Review

- Primitive Shapes
  - point
  - line
  - triangle
  - quad
  - rect
  - ellipse
- Processing Canvas
- Coordinate System
- Shape Formatting
  - Colors
  - Stroke
  - Fill

Review

- Random numbers
- mouseX, mouseY
- setup() & draw()
- frameRate(), loop(), noLoop()
- Mouse and Keyboard interaction
- Arcs, curves, bézier curves, custom shapes
- Red-Green-Blue color w, w/o alpha

Review

- Drawing Images
- Variables
- Variable types
- Integer division
- Conditionals: if - else if - else
- Motion simulation

Review

- Expressions and operators
- Iteration
  - while-loop
  - for-loop
- Loops
  - Condition
  - Index
- Functions
- Definition
- Call
- Parameters
- Return value

Execution

Write a Processing sketch that draws a red circle inside a white circle inside a black circle with all circles having a black border. All of the circles should have the same center point at the center of the sketch regardless of the sketch size. The white circle should have twice the diameter of the red circle. The black circle should have 3 times the diameter of the red circle.

Here is an example:

- Statements are executed one at a time in the order written
Execution

- Statements are executed one at a time in the order written.

- Execution order
  - Globals and initializations
  - setup() called once
  - draw() called repeatedly
  - If any mouse or keyboard events occur, the corresponding functions are called between calls to draw() – exact timing cannot be guaranteed.

Conditionals: if-statement

Programmatic branching ...

```java
if ( boolean_expression ) {
    statements;
}
```

// What does this do?
void draw() {
    if ( mouseX > 50 && mouseY > 50 ) {
        ellipse( mouseX, mouseY, 10, 10 );
    }
}
```

If – else if - else

- Pay close attention to the opening brace, { that starts a block and the closing brace, } that ends a block.
- Pay close attention to which if connects to which else.
- Read the code, one if statement at a time to make a decision tree based on the conditional statements.

If – else if – else (decision tree diagram)

```
if ( x < 100 ) {
    if ( y < 10 ) {
        print( "good job!" );
    } else if ( y > 7 ) {
        print( "not bad" );
    } else {
        print( "try again..." );
    }
}
```

If – else if – else (unreachable code)

```
if ( x < 100 ) {
    if ( y < 10 ) {
        print( "good job!" );
    } else if ( y > 7 ) {
        print( "not bad" );
    } else {
        print( "try again..." );
    }
}
```
Relational and Logical Expressions

- `<` is less than
- `>` is greater than
- `<=` is less than or equal to
- `>=` is greater than or equal to
- `==` is equivalent
- `!=` is not equivalent

Logical conjunction (and)
- Both expressions must be true for conjunction to be true

Logical disjunction (or)
- Either expression must be true for disjunction to be true

Logical negation (not)
- True → false, false → true

for Loop

- Pattern
  1. `<init>`
  2. `<condition>`
  3. `<update>`
  4. `<body>`

- Each section can be blank.
- Sequence: 1 2 3 ... 2 1 (condition fails)

while vs. for

void setup() {
  size(500, 500);
  smooth();
  float diameter = 500;
  while (diameter > 1) {
    ellipse(250, 250, diameter, diameter);
    diameter = diameter - 10;
  }
  void draw() { }
}

void setup() {
  size(500, 500);
  smooth();
  for (float diameter = 500; diameter >= 10; diameter -= 10) {
    ellipse(250, 250, diameter, diameter);
  }
  void draw() { }
}

Iteration: for-loop

What does the following code print?

```java
int num=0;
int adder = 1;
for (int i=0; i<=6; i++) {
  num = num + adder;
  adder = -adder;
}
println(num);
```

Write the code

Write a loop that prints all integers between 3 and 52 inclusive, that are divisible by 3.
(Partial credit: a loop that prints all integers between 3 and 52 inclusive)
Write the code

```
Write a loop that prints all integers between 3 and 52 inclusive, that are divisible by 3.
(Partial credit: a loop that prints all integers between 3 and 52 inclusive)
```

```
for (int i = 2; i <= 52; i++) {
  // from 3 to 52 inclusive
  if (i % 3 == 0) { // divisible by 3
    println(i); // print integer
  }
}
```

```
int i, j, end = 10;
for (i = 1; i <= end; i++) {
  for (j = 1; j <= i; j++) {
    print("*");
  }
  println();
}
```

---

### Nested for

```java
int i, j, end = 10;
for (i = 1; i <= end; i++) {
  for (j = 1; j <= i; j++) {
    print("*");
  }
  println();
}
```

---

### Function Examples

```java
void setup() { ... }
void draw() { ... }
```

- **Return value, function name, parameter list and function body**
- **A void function doesn’t return anything**

```java
void circleAndLine() {
  ellipse(random(width), random(height), 10, 10);
  line(random(width), random(height),
       random(width), random(height));
}
```

---

### Functions: Terminology

- **y = twice(x) = 2x**
- **Result**
- **Function argument**
- **Function definition**
- **Function parameter**
- **Function name**
- **Return value**

```
y = twice(5)
y = 10
```

---

### Functions: Defining Functions

```java
float twice(float x) {
  return 2*x;
} // twice()
```

---

### Trace the function

```
int A (int x) {
  int y = 100;
  for (int i=x; i<=10; i+=2) {
    y = y-i;
  }
  return y+1;
}
```
Convert this to a function that takes one argument that determines the number of rows.

```java
int i, j, end = 10;
for (i = 1; i <= end; i++) {
    for (j = 1; j <= i; j++) {
        print("*");
    }
    println();
}
```

Shadowing
- When there is a name conflict between variables of different scopes
  ```java
  int x = 10;
  void setup() {
      int y = x;
  }
  ```
  - The conflicting variables can not have different types (or it’s considered a re-declaration and is not allowed)
  - When shadowed, smaller (inner) scopes have precedence over larger (outer) scopes

What is printed?

```java
int a = 20;
void setup() {
    size(200, 200);
    background(51);
    stroke(255);
    noLoop();
}
void draw() {
    println(a);
    for (int a = 50; a < 80; a += 30) {
        println(a);
        anotherA();
    }
    otherA();
    println(a);
    otherA();
}
void anotherA() {
    println(a);
}
void otherA() {
    println(a);
}
```

Basics of Trigonometry assuming right/up axes

- Recall: \(a^2 + o^2 = h^2\)
  - \(\sin(q) = o/h\)
  - \(\cos(q) = a/h\)

Trigonometry on a unit circle
Drawing points along a circle

```java
int steps = 8;
int radius = 20;
float angle = 2*PI/steps;
for (int i=0; i<steps; i++) {
    float x = cos(angle*i)*radius;
    float y = sin(angle*i)*radius;
    // draw a point every 1/8th of a circle
    ellipse(x, y, 10, 10);
}
```

Decimal vs. Binary vs. Hexadecimal

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00000000</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>00000010</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>00000011</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>00000100</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>00000101</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>00000110</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>00000111</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>00001000</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>00001001</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>00001010</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>00001011</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>00001100</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>00001101</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>00001110</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>00001111</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>00010000</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>00010001</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>00010010</td>
</tr>
</tbody>
</table>

Syntax

- Function call
  - `line( 10, 10, 50, 80 );`
- Name
  - The commas
  - The parens ()
- The semicolon

- Code block
  - The curly braces {

- Comments
  - `//`
  - `/* and */`

Variable Uses

- Use a value throughout your program, but allow it to be changed
- As temporary storage for an intermediate computed result
- To parameterize – instead of hardcoding coordinates
- Special variables (preset variables)
  - `width`, `height`
  - `screen.width`, `screen.height`
  - `mouseX`, `mouseY`
  - `pmouseX`, `pmouseY`

Primitive Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Default</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>{ true, false }</td>
<td>false</td>
<td>1</td>
</tr>
<tr>
<td>byte</td>
<td>0..255</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>int</td>
<td>-2,147,483,648..2,147,483,647</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>-9,223,372,036,854..9,223,372,036,854</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>-3.40282347E+38..3.40282347E+38</td>
<td>0.0</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>much larger/smaller</td>
<td>0.0</td>
<td>8</td>
</tr>
<tr>
<td>color</td>
<td>{ #00000000..FFFFFFFF }</td>
<td>block</td>
<td>4</td>
</tr>
<tr>
<td>char</td>
<td>a single character 'a', 'b', ...</td>
<td>\u0000'</td>
<td>2</td>
</tr>
</tbody>
</table>

Mixing types and Integer Division

- `3*1.5`
- `value?`
- `type?`
- `3/2`
- `2/3`
- `x/y`
An aside ... Operators

+, -, *, / and ...

i++; equivalent to i = i + 1;
i += 2; equivalent to i = i + 2;
i--; equivalent to i = i - 1;
i -= 3; equivalent to i = i - 3;
i *= 2; equivalent to i = i * 2;
i /= 4; equivalent to i = i / 4;

i % 3; the remainder after i is divided by 3 (modulo)