1 point

1. Select the expression(s) that evaluate to `True`.

- [ ] `len([1, 2, 3]) == len(['a', 'b', 'c'])`
- [ ] `[1, 2, 3] in len('mom')`
- [ ] `len('mom') in [1, 2, 3]`
- [ ] `'a' in ['mom', 'dad']`
- [ ] `'3' in [1, 2, 3]`
- [ ] `int('3') in [len('a'), len('ab'), len('abc')]`

---

1 point

2. Consider this code:

```
def mystery(s):
    i = 0
    result = ''

    while not s[i].isdigit():
        result = result + s[i]
        i = i + 1

    return result
```

Select the function call(s) that result in an error.

- [ ] `mystery('abc')`
- [ ] `mystery('abc123')`
- [ ] `mystery('123')`
- [ ] `mystery('123abc')`

---

1 point

3. Consider this code:

```
def example(L):
    """ (list) -> list
    """
    i = 0
    result = []
    while i < len(L):
        result.append(L[i])
        i = i + 3
    return result
```

Which is the best docstring description for function `example`?

- Return a `list` containing every third *index* from `L` starting at index 0.
- Return an empty `list`.
- Return a `list` containing the items from `L` starting from index 0, omitting every third item.
- Return a `list` containing every third *item* from `L` starting at index 0.

---

1 point

## 4.

```
def compress_list(L):
    """ (list of str) -> list of str

    Return a new list with adjacent pairs of string elements from
        Lconcatenated together, starting with indices 0 and 1, 2 and 3,and
        so on.

    Precondition: len(L) >= 2 and len(L) % 2 == 0

    >>> compress_list(['a', 'b', 'c', 'd'])
    ['ab', 'cd']
    """
    compressed_list = []
    i = 0

    while i < len(L):
        compressed_list.append(L[i] + L[i + 1])
        # MISSING CODE HERE

     return compressed_list
```

Select the missing line of code.

- `i = i + 2`
- `i = i + i`
- `i = i + 1`
- `i = i * 2`

1 point

5. What is the sum of the `odd` numbers from 1523 through 10503, inclusive? Hint: write a `while` loop to accumulate the sum and print it. Then copy and paste that sum. For maximum learning, do it with a `for` loop as well, using `range`.

Enter answer here

1 point

6. Consider this code:

```
1  def while_version(L):
2      """ (list of number) -> number
3      """
4      i = 0
5      total = 0
6
7      while i < len(L) and L[i] % 2 != 0:
8           total = total + L[i]
9            i = i + 1
10
11       return total
```

The `while` loop stops as soon as an even number is found, and the sum of all the previous numbers is returned.

The four functions below use a `for` loop to try to accomplish the same task, although they keep iterating through all of the numbers in `L` regardless of whether the numbers are even or odd. Only one of them returns the same value as function `while_version`. Which one is it?

```
1  def for_version(L):
2      found_even = False
3      total = 0
4
5      for num in L:
6          if num % 2 != 0 and not found_even:
7              total = total + num
8          else:
9              found_even = True
10
11     return total
```

```
1  def for_version(L):
2      found_even = False
3      total = 0
4
5      for num in L:
6          if num % 2 != 0:
7              total = total + num
8          elif not found_even:
9              found_even = True
10
11     return total
```

```
def for_version(L):
    found_even = False
    total = 0

    for num in L:
        if num % 2 != 0:
            total = total + num
        found_even = True

    return total
```

```
def for_version(L):
    found_even = False
    total = 0

    for num in L:
        if num % 2 != 0:
            total = total + num
            found_even = True

    return total
```

1 point

7. Consider this code:

```
>>> numbers = [1, 4, 3]
>>> # MISSING CODE HERE
>>> print(numbers)[3, 4, 1]
```

Which of the following code fragments(s) could be the missing code in the program above?

- `reverse(numbers)`
- `numbers = numbers.reverse()`
- `numbers = reverse(numbers)`
- `numbers.reverse()`

1 point

8. Consider this code:

```
fruits = ['banana', 'apple', 'pear', 'peach']
fruits.insert(fruits.index('pear'), 'watermelon')
print(fruits)
```

What is printed by the code above?

- `['banana', 'apple', 'watermelon', 'pear', 'peach']`

- ['banana', 'apple', 'watermelon', 'peach']
- ['banana', 'apple', 'pear', 'watermelon', 'peach']
- ['banana', 'watermelon', 'apple', 'pear', 'peach']

---

1 point

9. Your younger sibling has just discovered music from the 1970's. They have put together a playlist of the same 5 songs repeated again and again. Here are the songs:

- ABC by The Jackson 5
- Venus by Shocking Blue
- Lola by the Kinks
- Let It Be by the Beatles
- Cecilia by Simon and Garfunkel

Here is an example of their playlist:

['Lola', 'Venus', 'Lola', 'Lola', 'Let It Be', 'Lola', 'ABC', 'Cecilia', 'Lola', 'Lola']

You want to make sure that Lola gets played at most 3 times, so you want to complete this function that edits the playlist:

```
1  def cap_song_repetition(playlist, song):
2      '''(list of str, str) -> NoneType
3
4      Make sure there are no more than 3 occurrences of song in playlist.
5
6      '''
```

Select the loop(s) that accomplish this.

```
1      while playlist.count(song) > 3:
2          playlist.pop(playlist.index(song))
```

```
1      while playlist.count(song) > 3:
2          playlist.pop(song)
```

```
1      while playlist.count(song) > 3:
2          playlist.remove(playlist.index(song))
```

```
1      while playlist.count(song) > 3:
2          playlist.remove(song)
```

```
1      while playlist.count(song) >= 3:
2          playlist.remove(song)
```

1 point

10. Consider this code:

```
1  >>> a = [1, 2, 3]
2  >>> b = a
3  >>> # MISSING CODE HERE
4  >>> print(a, b)
5  [1, 'A', 3] [1, 'A', 3]
```

Which of the following code fragments(s) could be the missing code in the program above?

- [ ] `b[-2] = 'A'`
- [ ] `b[1] = 'AB'`

  `a[1] = a[1][0]`
- [ ] `a = [1, 'A', 3]`

  `b = [1, 'A', 3]`
- [ ] `a[1] = 'A'`

1 point

11. Consider this code:

```
1  >>> a = [1, 2, 3]
2  >>> b = [1, 2, 3]
3  >>> # MISSING CODE HERE
4  >>> print(a, b)
5  [1, 'A', 3] [1, 'A', 3]
```

Which of the following code fragments(s) could be the missing code in the program above?

- [ ] `a[1] = 'A'`
- [ ] `a = [1, 'A', 3]`

  `b = [1, 'A', 3]`
- [ ] `b[1] = 'AB'`

  `a[1] = a[1][0]`
- [ ] `b[-2] = 'A'`

1 point

## 12. Consider this code:

```
1  def increment_items(L, increment):
2      i = 0
3      while i < len(L):
4          L[i] = L[i] + increment
5          i = i + 1
6
7  values = [1, 2, 3]
8  print(increment_items(values, 2))
9  print(values)
```

What is printed by the program above?

- `None`

  `[1, 2, 3]`

- `None`

  `[3, 4, 5]`

- `[3, 4, 5]`

  `[1, 2, 3]`

- `[3, 4, 5]`

  `None`

1 point

## 13. Select the code fragment(s) that print [3, 6, 9].

- ```
  1  values = []
  2  for num in range(3, 10, 3):
  3      values.append(num)
  4  print(values)
  ```

- ```
  1  values = []
  2  for num in range(1, 3):
  3      values.append(num * 3)
  4  print(values)
  ```

- ```
  1  values = []
  2  for num in range(1, 4):
  3      values.append(num * 3)
  4  print(values)
  ```

- ```
  1  values = []
  2  for num in range(3, 9, 3):
  3      values.append(num)
  4  print(values)
  ```

1 point

14. Select the function calls to `range` that, when used to fill in the blank, cause the code to produce the results below.

```
1  for num in ____________________:
2      print(num)
```

The loop should print this:

```
1  3
2  11
3  19
```

- [ ] `range(3, 23, 8)`
- [ ] `range(3, 20, 8)`
- [ ] `range(3, 19, 8)`
- [ ] `range(3, 8, 20)`

I understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account. Learn more about Coursera's Honor Code

Simon Uribe-Convers

Submit Quiz