''

  stop = len(s) - k
  best_start = -1
  best_goodness = 0
  for i in range(stop + 1):
    cur_slice = s[i:i+k]
    slice_goodness = goodness(cur_slice)
    if slice_goodness > best_goodness:
      best_start = i
      best_goodness = slice_goodness
  return best_start
```

Question 4. [8 MARKS]

A **compressed string** is a string where each alphabetic character is preceded by a single digit indicating the number of times that the character should be entered in the uncompressed version of the string. As a special case, if a character is to be entered only once, the compressed string may include the character `c` instead of `1c`.

For example:

- the compressed string `2a5b1c` is uncompressed to `aabbbbbc`
- The compressed string `1ab2c` is uncompressed to `abcc`
- The compressed string `a9b3bc` is uncompressed to `abbbbbbbbbbbbc`

Write the following function according to its docstring.

```
def uncompress (s):
  '''s is a compressed string. Return the corresponding uncompressed
  string.'''
```

Solution:

```
def uncompress (s):
  '''s is a compressed string. Return the
  corresponding uncompressed string.'''

  i = 0
  new_s = ''
  number = 1
  while i < len(s):
    if s[i].isdigit():
      number = int(s[i])
    else:
      character = s[i]
      new_s += character * number
      number = 1
    i += 1
  return new_s
```

Question 5. [8 MARKS]

We can use a Python list to represent a table or matrix with rows and columns as follows. Each element of the list is itself a list, containing the items from one row of the table. For example, this table

5	10	15
1	2	3
10	20	30
2	4	6

is represented by this list in Python: `[[5, 10, 15], [1, 2, 3], [10, 20, 30], [2, 4, 6]]`. Notice that all the sublists have the same length.

Write the following function according to its docstring.

```
def sums(L):
    '''L is a list of lists of ints.  All of L's sublists have the same length.
    Return a new list that contains the sums of all the "columns" of L.
    For example, for the list [[5, 10, 15], [1, 2, 3], [10, 20, 30], [2, 4, 6]],
    return [18, 36, 54].'''
```

Solution:

```
    ans = []
    for column in range(len(L[0])):
        sum = 0
        for row in range(len(L)):
            sum += L[row][column]
        ans.append(sum)
    return ans
```

Question 6. [8 MARKS]

A **question bank** is a text file consisting of one or more multiple choice questions. Each multiple choice question consists of the following lines, in order:

- A line containing only the characters `MC`
- One line, containing the question
- One or more lines, containing possible answers to the question

For example, here is a question bank with two questions. Notice that the number of answers to each question may be different.

```
MC
The instructor that has taught CSC108 the most is:
Diane
Dan
Steve
MC
The funniest CSC108 instructor is:
Dan
Daniel
Dan Z
Daniel Z
```

We want a function that will read such a file and produce a list of strings, each of which contains the full text of one of the multiple-choice questions (including the answers). For example, the list produced from the file above would be:

```
['The instructor that has taught CSC108 the most is:\nDiane\nDan\nSteve\n',
'The funniest CSC108 instructor is:\nDan\nDaniel\nDan Z\nDaniel Z\n']
```

(Notice that the newlines are preserved.) Such a list could be used to choose random questions and generate a test, although you will not be writing any such code today.

On the following page, write function `read_mcqs` according to its docstring.

```
def read_mcqs (f):
  '''f is an open question bank file with n >= 1 multiple choice questions.
  Return a list of n strings, each of which contains the entire question
  text and responses for one question.'''
```

Solution:

```
def read_mcqs (f):
  '''f is an open question bank file with n >= 1 multiple choice questions.
  Return a list of n strings, each of which contains the entire question
  text and responses for one question.'''

  strs = []
  line = f.readline().strip()
  while line == 'MC': # or 'while line:'
    cur = ''
    line = f.readline().strip()
    while line and line != 'MC':
      cur = cur + line + '\n'
      line = f.readline().strip()
    strs.append (cur)
  return strs
```

Question 7. [6 MARKS]

Define the **dictionary difference** between two dictionaries to be a new dictionary containing keys (and their values) that occur in one but not both of the dictionaries. If any key is in both original dictionaries, but with different values, it does not appear in the new dictionary.

For example, if we have these two dictionaries:

```
d1 = {1: 'a', 2: 'b', 3: 'c', 9: 'h'}
d2 = {1: 'k', 6: 'f', 2: 'g', 5: 'e', 8: 'c'}
```

their difference is `{3: 'c', 9: 'h', 6: 'f', 5: 'e', 8: 'c'}`.

Write the following function according to the docstring and the definition above.

```
def dict_diff(d1, d2):
    '''d1 and d2 are dicts. Return a new dict which is the dictionary
    difference between d1 and d2.'''
```

Solution:

```
d3 = {}
for k1 in d1:
    if k1 not in d2:
        d3[k1] = d1[k1]
for k2 in d2:
    if k2 not in d1:
        d3[k2] = d2[k2]
return d3
```

Question 8. [8 MARKS]

In some games there are more than two dice, and the number of dice to roll can vary. We need a program that asks the user for a positive integer indicating how many dice to roll. If 0 is entered, the program stops; otherwise, the specified number of dice are rolled and the process is repeated. Here is a sample execution.

```
Roll how many dice (0 to stop)? 4
2 2 4 5
Roll how many dice (0 to stop)? 2
1 6
Roll how many dice (0 to stop)? 1
5
Roll how many dice (0 to stop)? 0
```

This question continues on the next page.

Below are 9 lines of code in random order. Your task is to use each of these lines **at least once**, and add the proper indentation, to arrive at a program that implements the above description. Do **not** add new code; simply copy and rearrange the given code.

- `counter = counter + 1`
- `counter = 0`
- `while num_dice > 0:`
- `num_dice = int(raw_input ("Roll how many dice (0 to stop)? "))`
- `while counter < num_dice:`
- `print random.randint (1, 6),`
- `print ""`
- `if __name__ == '__main__':`
- `import random`

Assemble the complete program here:

Solution:

```
import random

if __name__ == '__main__':
  num_dice = int(raw_input ("Roll how many dice (0 to stop)? "))
  while num_dice > 0:
    counter = 0
    while counter < num_dice:
      print random.randint (1, 6),
      counter = counter + 1
    print ""
    num_dice = int(raw_input ("Roll how many dice (0 to stop)? "))
```

Question 9. [6 MARKS]

A **flight dictionary** is a dictionary where:

- The keys are strings. Each key is the name of a city.
- The values are lists of strings. For a key `k`, the value is the list of cities reachable on a direct flight from `k`. Every city in the list is also a key in the dictionary.

Here is an example flight dictionary:

```
flights = {'Montreal': ['Toronto', 'Tampa Bay'],
           'Toronto': ['Montreal', 'Tampa Bay'],
           'Tampa Bay': ['Atlanta', 'Toronto'],
           'Atlanta': ['Tampa Bay']}
```

A **one-hop** between two cities `a` and `b` exists when it is possible to get from city `a` to city `b` by taking two flights: one from city `a` to some intermediate city `c`, and then a second flight from `c` to the final destination `b`. For example, in the above dictionary, there is a one-hop from Toronto to Atlanta (through Tampa Bay), and a one-hop from Montreal to Atlanta (again through Tampa Bay), but there is **not** a one-hop from Atlanta to Montreal.

Write the following function according to its docstring.

```
def one_hop (flights, city1, city2):
  '''flights is a flight dictionary. city1 and city2 are keys in flights.
  Return True if and only if there is at least one one-hop from city1 to
  city2.'''
```

Solution:

```
def one_hop (flights, city1, city2):
  '''flights is a flight dictionary. city1 and city2 are keys in flights.
  Return True if and only if there is at least one one-hop from city1 to
  city2.'''

  potentials = flights[city1]

  for p in potentials:
    if city2 in flights[p]:
      return True
  return False
```

Question 10. [6 MARKS]

Throughout this question, assume that we are sorting lists into non-descending 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 4 passes through the data. The list currently contains `[10, 20, 30, 40, 16, 94, 8, 22]` Could we be doing selection sort? Circle one.

yes no

Solution: No. Selection sort is "selective" (or picky), so on each pass it chooses the next smallest item to put into place. The first four items would thus be the overall smallest four items, which they are not.

Suppose it is insertion sort that we are doing. We have completed those 4 passes through, and the list contains: `[10, 20, 30, 40, 16, 94, 8, 22]` Show the state of the list after the next (5^{th}) pass through it.

Solution: `[10, 16, 20, 30, 40, 94, 8, 22]`

Part (b) [2 MARKS]

We are partly through sorting a different list, and have completed 4 passes through the data. The list currently contains `[11, 22, 33, 44, 56, 81, 48, 72]` Could we be doing insertion sort? Circle one.

yes no

Solution: Yes.

Suppose it is selection sort that we are doing. We have completed those 4 passes through, and the list contains: `[11, 22, 33, 44, 56, 81, 48, 72]` Show the state of the list after the next (5^{th}) pass through it.

Solution: `[11, 22, 33, 44, 48, 56, 81, 72]`

Part (c) [2 MARKS]

We are partly through sorting a different list, and have completed 4 passes through the data. The list currently contains `[0, 1, 2, 3, 60, 70, 20, 40, 50, 30]` Could we be doing selection sort? Circle one.

yes no

Solution: Yes.

Suppose it is bubble sort that we are doing. We have completed those 4 passes through, and the list contains: `[0, 1, 2, 3, 60, 70, 20, 40, 50, 30]` Show the state of the list after the next (5^{th}) pass through it.

Solution: `[0, 1, 2, 3, 20, 60, 70, 30, 40, 50]`

Question 11. [9 MARKS]

Consider the following class:

```
class Member(object):
    '''A member of facebook, with a str name, a str status, and zero or
    more str friends.'''

    def __init__(self, s):
        '''A new member with name s, no friends, and status "no status yet".'''

        self.name = s
        self.friends = []
        self.status = "no status yet"

    def __str__(self):
        '''Return a str describing this member.'''

        result = "Name: %s; Status: %s" % (self.name, self.status)
        if self.friends == []:
            result += "; no friends yet"
        else:
            result += ", and friends: "
            for person in self.friends:
                result += person + ", "
            # Strip off the final comma.
            result = result[:-2]
        return result

    def add_friend(self, friend):
        '''Add str friend as a friend of this member.'''

        self.friends.append(friend)

    def update_status(self, new_status):
        '''Update this member's status to str new_status.'''

        self.status = new_status
```

Part (a) [2 MARKS] Create a `Member` variable called `m1`. His name is "Danny Z", his status is "torturing 108 students" and he has one friend named "Diane".

Solution:

```
m1 = Member("Dan")
m1.update_status("torturing 108 students")
m1.add_friend("Diane")
```

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

```
def friendliness(self):
    '''Return the number of friends this member has.'''
```

Solution:

```
    return len(self.friends)
```

Part (c) [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 members, they will come out in order from least friendly (having the fewest friends) to most friendly (having the most friends).

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

Solution:

```
def __cmp__(self, other):

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

Part (d) [1 MARK] Assume both functions, `friendliness` and `__cmp__`, have been implemented correctly. Suppose you have another variable of type `Member` called `m2`. His name is Steve, and you don't know how many friends he has. Write an expression that is `True` if Dan is more friendly than Steve and `False` otherwise.

Solution:

```
print m1 > m2
```

Question 12. [7 MARKS]

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

Part (a) [2 MARKS]

```
def f1(s):
    for i in range(len(s)):
        for j in range(10000):
            print s[i] * j
```

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):
    sum = 0
    while n > 0:
        sum = sum + n ** 2
        n = n / 2
    return sum
```

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 = num + 1
    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.

Solution:

```
(a) linear
(b) less than linear
(c) quadratic
```

Part (d) [1 MARK] The `int` that is returned from `f3("abcdefghijklmnopqrstuvwxyz")` is: _____

Solution: 25

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.
  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.
```