Question 1. [6 MARKS]

Beside each code fragment in the table below, write what is printed when the code fragment is executed. If the code would cause an error, write ERROR and give a brief explanation.

Code	**Output or Cause of Error**

`term = 'Fall' + 2017`
`print(term)`
	`Error, cannot concatenate str to int`

`msg = 'ten'`
`value = int(msg)`
`print(value)`
	`Error, cannot convert non-digits to int`

`for value in range(10, 3, -2):`
`    print(value)`
	`10`
	`8`
	`6`
	`4`

`L = ['a', 'b']`
`L = L.extend(['c'])`
`print(L)`
	`None`

`foods = ['fig', 'egg', 'yam', 'pie']`
`fave = foods[1:][1]`
`print(fave)`
	`yam`

`result = 'assessment'.count('ss', 2)`
`print(result)`
	`1`

Question 2. [2 marks]

Complete the docstring examples with arguments that will cause the function calls to return the values shown.

```
def midterm_function(s: str, i: int) -> bool:
    """
    Precondition: len(s) >= 1 and  0 <= i < len(s)

    # first argument: any str that has only digits from index i to end
    # second argument: 0 <= i < len(s)

    # There are many possible solutions.  Here is an example:
    >>> midterm_function('csc108', 3)
    True
    >>> midterm_function('123 go!', 1)
    False
    """

    return s[i:].isdigit()|
```

Question 3. [3 marks]

Step 1 of the Function Design Recipe (docstring examples) has been completed for the function `remove_occurrence`. Complete steps 2 and 3 of the Function Design Recipe: Fill in the function header (including the type contract) and write a good description.

Do not write the function body. Do not include preconditions.

```
def remove_occurrence(s: str, substr: str) -> str
    """Return a version of s in which the first occurence of substr has been
    removed.  If substr does not occur in s, return s.

    >>> remove_occurrence('cats scat', 'cat')
    's scat'
    >>> remove_occurrence('abcd', 'bc')
    'ad'
    >>> remove_occurrence('happy', 'day')
    'happy'
    """

    # DO NOT WRITE THE BODY OF THIS FUNCTION
```

Question 4. [4 marks]

Complete the following function according to its docstring.

```
def cooking_time(weight: float, stuffed: bool) -> int:
    """Return the cooking time (in minutes) for a turkey of a given weight
    (in pounds) that may or may not be stuffed, according to the times in
    the following table:

    weight of turkey                  cooking time when not stuffed
    ----------------                  -----------------------------
    under 14 pounds                   195 minutes
    14 to 20 pounds, inclusive        240 minutes
    over 20 pounds                    270 minutes

    Add 30 minutes to the cooking time when the turkey has been stuffed.

    Precondition: weight > 0

    >>> cooking_time(18.5, False)
    240
    >>> cooking_time(13.3, True)
    225
    >>> cooking_time(14.0, True)
    270
    """

    if weight < 14:
        result = 195
    elif weight <= 20:  # cannot just write: if weight <= 20:
        result = 240
    else:
        result = 270

    if stuffed:
        result = result + 30

    return result
```

Question 5. [5 marks]

Complete the following function according to its docstring.

```
def upper_lower_difference(s: str) -> int:
    """Return the difference between the number of uppercase and lowercase
    letters in s (the number of uppercase minus the number of lowercase).

    >>> upper_lower_difference('Hello99')
    -3
    >>> upper_lower_difference('LISTEN')
    6
    >>> upper_lower_difference('123HiLo')
    0
    """

    upper_count = 0
    lower_count = 0

    for ch in s:
        if ch.isupper():   # ch in 'ABCDE...'
            upper_count = upper_count + 1
        elif ch.islower():  # ch in 'abcde...'
            lower_count = lower_count + 1

    return upper_count - lower_count
```

Question 6. [3 marks]

Fill in the box with the while loop condition required for the function to work as described in its docstring.

```
def find_uppercase_vowel(msg: str) -> int:
    """Return the index of the first uppercase vowel (A, E, I, O, U) in msg,
    or the length of msg if it does not contain any uppercase vowels.

    >>> find_uppercase_vowel('CATS')
    1
    >>> find_uppercase_vowel('PYTHON')
    4
    >>> find_uppercase_vowel('aBCDe')
    5
    """


    i = 0
    while i < len(msg) and msg[i] not in 'AEIOU':
        i = i + 1
    return i
```