

Problem 1

Ask the user for a number. Depending on whether the number is even or odd, print out an appropriate message to the user. *Hint: how does an even / odd number react differently when divided by 2?* Extras:

1. If the number is a multiple of 4, print out a different message.
2. Ask the user for two numbers: one number to check (call it num) and one number to divide by (check). If check divides evenly into num, tell that to the user. If not, print a different appropriate message.
3. If the number is a prime number.

```
#INPUTS

print("Program that checks if two numbers are even or odd\n")
num = int(input("Enter one number (bigger than 1) to check: "))
check = int(input("Enter one number to divide by: "))
```

```
#TEST LOOPS
```

```
if num > 1:
    if (num%2) == 0:
        print(num,"is even.\n")

    else:
        print(num,"is odd\n")
    if (num%4) == 0:
        print(num,"is divisible by 4")
    else:
        print(num,"is not divisible by 4")

    if (num % check) == 0:
        print(check,"divides evenly into ", num)

    else:
        print(check,"doesn't divide evenly into ", num)

for cont in range(2,num):
    if (num % cont) == 0:
        print(num,"is not a prime number")
        break

else:
    print(num,"is a prime number")
```

Problem 1 Unit Test

```
import unittest

class NumbersTest(unittest.TestCase):

    def test_even(self):

        #Test that numbers between 0 and 5 are all even.

        for i in range(0, 6):
            with self.subTest(i=i):
                self.assertEqual(i % 2, 0)

    def test_odd(self):

        #Test that numbers between 0 and 5 are all odd.

        for i in range(0, 6):
            with self.subTest(i=i):
                self.assertEqual(i % 2, 1)

if __name__ == '__main__':
    unittest.main()
```

Problem 2. List Confusion

Take two lists, say for example these two:

a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

and write a program that returns a list that contains only the elements that are common between the lists (without duplicates). Make sure your program works on two lists of different sizes.

Extras:

1. Randomly generate two lists to test this
2. Write this in one line of Python (don't worry if you can't figure this out at this point - we'll get to it soon)

```
import random

#SET LISTS

a = [1, 1, 2, 2, 3, 5, 8, 13, 21, 34, 55, 89]
b = [1, 2, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

#COMPARE LISTS AND DISPLAY SAME ELEMENTS

c = set(a) & set(b)
print(c)

#RANDOM SUFFLE OF LISTS 'a' & 'b' AND COMPARE ELEMENTS

random.shuffle(a)
random.shuffle(b)

c = set(a) & set(b)
print("Scrambled list 'a':", a, "\nScrambled list 'b':", b, "\nMutual Elem:", c)
```

Problem 2 Unit Test

```
import unittest

class ListTest(unittest.TestCase):

    def setUp(self):

        super(ListTest, self).setUp()

        self.addTypeEqualityFunc(str, self.assertMultiLineEqual)

    def testString(self):

        a = [1, 1, 2, 2, 3, 5, 8, 13, 21, 34, 55, 89]

        b = [1, 2, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

        for count in range(max(len(a),len(b))):

            for counte in range(max(len(a),len(b))):

                with self.subTest(count=count+counte):

                    self.assertEqual(a[count], b[counte])

if __name__ == '__main__':

    unittest.main()
```

Problem 3

Ask the user for a string and print out whether this string is a palindrome or not.

```
wrd = input("Escriba una palabra:")

wrd2 = wrd[::-1]

if wrd == wrd2:
    print("Esta palabra es un Palindromo")
else:
    print("Esta palabra no es un Palindromo")
```

Problem 3 Unit Test

```
import unittest
import palindrome

class TestPalindrome(unittest.TestCase):

    def test_is_palindrome_true(self):
        value = palindrome.is_palindrome('racecar')
        self.assertEqual(value, True)

    def test_is_palindrome_false(self):
        value = palindrome.is_palindrome('dedent')
        self.assertEqual(value, False)

    def test_reverse_normal(self):
        value = palindrome.reverse('hello')
        self.assertEqual(value, 'olleh')

    def test_reverse_error(self):
        list_of_bad_value = [
            123,
            None,
        ]
        for bad_value in list_of_bad_value:
            self.assertRaises(
                TypeError,
                palindrome.reverse,
                bad_value
            )

if __name__ == '__main__':
    unittest.main()
```

Problem 4

Given two .txt files that have lists of numbers in them, find the numbers that are overlapping. One .txt file has a list of all prime numbers under 1000, and the other .txt file has a list of happy numbers up to 1000. The output should be stored in a third file, named as output.txt.

```
primeslist = []
with open('primenumbers.txt') as primesfile:
    line = primesfile.readline()
    while line:
        primeslist.append(int(line))
        line = primesfile.readline()

happieslist = []
with open('happynumbers.txt') as happiesfile:
    line = happiesfile.readline()
    while line:
        happieslist.append(int(line))
        line = happiesfile.readline()

overlaplist = []
for elem in primeslist:
    if elem in happieslist:
        overlaplist.append(elem)

print(overlaplist)
```

Problem 4 Unit Test

```
import unittest
import overlap

class Testoverlap(unittest.TestCase):

    """
    Return a list containing the elements which are in both primernumbers and
    happynumbers

    >>> overlap([2,3,5,7,11,13,17,19,23,29,31,37,41,43,47],
    [1,7,10,13,19,23,28,31,32,44,49,68,70,79,82])
    [7,13,19,23,31]
```

```
def test_overlap_numbers(primenumbers, happynumbers):  
    result = []  
    for element in primenumbers:  
        if element in happynumbers:  
            result.append(element)  
    return result  
  
if __name__ == '__main__':  
    unittest.main()
```

Problem 5

```
def reverse(userInput):
    userInputArray = userInput.split(" ")
    wordResult = ""
    for i in range(1, len(userInputArray)+1):
        wordResult += userInputArray[len(userInputArray)-i]
        if i < len(userInputArray):
            wordResult += " "
    return wordResult

if __name__ == "__main__":
    userInput = input("Ingrese una oracion: ")
    print(reverse(userInput))
```

Problem 5 Unit Test

```
import unittest
import reverseword

class TestReverse(unittest.TestCase):

    def test_reverse_true(self):
        result = reverseword.reverse("una oracion")
        self.assertEqual(result, "oracion una")

    def test_reverse_false(self):
        result = reverseword.reverse("una oracion")
        self.assertNotEqual(result, "una oracion")

if __name__ == '__main__':
    unittest.main()
```


Problem 6

```
import random
import string

numberAnswer = ''.join([random.choice(string.digits) for n in range(4)])

def evaluateWord(word):
    cowsBulls = [0, 0] #Cows posicion 0, Bulls posicion 1
    for n in range(4):
        if numberAnswer[n] == word[n]:
            cowsBulls[0] += 1
    for i in word:
        if i in numberAnswer:
            cowsBulls[1] += 1
    return cowsBulls

def defineAnswerManually(answer):
    global numberAnswer
    numberAnswer = answer

def play():
    cowsBulls = [0, 0] #Cows posicion 0, Bulls posicion 1
    print("Pista: ", numberAnswer)
    while cowsBulls[0] < 4:
        numberInput = [n for n in input("Ingresa un número de 4 dígitos: ")]
        numberInputClean = numberInput[:4]
        cowsBulls = evaluateWord(numberInputClean)
        print("Cows: ",cowsBulls[0]," Bulls: ",cowsBulls[1]-cowsBulls[0])
        if cowsBulls[0] >= 4:
            print("Número adivinado! Respuesta: ", numberAnswer)

if __name__ == "__main__":
    #defineAnswerManually("1234")
    play()
```

Problem 6 Unit Test

```
import unittest
import cowsbills

class Testcowsbills(unittest.TestCase):

    def test_cowsbills_true(self):
        cowsbills.defineAnswerManually("1234")
        result = cowsbills.evaluateWord("1234")
        self.assertEqual(result, [4,4])

if __name__ == '__main__':
    unittest.main()
```