# Launching into Computer Science Assignment 1: Part 2

## Question 1:

- Code a function called 'first_div_16'.
- ACCEPT two positive integers, n1 and n2, as inputs.
- RETURN the first number in range(n1,n2) that is divisible by 16.
- HOWEVER, if no number in the range is divisible by 16 RETURN 0.

Codio screenshot of Qu_1.py:

Qu_1.py × | Qu_2.py | Qu_3.py | Qu_4.py | Qu_5.py | Terminal

```python
def first_div_16(n1, n2):
  #check that given conditions are satisfied.
  if type(n1) != int or type(n2) != int or n1 < 1 or n2 < 1 or n2 < n1:
    return "Error: Inputs (n1, n2) must satisfy 0 < n1 < n2, where n1 and n2 are positive integers."
  #while loop checks each number in the given range in ascending order.
  while n1 <= n2:
    #use modulo % to check for remainder when dividing by 16.
    if n1 % 16 != 0:
      n1 += 1
    #remainder 0 indicates divisible by 16, so return this value.
    else:
      return n1
  #returns 0 if no value in the range is divisible by 16.
  #also returns 0 if initial input satisfies n2 > n1.
  if n1 > n2:
    return 0


#test 1 for finding divisible number: expect 16.
print("Test 1:", first_div_16(1, 16))
#test 2 for stopping at first divisible number: expect 32.
print("Test 2:", first_div_16(20, 50))
#test 3 for returning zero when no divisible numbers: expect 0.
print("Test 3:", first_div_16(23, 27))
#test 4 for when n1 = n2: expect 64.
print("Test 4:", first_div_16(64, 64))
#test 5 for type error: expect error message.
print("Test 5:", first_div_16(1.2, 16))
#test 6 for sign error: expect error message.
print("Test 6:", first_div_16(-16, -1))
#test 7 for order error: expect error message.
print("Test 7:", first_div_16(32, 1))
```

Terminal screenshot of tests with dummy data:

```
codio@warning-signal:~/workspace$ python3 Qu_1.py
Test 1: 16
Test 2: 32
Test 3: 0
Test 4: 64
Test 5: Error: Inputs (n1, n2) must satisfy 0 < n1 < n2, where n1 and n2 are positive integers.
Test 6: Error: Inputs (n1, n2) must satisfy 0 < n1 < n2, where n1 and n2 are positive integers.
Test 7: Error: Inputs (n1, n2) must satisfy 0 < n1 < n2, where n1 and n2 are positive integers.
```

All tests return expected values seen in the comments in the Codio screenshot above.

**Question 2:**

- Code a function called 'halve_to_2'.
- ACCEPT one numeric input.
- If the number <= 0, RETURN -1.
- If the number > 0, divide that integer over-and-over by 2 until it becomes smaller than 2.
- RETURN that smaller-than-2 number, e.g. input of 4 Will yield 1 (4->2->1), 5 yields 1.25 (5->2.5->1.25) etc.

Codio screenshot of Qu_2.py:

Qu_1.py | Qu_2.py | Qu_3.py | Qu_4.py | Qu_5.py | Terminal

```python
def half_to_2(i):
  #check if input satisfies specified input condition.
  if type(i) != int and type(i) != float:
    return "Error: input must be a real number."
  #check for if input i <= 0 and return -1 if so.
  if i <= 0:
    return -1
  #while loop to repeatedly halve input i until i < 2 and return i.
  while i >= 2:
    i /= 2
  return i


#test 1 for when input i <= 0: expect -1.
print("Test 1:", half_to_2(-5))
#test 2 for when input 0 < i < 2: expect 0.5.
print("Test 2:", half_to_2(0.5))
#test 3 for when input i = 2: expect 1.
print("Test 3:", half_to_2(2))
#test 4 for when input i = 4 as in question: expect 1.
print("Test 4:", half_to_2(4))
#test 5 for when input i = 5 as in question: expect 1.25.
print("Test 5:", half_to_2(5))
#test 6 for type error: expect error message.
print("Test 6:", half_to_2("5"))
```

Terminal screenshot of tests with dummy data:

```
codio@warning-signal:~/workspace$ python3 Qu_2.py
Test 1: -1
Test 2: 0.5
Test 3: 1.0
Test 4: 1.0
Test 5: 1.25
Test 6: Error: input must be a real number.
```

All tests return expected values seen in the comments in the Codio screenshot above.

**Question 3:**

- Code a function called 'string_expansion'.
- ACCEPT a non-empty string as input.
- RETURN a string that contains every other character, 2n+2 times, where n is the original index of the letter. e.g. Input of "Hello" should result in "HHllllllooooooooooo". Input of "ROBErt" should result in "RRBBBBBBrrrrrrrrrr".

Codio screenshot of Qu_3.py:

Qu_1.py | Qu_2.py | Qu_3.py | Qu_4.py | Qu_5.py | Terminal

```python
def string_expansion(user_string):
  #check given conditions are satisfied.
  if type(user_string) != str or user_string == "":
    return "Error: Input must be a non-empty string."
  #i for passing through input user_string.
  i = 0
  #new_string for creating output string.
  new_string = ""
  #while loop starts at index 0 and jumps 2 each time, i.e. 0, 2, 4, 6...
  while i < len(user_string):
    #takes user_string[i], multiplies it by (2*i +2), and adds it to the end of new_string.
    new_string += (2*i + 2) * user_string[i]
    i += 2
  return new_string


#test 1 "Hello" as in question: expect "HHllllllooooooooooo"
print("Test 1:", string_expansion("Hello"))
#test 2 "ROBErt" as in question: expect "RRBBBBBBrrrrrrrrrr"
print("Test 2:", string_expansion("ROBErt"))
#test 3 for type error: expect error message.
print("Test 3:", string_expansion(12345))
#test 4 for empty string: expect error message.
print("Test 4:", string_expansion(""))
```

Terminal screenshot of tests with dummy data:

```
codio@warning-signal:~/workspace$ python3 Qu_3.py
Test 1: HHllllllooooooooooo
Test 2: RRBBBBBBrrrrrrrrrr
Test 3: Error: Input must be a non-empty string.
Test 4: Error: Input must be a non-empty string.
```

All tests return expected values seen in the comments in the Codio screenshot above.

**Question 4:**

- Code a function called 'item_count_from_index'.
- ACCEPT two inputs, a list and an integer-index.
- RETURN a count (number) of how many times the item at that index appears in the list.
- HOWEVER, if the integer-index is out of bounds for the list RETURN the empty string ("") (e.g. list of 3 items, index of 5 is out of bounds).

Codio screenshot of Qu_4.py:

Qu_1.py | Qu_2.py | Qu_3.py | Qu_4.py | Qu_5.py | Terminal

```python
def item_count_from_index(user_list, item_index):
  #check inputs satisfy given conditions.
  if type(user_list) != list or type(item_index) != int:
    return "Error: inputs must follow the (list, integer) format."
  #check if given index is within the bounds of the given list.
  #this checks for positive and negative indexes.
  if item_index >= len(user_list) or abs(item_index) > len(user_list):
    return ""
  #create variable for term which is being counted in the list.
  count_term = user_list[item_index]
  #use .count() to count the amount of times the term appears in the input list.
  return user_list.count(count_term)


#test lists.
int_list = [1, 2, 3, 4, 3, 2, 4, 7, 9, 7, 5, 3, 5, 8, 9, 6, 42, 2, 1, 3, 4, 6]
string_list = ["red", "blue", "green", "brown", "green", "blue", "green"]

#test 1 for index 0 and integer list: expect 2.
print("Test 1:", item_count_from_index(int_list, 0))
#test 2 for positive index and string list: expect 3.
print("Test 2:", item_count_from_index(string_list, 2))
#test 3 for positive inedex outside range out list: expect ""
print("Test 3:", item_count_from_index(string_list, 8))
#test 4 for negative index: expect 1
print("Test 4:", item_count_from_index(int_list, -6))
#test 5 for negative index outside range: expect ""
print("Test 5:", item_count_from_index(string_list, -10))
#test 6 for list type error: expect error messsage.
print("Test 6:", item_count_from_index("List", 2))
#test 7 for int type error: expect error messsage.
print("Test 7:", item_count_from_index(int_list, "1"))
```

Terminal screenshot of tests with dummy data:

```
codio@warning-signal:~/workspace$ python3 Qu_4.py
Test 1: 2
Test 2: 3
Test 3:
Test 4: 1
Test 5:
Test 6: Error: inputs must follow the (list, integer) format.
Test 7: Error: inputs must follow the (list, integer) format.
```

All tests return expected values seen in the comments in the Codio screenshot above.

**Question 5:**

- Code a function called 'length_times_largest'.
- ACCEPT a list as input.
- RETURN the length of the list times the largest integer (not float) in the list.

  HOWEVER, if the list does not contain an integer, RETURN the empty string ("").

Codio screenshot of Qu_5.py:

Qu_1.py | Qu_2.py | Qu_3.py | Qu_4.py | Qu_5.py | Terminal

```python
def length_times_largest(num_list):
  #check input is list as specified.
  if type(num_list) != list:
    return "Error: input must be a list."
  #initialise list for storing integers only.
  int_list = []
  #for loop to pass over the input list and add the integers to int_list.
  for i in range(len(num_list)):
    #check type of each item in list and add to int_list if integer.
    if type(num_list[i]) == int:
      int_list.append(num_list[i])
  #if int_list is empty then no integers in input list return "" as instructed.
  if len(int_list) == 0:
    return ""
  #intialise variable to calculate
  #length of the input list multiplied by the max value in int_list.
  len_x_max = len(num_list) * max(int_list)
  return len_x_max


#test lists
int_list = [1, 3, 43, 34, 1.3, 24, 73, 74]
float_list = [1.2, 2.4, 3.4, 2.1]
mix_list = [1, 3, 4.5, 7.8, 9, 11.4, 10.3]

#test 1 for integer list only: expect 592
print("Test 1:", length_times_largest(int_list))
#test 2 for no integers in list: expect ""
print("Test 2:", length_times_largest(float_list))
#test 3 for mixed type list: expect 63
print("Test 3:", length_times_largest(mix_list))
#test 4 for type error: expect error message.
print("Test 4:", length_times_largest(12345))
```

Terminal screenshot of tests with dummy data:

```
Test 7: Error: inputs must follow the (list, integer) format.
codio@warning-signal:~/workspace$ python3 Qu_5.py
Test 1: 592
Test 2:
Test 3: 63
Test 4: Error: input must be a list.
```

All tests return expected values seen in the comments in the Codio screenshot above.