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: Harrison Latimer

email: latimerh@oregonstate.edu

Snippet 1

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


    return x**2
```

Feedback: The function seems to meet the requirement which is great, but doesn't follow the pylint or PEP8 style guide lines. I believe adding a commentate about what the function does and renaming the function to something more descriptive would improve readability of the code. Something like what I have put below.

```
def pow_2(x):
    '''Raises input by the power of 2'''
    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: Great job following style guidelines, informative comments, and clearly named functions. There are two possible approaches I might take to improve overall read ability and also take advantage of some of python lists comprehension capabilities for better performance (source https://stackoverflow.com/questions/46822789/python-generating-random-even-numbers-using-list-comprehension).

Both functions use a for loop with a nested while loop to generate either even or odd numbers. Although not such a large issue given the small range, over larger ranges this approach will start to perform poorly (maybe approaching O(n^2) territory). One way to get around nesting the while loop inside the for loop is to take advantage of random modules **randrange**

function. This function allows you to enter a range and a step parameter. A step of 2 over the range of 1,1000 will generate all odd numbers while this same step over the range of 2,1001 (you won't go over the 1000 book end of this range) will generate all even numbers (boosting performance to at worst O(n)).

You can also put all this logic into list comprehension format to have the result that is returned from this function into a new list without having to declare a list variable and append each new value.

```
def create_odds(num):
    '''Creates a list of size = n of random odd numbers between 1 and 1000'''
    return [random.randrange(1, 1000, 2) for _ in range(0, num)]


def create_evens(num):
    '''Creates a list of size = n of random even numbers between 1 and 1000'''
    return [random.randrange(2, 1001, 2) for _ in range(0, num)]
```

From here you could possibly make one function to generate either evens or odds but for the sake of clarity, leaving this functionality as two different functions makes the most sense.

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: Great job again on clearly named function and comment on what the function is supposed to do. There are just a few things I think could be done to improve what's written here.

Python has built-in way to check if a value is in a list object or not. So instead of having to iterate through the list and having more than one return statement you can refactor the code as shown below. The keyword "in" preceded by the value will return either true or false based upon whether or not the given value is "in" the list.

I also went ahead and reformatted the comment block to adhere to pylint / PEP8 standards and add what would be returned if the value was not found (for the sake of clarity).

```
def check_for_val(values, val):
    '''
    This member function checks to see if val
    exists in class member values. Returns True
    if found and false if not
    '''
    return val in values
```

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: Good job writing the clean and easy to follow code. Pro-tip; In python there is a built-in method available on lists called index. The index method will, as the name implies, return the index of the value you are looking for. You simply supply the value you are looking for to the index method. The one draw back to this built method though is that it throws an exception rather than a -1 for values not in the list. To work around this you can wrap the index method in a try-except block and return a -1 instead of the ValueError that the index method throws. Taking this approach will reduce performance overhead (since you won't have to iterate through the list and can avoid using the

dreaded break statement) and makes for a more readable snippet of code. I also went ahead and reformatted the comment block to avoid line to long issues.

```
def get_val_index(arr, val):
    '''
    Searches arr list for val and returns
    the index if found, otherwise returns
    -1
    '''
    try:
     return arr.index(val)
    except ValueError as e:
     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: Great work taking advantage of Python's built in sort method. Definitely reduces over head and keeps code clean and concise.

The only way I could think to improve this function would be to 1. Be more specific about what order the contents will be sorted in. Will it be smallest to largest? Largest to smallest? Making the more clear in the contents would help to understand the function better. 2. As far as a refactor could go, the functionality of sorting and printing of each element on a new line can be done in one line in python (see below). I believe taking this approach makes it clear to anyone reading (whether they even know what a for loop is!) that the array is being sorted and that you are printing the contents on a new line, instead of using a for loop. Resources used for refactor; in-place sort https://mail.python.org/pipermail/tutor/2015-March/104614.html print contents of list without a for loop https://www.geeksforgeeks.org/print-lists-in-python-4-different-ways/.

```
def print_sorted(arr):
    '''
    Sorts array from smallest to largest
    then prints contents to console
    '''
    print(*sorted(arr), sep="\n")
```