Review

- text(…)
- textSize(…), textAlign(…), fill(…)
- Custom functions
  - Declaring
  - Call and passing arguments
  - Returning a value
- Modularity and Reuse

One at a time …

- In Processing, only one statement is executed at a time
  - Execution order:
    1. Global declarations and initializations
    2. Setup() called once
    3. Draw() called repeatedly, (unless noLoop() is invoked)
    4. If mouse is pressed, mousePressed() is called, between calls to Draw()
    5. Also for other events, such as keyPressed() …
  - If a function is called (custom or built-in) the calling function suspends until the called function completes.

```
void setup() {
  println("Calling function1");
  function1();
  println("function1 complete");
}

void function1() {
  println("Executing function1");
}
```

```
Calling function1
Executing function1
function1 complete
```

Scope

- An enclosing context in a program where values and expressions are associated.
- A way to separate variables in different parts of your program from one another.
- To a first approximation, the scope of a section of your code is demarcated by { and }.

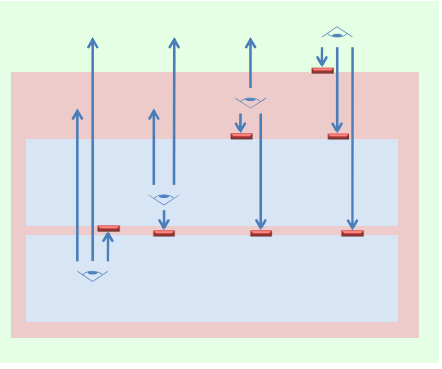
Kinds of Scope (that we've seen thus far)

- Global
- Function
- Block

```
// Global scope.
// Variables declared here are accessible by all.
int x1 = 10;

void setup() {
  // Inside setup function scope.
  int x2 = 20;

  for (int i=0; i<10; i++) {
    // Inside the scope of the for-loop
    // 'x2' is accessible here
    // 'i' is not accessible outside the block
  }

  if ( x2 > 10 ) {
    String s = "blah";
    // Inside the scope of the if-statement
    // 's' is only accessible within this block
  }
}
```

Scope and Variable Access Rules

1. A variable declared within a given scope (global, function, block) is accessible (read, write) from within that scope, as well as all nested (inner) scopes.
2. A variable declared in an inner (nested) scope cannot be accessed from code executing in an outer scope or an adjacent scope.
3. When a variable name is accessed from code, the local scope is checked for the variable first. If it is not found, the next outer (containing) scope is checked for the variable. This continues until all outer scopes are searched, in order.



Lifetime, of a variable

1. Variables cannot be accessed before they are declared.
2. A variable is destroyed when a program exits the block in which it was declared.
3. Variables can be declared in…
   - the global scope
   - the body of a function
   - the arguments of a function
   - A block (for, while, if, …).

Shadowing

- If a variable is declared within an inner (nested) scope has the same name as a variable declared in an outer scope, the inner-scope variable "shadows" (hides) the outer-scope variable. The inner declared variable is a distinct variable with the same name. It does not replace the outer variable or change it in any way.

* Note: All rules apply to variables of any type: int, float, String, boolean, ...

1

```
int v1 = 10;
int v2 = v1 + 1;
println("v1=" + v1);
println("v2=" + v2);
```

2

```
int v1 = 10;
int v2;

void setup()
{
  v2 = v1 + 1;

  println("v1=" + v1);
  println("v2=" + v2);
}
```

3

```
int v1 = 10;

void setup()
{
  int v2;
  v2 = v1 + 1;

  println("v1=" + v1);
  println("v2=" + v2);
}
```

4

```
int v1 = 1;

void setup()
{
  int v2 = 2;
  function1();
}

void function1()
{
  println("v1=" + v1);
  println("v2=" + v2);  // !!!
}
```

5

```
int v1 = 1;

void setup()
{
  int v2 = 2;
  function1(v2);
}

void function1(int v2)
{
  println("v1=" + v1);
  println("v2=" + v2);
}
```

7

```
int v1 = 1;

void setup()
{
  int v2 = 2;
  function1(v2);
}

void function1(int v3)
{
  println("v1=" + v1);
  println("v3=" + v3);
}
```

6

```
int v1 = 1;
int v2 = 2;

void setup()
{
  function1();
  function2();
}

void function1()
{
  println("v1=" + v1);
}

void function2()
{
  println("v2=" + v2);
}
```

8

```
int v1 = 1;

void setup()
{
  int v2 = 2;
  function1(v2);
  println("(in setup) v2=" + v2);
}

void function1(int v2)
{
  v2 = 234;
  println("(in function1) v2=" + v2);
}
```

9

```
int v1 = 1;

void setup()
{
  function1();
  function2();
}

void function1()
{
  int v2 = 2;
  println("v1=" + v1);
}

void function2()
{
  println("v2=" + v2);
}
```

10

```
int v1 = 1;

void setup()
{
  int v2 = 0;

  while( v2 < 5 )
  {
    println("v2=" + v2);
    v2++;
  }

  println("(after while) v2=" + v2);

}
```

11

```
int v1 = 1;

void setup()
{

  for( int v2=0; v2<5; v2++ )
  {
    println("v2=" + v2);
  }

  println("(after for) v2=" + v2);
}
```

12

```
int v1 = 1;

void setup()
{
  int v2;

  for( v2=0; v2<5; v2++ ) {
    println("v2=" + v2);
  }

  println("(after for) v2=" + v2);
}
```

13

```
int v1 = 1;

void setup()
{
  int v2 = 2;
  function1();
}

void function1()
{
  println("(before for) v1=" + v1);

  for( v1=0; v1<5; v1++ ) {
    println("v1=" + v1);
  }

  println("(after for) v1=" + v1);
}
```

14

```
int v1 = 1;
int v2 = 2;

void setup()
{
  function1();
}

void function1()
{
  int v2 = 3;

  println("(before for) v2=" + v2);

  for( int v2=0; v2<5; v2++ ) {   // !!!
    println("v2=" + v2);
  }

  println("(after for) v2=" + v2);
}
```

15

```
void setup()
{
  for( int i=0; i<5; i++) {
    println("(loop 1) " + i);
  }

  for( int i=0; i<5; i++) {
    println("(loop 2) " + i);
  }
}
```

16

```
void setup()
{
  int v1 = 1;

  {
    int v2 = 2;
    println("(block 1) v2=" + v2);
  }

  {
    int v2 = 3;
    println("(block 2) v2=" + v2);
  }

  println("(setup) v2=" + v2);  //!!!
}
```