The Taylor series expansion for $e^x$ has the following form:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

If we truncate the sum after $N$ terms, we have an approximation for $e^x$ as:

$$e^x \approx \sum_{n=0}^{N-1} \frac{x^n}{n!}$$

Given the following Python code:

```python
import math

# math.factorial(n) returns n!

def approx_ex(x, n):
    """ <type contract goes here>"

    Use Taylor series expansion (executed n times) to approximate the value of $e$ raised to the $x$th power. Return this result.

    >>> approx_ex(1, 100)
    2.7182818284590455
    """
    acc = 0
    ex = 1
    for stepn in range(n):
        acc += ex / math.factorial(stepn)
        ex *= x
    return acc

approx_ex(1, 100)

a) Write the type contract for function `approx_ex`. (number, int) \(\rightarrow\) float

b) The first time the for loop executes, what is the value of `stepn`? 0
c) The last time the for loop executes, what is the value of `stepn`? 99
d) After the first time the for loop executes, what is the value of `ex`? 1
concepts: understanding code, Python object types, computational process, including parameter passing, assignment, return values and side effects, numeric data

(2)(a) Complete the docstring for function `periscope`.

(2)(b) What is the result of executing `>>> q2()`?

```
def periscope(x, y):
    '''(int, int) -> int

    Subtract twice y from twice x;
    Return this result.

>>> periscope(7, 5)
4
'''
x = 2 * x
y = 2 * y
return x - y
```

```
def q2():
    '''() -> None
    What does this function do?
    '''
x = 7
y = 5
z = periscope(x, y)
print(x + y + z)
return None
```
concepts: understanding code, Python object types, computational process, including Python parameter passing, for loop, conditionals and Boolean data type

(3)(a) Complete the docstring for function q3.

(3)(b) What is the result of executing `>> q3('abccdef')`?

```python
def q3(s):
    '''(str) -> int

    Returns length of longest single-char string in s.
    
    >>> q3('abccdef')
    3
    >>> q3('')
    0
    >>> q3('abcdef')
    1
    '''
    if len(s) != 0:
        prev_char = s[0]
        dup_ct = 1
        high_ct = 1
    else:
        high_ct = 0

    for i in range(1, len(s)):
        if s[i] == prev_char:
            dup_ct += 1
        else:
            prev_char = s[i]

        if dup_ct > high_ct:
            high_ct = dup_ct
            dup_ct = 1
    return high_ct
```
(4)(a) Complete the docstring for function q4.

(4)(b) What is the result of executing `q4('CIS 210')`? True

```python
def q4(astring):
    """(str) -> Boolean

    Returns True if astring includes two or more numbers.
    """
    digits_ctr = 0
    for c in astring:
        if c.isdigit():
            digits_ctr += 1
    return (digits_ctr >= 2)
```

(5) Add the missing line of code:

```
total = 0
anum = 1
while anum <= 10:
    total += anum
    anum += 1
print(total)
```
(6) What is the result of executing `>>> q6(90)`?

```python
def q6(score):
    ''' No docstring on the exam '''
    gradepoint = 0
    if score >= 90:
        gradepoint = 4
    if score >= 80:
        gradepoint = 3
    if score >= 70:
        gradepoint = 2
    if score >= 60:
        gradepoint = 1
    return gradepoint
```

(7) Replace ?? (2 places) with the expected results, given the following code:

```python
def twice(x):
    ''' '''
    result = 2 * x
    print(result)
    return None
```

```python
>>> x = 99
>>> twice(10)
20
>>> x
99
```
concepts: understanding Python code, indefinite v. definite loops; good programming style

(8) Rewrite the while loop as a for loop:

```python
total = 0
astr = 'a b c d e f'
i=0
while i < len(astr):
    if astr[i] == ' ':  
        total += 1
    i += 1
print(total)

```

```python
total = 0
astr = 'a b c d e f'
for i in range(len(astr)):
    if astr[i] == ' ':  
        total += 1
print(total)
```
(9) An approximation for the square root of \( n \) can be generated using the following equation:

\[
x_{k+1} = \frac{1}{2} \left( x_k + \frac{n}{x_k} \right), \text{where } x_0 = 1
\]

Each value of \( x \) should be a better approximation for the square root of \( n \).

(a) Supply the type contract for function `approx_sqrt` below, consistent with this equation.

(b) Replace the `??` (2 places) with the code needed to implement the approximation.

```python
def approx_sqrt(n, k):
    '''(number, int) -> float

    Generates an approximate square root of n, a positive number, via an iterative process that runs k times. The approximate square root is returned.
    >>> approx_sqrt(1, 1)
    1.0
    >>> approx_sqrt(4, 1)
    2.5
    >>> approx_sqrt(4, 5)
    2.000000000000002
    ...
    x = 1

    for ctr in range(k):
        x = .5 * (x + n/x)

    return x
```