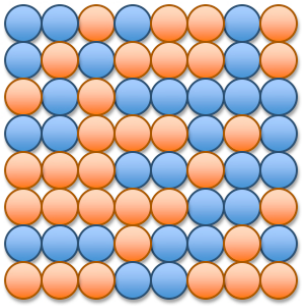


# UNIT 3A
# Algorithmic Thinking



## Algorithms

- An algorithm is "a precise rule (or set of rules) specifying how to solve some problem." (thefreedictionary.com)
- Mohammed al-Khowarizmi (äl-khōwärēz´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 cup of sugar
2 eggs
1 teaspoon of vanilla extract
½ cup of unsweetened cocoa
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 yumyum.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.

# 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 modulo 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
2. Return x as the GCD
Output: the GCD of the original x and y

loop body

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.

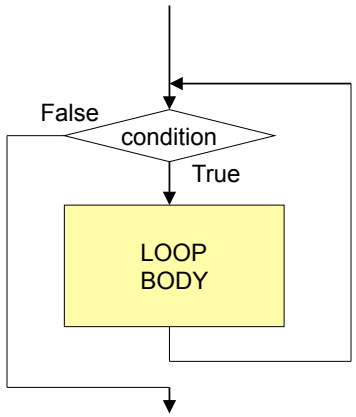
# `while` loop

Format:

```
while condition:
    loop body
```

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.






# While vs. For Loops

```
# Prints first 10 positive
# integers
i = 1
while  i <= 10:
    print(i)
    i = i + 1
```

```
# Prints first 10 positive
# integers
for i in range(1,11):
        print(i)
```

# Iterative Solution using Python

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



# Example: Accumulation by multiplying as well as by adding

An epidemic:

```
def compute_sick(n):
    # computes total sick after n days
    total_sick = 1
    newly_sick = 1
    for day in range(2, n + 1):
        # each iteration represents one day
        newly_sick = newly_sick * 2
        total_sick = total_sick + newly_sick
    return total_sick
```

Each newly infected person infects 2 people the next day.

# An epidemic (cont'd)

```
compute_sick(1)  => 1
compute_sick(2)  => 3
compute_sick(3)  => 7
compute_sick(4)  => 15
compute_sick(5)  => 31
compute_sick(6)  => 63
compute_sick(7)  => 127
compute_sick(8)  => 255
compute_sick(9)  => 511
compute_sick(10) => 1023
compute_sick(11) => 2047
compute_sick(12) => 4095
compute_sick(13) => 8191
compute_sick(14) => 16383
compute_sick(15) => 32767
compute_sick(16) => 65535
compute_sick(17) => 131071
compute_sick(18) => 262143
compute_sick(19) => 524287
compute_sick(20) => 1048575
compute_sick(21) => 2097151
```

In just three weeks, over 2 million people are sick!
(This is what Blown To Bits means by *exponential growth*. We will see important computational problems that get exponentially "harder" as the problems gets bigger.)



# Another version using a while loop

```
def time_to_catastrophe(population):
    days = 1
    newly_sick = 1
    total_sick = 1
    while total_sick < population:
        newly_sick = newly_sick * 2
        total_sick = total_sick + newly_sick
        days = days + 1
    return(days)


>>> time_to_catastrophe(1000000000)
30
```

(A billion people can all get sick in this model in just 30 days!)