Name: ______________________________________

Use this quiz to help make sure you understand the videos/reading. ***Answer all questions.*** Make additional notes as desired. ***Not sure of an answer?*** Ask your instructor to explain in class and revise as needed then.

Throughout, where you are asked to "circle your choice", you can circle or underline it (whichever you prefer).

Online reading: ***Overloading the Plus Operator***

1. Fill in the blanks:

   `11 + 22` evaluates to ____________________

   `'11' + '22'` evaluates to ____________________

   `'11' + str(3 + 3) + '22'` evaluates to ____________________

   `'11' + 33` evaluates to __________________ (this one is a trick question)

2. When the code snippet to the right is executed, what gets printed?

```
x = 1
y = 2
z = 3

print(x, y, z)

print(str(x) + str(y) + str(z))

print(x + y + z)
```

Output (*fill in the blanks*):

____

____

____

3. Implement (here, on paper, in the supplied box) the following function, per its specification. In doing so, you should use the concepts of string concatenation and the ***str*** function (per the online reading and the previous problems).

```
def print_equation(x, y):
    """
    Prints an equation for the sum of x and y, with no spaces.
    For example:
    -- If x is 65 and y is 11, then this function prints:   65+11=76
    -- If x is 305 and y is 41, then this function prints:  305+41=346
    Precondition:  The arguments are numbers.
    """
```

Online reading: ***Accumulating Sequences***

4. Implement (here, on paper, in the supplied box) the following function, per its specification.

```
def list_of_numbers(n):
    """
    Returns the list   [1, 2, 3, 4, ... n]
    where n is the given argument.  For example:
    -- If the argument is 5, this function returns: [1, 2, 3, 4, 5]
    -- If the argument is 2, this function returns: [1, 2]
    -- If the argument is 0, this function returns: [] (the empty list)

    Precondition: The argument is a non-negative integer.
    """
```

5. Implement (here, on paper, in the supplied box) the following function, per its specification.

```
def string_of_numbers(n):
    """
    Returns the string   '12345678910111213 ...' where the last number
    in the string is the given integer. For example:
    -- If the argument is 6, this function returns: '123456'
    -- If the argument is 25, this function returns:
          '12345678910111213141516171819202122232425'
    -- If the argument is 0, this function returns: ''

    Precondition: The argument is a non-negative integer.
    """
```

Video: ***Patterns for Iterating Through Sequences*** [15:21 minutes]

6. Implement (here, on paper, in the supplied box) the following function, per its specification.

```
def index_of_first_negative(sequence):
    """
    Returns the index of the first negative number in the given
    Sequence of numbers.  Returns None if the sequence contains
    no negative numbers.  For example, if the argument is:
    -- [4, 30, -19, 8, -3, -50, 100], this function returns 2
    -- [-8, 44, 33], this function returns 0
    -- [1, 29, 22, 8], this function returns None

    Precondition: The argument is sequence.
    """
```

7. Implement (here, on paper, in the supplied box) the following function, per its specification.

```
def number_of_stutters(string):
    Returns the number of "stutters" in the given string, where
    a "stutter" is a character repeated twice in a row.   For example:
    -- number_of_stutters('xhhbrrs')        returns 2
    -- number_of_stutters('zzzz')           returns 3
    -- number_of_stutters('xxx yyy xxxx')   returns 7
    -- number_of_stutters('xxxyyyxxxx')     returns 7
    Precondition: The argument is string.
    """
```

8. Implement (here, on paper, in the supplied box) the following function, per its specification.

```
def largest_number(sequence, m):
    Returns the largest number in the first  m  numbers of the
    given sequence of numbers, where  m  is the second argument.
    For example, if sequence X is [7, 4, 15, 20, 13, 40, 10], then:
    -- largest_number(X, 1) returns 7
    -- largest_number(X, 2) returns 7
    -- largest_number(X, 3) returns 15
    -- largest_number(X, 4) returns 20
    -- largest_number(X, 5) returns 20
    -- largest_number(X, 6) returns 40
    -- largest_number(X, 7) returns 40
    Precondition: The first argument is a non-empty sequence
                  and the second argument is a positive integer
                  no larger than the length of the given sequence.
    """
```