COMPSCI 101
Principles of Programming
Revision Exercises
Topics

- Dictionaries
  - Build a dictionary
  - Use a dictionary to solve a problem (application)
- Nested statements
  - Nested Loops
- Drawing
- Mutable & Immutable
  - Mutable objects: Lists
  - Immutable Objects: Tuples and Strings
  - Passing parameters which are mutable objects to functions means that the function code may change the object's data.
- doctests
Write a function, `get_word_frequency(a_string)`, which takes a string as a parameter and returns a word-frequency dictionary.

The dictionary contains each word and the frequency of each word in the text. You may assume that the text has no punctuation, words are separated by blank spaces and only lowercase letters are being used. If the parameter is an empty string, the function returns an empty dictionary.

```python
the_dict = get_word_frequency('all animals are equal but some animals are more equal than others')
print(the_dict)
```

may print:

```python
{'more': 1, 'animals': 2, 'than': 1, 'some': 1, 'but': 1, 'are': 2, 'equal': 2, 'others': 1, 'all': 1}
```
Build the dictionary

- **Returns:** the dictionary
  - **Key:** text;
  - **Value:** frequency

Create a new dictionary

Split the sentence into a list of words

For each word in the list
  - If the word is in the dictionary, increment the frequency
  - If not, create a new pair

Return the dictionary
The get_word_frequency(text) function

def get_word_frequency(text):
    word_list = text.__________()
    counters = ______
    for word in ________:
        if word _____ counters:
            counters[word] = ____________________
        else:
            counters[word] = __________
    return counters
Nested Loops

The placing of one loop inside the body of another loop is called nesting. When you "nest" two loops, the outer loop takes control of the number of complete repetitions of the inner loop.

```python
num2 = 0
while num2 <= 3:
    num1 = 0
    while num1 <= 2:
        print(num2, num1)
        num1 += 1
    num2 += 1
```

When working with nested loops, the outer loop changes only after the inner loop is completely finished (or is interrupted.).
Example: printing ...

- a square of characters

```python
def print_square(num_rows):
    for row in range(_______):
        for col in range(_______):
            print(____, end='')
        print()

def print_square2(num_rows):
    for row in range(__________):
        print('#' * _________ )
```

OR

```python
def print_square(num_rows):
    for row in range(_______):
        for col in range(_______):
            print(____, end='')
        print()

def print_square2(num_rows):
    for row in range(__________):
        print('#'* _________)
```
Example: printing ...

- A triangle of characters

```python
def print_triangle(num_rows):
    for row in range(_______):
        for col in range(____+___):
            print(____, end='')
        print()
```

```
def print_triangle2(num_rows):
    for row in range(__________):
        print('#'*(______ +__) )
```

<table>
<thead>
<tr>
<th>Row</th>
<th>numbers of #</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Example: printing ...

A triangle of characters

```python
def print_down_triangle(num_rows):
    for row in range(_______):
        for col in range(_________- ___):
            print(____, end='')
        print()

def print_down_triangle2(num_rows):
    for i in range(__________):
        print('#'*(_______ - __))
```

<table>
<thead>
<tr>
<th>Row</th>
<th>numbers of #</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
def rectangular_grid(a_canvas):
    number_of_columns = 3
    number_of_rows = 5
    left_hand_side = 10
    y_down = 10
    size = 10
    dist = size
    for row in range(number_of_rows):
        x_left = left_hand_side
        for col in range(row+1):
            rect = (x_left, y_down, x_left + size, y_down + size)
            a_canvas.create_rectangle(rect, fill='blue')
            x_left += dist #Position A
            y_down += size #Position B

- how many times is the statement marked Position A in the program above executed? $1 + 2 + \ldots$
- how many times is the statement marked Position B in the program above executed?
```python
def fiddle_lists(list1, list2):
    list1 = [1, 5]
    list2.append(list1[1])
    list1.append(list2[0])
    list1.append(list2[1])

def main():
    a_list1 = [5, 3, 2]
    a_list2 = [4, 6]

    fiddle_lists(a_list1, a_list2)

    print("a_list1:", a_list1)
    print("a_list2:", a_list2)

main()  
```

Example

2016 Exam

def fiddle_lists(list1, list2):
    list1 = [1, 5]
    list2.append(list1[1])
    list1.append(list2[0])
    list1.append(list2[1])

def main():
    a_list1 = [5, 3, 2]
    a_list2 = [4, 6]

    fiddle_lists(a_list1, a_list2)

    print("a_list1:", a_list1)
    print("a_list2:", a_list2)

main()  

```
```python
def fiddle_tuples(t1, t2):
    t3 = (t1[1], t2[0])
    t1 = (t3[1], t2[0])
    t2 = t1
    return t3

def main():
    t1 = (3, 5)
    t2 = (4, 7)
    t3 = t2
    t3 = fiddle_tuples(t1, t2)
    print(t1, t2, t3)

main()
```

```
(3, 5) (4, 7) (5, 4)
```
Which of the following is a failure test case.

def testable(x):
    
    """
    The 'testable' function returns the square root of its parameter, or 3, whichever is larger.
    >>> testable(7)
    3.0
    >>> testable(16)
    4.0
    >>> testable(9)
    3.0
    >>> testable(10) == 10 ** 0.5
    True
    >>> testable(1)
    1.0
    """
    if x < 9:
        return 3.0
    return x ** 0.5
Which of the following is a failure test case.

def calculate_average(numbers):
    """Returns the average
    Arguments: numbers - a list of numeric values
    Returns: the average of the list of numbers given
    >>> calculate_average([0.0, 0.0, 0.0])
    0.0
    >>> calculate_average([3])
    3.0
    >>> calculate_average([2, 3])
    2.5
    >>> calculate_average([])
    0.0
    """
    n = len(numbers)
    total = 0
    for element in numbers:
        total += element
    return total / n