Kernel: SageMath 10.7

Sieve of Eratosthenes

MATH 242 Modern Computational Mathematics

Primality Testing by Trial Division (from last time)

In the previous class, we wrote a function `isPrime(num)` that accepts a positive integer `num` and determines whether it is prime. Here is one way to do this.

In [0]:
```
def isPrime(num):
    # there are no primes smaller than 2
    if num < 2:
        return False

    # check whether num is even
    if num == 2:
        return True   # 2 is the only even prime
    if num % 2 == 0:
        return False  # other even numbers are not prime

    # check whether num is divisble by 3, 5, 7, 9, , ..., num-2
    for k in range(3, num//2, 2):
        if num % k == 0:
            return False

    # if we get here, then the loop finished without finding a divisor of
num, so num is prime
    return True
```

Test our `isPrime` function for various values of `n`:

In [0]:
```
print(isPrime(1))
print(isPrime(2))
print(isPrime(4))
print(isPrime(7))
print(isPrime(25))
print(isPrime(29))
```

Implementing the Sieve of Eratosthenes

Write a function that uses the Sieve of Eratosthenes to produce a list of primes. Your function should take one parameter, `nMax`. It should return a list of all primes from 2 up to `nMax`.

In [0]:
```
#
# BEFORE YOU TYPE ANY CODE,
```

```
# PLAN YOUR FUNCTION ON PAPER!
#
```

In [2]:
```
# here is a way to make a list
max = 20
nums = list(range(1,20))
nums
```

Out[2]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Here is a working implentation of the sieve of Eratosthenes:

In [12]:
```
def sieveEratosthenes(nMax):
    # initialize a list
    # if you start the list from zero, then each number in the list is
EQUAL to its index in the list, which is very convenient
    nums = list(range(0, nMax + 1))

    # we will replace non-prime numbers in the list with zeros
    # start by replacing 1 with 0
    nums[1] = 0

    # loop over list items until we reach sqrt(nMax)
    i = 2
    while i <= sqrt(nMax):
        # if nums[i] is not zero, then it is prime
        if nums[i] > 0:
            # replace all multiples of nums[i] with zeros
            for j in range(2*i, nMax + 1, i):
                nums[j] = 0

        # testing: print the current state of nums
        print(f"finished i={i}: nums={nums}\n")

        # increment i
        i += 1

    # return a list containing all nonzero elements of nums
    return [i for i in nums if i != 0]
```

Confirm that the sieve of Eratosthenes works. Notice how the `print` statement allows you to see what is happening in the list `nums` with each iteration of the loop.

In [13]:
```
primeList = sieveEratosthenes(100)

print("All done! Final list of primes is:")
print(primeList)
```

Out[13]:
```
finished i=2: nums=[0, 0, 2, 3, 0, 5, 0, 7, 0, 9, 0, 11, 0, 13, 0, 15, 0,
17, 0, 19, 0, 21, 0, 23, 0, 25, 0, 27, 0, 29, 0, 31, 0, 33, 0, 35, 0, 37, 0,
39, 0, 41, 0, 43, 0, 45, 0, 47, 0, 49, 0, 51, 0, 53, 0, 55, 0, 57, 0, 59, 0,
61, 0, 63, 0, 65, 0, 67, 0, 69, 0, 71, 0, 73, 0, 75, 0, 77, 0, 79, 0, 81, 0,
83, 0, 85, 0, 87, 0, 89, 0, 91, 0, 93, 0, 95, 0, 97, 0, 99, 0]

finished i=3: nums=[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17,
0, 19, 0, 0, 0, 23, 0, 25, 0, 0, 0, 29, 0, 31, 0, 0, 0, 35, 0, 37, 0, 0, 0,
```

```
41, 0, 43, 0, 0, 0, 47, 0, 49, 0, 0, 0, 53, 0, 55, 0, 0, 0, 59, 0, 61, 0, 0,
0, 65, 0, 67, 0, 0, 0, 71, 0, 73, 0, 0, 0, 77, 0, 79, 0, 0, 0, 83, 0, 85, 0,
0, 0, 89, 0, 91, 0, 0, 0, 95, 0, 97, 0, 0, 0]

finished i=4: nums=[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17,
0, 19, 0, 0, 0, 23, 0, 25, 0, 0, 0, 29, 0, 31, 0, 0, 0, 35, 0, 37, 0, 0, 0,
41, 0, 43, 0, 0, 0, 47, 0, 49, 0, 0, 0, 53, 0, 55, 0, 0, 0, 59, 0, 61, 0, 0,
0, 65, 0, 67, 0, 0, 0, 71, 0, 73, 0, 0, 0, 77, 0, 79, 0, 0, 0, 83, 0, 85, 0,
0, 0, 89, 0, 91, 0, 0, 0, 95, 0, 97, 0, 0, 0]

finished i=5: nums=[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17,
0, 19, 0, 0, 0, 23, 0, 0, 0, 0, 0, 29, 0, 31, 0, 0, 0, 0, 0, 37, 0, 0, 0,
41, 0, 43, 0, 0, 0, 47, 0, 49, 0, 0, 0, 53, 0, 0, 0, 0, 0, 59, 0, 61, 0, 0,
0, 0, 0, 67, 0, 0, 0, 71, 0, 73, 0, 0, 0, 77, 0, 79, 0, 0, 0, 83, 0, 0, 0,
0, 0, 89, 0, 91, 0, 0, 0, 0, 0, 97, 0, 0, 0]

finished i=6: nums=[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17,
0, 19, 0, 0, 0, 23, 0, 0, 0, 0, 0, 29, 0, 31, 0, 0, 0, 0, 0, 37, 0, 0, 0,
41, 0, 43, 0, 0, 0, 47, 0, 49, 0, 0, 0, 53, 0, 0, 0, 0, 0, 59, 0, 61, 0, 0,
0, 0, 0, 67, 0, 0, 0, 71, 0, 73, 0, 0, 0, 77, 0, 79, 0, 0, 0, 83, 0, 0, 0,
0, 0, 89, 0, 91, 0, 0, 0, 0, 0, 97, 0, 0, 0]

finished i=7: nums=[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17,
0, 19, 0, 0, 0, 23, 0, 0, 0, 0, 0, 29, 0, 31, 0, 0, 0, 0, 0, 37, 0, 0, 0,
41, 0, 43, 0, 0, 0, 47, 0, 0, 0, 0, 0, 53, 0, 0, 0, 0, 0, 59, 0, 61, 0, 0,
0, 0, 0, 67, 0, 0, 0, 71, 0, 73, 0, 0, 0, 0, 0, 79, 0, 0, 0, 83, 0, 0, 0, 0,
0, 89, 0, 0, 0, 0, 0, 0, 0, 97, 0, 0, 0]

finished i=8: nums=[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17,
0, 19, 0, 0, 0, 23, 0, 0, 0, 0, 0, 29, 0, 31, 0, 0, 0, 0, 0, 37, 0, 0, 0,
41, 0, 43, 0, 0, 0, 47, 0, 0, 0, 0, 0, 53, 0, 0, 0, 0, 0, 59, 0, 61, 0, 0,
0, 0, 0, 67, 0, 0, 0, 71, 0, 73, 0, 0, 0, 0, 0, 79, 0, 0, 0, 83, 0, 0, 0, 0,
0, 89, 0, 0, 0, 0, 0, 0, 0, 97, 0, 0, 0]

finished i=9: nums=[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17,
0, 19, 0, 0, 0, 23, 0, 0, 0, 0, 0, 29, 0, 31, 0, 0, 0, 0, 0, 37, 0, 0, 0,
41, 0, 43, 0, 0, 0, 47, 0, 0, 0, 0, 0, 53, 0, 0, 0, 0, 0, 59, 0, 61, 0, 0,
0, 0, 0, 67, 0, 0, 0, 71, 0, 73, 0, 0, 0, 0, 0, 79, 0, 0, 0, 83, 0, 0, 0, 0,
0, 89, 0, 0, 0, 0, 0, 0, 0, 97, 0, 0, 0]

finished i=10: nums=[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0,
17, 0, 19, 0, 0, 0, 23, 0, 0, 0, 0, 0, 29, 0, 31, 0, 0, 0, 0, 0, 37, 0, 0,
0, 41, 0, 43, 0, 0, 0, 47, 0, 0, 0, 0, 0, 53, 0, 0, 0, 0, 0, 59, 0, 61, 0,
0, 0, 0, 0, 67, 0, 0, 0, 71, 0, 73, 0, 0, 0, 0, 0, 79, 0, 0, 0, 83, 0, 0, 0,
0, 0, 89, 0, 0, 0, 0, 0, 0, 0, 97, 0, 0, 0]

All done! Final list of primes is:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
73, 79, 83, 89, 97]
```

Runtime Analysis

How fast is your sieve of Eratosthenes function?

How would you describe its runtime as a function of `nMax` ?

How long would it take to list all primes up to one billion?

Can you think of any ways to optimize your algorithm?

In [0]:

In [0]:

In [0]:

Patterns in the Primes

Now that you can quickly make a big list of prime numbers, what patterns do you notice? You might consider digits within the primes, the frequency with which primes occur, clusters of primes, or anything else that occurs to you.

In [0]:

In [0]:

In [0]: