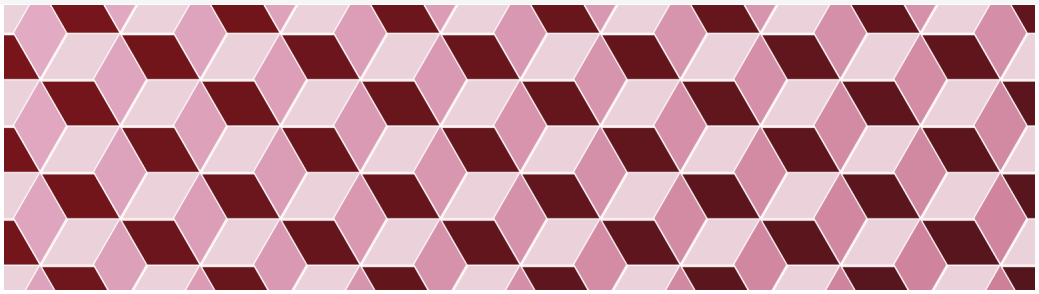


15-104 Introduction to Computing for Creative Practice

Fall 2022

37 Greatest Hits

Instructor: Tom Cortina, tcortina@cs.cmu.edu, GHC 4117, 412-268-3514

1

Basics

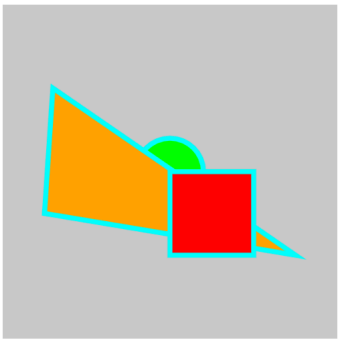
- Your p5.js programs consist of two basic functions:
  - `function setup() {`
    …
    `}`
    Runs first when your program launches to set up the canvas.
  - `function draw() {`
    …
    `}`
    Runs repeatedly, over and over, to draw on the canvas (unless you execute `noLoop();`)
- If you draw the same thing each time `draw()` runs, then it will look like a painting.
  If you draw something different each time, then it will look like an animation.

`frameRate(r);`
sets the number of times
draw repeats to r times per second.

2

The Canvas

- A canvas is made up of **pixels** (picture elements).
- Screen resolution is expressed in pixels (e.g. 1920 X 1080)
- The origin (0, 0) of the canvas is at the top left.
  - x coordinates increase from left to right
  - y coordinates increase from top to bottom
- Drawing is like painting…
  - It's sequential. New paint goes on top of old paint.
  - The order you write the instructions is the order that your painting will be constructed.



3

Functions / Parameters vs. Arguments

- We've used functions that are predefined. (e.g. `random`, `ellipse`, etc.)
- We pass arguments to these functions (a function call).
- Each function assigns these arguments to a set of parameters.
- When the function completes its computation, it can return a result.*
- Computation continues where we left off after the function call.
- When you call a function, you should supply the same number of arguments as it has parameters.
- We can define our own functions that can be called from `draw` (or from each other).
- General format:

```
function name ( parameterlist ) {
        function body
}
```

*In your own functions, you can return a value with `return` statement(s). Once a `return` statement executes, flow of control goes back to the calling function immediately.
General format: `return (`*expression*`);`

4

Shapes & Fill

- `ellipse(x, y, w, h);`
- `circle(x, y, d);`
- `rect(x, y, w, h);`
- `square(x, y, s);`
- `triangle(x1, y1, x2, y2, x3, y3);`
- `quad(x1, y1, x2, y2, x3, y3, x4, y4);`
- `fill(r, g, b, [alpha]);`
- `fill(grayvalue);`
- `fill(color);`
- `noFill();`

`rectMode(CENTER);` treats `x,y` as center rather than top left for subsequent rectangles/squares.

`r`	red (0 to 255, inclusive).
`g`	green (0 to 255, inclusive).
`b`	blue (0 to 255, inclusive).
`alpha`	opacity (0 = transparent, to 255=fully opaque).
`grayvalue`	black(0) to white (255)
`color`	a p5.Color object

5

Lines and Stroke

- `line(x1, y1, x2, y2);`
- `point(x, y);`
- `stroke(r, g, b, [alpha]);`
- `stroke(grayvalue);`
- `stroke(color);`
- `strokeWeight(weight);`
- `noStroke();`
- `dist(x1, y1, x2, y2)`
  returns the distance between (x1,y1) and (x2,y2)

`r`	red (0 to 255, inclusive).
`g`	green (0 to 255, inclusive).
`b`	blue (0 to 255, inclusive).
`alpha`	opacity (0 = transparent, to 255=fully opaque).
`grayvalue`	black(0) to white (255)
`color`	a p5.Color object

6

Variables

- A variable is a container that holds some data value.
  - Global variables – defined before the setup function (i.e. not inside any specific function)
  - Local variables – defined within a function
- We store a value in a variable using assignment (=).
  - An assignment statement is of the form: *variable* = *expression* `;`
  - Assignment overrides the previous value stored in the variable.
- p5.js has some variables that are predefined in the language to mean something:
  `mouseX, mouseY, width, height, mouseIsPressed`
- Variables in p5.js have implicit data types. (e.g. Number, Boolean)

7

Using Arithmetic

- In general, at any place you can write a number, you can write an arithmetic expression or a function call that evaluates to a number.
- Order of operations:
  `* / %` first (as they occur, left to right)
  `+ –` next (as they occur, left to right)
- Parentheses can override order of operations. (e.g. `(2 + 3) * 4 = 20`)
- Modulo operator:
  `x % y` (for integers x > 0, y >0): Divide x by y and keep the remainder.
  Examples: `45 % 10 = 5` `8 % 12 = 8`
- Exponentiation:
  `Math.pow(a,b)` returns a^b

8

Mouse and key processing

`mouseX` – contains the current horizontal position of the mouse, relative to (0, 0)

`mouseY` – contains the current vertical position of the mouse, relative to (0, 0)

`mouseIsPressed` – Boolean that is true while the mouse button is pressed down

`mousePressed()` – function that is called when the mouse button is pressed down

`mouseReleased()` – function that is called when the mouse button is released

`key` – contains the current key pressed as a string

`keyIsPressed` – Boolean that is true while a key is pressed down

`keyPressed()` – function that is called when a key is pressed down

9

Conditionals (the `if/else` statement)

- An `if` statement allows to test a logical condition to determine whether to run some code or not.
- An `if-else` statement allows to test a logical condition to determine whether to run some code or some other code.
- Logical conditions are expressions that evaluate to true or false.
- General forms for `if` and `if-else`:

```
if ( condition ) {
        instruction(s) if true
}
```

```
if ( condition ) {
        instruction(s) if true
} else {
        instruction(s) if false
}
```

10

Boolean expressions

- Expressions with the relational operators lead to true or false:

```
x <  y    less than
x >  y    greater than
x <= y    less than or equal to
x >= y    greater than or equal to
x == y    equal to   (also ===)
x != y    not equal to
```

- Expressions with logical operators lead to true or false:

```
a && b    logical and (true if both a and b are true)
a || b    logical or  (true if either a or b are true)
!a        logical not (true if a is false, and vice-versa)
```

11

Boolean shortcuts

```
if (on == true) { ...
```

```
if (on) { ...
```

```
if (on == false) { ...
```

```
if (!on) { ...
```

```
if (on) {
    on = false;
} else {
    on = true;
}
```

```
on = !on;
```

```
return (on == true);
```

```
return (on);
```

12

Bounds & Mapping

- `min(num1, num2)`
- `max(num1, num2)`
- `constrain(num, low, high)`
- `map(value, start1, stop1, start2, stop2)`
  - Re-maps a number from one range [start1,stop1] to another [start2, stop2].
- `floor(x)` – returns the greatest integer less than or equal to x
- `round(x)` – returns the nearest integer to x (a number with .5 rounds up)

13

Random values

`random(x, y)`

- Returns a random number between `x` (inclusive) and `y` (exclusive), uniformly.

`random(y)`

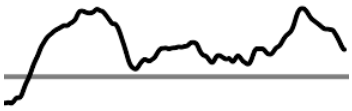
- Returns a random number between 0 (inclusive) and y (exclusive), uniformly.

`noise(xoff)`

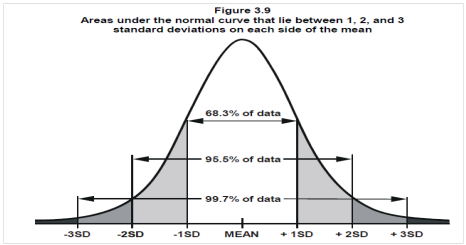
- Returns a random value between 0 and 1 from a Perlin noise function at offset xoff.

`randomGaussian(m, sd)`

- Returns a random value so that, over time, the mean is m and the standard deviation is sd.



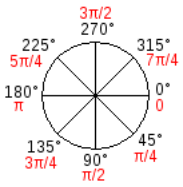
Perlin noise is a random sequence generator producing a more naturally ordered, harmonic succession of numbers compared to the standard **random()** function.



14

Transformations

- In p5.js, you can perform a transformation on the canvas to create interesting effects.
- Types of transformations:
  - `translate(x, y)` – Translation (shift horizontally and/or vertically)
  - `rotate(angle)` – Rotation (rotate a certain angle around the origin)
    - `angle` is in radians. Call `radians(d)` to convert degree value `d` to radians.
  - `scale(s)` – Scaling (expand or contract) by factor s



- Transformations occur by moving the coordinate system of the canvas, not the object itself.
- `push()` saves the current coordinate system and drawing properties.
- `pop()` returns you back to your previously saved coordinate system and drawing properties.

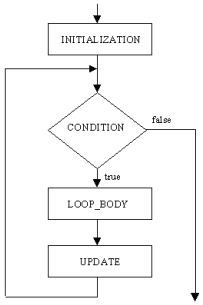
15

`for` Loop

```
for ( loop_initialization ; loop_condition ; loop_update ) {
      code to repeat
}
```

Example:

```
for (var i = 0 ; i < n ; i += 1) {
      // loop body goes here
}
```



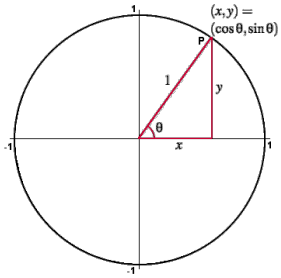
This is how programmers typically write a loop that runs `n` times where `i` is the loop counter. (`n` > 0)
The variable `i` cannot be used outside of the loop since it is defined locally (within the loop structure).
Loops can be nested:

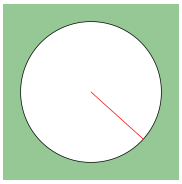
```
for (var row = 0; row < 5; row += 1) {
    for (var col = 0; col < 4; col += 1) {
        ...
    }
}
```

16

Polar Coordinates

- You can describe locations in terms of angle and radius (polar coordinates).
- cos and sin functions tell you X and Y coordinates of a point on a circle of radius 1. The input parameter for `cos` and `sin` is the *angle* (in radians): how far to rotate around the circle. The output is where you land in terms of X (using `cos`) and Y (using `sin`).





```
translate(width/2, height/2);
circle(0, 0, 2*r);
x = r * cos(radians(theta));
y = r * sin(radians(theta));
line(0, 0, x, y);
```

17

Arrays

- An array with n elements (n > 0) is an ordered collection of values of the same type, indexed from 0 to n-1. (ordered does not necessarily mean sorted here)

```
temps = [79, 81, 57, 64, 63, 57, 57]
```

- To access an array, we use "subscript" (index) notation:

```
average = sum / temps.length;
min = temps[0];
for (var j = 1; j < temps.length; j++) {
      if (temps[j] < min) { min = temps[j]; }
}
```

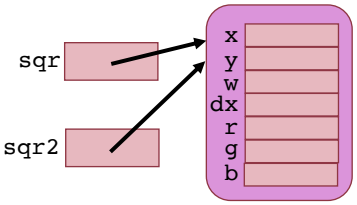
- Methods:
  - `push(element)` — appends element to an array (e.g. `temps.push(73);`)
  - `pop()` — deletes the last element, and returns it
  - `shift()` — returns the first element and shifts the rest down

18

Objects

- An object can be defined literally

```
var sqr = {x: 100, y: 100, w: 50, dx: 5,
           r: 255, g: 255, b:0};
```



- We can also create objects by construction within our program code.

```
var sqr = new Object();
sqr.x = 100; sqr.y = 100; sqr.w = 50; sqr.dx = 5;
sqr.r = 255; sqr.g = 255; sqr.b = 0;
```

`sqr2 = sqr;`
`sqr2` is an alias.

- To access any properties of the object, we use dot notation, listing the object variable name followed by a dot followed by the property (field) of the object.

```
fill(sqr.r, sqr.g, sqr.b);
```

- An object variable points to ("references") its own object.

19

Object Methods

```
function tulipDraw() {
    ...
    rect(this.x, this.y - this.height, 10, this.height);
    var y = this.y - this.height;
    ellipse(this.x + 5, y, 44, 44);
    ...
};
function tulipGrow(amount) {
    this.height += amount;
};
```

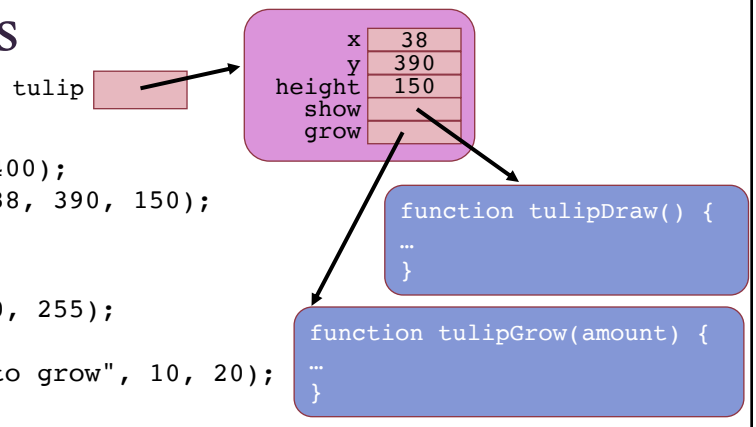
We use the reference `this` to indicate that we are writing a function and we are referencing a property of this object while performing the function.

```
function makeTulip(tx, ty, th) {        // constructor
    var tulip = {x: tx, y: ty, height: th,
                 show: tulipDraw, grow: tulipGrow};  // new object
    return tulip; // return the new object
};
```

20

Using objects

```
var tulip;
function setup() {
    createCanvas(400, 400);
    tulip = makeTulip(38, 390, 150);
}
function draw() {
    background(207, 250, 255);
    tulip.show();
    text("Press mouse to grow", 10, 20);
};
function mousePressed() {
    tulip.grow(5);
}
```

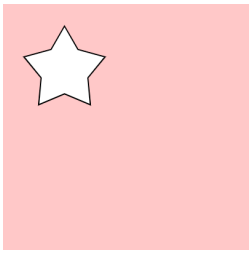


21

Uses of arrays

Custom Shape

```
var x = [50, 61, 83, 69, 71, 50, 29, 31, 17, 39];
var y = [18, 37, 43, 60, 82, 73, 82, 60, 43, 37];
beginShape();
for (var i = 0; i < nPoints; i++) {
    vertex(x[i], y[i]);
}
endShape(CLOSE);
```



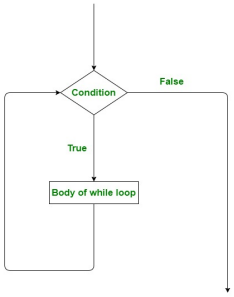
Array of Objects

```
var sqr_array = [];
sqr_array[0] = {x: 100, y: 100, w: 50, dx: 5, r: 255, g: 255, b:0};
sqr_array[1] = {x: 50, y: 50, w: 50, dx: 10, r: 0, g: 255, b:255};
fill(sqr_array[0].r, sqr_array[0].g, sqr_array[0].b);
square(sqr_array[0].x, sqr_array[0].y, sqr_array[0].w);
```

22

`while` Loop

```
while (condition ) {
        loop body
}
```

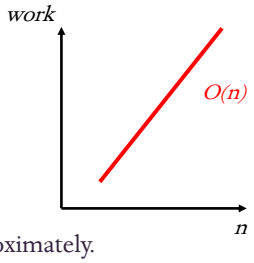



```
function linear_search(arr, element, index) {
    if (index < 0 || index >= arr.length) return -1;
    var i = index;
    while (i < arr.length) {
        if (arr[i] == element) return i;
        i++;
    }
    return -1;
}
```

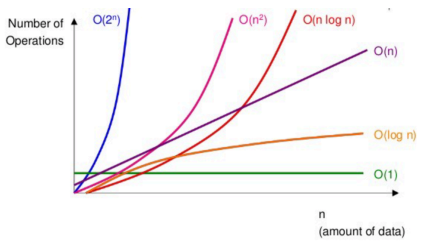
23

Big O

- We say linear search is **O(n)** in the worst case.
  - All algorithms in this class do an amount of work linearly proportional to the number of data values (n).
  - If an algorithm is O(n), then if we double the number of inputs/elements, then we can expect twice as much work, approximately.



- If an algorithm is **O(n^2)** (a quadratic algorithm), then if we double the number of data values, we can expect $4 = 2^2$ times as much work.
- Comparing algorithms: When *n* is small, the algorithm you pick doesn't really matter. But when *n* is large, it matters!

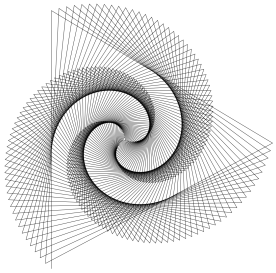


24

Turtle Graphics API

- `makeTurtle(x, y)` -- make a turtle at x, y, facing right, pen down
- `left(d)` -- turn left by d degrees
- `right(d)` -- turn right by d degrees
- `forward(p)` -- move forward by p pixels
- `back(p)` -- move back by p pixels
- `lowerPen()` -- set pen down
- `raisePen()` -- pick pen up
- `goto(x, y)` -- go straight to this location
- `setColor(color)` -- set the drawing color
- `setWeight(w)` -- set line width to w
- `face(d)` -- turn to this absolute direction in degrees
- `angleTo(x, y)` -- what is the angle from my heading to location x, y?
- `turnToward(x, y, d)` -- turn by d degrees toward location x, y
- `distanceTo(x, y)` -- how far is it to location x, y?

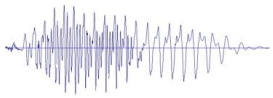
An Application Programmer's Interface (API) is a view of the methods (functions) of the object without seeing the details. The programmer can use the object just by knowing how to call the methods and what they return.



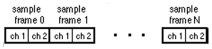
25

Sound

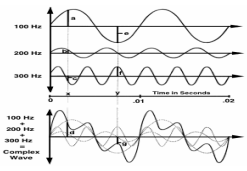
- Audio signals are essentially vibrations that travel through the air (and other materials) creating changes in pressure.



- The accuracy of the digital audio sequence compared to the original analog audio signal increases with increased sampling rate, and increased bits per sample.
  - e.g. CD audio: 44,100 Hz, 16 bits/sample (2^{16} sound levels), 2-channel audio
- Multichannel sound interleaves samples in a sound data file.

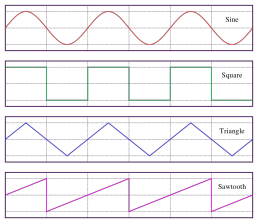


- To capture a frequency of X, you must sample the signal at a sample rate of 2X. (sampling theorem)
- A sound made up of a set of harmonic sinusoids at varying amplitudes and phases, summed together.



26

p5.Oscillator



- Creates a signal that oscillates between -1.0 and 1.0.
  - By default, the oscillation takes the form of a sinusoidal shape ('sine').
  - The frequency defaults to 440 oscillations per second (440Hz).
    - `start()` Start an oscillator.
    - `stop()` Stop an oscillator.
    - `amp()` Set the amplitude between 0 and 1.0.
    - `freq()` Set frequency of an oscillator to a value.
    - `setType()` Set type to 'sine', 'square', 'triangle', or 'sawtooth' .

Volume ↔ Pitch ↕

```
function soundSetup() {
    myTone = new p5.Oscillator();
    myTone.setType('sine');
    myTone.freq(880);
    myTone.start();
}
```

```
function draw() {
   myTone.amp(mouseX / width);
   myTone.freq(200 + 1000*(mouseY / height));
}
```

27

Drawing in 3D

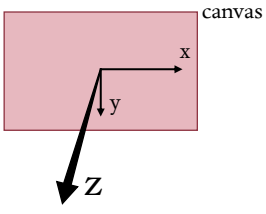
- The origin in WEBGL is in the center of the canvas by default.
  - x increases left to right (as before)
  - y increases top to bottom (as before)
  - z increases toward us. (by default, the canvas is the z=0 plane)

    `createCanvas(400, 250, WEBGL);`
- Camera views: perspective, orthographic
- 3D Shape Primitives (center is at origin):

  `box(width, height, depth);`

  `sphere(radius);`

  Use transformations to place shapes in scene.
- Lighting: ambient, directional, point
- Materials: basic, normal, ambient, specular



28

Code Style

- Comments help explain parts of your code to the reader

```
fill (255, 255, 255);  // this is a comment (to the end of the line)
/* this is a comment
   over several lines */
```

- Indentation shows code that is "inside" other code.
  - Examples: the body of a function, the body of a loop, nested code.
  - Typically a left bracket { increases indentation and a right bracket } decreases it.
- Functions compartmentalize the code into individually managed units which can be debugged/managed separately.
  - Functions manage complexity of your code by "hiding" finer details to help the programmer focus on the overall design task.

29