Objectives

* Developing recursive function.
* Implementing a memoize function.

Due Date

This assignment is due on Sunday, February 25, 2024, by 6:00 pm.

**Remarks:**

* When you are asked to hand in code, you cut-and-paste your code as text and paste into this Word document immediately following the prompt for your code.
* Include with your code several test case examples and your results.
* This lab is cooperative - talk with your partner as you go through, and make sure you are progressing together.

Value

This assignment is worth 10 points.

Activities

1. Create a test file which will be added to for each of the functions being built as described below. Call your test file test\_happy\_prime.py.
2. Define a recursive function named num2list(n) which converts any integer to a list of single-digit integers. For instance, the expression num2list(12345) would return   
   [1, 2, 3, 4, 5]. Provide a listing of your code below.

def num2list(self, n):  
 *"""Recursively Converts any integer n into a list of single-digit integers"""* if n == 0:  
 return []  
 else:  
 return self.num2list(n // 10) + [n % 10]

1. Define a non-recursive function named num2list2(n) which converts any integer to a list of single-digit integers. Provide a listing of your code below.

def num2list2(self, n):  
 *"""Non-Recursively converts any integer n into   
 a list of single-digit integers"""* digits = []  
 while n > 0:  
 digits.insert(0, n % 10) # Insert digits at the beginning for correct order  
 n //= 10  
 return digits

1. Enjoy this clip of the BBC television show Doctor Who: http://www.youtube.com/watch?v=ee2If8jSxUo.   
     
   Write a recursive function (and any helper functions necessary) to find the next happy prime in the sequence ..., 313, 331, 367, 379  
     
   “Any number that reduces to one when you take the sum of the square of its digits and continue iterating until it yields 1 is a happy number. Any number that doesn’t, isn’t.”
2. We already have a function from question 1 that explodes an integer into a list of digits.
3. Define a function sum-of-squares(L) that takes a list of integers and returns the sum of the squares of the integers in the list. Provide a listing of your code below.

def sum\_of\_squares(self, L):  
 *"""Calculates the sum of squares for use in ishappy"""* total = 0  
 for num in L:  
 total += num \*\* 2  
 return total

1. of the digits of a number and it produces a number we’ve already computed, we will always loop through the numbers we’ve already tried. We’ll need to keep track of the numbers we’ve tried so that we can stop if we come to a number we’ve already seen before. Maintain the values tried in a list. Check to see if the number already exists in the list and, if it does, then you have come upon a number that is not happy (a form of memoize).
2. Use all of these functions to define a function named ishappy(n) which, given a positive integer, returns True if the number is a happy number and False otherwise. Provide a listing of your code below.

def ishappy(self, n, seen=None):  
 *"""Outputs True of number is Happy (1) or False if number is Not Happy"""* if seen is None: # Create a set if not already created  
 seen = set()  
  
 if n == 1:  
 return True  
 if n in seen:  
 return False  
  
 seen.add(n) # Ok, next  
 next\_sum = self.sum\_of\_squares(self.num2list(n)) # Feed in from recursive list  
 return self.ishappy(next\_sum, seen)

1. Use your new ishappy(n) function and a function isprime(n) that tests for primality (see below) to define a function named happy\_prime(n) which, given a positive integer as a parameter, returns True if the number is a prime number that is also a happy number and False otherwise.

def isprime(n):  
 def does\_divide(a, b):  
 # Returns True if b is divisible by a  
 return b%a == 0  
 def smooth(k):  
 return (k >= 2) and \  
 ((does\_divide(k, n)) or (smooth(k-1)))  
 return not(smooth(int(n \*\* 0.5)))

1. Provide a description below this bullet of how this function is determining if the number n is prime or not. Put your description if courier font.

isprime() is the main function. This calls smooth(), and looks for a False. If the output of smooth() is false, then the output of isprime() will be true, the number is a prime number.

does\_divide() is a helper function to check if a divides into b, returns true if it does.

smooth() is a recursive function that checks if any number between 2 and the square root of the iterator \*n\* divides n. It calls does\_divide() to complete that operation before the comparing operation. If this returns true, it is not prime.

1. Provide a listing of your happy\_prime(n) code below.

def happy\_prime(self, n):  
 *"""Returns True if the number is both a prime and a happy number, otherwise False"""* return self.isprime(n) and self.ishappy(n)

1. Finally, write a recursive function next\_happy\_prime(n, n\_attempts) that accepts a positive integer and calculates the next highest happy-prime number. Once n\_attempts of prime numbers is achieved, then return False. Provide a listing of your code below.

def next\_happy\_prime(self, n, n\_attempts=0):  
 *"""Finds the next highest happy-prime number after 'n'. If 'n\_attempts' of prime numbers is reached, returns False."""* n += 1 # Start checking from the next number after 'n'  
 while n\_attempts > 0:  
 if self.isprime(n) and self.ishappy(n):  
 return n  
 n += 1  
 n\_attempts -= 1  
 return False

1. Provide a listing of your testing file test\_happy\_prime.py below.

import unittest  
from lab5 import HappyNumber  
class TestHappyNumber(unittest.TestCase):  
 *"""Tests for HappyNumber methods."""* def setUp(self):  
 *"""Setup method to create an instance of HappyNumber."""* self.happy\_number = HappyNumber(1)  
  
 def test\_num2list(self):  
 *"""Tests recursive conversion of integer to list of digits.  
 Input: Integer, Output: List of integers."""* result = self.happy\_number.num2list(123)  
 self.assertEqual(result, [1, 2, 3])  
  
 def test\_num2list2(self):  
 *"""Tests non-recursive conversion of integer to list of digits.  
 Input: Integer, Output: List of integers."""* result = self.happy\_number.num2list2(123)  
 self.assertEqual(result, [1, 2, 3])  
  
 def test\_sum\_of\_squares(self):  
 *"""Tests sum of squares of list elements.  
 Input: List of integers, Output: Integer."""* result = self.happy\_number.sum\_of\_squares([1, 2, 3])  
 self.assertEqual(result, 14)  
  
 def test\_ishappy(self):  
 *"""Tests if a number is happy.  
 Input: Integer, Output: Boolean."""* result = self.happy\_number.ishappy(19)  
 self.assertTrue(result)  
  
 def test\_isprime(self):  
 *"""Tests if a number is prime.  
 Input: Integer, Output: Boolean."""* result = self.happy\_number.isprime(5)  
 self.assertTrue(result)  
  
 def test\_happy\_prime(self):  
 *"""Tests if a number is both happy and prime.  
 Input: Integer, Output: Boolean."""* result = self.happy\_number.happy\_prime(7)  
 self.assertTrue(result)  
 def test\_next\_happy\_prime(self):  
 *"""Tests finding the next happy prime after a number within a set number of attempts.  
 Input: Integer (n), Integer (n\_attempts), Output: Integer or False."""* # Test case 1: Find the next happy prime after 7 with sufficient attempts  
 expected\_next\_happy\_prime = 13  
 result = self.happy\_number.next\_happy\_prime(7, 10)  
 self.assertEqual(result, expected\_next\_happy\_prime, "The next happy prime after 7 should be 13.")  
  
 # Test case 2: Test for a scenario where n\_attempts limit is reached without finding a next happy prime  
 result = self.happy\_number.next\_happy\_prime(7, 1) # Only 1 attempt, not enough to find the next happy prime  
 self.assertFalse(result, "Should return False if the n\_attempts limit is reached without finding a next happy prime.")  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()