Review

- Recursion (recursive function)
  - a function that calls itself
  - base case
  - reduction of the work to a smaller instance
- Rotation ccw in Processing – negative angle

What does this do?

```
void setup() {
  println(F(12));
}

int F(int n) {
  if (n == 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return F(n-1) + F(n-2);
  }
}
```

What does this do?

```
void setup() {
  println(F(12));
}

int F(int n) {
  if (n == 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return F(n-1) + F(n-2);
  }
}
```

Base Case for 0.

What does this do?

```
void setup() {
  println(F(12));
}

int F(int n) {
  if (n == 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return F(n-1) + F(n-2);
  }
}
```

Base Case for 0.

Base Case for 1.

What does this do?

```
void setup() {
  println(F(12));
}

int F(int n) {
  if (n == 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return F(n-1) + F(n-2);
  }
}
```

Base Case for 0.

Base Case for 1.

Calls itself with a smaller n

What does this do?

```
void setup() {
  println(F(12));
}

int F(int n) {
  if (n == 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return F(n-1) + F(n-2);
  }
}
```

Base Case for 0.

Base Case for 1.

Calls itself with a smaller n

Calls itself with a smaller n

What does this do?

```
1  void setup() {
2    println(F(3));
3  }
4
5  int F(int n) {
6    if (n == 0) {
7      return 0;
8    } else if (n == 1) {
9      return 1;
10   } else {
11     return F(n-1) + F(n-2);
12   }
13 }
14
15
```

Function	n	line number
F(3)	3	12
F(2)	2	12
F(1)	1	10
F(0)	0	7
1 + …	3	12
F(1)	1	10
1 + 1	3	12

What does this do?

```
1  void setup() {
2    println(F(12));
3  }
4
5  int F(int n) {
6    if (n == 0) {
7      return 0;
8    } else if (n == 1) {
9      return 1;
10   } else {
11     return F(n-1) + F(n-2);
12   }
13 }
14
15
```

Function	n	line number
F(12)	12	12

What does this do?

```
1  void setup() {
2    println(F(12));
3  }
4
5  int F(int n) {
6    if (n == 0) {
7      return 0;
8    } else if (n == 1) {
9      return 1;
10   } else {
11     return F(n-1) + F(n-2);
12   }
13 }
14
15
```

Function	n	line number
F(12)	12	12
F(11) + F(10)	11 10	12 12

What does this do?

```
1  void setup() {
2    println(F(12));
3  }
4
5  int F(int n) {
6    if (n == 0) {
7      return 0;
8    } else if (n == 1) {
9      return 1;
10   } else {
11     return F(n-1) + F(n-2);
12   }
13 }
14
15
```

Function	n	line number
F(12)	12	12
F(11) + F(10)	11 10	12 12
F(10) + F(9) F(9) + F(8)	10 9 9 8	12 12 12 12

What does this do?

```
1  void setup() {
2    println(F(12));
3  }
4
5  int F(int n) {
6    if (n == 0) {
7      return 0;
8    } else if (n == 1) {
9      return 1;
10   } else {
11     return F(n-1) + F(n-2);
12   }
13 }
14
15
```

Function	n	line number
F(12)	12	12
F(11) + F(10)	11 10	12 12
F(10) + F(9) F(9) + F(8)	10 9 9 8	12 12 12 12
F(9) + F(8) F(8) + F(7) F(8) + F(7) F(7) + F(6)	9 8 8 7 8 7 7 6	12 12 12 12 12 12 12 12

What does this do?

```
1  void setup() {
2    println(F(12));
3  }
4
5  int F(int n) {
6    if (n == 0) {
7      return 0;
8    } else if (n == 1) {
9      return 1;
10   } else {
11     return F(n-1) + F(n-2);
12   }
13 }
14
15
```

Function	n	line number
F(12)	12	12
F(11) + F(10)	11 10	12 12
F(10) + F(9) F(9) + F(8)	10 9 9 8	12 12 12 12
F(9) + F(8) F(8) + F(7) F(8) + F(7) F(7) + F(6)	9 8 8 7 8 7 7 6	12 12 12 12 12 12 12 12
F(8) + F(7) F(7) + F(6) F(7) + F(6) F(6) + F(5) F(7) + F(6) F(6) + F(5) F(6) + F(5) F(5) + F(4)	8 7 7 6 7 6 6 5 7 6 6 5 6 5 5 4	12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12

What does this do?

```
1  void setup() {
2    println(F(12));
3  }
4
5  int F(int n) {
6    if (n == 0) {
7      return 0;
8    } else if (n == 1) {
9      return 1;
10   } else {
11     return F(n-1) + F(n-2);
12   }
13 }
14
15
```

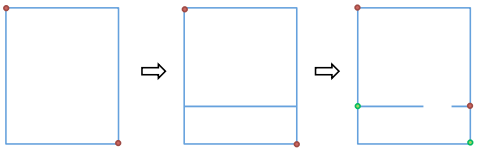
Function	n	line number
F(12)	12	12

Examples

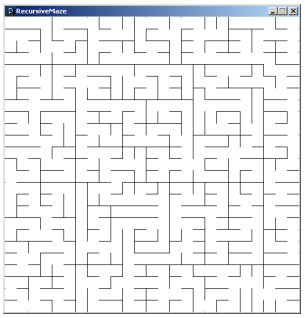
- recursiveLines
- recursiveShapes

Creating a maze, recursively

1. Start with a rectangular region defined by its upper left and lower right corners
2. Divide the region at a random location through its more narrow dimension
3. Add an opening at a random location
4. Repeat on two rectangular subregions



Inspired by http://weblog.jamisbuck.org/2011/1/12/maze-generation-recursive-division-algorithm



Examples

- recursive sum
- recursive sum with array
- recursive findMax

Lindenmayer System

- A formal grammar developed to model the development of biological systems
- Generates strings that represent movements
- When traced in the plane, produce remarkable lifelike plant systems
- Components
  - An alphabet (a set of symbols)
  - An axiom or start string
  - A rule set that defines substitutions

L-system Example

- Alphabet: {A, B}
- Axiom: A
- Rules
  1. {A –> AB}
  2. {B –> A}
- Generation:
  1. A axiom
  2. AB rule 1
  3. ABA rule 1&2
  4. ABAAB rule 1&2
  5. ABAABABA rule 1&2

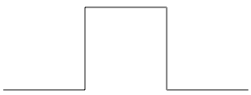
- All symbols that have available rules are substituted
- Substitutions are simultaneous

Turtle Graphics

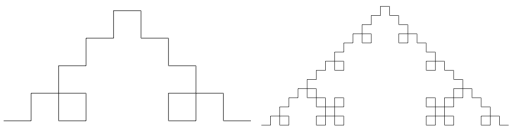
- Imaginary turtle with a pen
- Moves in the plane
  - Forward
  - Turn left
  - Turn right
- Traces with the pen as it moves
- Can put the pen up or down
  - Pen up: no trace
  - Pen down: trace

L-systems Example

- Alphabet: {F}
- Axiom: F
- Rules
  1. {F –> F+F-F-F+F}
- Interpretations:
  1. F Forward (pen down)
  2. + Turn left (pen up)
  3. - Turn right (pen up)

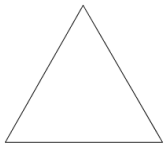


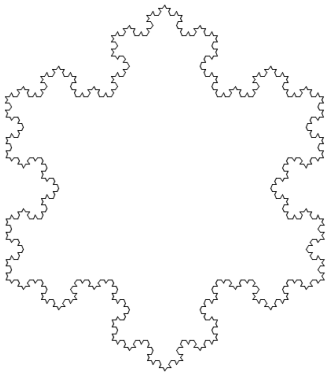
3 and 5 Iterations



Koch Snowflake

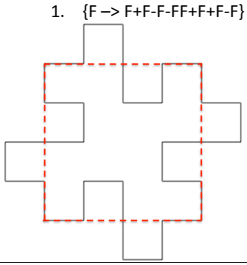
- Alphabet: {F}
- Axiom: F++F++F
- Rules
  1. {F –> F-F++F-F}

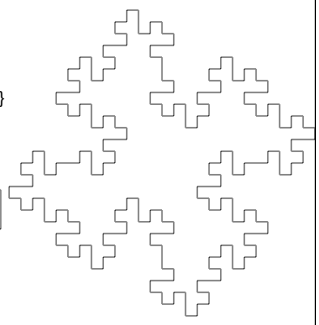




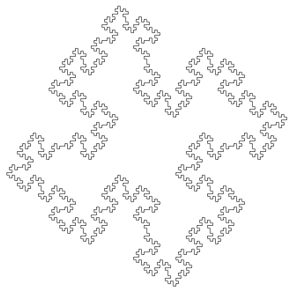
Quadratic Flake

- Alphabet: {F}
- Axiom: F+F+F+F
- Rules
  1. {F –> F+F-F-FF+F+F-F}

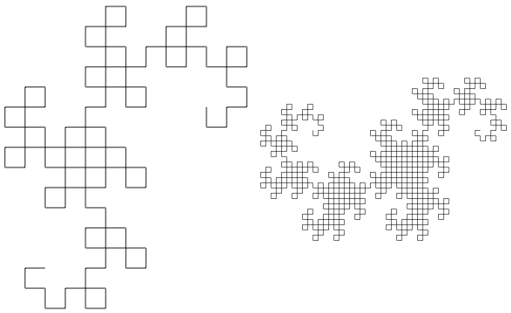




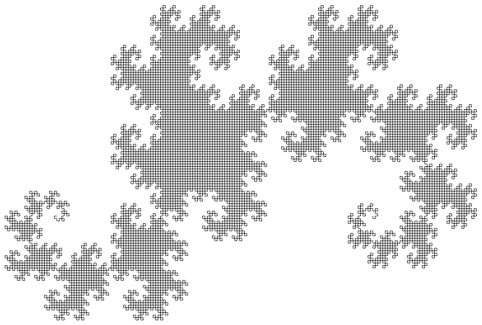
4 Iterations



Heighway Dragon (8 and 11 iterations)



15 Iterations



Plants

- Alphabet {F}
- Axiom: F
- Rules:
  1. {F –> F[-F]F[+F]F}