**CSE 212 – Programming with Data Structures**

**W10 Prove – Response Document**

|  |  |
| --- | --- |
| **Name:** | Jeff Marsh |
| **Date:** | 6/21/21 |
| **Teacher:** | Chad Macbeth |

*It is a violation of BYU-Idaho Honor Code to post or share this document with others or to post it online. Storage into a personal and private repository (e.g. private GitHub repository, unshared Google Drive folder) is acceptable.*

**Question 1: Provide the outline for the data structures tutorial you are creating for the final project. Use the Python Fundamentals Tutorial outline provided in the assignment instructions as an example.**

**Welcome:**

My final project will be on the principles of Stacks, Sets, and Recursion. These were the topics I found most interesting in this class, so I am excited to implement these in my final project.

My contact email is mar15035@byui.edu.

**Stacks:**

Introduction:

Stacks are a type of list; they follow an order of operations. The order is Last In First Out (LIFO for short) or First In Last Out (FILO for short).

Operations:

Stacks have two main operations; push and pop. Pushing something into a stack pushes it onto the top of the stack. This is the same as the append ability for lists. Popping an item on a stack takes it off the end. So when using stacks, these two operations are absolutely necessary.

Example:

stack = []

stack.append(1)

stack.append(2)

stack.append(3)

print(‘Original Stack: ‘, stack)

print()

print(‘Popped from stack: ‘)

print(stack.pop())

print(stack.pop())

print(stack.pop())

print(“This is the stack after it’s been popped: “, stack)

**Sets:**

Introduction:

Sets are cool because they can store unique data types and values without any particular order. They cannot contain any 2 of the same thing; they can only contain unique data.

Operations:

Sets are created with the inbuilt ‘set()’ function in python, or by using curly braces to enclose the items in your set. Sets, unlike lists and stacks, cannot be sorted, appended to, and popped from. Sets cannot be changed once created. If they have duplicates, they will automatically be unincluded in the final result.

Example:

set1 = set([1, 2, 2, 3, 4, 5, 5, 5, 5, 6, 7, 8, 9, 9, 9])

print(set1)

**Recursion:**

Introduction:

Recursion is a very useful part of data structures. Recursion allows problems to be solved by a single function called multiple times. When a function calls itself to solve a problem, this is called recursion. Using recursion, complicated problems can be split into easier tasks to accomplish. It also makes the whole code look cleaner.

Operations:

The only operation here is simply calling the function again.

Example:

def factorial(x, y = 1):

if (x == 0):

return y

return factorial(x – 1, x \* y)

print(factorial(6))