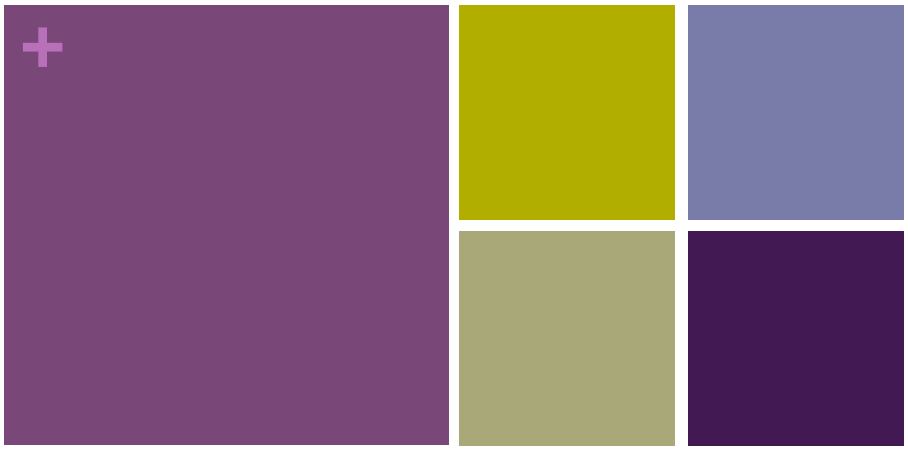


CS110 - review

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
- Loops
  - Condition
  - Index
- Functions
  - Definition
  - Call
  - Parameters
  - Return value
- Execution Order
- Variable Scope and Lifetime
- Trigonometry

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.

Variable Scope

- The region of code in which a particular variable is accessible.
- To a first approximation, the scope of a section of your code is demarcated by { and }.
  - Functions
  - Loops
  - Conditionals
- A variable is only accessible/available within the scope in which it is declared.

Variable Lifetime

- Variables cannot be referenced before they are declared.
- A variable is created and initialized when a program enters the block in which it is declared.
  - Functions
  - Loops
  - Conditionals
  - Function parameters
- A variable is destroyed when a program exists the block in which it was declared.

Global variables

- Variables that are declared outside of any scope are considered globals (versus locals).
- Global variables should be declared at the top of your program.
- Do not sprinkle them between functions!

Shadowing

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

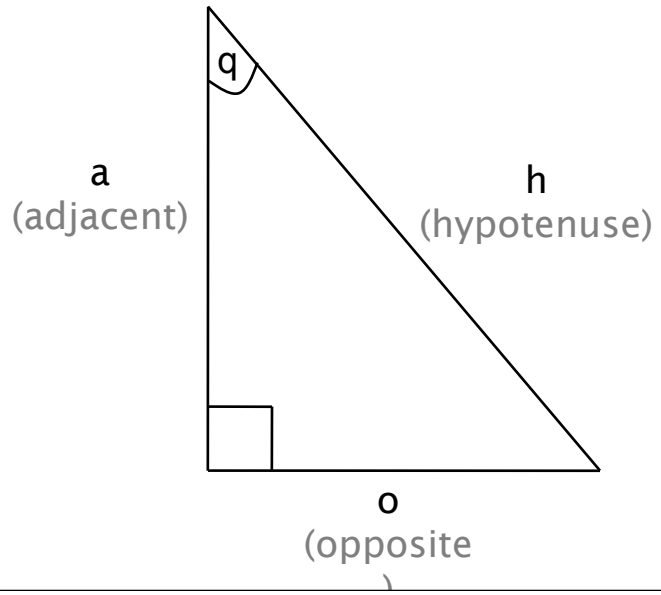
```
int x = 10;
void setup() {
  int x = 5;
  int y = x;
}
```

- 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

```
int a = 20;

void setup() {
  size(200, 200);
  background(51);
  stroke(255);
  noLoop();
}

void draw(){
  line(a, 0, a, height);
  for(int a=50; a<80; a += 2) {
    line(a, 0, a, height);
  }
  int a = 100;
  line(a, 0, a, height);
  drawAnotherLine()
  drawAnotherLine();
  drawYetAnotherLine()
  drawYetAnotherLine();
}

void drawAnotherLine() {
  int a = 185;
  line(a, 0, a, height);
}

void drawYetAnotherLine() {
  line(a+2, 0, a+2, height);
}
```

- What is drawn?

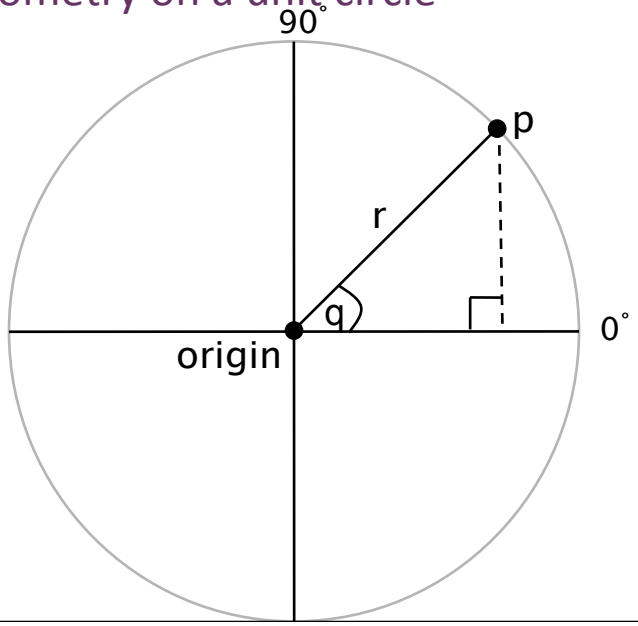
Basics of Trigonometry



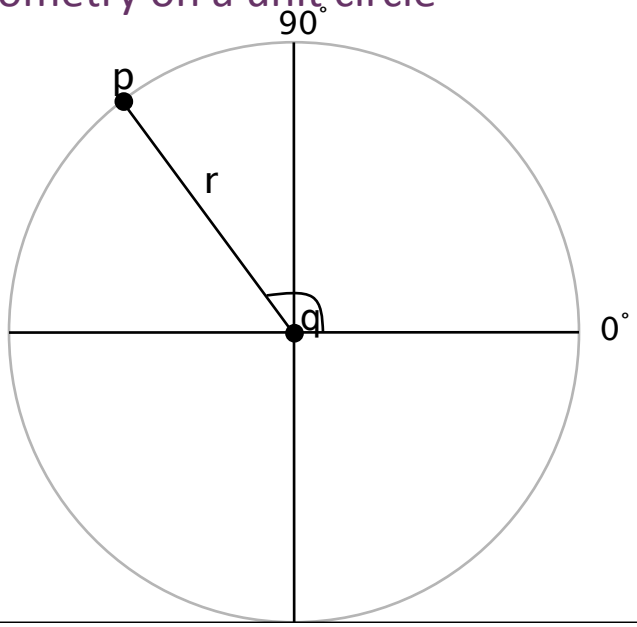
Definition

- sin(q) = o/h
- o = h*sin(q)
- cos(q) = a/h
- a = h*cos(q)
- tangent(q) = o/a = sin(q)/cos(q)

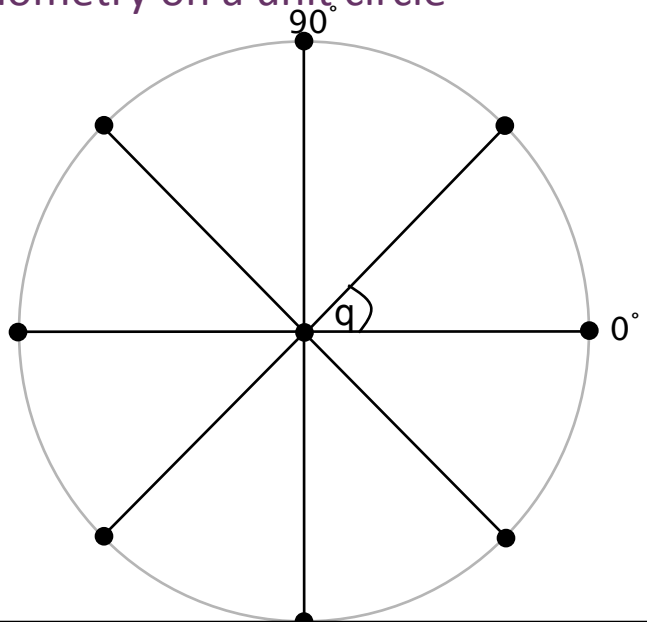
Trigonometry on a unit circle



Trigonometry on a 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 a 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	*much larger/smaller*	0.0	8
color	{ #00000000 .. #FFFFFFFF }	*black*	4
char	*a single character* 'a', 'b', …	'\u0000'	2

Data Type Conversion

- Variables of some types can be converted to other types.
- Type conversion function names are the types to which data will be converted

```
// binary(…), boolean(…), byte(…),
// char(…), float(…), str(…)


float f = 10.0;
int i;

//i = f;                  // Throws a runtime error
i = int(f);

println( char(65) );      // Prints the character 'A'
```

Mixing types and Integer Division

- 3*1.5
  - value?
  - type?
- 3/2
- 2/3
- x/y

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 );
     }
}
```

Relational Expressions

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

Conditionals: switch-statement

- Works like a if-else statement.
- Convenient for large numbers of value tests.

```
switch( expression ) {
  case label1:          // label1 equals expression
     statements;
     break;
  case label2:          // label2 equals expression
     statements;
     break;
  default:              // Nothing matches
     statements;
}
```

What does this do?

```
void setup() {
  size(500, 500);
  smooth();
}

void draw() {}

void keyPressed() {
  switch(key)
  {
    case 'l':
    case 'L':
      println("Turning left");
      break;
    case 'r':
    case 'R':
      println("Turning right");
      break;
  }
}
```

Expressions

- Collections of data values and variables related by operators and function calls, and grouped by parentheses.
- Expressions are automatically evaluated and replaced by the final evaluated value.
- Assignment: Expressions can be **assigned** to variables using "= "
  - Expression is always on right
  - Variable name is always on left

variable_name = expression;

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)
```

```
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() { }
```

Iteration

Repetition of a program block

- Iterate when a block of code is to repeated multiple times.

Options

- while-loop
- for-loop

Iteration: while-loop

```
while ( boolean_expression ) {
  statements;
  // continue;
  // break;
}
```

- Statements are repeatedly executed while the boolean expression remains true.
- Don't ever use these statements!
  - To break out of a while loop, call **break;**
    - use your *boolean expression* instead
  - To continue with next iteration, call **continue;**
    - (use conditional blocks instead)
- All iterations can be written as while-loops.

Iteration: for-loop

```
for ( initialization; continuation_test; update)
{
  statements;
  // continue;  // Continues with next iteration
  // break;             // Breaks out of loop
}
```

- A kind of iteration construct
- initialization, continuation test and increment commands are part of statement
- Don't ever use these statements!
  - To break out of a while loop, call **break;**
    - (use your *continuation test* instead)
  - To continue with next iteration, call **continue;**
    - (use conditional blocks instead)
- All for loops can be translated to equivalent while loops

Functions Informally

- A function A function is like a subprogram, a small program inside of a program.
- The basic idea – we write a sequence of statements and then give that sequence a name. We can then execute this sequence at any time by referring to the name.
- Function definition: this is where you create a function and define exactly what it does
- Function call: when a function is used in a program, we say the function is *called*.
- A function can only be defined once, but can be called many times.

Function Examples

```
void setup() { … }
void draw() { … }


void line( float x1, float y1, float x2, float y2) { … }
… and other graphic functions


float float( … )
… and other type-conversion functions

… etc.
```

Functions

Modularity

- Functions allow the programmer to break down larger programs into smaller parts.
- Promotes organization and manageability.

Reuse

- Enables the reuse of code blocks from arbitrary locations in a program.

Function Parameters

- Parameters (arguments) can be "passed in" to function and used in body.
- Parameters are a comma-delimited set of variable declarations.
- Parameters act as input to a function.
- Passing parameters provides a mechanism to execute a function with many different sets of input
- We can call a function many times and get different results by changing its parameters.

What happens when we call a function?

- Execution of the main (calling) program is suspended.
- The argument expressions are evaluated.
- The resulting values are copied into the corresponding parameters.
- The statements in the function's body are executed in order.
- Execution of the main program is resumed when a function exits (finishes).

Variable Scope

The part of the program from which a variable can be accessed.

Rules:

1. Variables declared in a block are only accessible within the block.
2. Variables declared in an outer block are accessible from an inner block.
3. Variables declared outside of any function are considered global (available to all functions).

Variable Lifetime

- Variables cannot be referenced before they are declared.
- Variables can be declared in…
  - the global scope
  - the body of a function or constructor
  - the arguments of a function or constructor
  - a statement block (for, while, if, …).
- A variable is created and initialized when a program enters the block in which it is declared.
- A variable is destroyed when a program exists the block in which it was declared.

```
int v1 = 1;

void setup() {
  int v2 = 2;

  for (int v3=3; v3 <= 3; v3++) {
    int v4 = 4;
    println("-------------");
    println("v1=" + str(v1));
    println("v2=" + str(v2));
    println("v3=" + str(v3));
    println("v4=" + str(v4));
    //println("v5=" + str(v5));
  }

  int v3 = 6;
  println("v3=" + str(v3));

  aFunction(v2);
}

void aFunction(int v5) {
  println("-------------");
  println("v1=" + str(v1));
  //println("v2=" + str(v2));
  //println("v3=" + str(v3));
  //println("v4=" + str(v4));
  println("v5=" + str(v5));
}

void draw() { }
```

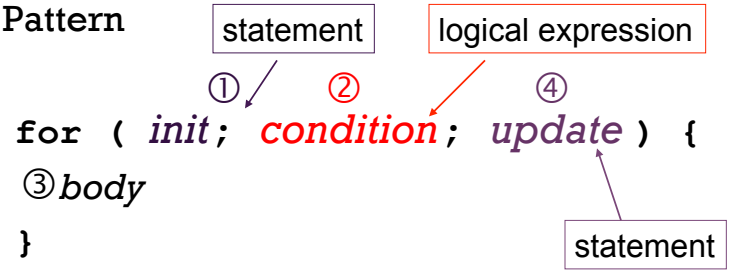
- What is printed?
- What happens if the second v3 declaration is removed?
- What would happen if the v5 print statement is executed?
- What would happen if commented statements in aFunction were called?

Review

- Loops
  - Condition
  - index

for Loop

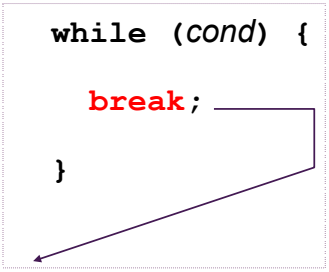
- Pattern



  - Each section can be blank.
  - Sequence: ① ②③④ … ②③④ ② (condition fails)

+ break Statements

- Exit from a loop
- Typically used with an `if` statement



44

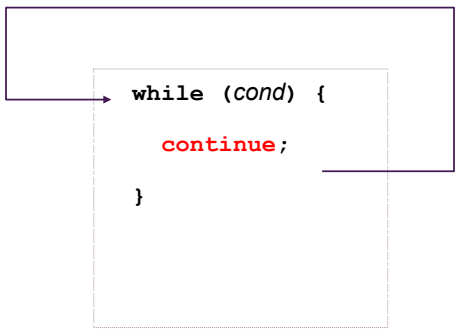
Example

```
for(int i=1; i<=100; i++) {
  if (i > 50)
    break;
  println(i);
}
```

45

`continue` Statements

- Continue to the beginning of a loop
  - I.e., the condition will be checked
- Typically used with an `if` statement



Lec04 46

Example

```
for(int i=1; i<=100; i++) {
  if (i >= 20 && i <= 30)
    continue;
  println(i);
}
```

47

```
void mousePressed() {
  for (int i = 0; i < 10; i++ ){
    print( i );
  }
  println();
}
void draw() { }
```

```
void mousePressed() {
  for (int i = 0; i < 10; i++ ) {
    if ( i % 2 == 1 ) continue;
    print( i );
  }
  println();
}
void draw() { }
```

Nested for

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

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

49

Review

- Functions
  - Definition
  - Call
  - Parameters
  - Return value

Identify Similar Code

```
float x, y, w, h;
int totalShapeCount = 1000;

void setup () {
  int i = 0;
  //other setup code here …
  stroke(255, 50);
  while (i<totalShapeCount) {
    fill(random(255), random(255),
         random(255), 50);
    x = random(width);
    y = random(height);
    w = random(5, 100);
    h = random(5, 100);
    rect(x, y, w, h);
    i += 1;
  }

  stroke(0, 50);
  for (i=0; i<totalShapeCount; i+=1) {
    fill(random(255), 50);
    x = random(width);
    y = random(height);
    w = random(5, 100);
    h = random(5, 100);
    ellipse(x, y, w, h);
  }
}
```

Similar unit

Similar unit

51

Identify Similar Code

```
float x, y, w, h;
int totalShapeCount = 1000;

void setup () {
  int i = 0;
  // other setup code here …
  stroke(255, 50);
  while (i<totalShapeCount) {
    drawRandomShape(1);
    i += 1;
  }
  stroke(0, 50);
  for (i=0; i<totalShapeCount; i++) {
    drawRandomShape(2);
  }
}

void drawRandomShape(int choice) {
  x = random(width); y = random(height);
  w = random(5, 100); h = random(5, 100);
  if (choice == 2) {  // circle
    fill(random(255), 50);
    ellipse(x, y, w, h);
  }
  else {
    fill(random(255), random(255), random(255), 50);
    rect(x, y, w, h);
  }
}
```

52