

UNIT 3

Algorithmic Thinking

Algorithms

- An algorithm is “a precise rule (or set of rules) specifying how to solve some problem.”
(thefreedictionary.com)
- Mohammed al-Khowarizmi (äl-khōwārēz’ mē)
Arab mathematician of the court of Mamun in Baghdad...the word *algorithm* is said to have been derived from his name. Much of the mathematical knowledge of medieval Europe was derived from Latin translations of his works.
(encyclopedia.com)
- The study of algorithms is one of the key foundations of computer science.

A Recipe is an Algorithm

½ cup of butter or margarine	1 teaspoon of vanilla extract
1 cup of sugar	½ cup of unsweetened cocoa
2 eggs	½ cup of flour

1. If butter or margarine is not melted, melt in a bowl in microwave for 30 seconds at high power.
2. Blend melted butter or margarine and sugar until the mixture has a creamy consistency.
3. Add eggs & vanilla, and stir the mixture 60 times.
4. Add cocoa and flour.
5. Mix until well blended.
6. Pour into greased round glass cake pan.
7. Microwave for 8 to 9 minutes on 50% power. Brownies will be done when they are slightly moist on top and pull away from the side of the pan.

Serves: 4

-- adapted from *yummy.com*

The Tax Code is an Algorithm

1. Write your total wages from your W-2 statements on Line 1.
2. Add up all the interest amounts from your 1099-INT forms and put the total on Line 2.
3. Gather all your 1099-G statement from the state agency that paid you unemployment compensation. Put the figure from the 1099-G on Line 3.
 - a. If you received Alaska Permanent Fund dividends only, put the figure reported to you by the State of Alaska on Line 3.
 - b. If you have both unemployment and Alaskan dividends, add the two figures together and put the total on Line 3.
4. Add lines 1, 2 and 3 to determine your Adjusted Gross Income (AGI) and write this on Line 4.
5. Determine your personal exemptions for Line 5.
 - a. If you are being claimed as a dependent, check the "Yes" box on Line 5. Otherwise, check the "No" box on Line 5.
 - b. If you are unmarried, or you are married and you are not filing a joint return, put the figure \$7,950 on Line 5. Otherwise, put the figure \$15,900 on Line 5.
6. Subtract Line 5 from Line 4 and write this total in Line 6. This is your taxable income.
etc.

- Adapted from the US Tax Code Form 1040EZ

Knitting is an Algorithm

1. Hold needle with stitches in left hand; insert point of right needle in first stitch, from front to back, just as in casting on.
2. With right index finger, bring yarn from ball under and over point of right needle.
3. Draw yarn through stitch with right needle point.
4. This step now differs from casting on: Slip loop on left needle off, so new stitch is entirely on right needle.
5. This completes one knit stitch. Repeat Steps 1 through 4 in each stitch still on left needle. When the last stitch is worked, one row of knitting is completed and you can move to Step 6.
6. Now measure your work. It should be about 7" wide. If it is too wide, start over and cast on fewer stitches; if it is too narrow, start over and cast on more stitches.

- Adapted from learntoknit.com

An algorithm is like a function

$$F(x) = y$$



Input

- Input specification
 - Recipes: ingredients, cooking utensils, ...
 - Tax Code: wages, interest, tax withheld, ...
 - Knitting: size of garment, length of yarn, needles ...
- Input specification for computational algorithms:
 - How much data is required?
 - What kind of data is required?
 - In what form will this data be received by the algorithm?

Computation

- An algorithm requires clear and precisely stated steps that express how to perform the operations to yield the desired results.
- Algorithms assume a basic set of primitive operations that are assumed to be understood by the executor of the algorithm.
 - Recipes: beat, stir, blend, bake, ...
 - Tax code: deduct, look up, check box, ...
 - Knitting: casting on, slip loop, draw yarn through, ...
 - Computational: add, set, modulo, output, ...

Output

- Output specification
 - Recipes: number of servings, how to serve
 - Tax Code: tax due or tax refund, where to pay
 - Knitting: final garment shape
- Output specification for computational algorithms:
 - What results are required?
 - How should these results be reported?
 - What happens if no results can be computed due to an error in the input? What do we output to indicate this?

What makes a “good” algorithm?

- A good algorithm should produce the correct outputs for any set of legal inputs.
- A good algorithm should execute efficiently with the fewest number of steps as possible.
- A good algorithm should be designed in such a way that others will be able to understand it and modify it to specify solutions to additional problems.

Is this a “good” algorithm?

- Input: slices of bread, jar of peanut butter, jar of jelly
 - 1. Pick up some bread.**
 - 2. Put peanut butter on the bread.**
 - 3. Pick up some more bread.**
 - 4. Open the jar of jelly.**
 - 5. Spread the jelly on the bread.**
 - 6. Put the bread together to make your sandwich.**
- Output?



First Algorithm: GCD

Input: two positive integers x and y

Algorithm:

1. While y is not 0, do the following:
 - a. Set temp equal to y
 - b. Set y equal to $x \bmod y$
 - c. Set x equal to temp
2. Return x as the GCD

Output: the GCD of the original x and y

Iterative Solution

Input: two positive integers x and y

Algorithm:

1. While y is not 0, do the following:

a. Set temp equal to y

b. Set y equal to x modulo y

c. Set x equal to temp

} loop
body

2. Return x as the GCD

Output: the GCD of the original x and y

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.

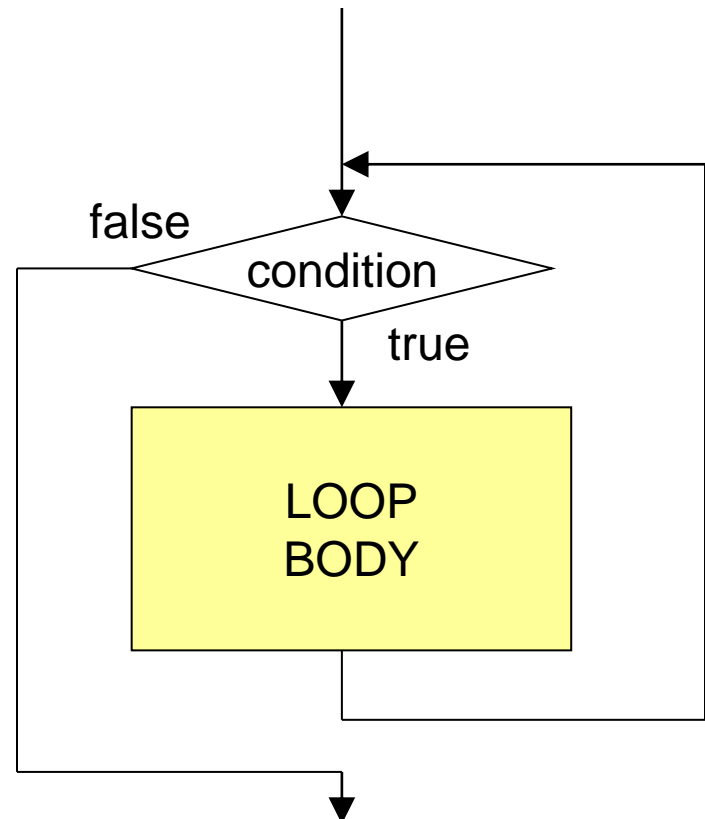
while loop

Format:

```
while condition do  
    loop body  
end
```

one or more instructions
to be repeated

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.



Iterative Solution using Ruby

```
def gcd1(x,y)
  while y != 0 do
    temp = y
    y = x % y
    x = temp
  end
  return x
end
```

Recursive Solution

A recursive algorithm is an algorithm that uses a simpler version of itself as part of its solution.

Input: two non-negative integers x and y

Algorithm:

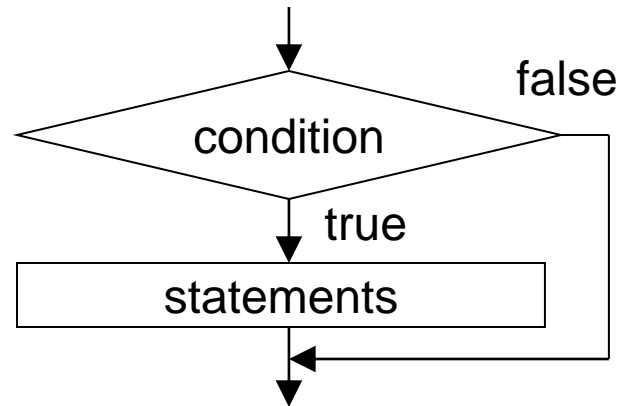
1. If y is equal to 0, return x as the GCD.
2. Otherwise,
return the GCD of y and $(x \bmod y)$ as the GCD.

Output: the GCD of the initial x and y

if statement

Format:

if condition then
statement_list
end



if/else statement

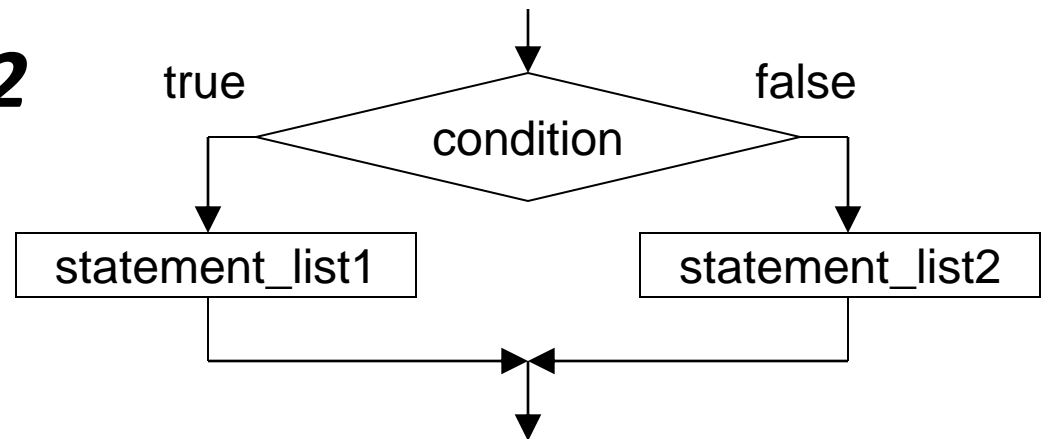
Format:

if condition then
statement_list1

else

statement_list2

end



Recursive Solution using Ruby

```
def gcd2(x, y)
  if y == 0 then
    return x
  else
    return gcd2(y, x % y)
  end
end
```



This is recursive since
gcd2 calls itself.
More about recursion soon.

Finding the maximum

How do we find the maximum in a sequence of integers shown to us one at a time?

183

What's the maximum?

Finding the maximum

Required: a non-empty *list* of integers.

1. Set *max_so_far* equal to the first number in the *list*.
2. For each number *n* in the *list*:
 - a. If *n* is greater than *max_so_far*, then set *max_so_far* equal to *n*.

Return: *max_so_far* as the maximum of the *list*.

Representing Lists in Ruby

In Ruby, we will use an **array** to represent a list of data values.

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

```
colors = ["red", "green", "blue"]
```

An array is an *ordered* list because the order of the elements matters.

Some Array Operations

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

```
scores.length      => 7
```

```
scores.first       => 78
```

```
scores.last        => 85
```

```
scores.first * 2    => 156
```

```
scores.include?(100) => true
```

```
scores[0]          => 78
```

```
scores << 92
```

```
=> [78, 93, 80, 68, 100, 94, 85, 92]
```


Finding the max using Ruby

```
def findmax(list)
  max_so_far = list.first      # or list[0]
  for i in (1..list.length-1) do
    if list[i] > max_so_far then
      max_so_far = list[i]
    end
  end
  return max_so_far
end
```


Alternate Version

```
def findmax(list)
  max_so_far = list.first
  for item in list do
    if item > max_so_far then
      max_so_far = item
    end
  end
  return max_so_far
end
```

“For each item in the list...”



Iterators: Using the **each** method

```
scores = [78, 93, 80, 68, 100, 94, 85]
scores.each { |item|      ← “For each item in scores...”
  print item, " "
}                          => 78 93 80 68 100 94 85
```

```
scores.each { |x|        ← “For each x in scores...”
  if x % 2 == 1 then
    print x, " "
  end
}                          => 93 85
```

Alternate Version #2

```
def findmax(list)
  max_so_far = list.first
  list.each { |item|
    if item > max_so_far then
      max_so_far = item
    end
  }
  return max_so_far
end
```

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 }
```

“For each element **n** in the array **scores**,
delete **n** if **n** is less than 80.”

```
=> [ 93, 80, 100, 94, 85]
```

```
scores.delete_if{ |n| n % 2 == 0 }
```

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*. (It must be prime. Why?)
4. Iterate over *numlist* to remove every number that is a multiple of the most recently discovered prime number.
5. Halt if every number in *numlist* is prime. Otherwise, go back to step 3.

Arrays: Two Special Cases

```
values = []
```

```
=> []
```

This is the empty array (an array with 0 length).

```
values = Array(1..8)
```

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

Starting the algorithm in Ruby

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

  . . .
```


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?

Use the modulo operator!

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 algorithm in Ruby

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

This algorithm has a loop

We need to repeat the following two steps:

```
primes << numlist.first  
numlist.delete_if { |x| x % primes.last == 0 }
```

Example: start with `numlist = Array(2..25)`

```
primes = [2]  
numlist = [3,5,7,9,11,13,15,17,19,21,23,25]
```

```
primes = [2,3]  
numlist = [5,7,11,13,17,19,23,25]
```

...

When do we stop?

We need to repeat the following two steps:

```
primes << numlist.first  
numlist.delete_if { |x| x % primes.last == 0 }
```

while what is true?

```
numlist.length > 0  
or  numlist.length >= 1  
or  numlist.length != 0
```

Final Algorithm in Ruby

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