Level 1 (16)

Filename: L1.py

1. Complete the code in function:

def all_equal_int(x1, x2, x3, x4):

that returns True if all input integers, x1, x2, x3, and x4 are equal.

Here is the Doctest for this function:

```
>>> all_equal_int(3, 3, 3, 3)
True

>>> all_equal_int(3, 2, 2, 3)
False

>>> all_equal_int(3, 2, 2, 2)
False

>>> all_equal_int(3, 3, 3, 2)
False
```

2. Complete the code in function:

def circles_overlapping(x1, y1, x2, y2, r):

that returns True if two circles with the same radius, r, located at (x1, y1) and (x2, y2) overlapped; return False, otherwise.

Here is the Doctest for this function:

```
>>> circles_overlapping(0, 0, 2, 0, 2)
True

>>> circles_overlapping(2, 2, 4, 2, 0.7)
```

False

>>> circles_overlapping(1, 1, 2, 2, 0.6)
False

>>> circles_overlapping(1, 1, 4, 5, 3)
True

Level 2 (16)

Filename: L2.py

1. Complete the code in function:

def num_even_digits(n):

that returns the total number of even digits in n

Here is the Doctest for this function:

>>> num_even_digits(223345)
3

>>> num_even_digits(973315)
0

>>> num_even_digits(88660022)
8

>>> num_even_digits(12345670)
4

2. Complete the code in function:

def possible_rectangle(size1, size2, size3, size4):

that
returns True if size1, size2, size3, and size4 could form a rectangle (when two pair of sizes

are equal)

Here is the Doctest for this function:

```
>>> possible_rectangle(10, 20, 20, 10)
True

>>> possible_rectangle(10, 30, 20, 10)
False

>>> possible_rectangle(10, 10, 10, 10)
True

>>> possible_rectangle(10, 10, 20, 10)
False
```

Level 3 (16)

Filename: L3.py

1. Complete the code in function:

def string_union(str1, str2):

that
returns a union of the characters in the two input strings str1 and str2; you can assume that the two strings have no duplicate characters

Here is the Doctest for this function:

```
>>> string_union("abcde", "abcwxyz")
'abcdewxyz'

>>> string_union("abcde", "wxyz")
'abcdewxyz'

>>> string_union("abcde", "wxyabzc")
'abcdewxyz'
```

2. Complete the code in function:

def string_intersect(str1, str2):

that returns an intersection of the characters in the two input strings str1 and str2;
you can assume that the two strings have no duplicate characters

Here is the Doctest for this function:

```
>>> string_intersect("abcde", "abcwxyz")
'abc'

>>> string_intersect("abcde", "wxyz")
''

>>> string_intersect("abcde", "wxyabzc")
'abc'
```

Level 4 (12)

Filename: L4.py

1. Complete the code in function:

def group_elements(l):

that returns a list of lists where each list groups the same element(s) from the input list, l, together

Here is the Doctest for this function:

```
>>> group_elements([1, 3, 3, 4, 2, 5, 8, 5, 6, 7])
[[1], [3, 3], [4], [2], [5, 5], [8], [6], [7]]

>>> group_elements([3, 3, 3, 3])
[[3, 3, 3, 3]]

>>> group_elements([3, 3, 1, 3, 3, 1])
```

```
[[3, 3, 3, 3], [1, 1]]

>>> group_elements([5, 3, 1, 2, 4, 10])
[[5], [3], [1], [2], [4], [10]]
```

2. Complete the code in function:

def dup_elements(l):

that returns a list that contains only duplicate elements of the input list, l; use the group_elements to help solve this one

Here is the Doctest for this function:

```
>>> dup_elements([1, 3, 3, 4, 2, 5, 8, 5, 6, 7])
[3, 5]

>>> dup_elements([3, 3, 3, 3])
[3]

>>> dup_elements([3, 3, 1, 3, 3, 1])
[3, 1]

>>> dup_elements([5, 3, 1, 2, 4, 10])
```

3. Complete the code in function:

def count_unique_elements(l):

that returns the total number of unique elements in the input list, l; use the group_elements to help solve this one

Here is the Doctest for this function:

```
>>> count_unique_elements([1, 3, 3, 4, 2, 5, 8, 5, 6, 7])
```

```
6

>>> count_unique_elements([3, 3, 3, 3])
0

>>> count_unique_elements([3, 3, 1, 3, 3, 1])
0

>>> count_unique_elements([5, 3, 1, 2, 4, 10])
6
```