Creator: Keith Nicholas Pallo

MOOC Programming Files Overview

This document provides a high-level overview of all programming assignments and exams that I completed while enrolled in the online MOOC *Introduction to Computer Science and Programming Using Python* on the Edx platform.

Each assignment is shown below, with a simple overview and the overall task for each individual problem. Additional notes are also included in parenthesis to include things such as functions names. Provided files are also documented, along with a brief description.

I have included comments in each file, but this document serves as an additional resource.

Problem Set 1: Basic Programming Part 1

Overview: Conditional Statements and Printing

1. Count vowels in a string
2. Find frequency of substring in a string
3. Count longest alphabetical substring

Problem Set 2: Basic Programming Part 2

Overview: Conditional Statements, Control Structures

1. Find remaining balance when paying off interest
2. Find minimum monthly payment to zero balance in a year
3. Repeat problem 2 using Bisection Search to find the best payment to the cent

Problem Set 3: Intro to Tuples, Lists, & Dictionaries

Overview: Interactive Hangman Game

1. Create a function to see if word is guessed (isWordGuessed)
2. Create a function to print out completed / missing letters (getGuessedWord)
3. Create a function to print out all available letters in current round (getAvailableLetters)
4. Create a function from all previous problems that plays a game of Hangman

Problem Set 4: Good Programming Practices Intro

Overview: Two versions of "Words with Friends" are implemented, provided files shown below. In the first implementation the user plays with inputs. This includes problems 1-6 shown below. In the second, they can play as the computer. This includes problem 7 shown below.

Creator: Keith Nicholas Pallo

Provided File Name	Description
ps4a.py	Helper functions are defined for problems 1-6
test_ps4a.py	Provides testing support for problems 1-6
words.txt	Set of words that are valid during play

1. Create a function to calculate the score of a word (getWordScore)
2. Create a function to update the player's hand after a guess (updateHand)
3. Create a function to determine if a player created word is valid (isValidWord)
4. Create a function to determine current hand length (calculateHandlen)
5. Create a function to play a hand using first method (playHand)
6. Create a function to play a game (playGame)
7. Create a function to play a game using method 2 (playGame, again)

Note: There are repetitive function imports throughout, to facilitate additional testing if desired. Also, the computer's list of playable words in problem 7 has been shortened to improve runtime

Midterm Exam

Overview: Review of foundational concepts

4. Create a function to see if a number is triangular (is_trangular)
5. Create a function to print a python string with all vowels removed (print_without_vowels)
6. Create a function to return the largest element that appears an odd number of time (largest_odd_times)
7. Create a function that takes in a dictionary with immutable values and returns the inverse dictionary (dict_invert)
8. Creates a function to evaluate a general polynomial based on input variables (general_poly)
9. Create a function to check if two lists are permutations of each other (is_list_permuation)

Creator: Keith Nicholas Pallo

Problem Set 5: Object Oriented Concepts

Overview: Implementation of a Caesar Cypher, provided files shown below

Provided File Name	Description
story.txt	Encrypted Story
words.txt	Set of words that are valid during play

- Classes created along with brief descriptions
  - Message: Encrypt or to decrypt a message
  - PlaintextMessage: Subclass of Message and has methods to encode a string using a specified shift value
  - CiphertextMessage: Class to decode a string based upon the classic Caesar Cypher

Final Exam

Overview: Review of all subjects covered in MOOC, with an emphasis on OPP problems

3. Create a function that returns the sum of all integers in a string (sum_digits)
4. Create a function to recursively return the maximum integer in a tuple, list, list of tuples, etc. (max_val)
5. Create a function that returns a dictionary mapping two strings to each other assuming each contains “n” unique lowercase letter via a specified code (cipher)
6. Create a subclass, Bag, to modify and provide information about hashable objects of any type from a predefined superclass called container. Also, create a separate subclass, ASet, which can remove objects from a hashable input, and test to see if a specified user input is contained within itself. Methods demonstrate knowledge of error handling.
7. Given two classes, create a class to add or remove “tents” from the MIT campus based on specified parameters