Prime Number Generator with Python

(** With help of CMU 15-110 Class Material)

Prime Numbers

- An integer is "prime" if it is not divisible by any smaller integers except 1.

- 10 is **not** prime because $10 = 2 \times 5$
- 11 **is** prime
- 12 is **not** prime because $12 = 2 \times 6 = 2 \times 2 \times 3$
- 13 **is** prime
- 15 is **not** prime because $15 = 3 \times 5$

Testing Divisibility in Python

- x is "divisible by" y if the remainder is 0 when we divide x by y
- 15 is divisible by 3 and 5, but not by 2:

```
>>> 15 % 3

0
>>> 15 % 5

0
>> 15 % 2

1
```

IsPrime(): dumb version

```
def IsPrime_dumb(n):
    if (n < 2):
        return False
    for factor in range(2, n):
        if (n % factor == 0):        # 모든숫자 n에 대해서 n번의 module 계산필요
            return False
    return True


for i in range(1,100):
     if IsPrime_dumb(i):
        print(i)
```

SIEVE OF ERATOSTHENES

A 2000 year old algorithm (procedure) for generating a table of prime numbers.

2, 3, 5, 7, 11, 13, 17, 23, 29, 31, ...

What Is a "Sieve" or "Sifter"?

Separates stuff you want from stuff you don't:

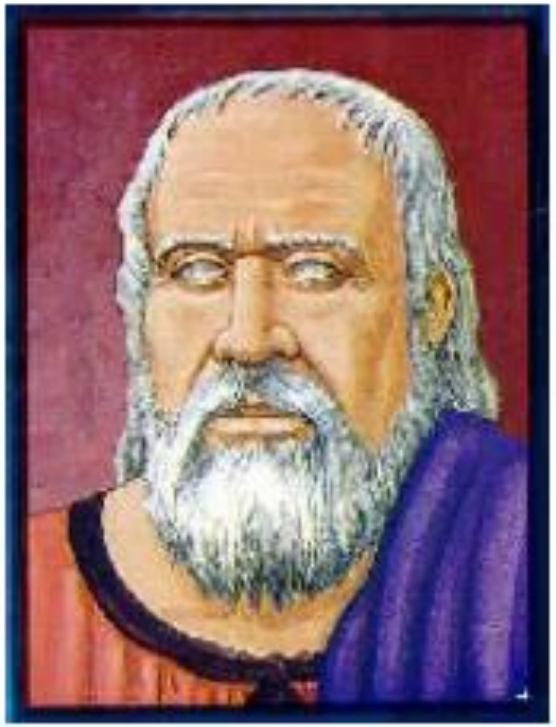
We want to separate prime numbers.

The Sieve of Eratosthenes

Start with a table of integers from 2 to N.

Cross out all the entries that are divisible by the primes known so far.

The first value remaining is the *next* prime.



Finding Primes Between 2 and 50

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

2 is the first prime

Finding Primes Between 2 and 50

	2	**3**	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Filter out everything divisible by 2.

Now we see that 3 is the next prime.

Finding Primes Between 2 and 50

	2	**3**	4	**5**	6	**7**	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Filter out everything divisible by 5.

Now we see that 7 is the next prime.

Finding Primes Between 2 and 50

	2	**3**	4	**5**	6	**7**	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Filter out everything divisible by 7.

Now we see that 11 is the next prime.

Finding Primes Between 2 and 50

	2	**3**	4	**5**	6	**7**	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Since 11 x 11 > 50, all remaining numbers must be primes. Why?

An Algorithm for Sieve of Eratosthenes

Input: A number *n*:

1. Create a list *numlist* with every integer from 2 to n, in order. (Assume $n > 1$.)

2. Create an empty list *primes*.

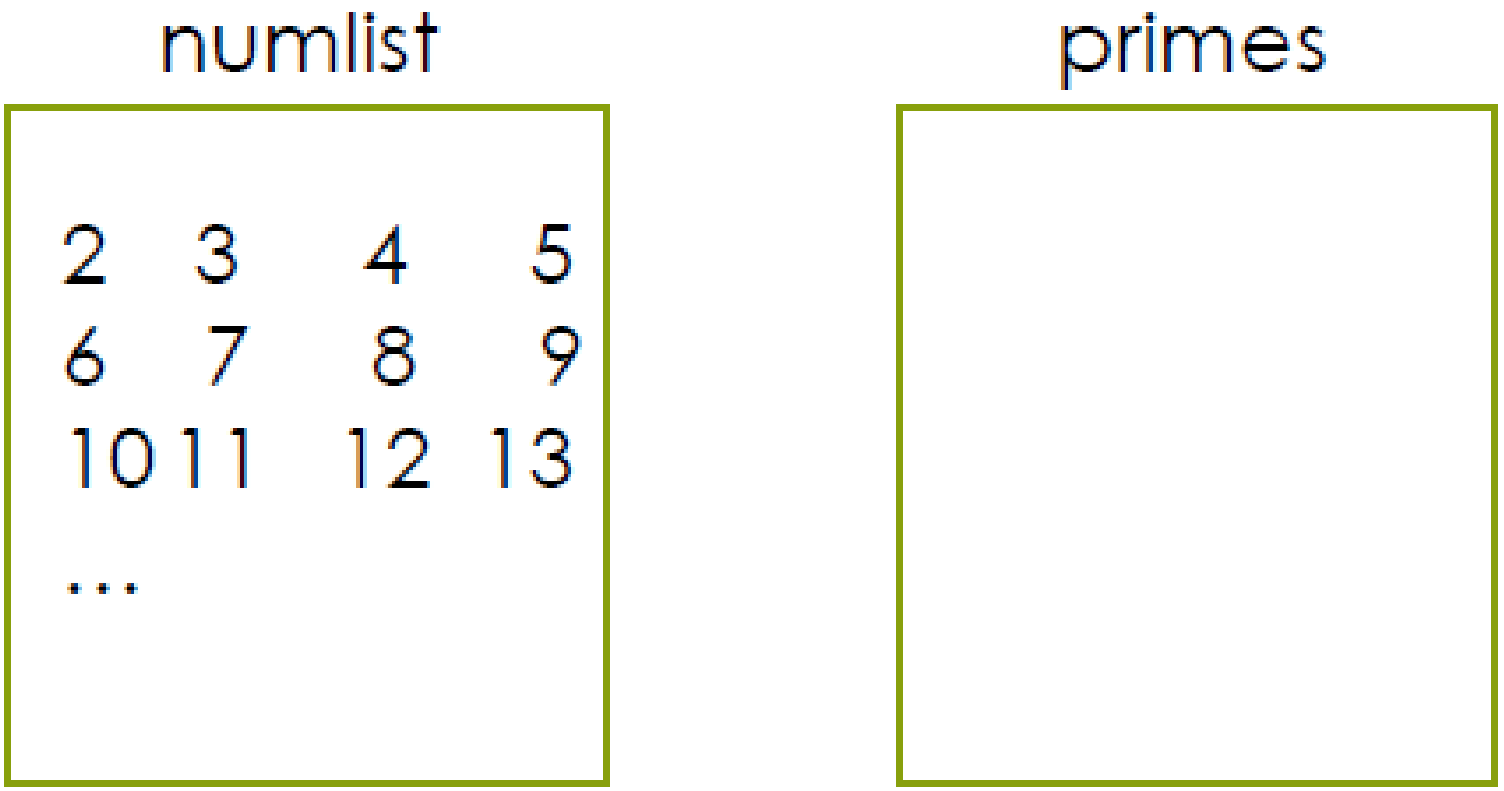
3. For each element in *numlist*

   a. If element is not marked, copy it to the end of *primes*.

   b. Mark every number that is a multiple of the most recently discovered prime number.

Output: The list of all prime numbers less than or equal to *n*

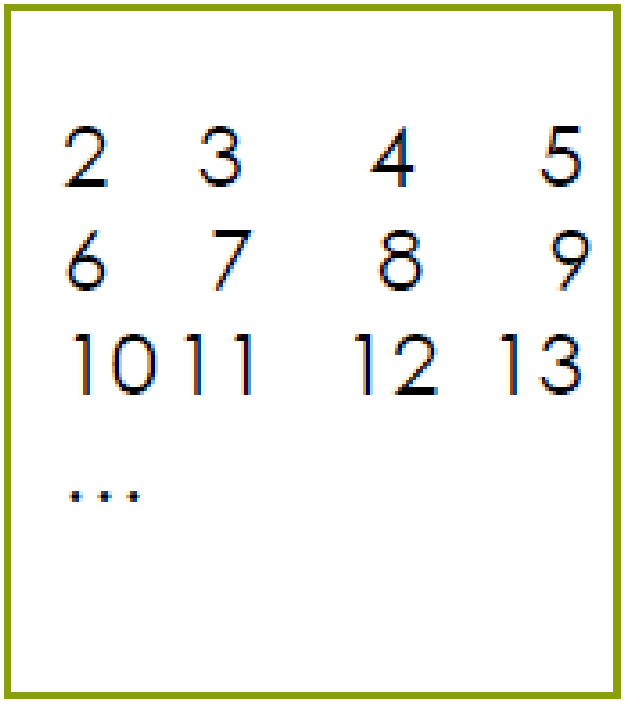
Automating the Sieve



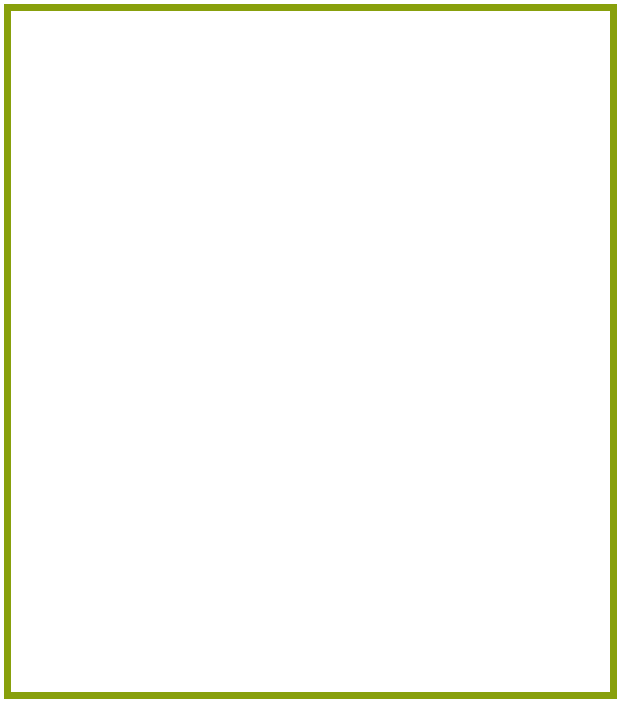
Use *two* lists: candidates, and confirmed primes.

Steps 1 and 2

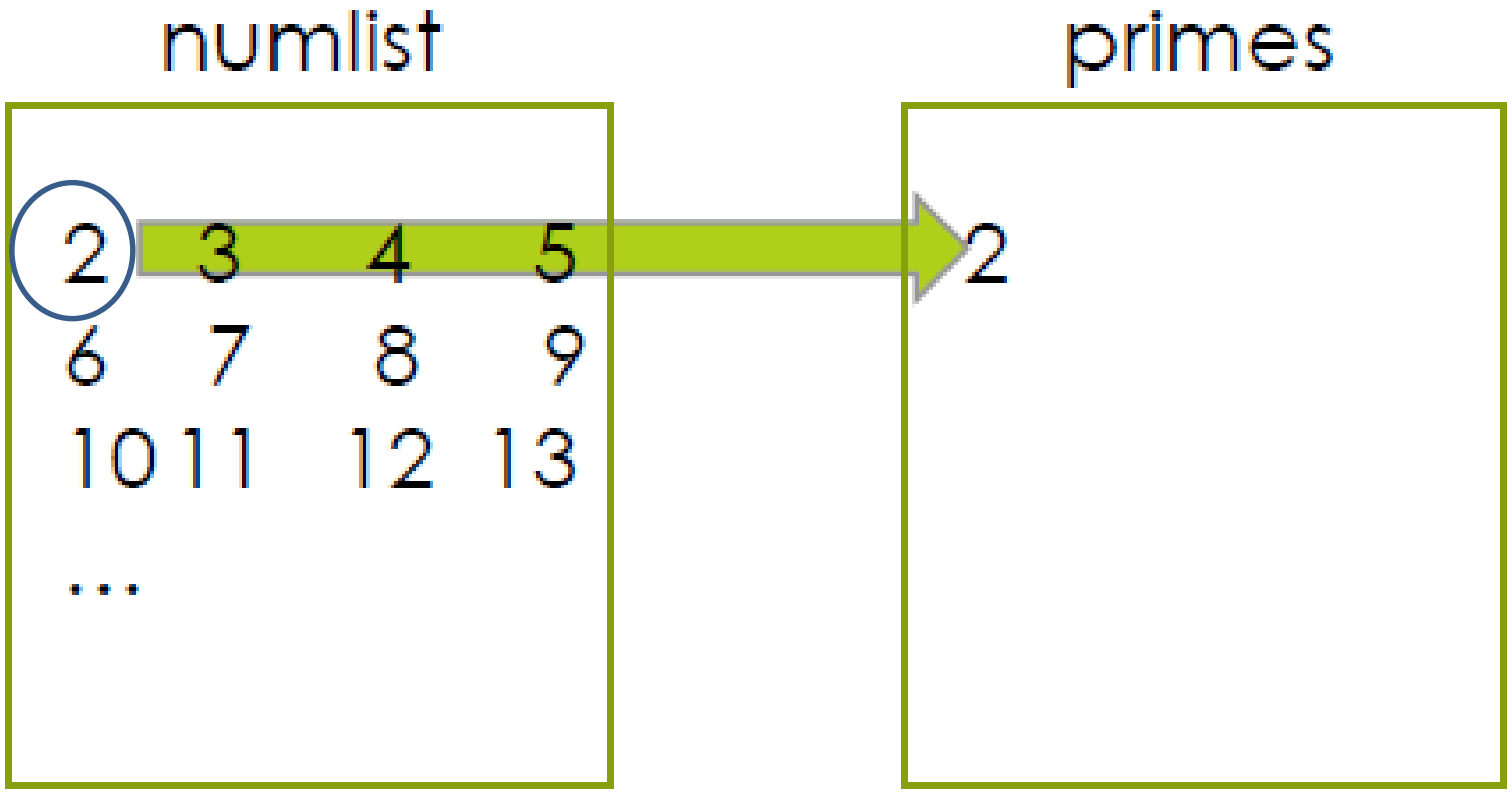
numlist



primes



Step 3a



Append the current number in numlist to the end of primes.

Step 3b

numlist

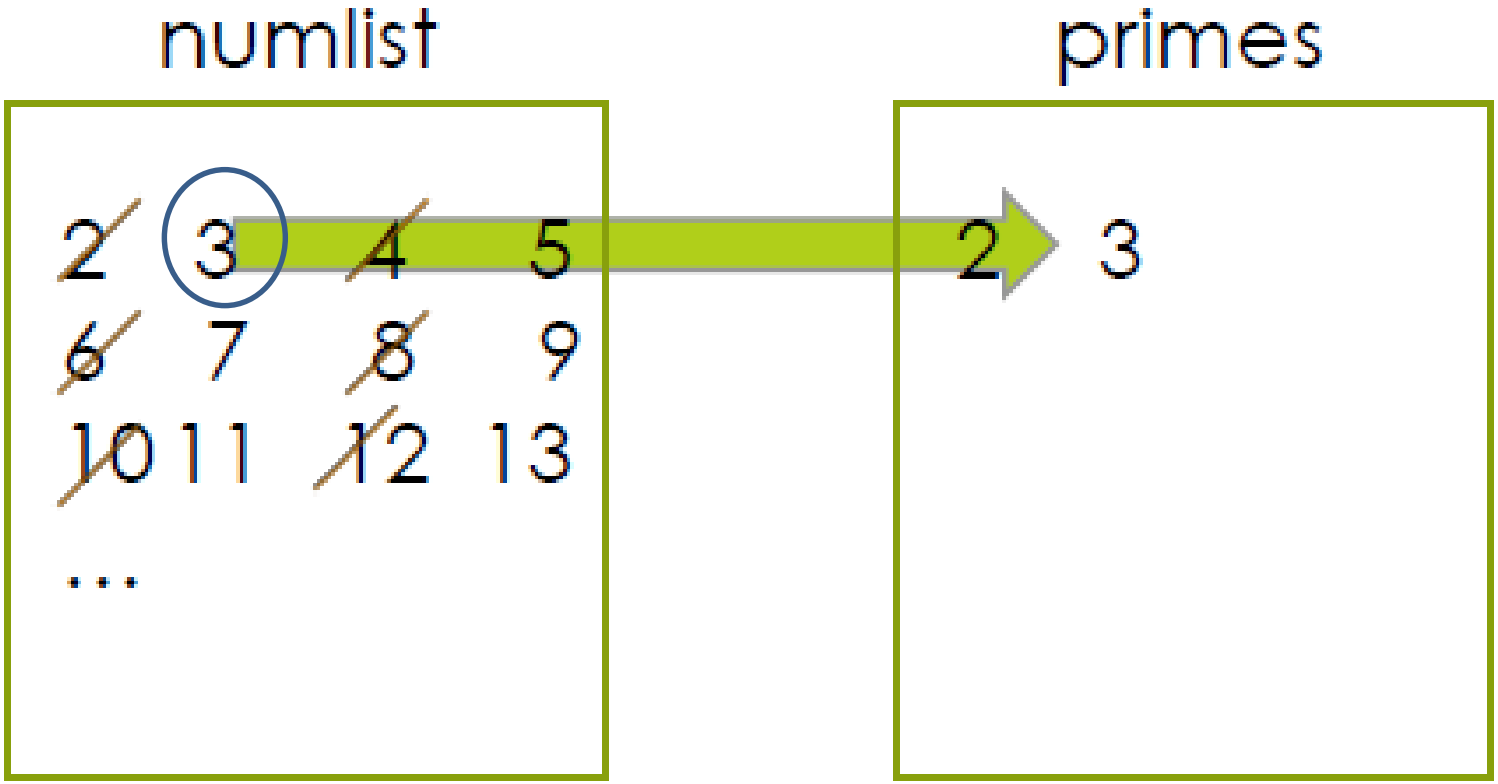
~~2~~ 3 ~~4~~ 5
~~6~~ 7 ~~8~~ 9
~~10~~ 11 ~~12~~ 13
...

primes

2

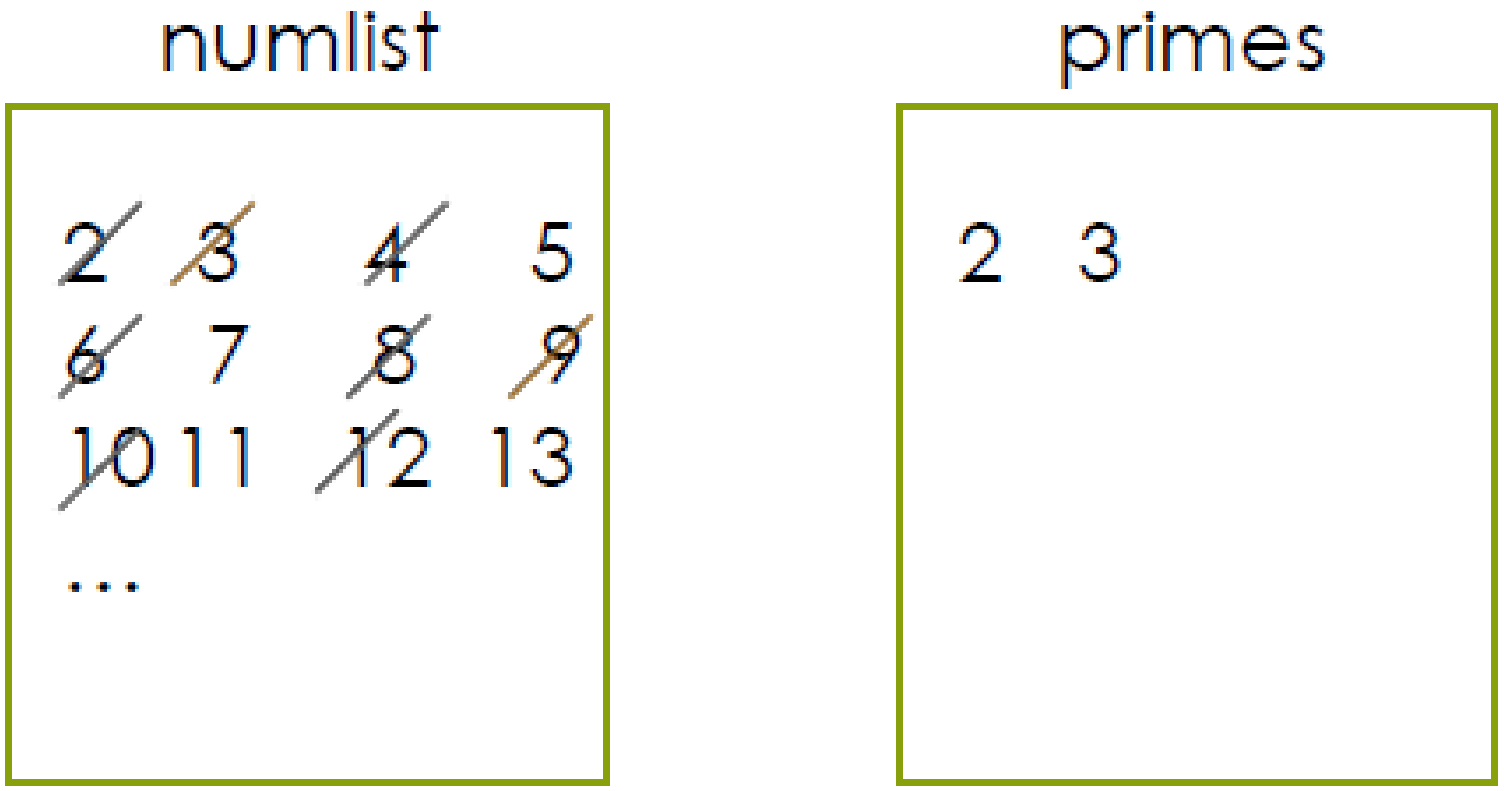
Cross out all the multiples of the last number in primes.

Iterations



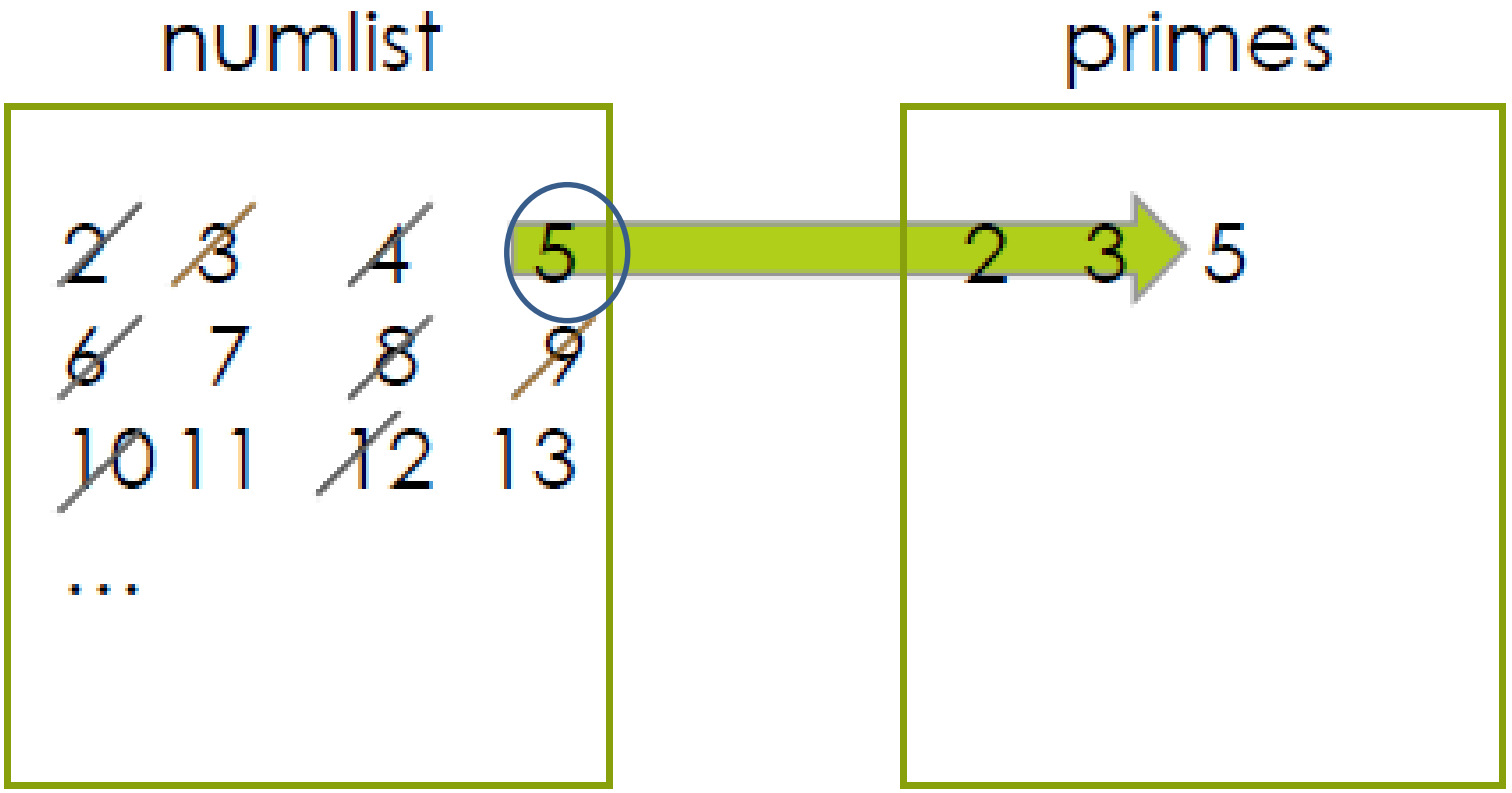
Append the current number in numlist to the end of primes.

Iterations



Cross out all the multiples of the last number in primes.

Iterations



Append the current number in numlist to the end of primes.

Iterations

numlist

~~2~~ ~~3~~ ~~4~~ ~~5~~
~~6~~ 7 ~~8~~ ~~9~~
~~10~~ 11 ~~12~~ 13
...

primes

2 3 5

Cross out all the multiples of the last number in primes.

An Algorithm for Sieve of Eratosthenes

Input: A number *n*:

1. Create a list *numlist* with every integer from 2 to n, in order. (Assume $n > 1$.)

2. Create an empty list *primes*.

3. For each element in *numlist*

   a. If element is not marked, copy it to the end of *primes*.

   b. Mark every number that is a multiple of the most recently discovered prime number.

Output: The list of all prime numbers less than or equal to *n*

Implementation Decisions

- How to implement *numlist* and *primes?*
  - For *numlist* we will use a list in which crossed out elements are marked with the special value None. For example,

```
[None, 3, None, 5, None, 7, None]
```

- Use a helper function for step 3.b. We will call it sift.

Relational Operators

- If we want to compare two integers to determine their relationship, we can use these relational operators:

<	less than	<=	less than or equal to
>	greater than	>=	greater than or equal to
==	equal to	!=	not equal to

- We can also write compound expressions using the Boolean operators and and or.

```
x >= 1 and x <= 1
```

Sifting: Removing Multiples of a Number

```
def sift(lst,k):
    # marks multiples of k with None
    i = 0
    while i < len(lst):
        if (lst[i] != None) and lst[i] % k == 0:
            lst[i] = None
        i = i + 1
    return lst
```

Filters out the multiples of the number k from list by marking them with the special value None (greyed out ones).

Sifting: Removing Multiples of a Number (Alternative version)

```
def sift2(lst,k):
    i = 0
    while i < len(lst):
        if lst[i] % k == 0:
            lst.remove(lst[i])
        else:
            i = i + 1
    return lst
```

Filters out the multiples of the number k from list by modifying the list. Be careful in handling indices.

A Working Sieve

```
def sieve(n):
    numlist = list(range(2,n+1))
    primes = []
    for i in range(0,len(numlist)):
        if numlist[i] != None:
            primes.append(numlist[i])
            sift(numlist,numlist[i])
    return primes
```

Use the first version of sift in this function, which does the filtering using Nones.

We could have used `primes[len(primes)-1]` instead.

Helper function that we defined before

Observation for a Better Sieve

We stopped at 11 because all the remaining entries must be prime since $11 \times 11 > 50$.

2	**3**	4	**5**	6	**7**	8	9	10	
11	12	**13**	14	15	16	**17**	18	**19**	20
21	22	**23**	24	25	26	27	28	**29**	30
31	32	33	34	35	36	**37**	38	39	40
41	42	**43**	44	45	46	**47**	48	49	50

Eratosthenes는 본인의 sieve algorithm에서 n까지 확인을 할 필요가 없이 square_root(n)까지만 확인하면 남아있는 숫자들은 전부 prime이라는것을 증명

A Better Sieve

```
def sieve(n):
    numlist = list(range(2, n + 1))
    primes = []
    i = 0 # index 0 contains number 2
    while (i+2) <= math.sqrt(n):
        if numlist[i] != None:
            primes.append(numlist[i])
            sift(numlist, numlist[i])
            i = i + 1
    return primes + numlist
```

Algorithm-Inspired Sculpture

The Sieve of Eratosthenes, 1999 sculpture by Mark di Suvero. Displayed at Stanford University.

IsPrime(): dumb version

```
def IsPrime_dumb(n):
    if (n < 2):
        return False
    for factor in range(2, n):
        if (n % factor == 0):        # 모든숫자 n에 대해서 n번의 module 계산필요
            return False
    return True


for i in range(1,100):
     if IsPrime_dumb(i):
        print(i)
```

IsPrime(): better version

```
def IsPrime_better(n):
    if (n < 2):
        return False
    if (n == 2):
        return True
    if (n % 2 == 0):
        return False
    for factor in range(3, n, 2):   # 2의 배수는 다 skip하므로 효과적!
        if (n % factor == 0):
            return False
    return True


for i in range(1,100):
     if IsPrime_better(i):
        print(i)
```

IsPrime(): best version

```
def IsPrime_best(n):
    if (n < 2):
        return False
    if (n == 2):
        return True
    if (n % 2 == 0):
        return False
    maxFactor = round(n**0.5)
    for factor in range(3, maxFactor+1, 2):
        if (n % factor == 0):          # Module적용횟수가 획기적으로 줄어듬
            return False
    return True



for i in range(1,100):
     if IsPrime_best(i):
        print(i)
```