Test Plan
Module H: Python Practical
170D Warrant Officer Basic Course
Due Date: August 11, 2022

Purpose: This test plan will specify how to run and rerun tests of the operability and functions within pyfamily.py.

Setting Up: Creating the unit test file and importing the appropriate modules to run the unit test and interact with the functions of pyfamily.py. I created a valid and invalid test person, which is an identical dictionary to those found in the family database(after the file is read in).

Test Case #1: test_format_person_information_valid(self):

Purpose: This function tests that the clean_person function properly accepts a person dictionary and returns a formatted string to meet exam output requirements.

Implementation: Passed a valid person dictionary object from TestPyFamily.

Results: If proper person dictionary is passed, the test should pass. You can input a print statement within the function to display the output.

Test Case #2: test_format_person_information_invalid(self):

Purpose: This function tests that the clean_person function does not accept person list and returns a formatted string to meet exam output requirements.

Implementation: Passed an invalid person dictionary(LIST) object from TestPyFamily.

Results: If the function clean_person is passed anything other than a dictionary, the test will pass, because an exception was raised.

Test Case #3: test_get_id_valid(self):

Purpose: This function test the get_id function's ability to accept any part of someone's name and returns that person information as a formatted string.

Implementation: Passing valid_person 'firstname'.

Results: The function will accept the name and send the dictionary through the cleaner to return a formatted string.

Test Case #4: test_get_id_invalid(self):

Purpose: This function test the get_id function's ability to accept an invalid name and checks that program.

Implementation: Passing invalid_person 'firstname'.

Results: The function will accept invalid input and return an empty list, rather that raising an error.

Test Case #5: test_get_details_valid(self):

Purpose: This function tests the get_details function’s ability to accept valid input and return a formatted string.

Implementation: Passing valid_person ‘idnumber’

Results: If ‘idnumber’ was valid, the test passes. If the ‘idnumber’ is not valid,

Test Case #6: test_get_details_invalid(self):

Purpose: This function tests the get_details function’s ability to accept invalid input and return an empty list.

Implementation: Passing invalid_person ‘idnumber’

Results: When passing an invalid ‘idnumber’ the function will return an empty list.

Test Case #7: test_get_siblings_valid(self):

Purpose: This functions tests the get_siblings function’s ability to accept valid idnumber and return a list of formatted strings for matching siblings.

Implementation: Passing a valid ‘idnumber’ of ‘30’

Results: Function will a list of formatted stings if it finds and siblings for the id number entered.

Test Case #8: test_get_siblings_invalid(self):

Purpose: This function tests the function get_siblings with an invalid input.

Implementation: Passing invalid_person ‘idnumber’

Results: Function will accept the input and return an empty list because there is no match.

Test Case #9: test_get_descendants_valid(self):

Purpose: This function tests the ability of get_descendants function to accept valid input and return an accurate list of descendants.

Implementation: Passing a valid ‘idnumber’ of ‘15’

Results: Function accepts the input and returns a list of descendants.

Test Case #10: test_get_descendants_invalid(self):

Purpose: This function tests the ability of get_descendants function to receive invalid input and return an empty list.

Implementation: Passing an invalid ‘idnumber from invalid_person.

Results: Function accepts the invalid input and returns an empty list without crashing.

Test Case #11: test_get_ancestors_valid(self):

Purpose: This function tests the ability of get_ancestors function to receive valid ‘idnumber’ and return a list of accurate ancestors

Implementation: Passing ‘idnumber’ of valid_person

Results: The test will pass when a list of ancestors are returned.

Test Case #12: test_get_ancestors_invalid(self):

Purpose: This function tests the ability of get_ancestors function to receive and handle invalid input.

Implementation: Passing the ‘idnumber’ of invalid_person.

Results: The test will return an empty list and will pass.

Test Case #13: test_get_intermarriage_valid(self):

Purpose: This function tests the ability of get_intermarriage to accept two valid last names and return a formatted list of those who are married.

Implementation: Passing the valid last names of Spamford and Hamworth.

Results: The test passes because a valid list is returned with those who are married.

Test Case #14: test_get_intermarriage_invalid(self):

Purpose: This function tests the ability of get_intermarriage function to accept two invalid last names and return an empty list without crashing.

Implementation: Passing two invalid last names of Spicer and Deberry

Results: The test will pass because the function returns an empty list because the last names are not present in the database and have no matches.

Test Case #15: test_get_all_valid(self):

Purpose: This function tests the get_all function with 0 positional arguments. The function will return a complete list of persons within the database.

Implementation: Calling the get_all function with 0 positional arguments

Results: The test passes because no exception was raised when the function was called with 0 positional arguments.

Test Case #16: test_get_all_invalid

Purpose: This function tests the get_all function with one positional argument. The get_all function take 0 positional arguments and will raise a TypeError if any arguments are passed through.

Implementation: Called get_all function with argument of ‘all’ (get_all(‘all’))

Results: The test will pass because the program raised a TypeError when the function was called with a positional argument.

Test Case #17: test_get_search_valid

Purpose: This function validates that the get_search function can parse key=value pairs with a helper function parse_search and return a list of each person associated with the search.

Implementation: searching with the following key=value pairs: hobby=ze, idnumber=30, motherid=39

Results: The test passes because all inputs are valid and no KeyError exception was raised.

Test Case #18: test_get_search_invalid

Purpose: This function validates that the get_search function will raise a KeyError when invalid information is searched for.

Implementation: searching with the following key=value pair: unknown=unknown

Results: A KeyError Exception was raised, this test passed.

Test Case #19: test_parse_search_valid

Purpose: This function test that the parse_search function accepts a key=value pair and properly parses it into a list of two strings for use in another function.

Implementation: passing the key=value pair of ‘hobby=ze’

Results: The test passes because the entry is valid and returns a list of strings.

Test Case #20: test_parse_search_invalid

Purpose: This function tests the parse_search function’s ability to accept an invalid key=value pair and raise a ValueError.

Implementation: passing the argument: ‘thats---anerror’

Results: The test passes because a ValueError was raised by the function.

Test Case #21: test_get_living_valid

Purpose: This function takes a list of formatted strings and determines if the last six letters are the word ‘LIVING’

Implementation: Passed a clean list of formatted strings with two strings. One living and one not living.

Results: The test passes because a list with only one entry, the living entry, is returned.

Test Case #22: test_get_living_invalid

Purpose: This test passes an invalid input through the get_living function to determine what error will be raised.

Implementation: Passed a list of dictionaries to the function.

Results: The function raised a TypeError and the test passes.