Workshop 4: More about Loops, Algorithms and Root Finding

11 November 2020

Goals:

- Know more about algorithms and writing programs
- Gain experience using algorithms using loops and conditionals

If you have any questions during this session please ask! You should ensure that you have completed all exercises before the next workshop.

NB. We may see a few different ways of formating a print statement. They are largely equivalent and you may use whatever works best. For example:

```
 """Example print statements"""
y = 25.1234
x = 7.34587
print(’Old way: y = {}, x = {} ’.format(y, x) )     # rather clunky
print(f’New way: y = {y}, x = {x} ’ )               # easier to use
print(f’New way with some formatting: y = {y:3.2f}, x = {x:7.3f} ’ )
print(’Casual way: y =  ’, y,  ’x = ’, x )        # fine for casual use,
                                                  # hard to get proper formatting though.
```

Algorithms

An **algorithm** is a process, set of instructions, or set of rules used in a computation. In the last workshop we experimented with the method of Heron of Alexandria and Exhaustive Enumeration. In the next sections we will explore two additional algorithms to advance our experience with loops, conditionals, variables, and lists.

Root finding algorithms

The idea of root finding algorithms is to solve the equation:

$$F(x) = 0.$$

In our example, if we are interested in finding the square root of a number x_0 with the answer being x then we are interested in,

$$x_0 = \sqrt{x},$$

and thus the equation we're trying to solve with a root solver is,

$$\sqrt{x} - x_0 = 0, \qquad (1)$$

Refering to equation (1), we can test this equation by looking at a familiar example: in the case where we are searching for the square root of 4, y is 4 and x_0 is 2. If we were searching for the square root of 5, then $y = 5$ and x_0 is the unknown number we're going to compute with an algorithm.

therefore,

$$F(x) = \sqrt{y} - x = 0.$$

Or, if we square each side of equation (1) we can alternatively write this,

$$F(x) = y - x^2 = 0.$$

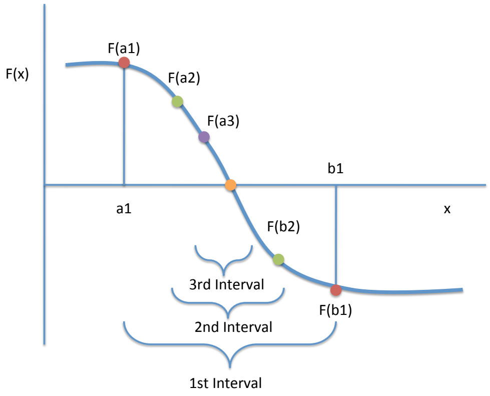
1. **Exhaustive Enumeration (reminder)** or “brute force search” is a general method of finding the solution to a problem. We begin with a guess, change it by a small amount, check, and continue until it’s close enough. Here is demonstration code from the class reference book (*Introduction to Computation and Programming using Python, by John Guttage*, Figure 3.3) that finds the square root of x:

```
"""Find square root of number using exhaustive enumeration."""
y = 25.0
epsilon = 0.01
step = epsilon**2
num_guesses = 0
tot_guesses = 1.e7
ans = 0.0
while abs(ans**2 - y) >= epsilon and num_guesses <= tot_guesses:
    ans += step
    num_guesses += 1
print(’Num Guesses: {}’.format(num_guesses))
if abs(ans**2 - y) >= epsilon:
    print(f’Failed to find sqrt of {x}’)
else:
    print(f’{ans} is close to sqrt of {y}’)
```

2. **Bisection Search** is a root-finding method that repeatedly bisects an interval, then chooses which subinterval contains the root before continuing the search. The method is often robust, but also not as efficient as some other methods.[1]

   The following code is an algorithm to approximate the square root of x using the Bisection Method, from Guttag’s book.

```
"""Find square root of number using bisection."""
y = 25
eps = .01
num_guesses = 0
tot_guesses = 1e2
low = 0.0
```



[1] Bisection method illustration

```
high = max(1.0, y)
ans = (high + low)/2.0

while abs(ans**2 - y) >= eps and num_guesses < tot_guesses:
    print(’L= {} H= {} ANS= {}’.format(low, high, ans))
    num_guesses += 1
    if ans**2 < y:
        low = ans
    else:
        high = ans
    ans = (high + low) / 2.0

print(’num_guesses = {}’.format(num_guesses))
print(’{} is close to square root of {}’.format(ans, x))
```

3. **Newton-Raphson** is an approximation algorithm attributed to Isaac Newton and Joseph Raphson. It is often simply called Newton’s Method. It can be used to find the roots of many different types of functions. For us we will use it only for our polynomial problem. First, to more easily understand the Newton-Raphson algorithm, we consider an alternative way of thinking about finding the root than we did in with equation (1). Consider

$$F(y) = 0, \qquad y \text{ is a real number.}$$

Define the derivative of F to be $F' = \frac{dF}{dy}$. If we begin with a guess y_0 then a good second guess, according to the Newton-Raphson method is,

$$y_1 = y_0 - \frac{F(y_0)}{F'(y_0)}.$$

This process is then repeated. For programs to find the square root we are considering the polynomial $F(y) = y \times y - x = 0$, where x represents the number we want to know the square root of, so it is fixed. Taking the derivative we find that $F'(y) = 2y$. So for the algorithm of interest the new guess for the iteration $n + 1$ can be written:

$$y_{n+1} = y_n - \frac{y_n * y_n - x}{2 * y_n}.$$

The following is an algorithm to approximate the square root of x using the Newton–Raphson method from our class text book Figure 3.5, “Introduction to Computation and Programming Using Python” by John V. Guttag.

```
"""Find square root of number using Newton-Raphson."""
y = 25.0
epsilon = .01
guess = y / 2.0   # first guess

while abs(guess * guess - y) >= epsilon:
    guess -= (guess**2 - y) / (2 * guess)
print('{} is close to square of {}'.format(guess, y))
```

It turns out that this algorithm is exactly the same as the one due to Heron of Alexandria we used in the last Workshop. Can you see why?

Strategies for developing an algorithm

1. For your assessment you'll be asked to write an algorithm that solves a problem using python.
2. These are a few ideas you can use to develop an algorithm
   (a) Use a flow chart – see Figure 1.

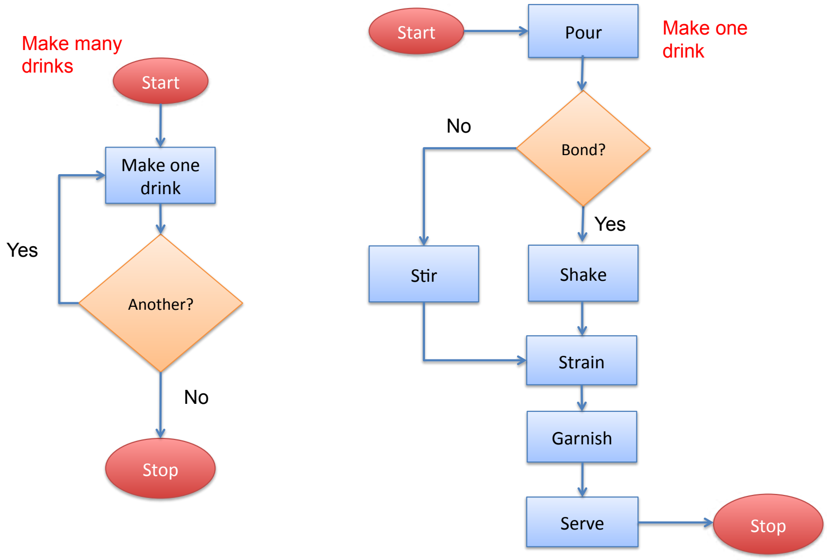


Figure 1: Flow Chart Example: Branching can be seen easily with if statements. Instructions for actions in boxes.

   i. Identify the main processes (or calculations)
   ii. Identify the main decision points (or if/elif/else)
   iii. Identify possible loops (or iterations)
   iv. Identify the variables that you know. These may need to be initialized.
   v. Identify the outputs that you want.

vi. Sketch the flow of the algorithm by using a flow chart – see Figure (1). One of the best ways to get started making a flow chart is to write what you know at the top of a piece of paper (these variables may need to be initialized) and what you want at the bottom. Fill in the required steps and conditionals to get from what you know to what you need.

vii. Step through what you have to see if it makes sense.

(b) The creative drawing method can help. Begin by writing down the processes, conditional tests, data that you know, the output or endpoint on a piece of paper. Circle each step. Draw circles around groups of processes that are connected. This is illustrated on the left side of Figure (1). Use lines and arrows to show yourself where they fit into the 'flow' of the algorithm. Once you have a good idea, redraw the flow. This is illustrated on the right side of Figure (1). Do this until you have sketched the flow of the algorithm.

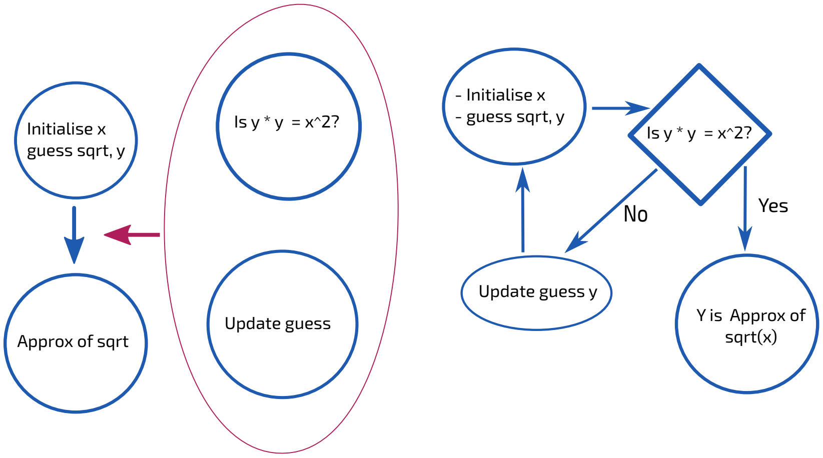


Figure 2: The sketch on the left shows an initial look at the main steps of the square root algorithm. Each step is circled in blue. A collection of more closely related steps are circled in red, and you can see they must fit in between the what is known, and what you want at the end (the square root of x). The sketch on the right is a refinement of the one on the left. The idea is to modify the sketch until you can clearly see each step of the algorithm.

(a) There are many ways of developing and writing the same algorithm, be creative in how you think about it.

Exercises

These exercises should be completed before the next week. For each program you write you should make sure you test each aspect (e.g., each branch when using conditional statements) and ensure that you have used suitable comments.

1. Copy and debug the program for finding the square root using the Bisection Method.
   (a) Add comments using the hash # mark that describes what each part of the code does.
   (b) Run the program. When you run it it should print out many lines, and finally that the num_guesses = 13.
   (c) Does it find the same answer when x=25 as the Exhaustive Enumeration method? It's OK if it isn't exact, since it is still a very good approximation.
   (d) How many iterations does it take if y =12000? What does this code do if y=-25?[2]

2. Copy this program to another file and rename it *cuberoot_bisection.py*. Change the code so that rather than approximating the square root it will approximate the cube root.
   (a) Add comments using the hash # mark that describes what each part of the code does.
   (b) Find the cube root of 9, 42, and 343.

3. Copy and debug the program for using the Newton-Raphson method to approximate the square root.
   (a) Add comments using the hash # mark that describes what each part of the code does.
   (b) Run the program.
   (c) Add some code to the algorithm so that it keeps track of the number of iterations used to find the root.
   (d) How many iterations does Newton-Raphson take to find the square root of 25?
   (e) Of 12000?

[2] **Hint:** think about changing low or high to ensure the answer lies within the region being searched! Remember to use Ctrl + c to exit a python program that's running.

(f) Compare Newton-Raphson's efficiency to the Bisection method and the Exhaustive Enumeration method.

(g) Does each method give the same approximation for the square root of 25? How different are they?

4. Copy this program to another file and rename it `cuberoot_newton.py`. Change the code so that rather than approximating the square root it will approximate the cube root.

(a) Add comments using the hash # mark that describes what each part of the code does.

(b) Find the cube root of 5, 9, 42, and 343.

5. In terms of number of operations and accuracy, how well does the Newton method compare to the Bisection method for finding the cube root of 5, 9, 42, and 343? **Hint:** You get to decide on which numbers of operations to count, and what accuracy means. For example, it may take 10000 guesses to find the square root of 25 that is accurate to within .0001 of the correct answer.

6. Show that the Newton method for square roots is essentially the same as the method of Heron of Alexandria. (However, Newton's method is more general and can be used for other problems, like the cube root.)