



CS110 - review

+ 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
- Execution Order
- Variable Scope and Lifetime
- Trigonometry
- 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 can not be guaranteed.

+ type vs. value

Given the following variable declarations, what **type** does the following expression evaluate to?

```
// variable declarations
int a = 2, b = 5;
float x = 2.0;
```

Given the following variable declarations, what **value** does the following expression evaluate to?

```
// expression
b/a * x;
```

declarations

```
int a = 2, b = 5;
float x = 2.0;
```

expression

```
b/a * x;
```

+ Conditionals: if-statement

Programmatic branching ...

```
if ( boolean_expression ) {
    statements;
}

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

+ If - else if - else

```
if ( x < 100 ) {
    if ( y < 10 ) {
        println("good job!");
    }
} else if ( x < 50 ) {
    if ( y > 10 ) {
        println("great job!");
    } else {
        println("what happened?");
    }
} else {
    if ( y > 7 ) {
        println("not bad!");
    } else {
        println("nice try...");
    }
}
```

- 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 ) {
        println("good job!");
    }
} else if ( x < 50 ) {
    if ( y > 10 ) {
        println("great job!");
    } else {
        println("what happened?");
    }
} else {
    if ( y > 7 ) {
        println("not bad!");
    } else {
        println("nice try...");
    }
}
```

+ If - else if - else (unreachable code)

```
if ( x < 100 ) {
    if ( y < 10 ) {
        println("good job!");
    }
} else if ( x < 50 ) {
    if ( y > 10 ) {
        println("great job!");
    } else {
        println("what happened?");
    }
} else {
    if ( y > 7 ) {
        println("not bad!");
    } else {
        println("nice try...");
    }
}
```

What about this branch???

+ Relational and Logical Expressions

<	less than	&&	logical conjunction (and)
>	is greater than	■	both expressions must be true for conjunction to be true
<=	is less than or equal to		logical disjunction (or)
>=	is greater than or equal to	■	either expression must be true for disjunction to be true
==	is equivalent	!	logical negation (not)
!=	is not equivalent	■	true → false, false → true

Assume that the variables `x`, `low` and `high` have been declared and initialized with `int` values such that `low` is less than or equal to `high`. Which of the following is a valid expression that evaluates to `true` if the value of `x` is within the range `low` to `high`, inclusive, and `false` otherwise?

Select one:

- a. `low < x && x <= high`
- b. `low < x < high`
- c. `low <= x || x <= high`
- d. `low <= x <= high`
- e. `low <= x && x <= high`

for Loop

- Pattern


```

for ( ①init; ②condition; ④update ) {
  ③body
}
            
```

Labels: `statement` (points to `init`), `logical expression` (points to `condition`), `statement` (points to `body`).
- Each section can be blank.
- Sequence: ① ②③④ ... ②③④ ② (condition fails)

+ Iteration: for-loop

What does the following code print?

```

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

+ 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 > 1; diameter -- 10 ) {
    ellipse( 250, 250, diameter, diameter);
  }
}

void draw() { }
            
```

+ 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)

Preformatted ▾ B I

Path: pre

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

```

+ Nested for

```

int i, j, end = 10;

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

```

20

+ Function Examples

```

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

```

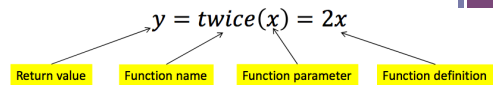
- Return value, function name, parameter list and function body
- A void function doesn't return anything

```

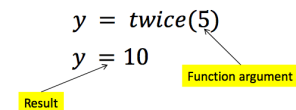
void circleAndLine() {
  ellipse(random(width), random(height), 10, 10);
  line(random(width), random(height),
       random(width), random(height));
}

```

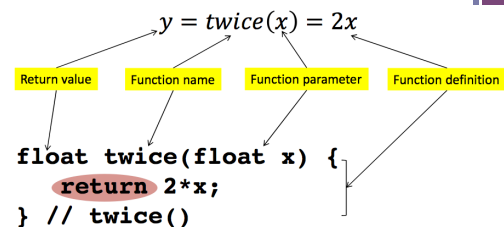
+ Functions: Terminology



Function application:



+ Functions: Defining Functions



+ Trace the function

- A(10);
- A(20);
- A(5);

```

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.

```
int i, j, end = 10;

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

25

Shadowing

■ When there is a name conflict between variables of different scopes

```
int x = 10;
void setup() {
  int x = 5;
  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

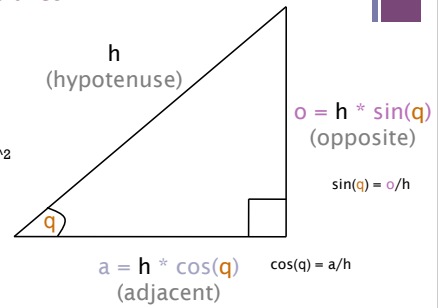
+ What is printed?

```
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();
  }
  yetAnotherA();
  int a = 100;
  println(a);
  yetAnotherA();
}
```

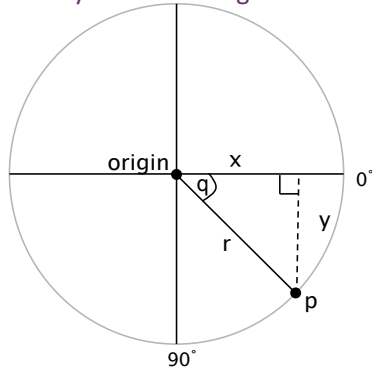
```
void anotherA() {
  int a = 185;
  println(a);
}
void yetAnotherA() {
  println(a);
}
```

+ Basics of Trigonometry assuming right/up axes

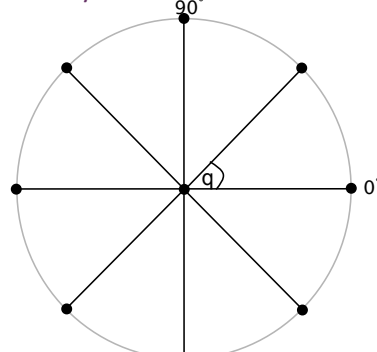
Recall:
 $a^2 + o^2 = h^2$



+ Trigonometry on Processing unit circle



+ Trigonometry on a unit circle



Drawing points along a circle

```

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

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

- ### + 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

Type	Range	Default	Bytes
boolean	{ true, false }	false	?
byte	{ 0..255 }	0	1
int	{ -2,147,483,648 .. 2,147,483,647 }	0	4
long	{ -9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807 }	0	8
float	{ -3.40282347E+38 .. 3.40282347E+38 }	0.0	4
double	<i>much larger/smaller</i>	0.0	8
color	{ #00000000 .. #FFFFFF }	<i>black</i>	4
char	<i>a single character 'a', 'b', ...</i>	<i>'\u0000'</i>	2

- ### + 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)
```