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**

`s = 'two'`
`v = int(s)`
`print(v)`
	`Error, cannot convert non-digits to int`

`x = '1' + 2`
`print(x)`
	`Error, cannot concatentate str to int`

`list1 = [1, 2, 3, 4]`
`element = list1[1:][1]`
`print(element)`
	`3`

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

`for value in range(7, 2, -2):`
`    print(value)`
	`7`
	`5`
	`3`

`result = 'banana'.find('an', 2)`
`print(result)`
	`3`

Question 2. [2 marks]

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

```
def midterm_function(s1: str, s2: str) -> bool:
    """
    Precondition: len(s1) >= 1 and len(s2) >= 1

    # first argument: any str that starts with a digit
    # second argument: any str that ends with a letter

    # There are many possible solutions.  Here is an example:
    >>> midterm_function('416', 'Toronto')
    True
    >>> midterm_function('YYZ', '6ix')
    False
    """

    return s1[0].isdigit() and s2[-1].isalpha()
```

Question 3. [3 marks]

Step 1 of the Function Design Recipe (docstring examples) has been completed for the function `repeat_first_letter`. 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 repeat_first_letter(s: str, count: int) -> str:
    """Return a new string with the first character from s repeated count
    times followed by the remaining characters in s.

    >>> repeat_first_letter('abc', 2)
    'aabc'
    >>> repeat_first_letter('', 5)
    ''
    >>> repeat_first_letter('CS', 4)
    'CCCCS'
    """

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

Question 4. [4 marks]

Complete the following function according to its docstring.

```
def practice_time(age: int, beginner: bool) -> int:
    """Return the practice time (in minutes) for a player of the given age
    who may or may not be a beginner player, according to the practice
    times in the following table:

    age of player              practice time for a beginner player
    -------------              -----------------------------------
    under 6 years              30 minutes
    6 to 8 years, inclusive    50 minutes
    over 8 years               75 minutes

    Add 15 minutes to the practice time for a player who is not a beginner.

    Precondition: age >= 0

    >>> practice_time(8, True)
    50
    >>> practice_time(9, True)
    75
    >>> practice_time(7, False)
    65
    """

    if age < 6:
        result = 30
    elif age <= 8:  # cannot instead just write: if age <= 8:
        result = 50
    else:
        result = 75

    if not beginner:
        result = result + 15

    return result
```

Question 5. [5 marks]

Complete the following function according to its docstring.

```
def has_letter_cases(s: str) -> bool:
    """Return True if and only if s contains at least one lowercase letter
    and at least one uppercase letter.

    >>> has_letter_cases('abcDEF')
    True
    >>> has_letter_cases('abc123')
    False
    >>> has_letter_cases('ABCXYZ')
    False
    """

    lowercase = False
    uppercase = False

    for ch in s:
        if ch.islower():      # ch in 'abcde...'
            lowercase = True
        elif ch.isupper():    # ch in 'ABCDE...'
            uppercase = True

    return uppercase and lowercase
```

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_first_fluffy(msg: str) -> int:
    """Return the index of the first fluffy character (f, l, u, y) in msg,
    or the length of msg if it does not contain any fluffy characters.
    Fluffy characters are those that appear in the word 'fluffy'.

    >>> find_first_fluffy('firefly')
    0
    >>> find_first_fluffy('cats')
    4
    >>> find_first_fluffy('Fully')
    1
    """

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