Kernel: SageMath 10.4

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(n)` that accepts a positive integer `n` and determines whether it is prime. Here is one way to do this.

In [0]:
```
def isPrime(num):
    # loop over possible divisors
    for i in range(2, floor(sqrt(num))+1):  # NOTE: earlier version as
missing the +1 here!
        if num % i == 0:
            return False

    # if we get here, then we didn't find a divisor
    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 [1]:
```
# here is one way to make a list
list(range(2,10))
```

Out[1]: [2, 3, 4, 5, 6, 7, 8, 9]

In [2]:
```
# you can also use a list comprehension
[i for i in range(2,10)]
```

Out[2]: [2, 3, 4, 5, 6, 7, 8, 9]

In [3]:
```
# Sieve of Eratosthenes from the end of class
# (not quite finished)
def sieveEratos(nMax):
    # initialize a list of numbers
    nums = list(range(2, nMax+1))

    # initalize index
    i = 0

    # main loop
    while nums[i] <= sqrt(nMax):
        # check whether we have reached the next nonzero number
        if nums[i] > 0:
            # replace all multiples of p=nums[i] with zero
            j = i + nums[i] # first position to replace with zero
            while j < len(nums):
                nums[j] = 0
                j += nums[i]  # increment j by p=nums[i]
        # increment i
        i += 1

    return nums
```

In [4]:
```
sieveEratos(20)
```

Out[4]:
```
[2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17, 0, 19, 0]
```

In [6]:
```
print(sieveEratos(100))
```

Out[6]:
```
[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]
```

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]: