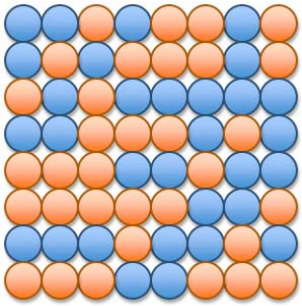


UNIT 3B
Algorithmic Thinking

Last Lecture

- Ranges, arrays
  - Methods for arrays
- Iterators
  - each, collect, select, delete_if
- Sieve of Eratosthenes: a procedure to find prime numbers
  - Relational operators (==, !=, >)
  - Logical *and* operator
  - *modulo* operator (%)

Arrays Review (1)

Examples:

a = [2, 5, 9, 8]

a[0] => 2

a[2] => 9

a. first => 2

b = [2, “anything”, 3.0]

b.last => 3.0

Arrays Review (2)

- The empty array is written []. It has length 0.
- Converting a range to an array

```
values = (1..9).to_a
=> [1, 2, 3, 4, 5, 6, 7, 8, 9]
values = Array(1..9)
=> [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Arrays Review (3)

- Appending to an array

  values = (1..9).to_a

  => [1, 2, 3, 4, 5, 6, 7, 8, 9]

  values << 10

  => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

- Concatenating two arrays

  a = [1, 2]

  b = [" red", "green"]

  a + b

  => [1, 2, "red", "green"]

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

```
scores = [78, 93, 80, 68, 100, 94, 85]
scores.length == 7          => true
scores.first > 80           => false
```

Arrays: The **delete_if** method

scores = [78, 93, 80, 68, 100, 94, 85]

scores.delete_if{ |n| n < 80 }
⇒[93, 80, 100, 94, 85]

"For each element n in scores delete n if n is less than 80."
scores
⇒[93, 80, 100, 94, 85]

Destructive method

This Lecture

- Control structures
  - Conditionals (if and if/else statements), while loops
- Sieve of Eratosthenes
  - Write a complete function

Conditionals in blocks

Example: Use an iterator and an if modifier to print the even numbers in the range

```
nums = [ 1, 2, 3, 4, 5, 6, 7, 8, 9,10]
nums.each { |x|  puts x if  x % 2 == 0}
2
4
6
8
10
=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```


If Statement

Example: Give praise if the grade is "A"

```
if grade == "A" then
  puts "Outstanding"
  puts "Keep up the good work"
end
```

General syntax:

```
if condition then
   statement list
end
```

If/else Statement

Example: Give praise if the grade is "A", and "can do better" otherwise

```
if grade == "A" then
  puts "Outstanding"
  puts "Keep up the good work"
else
  puts "Can do better"
end
```

Can use multiway if statements and nest them

If/else Statement

```
if grade == "A" then
  puts "Outstanding"
else
  if grade == "B" then
    puts "Can do better"
  else
    if grade == "C" then
      puts "Can do much better"
    else
      puts "Not much hope"
    end
  end
end
```

If/else Statement

```
if grade == "A" then
  puts "Outstanding"
elsif grade == "B" then
  puts "Can do better"
elsif grade == "C" then
  puts "Can do much better"
else
  puts "Not much hope"
end
```


While Loops

Example: Print first 10 positive integers

```
i = 1
while  i <= 10  do
  puts i
  i = i + 1
end
```

General syntax:

```
while loop condition then
   loop body
end
```

While Loop Examples

```
# Prints first 10 positive
# integers
i = 1
while  i <= 10  do
  puts i
  i = i + 1
end
```

How about the following?

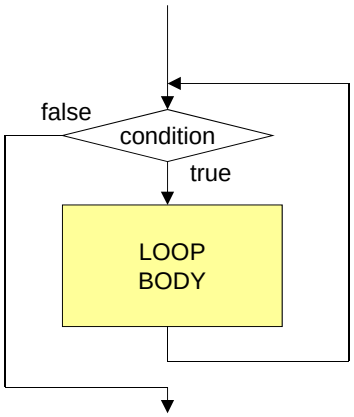
```
i = 0
while  i <= 10  do
  i = i + 1
  puts i
end
```


While vs. For Loops

```
# Prints first 10 positive
# integers
i = 1
while  i <= 10  do
  puts i
  i = i + 1
end
```

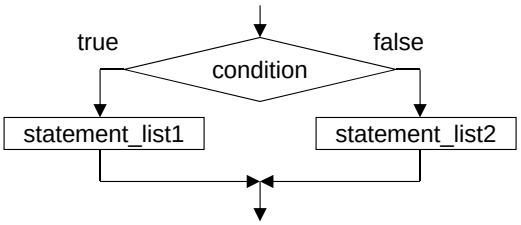
```
# Prints first 10 positive
# integers
for i in 1..10 do
    puts i
end
```

Control Flow for while Loops



If the loop condition becomes false during the loop body, the loop body still runs to completion before we exit the loop and go on with the next step.

Control Flow for if Statements



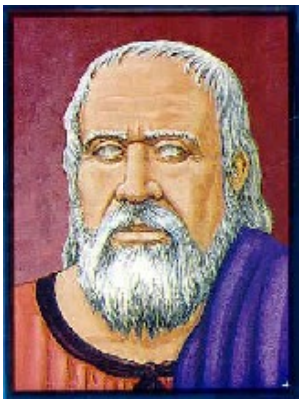
RECALL
SIEVE OF ERATOSTHENES

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 is the first prime.

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

Finding Primes Between 2 and 50

Filter out everything divisible by 2:
items.delete_if { |i| (i>2) & (i%2 == 0) }

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

Finding Primes Between 2 and 50

Filter out everything divisible by 3:

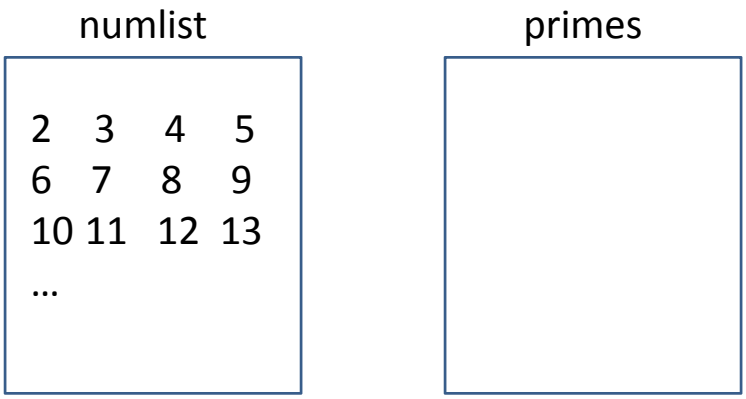
```
items.delete_if { |i| (i>3) & (i%3 == 0) }
```

```
    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
```


Designing an Algorithm

- Algorithm: a precise rule (or set of rules) specifying how to solve a problem (thefreedictionary.com)
  - What are the inputs and outputs for the computation
  - The order in which the steps will be executed during computation

Automating the Sieve



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

An Algorithm for Sieve of Eratosthenes

To make a list of every prime number less than n:

1. Create an array *numlist* with every integer from 2 to n, in order. (Assume n > 1.)

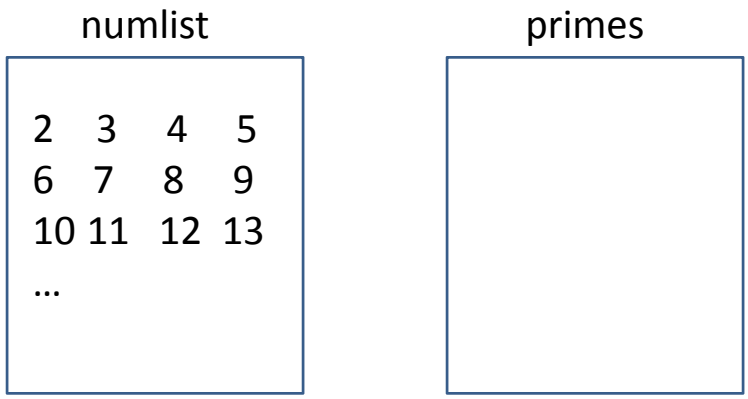
2. Create an empty array *primes*.

3. Copy the first number in *numlist* to the end of *primes*.

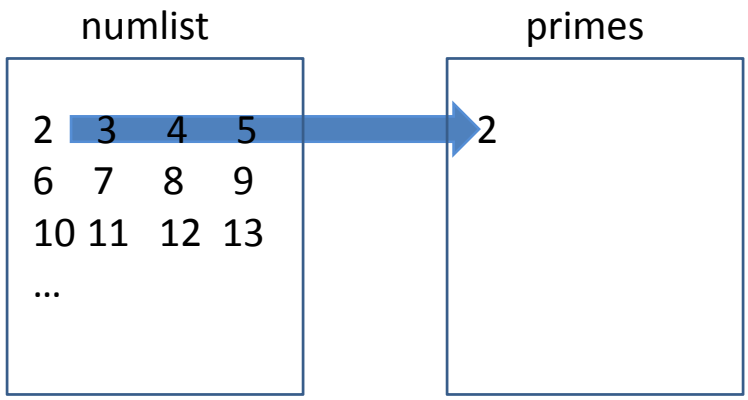
4. Iterate over *numlist* to remove every number that is a multiple of the most recently discovered prime number.

5. Halt when *numlist* is empty. Otherwise, go back to step 3.

Steps 1 and 2

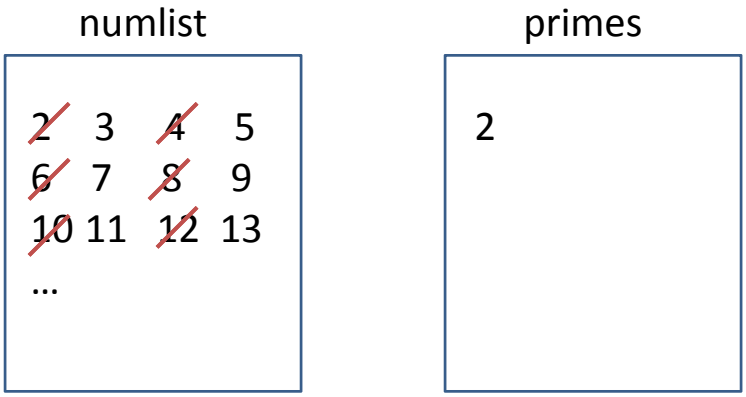


Step 3



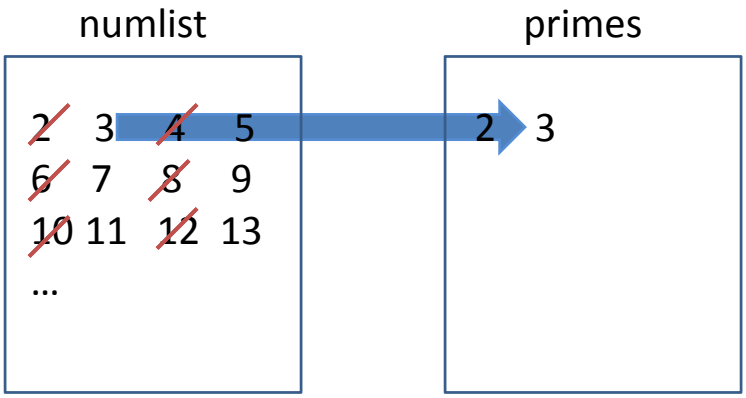
Append the first number in numlist to the end of primes.

Step 4



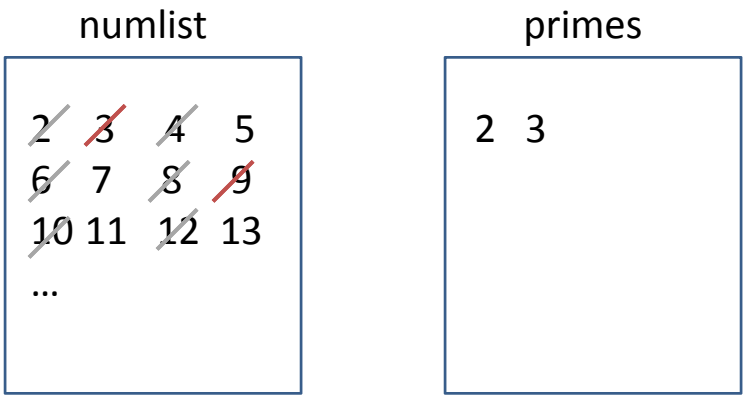
Cross out all the multiples of the last number in primes.

Iterations



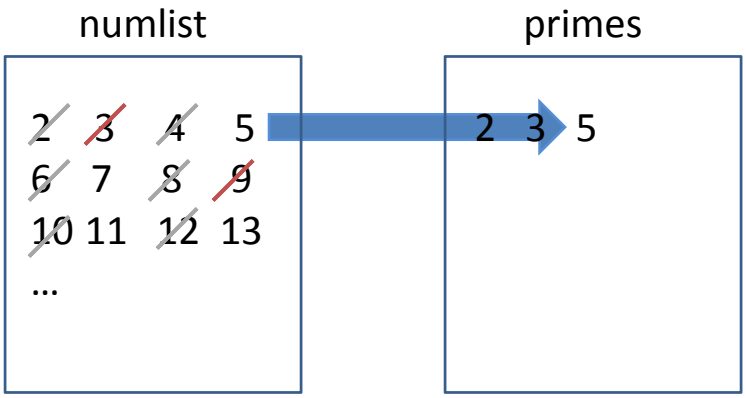
Append the first number in numlist to the end of primes.

Iterations



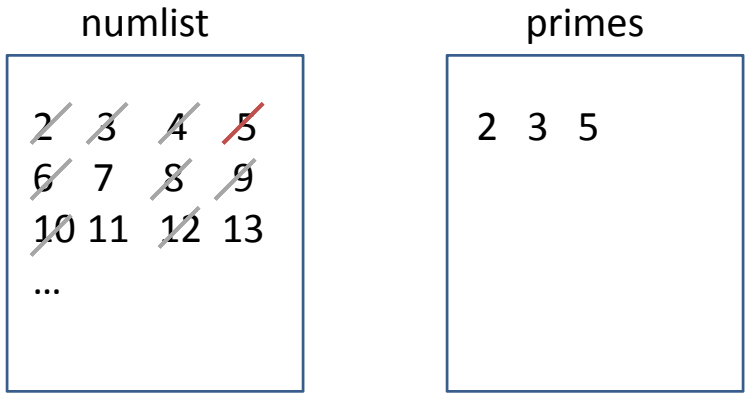
Cross out all the multiples of the last number in primes.

Iterations



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

Iterations



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

Coding the Algorithm in Ruby

```
def sieve(n)
        numlist = Array(2..n)      # create the input array
        primes = [ ]               # initialize the primes array
        primes << numlist.first    # append the first number
                                   # to primes
        . . .
```

How do we know that numlist.first is a prime?

Removing Multiples of a Prime

- Where is the most recent prime added to the **primes** list?
  primes.last
- How do we determine whether a number **x** is a multiple of the most recent prime?
  x % primes.last == 0
- If x is a multiple of the most recent prime, it's not prime!
  numlist.delete_if { |x| x % primes.last == 0 }

Continuing the function in Ruby

```
def sieve(n)
    numlist = Array(2..n)
    primes = []
    primes << numlist.first
    numlist.delete_if { |x|
        x % primes.last == 0
    }
    . . .
```

This part has to be repeated until numlist is empty

Use a Loop

```
while loop condition do
        primes << numlist.first
        numlist.delete_if { |x| x % primes.last == 0 }
end
```

What should the loop condition be?
It determines when we stop looping.

A Working Sieve

```
def sieve(n)
        numlist = Array(2..n)
        primes = []

        while not numlist.empy? do
            primes << numlist.first
            numlist.delete_if { |x| x % primes.last == 0 }
        end

        return primes
end
```

Recall the Last Lecture

We stopped at 11 because all the remaining entries must be prime since 11 × 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

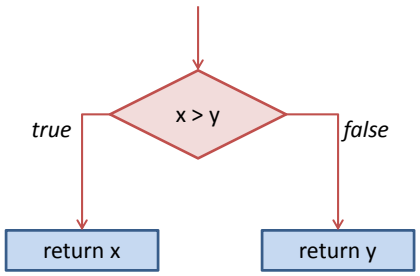
```
def sieve(n)
        numlist = Array(2..n)
        primes = []

       while numlist.first < sqrt(n) do
          primes << numlist.first
          numlist.delete_if { |x| x % primes.last == 0 }
        end

        return primes  + numlist
end
```

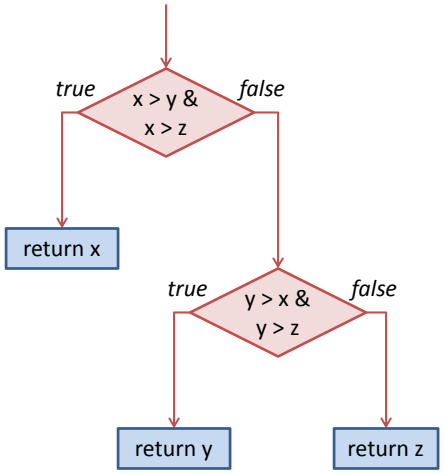
Fun With IF Statements

```
def max2 (x, y)
  if x > y then
    return x
  else
    return y
  end
end
```



Fun With IF Statements

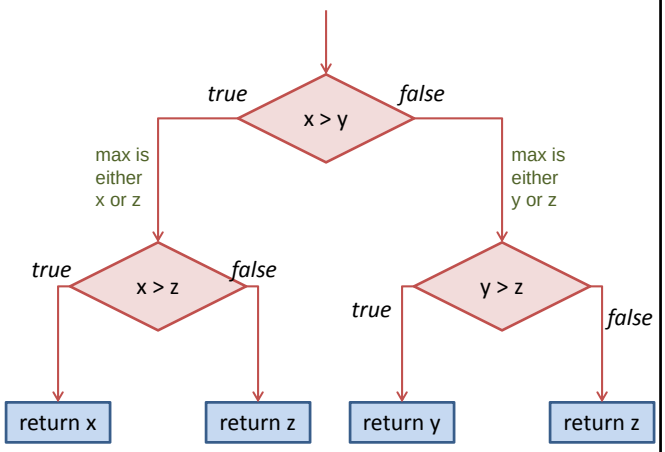
```
def max3(x, y, z)
  if x > y and x > z then
    return x
  elsif y > x and y > z then
    return y
  else
    return z
  end
end
```



May involve up to 4 comparisons.

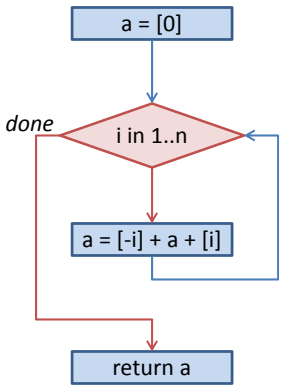
Fun With IF Statements

```
def max3 (x, y, z)
  if x > y then
    if x > z then
      return x
    else
      return z
    end
  elsif y > z then
    return y
  else
    return z
  end
end
```



Fun With FOR Loops

```
def both_ends(n)
  a = [ 0 ]
  for i in 1..n do
    a = [-i] + a + [i]
  end
  return a
end
```

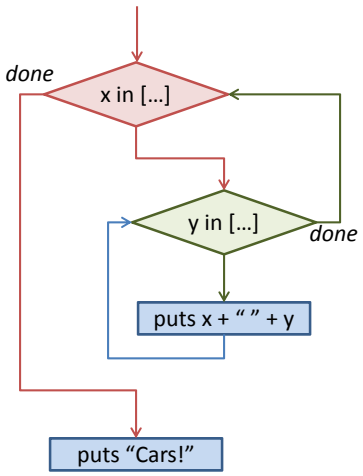



```
>> both_ends(5)
=> [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
```

Nested FOR Loops: Cross Product

```
for x in [“red”, “blue”, “silver”] do
  for y in [“sedan”, “SUV”] do
    puts x + “ ” + y
  end
end
puts “Cars!”
```

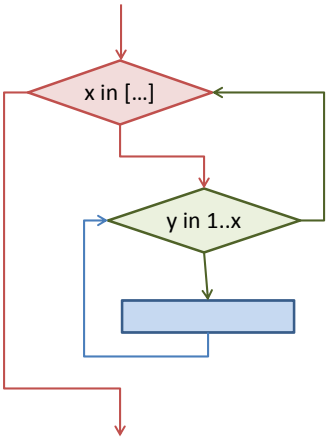
```
red sedan
red SUV
blue sedan
blue SUV
silver sedan
silver SUV
Cars!
```



Nested FOR Loops: Dependent

```
for x in [4, 2, 6] do
  for y in 1..x do
    puts x.to_s * x
  end
end
```

```
4444
4444
4444
4444
22
22
666666
666666
666666
666666
666666
666666
```



Measuring Run Time w/RubyLabs

```
>> time { slow_sieve(50000) }
=> 6.715435

>> time { sieve(50000) }
=> 0.212703
```