Started on	Sunday, 23 August 2020, 10:30 PM
State	Finished
Completed on	Monday, 24 August 2020, 10:23 PM
Time taken	23 hours 53 mins
Grade	**11.90** out of 12.00 (**99**%)

Question **1**

Correct

Mark 1.00 out of 1.00

This question uses a tuple to represent a student by their student id (an integer) and name. For example: `(5, "Smith")` represents a student with name as "Smith" and student id: 5.

Below is an implementation of the insertion sort algorithm that takes a list of numbers and sorts them in ascending order. Modify this implementation to instead sort a list of student tuples by their **name** in ***descending*** order.

```
def insertion_sort(data):
    for index in range(1, len(data)):
        item_to_insert = data[index]
        i = index - 1
        while i >= 0 and data[i] > item_to_insert:
            data[i + 1] = data[i]
            i -= 1
        data[i + 1] = item_to_insert
```

For example:

Test	Result

`s1 = (7, "Smith")`
`s2 = (5, "Brown")`
`s3 = (4, "Chan")`
`s4 = (2, "Kim")`
`s5 = (0, "Lin")`
`s6 = (1, "Singh")`
`s7 = (3, "Gupta")`
`students = [s1, s2, s3, s4, s5, s6, s7]`
`insertion_sort(students)`
`for x in students:`
`  print(x)`
	`(7, 'Smith')`
	`(1, 'Singh')`
	`(0, 'Lin')`
	`(2, 'Kim')`
	`(3, 'Gupta')`
	`(4, 'Chan')`
	`(5, 'Brown')`

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

```
def insertion_sort(data):
    for index in range(1, len(data)):
        item_to_insert = data[index]
        i = index - 1
        while i >= 0 and data[i][1] < item_to_insert[1]:
            data[i+1] = data[i]
            i -= 1
        data[i + 1] = item_to_insert
```

	Test	Expected	Got	

✔	`s1 = (7, "Smith")`
	`s2 = (5, "Brown")`
	`s3 = (4, "Chan")`
	`s4 = (2, "Kim")`
	`s5 = (0, "Lin")`
	`s6 = (1, "Singh")`
	`s7 = (3, "Gupta")`
	`students = [s1, s2, s3, s4, s5, s6, s7]`
	`insertion_sort(students)`
	`for x in students:`
	`  print(x)`
		`(7, 'Smith')`
		`(1, 'Singh')`
		`(0, 'Lin')`
		`(2, 'Kim')`
		`(3, 'Gupta')`
		`(4, 'Chan')`
		`(5, 'Brown')`
			`(7, 'Smith')`
			`(1, 'Singh')`
			`(0, 'Lin')`
			`(2, 'Kim')`
			`(3, 'Gupta')`
			`(4, 'Chan')`
			`(5, 'Brown')`
				✔

	Test	Expected	Got	

✔	`s1 = (1, "Williams")`
	`s2 = (2, "Wilson")`
	`s3 = (3, "Taylor")`
	`s4 = (4, "Wang")`
	`s5 = (5, "Jones")`
	`s6 = (6, "Patel")`
	`s7 = (7, "Li")`
	`students = [s1, s2, s3, s4, s5, s6, s7]`
	`insertion_sort(students)`
	`for x in students:`
	`    print(x)`
		`(2, 'Wilson')`
		`(1, 'Williams')`
		`(4, 'Wang')`
		`(3, 'Taylor')`
		`(6, 'Patel')`
		`(7, 'Li')`
		`(5, 'Jones')`
			`(2, 'Wilson')`
			`(1, 'Williams')`
			`(4, 'Wang')`
			`(3, 'Taylor')`
			`(6, 'Patel')`
			`(7, 'Li')`
			`(5, 'Jones')`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Fix the faulty function below called `double(my_list)` which is supposed to take a list, my_list, as a parameter and multiply each item in the list by 2. The function must change the original list (my_list) in place, not create a new list.

A faulty solution has been provided below:

```
def double(my_list):
    for index in range(my_list):
        my_list[index] = index * 2
```

Identify the faults and submit a corrected version of this code.

For example:

Test	Result

`my_list = [1, 2, 3]`
`double(my_list)`
`print(my_list)`
	`[2, 4, 6]`

`numbers = [3, 2, 1]`
`id_before = id(numbers)`
`double(numbers)`
`print(numbers)`
`id_after = id(numbers)`
`print(id_before == id_after) #same reference before and after`
	`[6, 4, 2]`
	`True`

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

Reset answer

```
def double(my_list):
    for index in range(len(my_list)):
        my_list[index] = my_list[index] * 2
```

	Test	Expected	Got	

✔	`my_list = [1, 2, 3]`
	`double(my_list)`
	`print(my_list)`
		`[2, 4, 6]`	`[2, 4, 6]`	✔

✔	`numbers = [3, 2, 1]`
	`id_before = id(numbers)`
	`double(numbers)`
	`print(numbers)`
	`id_after = id(numbers)`
	`print(id_before == id_after) #same reference before and after`
		`[6, 4, 2]`
		`True`
			`[6, 4, 2]`
			`True`
				✔

✔	`my_list = [1, 2, 3]`
	`double(my_list)`
	`print(type(my_list))`
		`<class 'list'>`	`<class 'list'>`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **3**

Correct

Mark 1.00 out of 1.00

Define the `get_middle_number(numbers)` function which is passed a list of integers as a parameter. The function returns the middle number from the list if the list were sorted. The function returns the element at the index given by the length of the list divided by 2 using floor division. For example, the following code:

```
numbers1 = [20, 24, 3, 8, 9]  #if sorted [3, 8, 9, 20, 24]
numbers2 = [15, 28, 22, 21]   #if sorted [15, 21, 22, 28]
numbers3 = [18, 9, 8, 5, 12, 25, 4, 3, 7].  #if sorted [3, 4, 5, 7, 8, 9, 12, 18, 25]

print("1.", get_middle_number(numbers1))
print("2.", get_middle_number(numbers2))
print("3.", get_middle_number(numbers3))
```

prints:

```
1. 9
2. 22
3. 8
```

For example:

Test	Result

`numbers3 = [18, 9, 8, 5, 12, 25, 4, 3, 7]  #if sorted [3, 4, 5, 7, 8, 9, 12, 18, 25]`
`numbers4 = [8, 24, 4, 10, 10, 25, 23, 21, 24, 5, 4, 6, 23, 23, 19]`
`print("3.", get_middle_number(numbers3))`
`print("4.", get_middle_number(numbers4))`
	`3. 8`
	`4. 19`

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

```
def get_middle_number(numbers):
    numbers = sorted(numbers)
    index = len(numbers)//2
    return numbers[index]
```

	Test	Expected	Got	

✔	`numbers1 = [20, 24, 3, 8, 9]  #if sorted [3, 8, 9, 20, 24]`
	`numbers2 = [15, 28, 22, 21]   #if sorted [15, 21, 22, 28]`
	
	`print("1.", get_middle_number(numbers1))`
	`print("2.", get_middle_number(numbers2))`
		`1. 9`
		`2. 22`
			`1.`
			`9`
			`2.`
			`22`
				✔

✔	`numbers3 = [18, 9, 8, 5, 12, 25, 4, 3, 7]  #if sorted [3, 4, 5, 7, 8, 9, 12, 18, 25]`
	`numbers4 = [8, 24, 4, 10, 10, 25, 23, 21, 24, 5, 4, 6, 23, 23, 19]`
	`print("3.", get_middle_number(numbers3))`
	`print("4.", get_middle_number(numbers4))`
		`3. 8`
		`4. 19`
			`3.`
			`8`
			`4.`
			`19`
				✔

Passed all tests! ✔

Correct

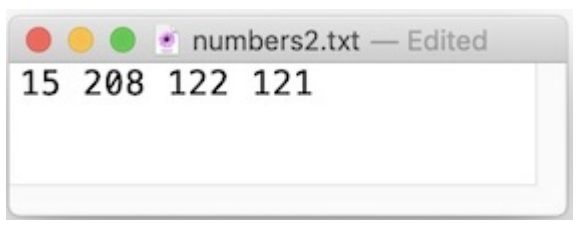
Marks for this submission: 1.00/1.00.

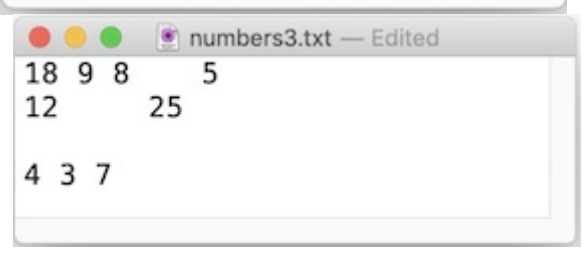
Question **4**

Correct

Mark 1.00 out of 1.00

Define the `get_middle_number_from_file(filename)` function which is passed the name of a file as a parameter. The file is made up of integer numbers separated by white space and may contain one or more lines. The function returns the middle number from the numbers read from the file. Three example files are shown below:






```
print("1.", get_middle_number_from_file('numbers1.txt'))
print("2.", get_middle_number_from_file('numbers2.txt'))
print("3.", get_middle_number_from_file('numbers3.txt'))
```

prints:

```
1. 24
2. 122
3. 8
```

You may wish to reuse the *get_middle_number(numbers)* function developed in the previous question.

For example:

Test	Result
`print("1.", get_middle_number_from_file('numbers1.txt'))` `print("2.", get_middle_number_from_file('numbers2.txt'))`	`1. 24` `2. 122`

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

```
def get_middle_number_from_file(filename):
    file = open(filename, "r")
    contents = file.read()
    file.close()
    contents = contents.split()
    contents = [int(item) for item in contents]
    contents = sorted(contents)
    index = len(contents)//2
    return contents[index]

```

	Test	Expected	Got	
✔	`print("1.", get_middle_number_from_file('numbers1.txt'))` `print("2.", get_middle_number_from_file('numbers2.txt'))`	`1. 24` `2. 122`	`1. 24` `2. 122`	✔
✔	`print("3.", get_middle_number_from_file('numbers3.txt'))`	`3. 8`	`3. 8`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **5**

Correct

Mark 0.95 out of 1.00

Continuing on from the previous question, we would like to improve the `get_middle_number_from_file(filename)` function. The function should be able to return the middle number from the numbers read from the text file even though the text file contains some invalid values, as in the examples below.

Problem 1: For example, if the file contains the following:

```
12 3              a 5   67
```

Your function should just ignore the letter 'a', continue to read the remaining integers and return 12. You also need to print an error message for **each** invalid value.

Problem 2: If the file contains the following:

```
5 5.5 6 2.5 8
```

Your function should just ignore 5.5 and 2.5 floating point numbers, continue to read the remaining integers and return 6. You also need to print an error message for **each** invalid value.

Problem 3: If the file is empty or contains whitespace only or contains invalid values only, your function should return an error message, e.g. ERROR: "test1.txt" is empty.

Problem 4: If the file does not exist in the current folder, your function should return an error message, e.g ERROR: "test0.txt" does not exist.

Note: you can assume that the get_middle_number(numbers) function has been done for you. Download all sample text files here

For example:

Test	Result
`print(get_middle_number_from_file('test0.txt'))`	`ERROR: "test0.txt" does not exist.`
`print(get_middle_number_from_file('test1.txt'))`	`ERROR: "test1.txt" is empty.`
`print(get_middle_number_from_file('test2.txt'))`	`12`

`print(get_middle_number_from_file('test8.txt'))`	`ERROR: "test8.txt" contains an invalid value.`
	`ERROR: "test8.txt" contains an invalid value.`
	`6`

`print(get_middle_number_from_file('test9.txt'))`	`ERROR: "test9.txt" contains an invalid value.` `12`

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

```
def get_middle_number_from_file(filename):
    try:
        file = open(filename, "r")
        contents = file.read()
        file.close()
        nums_list1 = contents.split()

        if len(nums_list1) == 0:
            return 'ERROR: "{}" is empty.'.format(filename)

        for i in range(len(nums_list1)-1, -1, -1):
            if nums_list1[i].isdigit() == False:
                print('ERROR: "{}" contains an invalid value.'.format(filename))
                nums_list1.remove(nums_list1[i])
            else:
                nums_list1[i] = int(nums_list1[i])

        nums_list1.sort()
        mid_index = len(nums_list1)//2
        return nums_list1[mid_index]

    except ValueError:
```

	Test	Expected	Got	
✔	`print(get_middle_number_from_file('test0.txt'))`	`ERROR: "test0.txt" does not exist.`	`ERROR: "test0.txt" does not exist.`	✔
✔	`print(get_middle_number_from_file('test1.txt'))`	`ERROR: "test1.txt" is empty.`	`ERROR: "test1.txt" is empty.`	✔
✔	`print(get_middle_number_from_file('test2.txt'))`	`12`	`12`	✔
✔	`print(get_middle_number_from_file('test3.txt'))`	`12`	`12`	✔

	Test	Expected	Got	

✔	`print(get_middle_number_from_file('test8.txt'))`	`ERROR: "test8.txt" contains an invalid value.`
		`ERROR: "test8.txt" contains an invalid value.`
		`6`
			`ERROR: "test8.txt" contains an invalid value.`
			`ERROR: "test8.txt" contains an invalid value.`
			`6`
				✔

✔	`print(get_middle_number_from_file('test9.txt'))`	`ERROR: "test9.txt" contains an invalid value.` `12`	`ERROR: "test9.txt" contains an invalid value.` `12`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.95/1.00**.

Question **6**

Correct

Mark 1.00 out of 1.00

This question uses a tuple to represent a student by their name and student id number. For example: `(5, "Smith")` represents a student with name as "Smith" and student id: 5.

Write a function `linear_search(student_id, students)` that takes an integer representing a student id as its first parameter and an **unsorted** list of tuples as its second parameter, and returns the name of the student with the **specified student id**. If the student is not present in the list, the function should return **None**.

For example:

Test	Result

`s1 = (7, "Smith")`
`s2 = (5, "Brown")`
`s3 = (4, "Chan")`
`s4 = (2, "Kim")`
`s5 = (0, "Lin")`
`students = [s1, s2, s3, s4, s5]`
`print(linear_search(0, students))`
	`Lin`

`s1 = (1, "Anderson")`
`s2 = (2, "Huang")`
`s3 = (3, "Ng")`
`s4 = (4, "Roberts")`
`s5 = (5, "Smith")`
`s7 = (7, "Zhou")`
`students = [s1, s2, s3, s4, s5, s7]`
`print(linear_search(-456, students))`
	`None`

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

```
def linear_search(student_id, students):
    student_name = ""
    for item in students:
        if item[0] == student_id:
            student_name = (item[1])
    if len(student_name) > 0:
        return student_name
    else:
        return None

```

	Test	Expected	Got	

✔	`s1 = (7, "Smith")`
	`s2 = (5, "Brown")`
	`s3 = (4, "Chan")`
	`s4 = (2, "Kim")`
	`s5 = (0, "Lin")`
	`students = [s1, s2, s3, s4, s5]`
	`print(linear_search(0, students))`
		`Lin`	`Lin`	✔

✔	`s1 = (1, "Anderson")`
	`s2 = (2, "Huang")`
	`s3 = (3, "Ng")`
	`s4 = (4, "Roberts")`
	`s5 = (5, "Smith")`
	`s7 = (7, "Zhou")`
	`students = [s1, s2, s3, s4, s5, s7]`
	`print(linear_search(-456, students))`
		`None`	`None`	✔

✔	`students = []` `print(linear_search(1, students))`	`None`	`None`	✔
✔	`students = [(1, "Smith")]` `print(linear_search(1, students))`	`Smith`	`Smith`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **7**

Correct

Mark 1.00 out of 1.00

Fix the faulty function below named `get_unique_letters(word_a, word_b)` which takes two strings as parameters. The function should return a string containing any letters that are present in only one of the words, but **not** both of the words. The letters should be ordered alphabetically. Each letter should appear in the resulting string only **once**.

A faulty solution has been provided below:

```
def get_unique_letters(word_a, word_b):
    result = ''
    for letter in word_a:
        if letter in word_b and letter not in result:
            result.append(letter)
    return ''.join(sorted(result))
```

Identify the faults and submit a corrected version of this code.

For example:

Test	Result
`print(get_unique_letters('hello', 'world'))`	`dehrw`
`print(get_unique_letters('world', 'hello'))`	`dehrw`

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

Reset answer

```
def get_unique_letters(word_a, word_b):
    result = []
    for letter in word_a:
        if letter not in word_b and letter not in result:
            result.append(letter)
    for letter in word_b:
        if letter not in word_a and letter not in result:
            result.append(letter)
    return ''.join(sorted(result))
```

	Test	Expected	Got	
✔	`print(get_unique_letters('hello', 'world'))`	`dehrw`	`dehrw`	✔
✔	`print(get_unique_letters('world', 'hello'))`	`dehrw`	`dehrw`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **8**

Correct

Mark 1.00 out of 1.00

Write a function `rotate_4(numbers)` that takes a list as a parameter and rotates the list by **4** positions and returns the new list. Given a list numbers, shift each element of the list by **4** positions to the left, if the list has has more than **4** elements. If the list does not have more than **4** elements the function should return the original list.

Note: You must not modify the original parameter list. Rotation means that numbers leaving at the **left** will be inserted at the **end of the list**.

For example:

Test	Result
`numbers = [101,67,63,71,69,19,63,11,93,40,47,57,88]` `print(rotate_4(numbers))`	`[69, 19, 63, 11, 93, 40, 47, 57, 88, 101, 67, 63, 71]`

`numbers = [1]`
`result = rotate_4(numbers)`
`print(result)`
	`[1]`

`numbers = [10,23,101,33,50,82,41,11,56,53,35,2]` `print(rotate_4(numbers))`	`[50, 82, 41, 11, 56, 53, 35, 2, 10, 23, 101, 33]`

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

```
def rotate_4(numbers):
    if len(numbers) < 5:
        return numbers
    else:
        count = 0
        while count < 4:
                numbers.append(numbers[0])
                numbers.pop(0)
                count += 1
    return numbers

```

	Test	Expected	Got	
✔	`numbers = [101,67,63,71,69,19,63,11,93,40,47,57,88]` `print(rotate_4(numbers))`	`[69, 19, 63, 11, 93, 40, 47, 57, 88, 101, 67, 63, 71]`	`[69, 19, 63, 11, 93, 40, 47, 57, 88, 101, 67, 63, 71]`	✔

✔	`numbers = [1]`
	`result = rotate_4(numbers)`
	`print(result)`
		`[1]`	`[1]`	✔

✔	`numbers = [10,23,101,33,50,82,41,11,56,53,35,2]` `print(rotate_4(numbers))`	`[50, 82, 41, 11, 56, 53, 35, 2, 10, 23, 101, 33]`	`[50, 82, 41, 11, 56, 53, 35, 2, 10, 23, 101, 33]`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **9**

Correct

Mark 1.00 out of 1.00

Define the `get_list_of_odd_maximums(a_list_of_lists)` function which is passed one parameter: a list of lists, i.e. each element is a list of integers. The function returns a list of the maximum **odd** values of each of the list elements in the parameter list. If the list does not contain any odd values then the value ***None*** should be appended to the list which is returned. You can assume that none of the elements of the parameter list are empty lists. For example, the following code:

```
a_list_of_lists = [ [4, -4],
                [-5, -4, -7, 0],
                [-82],
                [-5, 5, 3, -2] ]
result = get_list_of_odd_maximums(a_list_of_lists)
print("Odd maximums:", result)
```

prints:

```
Odd maximums: [None, -5, None, 5]
```

For example:

Test	Result

`a_list_of_lists = [ [3, 42, 678, -5, -5],`
`[-4, -2, -33, -29, 0],`
`[51],`
`[4, 6, -4],`
`[-309, -3, -34] ]`
 	
`result = get_list_of_odd_maximums(a_list_of_lists)`
`print("Odd maximums:", result)`
	`Odd maximums: [3, -29, 51, None, -3]`

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

```
def get_list_of_odd_maximums(a_list_of_lists):
    final = []
    for the_list in a_list_of_lists:
        odd_num = -9999
        for num in the_list:
            if num % 2 != 0 and num > odd_num:
                odd_num = num
        if odd_num == -9999:
            final.append(None)
        else:
            final.append(odd_num)
    return final
```

	Test	Expected	Got	

✔	`a_list_of_lists = [ [4, -4],`
	`[-5, -4, -7, 0],`
	`[-82],`
	`[-5, 5, 3, -2] ]`
	`result =`
	`get_list_of_odd_maximums(a_list_of_lists)`
	`print("Odd maximums:", result)`
		`Odd maximums: [None, -5, None, 5]`	`Odd maximums: [None, -5, None, 5]`	✔

✔	`a_list_of_lists = [ [3, 42, 678, -5, -5],`
	`[-4, -2, -33, -29, 0],`
	`[51],`
	`[4, 6, -4],`
	`[-309, -3, -34] ]`
	
	`result =`
	`get_list_of_odd_maximums(a_list_of_lists)`
	`print("Odd maximums:", result)`
		`Odd maximums: [3, -29, 51, None, -3]`	`Odd maximums: [3, -29, 51, None, -3]`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **10**

Correct

Mark 0.95 out of 1.00

Write a function called `draw_triangle(size)` that takes an integer value **size** as a parameter and draws an isosceles triangle using print statements, 'X' (capital X) and '-' characters (and spaces).

- The outline of the triangle uses 'X' characters and the interior of the triangle is filled with '-' characters.
- The base of the triangle is equal to the size, and each row above that is narrower by a single character on each side.
- If no parameter is passed, a triangle with a default **size** of 5 should be drawn.
- A triangle must be at least **size** 3. If a value less than 3 is passed then a ValueError exception should be raised.

For example:

Test	Result

`draw_triangle()`	`  X`
	`X-X`
	`XXXXX`

`draw_triangle(2)`	`ERROR: The size is too small.`

`draw_triangle(10)`	`    XX`
	`  X--X`
	`X----X`
	`X------X`
	`XXXXXXXXXX`

`draw_triangle("two")`	`ERROR: Invalid input!`

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

```
def draw_triangle(size = None):
    hyphens = 1
    try:
        if size == None:
            size = 5
            for row in range(0,(size//2)):
                print(" " * (size//2 - row), "X", ("-" * row), "X" * row, sep = "")
            print("X" * size)
        elif size >= 3 and size % 2 == 0:
            for row in range(0,(size//2)-1):
                print(" " * (size//2 - row - 1), "X", ("-" * row * 2), "X", sep = "
            print(size * "X")

        elif size >= 3 and size % 2 != 0:
            for row in range(0, (size//2)):
                if row == 0:
                    print(" " * (size//2 - row), "X", ("-" * row), "X" * row, sep =
                if row > 0:
                    print(" " * (size//2 - row), "X", ("-" * hyphens), "X", sep ="")
                    hyphens += 2
            print("X" * size)

```

	Test	Expected	Got	

✔	`draw_triangle()`	`  X`
		`X-X`
		`XXXXX`
			`  X`
			`X-X`
			`XXXXX`
				✔

✔	`draw_triangle(2)`	`ERROR: The size is too small.`	`ERROR: The size is too small.`	✔

✔	`draw_triangle(10)`	`    XX`
		`  X--X`
		`X----X`
		`X------X`
		`XXXXXXXXXX`
			`    XX`
			`  X--X`
			`X----X`
			`X------X`
			`XXXXXXXXXX`
				✔

✔	`draw_triangle("two")`	`ERROR: Invalid input!`	`ERROR: Invalid input!`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.95/1.00**.

Question **11**

Correct

Mark 1.00 out of 1.00

Write a function `swaps(numbers)` that takes an unsorted list as a parameter, and returns the number of times items in the list would need to have their positions swapped if sorting the list in ***ascending*** order using ***selection sort***.

Note: a swap is only counted if values change position. If a number is already in the correct position, it is not swapped.

For example:

Test	Result
`numbers = [0, 4, 2, 7, 5]` `print(swaps(numbers))`	2

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

```
def swaps(numbers):
    count = 0
    largest = -9999
    for index in range(len(numbers)-1, 0, -1):
        largest = max(numbers[0:index+1])
        num = 0
        while num < index:
            if numbers[num] == largest and numbers[num] != numbers[index]:
                numbers[num], numbers[index] = numbers[index], numbers[num]
                count += 1
            num += 1
    return count

```

	Test	Expected	Got	
✔	`numbers = [0, 4, 2, 7, 5]` `print(swaps(numbers))`	2	2	✔
✔	`numbers = [5, 2, 1, 8, 0, 3, 7]` `print(swaps(numbers))`	4	4	✔
✔	`numbers = [5]` `print(swaps(numbers))`	0	0	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **12**

Correct

Mark 1.00 out of 1.00

This question uses a tuple to represent a student by their name and student id number. For example: `(5, "Smith")` represents a student with name as "Smith" and student id: 5.

Write a function `check_count(student_id, students)` that takes a student id as its first parameter and an unsorted list of tuples as its second parameter. This function must search the list of tuples for a student with the given student id using **Binary Search**. The function must then return the number of locations in the list that were checked to see if they contained the student in question. The function does not need to return the student itself. If the list does not contain a student with the given student id, the function should return -1.

Hint: You may find the floor division operator (//) useful.
Hint: You may find it necessary to modify the list of students in order to perform binary search.
For example:

Test	Result

`s1 = (7, "Smith")`
`s2 = (5, "Brown")`
`s3 = (4, "Chan")`
`s4 = (2, "Kim")`
`s5 = (0, "Lin")`
`students = [s1, s2, s3, s4, s5]`
`print(check_count(0, students))`
	2

`s1 = (7, "Smith")`
`s2 = (5, "Brown")`
`s3 = (4, "Chan")`
`s4 = (2, "Kim")`
`s5 = (0, "Lin")`
`students = [s1, s2, s3, s4, s5]`
`print(check_count(7, students))`
	3

Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

```
def check_count(student_id, students):
    max_index = len(students) -1
    min_index = 0

    count = 0

    new_list = []
    for item in sorted(students):
        new_list.append(item[0])

    if student_id not in new_list:
        return -1

    while (min_index <= max_index):

        mid_index = (max_index + min_index)//2

        if new_list[mid_index] == student_id:
            count += 1
            return count
        elif new_list[mid_index] < student_id:
```

	Test	Expected	Got	

✔	`s1 = (7, "Smith")`
	`s2 = (5, "Brown")`
	`s3 = (4, "Chan")`
	`s4 = (2, "Kim")`
	`s5 = (0, "Lin")`
	`students = [s1, s2, s3, s4, s5]`
	`print(check_count(0, students))`
		2	2	✔

✔	`s1 = (7, "Smith")`
	`s2 = (5, "Brown")`
	`s3 = (4, "Chan")`
	`s4 = (2, "Kim")`
	`s5 = (0, "Lin")`
	`students = [s1, s2, s3, s4, s5]`
	`print(check_count(7, students))`
		3	3	✔

	Test	Expected	Got	

✔	`s1 = (7, "Smith")`
	`s2 = (5, "Brown")`
	`s3 = (4, "Chan")`
	`s4 = (2, "Kim")`
	`s5 = (0, "Lin")`
	`students = [s1, s2, s3, s4, s5]`
	`print(check_count(8, students))`
		`-1`	`-1`	✔

✔	`students = []` `print(check_count(1, students))`	`-1`	`-1`	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

◄ Lab06 (1%) - Sorting and Searching

Jump to...

Lab08 (1%) - Classes ►