Python Practices

Compute molecular weight

```
# Compute molecular weight
# Here are basic weights
#    carbon(c): 12.011
#    hydrogen(H): 1.0079
#    oxygen(O): 15.9994
#
# use round(46.0688, 2) ==> 46.07

def molecular_wight():
    print("Please enter the number of each atom")
    C = input("carbon: ")
    H = input("hydrogen: ")
    O = input("oxygen: ")
    W = C*12.011 + H*1.0079 + O*15.9994
    print ("The molecular weight of C",C,"H",H,"O",O,"is: ",  round(W,2) )

def molecular_wight_correct():
    print("Please enter the number of each atom")
    C = eval(input("carbon: "))
    H = eval(input("hydrogen: "))
    O = eval(input("oxygen: "))
    W = C*12.011 + H*1.0079 + O*15.9994
    print ("The molecular weight of C",C,"H",H,"O",O,"is: ", round(W,2)
```

Palindrome Checker [1/2]

Palindrome이란 철자를 거꾸로 놓아도 원래와 같은 글귀를 말합니다.
부호와 빈칸을 제외하고 대소문자 구분없이 알파벳이 대칭을 이루는 문장
예를 들어, 'abcdcba'는 뒤집어도 똑같으므로 palindrome이라고
할 수 있습니다
#
조금 더 복잡한 예로
'Are we not drawn onward, we few, drawn onward to new era'
'Do geese see God'
'Dennis and Edna sinned'

Palindrome Checker [2/2]

```
def pallindrome_decider():
    Pallindrome_candidate = input("Type your pallindrome candiate: ")
    print ("Here is your pallindrome candiate:", Pallindrome_candidate)
    Pallindrome_candidate = Pallindrome_candidate.lower()
    print ("After lowering characters ==> ", Pallindrome_candidate)
    #
    isPallindrome_candidate = True
    p1 = 0
    p2 = len(Pallindrome_candidate)-1
    #
    while isPallindrome_candidate and p1 < p2:
        if Pallindrome_candidate[p1].isalpha():
            if Pallindrome_candidate[p2].isalpha():
                if Pallindrome_candidate[p1]==Pallindrome_candidate[p2]:
                    p1 = p1+1
                    p2 = p2-1
                else:
                    isPallindrome_candidate = False
            else:  # if not alphabet ==> move p2 to left
                p2 = p2-1
        else: # if not alphabet ==> move p1 to right
            p1 = p1+1
    #
    if isPallindrome_candidate:
        print ("Yes, your pallindrome candiate", Pallindrome_candidate,  "is a real pallindrome!")
    else:
        print ("No, your pallindrome candiate", Pallindrome_candidate,  "is not a real pallindrome!")
```

Happy Birth Day Song

```
# sing("Kang Min")
#
# Happy birthday to you!
# Happy birthday to you!
# Happy birthday, dear Kang Min.
# Happy birthday to you!

def sing(name):
    print("Happy birthday to you! ")
    print("Happy birthday to you! ")
    print("Happy birthday, dear %s "    %name)
    print("Happy birthday to you! ")
```

Euclean Distance Computation

```
# Euclidean Distance란 직교 좌표계에서 두 점의 거리를 나타냅니다.
# 예를 들어, 2차원 평면에서 두 점 (x1, y1), (x2, y2)의 거리는
# Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2)) 로 계산
# 이와 같이 임의의 차원에서의 거리를 구하는 함수를 구현해보세요.

# 함수가 받을 parameter는 총 3개로,
# 첫번째 parameter n은 차원 수, parameter p1, parameter p2는 길이 가 n인 리스트

import math

def eucDist(n, p1, p2):
    distance = 0
    for i in range(n):
        distance = distance + (p2[i]-p1[i])**2
    #
    return math.sqrt(distance)
```

Temperature Warning

```
# input은  '20.3F'  '-10C'   '32.5C'  같은방식의 string으로 입력
# output은
#     물의 끓는 점 이상일 경우 Be careful!
#     물이 어는 점 이하일 경우 Don't get frozen!
#     물 밀도가 가장 높은 점(섭씨 3도에서 5도 사이로 가정)일 경우 You will be fine!

def FtoC(F):
    C = (F-32)*5/9
    return C

def TempOK(C):
    if C >= 100:
        print ("Be careful!")
    if C <= 0:
        print ("Don't get frozen!")
    if C >= 3 and C <= 5:
        print ("You will be fine")

def WeatherMessage():
    temp = input("Type your temperature in string format:")
    if temp[-1] == "C":
        Centi = float(temp[:-1])
        TempOK(Centi)
    elif temp[-1] == "F":
        Fahren = float(temp[:-1])
        TempOK(FtoC(Fahren))
    else:
        print("Pardon?")
```

Leap Year Checker

```
# 윤년은 1년이 366일로 이루어져 있는 해인데
# 규칙은 다음 Wolfram.com에서 주어진 정의와 같습니다.
# Leap years were therefore 45 BC, 42 BC, 39 BC, 36 BC, 33 BC,
# 30 BC, 27 BC, 24 BC, 21 BC, 18 BC, 15 BC, 12 BC, 9 BC, 8 AD,
# 12 AD, and every fourth year thereafter (Tøndering), until the
# Gregorian calendar was introduced (resulting in skipping three out
# of every four centuries).


def yun_year_checker():
    target_year = input("Please type your year:")
    yun_year = False
    #
    if target_year in [-45, -42, -39, -33, -30, -27, -24, -21, -18, -15, -12, -9, 8, 12]:
        yun_year = True
    #
    elif target_year > 12 and target_year % 4==0:
        yun_year = True
        if target_year % 100==0:
            yun_year = False
            if target_year % 400==0:
                yun_year = True
    #
    if yun_year:
        print ("Yes, the year", target_year, " is a leap year!")
    else:
        print ("No, the year", target_year, " is not a leap year!")
```

Valid Date Checker [1/2]

```
# 입력된 날짜가 유효할 경우 valid, 입력된 날짜가 유효하지 않거나 입력된 값이 날짜
# 형태가 아닐 경우 invalid을 출력합니다. 유효한 날짜는 달력 상 존재하는 날짜를
# 의미합니다. 예를 들어, -5/12/17은 기원전 5년의 12월 17일을 의미하므로 유효합니다.
# 하지만 0년은 존재하지 않습니다.


def LeapYear(y):
    year = y
    yun = False
    if year in [-45,-42,-39,-33,-30,-27,-24,-21,-18,-15,-12,-9,8,12]:
        yun = True
    elif year > 12 and year%4==0:
        yun = True
        if year%100==0:
            yun = False
            if year%400==0:
                yun = True
    return yun

def MonthDate(y, m):
    if m in [1, 3, 5, 7 ,8, 10, 12]:
        return 31
    elif m == 2:
        if LeapYear(y):
            return 29
        else:
            return 28
    else:
        return 30
```

Valid Date Checker [2/2]

```
def valid_date_checker():
    Target_Date = input("Type your date in yyyy/mm/dd string format:")
    print (“Your Target Date is:”,  Target_Date)

    try:
        date = Target_Date.split("/")
        print ("Your typed date is:", "Year", date[0], "Month", date[1], "Day", date[2])
        if int( date[0] ) ==0:
            print ("Your typed date is invalid”)
        elif int( date[1] ) in [1,2,3,4,5,6,7,8,9,10,11,12]:
            daylist = []
            for i in range( MonthDate(int(date[0]), int(date[1])) ):
                daylist.append( i+1 )
            if int(date[2]) in daylist:
                print ("Your typed date is valid!”)
            else:
                print ("Your typed date is invalid!”)
        else:
            print ("Your typed date is invalid”)

    except:
        print ("Your typed date is invalid”)
```

Prime Number Generator with Python

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

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.

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:

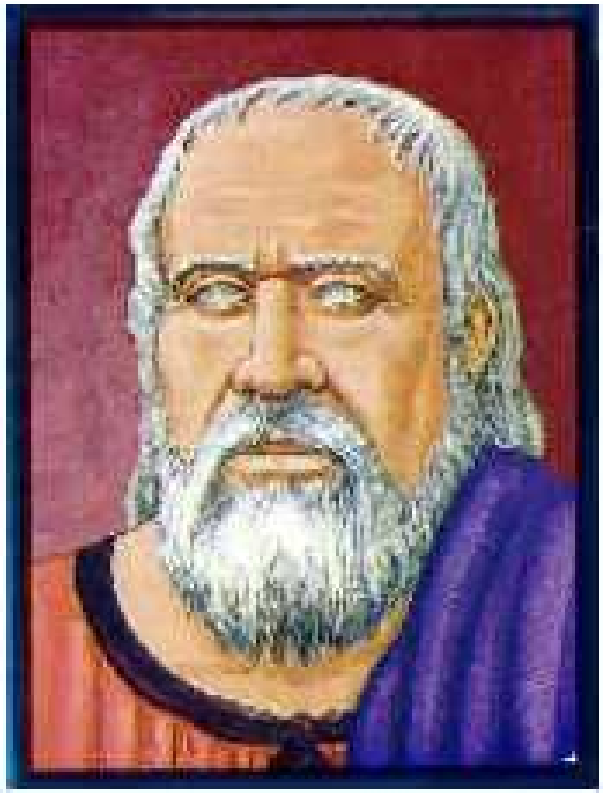
```
>>> 15 % 3

0
>>> 15 % 5

0
>> 15 % 2

1
```

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*.

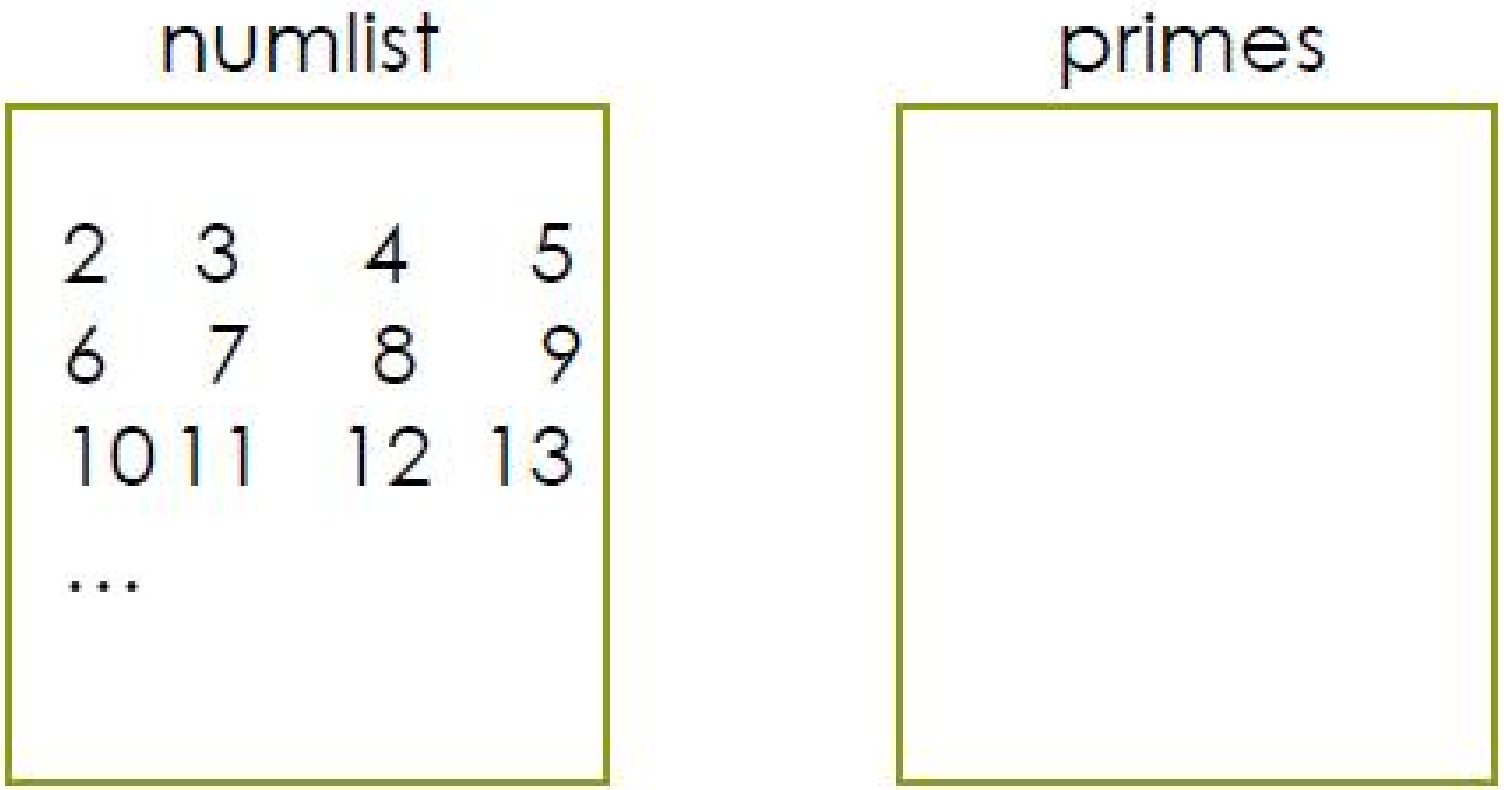
3. For each element in *numlist*

    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.

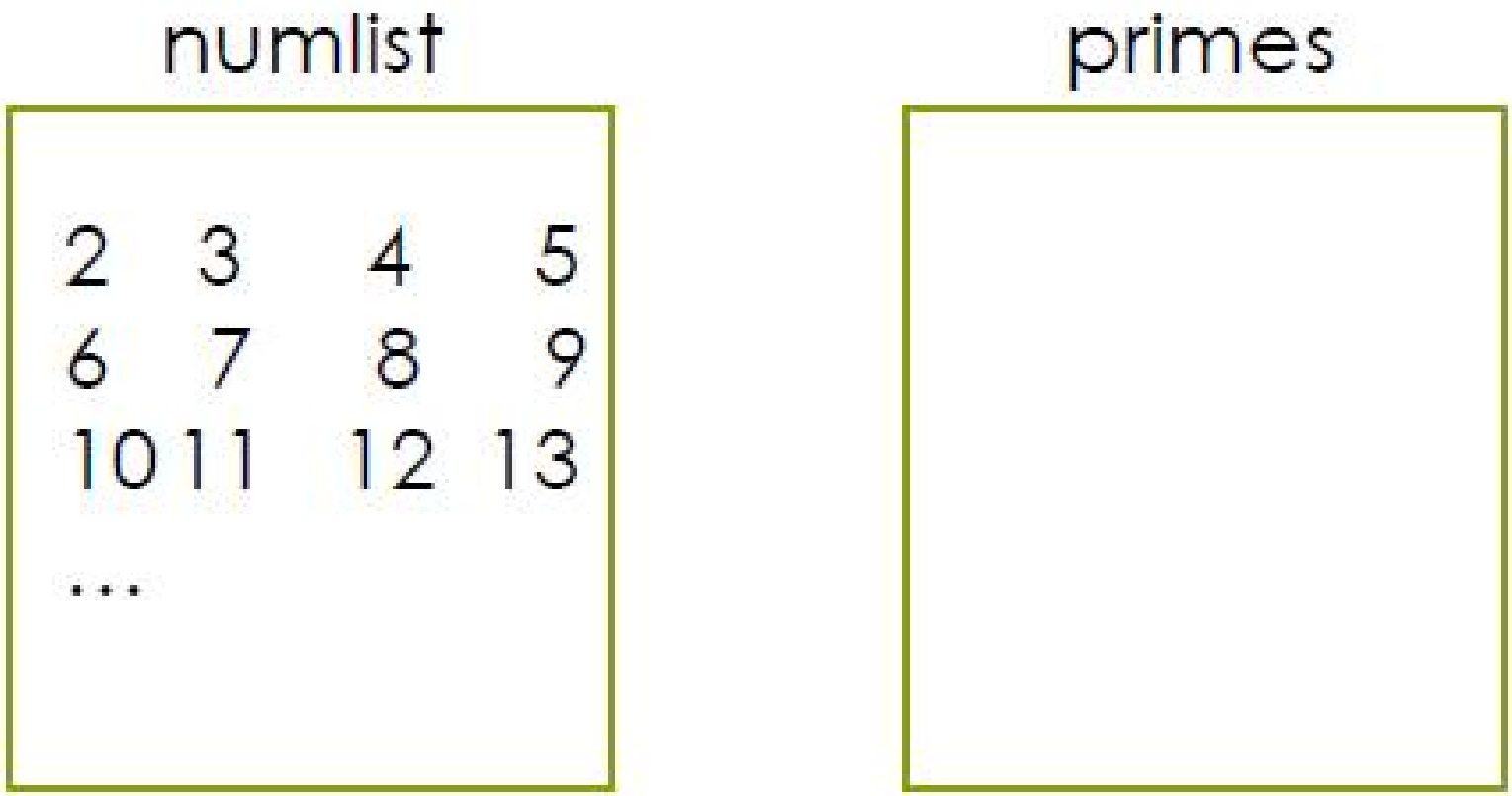
Output: The list of all prime numbers less than or equal to *n*

Automating the Sieve

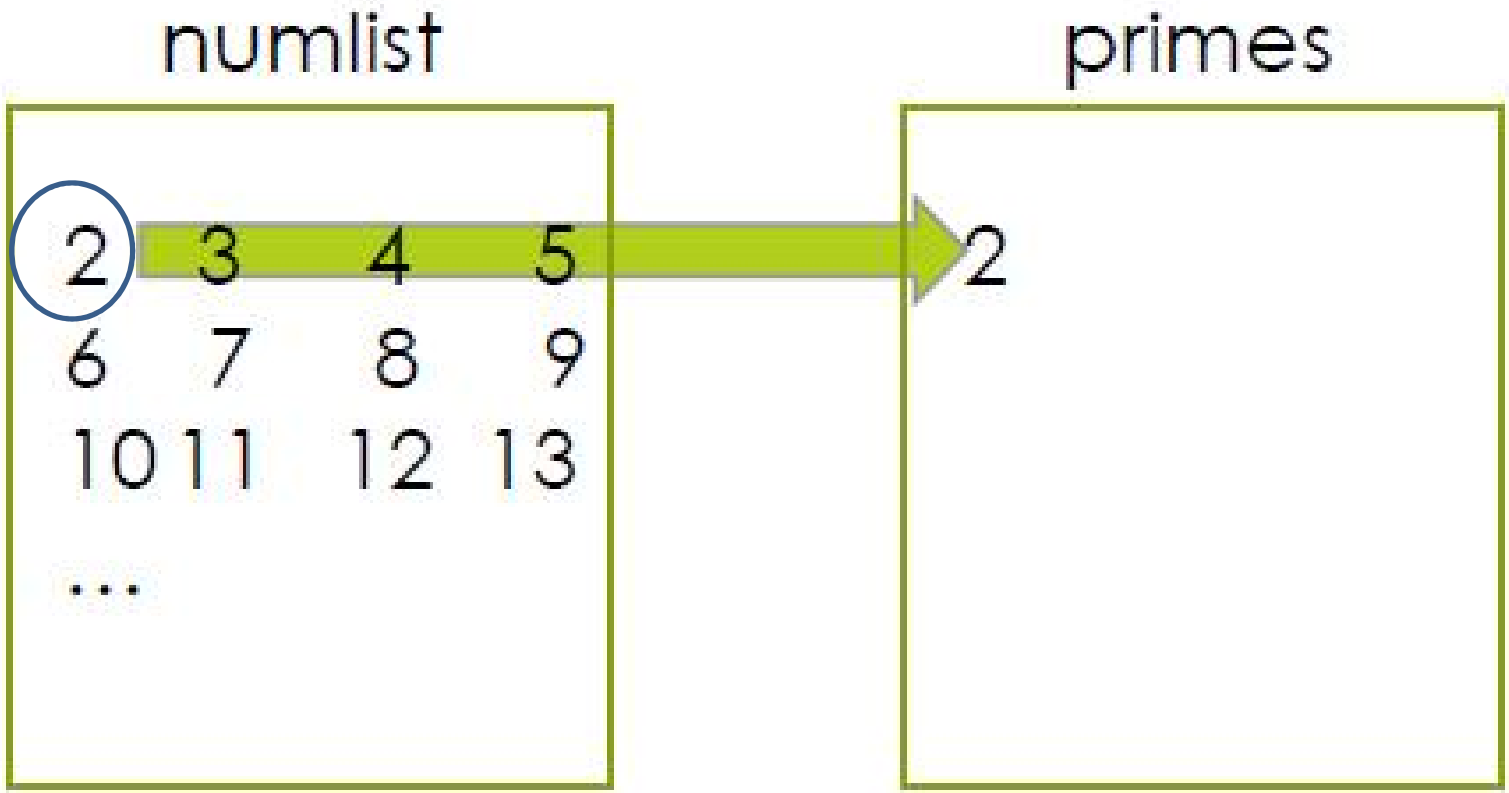


Use *two* lists: candidates, and confirmed primes.

Steps 1 and 2

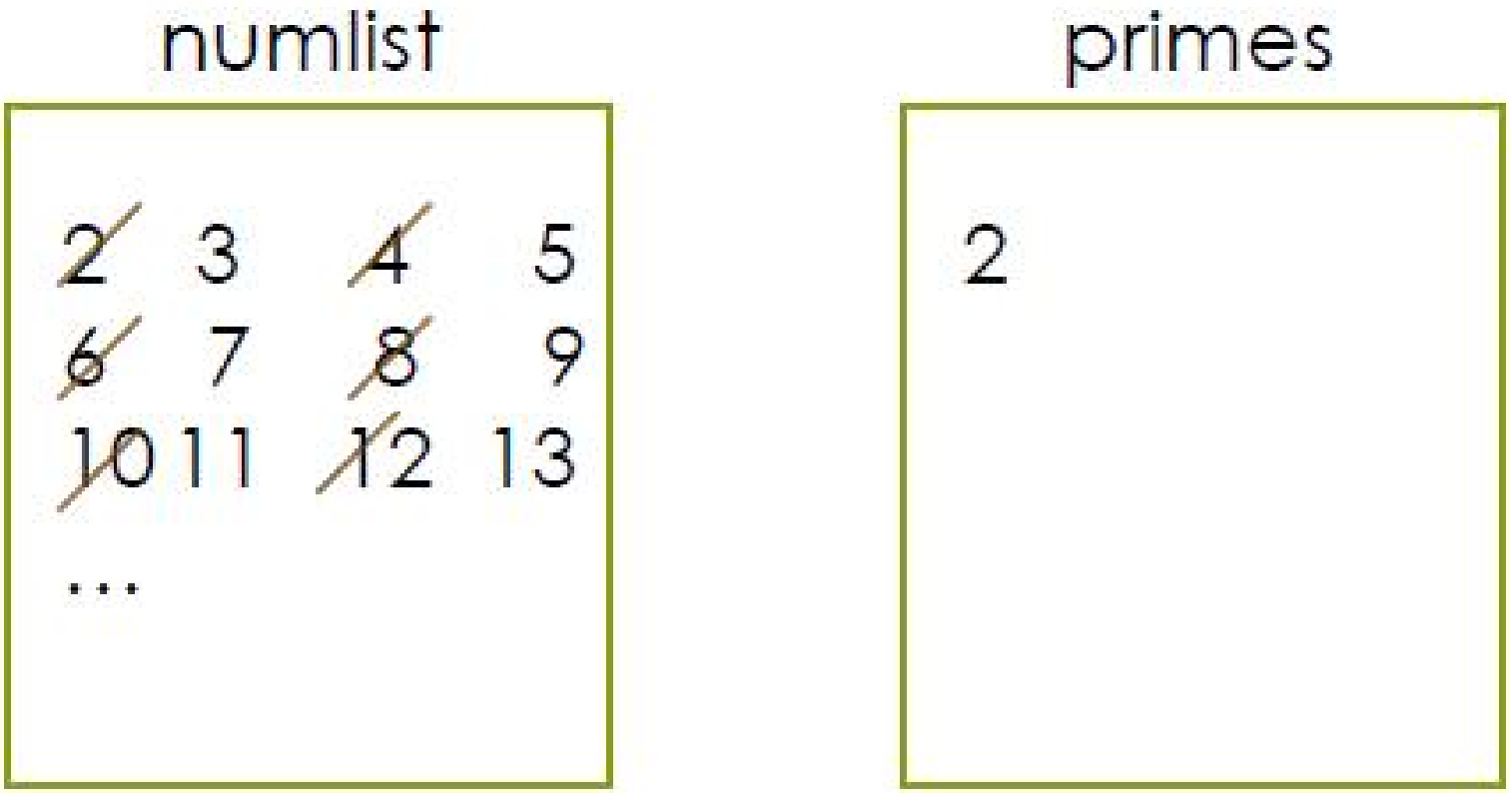


Step 3a



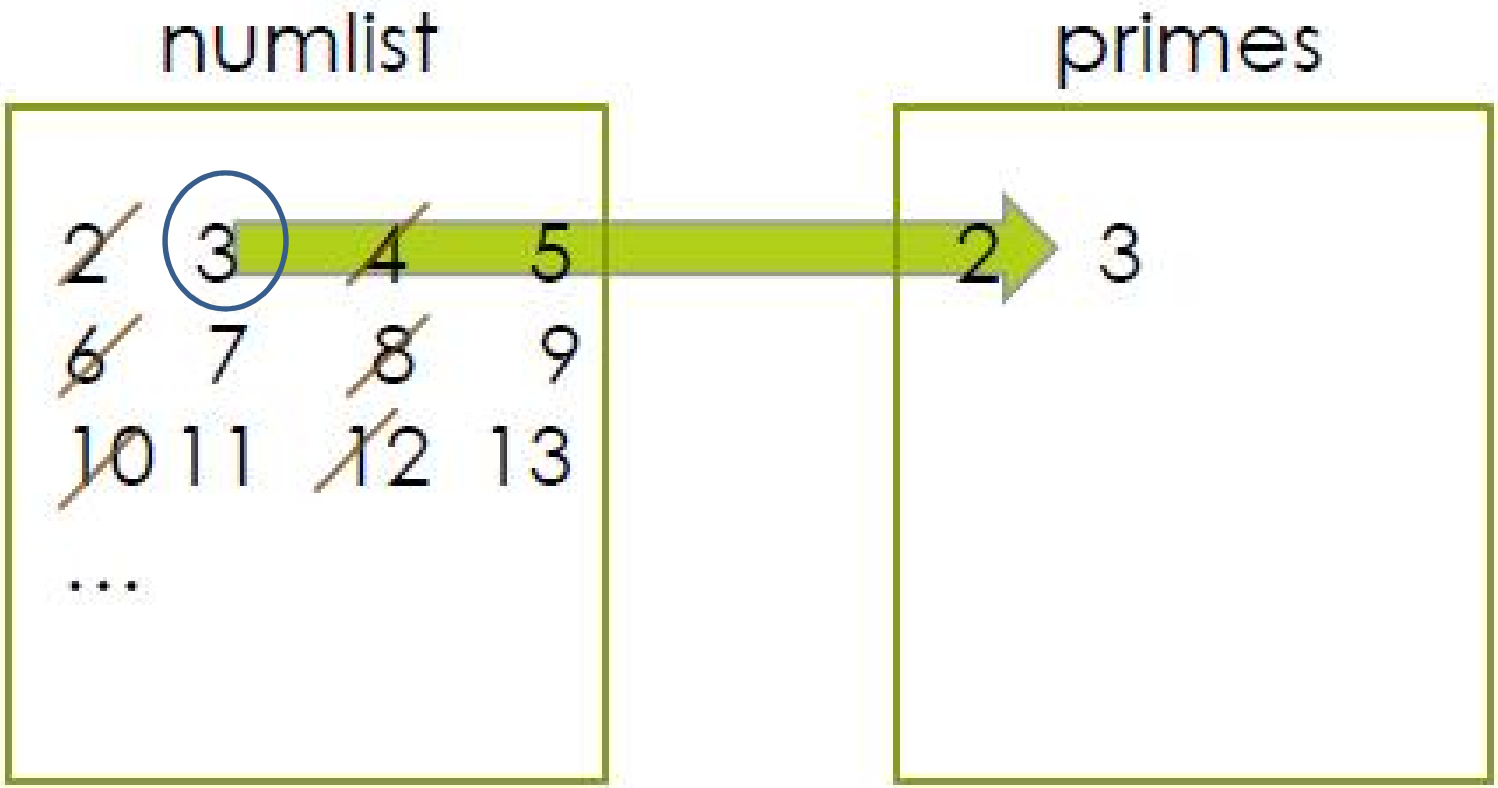
Append the current number in numlist to the end of primes.

Step 3b



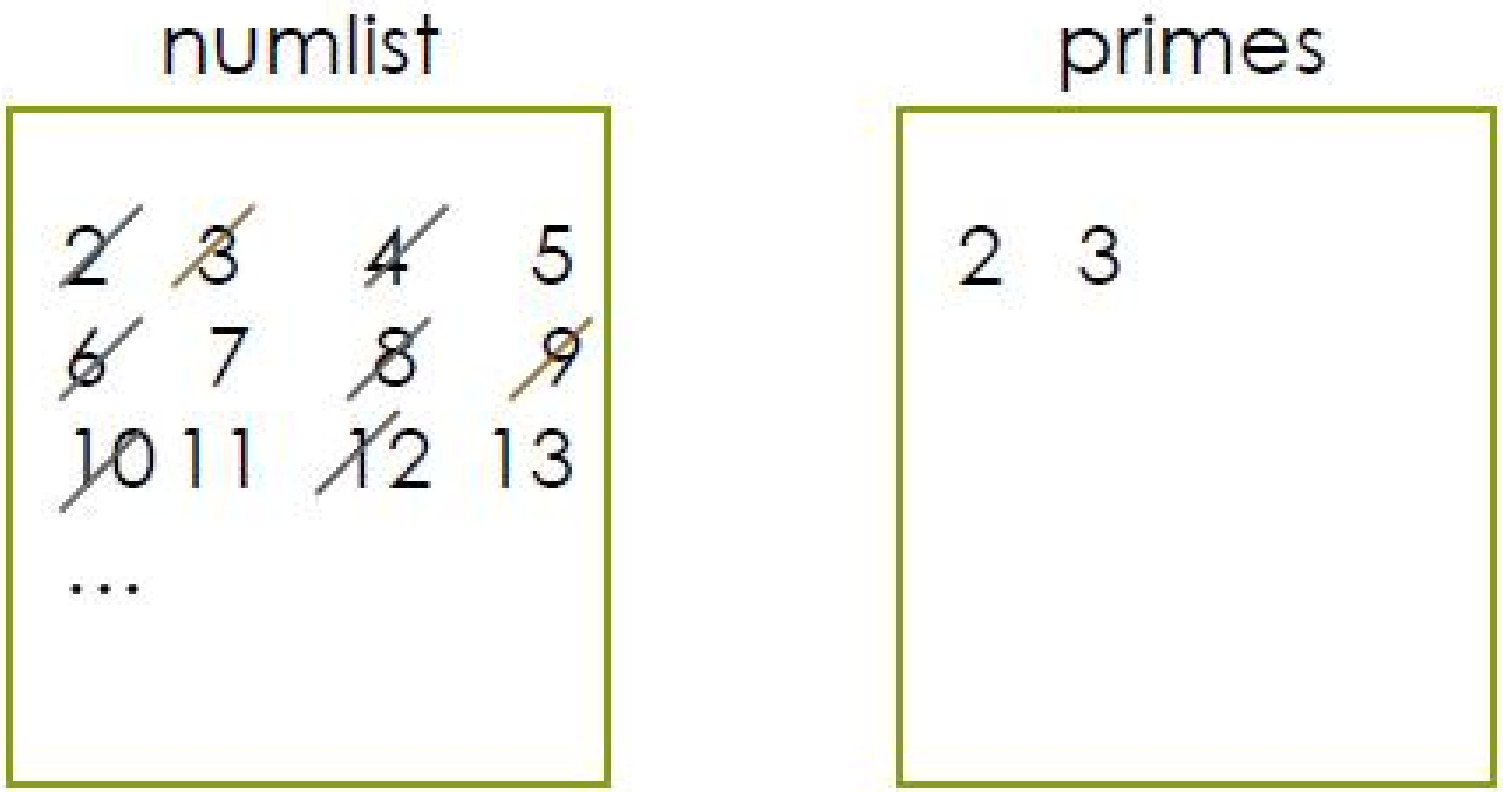
Cross out all the multiples of the last number in primes.

Iterations



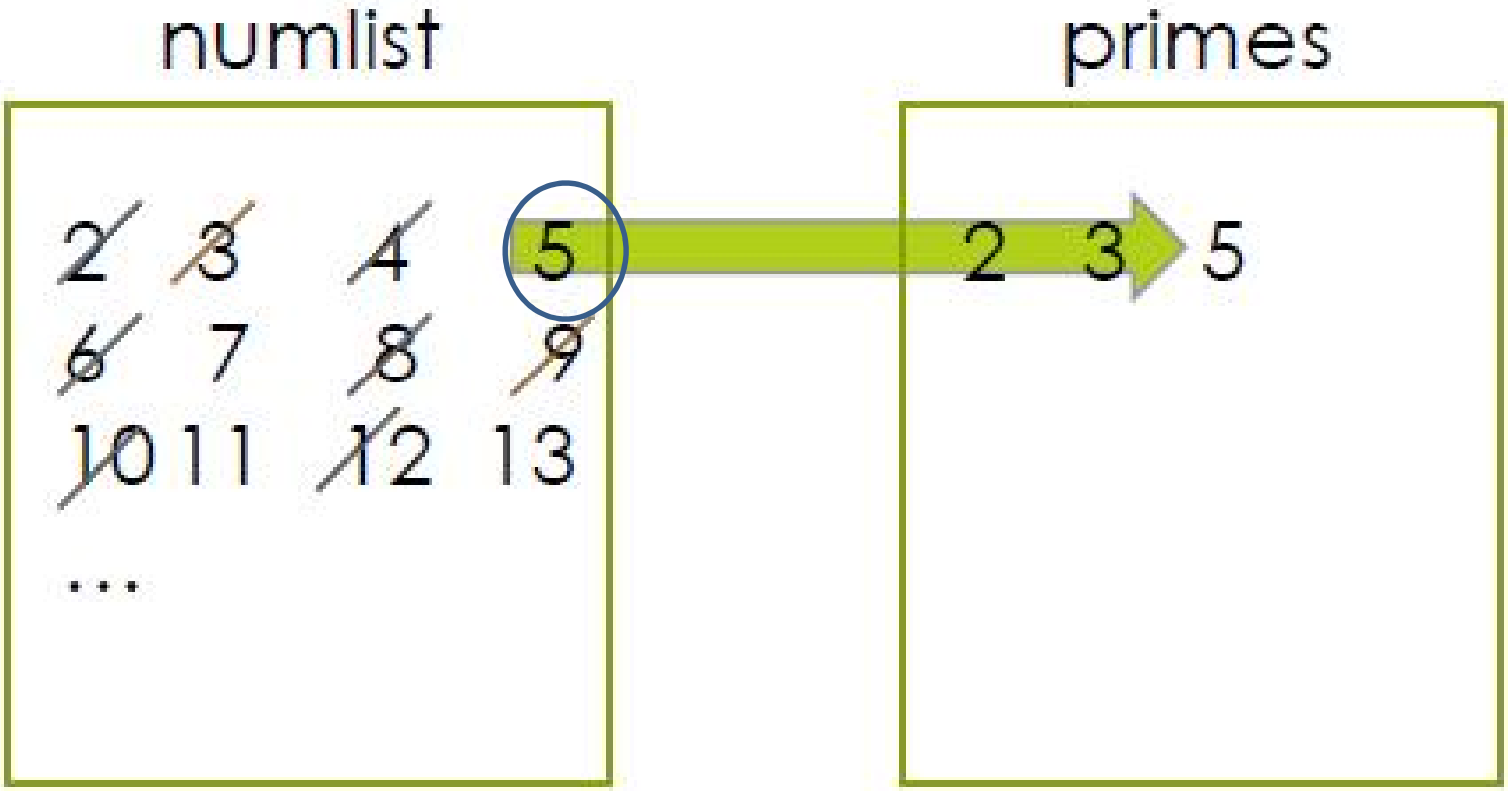
Append the current number in numlist to the end of primes.

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

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):
            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):
        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):
            return False
    return True


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