Review

Exam 2 Study Guide Pg. 1 of 5

- Arrays (of any type: `boolean, char, int, float, PImage, String, PVector`, or an arbitrary Class)
  - declare
  - instantiate
  - initialize all values
  - get the size
  - assign a value to an arbitrary location
  - get a value from an arbitrary location
  - iterate through any number (1 to array.length) of elements of an array starting from any valid index.
  - compare a value in an array with another value
  - compare two values in an array
  - use built in functions on arrays such as max(), min(), etc.

+ Exam 2 Study Guide Pg. 2 of 5

- ArrayLists (of any type: `boolean, char, int, float, PImage, String, PVector,` or an arbitrary Class)
  - declare
  - instantiate
  - initialize all values
  - get the size
  - assign a value to an arbitrary location
  - get a value from an arbitrary location
  - iterate through any number (1 to arrayList.size()) of elements of an ArrayList starting from any valid index.
  - compare a value in an ArrayList with another value
  - compare two values in an ArrayList

+ Exam 2 Study Guide Pg. 3 of 5

- String/PVector/PImage/(arbitrary class defined in test)*
  - declare
  - instantiate (using constructor)
  - using keyword `this` *
  - identify*/use/assign fields
  - identify*/call methods
  - compare the values of 2 instances of the class using .equals()
  - use Processing methods and operators on Strings●
  - write the contents of (fill in) a method based on description*

* just for arbitrary defined class.

● just for Strings.

Exam 2 Study Guide Pg. 4 of 5

- loops
  - read a loop and identify
    - the number of times the loop will iterate and/or the reason the loop will stop
    - the values of each variable used in the loop at each iteration and after the loop has ended
    - the output of the loop (if anything is printed or displayed)
    - Ex:

      ```
      int[] test = { 3 , 7, 11 };
      int c = 10;
      for (int i = 0; i < test.length; ++i) {
        c += i;
        test[i] = test[i] * test[i];
        c *= (i+1);
      } // Note: the variables in the loop
        // are c, i, and test
      ```

Exam 2 Study Guide Pg. 5 of 5

- functions/methods
  - identify name/parameters/return type
  - write a function from a description

- Processing functions (you should feel comfortable with these)
  - print(), println(), loadStrings(), color(), random(), abs(), ceil(), dist(), floor(),pow(),round(),sq(),sqrt(), cos(),sin(),tan()

- Recursion
  - given a base case(s) and a recursive case, write the recursive function. For example: write a recursive function fib, that returns an int
    base cases: fib(1) = 1, fib(2) = 1
    recursive case: fib(n) = fib(n-2) + fib(n-1)

- Be comfortable with material from Exam 1.

Exam 2 Review
Objects, Arrays, Strings, Recursion, (Inheritence)

Objects

- Defined by template given in as class statement.
- An object is created by invoking the class's constructor using the new keyword.
- An objects is stored in a variable declared with class as type
- Values passed to a constructor must be copied to object fields to "stick”

```
Tree myMaple;  // Variable defined as type Tree

void setup() {
  myMaple = new Tree("maple", 30.3);   // Create
}

class Tree {
```

fields

```
  String name;
  float height;
```

constructor

```
  Tree( String tname, float theight) {
    name = tname;
    height = theight;
  }
```

method

```
  void draw() {
    fill( 0, 255, 0 );
    ellipse(random(width),random(height),50,50);
  }
}
```

Creating Objects

1. Declare a variable with the class as type
2. Invoke the constructor using the new keyword and assign to variable

```
Tree myMaple;           // Variable defined as type Tree

myMaple = new Tree("maple", 30.3);   // Create and assign


// -----

// Two steps combined in one
Tree myMaple = new Tree("maple", 30.3);
```

Creating Objects

- What is wrong with this?

```
Tree myMaple;          // Variable defined as type Tree

void setup() {
  Tree myMaple = new Tree("maple", 30.3);   // Combined
}
```

Using Objects

- variable :: fields (variable inside an object)
- function :: method (function inside an object)

- An variable that stores an object is used to scope access to the fields and methods of that particular object

Using Objects

```
Tree myMaple;

void setup() {
  myMaple = new Tree("maple", 30.3);
}

void draw() {
  myMaple.draw();
}

class Tree {
  String name;
  float height;

  Tree( String tname, float theight) {
    name = tname;
    height = theight;
  }

  void draw() {
    fill( 0, 255, 0 );
    rect( 10, 10, 50, 300 );
  }

}
```

Using Objects

What is wrong with this?

```
Tree myMaple;

void setup() {
  myMaple = new Tree("maple", 30.3);
}

void draw() {
  Tree.draw();
}

class Tree {
  String name;
  float height;

  Tree( String tname, float theight) {
    name = tname;
    height = theight;
  }

  void draw() {
    fill( 0, 255, 0 );
    rect( 10, 10, 50, 300 );
  }

}
```

Arrays - Creating

- A structure that can hold multiple items of a common data type
- Arrays can hold any data type, including objects
- The data type to be held by an array must be declared as part of the array declaration
- Arrays are themselves a kind of type, which is made by adding brackets to the type that the array can hold

Arrays – Creating and Init'ng (3 Steps)

1. Declare an array variable
   - The variable is NOT an array
2. Create an array and assign it to the variable
   - Use the new keyword and size
   - The array is filled with default values
     - int <- 0
     - float <- 0.0
     - boolean <- false;
     - any object including String <- null
3. Fill the array with items of appropriate type

```
Tree[] trees;
```

Step 1

trees ← No array. Only a variable that can hold an array.

```
Tree[] trees;
trees = new Tree[5];
```

Step 2

	trees
0	null
1	null
2	null
3	null
4	null

← An empty array. null Tree objects.

Step 3

```
Tree[] trees;
trees = new Tree[5];
trees[0] = new Tree("maple", 20.0);
trees[1] = new Tree("oak", 203.4);
```

	trees

0	name="maple";
	height=20.0;

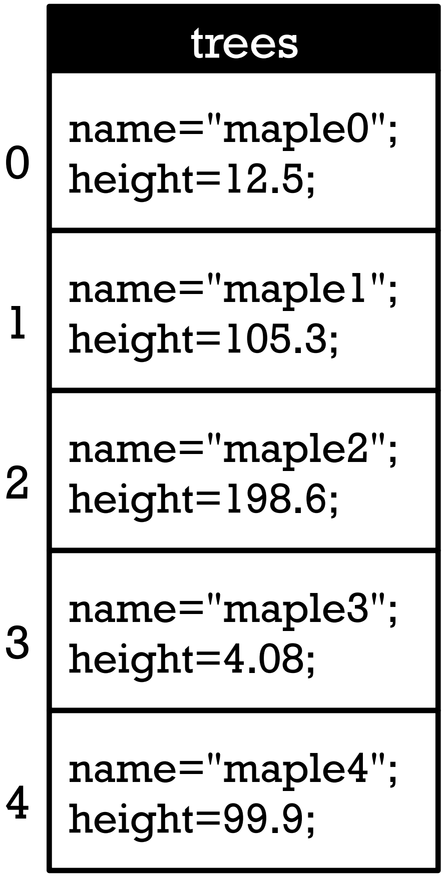
1	name="oak";
	height=203.4;

2	null
3	null
4	null

← An array with two Tree objects.

Step 3

```
Tree[] trees;
trees = new Tree[5];
for (int i=0; i<5; i++) {
  trees[i] = new Tree( "maple"+i, random(200.0) );
}
```



trees

0 name="maple0"; height=12.5;

1 name="maple1"; height=105.3;

2 name="maple2"; height=198.6;

3 name="maple3"; height=4.08;

4 name="maple4"; height=99.9;

← An array with five Tree objects.

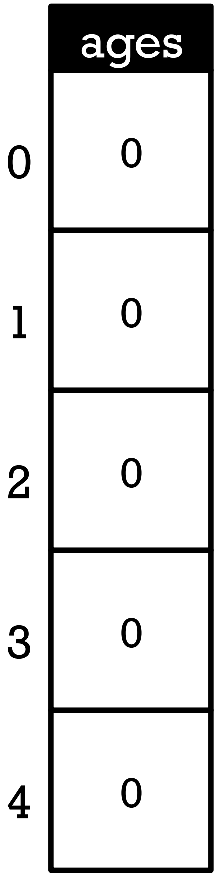
```
int[] ages;
```

Step 1

ages ← No array. Only a variable that can hold an array.

```
int[] ages;
ages = new int[5];
```

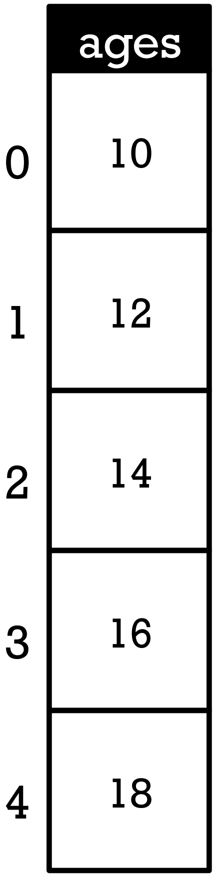
Step 2



← An empty array. Default ints (0).

Step 3

```
int[] ages;
ages = new int[5];
for (int i=0; i<5; i++) {
  ages[i] = 10 + 2*i;
}
```



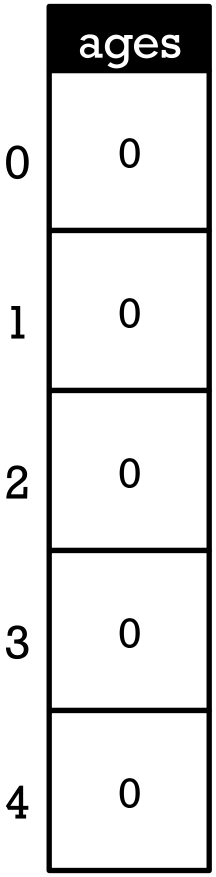
	ages
0	10
1	12
2	14
3	16
4	18

🡐 An array with five integers.

Step 1+2

```
int[] ages = new int[5];

// Same as
// int[] ages;
// ages = new int[5];
```



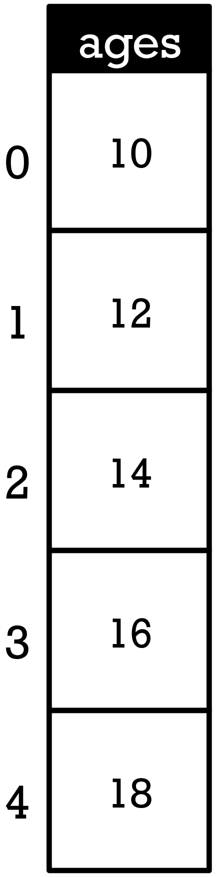
	ages
0	0
1	0
2	0
3	0
4	0

🡐 An empty array. Default ints (0).

```
int[] ages = new int[] {10, 12, 14, 16, 18};

// Same as
// int[] ages = new int[5];
// for (int i=0; i<5; i++) { ages[i] = 10 + 2*i; }
```

Step 1+2+3



	ages
0	10
1	12
2	14
3	16
4	18

← An array with five integers.

Arrays – Using

- An item in an array is accessed by following an array variable with square brackets containing the item number (index)

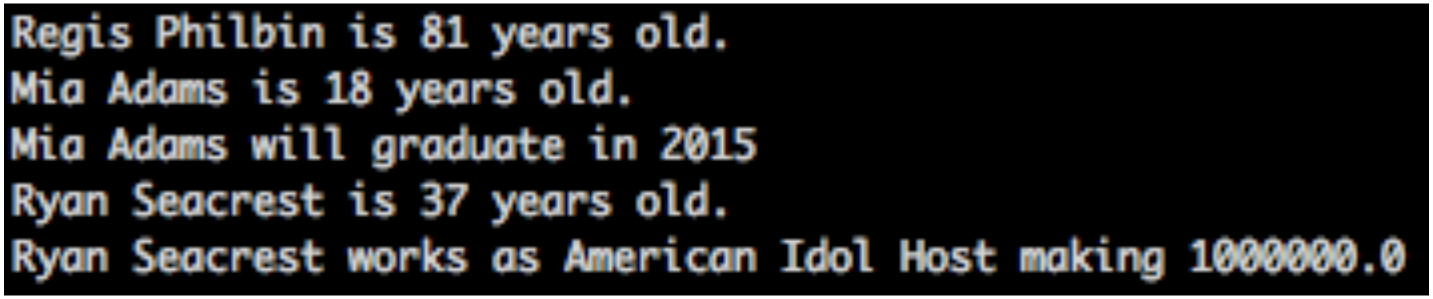
- Array indexes start with 0

- Once accessed with brackets, the result can be used as if it was the item at the location in the array

```
Person[] people;

void setup() {
  people = new Person[3];
  people[0] = new Person("Regis Philbin", 81);
  people[1] = new Student("Mia Adams", 2015);
  Employee rs = new Employee("Ryan Seacrest", 37);
  rs.hire("American Idol Host", 1000000.0);
  people[2] = rs;

  for (int i=0; i<people.length; i++ ) {
    people[i].pr();
    people[i].stats();
  }
}
```



```
Regis Philbin is 81 years old.
Mia Adams is 18 years old.
Mia Adams will graduate in 2015
Ryan Seacrest is 37 years old.
Ryan Seacrest works as American Idol Host making 1000000.0
```

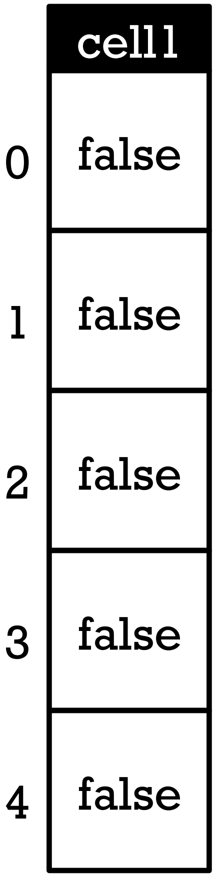
Arrays of arrays (2D Arrays)

- If an array can be made of any type by adding brackets, and …
- an array is a kind of type, then …
- an array of arrays should be possible by adding a second set of brackets

```
boolean[] cell1;     // A variable that holds an array of booleans

boolean[][] cell2;   // A variable that holds an array of
                     // boolean arrays
```

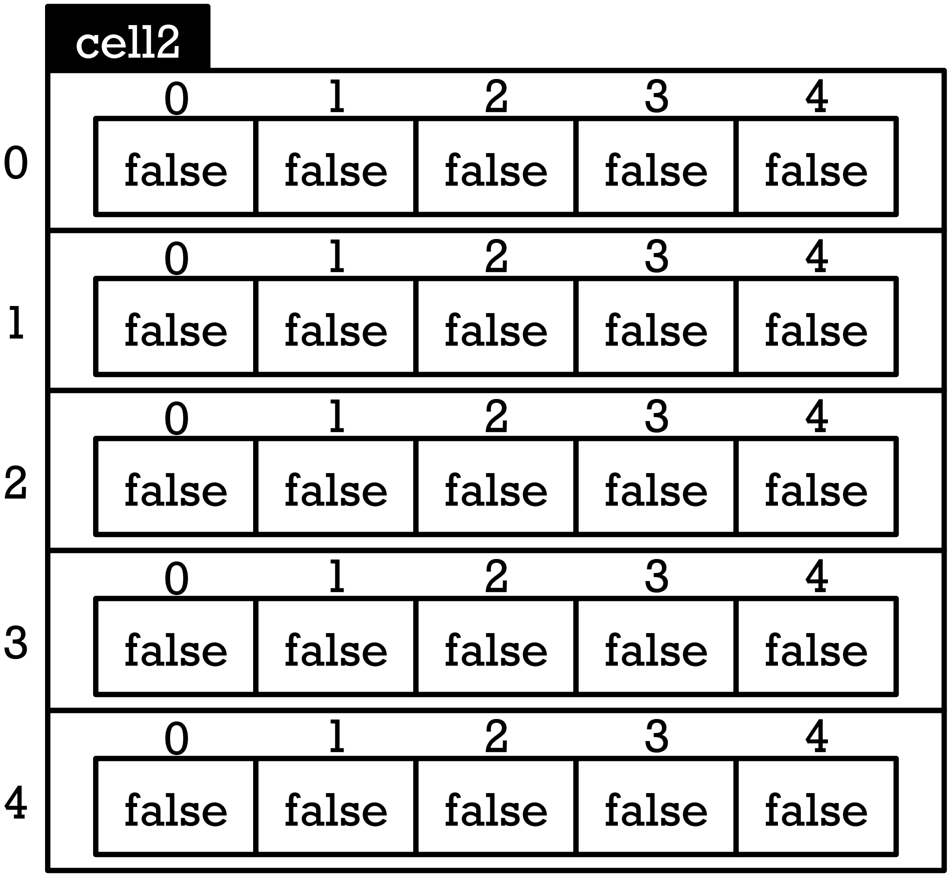
```
boolean[] cell1;
cell1 = new boolean[5];
```



	cell1
0	false
1	false
2	false
3	false
4	false

← One-dimensional array

```
boolean[][] cell2;
cell2 = new boolean[5][5];
```



cell2

	0	1	2	3	4
0	false	false	false	false	false
1	false	false	false	false	false
2	false	false	false	false	false
3	false	false	false	false	false
4	false	false	false	false	false

← Two-dimensional array

… an array of arrays

```
boolean[][] cell2;
cell2 = new boolean[5][5];

cell2[1][2] = true;
```

cell2	0	1	2	3	4
0	false	false	false	false	false
1	false	false	**true**	false	false
2	false	false	false	false	false
3	false	false	false	false	false
4	false	false	false	false	false

Proving a 2D array is an array of arrays

- Access fields and methods of top-level array

```
void setup() {

  boolean[][] cell2;
  cell2 = new boolean[5][5];   // Create array of arrays

  println( cell2[0].length );  // Access array

  cell2[1][2] = true;          // Access array in array
  println( cell2[1] );         // Access array
}
```

```
5
[0] false
[1] false
[2] true
[3] false
[4] false
```

Proving a 2D array is an array of arrays

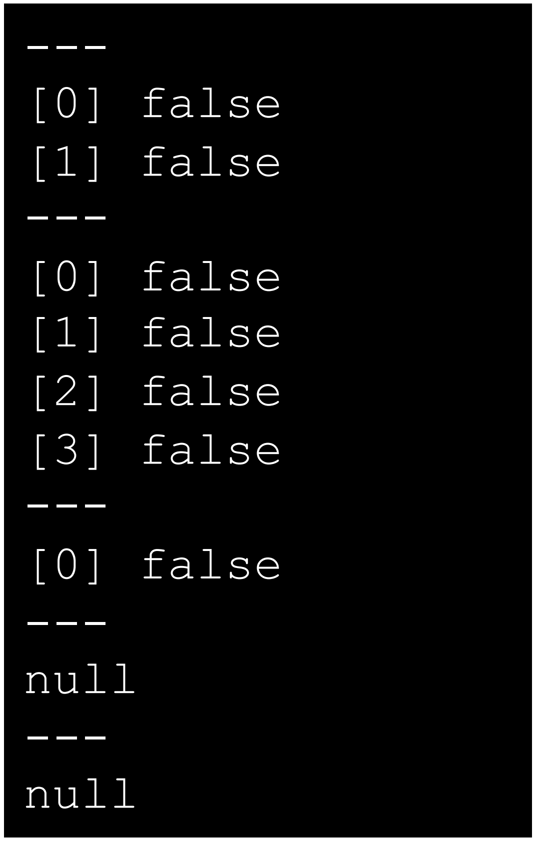
- Build a "ragged array"

```
void setup() {

  boolean[][] cell2;
  cell2 = new boolean[5][];

  cell2[0] = new boolean[2];
  cell2[1] = new boolean[4];
  cell2[2] = new boolean[1];

  println("---");
  println(cell2[0]);
  println("---");
  println(cell2[1]);
  println("---");
  println(cell2[2]);
  println("---");
  println(cell2[3]);
  println("---");
  println(cell2[4]);
}
```



```
---
[0] false
[1] false
---
[0] false
[1] false
[2] false
[3] false
---
[0] false
---
null
---
null
```

Making Strings

- Declaring String objects with no chars

```
String myName;
String myName = new String();
```

- Declaring String objects init'd w/ char array

```
String myName = "Fred";
String myName = new String("Fred");
```

String class methods

- `charAt(`*`index`*`)`
  - Returns the character at the specified index
- `equals(`*`anotherString`*`)`
  - Compares a string to a specified object
- `equalsIgnoreCase(`*`anotherString`*`)`
  - S/A ignoring case (i.e. 'A' == 'a')
- `indexOf(`*`char`*`)`
  - Returns the index value of the first occurrence of a character within the input string
- `length()`
  - Returns the number of characters in the input string
- `substring(`*`startIndex, endIndex`*`)`
  - Returns a new string that is part of the input string
- `toLowerCase()`
  - Converts all the characters to lower case
- `toUpperCase()`
  - Converts all the characters to upper case
- `concat(`*`anotherString`*`)`
  - Concatenates String with anotherString

http://docs.oracle.com/javase/7/docs/api/

Built-in String functions (not methods)

split(*bigString, splitChar*)

- Breaks a String into a String Array, splitting on splitChar
- Returns new String Array

splitTokens(*bigString, splitCharString*)

- Breaks a String into a String Array, splitting on any char in splitCharString

join(*stringArray, joinChar*)

- Builds a new String by concatenating all Strings in stringArray, placing joinChar between each
- Inverse of split() function

text(*theString, x, y*)

text(*theString, x, y, width, height*)

- Draws *theString* on the sketch at (x, y)

Given the commands:

```
String aPalindrome = "a man,a plan,a canal Panama";
String[] strs = splitTokens(aPalindrome, ",");
```

Answer the following questions:

(3 pts) What will be the length of strs?

a) 1
b) 2
c) 3
d) 4

(3 pts) What will be the value of strs[1]?

a) "a man"
b) "a plan"
c) "a canal Panama"
d) 3

(3 pts) Write the expression used to obtain the number of elements in strs.

The following program was designed to count and print the number of duplicates in the myArray String array. Unfortunately, it doesn't work properly. When I test it with the given data, it tells me that I have 11 duplicates, but I know that there are only two. Fix the program so that it works correctly.

```
// Count and print the number of duplicate strings in myArray

String [] myArray = {"A", "B", "C", "D", "A", "F", "C"};

void setup() {
  int count = 0;

  for (int i=0; i<myArray.length; i++) {
    for (int j=0; j<myArray.length; j++) {
      if (myArray[i].equals( myArray[j] )) {
        count++;
      }
    }
  }

  println("There are " + count + " duplicates.");
}
```

Recursion

Factorial

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

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

$5! = 5 \times 4!$

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

Factorial can be defined in terms of itself

$4! = 4 \times 3 \times 2 \times 1 = 24$

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

$5! = 5 \times 4!$

$4! = 4 \times 3!$

$3! = 3 \times 2!$

$2! = 2 \times 1$

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	**i**	**fim1**	**returns**
factorial(10)	10		

Last In First Out (LIFO) Stack of Plates

Call Stack

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	**i**	**fim1**	**returns**
factorial(10)	10		
factorial(9)	9		

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10		
factorial(9)	9		
factorial(8)	8		

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10		
factorial(9)	9		
factorial(8)	8		
factorial(7)	7		
factorial(6)	6		
factorial(5)	5		
factorial(4)	4		
factorial(3)	3		
factorial(2)	2		
factorial(1)	1		
factorial(0)	0	--	1

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10		
factorial(9)	9		
factorial(8)	8		
factorial(7)	7		
factorial(6)	6		
factorial(5)	5		
factorial(4)	4		
factorial(3)	3		
factorial(2)	2		
factorial(1)	1	1	1
factorial(0)	0	--	1

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10		
factorial(9)	9		
factorial(8)	8		
factorial(7)	7		
factorial(6)	6		
factorial(5)	5		
factorial(4)	4		
factorial(3)	3		
factorial(2)	2	1	2
factorial(1)	1	1	1
factorial(0)	0	--	1

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	**i**	**fim1**	**returns**
factorial(10)	10		
factorial(9)	9		
factorial(8)	8		
factorial(7)	7		
factorial(6)	6		
factorial(5)	5		
factorial(4)	4		
factorial(3)	3	2	6
factorial(2)	2	1	2
factorial(1)	1	1	1
factorial(0)	0	--	1

```
// Compute factorial of a number
// Recursive implementation

void setup() {}
void draw() {}

void mousePressed() {
  int f = factorial(10);
  println(f);
}

int factorial( int i) {
  if( i == 0) {
    return 1;
  } else {
    int fim1 = factorial(i-1);
    return i*fim1;
  }
}
```

Call	i	fim1	returns
factorial(10)	10	362880	3628800
factorial(9)	9	40320	362880
factorial(8)	8	5040	40320
factorial(7)	7	720	5040
factorial(6)	6	120	720
factorial(5)	5	24	120
factorial(4)	4	6	24
factorial(3)	3	2	6
factorial(2)	2	1	2
factorial(1)	1	1	1
factorial(0)	0	--	1

```
String[] parts = new String[] {"a", "b", "c", "d"};

void setup() {}
void draw() {}

void mousePressed() {
  String joined = reverseJoin(3);
  println( joined );
}

String reverseJoin( int i ) {
  if (i == 0) {
    return parts[0];
  }
  else {
    String rjim1 = reverseJoin(i-1)
    return parts[i] + rjim1;
  }
}
```

Call	i	parts[i]	rjim1	returns
reverseJoin(3)	3	"d"	"cba"	"dcba"
reverseJoin(2)	2	"c"	"ba"	"cba"
reverseJoin(1)	1	"b"	"a"	"ba"
reverseJoin(0)	0	"a"	--	"a"

Inheritance

- Superclass (base class) – higher in the hierarchy
- Subclass (child class) – lower in the hierarchy
- A subclass is derived from from a superclass
- Subclasses inherit all the fields and methods of their superclass
- Subclasses can override a superclass method by redefining it.
  - They can replace anything by redefining locally

```
class Person {
  String name; int age;

  Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
  void pr() {
    println(name + " is " + age + " years old.");
  }
  void stats() {}
}


class Student extends Person {
  int year; float GPA;

  Student(String name, int year, float GPA) {
    super(name, 18);
    this.year = year;
    this.GPA = GPA;
  }

  void stats() {
    println(name + " will graduate in " + year);
  }
}
```

```
class Employee extends Person {
  float salary; String position; boolean current;

  Employee(String name, int age) {
    super(name, age);
  }

  void hire(String position, float salary) {
    this.position = position;
    this.salary = salary;
    current = true;
  }

  void fire() {
    current = false;
  }

  void stats() {
    if (current) {
      println(name + " works as " + position + " making " + salary);
    }
    else {
      println(name + " is not working for us.");
    }
  }
}
```