|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Part** | **1** | **2** | **3** | **4** | **Total** |
| *maximum* | **25** points | **25** points | **25** points | **25** points | **100**G101010 pointsG |
| ***Your Score*** |  |  |  |  |  |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**An Overview of Collections**

Reading Assignment: Thoroughly read Chapter 2 in the course textbook.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 1 Glossary Terms**

Define, in detail, each of these glossary terms from the realm of computer programming logic and design and computer topics, in general. If applicable, use examples to support your definitions. Consult your notes or course textbook(s) as references or by visiting Web sites such as: [**http://www.ask.com**](http://www.ask.com),[**http://www.bing.com**](http://www.bing.com), [**http://www.webopedia.com**](http://www.webopedia.com)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(a) Bag Collection**

|  |
| --- |
| A type of unordered collection (think of a bag of marbles). Lambert does not go into detail in Chapter 2 on Bags, however a quick google shows that the Counter built-in data type operates similarly to Bags in other languages (https://docs.python.org/3/library/collections.html#collections.Counter).  After some more research, Bags appear to be useful when what the client wants to know is what is in the collection and how many of each item is in the collection. |

**(b) Graph Collection**

|  |
| --- |
| A graph collection is a non-hierarchical collection that contains data points that may have many predecessors and successors, called “neighbors” in this non-hierarchical structure. Airline routes between cities is a good example of a “graph” collection. Essentially, a graph has a web of relationships between nodes.   GraphQL, a popular query language for building and interacting with REST APIs in the world of JavaScript web development, leverages the graph data structure when building queries based on the non-hierarchical relationships within a custom schema file. |

**(c) Hierarchical Collection**

|  |
| --- |
| A hierarchical collection is a collection that makes use of parent/child (or, successor / predecessor) relationships in a hierarchy. A tree is an example of a hierarchical collection, and binary search algorithms can be used to recursively search through the tree to find nodes. |

**(d) Immutable Collection**

|  |
| --- |
| An immutable collection cannot be changed (mutated) once it has been instantiated. Tuples and strings are examples of immutable collection types. |

**(e) Unordered Collection**

|  |
| --- |
| An unordered collection has no relationships defined within it (as a graph does), nor any hierarchy (as a tree does), and is a good data structure for storing key/value pairs (as in a Dict), unique elements (as in a Set) or for making Bags for counting certain data items. |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 2 True / False Exercises**

For each of these exercises, enter True or False in the spaces provided.

**TRUE** **(1)** To determine equality of two collections, use the == operator.

**FALSE** **(2)** The pop method is used to add items to a Python list.

**FALSE (3)** If you clone an object using the = operator, as in

**myList = list(yourList)**

,

the is operator returns True while the == operator returns False.

**FALSE** **(4)** The map, filter, and reduce functions can only be used on list collections.

**TRUE** **(5)** To be considered equal, two lists must have the same length and the same items in each position, whereas the same sets must simply contain exactly the same items, but in no particular order.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 3 Multiple Choice Exercises**

Select the correct response or responses.

**(1)** What is a group of zero or more items that can be operated on as a unit?

(a) grouping **(b) collection**

(c) array (d) organization

**(2)** Which of the following is true about Python collections?

**(a) most can be heterogeneous** (b) all are homogeneous

(c) all are immutable object types (d) they are typically static

**(3)** Which object type is immutable?

(a) list (b) set

(c) dict **(d) tuple**

**(4)** What is the value of **c** after the following code executes?

**a = [10, 20, 30]**

**b = [40, 50, 60]**

**c = a + b**

(a) [50, 70, 90] (b) {[10, 20, 30], [40, 50, 60]}

**(c) [10, 20, 30, 40, 50, 60]** (d) ([10, 40], [20, 50], [30, 60])

**(5)** What is the value of **aList** after the following code is executed?

**aList = [10, 20, 30]**

**aList.pop()**

(a) [10, 20, 30] **(b) [10,20]**

(c) [] (d) [20, 30]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 4 Programming Exercises**

**(1)** **( Triangular Numbers: Repetitive Program Control with Lists )**

def triangularNumbers(*nth*):

*return* ((pow(*nth*, 2)) + *nth*)/2

triangleNums = []

*# populate the list*

*for* nth *in* range(1, 11):

triangleNums.append(triangularNumbers(nth))

*# display the list elements*

print(triangleNums)

*# returns [1.0, 3.0, 6.0, 10.0, 15.0, 21.0, 28.0, 36.0, 45.0, 55.0]*

*# 10th element is 55.0*

**(2)** **( Data Science: Looping Program Control and the Standard Deviation )**

*from* math *import* sqrt

*# sample data*

data\_list = [280, 255, 242, 207, 201]

*# determine the mean value*

sums = 0

*for* i *in* range(len(data\_list)):

sums += data\_list[i]

mean = sums / len(data\_list)

*# for each value, find the square deviation from the mean*

difference\_squared = 0

*for* i *in* range(len(data\_list)):

difference\_squared += (data\_list[i] - mean) \*\* 2

*# find the square root*

print(sqrt(difference\_squared))

*# returns 66.2872536767062*

**(3) ( Loops and Control Breaks )**

sum = 0

*for* value *in* range(1, 4):

*if* (value != 2):

sum = sum\*\*2

sum += value

print(sum)

*# returns 12*

**(4) ( Loops and Control Breaks )**

y = 0

z = 0

*for* x *in* range(3, 7):

*if* (y > z):

z, y = y, z

y = y + x

print(x, y, z)

*# returns*

*# 3 0 0*

*# 4 0 0*

*# 5 0 0*

*# 6 0 0*

**(5) ( Lists: Squaring the Odd Numbers )**

*# define a list*

oddSquares = []

*# populate the list*

*for* x *in* range(1, 11):

*if* (x % 2 == 1):

oddSquares.append(x\*\*2)

*# display the list contents*

print(oddSquares)

print(f"length of list is: {len(oddSquares)}")

*# prints*

*# [1, 9, 25, 49, 81]*

*# length of list is: 5*