Kernel: SageMath 10.4

Finding Primes

MATH 242 Modern Computational Mathematics

Primality Testing

Write a function `isPrime(n)` that accepts a positive integer `n` and determines whether it is prime. Your function should return `True` if `n` is prime and `False` otherwise.

In [4]:
```
n(sqrt(5))
```

Out[4]: 2.23606797749979

In [5]:
```
int(n(sqrt(5)))
```

Out[5]: 2

In [13]:
```
floor(sqrt(5))
```

Out[13]: 2

In [7]:
```
ceil(sqrt(5))
```

Out[7]: 3

In [3]:
```
def isPrime(num):
    # loop over possible divisors
    for i in range(2, floor(sqrt(num)+1)):
        print(i)
        if num % i == 0:
            return False

    # if we get here, then we didn't find a divisor
    return True
```

In [4]:
```
print(isPrime(25))
```

Out[4]:
```
2
3
4
5
False
```

In [14]:
```
isPrime(3454241)
```

Out[14]: False

Listing Primes

Use your function `isPrime` to make a list of all primes up to some number N.

How fast is this?

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

Can you think of a faster way to list the primes?

In [16]:
```
primeList = []  # empty list in Python
for num in range(2,100):
    if isPrime(num):
        primeList.append(num)

print(primeList)
```

Out[16]:
```
[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]
```

In [18]:
```
def listPrimes(nMax):
    primeList = []  # empty list in Python
    for num in range(2, nMax):
        if isPrime(num):
            primeList.append(num)

    return primeList
```

In [24]:
```
%time myPrimeList= listPrimes(10000)
```

Out[24]:
```
CPU times: user 1.33 s, sys: 0 ns, total: 1.33 s
Wall time: 1.41 s
```

In [0]:
```

```