JJenkins_HW#3

August 27, 2018

1 Assignment 3

You may use any methods available to work the following 2 programming problems. They are bit more challenging, than the previous assignments, but not much. There are many ways to solve both problems, so I am not looking for a single 'correct' solution.

1.1 Problem 1 (8 Points)

One of the unsolved problems in mathematics is the Collatz or Hailstone problem. Basically, here are the rules:

- Supply an integer to the function
- If the integer is even, divide by 2
- If the integer is odd, multiply by 3 and add 1
- Take the new value and go again... if even, divide by 2, if odd, multiply by 3 and add 1
- The process stops when the value reaches 1

For example, if you write the program correctly, using:
collatz(6)
Will result in the function determining that 6 is even and then dividing 6/2 producing 3. 3 is odd, so the function will perform 3*(3)+1 = 10. 10 is even, which results in 10/2 = 5. 5 is odd -> 3*(5)+1 = 16. 16 is even, 16/2 = 8. 8 is even, 8/2 = 4. 4 is even, 4/2 = 2. 2 is even, 2/2 = 1 and we are finished.

So, the collatz(6) function call results in the following Collatz sequence:
6 - 3 - 10 - 5 - 16 - 8 - 4 - 2 - 1

The unsolved portion is whether or not ALL integers eventually reach 1 or stop. Nobody knows. Some integers reach 1 quickly and others take several iterations or loops. For example, collatz(97) takes 118 steps.

Your taks is to create a function, collatz(), that will take an integer you supply and then print the resulting Collatz numbers until the number 1 is reached.

```
In [5]: def collatz(n):
            # take a supplied integer and prints the resulting Collatz numbers until the numbe
            original_n = n #Storing the original n to use in print statement
            collatz_sequence = [n]
            if n < 1:
                # only dealing with positive integers greater than 1 and returning an empty li
                return []
```

```
    while n > 1:
        # if integer is even divide by 2
        if n % 2 == 0:
            n = n / 2
        else:
            # if integer is odd multiply by 3 and add 1
            n = n * 3 + 1
        # append the new value to the list and take this value and loop through again
        collatz_sequence.append(int(n))
    return print("\nFor the integer " + str(int(original_n)) +
                 ", there are " + str(len(collatz_sequence)) +
                 " steps until the Collatz number reaches 1" +
                 " as shown in the following sequence:\n\n", collatz_sequence)

collatz(6)
collatz(97)
```

```
For the integer 6, there are 9 steps until the Collatz number reaches 1 as shown in the follow

 [6, 3, 10, 5, 16, 8, 4, 2, 1]

For the integer 97, there are 119 steps until the Collatz number reaches 1 as shown in the fol

 [97, 292, 146, 73, 220, 110, 55, 166, 83, 250, 125, 376, 188, 94, 47, 142, 71, 214, 107, 322,
```

1.2 Problem 2 (4 Points)

We wrote a function to find out if a number is prime. Write a program that will print the prime numbers from 5 to 10000. 2 is the only even prime number and 3 is almost as trivial, so we will start with 5 to make your task easier.

You can use the isprime program written in the last notebook, but you will need to add something to the program to solve the problem. Also, try BOTH (n/2) for stopping the divisor and then try (n**0.5). Which one is faster? If you can't tell, try printing the prime to 100000 using both divisor stopping methods.

I apologize but I had to put the different portions of Problem 2 in different cells so I could focus on what I was trying to do.

I used the next cell to look at (n/2) as my stopping divisor.

```
In [6]: """Using this cell to look at (n/2) as my stopping divisor"""
        def isprime(n):
            from timeit import default_timer as timer
            initial_i = 5
            i = initial_i
            upper_bound = n+1
```

```
    stopping_divisor0 = n
    stopping_divisor1 = (n/2)
    stopping_divisor2 = (n**(1/2))
    my_primes1 = []
    my_primes2 = []


    for number in range(i, int(stopping_divisor1)):
        start1 = timer()
        # only looking at positive integers
        if number > 1:
            for i in range(i, number):
                if (number % i) == 0:
                    break
                else:
                    my_primes1.append(i)
#                     print("The following number " + str(i) + " is prime.")
                i = i + 2
        end1 = timer()
        elapsed1 = end1 - start1


#     print("Using (n/2) as the stopping divisor, it took " + ("%.15f" % elapsed1) +
#           " seconds to determine that there are " + str(len(my_primes1)) + " prime n
#           str(initial_i) + " and " + str(n) + ".\n")

    for number in range(i, int(stopping_divisor2)):
        start2 = timer()
        # only looking at positive integers
        if number > 1:
            for i in range(i, number):
                if (number % i) == 0:
                    break
                else:
                    my_primes2.append(i)
#                     print("The following number " + str(i) + " is prime.")
                i = i + 2
        end2 = timer()
        elapsed2 = end2 - start2


    print("Using (n/2) as the stopping divisor, it took " + ("%.15f" % elapsed1) +
          " seconds to determine that there are {0:,g}".format(len(my_primes1)) + " pri
          str(initial_i) + " and {0:,g}".format(n) + ".\n")

    print("Using (n/2) as the stopping divisor, the ".format(len(my_primes1)) + " prime
          str(initial_i) + " and {0:,g}".format(n) + " are:\n", my_primes1)
```

```
isprime(100000)
```

```
Using (n/2) as the stopping divisor, it took 0.000000300457202 seconds to determine that there

Using (n/2) as the stopping divisor, the  prime numbers between 5 and 100,000 are:
 [5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51,
```

I used the next cell to look at (n0.5) as my stopping divisor.**

```
In [7]: """Using this cell to evaluate (n**0.5) as the stopping divisor"""
        def isprime(n):
            from timeit import default_timer as timer
            initial_i = 5
            i = initial_i
        #     upper_bound = n+1
        #     stopping_divisor0 = n
        #     stopping_divisor1 = (n/2)
            stopping_divisor2 = (n**(1/2))
        #     my_primes1 = []
            my_primes2 = []


            for number in range(i, int(stopping_divisor2)):
                start2 = timer()
                # only looking at positive integers
                if number > 1:
                    for i in range(i, number):
                        if (number % i) == 0:
                            break
                        else:
                            my_primes2.append(i)
        #                     print("The following number " + str(i) + " is prime.")
                        i = i + 2
                end2 = timer()
                elapsed2 = end2 - start2


            print("Using (n**0.5) as the stopping divisor, it took " + ("%.15f" % elapsed2)
                  + " seconds to determine that there are {0:,g}".format(len(my_primes2))
                  + " prime numbers between " + str(initial_i) + " and {0:,g}".format(n) + ".\

            print("Using (n**0.5) as the stopping divisor, the {0:,g}".format(len(my_primes2))
                  " prime numbers between " + str(initial_i) + " and {0:,g}".format(n) + " are

        isprime(100000)
```

```
Using (n**0.5) as the stopping divisor, it took 0.000000600914404 seconds to determine that th
```

```
Using (n**0.5) as the stopping divisor, the 155 prime numbers between 5 and 100,000 are:
 [5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51,
```

Which one was faster, (n/2) or (n0.5)??**

```
In [1]: def speedcheck1(n):
            from timeit import default_timer as timer
            initial_i = 5
            i = initial_i
            upper_bound = n+1
            stopping_divisor0 = n
            stopping_divisor1 = (n/2)
            stopping_divisor2 = (n**(1/2))
            my_primes1 = []
            my_primes2 = []


            for number in range(i, int(stopping_divisor1)):
                start1 = timer()
                # only looking at positive integers
                if number > 1:
                    for i in range(i, number):
                        if (number % i) == 0:
                            break
                        else:
                            my_primes1.append(i)
        #                     print("The following number " + str(i) + " is prime.")
                        i = i + 2
                end1 = timer()
                elapsed1 = end1 - start1

        #     print("Using (n/2) as the stopping divisor, it took " + ("%.15f" % elapsed1) +
        #           " seconds to determine that there are {0:,g}".format(len(my_primes1)) + "
        #           str(initial_i) + " and {0:,g}".format(n) + ".\n")



            return ("%.15f" % elapsed1)

        def speedcheck2(n):
            from timeit import default_timer as timer
            initial_i = 5
            i = initial_i
            upper_bound = n+1
            stopping_divisor0 = n
            stopping_divisor1 = (n/2)
            stopping_divisor2 = (n**(1/2))
```

```
    my_primes1 = []
    my_primes2 = []


    for number in range(i, int(stopping_divisor2)):
        start2 = timer()
        # only looking at positive integers
        if number > 1:
            for i in range(i, number):
                if (number % i) == 0:
                    break
                else:
                    my_primes2.append(i)
#                     print("The following number " + str(i) + " is prime.")
                i = i + 2
        end2 = timer()
        elapsed2 = end2 - start2

#     print("Using (n**0.5) as the stopping divisor, it took " + ("%.15f" % elapsed2) +
#           " seconds to determine that there are {0:,g}".format(len(my_primes2)) + " p
#           str(initial_i) + " and {0:,g}".format(n) + ".\n")
    return ("%.15f" % elapsed2)

print("The time to run with (n/2) as the stopping divisor in seconds was: ", speedcheck
print("The time to run with (n**0.5) as the stopping divisor in seconds was: ", speedch

elapsed1 = float(speedcheck1(100000))
elapsed2 = float(speedcheck2(100000))
diff = elapsed1 - elapsed2 #means (n/2) took longer to run than (n**0.5)
#print(diff)
if diff > 0:
    print("\nUsing (n**0.5) as the stopping divisor was " + ("%.15f" % diff) + " second
else:
    print("\nUsing (n/2) as the stopping divisor was " + ("%.15f" % diff) + " seconds f
```

```
The time to run with (n/2) as the stopping divisor in seconds was:  0.000000300457206
The time to run with (n**0.5) as the stopping divisor in seconds was:  0.000000300457206

Using (n**0.5) as the stopping divisor was 0.000000300457206 seconds faster than using (n/2).
```