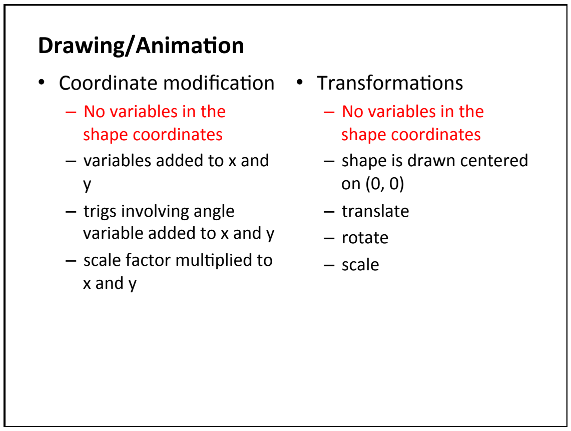
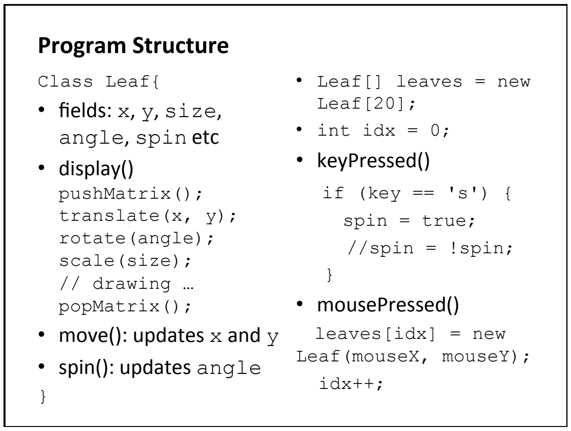


Drawing/Animation

- Coordinate modification
  - No variables in the shape coordinates
  - variables added to x and y
  - trigs involving angle variable added to x and y
  - scale factor multiplied to x and y
- Transformations
  - No variables in the shape coordinates
  - shape is drawn centered on (0, 0)
  - translate
  - rotate
  - scale



Program Structure

```
Class Leaf{
```

- fields: `x, y, size, angle, spin` etc
- display()

```
  pushMatrix();
  translate(x, y);
  rotate(angle);
  scale(size);
  // drawing …
  popMatrix();
```

- move(): updates `x` and `y`
- spin(): updates `angle`

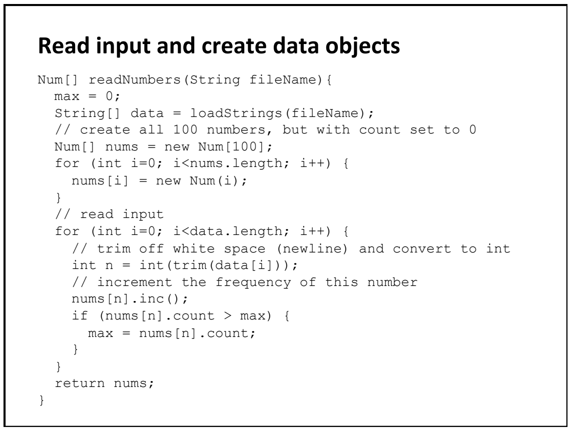
```
}
```

- `Leaf[] leaves = new Leaf[20];`
- `int idx = 0;`
- keyPressed()

```
  if (key == 's') {
    spin = true;
    //spin = !spin;
  }
```

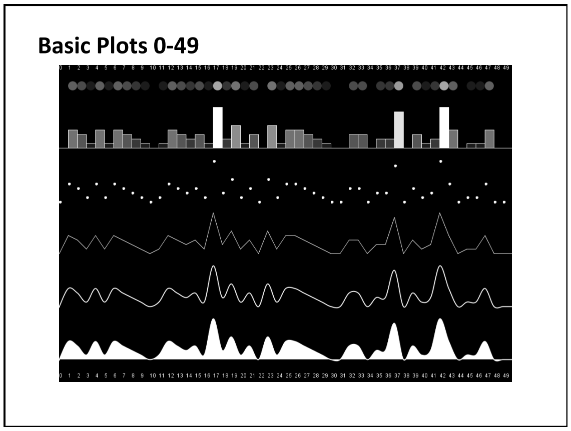
- mousePressed()

```
  leaves[idx] = new
Leaf(mouseX, mouseY);
  idx++;
```

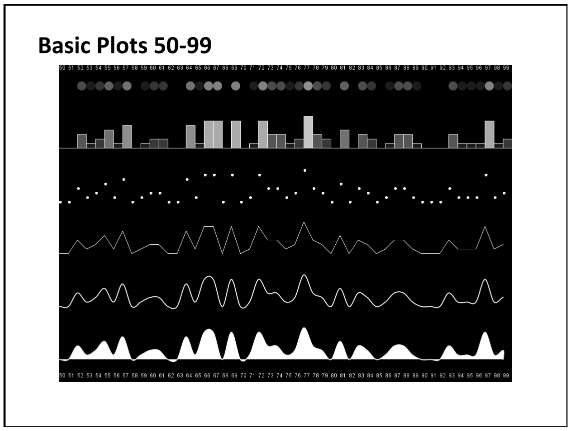


Read input and create data objects

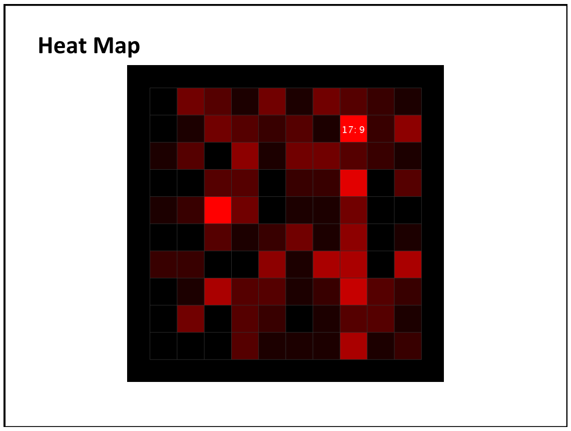
```
Num[] readNumbers(String fileName){
  max = 0;
  String[] data = loadStrings(fileName);
  // create all 100 numbers, but with count set to 0
  Num[] nums = new Num[100];
  for (int i=0; i<nums.length; i++) {
    nums[i] = new Num(i);
  }
  // read input
  for (int i=0; i<data.length; i++) {
    // trim off white space (newline) and convert to int
    int n = int(trim(data[i]));
    // increment the frequency of this number
    nums[n].inc();
    if (nums[n].count > max) {
      max = nums[n].count;
    }
  }
  return nums;
}
```



Basic Plots 0-49



Basic Plots 50-99



Heat Map

split

- `split` breaks a string into pieces using a delimiter
- a string array is returned containing the pieces

- `String[] paramString = split(lines[0], " ");`
- `String[] paramPieces = split(paramString[1], ",");`

Raw Data

```
# 41556,-0.3667764,0.35192886,0.4181981,0.87044954
00210  0.3135056  0.7633538  Portsmouth, NH
00211  0.3135056  0.7633538  Portsmouth, NH
00212  0.3135056  0.7633538  Portsmouth, NH
00213  0.3135056  0.7633538  Portsmouth, NH
00214  0.3135056  0.7633538  Portsmouth, NH
00215  0.3135056  0.7633538  Portsmouth, NH
00501  0.30247012 0.7226447  Holtsville, NY
00544  0.30247012 0.7226447  Holtsville, NY
...
```

file name: zips.txt

Examples

- Zip
- ZipInteractive

Abstract classes

- keyword abstract defines a class that can not be instantiated
  - A a1 = A(0, 0);
- A generic class with abstract (undefined) methods
- Subclasses of an abstract class MUST implement all abstract methods

```
abstract class A {
  int x; int y;
  A(int x, int y){
   this.x = x;
   this.y = y;
  }
  abstract void display();
}

class Ball extends A {
  Ball(int x, int y){
    super(x, y);
  }
  void display() {
    ellipse(x, y, 20, 20)
  }
}
```

Factorial

- The factorial of a positive integer N is computed as the product of N with all positive integers less than or equal to N.

4! = 4 × 3 × 2 × 1 = 24

30! = 30 × 29 × … × 2 × 1 = 265252859812191058636308480000000

Factorial - Iterative Implementation

```
void setup() {
  int A = 10;
  int B = factorial(5);
  println( B );
}

int factorial(int N) {
  int F = 1;

  for( int i=N; i>=1; i--) {
    F = F * i;
  }

  return F;
}
```

Trace it.

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$4! = 4 \times 3 \times 2 \times 1$

$5! = 5 \times 4!$

$N! = N \times (N-1)!$

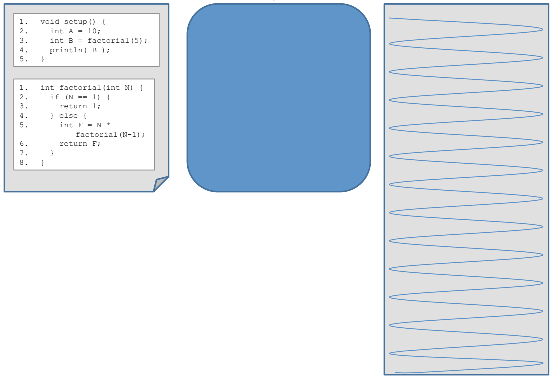
Factorial can be defined in terms of itself

Factorial – Recursive Implementation

```
void setup() {
  int A = 10;
  int B = factorial(5);
  println( B );
}

int factorial(int N) {
  if (N == 1) {
    return 1;
  } else {
    int F = N * factorial(N-1);
    return F;
  }
}
```

Last In First Out (LIFO) Stack of Plates



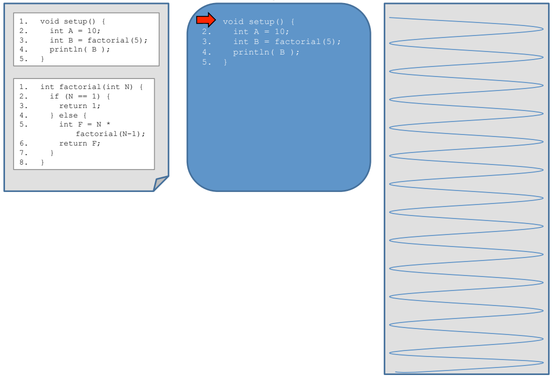
Compiled Code

```
void setup() {
  int A = 10;
  int B = factorial(5);
  println( B );
}
```

```
int factorial(int N) {
  if (N == 1) {
    return 1;
  } else {
    int F = N *
        factorial(N-1);
    return F;
  }
}
```

Executing Function

Call Stack



Compiled Code

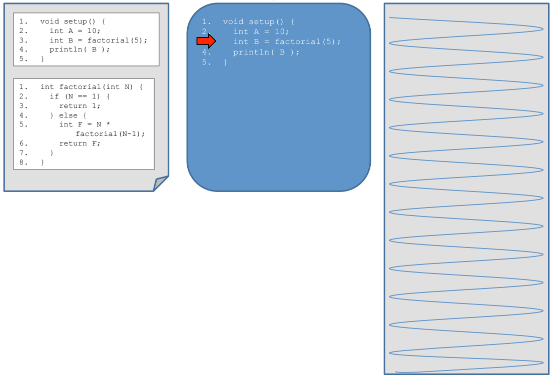
```
void setup() {
  int A = 10;
  int B = factorial(5);
  println( B );
}
```

```
int factorial(int N) {
  if (N == 1) {
    return 1;
  } else {
    int F = N *
        factorial(N-1);
    return F;
  }
}
```

Executing Function

```
void setup() {
  int A = 10;
  int B = factorial(5);
  println( B );
}
```

Call Stack



Compiled Code

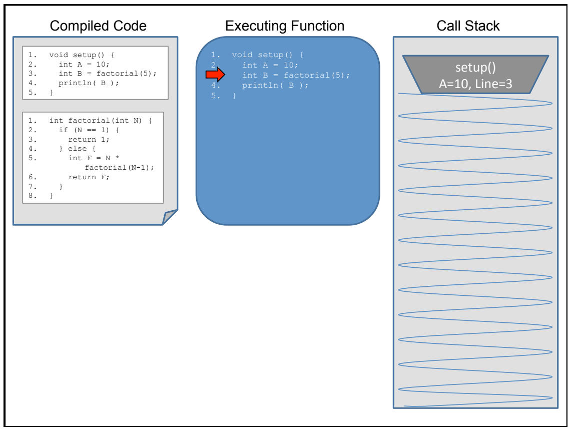
```
void setup() {
  int A = 10;
  int B = factorial(5);
  println( B );
}
```

```
int factorial(int N) {
  if (N == 1) {
    return 1;
  } else {
    int F = N *
        factorial(N-1);
    return F;
  }
}
```

Executing Function

```
void setup() {
  int A = 10;
  int B = factorial(5);
  println( B );
}
```

Call Stack


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

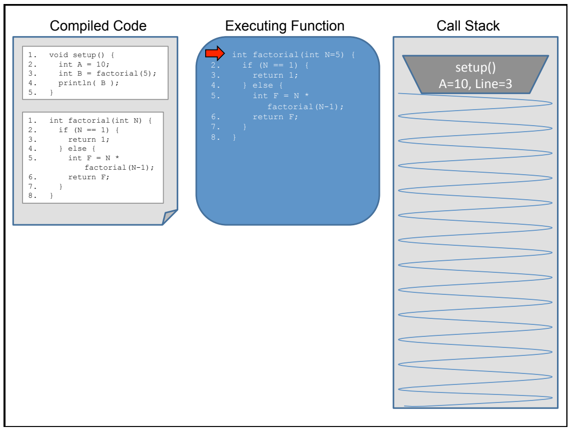
```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

Call Stack

setup()
A=10, Line=3


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

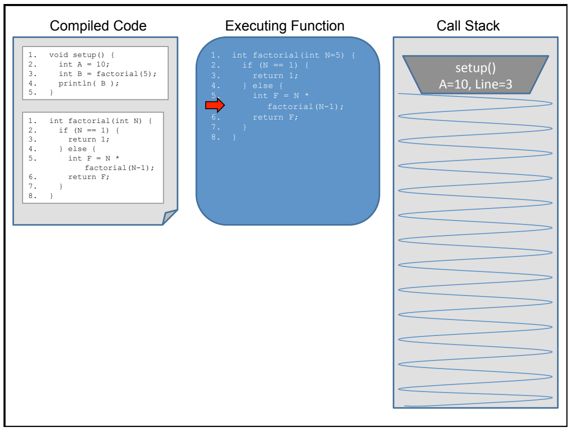
```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
     int factorial(int N=5) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Call Stack

setup()
A=10, Line=3


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

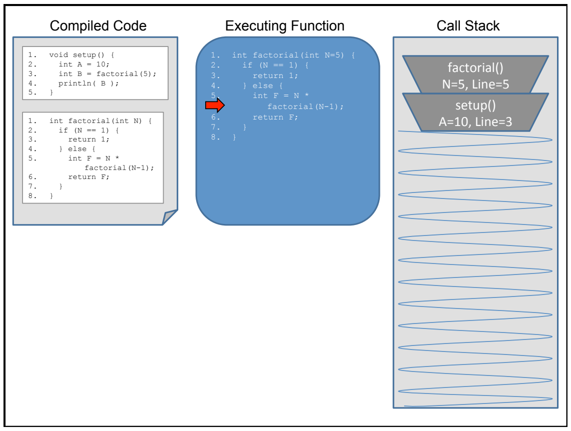
```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int N=5) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Call Stack

setup()
A=10, Line=3


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

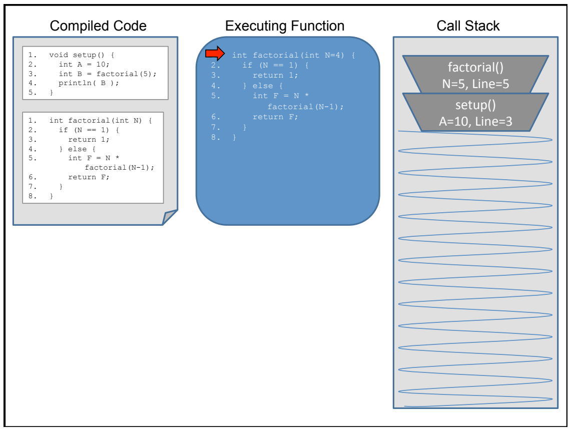
Executing Function

```
1.   int factorial(int N=5) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Call Stack

factorial()
N=5, Line=5

setup()
A=10, Line=3


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

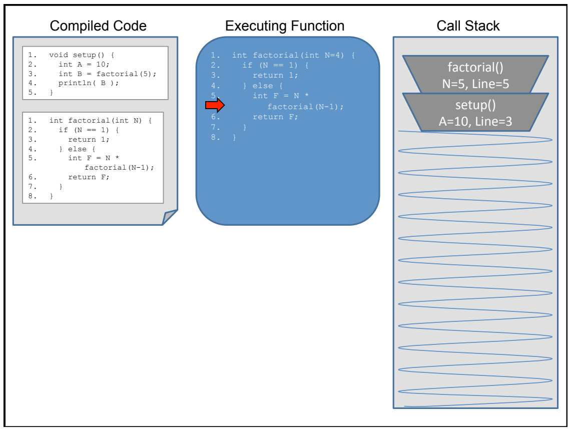
Executing Function

```
     int factorial(int N=4) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Call Stack

factorial()
N=5, Line=5

setup()
A=10, Line=3


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

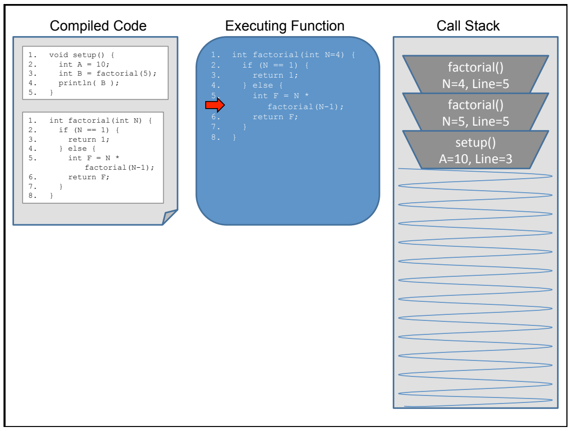
Executing Function

```
1.   int factorial(int N=4) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Call Stack

factorial()
N=5, Line=5

setup()
A=10, Line=3


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

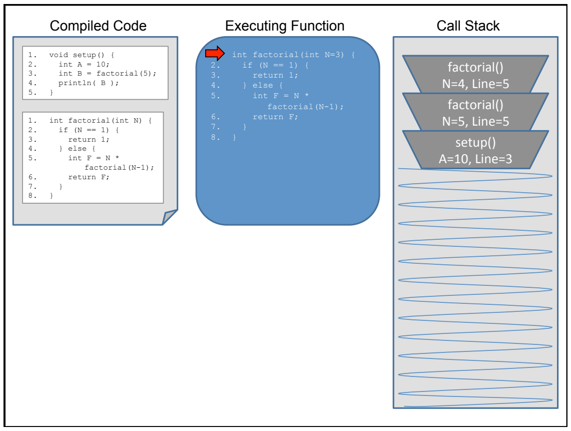
```
1.   int factorial(int N=4) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

Call Stack

factorial()
N=4, Line=5

factorial()
N=5, Line=5

setup()
A=10, Line=3


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

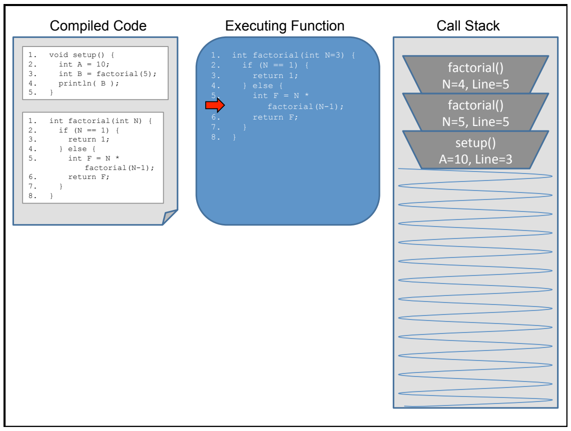
```
1.   int factorial(int N=3) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

Call Stack

factorial()
N=4, Line=5

factorial()
N=5, Line=5

setup()
A=10, Line=3


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

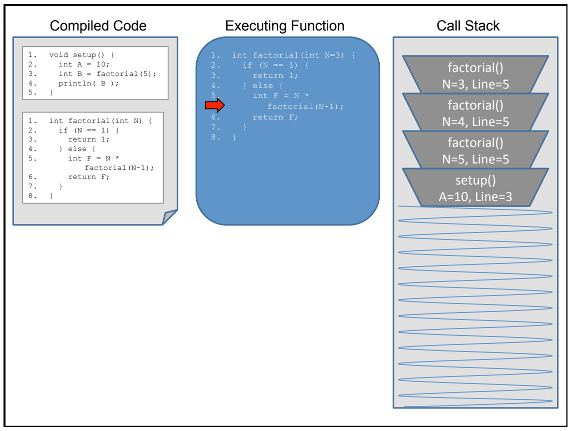
```
1.   int factorial(int N=3) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

Call Stack

factorial()
N=4, Line=5

factorial()
N=5, Line=5

setup()
A=10, Line=3


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int N=3) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

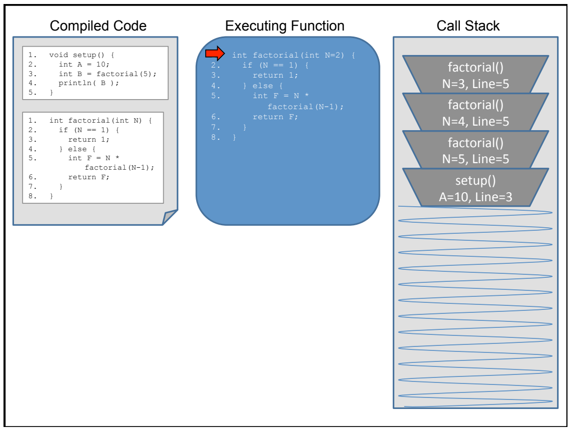
Call Stack

factorial()
N=3, Line=5

factorial()
N=4, Line=5

factorial()
N=5, Line=5

setup()
A=10, Line=3


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int N=2) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

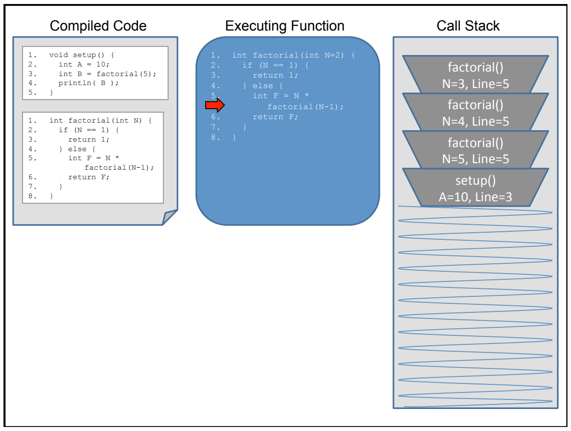
Call Stack

factorial()
N=3, Line=5

factorial()
N=4, Line=5

factorial()
N=5, Line=5

setup()
A=10, Line=3


Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
             factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int N=2) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

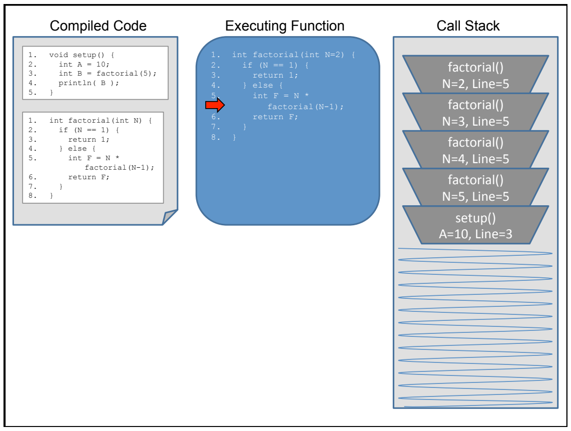
Call Stack

factorial()
N=3, Line=5

factorial()
N=4, Line=5

factorial()
N=5, Line=5

setup()
A=10, Line=3



Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int N=2) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

Call Stack

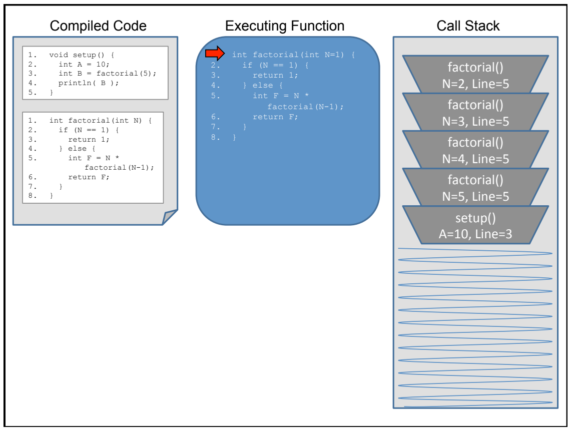
factorial()
N=2, Line=5

factorial()
N=3, Line=5

factorial()
N=4, Line=5

factorial()
N=5, Line=5

setup()
A=10, Line=3



Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
     int factorial(int N=1) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

Call Stack

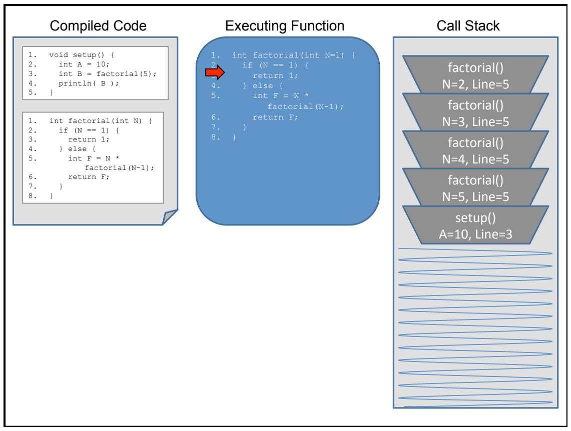
factorial()
N=2, Line=5

factorial()
N=3, Line=5

factorial()
N=4, Line=5

factorial()
N=5, Line=5

setup()
A=10, Line=3



Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int N=1) {
2.     if (N == 1) {
         return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

Call Stack

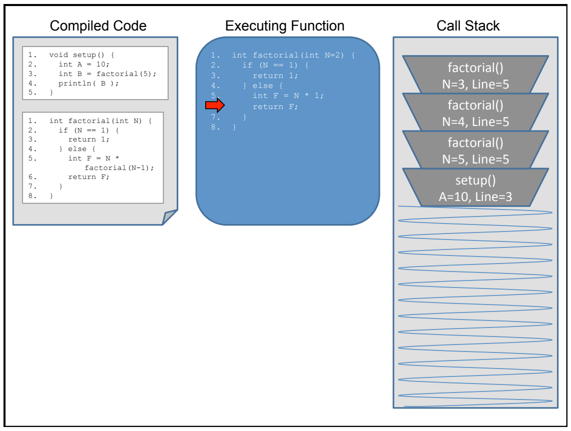
factorial()
N=2, Line=5

factorial()
N=3, Line=5

factorial()
N=4, Line=5

factorial()
N=5, Line=5

setup()
A=10, Line=3



Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

```
1.   int factorial(int N=2) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N * 1;
         return F;
7.     }
8.   }
```

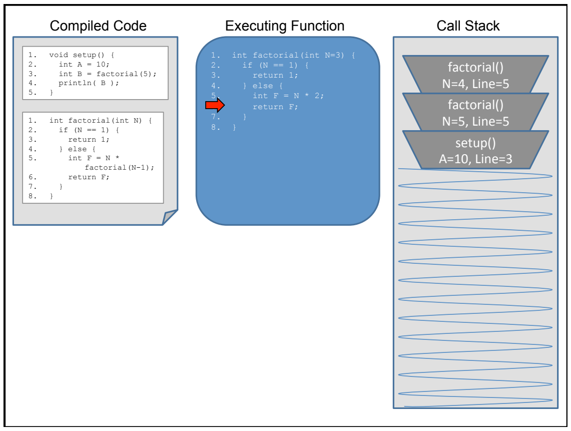
Call Stack

factorial()
N=3, Line=5

factorial()
N=4, Line=5

factorial()
N=5, Line=5

setup()
A=10, Line=3



Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

Executing Function

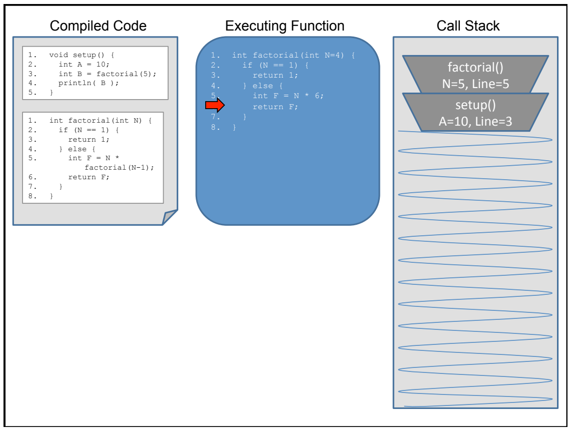
```
1.   int factorial(int N=3) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N * 2;
         return F;
7.     }
8.   }
```

Call Stack

factorial()
N=4, Line=5

factorial()
N=5, Line=5

setup()
A=10, Line=3



Compiled Code

```
1.   void setup() {
2.     int A = 10;
3.     int B = factorial(5);
4.     println( B );
5.   }
```

```
1.   int factorial(int N) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N *
            factorial(N-1);
6.       return F;
7.     }
8.   }
```

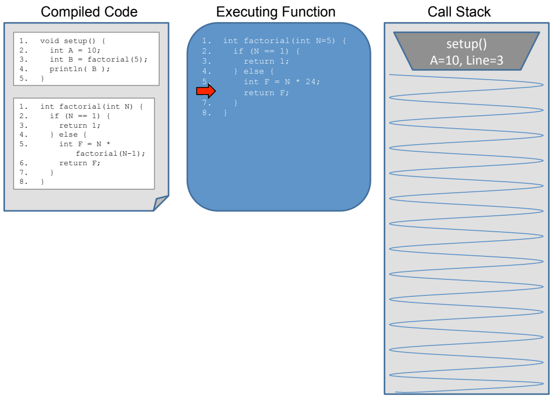
Executing Function

```
1.   int factorial(int N=4) {
2.     if (N == 1) {
3.       return 1;
4.     } else {
5.       int F = N * 6;
         return F;
7.     }
8.   }
```

Call Stack

factorial()
N=5, Line=5

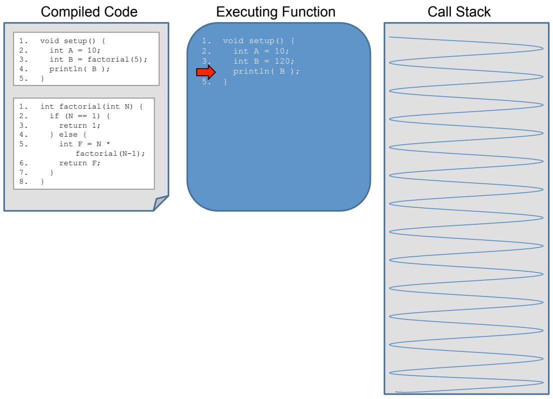
setup()
A=10, Line=3



```
1.  void setup() {
2.    int A = 10;
3.    int B = factorial(5);
4.    println( B );
5.  }
```

```
1.  int factorial(int N) {
2.    if (N == 1) {
3.      return 1;
4.    } else {
5.      int F = N *
          factorial(N-1);
6.      return F;
7.    }
8.  }
```

```
1.  int factorial(int N=5) {
2.    if (N == 1) {
3.      return 1;
4.    } else {
5.      int F = N * 24;
        return F;
7.    }
8.  }
```



```
1.  void setup() {
2.    int A = 10;
3.    int B = factorial(5);
4.    println( B );
5.  }
```

```
1.  int factorial(int N) {
2.    if (N == 1) {
3.      return 1;
4.    } else {
5.      int F = N *
          factorial(N-1);
6.      return F;
7.    }
8.  }
```

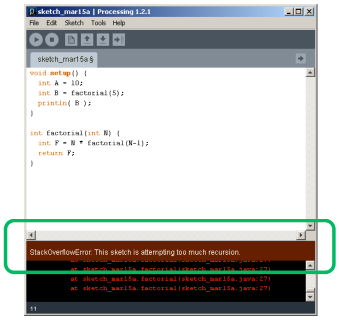
```
1.  void setup() {
2.    int A = 10;
3.    int B = 120;
      println( B );
5.  }
```

Call Stack

The Call Stack keeps track of …

1. all functions that are suspended, in order
2. the point in the function where execution should resume after the invoked subordinate function returns
3. a snapshot of all variables and values within the scope of the suspended function so these can be restored upon continuing execution

What happens if there is no stopping condition, or "base case"?



```
void setup() {
  int A = 10;
  int B = factorial(5);
  println( B );
}

int factorial(int N) {
  int F = N * factorial(N-1);
  return F;
}
```

```
StackOverflowError: This sketch is attempting too much recursion.
        at sketch_mar15a.factorial(sketch_mar15a.java:27)
        at sketch_mar15a.factorial(sketch_mar15a.java:27)
        at sketch_mar15a.factorial(sketch_mar15a.java:27)
        at sketch_mar15a.factorial(sketch_mar15a.java:27)
```

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);
  }
}
```

Examples

- recursiveCircles
- recursiveTree
- recursiveTreeTransform