Solution of
Practice Midterm Questions

Note: These questions are for you to practice for the midterm exam. The format and coverage of the midterm questions may not be the same.

Appendix at the End

The appendix at the end of the questions contains some useful function and methods.

1. Short Answer

1.1. Python allows you to enter strings using either single quotes (`'`), double quotes (`"`) or triple quotes (`"""`). Is it necessary to provide all these options? If it is necessary, briefly explain why. If it is not necessary, briefly explain why Python allows the options anyway.

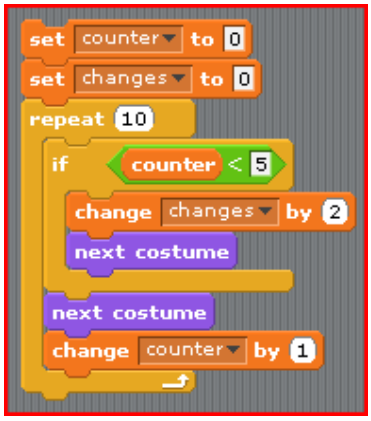
- We only need one that allows a string to span over multiple lines (like """)
- Python provides the other options for convenience. We can use one type of quotes to delimit a string that includes a quote of the other quote type.

1.2. Suppose we define a Python module for estimating personal income tax and save it in a file named `income.py`. What would this module's name be when we open it in Spyder and run it by clicking the green arrow? What would this module's name be when we import it in another Python program that is stored in a file called `myincome.py`?

- When we run it the module's name will be "__main__".
- When we imported in any other file, its name will be "income".

2. Code Tracing

2.1. What are the contents of `counter` and `changes` after the following script is run?



counter → 10

changes → 10

2.2. Consider the following function:

```
def bar1(y):
    x = 0
    z = ""
    while x < len(y):
        if x % 2 == 0 :
            z = z + y[x]
        x = x + 1
```

Where `len(s)`, `s[k]`, and `x % y` are the string operators explained in the appendix at the end of this document.

Suppose the main program that calls this function has the following statements:

```
x = 20
y = "big deal"
r = bar1(y)
```

What are the values of x, y, `z` and r after these statements are executed?

`x` → `20`

`y` → `"big deal"`

`r` → `None` (The function bar1 DOES NOT RETURN ANYTHING!)

`z does not exist; it is local to the function`

2.3. Consider the following python code:

```
def bar1(y):
    x = 0
    z = ""
    while x < len(y):
        if x % 2 == 0 :
            z = z + y[x]
        x = x + 1
    return z

# Main Program
x = 20
y = "big deal"
r = bar1(y)
```

What are the values of x, y, and r after these statements are executed?

x → 20

y → "big deal"

r → " bgda"

2.4. Consider the following code :

```
n = 0
text = "Python is a different language than Scratch"
ntext = ""
vtext = "aeiouy"
for p in text:
    if p in vtext:
        n += 1
    else:
        ntext += p
```

What are the values of `n`, `text`, `ntext` and `vtext` after executing this code?

n → 13

text → "Python is a different language than Scratch"

ntext → "Pthn s dffrnt lngg thn Scrtch"

vtext → "aeiouy"

3. Code Summarization

3.1. Briefly **summarize** what the following Python function does when it is called with a string as its first argument a character (a string with one character) as its second argument and a number as its third argument.

```
def foo(str1, char, n):
    res = ''
    for i in range(len(str1)):
        if(len(res) < n):
            res = res + str1[i] + char
    return res
```

The function foo takes as its arguments a string, a character and an integer n and produces a **new** string of n characters that consists of the first n/2 characters of the given string with the given character in between them.

What will the function return if it is called by the statement foo("Hello World", "*", 6)?

It will return: "H*e*l*"

3.2. Briefly **summarize** what the following Python function does when called with a string as its argument .

```
def grap(x):
    z = ""
    for y in x:
        z = y + z
    return z
```

The function grap accepts a string and returns a **new** string that has the came characters as the input sting but in the reverse order.

Or

The function accepts a string and returns a new string that is the reverse of the original string.

3.3. Briefly **summarize** the effect of the function `bar1` in question 2.3 above

The function accepts a string and returns a new string that has every second character of the original string in the same order that appears in the given string.

4. Fill in the Code

4.1. Fill in the missing lines of code in the following code. The code reads in a limit amount and a list of prices and prints the largest price that is less than the limit. You can assume that all prices and the limit are positive numbers. When a price 0 is entered the program terminates and prints the largest price that is less than the limit if there was one, or a message indicating that no price was less than the limit.

```
# Read the limit
limit = float(input("Enter the limit: "))
max_price = 0
# Read the next price
next_price = float(input("Enter a price or 0 to stop: "))

while next_price > 0 :
    << write your code here>>
```

```
      if next_price < limit and next_price > max_price :
              max_price = next_price
```

```
    # Read the next price
    << write your code here>>
```

```
      next_price = float(input("Enter a price or 0 to stop: "))
```

```
if max_price > 0:
    << write your code here>>
```

```
      Print( "The largest price less than", limit, "is", max_price )
```

```
else :
    << write your code here>>
```

```
      Print( "None of the given prices was less than the limit.")
```

4.2. Write a Python version of the Scratch code in question 2.1 above. If the Scratch code changes the sprite's values (like its position or costume, etc), your program should just print out the changes.

```
counter = 0

changes =  0

for i in range(10):

        if counter < 5:

                changes += 2

                print ("costume has changed to next costume")

        print( "costume has changed to next costume")

        counter += 1
```

5. Find the Bug

5.1. There is a bug in the following function; in other words, the function does not do what the docstring says that it does. Where is the bug? How might you fix it?
You should assume that the function `retain(original_string, filter_string)` is defined, has no bugs and returns the characters of the original string that appear in the filter string. For instance, `retain("introduction", "ion")` will return `"inoion"`.

Note that a DNA sequence is a string in which each character is `G, C, A or T`. For instance `"AGGAAGGGAATTTAGCAGC"` is a DNA sequence, while `"CGGACUNGCATATAT"` is not.

```
def compute_ratio(seq):
    """ (str) -> int
    Compute the GC ratio for a sequence.

    Inputs: seq - A string containing a sequence of characters.

    Outputs: A integer number specifying the percentage of GC in the
    sequence. That is, a value 15 means that 15% of the sequence cbases
    are either G or C. If the sequence is not DNA (eg: if there are
    any sequence characters which are not G, C, A, or T),
    then None is returned.

    >>> compute_ratio("AAATGGGCC")
    55
    """

    gc = retain(seq, "GC")
    at = retain(seq, "AT")
```

```
    if len(gc) + len(at) != len(seq):
        return None

    return len(gc) // len(seq)
```

Two errors:

- It does not check for the case that the sequence is empty. In this case len(seq) is 0 and the division will produce an error. We need to add the following code at the beginning:
- Last line must use a floating point division and then change the result to a percentage by multiplying it by 100 and change it into an int.
- The correct code is as following:

```
if len(seq) == 0 :
        return None

gc = retain(seq, "GC")
at = retain(seq, "AT")

if len(gc) + len(at) != len(seq):
        return None

return int( len(gc) / len(seq) * 100 )
```

6. Write Code

6.1. Define a function `print_triangle` which implements the following task. Do not forget to include a header line.

- Input: A character `char` and a positive integer `n`.
- Task: Prints an isosceles triangle filled with the given character whose orientation is as following:
  The triangle's base spans over 2n lines and is at the leftmost end of the page, and its top vertex is between the n-th and (n+1)-th lines.
- For instance `print_triangle("*", 4)` will print the following

```
*
**
***
****
****
***
**
*
```

```
def print_triangle(char, length) :
```

```
    for i in range(1, length +1):
        print( char * i )
    for i in range(length, 0, -1):
        print( char * i )
```

6.2. Write a version of the function bar1() from question 2.3 that uses a for loop instead of a while loop.

```
def bar1(y):
    z = ""
    for x in range( len(y) ):
        if x % 2 == 0 :
            z = z + y[x]
    return z
```

7. Testing

7.1. Consider again the function that computes the GC ration of a DNA sequence which was defined in 5.1. The function header and specification (docstring) is as following:

```
def compute_ratio(seq):
 """ (str) -> int
 Compute the GC ratio for a sequence.

 Inputs: seq - A string containing a sequence of characters.

 Outputs: A integer number specifying the percentage of GC in the
 sequence. That is, a value 15 means that 15% of the sequence cbases
 are either G or C. If the sequence is not DNA (eg: if there are
 any sequence characters which are not G, C, A, or T),
 then None is returned.

 >>> compute_ratio("AAATGGGCC")
 55
 """
```

Following the guidelines we discuss in the class, list a set o test cases that will provide a good black box testing for this function.

- **Typical Case : A string with a correct DNA sequence and any length greater than 4 (has at least one of the four bases):**

  compute_ratio("AACAGATTTAGT")

- **Typical Case : A string with an incorrect DNA sequence and any length:**

  compute_ratio("AACCUGATUTAGD")

- **Boundary Case : A sting with only A's and T's:**

  compute_ratio("AAATTTATAT")

- **Boundary Case : A sting with an empty sequence:**

  compute_ratio("")

- **Interesting Case : A correct sting with one character:**

  compute_ratio("G")

- **Interesting Case : An incorrect sting with one character:**

  compute_ratio("U")

7.2. Add the doctest with the cases you defined in 7.2 in the docstring for this function.

Note: When The returned value is expected to be `None`, you indicate that with an empty line in the doctest. That is, the value of `None` is nothing. (It is fine for the exam if you write `None` instead of empty line, but doctest won't accept it.

```
def compute_ratio(seq):
 """ (str) -> int
 Compute the GC ratio for a sequence.

 Inputs: seq - A string containing a sequence of characters.

 Outputs: A integer number specifying the percentage of GC in the
 sequence. That is, a value 15 means that 15% of the sequence cbases
 are either G or C. If the sequence is not DNA (eg: if there are
 any sequence characters which are not G, C, A, or T),
 then None is returned.


 >>> compute_ratio("AACAGATTTAGT")
 25
 >>> compute_ratio("AACCUGATUTAGD")
```

```
>>> compute_ratio("AATTATT")
0
>>> compute_ratio("")

>>> compute_ratio("G")
100
>>> compute_ratio("U")

"""
```

8. Importing and using modules

Below is the code contained within a file, 'baz.py':

```
def spam(x):
    print ('Spam says its home is', __name__)
    return x / 5

def eggs(x):
    print ('Eggs says its home is', __name__)
    return x + 3

if __name__ == '__main__':
    print ("I'm home!")
else:
    print("This isn't where I live...")
```

8.1. If the following commands are typed into the console in succession, with the directory containing baz.py being the current directory, what will appear in the console? If an error is thrown, briefly describe why the error occurred.

```
>>> import baz
This isn't where I live...

>>> spam(10)
Error :name 'spam' is not define

>>> from baz import spam
Nothing will be shown. The module is already imported.
```

8.2. If you needed to use the function spam(x) from baz, give **one advantage and one disadvantage** of using the command 'from baz import spam' instead of just 'import baz'.

Advantage: we can call this function as **spam(x)** instead of **baz.spam(x)**

Disadvantage: If we import another function named spam() from another module, the call "spam(x)" will call the second function.

8.3. If baz.py were to be run in Spyder (by clicking the green arrow), rather than imported, what would be printed on the screen if the following commands were typed into the console? If an error is thrown, briefly describe why the error occurred.

```
>>> baz.eggs(3)
```

Error: name 'baz' is not defined

```
>>> import baz
```

This isn't where I live...

```
>>> eggs(spam(5))
```

Spam says its home is __main__
Eggs says its home is __main__
4.0

APPENDIX
Some Potentially Useful Python Functions, Methods and Syntax

Numbers

- **int(**x**)** : converts x to an integer
- **float(**x**)** : converts x to a floating point number
- **round(**x**,** n **= 0)**: rounds x to n digits, returns a float.
- **x % y** returns the remainder when x is divided by y. For instance: `10%3` is 1; `12%3` is 0 ; `0%3` is 0.

Collections / sequences (strings, lists, tuples)

- **len(**x**)** : returns the length of collection x
- v **in** c: returns True if item v occurs in collection c
- c[i]: returns a reference to the i^{th} element of the sequence.
- c[i:j]: returns a new sequence containing references to elements i to j-1.

Strings

- **str(**x**)** : converts x to a string
- **len(**s**)** returns the length of the string s
- **s[**k**]** is the k-th character of the string s ; k can be any number from 0 to 1 less the length of s.
  For instance: If s="abcd" then `s[0]` returns 'a'; `s[3]` returns 'd'; `s[4]` produces an error.
- s**.strip()**: returns the string s with any leading or trailing whitespace removed
- s**.split(**sep**)** : returns a list of substrings constructed by splitting s at the separator character(s) given by string sep
- s**.find(**s2**,** beg **= 0)** : returns the index of the first occurrence of substring s2 in s after index beg. Returns -1 if s1 does not occur.

General

- **input(**string**)** : displays the string, reads a single line of text from the keyboard and returns it as a string.
- **range(**stop**)**: Returns a sequence of integers from 0 to stop-1.
- **range(**start**,** stop**)**: Returns a sequence of integers from start to stop-1.
- **print(**value**, …,** sep **= ' ',** end **= '\n')**: displays one or more values to the screen, separated by the character sep and finishing with the character end.

Comparison Operators : ==, !=, <=, <, >=, >
Boolean Operators: **and, or, not**
Control Statements:

```
def function_name(inputs):
    block
for var in sequence:
    block
while condition:
    block
assert condition
```

```
if condition:
    block1
elif:
    block2
else:
    blockN
```