Python Self Assessment

Author: Nghia T. Le

Expected Time = 30 - 40 minutes

Total Points = 20 points

Overview ¶

This self-assessment exercise was designed to provide an opportunity for you to evaluate your competency with Computer Science (CS) foundations, the Python programming language, and Jupyter notebooks prior to program enrollment. Passing this exercise will indicate your readiness for the challenging material ahead of you in the program, but will not guarantee success. Should you not pass this self-assessment, we recommend you review materials that will allow you to strenghten weaknesses in your programming skill set. A few suggested resources are listed below.

Learning Objectives

Passing this self-assessment will demonstrate your ability to:

- Understand and implement fundamental CS data structures (*e.g.* lists and trees) and algorithms (*e.g.* tree traversals and recursions) in the Python programming language
- Recognize and utilize the appropriate built-in Python functions for string, list, tuple, and dictionary data structures
- Process an input text file into the Python lists and dictionaries

Review topics

- Fundamentals CS data structures including, but not limited to, trees, lists, tuples, and dictionaries
- Fundamental CS concepts including, but not limited to, tree traveral algorithms, recursion, and efficiency
- Basic built-in and user-defined Python functions
- Reading and processing a text file in Python

Suggested Python review materials

(note: appearance here does not constitute an endorsement)

- python.org (https://docs.python.org/3/tutorial/)
- learnpython.org (https://www.learnpython.org/)
- w3schools.com (https://www.w3schools.com/python/default.asp)

Table of Content

Summary

This self-assessment is divided into two sections: the first section will test your foundation in Computer Science, while the second section will assess your ability to program using Python.

Run the following cell to import modules and declare constants required for this programming assignment

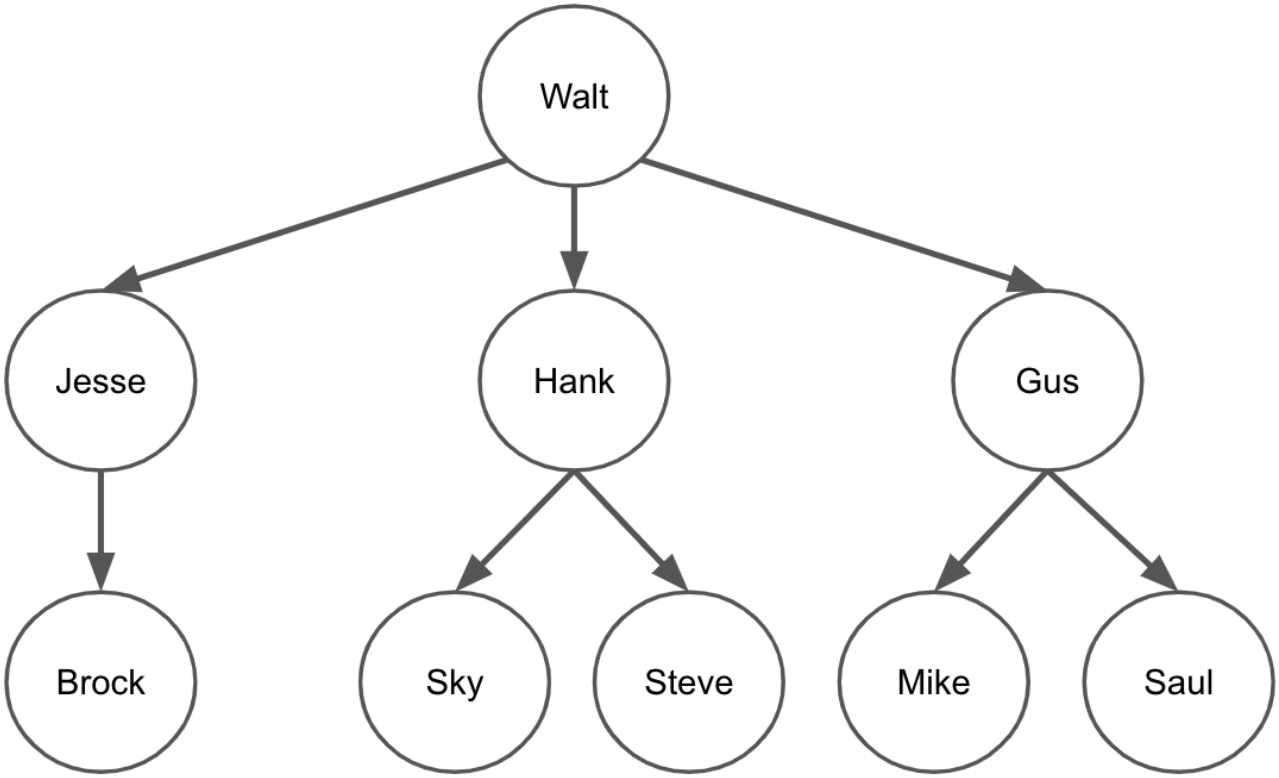
In [1]:
```
from typing import List, Tuple, Dict, Any

INPUT_TEXT_FILE = './data/people.txt'
```

Computer Science Foundations [10 pts]

This section is designed to test your familarity with fundamental Computer Science knowledge. Topics include tree data structure, tree traversal algorithms, recursion, and efficiency.

Please answer questions 1 and 2 according the tree diagram below.



There are two basic approaches to traverse a tree:

- **Depth-First Search (DFS)**: The idea is to traverse one of the subtrees (*i.e.* branches) first, before moving on to the next subtree. You can check out these resources on Wikipedia (https://en.wikipedia.org/wiki/Depth-first_search) or GeeksForGeeks (https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/) for more detailed information
- **Breath-First Search (BFS)**: In this algorithm, we traverse all the nodes *on the same level* (from left to right order) before traversing nodes on the next level (Wikipedia (https://en.wikipedia.org/wiki/Breadth-first_search), GeeksForGeeks (https://www.geeksforgeeks.org/level-order-tree-traversal/))

Throughout this assignment, for both DFS and BFS, assume that we first visit the root node, then we visit the nodes/subtrees in left to right order.

Question 1: Depth-First Search [2 pts]

List the nodes in the above tree diagram in order of a **Depth-First Search (DFS)** traversal. Assign your answer as a list of strings to the variable `ans1`, such as `ans1 = ['Hank', 'Gus', 'Mike']`

In [22]:
```
ans1 = ['Walt', 'Jesse', 'Brock', 'Hank', 'Sky', 'Steve', 'Gus', 'Mike
', 'Saul']
```

Question 2: Breadth-First Search [2 pts]

List the nodes in the above tree diagram in order of a **Breadth-First Search (DFS)** traversal. Assign your answer as a list of strings to the variable `ans2` , such as `ans2 = ['Hank', 'Gus', 'Mike']`

In [2]:
```
ans2 = ['Walt', 'Jesse', 'Hank', 'Gus', 'Brock', 'Sky', 'Steve', 'Mike
', 'Saul']
```

Question 3: Recursion with Binary Tree [3 pts]

Below we give you the template for `has_val` function, which checks if a binary tree contains a value. The inputs are the tree of type `Node` (a simple data structure for binary tree) and a value of any type. The function returns `True` if the tree contains the value, and `False` otherwise.

Please complete the `has_val` function by replacing the commented lines with your own code. Your solution should traverse the tree in a DFS, left-to-right manner. Your solution should not add extra lines.

In [3]:
```
class Node:
    """A simple data structure for Binary Tree. You can create a 2-lev
el tree with
    parent (root) node `Walt`, left child `Jesse`, right child `Hank`
as follows:
    tree = Node("Walt")
    tree.left_node = Node("Jesse")
    tree.right_node = Node("Hank")
    """
    def __init__(self, value: Any):
        self.value: Any = value
        self.left_node: Node = None
        self.right_node: Node = None


def has_val(tree: Node, value: Any) -> bool:
    if tree == None:
        return False
    elif tree.value == value:
        return True
    else:
        result = has_val(tree.left_node, value)
        if not result:
            result = has_val(tree.right_node, value)
        return result
```

To make your life easier, we provide a wrapper function `create_test_tree` below that creates a tree on which your implementation will be tested. You can use this to debug your program.

Note: you will also be tested on a hidden tree to which you don't have access.

In [4]:
```
def create_test_tree() -> Node:
    """A wrapper function to create the tree from the diagram in quest
ions 1
    and 2 above, without the right-most subtree
    """
    walt = Node('Walt')
    jesse = Node('Jesse')
    jesse.left_node = Node('Brock')
    hank = Node('Hank')
    hank.left_node = Node('Sky')
    hank.right_node = Node('Steve')
    walt.left_node = jesse
    walt.right_node = hank
    return walt

# creating the tree
testTree = create_test_tree()
# you can now use 'testTree' to check your implementation, for example
# has_val(testTree, 'Walt') should return True, while has_val(testTree
, 'Gus')
# returns False
```

Questions 4 and 5 test your knowledge about [recursion (https://www.geeksforgeeks.org/recursion-in-python/)](https://www.geeksforgeeks.org/recursion-in-python/) and efficiency.

Question 4: Efficiency - Fibonacci [2 pts]

We have the following recursive function that computes the n^{th} [Fibonacci number (https://en.wikipedia.org/wiki/Fibonacci_number)](https://en.wikipedia.org/wiki/Fibonacci_number).

In [5]:
```
def fib(n):
    if (n <= 1):
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

How many times does the function `fib` get called when `fib(7)` is called (*i.e.* calling `fib` at $n = 7$)?

As examples, number of calls to `fib` at $n = 0$ is 1, and at $n = 4$ is 9.

Assign your answer as a Python integer to variable `ans4` below.

In [6]:
```
ans4 = 41
```

Question 5: Efficiency - Memoized Fibonacci [1 pt]

Now consider a "memoized" (https://en.wikipedia.org/wiki/Memoization) version of the `fib(n)` function.

In [7]:
```
d = dict()
d[0] = 1
d[1] = 1
def fib_memoized(n):
    if n in d:
        return d[n]
    else:
        d[n] = fib_memoized(n-1) + fib_memoized(n-2)
        return d[n]
```

How many times does the function `fib_memoized` get called when `fib_memoized(7)` is called *for the first time*? As examples:

- Number of times `fib_memoized` is called when calling `fib_memoized(0)` for the first time is 1.
- Number of times `fib_memoized` is called when calling `fib_memoized(4)` for the first time is 7.

Similar to question 4, assign your answer as a Python integer to variable `ans5` below.

In [8]:
```
ans5 = 13
```

Note that the number of calls to `fib_memoized(7)` is much less than that of `fib(7)`, making `fib_memoized` much more efficient than `fib`.

Python Programming [10 pts]

This section is designed to test your familarity with Python programming language. Topics include string methods, type conversion, data structures (*e.g.* list, tuple, dictionary), and file IO.

Question 6: String Methods [2 pts]

Consider the string below assigned to `s`. Your task is to invoke the appropriate string method (https://docs.python.org/3.7/library/string.html) on `s` so that a list is returned. Every element in the returned list should be an individual element from `s`, separated by whitespace.

Please assign the returned list to `ans6`. Please also refrain from "hard-coding" the answer, such as by doing `ans6 = ['Alan_Turing', 'Computer_Scientist', 'Mathematician']`.

In [9]:
```
s = "Alan_Turing Computer_Scientist Mathematician"


ans6 = s.split(' ')
```

Question 7: Type Casting [1 pt]

Cast the list `l` defined below to a Python tuple (https://www.geeksforgeeks.org/tuples-in-python/) with variable name `ans7`. Please refrain from "hard-coding" the answer by doing `ans7 = ('Alan_Turing', 'Computer_Scientist', 'Mathematician')`

In [10]:
```
l = ['Alan_Turing', 'Computer_Scientist', 'Mathematician']

ans7 = tuple(l)
```

Questions 8, 9, 10 are designed to test your familiarity with converting a text file to Python data structures.

We provide you the text file `people.txt`, in which the filepath is stored in variable `INPUT_TEXT_FILE` (defined in the very first code block at the beginning of this notebook). The file contains several lines of text. Each line has the format: `name occupation1 occupation2 ...`, where each word is separated by spaces (see figure below).

```
Plato Philosopher
Isaac_Newton Mathematician Physicist Philosopher
Albert_Einstein Physicist
Alan_Turing Computer_Scientist Mathematician
Richard_Feynman Physicist
```

Question 8: File Input - List [3 pts]

Your task is to implement function `read_to_list` below that takes, as input, `people.txt` and converts the text into a Python 2D list of strings, where the rows denote lines in the text, and the columns denote words *without whitespaces* within each line. Note that the number of words for each line can be different (see `people.txt`). **Please also get rid of all whitespaces (e.g. newlines, spaces).**

For example, `read_to_list` converts

```
Plato Philosopher \
Isaac_Newton Mathematician Physicist
```

into

```
[['Plato', 'Philosopher'], ['Isaac_Newton', 'Mathematician', 'Physicist']]
```

In [39]:
```python
def read_to_list(filepath: str) -> List[List[str]]:
    result = []
    with open(filepath) as f:
        for line in f.readlines():
            result.append(line.strip().split(' '))
    return result
```

Question 9: File Input - Dictionary [3 pts]

Now implement function `read_to_dict` below that takes, as input, `people.txt` and converts the text into a Python dictionary, where the key denotes the occupation, and the values are a list of people with that occupation. Similar to question 8, **please also get rid of all whitespaces (*e.g.* newlines, spaces) in your answer.**

For example, `read_to_dict` converts

```
Isaac_Newton Mathematician Physicist \
Albert_Enstein Physicist
```

into

```
{ 'Mathematician': ['Isaac_Newton'], 'Physicist' : ['Isaac_Newton',
'Albert_Enstein'] }
```

In [11]:
```
def read_to_dict(filepath: str) -> Dict[str, List[str]]:
    result = dict()
    with open(filepath) as f:
        for line in f.readlines():
            line = line.strip().split(' ')
            name, occupations = line[0], line[1:]
            for occupation in occupations:
                if occupation not in result:
                    result[occupation] = []
                result[occupation].append(name)
    return result
```

Question 10: Using Dictionary [1 pt]

In the dictionary `scientists` defined below, many people are a `Physicist` (*i.e.* how many elements are in the list value of key `Physicist`)? Please refrain from hard-coding the answer and use the appropriate Python methods and syntax to answer this question. Assign your solution to variable `ans10`.

In [12]:
```
scientists = {
    'Mathematician': ['Isaac_Newton'],
    'Physicist'    : ['Isaac_Newton', 'Albert_Enstein', 'Richard_Feynman']
}

ans10 = len(scientists['Physicist'])
```