CS206 - HW1

In this assignment, we are going to implement the "statement coverage" in software testing

Description

Our example programs are `capitalize.py` and `bubblesort_recursive.py` with corresponding inputs `inputs_capitalize.txt` and `inputs_capitalize.txt`. We are going to write the testing program which will run the main parograms above and generate the statement coverage testing result.

Environment

```
Python 3.9.6
```

Requirements

Please run this command first:

```
pip install -r requirements.txt
```

or

```
pip3 install -r requirements.txt
```

Test Programms

```
* `test_prog_capitalize.py`: testing program for `capitalize.py`
* `test_prog_bubblesort.py`: testing program for `bubblesort_recursive.py`
```

RUN

Test `capitalize.py`

Main `capitalize.py` code snippet:

```python
def capitalize(sentence: str) -> str:
    """
    Capitalizes the first letter of a sentence or word.

    """
    from string import ascii_lowercase, ascii_uppercase
    if not sentence:
        return ""

    # Create a dictionary that maps lowercase letters to uppercase letters
    # Capitalize the first character if it's a lowercase letter
    # Concatenate the capitalized character with the rest of the string
    lower_to_upper = dict(zip(ascii_lowercase, ascii_uppercase))
    return lower_to_upper.get(sentence[0], sentence[0]) + sentence[1:]
```

Inputs `inputs_capitalize.txt` code snippet:

```
hello world
python
123test
33333
```

Command for running testing:

```
python3 test_prog_capitalize.py inputs_capitalize.txt
```

The above command will generate the expeted output from main function and print out the statement coverage:

```
====================================================
Input: hello world -> Output: Hello world
++++++++++++++++++++++++++++++++++++++++++++++++++++
#   1 def capitalize(sentence: str) -> str:
#   2 """
#   3 Capitalizes the first letter of a sentence or word.
#   4
#   5 """
    6 from string import ascii_lowercase, ascii_uppercase
#   7
    8 if not sentence:
#   9 return ""
#  10
#  11 # Create a dictionary that maps lowercase letters to uppercase
letters
#  12 # Capitalize the first character if it's a lowercase letter
#  13 # Concatenate the capitalized character with the rest of the string
   14 lower_to_upper = dict(zip(ascii_lowercase, ascii_uppercase))
   15 return lower_to_upper.get(sentence[0], sentence[0]) + sentence[1:]
====================================================
====================================================
Input: python -> Output: Python
++++++++++++++++++++++++++++++++++++++++++++++++++++
#  1 def capitalize(sentence: str) -> str:
#  2 """
#  3 Capitalizes the first letter of a sentence or word.
#  4
#  5 """
   6 from string import ascii_lowercase, ascii_uppercase
#  7
   8 if not sentence:
#  9 return ""
# 10
# 11 # Create a dictionary that maps lowercase letters to uppercase
letters
# 12 # Capitalize the first character if it's a lowercase letter
# 13 # Concatenate the capitalized character with the rest of the string
  14 lower_to_upper = dict(zip(ascii_lowercase, ascii_uppercase))
  15 return lower_to_upper.get(sentence[0], sentence[0]) + sentence[1:]
====================================================
====================================================
Input: 123test -> Output: 123test
++++++++++++++++++++++++++++++++++++++++++++++++++++
#  1 def capitalize(sentence: str) -> str:
#  2 """
#  3 Capitalizes the first letter of a sentence or word.
#  4
#  5 """
   6 from string import ascii_lowercase, ascii_uppercase
#  7
   8 if not sentence:
#  9 return ""
# 10
# 11 # Create a dictionary that maps lowercase letters to uppercase
letters
# 12 # Capitalize the first character if it's a lowercase letter
# 13 # Concatenate the capitalized character with the rest of the string
  14 lower_to_upper = dict(zip(ascii_lowercase, ascii_uppercase))
  15 return lower_to_upper.get(sentence[0], sentence[0]) + sentence[1:]
====================================================
====================================================
Input: 33333 -> Output: 33333
++++++++++++++++++++++++++++++++++++++++++++++++++++
#  1 def capitalize(sentence: str) -> str:
#  2 """
#  3 Capitalizes the first letter of a sentence or word.
#  4
#  5 """
   6 from string import ascii_lowercase, ascii_uppercase
#  7
   8 if not sentence:
#  9 return ""
# 10
# 11 # Create a dictionary that maps lowercase letters to uppercase
letters
```

```
# 12 # Capitalize the first character if it's a lowercase letter
# 13 # Concatenate the capitalized character with the rest of the string
  14 lower_to_upper = dict(zip(ascii_lowercase, ascii_uppercase))
  15 return lower_to_upper.get(sentence[0], sentence[0]) + sentence[1:]
====================================================
====================================================
Input:  -> Output: 
++++++++++++++++++++++++++++++++++++++++++++++++++++
#  1 def capitalize(sentence: str) -> str:
#  2 """
#  3 Capitalizes the first letter of a sentence or word.
#  4
#  5 """
   6 from string import ascii_lowercase, ascii_uppercase
#  7
   8 if not sentence:
   9 return ""
# 10
# 11 # Create a dictionary that maps lowercase letters to uppercase
letters
# 12 # Capitalize the first character if it's a lowercase letter
# 13 # Concatenate the capitalized character with the rest of the string
  14 lower_to_upper = dict(zip(ascii_lowercase, ascii_uppercase))
  15 return lower_to_upper.get(sentence[0], sentence[0]) + sentence[1:]
====================================================
```

Coverage Report

You can run this command:

```
coverage report -m
```

to generate the report of statement coverage

```
Name              Stmts   Miss  Cover   Missing
-----------------------------------------------
capitalize.py         6      1    83%   1
-----------------------------------------------
TOTAL                 6      1    83%
```

The above report shows that for our current inputs, line 1 is always missing and total statement coverage is 83%.

Test `bubblesort_recursive.py`

Main `bubblesort_recursive.py` code snippet:

```
from typing import Any, List
def bubble_sort_recursive(collection: List[Any]) -> List[Any]:
    """
    It is similar to iterative bubble sort but recursive.

    :param collection: mutable ordered sequence of elements
    :return: the same list in ascending order
    """
    from typing import Any, List

    length = len(collection)
    swapped = False
    for i in range(length - 1):
        if collection[i] > collection[i + 1]:
            collection[i], collection[i + 1] = collection[i + 1],
collection[i]
            swapped = True

    return collection if not swapped else
bubble_sort_recursive(collection)
```

Inputs `inputs_bubblesort.txt` code snippet:

```
[-23, 0, 6, -4, 34]
[3, 1, 4, 1, 5]
[10, 2, 8, 6, 4]
[7, 3, 9, 0, 1]
[]
[1]
```

Command for running testing:

```
python3 test_prog_bubblesort.py inputs_bubblesort.txt
```

The above command will generate the expeted output from main function and print out the statement coverage:

```
====================================================
Input: [-23, 0, 6, -4, 34] -> Output: [-23, -4, 0, 6, 34]
++++++++++++++++++++++++++++++++++++++++++++++++++++
#  1 from typing import Any, List
#  2 def bubble_sort_recursive(collection: List[Any]) -> List[Any]:
#  3 """
#  4 It is similar to iterative bubble sort but recursive.
#  5
#  6 :param collection: mutable ordered sequence of elements
#  7 :return: the same list in ascending order
#  8 """
   9 from typing import Any, List
# 10
  11 length = len(collection)
  12 swapped = False
  13 for i in range(length - 1):
  14 if collection[i] > collection[i + 1]:
  15 collection[i], collection[i + 1] = collection[i + 1], collection[i]
  16 swapped = True
# 17
  18 return collection if not swapped else
bubble_sort_recursive(collection)
====================================================
====================================================
Input: [3, 1, 4, 1, 5] -> Output: [1, 1, 3, 4, 5]
++++++++++++++++++++++++++++++++++++++++++++++++++++
#  1 from typing import Any, List
#  2 def bubble_sort_recursive(collection: List[Any]) -> List[Any]:
#  3 """
#  4 It is similar to iterative bubble sort but recursive.
#  5
#  6 :param collection: mutable ordered sequence of elements
#  7 :return: the same list in ascending order
#  8 """
   9 from typing import Any, List
# 10
  11 length = len(collection)
  12 swapped = False
  13 for i in range(length - 1):
  14 if collection[i] > collection[i + 1]:
  15 collection[i], collection[i + 1] = collection[i + 1], collection[i]
  16 swapped = True
# 17
  18 return collection if not swapped else
bubble_sort_recursive(collection)
====================================================
====================================================
Input: [10, 2, 8, 6, 4] -> Output: [2, 4, 6, 8, 10]
++++++++++++++++++++++++++++++++++++++++++++++++++++
#  1 from typing import Any, List
#  2 def bubble_sort_recursive(collection: List[Any]) -> List[Any]:
#  3 """
#  4 It is similar to iterative bubble sort but recursive.
#  5
#  6 :param collection: mutable ordered sequence of elements
#  7 :return: the same list in ascending order
#  8 """
   9 from typing import Any, List
# 10
  11 length = len(collection)
  12 swapped = False
  13 for i in range(length - 1):
  14 if collection[i] > collection[i + 1]:
```

```
  15 collection[i], collection[i + 1] = collection[i + 1], collection[i]
  16 swapped = True
# 17
  18 return collection if not swapped else
bubble_sort_recursive(collection)
====================================================
====================================================
Input: [7, 3, 9, 0, 1] -> Output: [0, 1, 3, 7, 9]
++++++++++++++++++++++++++++++++++++++++++++++++++++
#  1 from typing import Any, List
#  2 def bubble_sort_recursive(collection: List[Any]) -> List[Any]:
#  3 """
#  4 It is similar to iterative bubble sort but recursive.
#  5
#  6 :param collection: mutable ordered sequence of elements
#  7 :return: the same list in ascending order
#  8 """
   9 from typing import Any, List
# 10
  11 length = len(collection)
  12 swapped = False
  13 for i in range(length - 1):
  14 if collection[i] > collection[i + 1]:
  15 collection[i], collection[i + 1] = collection[i + 1], collection[i]
  16 swapped = True
# 17
  18 return collection if not swapped else
bubble_sort_recursive(collection)
====================================================
====================================================
Input: [] -> Output: []
++++++++++++++++++++++++++++++++++++++++++++++++++++
#  1 from typing import Any, List
#  2 def bubble_sort_recursive(collection: List[Any]) -> List[Any]:
#  3 """
#  4 It is similar to iterative bubble sort but recursive.
#  5
#  6 :param collection: mutable ordered sequence of elements
#  7 :return: the same list in ascending order
#  8 """
   9 from typing import Any, List
# 10
  11 length = len(collection)
  12 swapped = False
  13 for i in range(length - 1):
  14 if collection[i] > collection[i + 1]:
  15 collection[i], collection[i + 1] = collection[i + 1], collection[i]
  16 swapped = True
# 17
  18 return collection if not swapped else
bubble_sort_recursive(collection)
====================================================
====================================================
Input: [1] -> Output: [1]
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++
#  1 from typing import Any, List
#  2 def bubble_sort_recursive(collection: List[Any]) -> List[Any]:
#  3 """
#  4 It is similar to iterative bubble sort but recursive.
#  5
#  6 :param collection: mutable ordered sequence of elements
#  7 :return: the same list in ascending order
#  8 """
   9 from typing import Any, List
# 10
  11 length = len(collection)
  12 swapped = False
  13 for i in range(length - 1):
  14 if collection[i] > collection[i + 1]:
  15 collection[i], collection[i + 1] = collection[i + 1], collection[i]
  16 swapped = True
# 17
  18 return collection if not swapped else
bubble_sort_recursive(collection)
====================================================
```

Coverage Report

You can run this command:

```
coverage report -m
```

to generate the report of statement coverage

```
Name                        Stmts   Miss  Cover   Missing
-------------------------------------------------------
bubblesort_recursive.py        10      2    80%   1-2
-------------------------------------------------------
TOTAL                          10      2    80%
```

The above report shows that for our current inputs, line 1 & 2 are always missing and total statement coverage is 80%.