UCR Extension
Professional Studies

# Assignment #4: Recursion
## CS X458.03: Programming and Design I

### Section I – Difficulty level: Easy

1. Write an iterative function called `iterPower(base, exp)` that calculates the exponential $base^{exp}$ by simply using successive multiplication. For example, `iterPower(base, exp)` should compute $base^{exp}$ by multiplying base times itself, `exp` number of times.

   The function should take two values, `base` can be a float or integer, `exp` will be an integer that is >= 0. The function should return one numerical value. Your code must be iterative, so do not use the ** operator.

   Remember to include a docstring. (**5 marks**)

2. In the first question (1) above, we computed an exponential by iteratively executing successive multiplications. We can use the same idea, but in a recursive function.

   Write a function, `recurPower(base, exp)` which computes $base^{exp}$ by recursively calling itself to solve a smaller version of the same problem, then multiplying the result by `base` to solve the initial problem.

   The function should take two values, `base` can be a float or integer, `exp` will be an integer that is >= 0. The function should return one numerical value. Your code must be iterative, so do not use the ** operator.

   Remember to include a docstring. (**5 marks**)

### Section II – Difficulty level: Moderate

1. The greatest common divisor of two positive integers is the largest integer that divides each of them without a remainder. For example:
   - `gcd(2, 12) = 2`
   - `gcd(6, 12) = 6`
   - `gcd(9, 12) = 3`
   - `gcd(17, 12) = 1`

   Write an iterative function, `gcdIter(a, b)`, that implements this idea. One easy way to do this is to begin with a test value equal to the smaller of the two input arguments, and iteratively reduce this test value by 1 until you either reach a case where the test divides both `a` and `b` without a remainder (use modulo division, %), or until you reach 1, whichever comes first.
   (**10 marks**)

Test cases:

- `gcdIter(238, 182) = 14`
- `gcdIter(14, 12) = 2`
- `gcdIter(8, 92) = 4`
- `gcdIter(72, 216) = 72`
- `gcdIter(2, 26) = 2`
- `gcdIter(57, 30) = 3`
- `gcdIter(72, 72) = 72`
- `gcdIter(160, 30) = 10`
- `gcdIter(45, 50) = 5`
- `gcdIter(108, 36) = 36`

2. A clever mathematical trick (due to Euclid), makes it easy to find greatest common divisors. Suppose that `a` and `b` are two positive integers:
   - `gcd(a, 0) = a`
     - That is, if `b = 0`, then the answer is `a`
   - Otherwise, `gcd(a, b)` is the same as `gcd(b, a%b)`

   For example, to compute `gcd(48,18)`, divide 48 by 18 to get a quotient of 2 and a remainder of 12. Then divide 18 by 12 to get a quotient of 1 and a remainder of 6. Then divide 12 by 6 to get a remainder of 0, which means that 6 is the gcd. Note that we ignored the quotient in each step except to notice when the remainder reached 0, signaling that we had arrived at the answer.

   Write a function, `gcdRecur(a, b)`, that implements this idea recursively. This function will take two positive integers and returns one integer.

   (**10 marks**)

   Test cases:

   - `gcdRecur(64, 28) = 4`
   - `gcdRecur(176, 99) = 11`
   - `gcdRecur(11, 176) = 11`
   - `gcdRecur(75, 54) = 3`
   - `gcdRecur(90, 288) = 18`
   - `gcdRecur(120, 204) = 12`
   - `gcdRecur(228, 48) = 12`
   - `gcdRecur(20, 50) = 10`
   - `gcdRecur(91, 260) = 13`
   - `gcdRecur(7, 84) = 7`

## Section III – Difficulty level: Moderate to Hard

1. Trace the following code for the function `foo1(3, 12)`. Show all your work and calculations. Use the example in slide 14 as a guide. **(10 marks)**

```
def foo1(x, y):
  '''
  Example recursive function
  '''
  if (x < y):
    return foo1(x+1, y-2)
  elif (x == y):
    return 2*foo1(x+2, y-3) - 3
  else:
    return 2*x + 3*y

print(foo1(3, 12))
```

2. Back to our Palindrome algorithm!

   In the pseudocode below, you will find a recursive algorithm that determines if a phrase is a palindrome or not. Convert the pseudocode into a Python function. Make sure the function is case insensitive to the phrase you are checking.

   Notice that this is a case where we have used inductive reasoning to arrive at a recursive solution. **(10 marks)**

```
begin isPalindrome(PHRASE)
  LENGTH = Get the length of PHRASE
  if LENGTH <= 1 then
    return TRUE
  else
    # Get only middle parts of the phrase
    MID_PHRASE = PHRASE[]  # Use Python's sequence slicing here

    # Call the recursive function
    # The phrase is a palindrome if the
    # 1st and last characters match *and*
    # if the middle characters make a palindrome
    # Remember to make the function case insensitive here
    if ((PHRASE[0] = PHRASE[LENGTH -1]) AND
         isPalindrome(MID_PHRASE) = TRUE) then
      return TRUE
    else
      return FALSE
    end if
  end if
end function
```

## Test Cases:

“Racecar”, “Anna”, “Madam”, “Kayak”

## Coding Challenge:

From our first course, do you remember how we stripped out all punctuation and symbols? Write a function that strips out all punctuation from the phrase. Put that function in a separate Python .py file and import it as a module into your palindrome program.

Make a call to the function and strip out any of the unnecessary characters, then make a call to `isPalindrome()`.

## Test Cases if you do the Coding Challenge:

“Step on no pets”, “Red rum, sir, is murder”, “No lemon, no melon”
“A nut for a jar of tuna.”
“Al lets Della call Ed 'Stella'”
“Able was I, ere I saw Elba.”
“Are we not drawn onward, we few, drawn onward to new era?”
“Are we not pure? 'No, sir!' Panama's moody Noriega brags. 'It is garbage!' Irony dooms a man - a prisoner up to new era.”