Lab 1 – Converting Pseudocode to Python Code

(a.k.a. a bit more Python programming)

Release: 24 Aug 2020 (Mon, Week 2)
1 week to attempt
Due: 30 Aug 2020, 11pm (Sun)

Some Words:

Most algorithms documented in textbooks are given in the form of pseudocode. This lab consists of exercises that require students to convert pseudocode into Python code. Pseudocode is not standardized, and hence you will see different "versions" in textbooks and on the Internet. The objective of this lab is not to understand the formulas behind algorithms, but to practice converting pseudocode to actual Python code that you can run.

Instructions:

- There are 4 questions in this exercise to be completed individually.
- For this exercise, your team ID is your name (i.e. you are the only member of your team).
- You need to submit code for this exercise at the Submission Server. No written submission is required.
- Edit **lab1a.py, lab1b.py, lab1c.py** and **lab1d.py** that are given to you, and submit them to the Submission Server.
- You can submit your solutions to the Submission Server as many times as you wish, but the final submission on the deadline will be taken as your final submission.

In this lab, you will write the Python code for four different algorithms to solve the same problem: to find the GCD (Greatest Common Divisor) of two positive integers. The first two algorithms will be covered in week 3's lecture: (i) brute force and (ii) Dijkstra's algorithm. The other two are well documented on the Internet: (iii) Euclid's algorithm and (iv) Binary/Stein's algorithm.

Requirements (same for all four questions):

- Write a function that takes in two arguments **x** and **y** (both positive integers), and return the GCD of **x** and **y**, using the prescribed algorithm.
- The functions for Q1(a), (b), (c) and (d) are called **gcd_a**, **gcd_b**, **gcd_c** and **gcd_d** respectively. They are found in **lab1a.py**, **lab1b.py**, **lab1c.py** and **lab1d.py** respectively.

You are given the following file(s) for this exercise:

File name	Description	Comments
lab1a.py, lab1b.py, lab1c.py, lab1d.py	Contains the **gcd_a, gcd_b, gcd_c** and **gcd_d** functions that you will write.	You need to modify and submit these files. Do not modify the file names or the function signatures.
lab1_main.py	Loads **lab1a.py, lab1b.py, lab1c.py** and **lab1d.py** and calls the **gcd** function using the test cases described below.	Do not submit this file; use it to check if your functions in **lab1a/b/c/d.py** work correctly before submitting them to the Submission Server. You should start testing your **lab1a.py** immediately once you have completed Q1(a) using **lab1_main.py**. Do not wait till you have completed all four questions before running **lab1_main.py**.

Your task:

- Edit **lab1a/b/c/d.py** provided to meet the requirements.
  There will be six test cases used to evaluate your functions in **lab1_main.py** and the submission server:
  1. Test case 1: **gcd_x(5352, 6690)** # should return **1338**
  2. Test case 2: **gcd_x(7800111, 393945)** # should return **78789**
  3. Test case 3: **gcd_x(75116, 6752)** # should return **844**
  4. Test case 4: **gcd_x(7999992, 1999998)** # should return **1999998**
  5. Test case 5: **gcd_x(2, 6)** # should return **2**
  6. Test case 6: **gcd_x(1000, 1)** # should return **1**

To submit:

- **lab1a.py, lab1b.py, lab1c.py** and **lab1d.py** (to submission server). Edit the comments at the top of your Python file to indicate your name and section.

Assessment:

- This exercise is not graded but submission of a working answer is mandatory.
- For this exercise, you should ignore the “Time Taken” on the Scoreboard. The Quality Score will always be “1.0” if your solution is correct. So, as long as your team has a valid “Time Taken” and “Quality Score” on the Scoreboard, your solution is correct.

For all four algorithms, the inputs are two non-negative integers **x** and **y**. And the output (i.e. what the function returns) is the GCD of the original input values. The descriptions and algorithms (described in various forms of pseudocode) are given below.

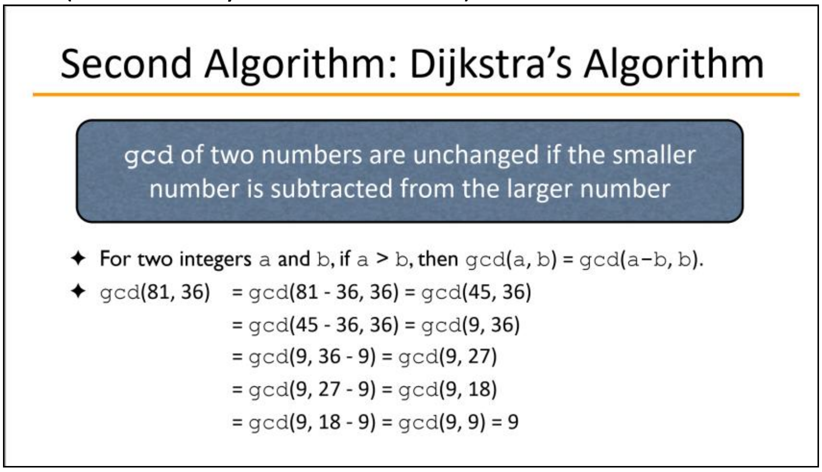
Q1(a) 1st algorithm: Brute Force

Description:
- For two numbers **x** and **y**, its GCD is between **1** and the smaller of **x** and **y**.
- Try all reasonable possibilities (i.e. try (**x** or **y**), … 3, 2, 1)

Algorithm:
1) set **t** to the minimum of **x** and **y**
2) repeat until **t** equals **1**:
   a) if **x** and **y** are both divisible by **t**, return **t** as output
   b) else, subtract **1** from **t** (i.e. **t** = **t** – **1**)

Q1(b) 2nd algorithm: Dijkstra's algorithm

Description (taken from your week 3 slides):



Algorithm:
1) repeat until **x** is equal to **y**:
   a) if **x** is larger than **y**, subtract **y** from **x**
   b) else, subtract **x** from **y**
2) return **x** as output

Q1(c) 3rd algorithm: Euclid's algorithm

Description[1]:
- Given two integers x and y (say x > y), then
  - GCD(x, y) = GCD(y, x mod y)
  - One can continue using the above recursion until the second term becomes 0. The GCD(x, y) will be then the value of the first term, because GCD(k, 0) = k
- Examples:
  - GCD(120, 45) = GCD(45, 30) = GCD(30, 15) = GCD(15, 0) → GCD is 15
  - GCD(45, 12) = GCD(12, 9) = GCD(9, 3) = GCD(3, 0) → GCD is 3
  - GCD(53, 30) = GCD(30, 23) = GCD(23, 7) = GCD(7, 2) = GCD(2, 1) = GCD(1, 0) → GCD is 1

[1] Explanation taken from https://www.youtube.com/watch?v=SNJq2f0vXwc

Algorithm:

- Use the following pseudocode[2]:

```
function gcd(x, y)
  while y ≠ 0
    t := y;
    y := x mod y;
    x := t;
  return x;
```

Q1(d) 4th algorithm: Binary GCD algorithm (aka Stein's algorithm)

Description:

- See appendix.

Algorithm:

Use the following pseudocode[3]:

```
k := 0

while x and y are both even
  x := x / 2
  y := y / 2
  k := k + 1

while x ≠ y
  if x is even then x := x/2
  else if y is even then y := y/2
  else if x > y then x := (x − y)/2
  else y := (y − x)/2

return x × 2^k
```

~ End

[2] Adapted from https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations

[3] Adapted from https://en.wikipedia.org/wiki/Greatest_common_divisor#Binary_GCD_algorithm

Appendix: Binary GCD algorithm (aka Stein's algorithm)

More information about Stein's algorithm is given here for students interested in how this works.

Stein's algorithm uses a number of rules[4]:

1. If both of the values are even numbers we know that two is a common divisor. We can divide both values by two and find the GCD of the two new values. Multiplying this result by two gives the GCD of the original values. ie. *GCD(a,b) = 2 * GCD(a/2,b/2)*.
2. If only one of the values is even, we know that the value two is not a common divisor. We can therefore divide the even value by two and recalculate the GCD. ie. *GCD(even,odd) = GCD(even/2,odd)*.
3. If both of the values are odd, we need to use subtraction in the same manner as in a single step of Dijkstra's algorithm. The smaller value is subtracted from the larger and the result is used with the smaller value to calculate the GCD. ie. *GCD(large,small) = GCD(large-small,small)*. We can go a step further than this. When one odd value is subtracted from another we know that the result will be even. This means that we will be calculating the GCD of an odd and an even value. We can therefore divide the even number by two, as in step 4. ie. *GCD(large,small) = GCD((large-small)/2,small)*.
4. Repeat steps 3-5 until either the conditions in step 1 or step 2 are fulfilled.
5. The final GCD is computed as: $\mathbf{a * 2^k}$ where k is the number of common factors of two found in step 3.

[4] Adapted from http://www.blackwasp.co.uk/SteinsAlgorithm.aspx