Do not turn this page until you have received the signal to start.

(Please fill out the identification section above, write your name on the back of the test, and read the instructions below.)

Good Luck!

This midterm consists of 5 questions on 6 pages (including this one). When you receive the signal to start, please make sure that your copy is complete.

- Comments are not required except where indicated, although they may help us mark your answers.

- No error checking is required: assume all user input and all argument values are valid.

- If you use any space for rough work, indicate clearly what you want marked.

- You may use a pencil; however, work written in pencil will not be considered for remarking.

# 1: _____/ 5
# 2: _____/ 2
# 3: _____/ 4
# 4: _____/ 5
# 5: _____/ 4

TOTAL: _____/20

Total Pages = 6
Question 1. [5 marks]

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

tweet1 = '#uoft_cs Turing award winner Steve Cook'
tweet2 = 'Want cheap snacks? Visit @cssu office in BA2283'

<table>
<thead>
<tr>
<th>Code</th>
<th>Output or Cause of Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>print('Turing' in tweet1)</td>
<td></td>
</tr>
<tr>
<td>print('in' in tweet1 and 'in' in tweet2)</td>
<td></td>
</tr>
<tr>
<td>print(tweet1[1] + tweet2[-3])</td>
<td></td>
</tr>
<tr>
<td>print(tweet1[tweet2[-1]])</td>
<td></td>
</tr>
<tr>
<td>print(tweet2[tweet2.find('@'):])</td>
<td></td>
</tr>
</tbody>
</table>

Question 2. [2 marks]

Complete the example function calls by adding arguments that result in the return values shown. (For each example call, there are several correct answers, and providing any one of them will earn full marks.)

def mystery(L):
    """ (list of str) -> str
    >>> mystery(             )
    'zz'
    >>> mystery(             )
    'acdfgi'
    """
    result = ''
    for item in L:
        result = result + item[0] + item[-1]
    return result
Question 3.  [4 marks]

Read the function header and body and then complete the docstring. Give a meaningful function name, the type contract, the description, and two examples that return different values.

```python
def ______________ (s):
    """
    i = len(s) - 1
    x = ''
    while i >= 0:
        if s[i].isdigit():
            x = s[i]
        i = i - 1
    return x
    """
```

i = len(s) - 1
x = ''
while i >= 0:
    if s[i].isdigit():
        x = s[i]
    i = i - 1
return x
Question 4. [5 marks]

Some programming languages use the *camel case* naming style for variable and function names. In this style, words in a name after the first word start with an uppercase letter and all other letters are lowercase (e.g., *camelCase*). Python, on the other hand, uses the *pothole case* naming style, where all letters are lowercase, with an underscore between each word and digit (e.g., *pothole_case*). Complete this function according to its docstring description.

```python
def camel_to_pothole(camel):
    """ (str) -> str

    Precondition: len(camel) >= 1 and camel[0].isalpha() and camel.isalnum()

    Return a new string with camel converted to the pothole case naming style.

    >>> camel_to_pothole('computerScience')
    'computer_science'
    >>> camel_to_pothole('numVowels30')
    'num_vowels_3_0'
    """
```

```
Question 5.  [4 marks]

A mini-tweet is a string that is 1 to 10 characters long. Using the constant MAX_LENGTH when possible, complete the function according to its docstring description.

MAX_LENGTH = 10

```python
def split_into_minitweets(s):
    """ (str) -> list of str

    Precondition: len(s) >= 1

    Return a list containing the mini-tweets in s. Maximize the length of each mini-tweet, so that each mini-tweet (except possibly the last) has a length equal to MAX_LENGTH.
    """

>>> s = 'code.org'
>>> split_into_minitweets(s)
['code.org']
>>> s = 'Do not just play on your phone - program it. - @BarackObama'
>>> split_into_minitweets(s)
['Do not just', 'play on ', 'your phone', ' - program', ' it. - @BarackObama']
```

```
Short Python function/method descriptions:

__builtins__:

- **int**(x) -> int
  - Convert x to an integer, if possible. A floating point argument will be truncated towards zero.
- **len**(x) -> int
  - Return the length of list, tuple, or string x.

- **print**(value) --> NoneType
  - Prints the values.
- **range**([start], stop, [step]) -> list-like-object of int
  - Return 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 sequence starts at 0. If step is not specified, the values are incremented by 1.
- **str**(x) -> str
  - Return an object converted to its string representation, if possible.

str:

- **x in s** --> bool
  - Produce True if and only if x is in s.
- **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]) -> int
  - 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.isalpha()** --> bool
  - Return True if and only if all characters in S are alphabetic and there is at least one character in S.
- **S.isalnum()** --> bool
  - Return True if and only if all characters in S are alphanumeric and there is at least one character in S.
- **S.isdigit()** --> bool
  - Return True if and only if all characters in S are digits and there is at least one character in S.
- **S.islower()** --> bool
  - Return True if and only if all cased characters in S are lowercase and there is at least one cased character in S.
- **S.isupper()** --> bool
  - Return True if and only if all cased characters in S are uppercase and there is at least one cased character in S.
- **S.lower()** --> str
  - Return a copy of the string S converted to lowercase.
- **S.replace**(old, new) --> str
  - Return a copy of string S with all occurrences of the string old replaced with the string new.
- **S.upper()** --> str
  - Return a copy of the string S converted to uppercase.

list:

- **x in L** --> bool
  - Produce True if and only if x is in list L.
- **L.append**(object) --> NoneType
  - Append object to end of list L.
- **L.extend**(iterable) --> NoneType
  - Extend list L by appending elements from the iterable. Strings and lists are iterables whose elements are characters and list items respectively.