Activity 3 - Code Review Hands One

Code Snippets to Review

You will need to provide feedback as demonstrated in the assignment description for the following five snippets. Please use this template document to complete this assignment. You will need to provide written feedback and also "corrected" code just like in the examples in the assignment description.

Name: Vincent Yasi

email: yasiv@oregonstate.edu

Snippet 1

```
# Pull request 1
def my_func(x):


    return x**2
```

Feedback:

This function is simple and smooth, without any extra and unneeded elements, but this is also part of its downfall, too. I understand that this function's purpose is to square whatever number it is passed in, but this seems almost so simple as to not need an entire function of its own. It takes less time to just write the expression in code than it would take to write out the function name and variables to call it. However, I am not sure what the function does in your greater program, so separating it out to its own function may be needed/helpful.

Another issue is the name of the function. It is not descriptive and does not tell what the function does. A better option may be "var_square" or something similar. A function name should help to tell what it does without needing to look at its code. It also makes readability of the overall program easier.

My last major piece of feedback is there are too many blank lines in the middle of the function. You want to bring the bottom line up to just under the function declaration to conform to proper formatting and also make it easier to read.

Overall, it is a good function with just a few formatting issues. A tightened up version may look something like this:

```
def var_square(x):
    return x**2
```

Snippet 2

```
# Pull request 2
def create_odds(num):
    """Creates a list of len(num) of random odd numbers between 1 and 1000"""
    num_list = []
    for i in range(0, num):
        new_num = 2
        while new_num % 2 == 0:
            new_num = random.randint(1, 1000)
        num_list.append(new_num)
    return num_list

def create_evens(num):
    """Creates a list of len(num) of random even numbers between 1 and 1000""
"
    num_list = []
    for i in range(0, num):
        new_num = 1
        while new_num % 2 != 0:
            new_num = random.randint(1, 1000)
        num_list.append(new_num)
    return num_list
```

Feedback:

These functions are rather well written, with good formatting and logic. I do have a few suggestions to make them better. The first is to change the names of some of your variables. You use num a lot, but in slightly different contexts each time. The value passed in is only used to tell how many numbers to insert into the list, so could be changed to something like "length". "new_num" could then be changed to just "num" as it is the central variable of the function, holding the number that will eventually be added to the list. "num_list" could stay the same, as it is a list of nums, but it could also be changed to reflect more of what it is, like "odd_list" or "even_list", if you wanted. Overall, "num" is used many times in the functions, and could be changed in places to be more descriptive of what each variable is, and to better differentiate them.

Another change that could be made is with the parameters of range( ). If you are starting at 0, you can leave that number out and just put in "num". It will run from 0 to num, as 0 is the default starting value. (I do like that you remembered that range is not inclusive of the max value, and so starting at 0 still gives you the proper amount of runs without needing to modify num).

The comments at the top of create_evens( ) wraps onto the next line, and should probably be fixed. As you do not use # comments in any other place in the functions (and so do not need two different styles to differentiate them), you could switch the comment style to # and save space in both functions. The comments are also a bit long, and could be made shorter.

Lastly, a lot of the code is reused between the two functions, doing much the same thing (only changing if it is dealing with evens or odds). The two functions could be merged into one function, with an additional flag passed in to denote if it is looking for evens or odds. However, I am not sure what the functions' roles are in the greater program, so keeping them separate may be needed/more useful.

Overall, the code is good, just needs a few cosmetic tweaks. An updated version (only showing one, as they both are similar) could be:

```
def create_odds(length):
    # Create list of length of random odd ints between 1 and 1000
    even_list = []
    for i in range(length):
        num = 2
        while num % 2 == 0:
            num = random.randint(1, 1000)
        even_list.append(num)
    return even_list
```

Snippet 3

```
# Pull request 3
def check_for_val(self, val):
    """This member function checks to see if val exists in the class member
    values and returns True if found"""
    for i in range(len(self.values)):
        if self.values[i] == val:
            return True
    return False
```

Feedback:

This is overall a well made function with little to comment on. I do have a few, however.

The first is the name of the function. It is okay, but is a bit clunky. It is a sentence, and doesn't really say that the function searches values for the val. I feel it could be shortened and made more descriptive. It could be changed to something like "in_values".

The second is the use of val as the passed in value, and the list being called values. They are very similar, and one could be changed. As "values" appears to be a list of values, that can stay, but "val" could be changed to something like "num".

Third is that the for loop expression used is not needed. When searching a list, the for loop can do that automatically by just referencing the list directly with "for i in self.values". This does the action for each element of values. It also saves the value at that element into I, so you can just compare i and val directly in the loop. The way you did it would be needed in a language like C, though, but not Python.

My final point is that the comment could be formatted better. For one, # could be used instead, as that is typically cleaner. It could also be broken into a few different lines, instead of one whole sentence.

The code looks good, but an updated version if the mentioned changes may look like:

```
def in_values(self, num):
    # Member function

    # Checks to see if num exists in the class member values[]

    # Returns True if found, and False elsewise
    for i in self.values:
        if i == num:
            return True
    return False
```

Snippet 4

```
# Pull request 4
def get_val_index(arr, val):
    """Searches arr for val and returns the index if found, otherwise -1"""
    index = -1
    for i in range(len(arr)):
        if arr[i] == val:
            index = i
            break
    return index
```

Feedback:

This looks good, with good logic and mostly good code. There are a few issues, though.

The major issue is the use of break in the for loop. It is un-needed, and generally bad practice to use it if needed. Its purpose is to break out of the for loop if the value is found (and therefore its index is found), and to then return the index with the last line of code. However, this can be done cleaner by replacing the break with its own return statement, returning the index. Also, because the last line will not be used to return index both if it is, and is not, found, the final line can also be changed from returning index (which will be -1 if the value is not found), to just returning -1. This will make it more explicit what that line is returning, and that it will be run only if the value is not found in the for loop. It also removes the necessity for the "index = -1" line.

The only other advice is that the comment can be changed to # style to make it a bit shorter and look better.

Overall, this code looks good and there is not much to change. While you are cycling through each element of an array, which can be done in Python without range( ), you noticed that that will not work in this case, as you need to store what the index is in i, and not the value itself, so you need to use the for loop like it is typically used in other

languages, like C++. My comments mainly came down to the use of the break statement, and the ripple effects of changing that. An updated code may look like:

```
def get_val_index(arr, val):
    # Searches arr for val and returns the index if found, otherwise -1
    for i in range(len(arr)):
        if arr[i] == val:
            index = i
            return index
    return -1
```

Snippet 5

```
# Pull request 5
int_arr = [1, 2, 5, 2, 10, 45, 9, 100]

def print_sorted(arr):
    """Prints the items in the array after sorting"""
    arr.sort()
    for num in arr:
        print(num)
```

Feedback:

This code looks great. Very well written and concise. The name of the function describes exactly what it does. I am not sure exactly how this function will be used, but depending on how it is, I have a few recommendations.

The first is the only "error" I can seem to see, which is that while the function prints out each number in the array, in ascending order, it puts no spaces or similar characters between them, so they will ultimately print to console as a long string of numerals. Some delineator should be added to separate the numbers, such as a space or dash.

Another comment is whether or not you want to keep the sorted array. As the function is, it sorts the array and then prints its contents, but then the sorted array is lost. If you want to keep it, you can return it at the end of the function.

Running off from the last comment, it is generally a good idea to have a function return, even if it is not returning anything. It clearly marks where the end of the function is, and also helps the interpreter to know that the function has ended.

Another small one is the comments could be switched to # form to make them cleaner and shorter, and a space could be added between the sorting line and the for loop.

Lastly, this function could, depending upon how it is to be used, be divided into two functions. This function does two different tasks: sorting an array and then printing an array. These could be divided into their own subfunctions to that a single function only does one task, as opposed to many. This is not necessarily needed as this is so simple, but is a recommendation. Sorting is already like this, as it technically calls a subfunction

already, but the printing part could be made its own subfunction. If it were, it could even be useful in the future if you have an array you want to print, and don't need to sort it. As it is, you can only print an array if you also sort it. As said, depends on how you want to use the overall function.

As I said, this is well written, and I have little to actually comment on in it. The comments are mostly my personal preferences and recommendations than anything. Updated code with my minimal changes could look like:

```
def print_sorted(arr):
    # Prints the items in the array after sorting
    arr.sort()


    for num in arr:
        print(num)

    return
```

OR

```
def print_array(arr):
    # Prints the items in an array
    for num in arr:
        print(num)

    return


def print_sorted(arr):
    # Sorts array, then prints it
    arr.sort()
    print_array(arr)

    return
```