CS 163 Recursion Lab

Fall 2019

October 10, 2019

1 Introduction

Recursion is a method of performing the same operation successively. It is occasionally used by programmers.

1.1 Loops

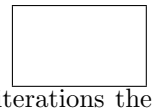
Usually we use loops for repetitive tasks. For example, if we are trying to calculate the factorial of a number, we can use a loop:

```
num = 5               # Number to compute factorial of
fact = 1              # Running product of the factorial

while num > 1:        # Loop until we get down to 1...
    fact = fact*num   # Multiply running total by number
    num = num - 1     # Decrement num

print("factorial = " + str(fact))
```

This program uses a loop to compute the factorial of `num`. Notice that the instructions inside the loop are indented. Python uses indentations to tell what lines are part of a loop. Type this program in to your python interpreter and run it.

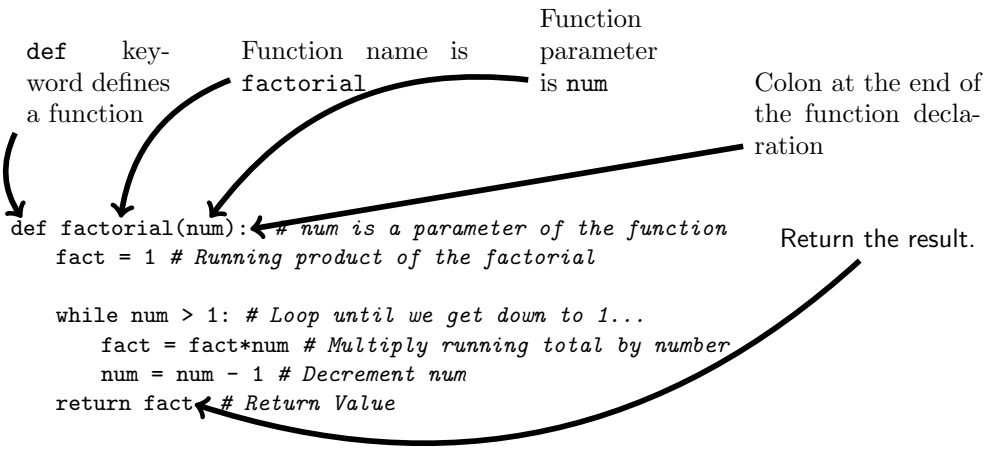
What result does it compute for the factorial of 5? 

Now modify the program to track how many iterations the loop runs. To do this, you'll need to add a new variable (right after the line `fact = 1`) and initialize it to zero. Increment the variable on each iteration of the loop and print its value at the end of the program.

How many iterations does this program run? 

1.2 Functions

A function is a piece of code that does a specific job. For example, if we have a lot of places in our code where we use the factorial operation, we might want to encapsulate it in a function so we don't have to keep copy-pasting that while loop every time we want to compute the factorial of a number. Here's how we would encapsulate the factorial operation into a function:




```
def factorial(num): # num is a parameter of the function
    fact = 1 # Running product of the factorial

    while num > 1: # Loop until we get down to 1...
        fact = fact*num # Multiply running total by number
        num = num - 1 # Decrement num
    return fact # Return Value

result = factorial(5) # Call our function to compute the factorial of 5
print("factorial␣=␣" + str(result))
```

In the first line, we declare the function with the `def` keyword. The function's name (`factorial`) goes after `def`, followed by its input, which is surrounded in parentheses. We also need a colon after the list of inputs. In python, we have to indent each line inside the function, just like lines in a loop.

When the function finishes, we are going to return a value to the caller. In the example above, `factorial()` returns the variable `fact`, which is assigned to `result`.

Calling Functions On the second to the last line of the above program, we call the function `factorial`, passing it the parameter 5. The function will compute 5! and store the result in the variable `result`, which we print to the terminal. Type this program in and call it with a few different parameters to make sure it works.

Writing your Own Function Now write another function that uses factorial to compute $\binom{n}{k}$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

I'll help you get started:

```
def nchoosek(n,k):
    nfact = factorial(n)
    kfact = factorial(k)

    # Insert other lines here to compute n choose k...
    return answer
```

This function calls the `factorial` function, so we need to have both of them in the same python file in order for it to work. Demo your function to Neil once you get it working.

2 Recursive Functions

A recrusive function is a function that calls itself. They are useful for algorithms that perform the same operation on some data over and over again until completion. Factorial is perfect for recursion—it performs multiplication over and over again on sequentially smaller integers until the integer is 1. Here's how we would implement factorial as a recursive function:

```
def factorial(n):
    if n > 1:
        return n*factorial(n-1)
    else:
        return 1
```

This function calls itself over and over again. Each time it calls itself it passes the next smaller integer until the parameter `n` is 1. This is the terminating condition.

Type this function in and run it with a few different parameters to make sure it works.

2.1 The Fibonacci Sequence

The Fibonacci Sequence is a series of integers in which the next number is the sum of the previous two:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...$$

Write a recursive function in Python that prints out the first k elements of the Fibonacci sequence. Demo to Neil when done.