1 point

1. Select the expression(s) that evaluate to a `float` value.

- [ ] `8.0 % 4`
- [ ] `7 + 8.5`
- [ ] `8 % 6`
- [ ] `3 // 4`

---

1 point

2. Consider this code:

```
1  a = 7
2  b = a + 3
3  a = 9
```

After the code above has been executed, what value does `b` refer to?

10

---

1 point

3. Consider this code:

```
1  def f(y):
2      x = y * 3
3      return y + x
```

What value is returned by a call on function `f` with argument `10`?

40

---

1 point

4. Consider this code:

```
1  start = 'L'
2  middle = 8
3  end = 'R'
```

Write an expression that evaluates to the string 'L8R' using only the variables `start`, `middle`, `end`, one call on function `str`, and string concatenation.

Do not use unnecessary parentheses: you need them for the function call, but nothing else.

```
start+str(middle)+end
```

---

1 point

5. Consider this function:

```
def larger_of_smallest(L1, L2):
    '''(list of int, list of int) -> int

    Return the larger of the smallest value in L1 and the small    est value
        inL2.

    Precondition: L1 and L2 are not empty.

    >>> larger_of_smallest([1, 4, 0], [3, 2])
    2
    >>> larger_of_smallest([4], [9, 6, 3])
    4
    '''

   return # CODE MISSING HERE
```

The expression for the `return` statement is missing. Write it below. Use only the parameters, one call on function `max`, and two calls on function `min`.

Do not use unnecessary parentheses: you need them for the function calls, but nothing else. Do not include the word `return`; just write the expression.

```
max(min(L1),min(L2))
```

---

1 point

6. Consider this function:

```
def same_length(L1, L2):
    '''(list, list) -> bool

    Return True if and only if L1 and L2 contain the same number of elements.
    '''

    if len(L1) == len(L2):
       return True
    else:
       return False
```

The function works, but the `if` statement can be replaced with a single `return` statement:

```
1  return # CODE MISSING HERE
```

Write the missing expression. Use only the parameters, two calls on function `len`, and operator `==` once.

Do not use unnecessary parentheses: you need them for the function calls, but nothing else. Do not include the word `return`; just write the expression.

len(L1) == len(L2)

---

1 point

7. Consider these two functions; we provide only the headers, type contracts, and a precondition:

```
1  def moogah(a, b):
2      '''(str, int) -> str'''
```

```
1  def frooble(L):
2      '''(list of str) -> int
3      Precondition: L has at least one element.'''
```

Below are code fragments that call these two functions in various ways. Select the code fragment(s) below that are valid according to the function headers and the type contracts.

- [ ] `moogah('a', frooble(['a']))`
- [ ] `moogah('a', moogah(['a']))`
- [ ] `moogah(frooble(['a']), 'a')`
- [ ] `lst = ['a', 'b', 'c']`
  `moogah(lst[0], len(lst))`

---

1 point

8. Consider this code:

```
def gather_every_nth(L, n):
    '''(list, int) -> list

    Return a new list containing every n'th element in L, starting at index 0
        .

    Precondition: n >= 1

    >>> gather_every_nth([0, 1, 2, 3, 4, 5], 3)

    [0, 3]
    >>> gather_every_nth(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i'], 2)
    ['a', 'c', 'e', 'g', 'i']
    '''

    result = []
    i = 0
    while i < len(L):
        result.append(L[i])
        i = # CODE MISSING HERE

    return result
```

Write the missing expression. Do not use parentheses. Do not include "`i =`". Just write the missing expression.

i + n

---

1 point

9. Consider this code:

```
def get_keys(L, d):
    '''(list, dict) -> list

    Return a new list containing all the items in L that are keys in d.

    >>> get_keys([1, 2, 'a'], {'a': 3, 1: 2, 4: 'w'})
    [1, 'a']
    '''

    result = []
    for # CODE MISSING HERE
       if k in d:
          result.append(k)

    return result
```

Write the missing code for the first line of the `for` loop — everything after the word "`for`", up to and including the colon **:**.

Do not use any parentheses, and do not call any functions or methods.

k in L:

---

1 point

10. Consider this code:

```
1  def are_lengths_of_strs(L1, L2):
2      '''(list of int, list of str) -> bool
3
4      Return True if and only if all the ints in L1 are the lengths of the
           strings
5      in L2 at the corresponding positions.
6
7      Precondition: len(L1) == len(L2)
8
9      >>> are_lengths_of_strs([4, 0, 2], ['abcd', '', 'ef'])
10     True
11     '''
12
13     result = True
14     for i in range(len(L1)):
15         if # CODE MISSING HERE
16             result = False
17
18     return result
```

Write the missing code for the `if` statement — everything after the word "`if`", up to and including the colon **:**.

Your answer should be of the form `expr1 != expr2:`, where `expr1` and `expr2` are expressions. Use only variables `i`, `L1`, `L2`, indexing, and function `len`.

Do not use parentheses except for the call on `len`.

```
L1[i] != len(L2[i]):
```

---

1 point

11. Consider this function:

```
1  def double_values(collection):
2      for v in range(len(collection)):
3          collection[v] = collection[v] * 2
```

Strings, tuples, lists, and dictionaries can all be iterated over, and function `len` works with all of them.

Select the code fragment(s) below that run without error.

- [ ]
```
1  L = [1, 2, 3]
2  double_values(L)
```

- [ ]
```
1  d = {0: 10, 1: 20, 2: 30}
2  double_values(d)
```

- [ ]
```
1  s = '123'
2  double_values(s)
```

```
t = (1, 2, 3)
double_values(t)
```

```
d = {1: 10, 2: 20, 3: 30}
double_values(d)
```

---

1 point

12. Consider this function:

```
def get_negative_nonnegative_lists(L):
    '''(list of list of int) -> tuple of (list of int, list of int)

    Return a tuple where the first item is a list of the negative ints in the
    inner lists of L and the second item is a list of the non-negative ints
    in those inner lists.

    Precondition: the number of rows in L is the same as the number of
    columns.

    >>> get_negative_nonnegative_lists([[-1,  3,  5], [2,  -4,  5], [4,  0,
        8]])
    ([-1, -4], [3, 5, 2, 5, 4, 0, 8])
    '''

    nonneg = []
    neg = []
    for row in range(len(L)):
        for col in range(len(L)):
            # CODE MISSING HERE

    return (neg, nonneg)
```

Select the code fragment(s) that correctly complete this function.

```
            val = L[row][col]
            if val < 0:
                neg.append(val)
            else:
                nonneg.append(val)
```

```
            if L[row][col] < 0:
                neg.append(L[row][col])

            nonneg.append(L[row][col])
```

```
            if L[row][col] < 0:
                neg.append(L[row][col])
            else:
                nonneg.append(L[row][col])
```

```
if L[row][col] < 0:
    nonneg.append(L[row][col])
else:
    neg.append(L[row][col])
```

```
if L[row][col] > 0:
    nonneg.append(L[row][col])
else:
    neg.append(L[row][col])
```

Final Exam

Quiz, 13 questions

1 point

13. Consider this code:

```
def count_chars(s):
    '''(str) -> dict of {str: int}

    Return a dictionary where the keys are the characters in s and the values
    are how many times those characters appear in s.

    >>> count_chars('abracadabra')
    {'a': 5, 'r': 2, 'b': 2, 'c': 1, 'd': 1}
    '''
    d = {}

    for c in s:
        if not (c in d):
            # CODE MISSING HERE
        else:
            d[c] = d[c] + 1

    return d
```

Write the missing assignment statement. Do not call any functions or methods. Do not use unnecessary parentheses.

d[c] = 1

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