Loops

Sep 25

scope, loops, ++, return

Powers of N

- Write a program that takes two parameter from the command line, both positive integers.
  - call the first base
  - call the second maxx
- Find the smallest power of base that exceeds max.
- Print the number, the power, and base raised to that power.

PowerIf

```
public class PowerIf {
    public static void main(String[] args) {
        int base = Integer.parseInt(args[0]);
        int maxx = Integer.parseInt(args[1]);
        int power = 1;
        int basePower = base;
        if (basePower > maxx) {
            System.out.println(base + " " + maxx + " power: " + power + " basePower: " + basePower);
        } else {
            power++;
            basePower *= base;
            if (basePower > maxx) {
                System.out.println(base + " " + maxx + " power: " + power + " basePower: " + basePower);
            } else {
                power++;
                basePower *= base;
                if (basePower > maxx) {
                    System.out.println(base + " " + maxx + " power: " + power + " basePower: " + basePower);
                } else {
                    power++;
                    basePower *= base;
                    if (basePower > maxx) {
                        System.out.println(base + " " + maxx + " power: " + power + " basePower: " +
basePower);
                    } else {
                        power++;
                        basePower *= base;
                    }
                }
            }
        }
    }
}
```

PowerIf Problems

- Did not handle case where numbers are negative
- What if the target requires greater that base raised to the fourth power?
  - In this program, print NOTHING!
    - Improve?
- Improve using "return"
  - allows the program to exit early
  - better in terms of fewer {} and no "else"

Loops

- Loops allow a program to execute repeated statements without the programmer having to write them
- Even better, execute statements and unbounded number of times!!!

- Problem, stopping the loop
  - before you write a loop, you must know how it is going to end
  - "infinite loops"

While loop

- Idea,
  - execute the "body" until the condition is false

```
while (CONDITION) {
    BODY
}
```

  -
```
int i=1;
while (i<5) {
    i++;
    System.out.println(i);
}
```
  - An infinite loop?

Powers of N

using While

```
public class PowerWhile {
    public static void main(String[] args) {
        int base = Integer.parseInt(args[0]);
        int maxx = Integer.parseInt(args[1]);
        int power = 1;
        int basePower = base;
        while (basePower < maxx) {
            power++;
            basePower *= base;
        }
        System.out.println(base + " " + maxx + " power: " + power + " basePower: " + basePower);

    }
}
```

Sum Power

- Calculate SUM(i=1 to n)(k^i)
  - For instance if n = 3 and k = 5, sum = 5^1 + 5^2 + 5^3

- Algorithm
- Program

Randomness

- Lots of times it is useful to get a random number.
  - Computers are really bad at this
    - Why?
  - Lots of work at making computers give numbers that are indistinguishable from random
    - Tippett A Million Random Digits with 100,000 Normal Deviates.
    - "pseudo-random"

- Java provides

Java Random number generator

- Random double in range 0.0 .. <1.0
  - double d = Math.random()

- Random doubl1 in range 0.0 .. <100.0
  - double d = Math.random()*100

- Random double in range -50 .. <50
  - double d = (Math.random()*50)-50

- Random integer in range 0 .. <100
  - int i = (int)(Math.random()*100)

Whiteboard

Crapping out

- In the casino game of craps (which uses two standard 6-sided dice), everyone on the "pass line" looses when a 2, 3, or 12 is rolled. This is referred to as "crapping out"
- Write a program that
  - Calculates the number of times a pair of dice must be rolled before crapping out.

- Note: each time the program runs you would might get a different answer