For Loops Over Indices, Parallel Lists and Strings, and Files

Quiz, 13 questions

1 point

1. Consider this code:

```
1  def merge(L):
2      merged = []
3      for i in range(0, len(L), 3):
4          merged.append(L[i] + L[i + 1] + L[i + 2])
5      return merged
6
7  print(merge([1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

What is printed by the code above?

- [1, 4, 7]
- [6, 15, 24]
- [12, 15, 18]
- [123, 456, 789]

---

1 point

2. Consider this code:

```
1  def mystery(s):
2      """ (str) -> bool
3      """
4      matches = 0
5      for i in range(len(s) // 2):
6          if s[i] == s[len(s) - 1 - i]: # <--- How many times is this
               line reached?
7              matches = matches + 1
8
9      return matches == (len(s) // 2)
10
11 mystery('civil')
```

Trace the function call `mystery('civil')` using the Python Visualizer. How many times is the line marked above reached?

Enter answer here

---

1 point

3. Consider this code:

```
def mystery(s):
    """ (str) -> bool
    """
    matches = 0
    for i in range(len(s) // 2):
        if s[i] == s[len(s) - 1 - i]:
            matches = matches + 1

    return matches == (len(s) // 2)
```

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

- Return `True` if and only if `s` is equal to the reverse of `s`.
- Return `True` if and only if there are exactly `len(s) // 2` characters in `s` that are the same character.
- Return `True` if and only if the number of duplicate characters in `s` is equal to `len(s) // 2`.
- Return `True` if and only if `s[:len(s) // 2]` is the same as `s[len(s) // 2:]`.

---

1 point

4. In one of the Week 6 lecture videos, we wrote the function `shift_left`. Consider this function, which shifts in the other direction:

```
def shift_right(L):
    ''' (list) -> NoneType

    Shift each item in L one position to the rightand shift the    last item
        to the first position.

    Precondition: len(L) >= 1
    '''

    last_item = L[-1]

    # MISSING CODE GOES HERE

    L[0] = last_item
```

Select the code fragment that correctly completes function `shift_right`.

Hint: the correct answer works from the end to the beginning of `L`.

-
```
for i in range(len(L) - 1):
    L[i] = L[i + 1]
```
-
```
for i in range(1, len(L)):
    L[i] = L[i + 1]
```

```
for i in range(len(L)):
    L[i + 1] = L[i]
```

```
for i in range(1, len(L)):
    L[len(L) - i] = L[len(L) - i - 1]
```

1 point

5. Consider the code (these type contracts get a little tough to write!):

```
def make_pairs(list1, list2):
    ''' (list of str, list of int) -> list of [str, int] list

    Return a new list in which each item is a 2-item list with     the string
        from thecorresponding position of list1 and the     int from the
        corresponding position of list2.

    Precondition: len(list1) == len(list2)

    >>> make_pairs(['A', 'B', 'C'], [1, 2, 3])
    [['A', 1], ['B', 2], ['C', 3]]
    '''

    pairs = []

    # CODE MISSING HERE

    return pairs
```

Select the code fragment(s) that make the function above match its docstring description.

```
inner_list = []
for i in range(len(list1)):
    inner_list.append(list1[i])
    inner_list.append(list2[i])
    pairs.append(inner_list)
```

```
for i in range(len(list1)):
    inner_list = []
    inner_list.append(list1[i])
    inner_list.append(list2[i])
pairs.append(inner_list)
```

```
for i in range(len(list1)):
    pairs.append([list1[i], list2[i]])
```

```
for i in range(len(list1)):
    inner_list = []
    inner_list.append(list1[i])
    inner_list.append(list2[i])
    pairs.append(inner_list)
```

1 point

6. Consider this code:

```
1  numbers = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Using `numbers` and indexing with **non-negative indices**, write an expression that evaluates to `7`. Do not use addition, subtraction, or parentheses ( ) (brackets ([] are required).

Enter answer here

---

1 point

7. Consider this code:

```
1  breakfast = [['French', 'toast'], ['blueberry', 'pancakes'], ['scrambled',
       'eggs']]
```

Using `breakfast` and indexing with **only negative indices**, write an expression that evaluates to '`blueberry`'. Do not use addition, subtraction, or parentheses ( ) (brackets [ ] are required).

Enter answer here

---

1 point

8. Consider this code:

```
1  for i in range(2, 5):
2      for j in range(4, 9):
3          print(i, j)
```

Trace the code above in the Python Visualizer. How many times is `print(i, j)` executed?

- 3
- 15
- 24
- 5

---

1 point

9. Consider this code:

```
1  def contains(value, lst):
2      """ (object, list of list) -> bool
3
4      Return whether value is an element of one of the nested        lists in
           lst.
5
6      >>> contains('moogah', [[70, 'blue'], [1.24, 90, 'moogah'],    [80, 100]])
7      True
8      """
9
10     found = False  # We have not yet found value in the list.
11
12     # CODE MISSING HERE
13
14     return found
```

Select the code fragment(s) that make the function above match its docstring description.

- [ ]
```
1      for i in range(len(lst)):
2          for j in range(len(lst[i])):
3              found = (lst[i][j] == value)
```

- [ ]
```
1      for sublist in lst:
2          if value in sublist:
3              found = True
```

- [ ]
```
1      for item in lst:
2          if value == item:
3              value = True
```

- [ ]
```
1      for i in range(len(lst)):
2          for j in range(len(lst[i])):
3              if lst[i][j] == value:
4                  found = True
```

---

1 point

10. A file has a section at the top that has a preamble describing the contents of the file, then a blank line, then a list of high temperatures for each day in January all on one line, then a list of high temperatures for each day in February all on one line, then lists for March, April, and so on through December, each on one line. There are thousands of lines of information after that temperature data that you aren't currently interested in.

You want to write a program that prints the average of the high temperatures in January. Which of the four file-reading approaches should you use?

Hint: review the Reading Files lecture.

- ( ) The **for *line* in *file*** approach
- ( ) The **read** approach

- The **readlines** approach
- The **readline** approach

---

1 point

11. Consider this code:

```
1  # data_file refers to a file open for reading.
2  for line in data_file:
3      print(line)
```

The program above prints the lines of the file but adds an extra blank line after each line. Select the code fragment(s) that when used as replacement(s) for `print(line)` will print the lines without extra blank lines.

Note: use `help` to find out information about any functions or methods that you are not familiar with.

- `print(line.rstrip('\n'))`
- `print(line - '\n')`
- `print(line, end='')`
- `print(line.strip())`

---

1 point

12. Consider this code:

```
1  def lines_startswith(file, letter):
2      """ (file open for reading, str) -> list of str
3
4      Return the list of lines from file that begin with letter.     The lines
           should have thenewline removed.
5
6      Precondition: len(letter) == 1
7      """
8
9      matches = []
10
11     # CODE MISSING HERE
12
13     return matches
```

Select the code fragment(s) that make the function above match its docstring description.

```
1      matches.append(line.startswith(letter).rstrip('\n'))
```

```
for line in file:
    if line.startswith(letter):
        matches.append(line.rstrip('\n'))
```

```
for line in file:
    if letter in line:
        matches.append(line.rstrip('\n'))
```

```
for line in file:
    if letter == line[0]:
        matches.append(line.rstrip('\n'))
```

---

1 point

13. Consider this code:

```
def write_to_file(file, sentences):
    """ (file open for writing, list of str) -> NoneType

    Write each sentence from sentences to file, one per line.

    Precondition: the sentences contain no newlines.
    """

    # CODE MISSING HERE
```

Select the code fragment(s) that make the function above match its docstring description.

```
for s in sentences:
    file.write(s)
file.write('\n')
```

```
file.write(sentences)
```

```
for s in sentences:
    file.write(s + '\n')
```

```
for s in sentences:
    file.write(s)
```

```
for s in sentences:
    file.write(s)
    file.write('\n')
```

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