# Computer Science I — CSCI-141
# GCD Two Ways — Homework 4

09/05/2020

## 1 Task 1

Write and test a *tail-recursive* function that determines the greatest common divisor (GCD) of two numbers. Place the code in a file named `gcd.py`.

Sometimes called the greatest common denominator, the gcd function has this mathematical definition:

$$gcd(a,b) = \begin{cases} a, & \text{if } b = 0. \\ gcd(b, a \ mod \ b), & \text{otherwise.} \end{cases} \tag{1}$$

### 1.1 Requirements

- Name your function `gcd_rec`. Your function must have two parameters, both of which are integers, and you may not assume that these values are in any particular order.
  You may assume that the argument values provided to your function are integer values; it is a pre-condition that the arguments are integers, and you do not need to do any error-checking to confirm that they are integer values.

- Write a test function called `test_gcd_rec` that tests your `gcd_rec` function by calling it several times with different argument values. You should have at least six (and perhaps more) tests. Choose arguments that test different aspects of your function. For example, you may have one test `gcd_rec(5,5)`, that checks that the function works properly when the arguments are both positive and the same value. Once you have this test, you do not need another test `gcd_rec(8,8)`, as this checks the same thing.
  You should not use any tests involving arguments that are not integers.

### Example Function Operation

Output would look like this if you were to call `gcd_rec` from the interactive console:

```
>>> from gcd import gcd_rec
>>> gcd_rec(6,4)
2
```

## 2 Task 2

Write and test an *iterative* function that determines the greatest common divisor (GCD) of two numbers. Add the code to the `gcd.py` file.

Note that the recursive algorithm is already in "accumulator" style. When recursion stops, the answer is found in one of the function's parameters. This may help you form the iterative solution.

### 2.1 Requirements

- Name your function `gcd_iter`. Your function must have two parameters. You may assume that the arguments provided are integer values. That is, you do not need to do any error-checking to confirm that they are integer values.
- Write a test function called `test_gcd_iter` that tests your `gcd_iter` function by calling it several times with different argument values. You should have at least six (and perhaps more) tests. Choose arguments that test different aspects of your function.
  For example, you may have one test `gcd_iter(5,5)`, that checks that the function works properly when the arguments are both positive and the same value. Once you have this test, you do not need another test `gcd_iter(8,8)`, as this checks the same thing.
  You should not use any tests involving arguments that are not integers.

**Example Function Operation**

Output would look like this if you were to call `gcd_iter` from the interactive console:

```
>>> from gcd import gcd_iter
>>> gcd_iter(6,4)
2
```

## 3 Task 3

Write a `main` function to allow a user to choose and run one of GCD functions. Add the code to the `gcd.py` file.

The `main` function should display the available functions to the user and allow them to choose the function to use. Then, based on the function selected, it should prompt the user for the required function inputs.

**Example Program Operation Output**

```
Select the GCD function to use:
```

```
1. Recursive
2. Iterative

Please select a function: 1

Please enter the first number: 48
Please enter the second number: 18

The greatest common denominator is 6
```

Since others may want to import the `gcd.py` module, you must put the calls to the `main` function and the test functions inside an `if __name__ == "__main__":` statement. This statement is the last code of the file, and it should be at the left margin of the file. Here is an example.

```
if __name__ == "__main__":
    # insert calls to your test functions here
    main()
```

By using this `if __name__ == "__main__":` statement, other code can import the module without executing the tests or the main.

## 4 Programming Tips

### 4.1 Modulo

Python provides an operator that computes the remainder when dividing one number by another. This operator is known variously by the names *modulo*, *modulus*, or *remainder*.

Here are some examples in the interactive console:

```
>>> 8 % 3
2
>>> 4 % 11
4
>>> 60 % 5
0
```

### 4.2 Testing

The testing functions should each call the respective functions with a few different *canned and hard-coded* values. Consider cases that are interesting, which might include:

1. Two numbers that have a GCD value that is less than either number
2. Two numbers in which one of them is the GCD
3. The same two numbers

4. Two numbers in which the GCD is 1
5. First number is larger then the second and vice versa
6. Other interesting cases

For each test case, the output from the tests should be something similar to:

```
GCD(48, 18) = 6
```

### 4.3 Naming

The grading process will import your module and run other, private tests to verify that your functions work properly.

**You must name your file and your functions as specified in the assignment**.

## 5 Grading

- 30%: Correct functionality and design of the `gcd_rec` function. Deductions up to this limit are:
  - -20% if the implementation type is not tail-recursive.
  - -5% if the algorithm is incorrect.
  - -5% if the function name and *parameter specs* are not as required.
  - -10% for infinite recursion and faulty base case(s).
- 30%: Correct functionality and design of the `gcd_iter` function. Deductions up to this limit are:
  - -20% if the implementation does not use `while`).
  - -5% if the algorithm is incorrect.
  - -5% if the function name and *parameter specs* are not as required.
  - -10% for infinite recursion and faulty base case(s).
- 10%: Correct functionality and design of the `main` function. Distribution is:
  - 2% displaying a menu
  - 2% handling user selection
  - 2% prompting for inputs to the function
  - 2% calling the function and getting the result
  - 2% displaying the result.
- 10%: Testing provided in `test_gcd_rec`. Deductions up to this limit are:
  - -2% if the function name is not as required.
  - up to -8% if there is “insufficient testing”.
- 10%: Testing provided in `test_gcd_iter`. Deductions up to this limit are:
  - -2% if the function name is not as required.
  - up to -8% if there is “insufficient testing”.

- 5%: Each function has a *docstring* containing a sentence describing its purpose. This documentation helps others understand how they may reuse the function. An example is provided on the Course Resources webpage:
  `http://www.cs.rit.edu/~csci141/Docs/style-example-py.txt`
- 5% The program has correct, standard style, starting with a *docstring* for the whole file. This program file docstring must contain your *first and last* name.

### 5.1 Submission

Put your program code with the appropriate documentation into a file called `gcd.py` and zip the file into `hw04.zip`.

Submit the zip file to the correct MyCourses assignment dropbox.