# **CIS 61 :: Lab 04 - Recursion**

#### **Student Name: Jonathan Adrianto Saleh**

Instructions: Download the template file to complete this lab. Attached screenshots of your code and your test run. Make sure the screenshots are readable.

### Q1: Skip Add

### Write a function skip\_add that takes a single argument n and computes the sum of every other integer between 0 and n. Assume n is non-negative.

|  |
| --- |
| **def** **skip\_add**(n):  """ Takes a number x and returns x + x-2 + x-4 + x-6 + ... + 0.  >>> skip\_add(5) # 5 + 3 + 1 + 0  9  >>> skip\_add(10) # 10 + 8 + 6 + 4 + 2 + 0  30  >>> # Do not use while/for loops! |
| Code: def skip\_add(n):  """ Takes a number x and returns x + x-2 + x-4 + x-6 + ... + 0.  >>> skip\_add(5) # 5 + 3 + 1 + 0  9  >>> skip\_add(10) # 10 + 8 + 6 + 4 + 2 + 0  30  >>> # Do not use while/for loops!  """  if n == 0:  return n  elif n == 1:  return n  else:  return n + skip\_add(n-2) |
| Test Run: |

### Q2: Hailstone

Recall the hailstone function from Lab 2. First, pick a positive integer n as the start. If n is even, divide it by 2. If n is odd, multiply it by 3 and add 1. Repeat this process until n is 1. Write a recursive version of hailstone that prints out the values of the sequence and returns the number of steps.

*Hint:* When taking the recursive leap of faith, consider both the return value and side effect of this function.

|  |
| --- |
| **def** **hailstone**(n):  """Print out the hailstone sequence starting at n, and return the number of elements in the sequence.  >>> a = hailstone(10)  10  5  16  8  4  2  1  >>> a  7 |
| def hailstone(n):  """Print out the hailstone sequence starting at n, and return the  number of elements in the sequence.  >>> a = hailstone(10)  10  5  16  8  4  2  1  >>> a  7  """    if n == 1:  return 1  else:  if n %2 == 0:  print(int(n))  return hailstone(n/2)    elif n %2 != 0:  print(int(n))  return hailstone(n\*3 + 1) |
|  |

### Q3: Summation

### Write a recursive implementation of summation, which takes a positive integer n and a function term. It applies term to every number from 1 to n including n and returns the sum of the results.

|  |
| --- |
| **def summation(n, term):**  """Return the sum of the first n terms in the sequence defined by term.  Implement using recursion!  >>> summation(5, lambda x: x \* x \* x) # 1^3 + 2^3 + 3^3 + 4^3 + 5^3  225  >>> summation(9, lambda x: x + 1) # 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10  54  >>> summation(5, lambda x: 2\*\*x) # 2^1 + 2^2 + 2^3 + 2^4 + 2^5  62  >>> # Do not use while/for loops!  """ |
| def summation(n, term):  """Return the sum of the first n terms in the sequence defined by term.  Implement using recursion!  >>> summation(5, lambda x: x \* x \* x) # 1^3 + 2^3 + 3^3 + 4^3 + 5^3  225  >>> summation(9, lambda x: x + 1) # 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10  54  >>> summation(5, lambda x: 2\*\*x) # 2^1 + 2^2 + 2^3 + 2^4 + 2^5  62  >>> # Do not use while/for loops!  """  if n == 0:  return 0  elif n == 1:  return term(n)  else:  return term(n) + summation(n-1,term) |
|  |

### Q4: Is Prime

Write a function is\_prime that takes a single argument n and returns True if n is a prime number and Falseotherwise. Assume n > 1. We implemented this iteratively before, now time to do it recursively!

*Hint*: You will need a helper function! Remember helper functions are useful if you need to keep track of more variables than the given parameters, or if you need to change the value of the input

|  |
| --- |
| **def** **is\_prime**(n):  """Returns True if n is a prime number and False otherwise.  >>> is\_prime(2)  True  >>> is\_prime(16)  False  >>> is\_prime(521)  True  """ |
| def is\_prime(n):  """Returns True if n is a prime number and False otherwise.  >>> is\_prime(2)  True  >>> is\_prime(16)  False  >>> is\_prime(521)  True  """  if n <= 1:  return False  else:  def help(i):  if n <= i :  return True  elif n % i == 0:  return False  else :  return help(i+1)  return help(2) |
|  |

### 

### Q5: GCD

The greatest common divisor of two positive integers a and b is the largest integer which evenly divides both numbers (with no remainder). Euclid, a Greek mathematician in 300 B.C., realized that the greatest common divisor of a and b is one of the following:

* the smaller value if it evenly divides the larger value, or
* the greatest common divisor of the smaller value and the remainder of the larger value divided by the smaller value

In other words, if a is greater than b and a is not divisible by b, then

gcd(a, b) = gcd(b, a % b)

Write the gcd function recursively using Euclid's algorithm.

|  |
| --- |
| **def** **gcd**(a, b):  """Returns the greatest common divisor of a and b.  Should be implemented using recursion.  >>> gcd(34, 19)  1  >>> gcd(39, 91)  13  >>> gcd(20, 30)  10  >>> gcd(40, 40)  40  """ |
| def gcd(a, b):  """Returns the greatest common divisor of a and b.  Should be implemented using recursion.  >>> gcd(34, 19)  1  >>> gcd(39, 91)  13  >>> gcd(20, 30)  10  >>> gcd(40, 40)  40  """  if b == 0:  return a  else:  return gcd(b, a%b) |
|  |

Q6: Count Stairs

You want to go up a flight of stairs that has n steps. You can either take 1 or 2 steps each time. How many different ways can you go up this flight of stairs? Write a function count\_stair\_ways that solves this problem. Assume n is positive.

Before we start, what’s the base case for this question? What is the simplest input?

What do count\_stair\_ways(n - 1) and count\_stair\_ways(n - 2) represent?

Use those two recursive calls to write the recursive case:

|  |
| --- |
| def count\_stair\_ways(n):  """  >>> count\_stair\_ways(1)  1  >>> count\_stair\_ways(2)  2  >>> count\_stair\_ways(3)  3  >>> count\_stair\_ways(4)  5  >>> count\_stair\_ways(5)  8  """ |
| def count\_stair\_ways(n):    def repeat(n):  if n <=3 and n>0 :  return n  else:  return repeat(n-1)+repeat(n-2)  return repeat(n) |
|  |

Q7: Count Stairs with k steps(

Consider a special version of the count\_stairways problem, where instead of taking 1 or 2 steps, we are able to take up to and including k steps at a time.

Write a function count\_k that figures out the number of paths for this scenario. Assume n and k are positive.

|  |
| --- |
| def count\_k(n, k):  """ >>> count\_k(3, 3) # 3, 2 + 1, 1 + 2, 1 + 1 + 1  4  >>> count\_k(4, 4)  8  >>> count\_k(10, 3)  274  >>> count\_k(300, 1) # Only one step at a time  1  """ |
| def count\_k(n, k):  """ >>> count\_k(3, 3) # 3, 2 + 1, 1 + 2, 1 + 1 + 1  4  >>> count\_k(4, 4)  8  >>> count\_k(10, 3)  274  >>> count\_k(300, 1) # Only one step at a time  1  """  if n ==0:  return 1  elif n<0:  return 0  else:  i = 1  total = 0  while i <= k:  total = total + count\_k(n-i,k)  i = i + 1  return total |
|  |

### (Optional) Q8: Ten-pairs

Write a function that takes a positive integer n and returns the number of ten-pairs it contains. A ten-pair is a pair of digits within n that sums to 10. *Do not use any assignment statements.*

The number 7,823,952 has 3 ten-pairs. The first and fourth digits sum to 7+3=10, the second and third digits sum to 8+2=10, and the second and last digit sum to 8+2=10. Note that a digit can be part of more than one ten-pair.

*Hint*: Use a helper function to calculate how many times a digit appears in n.

|  |
| --- |
| **def** **ten\_pairs**(n):  """Return the number of ten-pairs within positive integer n.  >>> ten\_pairs(7823952)  3  >>> ten\_pairs(55055)  6  >>> ten\_pairs(9641469)  6  """ |
| def ten\_pairs(n):  """Return the number of ten-pairs within positive integer n.  >>> ten\_pairs(7823952)  3  >>> ten\_pairs(55055)  6  >>> ten\_pairs(9641469)  6  """  if n < 10:  return 0  else:  return ten\_pairs(n//10) + count\_upto(n//10, 10- n%10)    def count\_upto(i,number):  if i == 0:  return 0  elif i % 10 == number:  return count\_upto(i //10, number) + 1  else:  return count\_upto(i//10, number) |
|  |