Kodi Durham
CSC310 Hw1
Report

CSC310_Hw1_1

```python
if __name__ == "__main__":
    userInputs = []                                 # List for user Inputs

    while True:                                     # Loop to collect user input
        try:
            userInputs.append(input("Enter: "))     # Collects user Input
        except EOFError:                            # Catches EOF and breaks loop to continue with program
            break
    for i in range(len(userInputs)):                # Loop to print out user input backwards
        print(userInputs[len(userInputs)-1-i])
```

I chose to use a list for the input to keep track of what the user put in and to keep track of when they entered it by appending it to the end of the list. I did a while true to keep it repeating for the user to keep entering and had a try except to catch the EOF Error to break the loop of input. I use a for loop to print out the out put using the variable I as the index tracking variable and for the length of the list I created in the beginning. For the print line for it to print backwards I would need to take the length of the list subtracting the index variable and 1 to not to start out of bounds of the list.

Test:

```
CSC310_Hw1_1
"C:\Users\Kodi D\PycharmProjects\CSC_Hw1\venv\Scripts\python.exe"
Enter: CSC310
Enter: HW1
Enter: Python
Enter: ^D
Python
HW1
CSC310

Process finished with exit code 0
```

Kodi Durham
CSC310 Hw1
Report

CSC310_Hw1_2

CSC310_Hw1_1.py | CSC310_Hw1_2.py | CSC310_Hw1_3.py | CSC310_Hw1_4.py

```python
def test_list():
    if len(userInputs) == 1 or len(userInputs) == 0:    # checks if there is no products
        return False
    for i in range(len(userInputs)):                    # Loops through the list for each int and checks products
        for j in range(i+1, len(userInputs)):
            if ((userInputs[i]*userInputs[j]) % 2) == 1:
                return True                             # True meaning there is an odd product
    return False                                        # False meaning there is not an odd product
```

I chose to make a separate function for ease and reusability. I need to check if the list was long enough to even produce a product if it wasn't it wouldn't have an odd product because it wasn't long enough so it would return a false for an odd product. I then chose to use to use two loops if it was long enough for products. The first on for the length of the list with I as the index tracking. The second is used to compare the other numbers besides the i index and the ones before it because it would have been already checked. I check if it is odd because it would the fastest way to kick back the main and it only takes one odd for this to be true.

```python
    return False                                        # False meaning there is not an odd product


if __name__ == "__main__":
    userInputs = []                                     # List for user Inputs

    while True:                                         # Loop to collect user input
        try:
            try:
                uInput = int(input("Enter integer: "))
                userInputs.append(uInput)
            except ValueError:                          # Catches non int values
                print("Not a integer please try again")
        except EOFError:                                # Catches EOF and breaks loop to continue with program
            break

    if test_list():                         # Determines if it has an odd product or not and tells user
        print("There is an odd product")
    else:
        print("There is not an odd product")
```

In the main of the program I decided to use a similar set up as the previous problem due to how similar the input was, but decided I need to check if the inputs were integers so it wouldn't crash the program so I used a try except for Value errors when trying to cast the input as an int type. I also had it like problem 1 to end the input collecting by EOF Error. Then I checked if had an odd and based on that had it print out if it did or not instead of true and false to help with the confusion to the user.

Tests:

```
CSC310_Hw1_2
"C:\Users\Kodi D\PycharmProjects\CSC_Hw1\venv\Scripts\
Enter integer: 1
Enter integer: 2
Enter integer: 2
Enter integer: 3
Enter integer: ^D
There is an odd product

Process finished with exit code 0
```

```
CSC310_Hw1_2
"C:\Users\Kodi D\PycharmProjects\CSC_Hw1\venv\Scripts
Enter integer: 1
Enter integer: 2
Enter integer: ^D
There is not an odd product

Process finished with exit code 0
```

CSC310_Hw_1_3

CSC310_Hw1_1.py | CSC310_Hw1_2.py | CSC310_Hw1_3.py | CSC310_Hw1_4.py

```python
def perm_list(ulist, swap, lLength):                    # this takes a list, an swapping point, and length of list
    if swap == lLength:                                 # This checks if its and the end of branch and prints output
        print(*ulist, sep="")
    else:
        for i in range(swap, lLength):                  # loop to get various permutations
            ulist[i], ulist[swap] = ulist[swap], ulist[i]   # Swapping objects to get the permutations
            perm_list(ulist, swap + 1, lLength)         # recurs changing swap index to to go further down a branch
            ulist[i], ulist[swap] = ulist[swap], ulist[i]   # Swapping back objects after completing that part of branch
```

I chose to make it a separate function because I figured I could make it recursive with less and simpler code. I thought of it as a branching problem with when the swapping index reached the end of the length of the list it would print and then go back up a stack and swap back the indexes I swapped originally. That's is why I had the for loop for starting at the swap index to the length of the list.

```python
if __name__ == "__main__":
    while True:
        try:                                            # Gets input from user and then check eofError
            uInput = input("What do you want to get permutations of? ")
            perm_list(list(uInput), 0, len(uInput))     # Calls function to print permutations
            print()
        except EOFError:                                # Lets user decide when to end program by use of end of file command
            break
```

For the main for this program I used a similar set up as the last ones but the EOF error would control when the stop the program from repeating so the user could get multiple permutations without restarting the program over.

Test:

```
CSC310_Hw1_3
"C:\Users\Kodi D\PycharmProjects\CSC_Hw1\venv\Scr
What do you want to get permutations of? 123
123
132
213
231
321
312

What do you want to get permutations of? ^D
```

CSC310_Hw1_4

CSC310_Hw1_1.py | CSC310_Hw1_2.py | CSC310_Hw1_3.py | CSC310_Hw1_4.py

```python
class HammeringD:                          # a class that takes two points converts them to binary and then checks the
                                           # differences in the binary numbers
    def __init__(self, x, y):              # constructor that takes x and y and stores them and also stores the binary
        self._x = x
        self._y = y
        self._xbin = self.to_binary(x)     # stores binary of x
        self._ybin= self.to_binary(y)      # stores binary of y

    def get_x(self):                       # the gets of the class to make class reusable and expandable
        return self._x

    def get_y(self):
        return self._y

    def get_xbin(self):
        return self._xbin

    def get_ybin(self):
        return self._ybin
```

I created a class for this one because of the reusability of the class and it could easily be expandable. I have the constructor take in an x and y integer and create four variables for them the int x, y, and the str binary of them. The binary version calls a static function in the class the converts the number to a binary str. Then I have the get functions which would return the values. I chose not to have set because the constructor really should be the only one that sets the values in the type of class.

Kodi Durham
CSC310 Hw1
Report

```python

    @staticmethod
    def to_binary(num):                     # static method that takes and int and returns the binary in str
        binary = ""
        while num > 0:
            binary = str(num % 2) + binary
            num = int(num/2)
        return binary

    def hamming_d(self):                    # the function to call to get the difference of the ints provided
        count = 0                           # a count of the differences
        list_x = list(self._xbin)           # converts the str of binary to lists
        list_y = list(self._ybin)

        big = max(len(list_x), len(list_y))     # find the number with the biggest length and stores it

        for i in range(len(list_x), big):       # makes sure both lists are the same length
            list_x.insert(0, "0")

        for i in range(len(list_y), big):
            list_y.insert(0, "0")

        for i in range(big):                    # checks the same index of both lists for difference
            if list_x[i] != list_y[i]:
                count += 1
        return count                            # return difference

```

The binary function is necessary for the class to convert the number into binary str I chose for it to be a str because I feel like it is easier to manipulate. For the hamming distance function I thought I would need to keeping a running count for how many distances to return because of the problem. I decided to change the strings to list to easily use their list index. I decided to find the max length of the binary numbers so I can add zeros to the beginning of the one that is smaller. Then I needed to go through each index and check if it id different and keeping count when it did under count. And then return the count value.

```python


if __name__ == "__main__":
    while True:
        try:                                    # collects user inputs and makes sure it is an int
            uInput_1 = int(input("Enter your x integer: "))
            break
        except ValueError:
            print("Not an integer")
    while True:
        try:
            uInput_2 = int(input("Enter your y integer: "))
            break
        except ValueError:
            print("Not an integer")

    h_d = HammeringD(uInput_1, uInput_2)        # declares object and print out the hammering distance

    print("The hammering distance between ", uInput_1, " and ", uInput_2, " is ", h_d.hamming_d())
```

In the main I used the similar code as the second program to collect the two inputs from the user have two of the while true try loops to catch non integers. I chose to do it this way so the user can still us the good first starting number if the mistakenly entered a wrong variable. Then I have to declare the new object of the class with the two numbers I got from the user to find the hammering distance then use the .hammering_d() function to give the user the differences

Test:

```
CSC310_Hw1_4
"C:\Users\Kodi D\PycharmProjects\CSC_Hw1\venv\Scripts\
Enter your x integer: 1
Enter your y integer: 4
The hammering distance between  1  and  4  is  2

Process finished with exit code 0
```