# Feedback — Week 5 Exercise

You submitted this homework on **Mon 23 Sep 2013 7:59 PM PDT (UTC -0700)**. You got a score of **13.75** out of **14.00**. You can attempt again, if you'd like.

These exercises are intended to get you to open IDLE and work on small pieces of code. If you have been copying/pasting/editing/running the code and trying out expressions in the Python shell, then congrats, you're doing the right thing!

If, on the other hand, you have been avoiding using IDLE, and you're struggling with the exercises, then you aren't learning as much as you should. Open up IDLE and try out all the code as you do the exercise!

## Question 1

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

| Your Answer | | Score | Explanation |
|---|---|---|---|
| ☐ `'3' in [1, 2, 3]` | ✓ | 0.25 | |
| ☑ `int('3') in [len('a'), len('ab'), len('abc')]` | ✓ | 0.25 | |
| ☑ `len([1, 2, 3]) == len(['a', 'b', 'c'])` | ✓ | 0.25 | |
| ☐ `[1, 2, 3] in len('mom')` | ✓ | 0.25 | |
| Total | | 1.00 / 1.00 | |

**Question Explanation**

Evaluate each of these expressions in the Python shell and, if the results surprise you, you should also evaluate each subexpression.

## Question 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.

| Your Answer | | Score | Explanation |
|---|---|---|---|
| [✓] `mystery('abc')` | ✓ | 0.25 | |
| [ ] `mystery('abc123')` | ✓ | 0.25 | |
| [ ] `mystery('123abc')` | ✓ | 0.25 | |
| [ ] `mystery('123')` | ✓ | 0.25 | |
| Total | | 1.00 / 1.00 | |

**Question Explanation**

Run the code. To understand why the error occurs (or does not), trace the code in the visualizer.

## Question 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`?

| Your Answer | | Score | Explanation |
|---|---|---|---|
| [✓ ●] Return a `list` containing every third *item* from `L` starting at index 0. | ✓ | 1.00 | |
| ○ 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 *index* from `L` starting at index 0. | | | |
| Total | | 1.00 / 1.00 | |

**Question Explanation**

Run the code several times, passing various lists as arguments, to gain a better understanding of what it does.

## Question 4

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

    Return a new list with adjacent pairs of string elements from L
    concatenated 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.

| Your Answer | | Score | Explanation |
|---|---|---|---|
| ● `i = i + 2` | ✓ | 1.00 | |
| ○ `i = i + 1` | | | |
| ○ `i = i + i` | | | |
| ○ `i = i * 2` | | | |
| Total | | 1.00 / 1.00 | |

**Question Explanation**

Run the code to gain a better understanding of what it does and what effect each choice has.

## Question 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`.

**You entered:**

27004383

| Your Answer | | Score | Explanation |
|---|---|---|---|
| 27004383 | ✓ | 1.00 | |
| Total | | 1.00 / 1.00 | |

**Question Explanation**

Start by solving a simpler problem, such as printing the odd integers between 1 and 9. Make sure that you print 1, 3, 5, 7, and 9. Then introduce a variable, perhaps called `sum`, that starts at 0, and in each iteration adds the current integer to `sum`. The sum of the odd numbers from 1 through 9, inclusive, is 25. Then reattempt the question.

## Question 6

Consider this code:

```
def while_version(L):
    """ (list of number) -> number
    """
    i = 0
    total = 0

    while i < len(L) and L[i] % 2 != 0:
        total = total + L[i]
        i = i + 1

    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?

| Your Answer | | Score | Explanation |
|---|---|---|---|
| ● def for_version(L):<br>    found_even = False<br>    total = 0<br><br>    for num in L:<br>        if num % 2 != 0 and not found_even:<br>            total = total + num<br>        else:<br>            found_even = True<br><br>    return total | ✓ | 1.00 | |
| ○ | | | |

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

    for num in L:
        if num % 2 != 0:
            total = total + num
        elif not found_even:
            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
```

---

○

```
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
```

---

| | |
|---|---|
| Total | 1.00 / 1.00 |

**Question Explanation**

Pay close attention to the variables involved in the loop condition: `i` and `L`. Read through the code or trace it in the visualizer to determine what that `while` loop does and when it exits. You should also run each of the `for_version` functions.

## Question 7

Consider this code:

```
>>> letters = ['b', 'd', 'a']
>>> # MISSING CODE HERE
>>> print(letters)
['a', 'b', 'd']
```

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

| Your Answer | | Score | Explanation |
|---|---|---|---|
| [✓] `letters.sort()` | ✓ | 0.25 | |
| ☐ `sort(letters)` | ✓ | 0.25 | |
| ☐ `letters = letters.sort()` | ✓ | 0.25 | |
| ☐ `letters = sort(letters)` | ✓ | 0.25 | |
| Total | | 1.00 / 1.00 | |

**Question Explanation**

Run the code and try each code fragment.

## Question 8

Consider this code:

```
veggies = ['carrot', 'broccoli', 'potato', 'asparagus']
veggies.insert(veggies.index('broccoli'), 'celery')
print(veggies)
```

What is printed by the code above?

| Your Answer | | Score | Explanation |
|---|---|---|---|
| [●] `['carrot', 'celery', 'broccoli', 'potato', 'asparagus']` | ✓ | 1.00 | |
| ○ `['carrot', 'celery', 'potato', 'asparagus']` | | | |

○ `['celery', 'carrot', 'broccoli', 'potato', 'asparagus']`

○ `['carrot', 'broccoli', 'celery', 'potato', 'asparagus']`

| | |
|---|---|
| Total | 1.00 / 1.00 |

**Question Explanation**

Run the code and read the help for `list.index` and `list.insert`.

## Question 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:

```
def cap_song_repetition(playlist, song):
    '''(list of str, str) -> NoneType

    Make sure there are no more than 3 occurrences of song in playlist.
    '''
```

Select the loop(s) that accomplish this.

| Your Answer | Score | Explanation |
|---|---|---|

| | | |
|---|---|---|
| ☑ while playlist.count(song) > 3:<br>    playlist.remove(song) | ✓ | 0.25 |
| ☐ while playlist.count(song) > 3:<br>    playlist.pop(playlist.index(song)) | ✗ | 0.00 |
| ☐ while playlist.count(song) >= 3:<br>    playlist.remove(song) | ✓ | 0.25 |
| ☐ while playlist.count(song) > 3:<br>    playlist.remove(playlist.index(song)) | ✓ | 0.25 |
| Total | | 0.75 / 1.00 |

**Question Explanation**

Look at `help(list.remove)` and `help(list.pop)` and see if you can determine which uses are correct.
You can also enter the different function completions into IDLE and test them on an equivalent example:

```
>>> playlist = ['A', 'B', A', 'A', 'C', 'A', 'D', 'E', 'A', 'A']
>>> cap_song_repetition(playlist, 'A')
>>> print(playlist)
```

## Question 10

Consider this code:

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

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

| Your Answer | Score | Explanation |
|---|---|---|

| | | | |
|---|---|---|---|
| ☑ a[1] = 'A' | ✓ | 0.25 | |
| ☑ b[-2] = 'A' | ✓ | 0.25 | |
| ☑ a = [1, 'A', 3]<br>b = [1, 'A', 3] | ✓ | 0.25 | |
| ☑ b[1] = 'AB'<br>a[1] = a[1][0] | ✓ | 0.25 | |
| Total | | 1.00 / 1.00 | |

**Question Explanation**

Run the code, substituting each code fragment in for the missing code. Note that if a[1] is "CAT", then a[1][0] is "C".

## Question 11

Consider this code:

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

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

| Your Answer | | Score | Explanation |
|---|---|---|---|
| ☑ a = [1, 'A', 3]<br>b = [1, 'A', 3] | ✓ | 0.25 | |
| ☐ a[1] = 'A' | ✓ | 0.25 | |
| ☐ b[1] = 'AB' | ✓ | 0.25 | |

| | | | |
|---|---|---|---|
| a[1] = a[1][0] | | | |
| ☐ b[-2] = 'A' | ✓ | 0.25 | |
| Total | | 1.00 / 1.00 | |

**Question Explanation**

Run the code, substituting each code fragment in for the missing code. Note that if a[1] is "CAT", then a[1][0] is "C".

## Question 12

Consider this code:

```
def increment_items(L, increment):
    i = 0
    while i < len(L):
        L[i] = L[i] + increment
        i = i + 1

values = [1, 2, 3]
print(increment_items(values, 2))
print(values)
```

What is printed by the program above?

| Your Answer | | Score | Explanation |
|---|---|---|---|
| ● None<br>[3, 4, 5] | ✓ | 1.00 | |
| ○ None<br>[1, 2, 3] | | | |
| ○ [3, 4, 5]<br>None | | | |
| ○ [3, 4, 5] | | | |

[1, 2, 3]

| | | |
|---|---|---|
| Total | 1.00 / 1.00 | |

**Question Explanation**

Run the code.

## Question 13

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

| Your Answer | | Score | Explanation |
|---|---|---|---|
| ✓ values = [] for num in range(3, 10, 3): values.append(num) print(values) | ✓ | 0.25 | |
| ✓ values = [] for num in range(1, 4): values.append(num * 3) print(values) | ✓ | 0.25 | |
| ☐ values = [] for num in range(3, 9, 3): values.append(num) print(values) | ✓ | 0.25 | |
| ☐ values = [] for num in range(1, 3): values.append(num * 3) print(values) | ✓ | 0.25 | |
| Total | | 1.00 / 1.00 | |

**Question Explanation**

Call `help(range)` to learn about the function `range` and its two optional arguments.

## Question 14

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

```
for num in __________________:
    print(num)
```

The loop should print this:

```
3
11
19
```

| Your Answer | | Score | Explanation |
|---|---|---|---|
| [✓] `range(3, 20, 8)` | ✓ | 0.25 | |
| [✓] `range(3, 23, 8)` | ✓ | 0.25 | |
| [ ] `range(3, 8, 20)` | ✓ | 0.25 | |
| [ ] `range(3, 19, 8)` | ✓ | 0.25 | |
| Total | | 1.00 / 1.00 | |

**Question Explanation**

Call `help(range)` to learn about the function `range` and it's two optional arguments.