EECS 367 Lab
Pendularm Code Overview

Administrative

Assignment 1: Path Planning

- Regrade policy described on course website
- Up to 80% credit can be earned after grading

Assignment 2: Pendularm

- Due Wednesday, September 30

Lab Takeaways

1. Assignment overview
2. Stencil walkthrough
3. Pendularm demo
4. Coding considerations

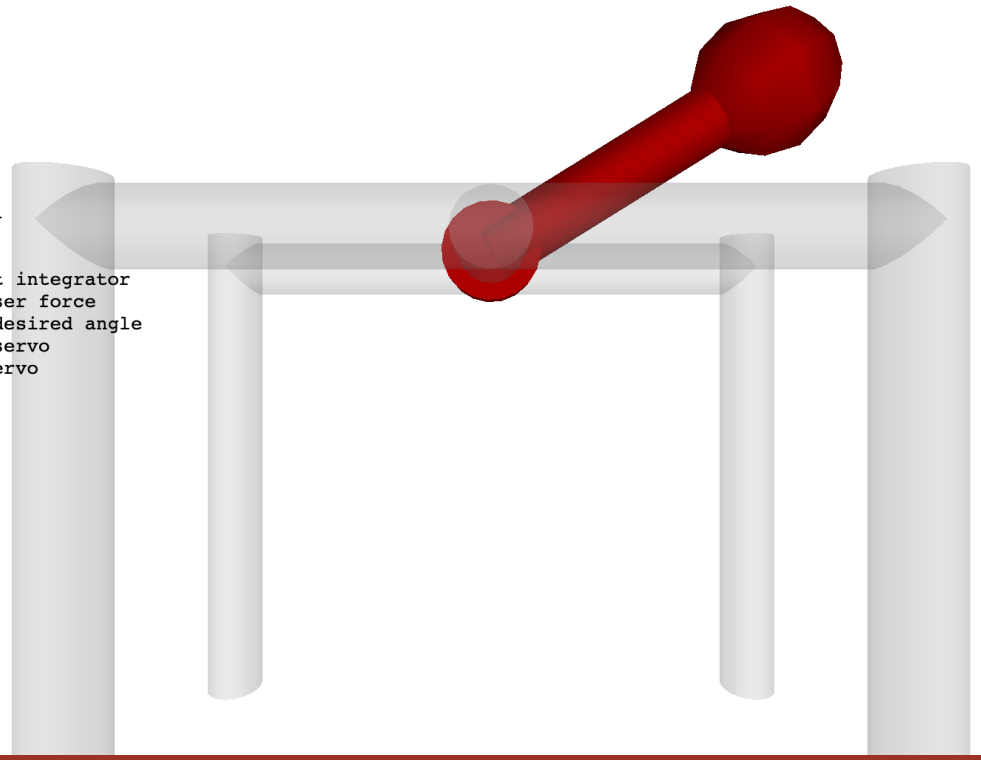
Pendularm Overview

```
System
t = 69.60 dt = 0.05
integrator = velocity verlet
x = -4.16
x_dot = 0.00
x_desired = -4.16

Servo: active
u = 33.33
kp = 150.00
kd = 60.00
ki = 4.00

Pendulum
mass = 2.00
length = 2.00
gravity = 9.81

Keys
[0-4] - select integrator
a/d - apply user force
q/e - adjust desired angle
c|x - toggle servo
s - disable servo
```



We will be implementing a servo controller for the pendularm!

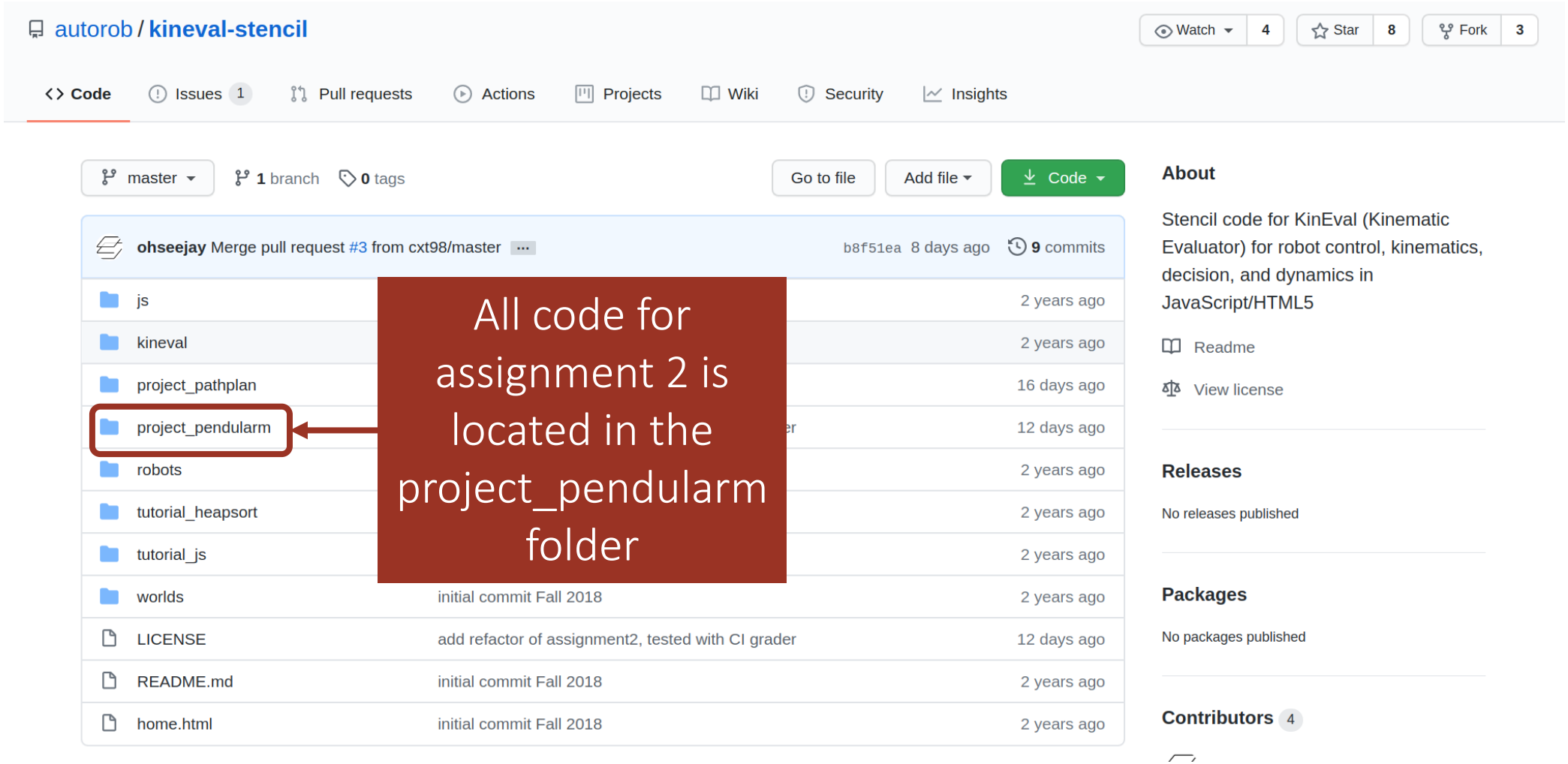
Pendularm Overview

Assignment 2: Pendularm		
4	All	Euler integrator
4	All	Velocity Verlet integrator
4	All	PID control
1	Grad	Verlet integrator
2	Grad	RK4 integrator
3	Grad	Double pendulum

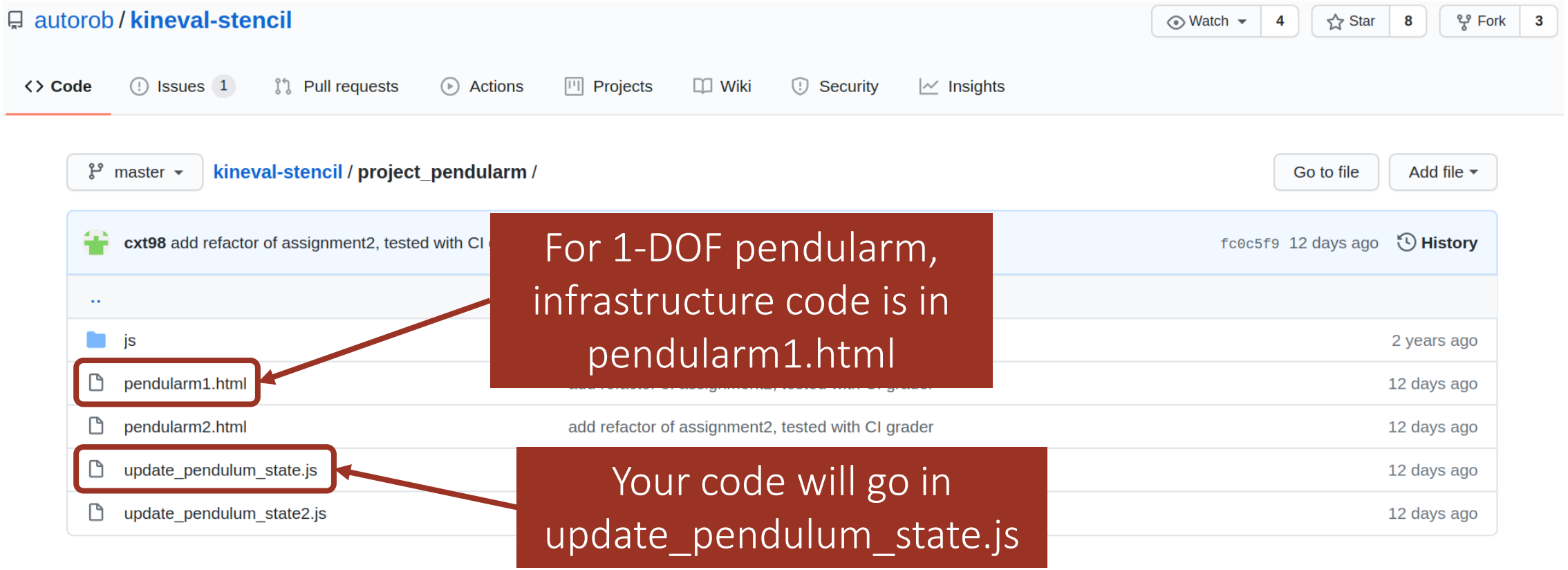
Features assigned to all sections

Features assigned to grad section only

Stencil Walkthrough



Stencil Walkthrough



Stencil Walkthrough

pendularm1.html

```
28  <!-- ////////////////////////////////////////////////
29       /////     JAVASCRIPT INCLUDES
30       //////////////////////////////////////////////// -->
31
32
33  <!-- threejs - https://github.com/mrdoob/three.js/ -->
34  <script src="js/three.min.js"></script>
35
36  <!-- threejs camera controls helpers -->
37  <script src="js/OrbitControls.js"></script>
38
39  <!-- threejs keyboard input helper -->
40  <script src="js/THREEx.KeyboardState.js"></script>
41
42  <!-- functions to be implemented -->
43  <script src="update_pendulum_state.js"></script>
44
45  <script>
46
47  ////////////////////////////////////////////
48
49
```

`<html>` open tag
`<body>` open tag
mean that what follows will appear on webpage

Include useful JavaScript libraries for visualization and control
https://threejs.org

`init()` function initializes environment
`animate()` function executes algorithms

Stencil Walkthrough

pendularm1.html

```
62  function init() {
63
64      // create pendulum object and its kinematic and dynamic parameters
65      pendulum = {length:2.0, mass:2.0, angle:Math.PI/2, angle_dot:0.0, angle_previous:0.0};
66
67      // initialize pendulum controls
68      pendulum.control = 0;
69      pendulum.desired = -Math.PI/2.5;
70
71      // initialize integral term accumulated error to zero
72      accumulated_error = 0;
73
74      // set gravity
75      gravity = 9.81;  // Earth gravity
76
77      // initialize pendulum PID servo gains
78      pendulum = set_PID_parameters(pendulum)
79
80      // initialize time and set timestep
81      t = 0;
82      dt = 0.05;  // default
```

Global variable initialization

Stencil Walkthrough

pendularm1.html

```
120   function animate() {
121
122       // note: three.js includes requestAnimationFrame shim
123       // alternative to using setInterval for updating in-browser drawing
124       // this effectively request that the animate function be called again for next draw
125       // http://learningwebgl.com/blog/?p=3189
126       requestAnimationFrame( animate );
...
219       // threejs rendering update
220       renderer.render( scene, camera );
221
222   }
223
```

Set up next call to `animate()`

Use three.js to render scene

Stencil Walkthrough

pendularm1.html

```
128      // switch between numerical integrators based on user input
129      if (keyboard.pressed("0"))
130          numerical_integrator = "none";
131      if (keyboard.pressed("1"))
132          numerical_integrator = "euler";
133      if (keyboard.pressed("2"))
134          numerical_integrator = "verlet";
135      if (keyboard.pressed("3"))
136          numerical_integrator = "velocity verlet";
137      if (keyboard.pressed("4"))
138          numerical_integrator = "runge-kutta";
139
140      // update servo desired state from user interaction
141      if ( keyboard.pressed("e") )
142          pendulum.desired += 0.05; // move the desired angle for the servo
143      if ( keyboard.pressed("q") )
144          pendulum.desired += -0.05; // move the desired angle for the servo
145
146
147      // add user force from user interaction
148      if ( keyboard.pressed("d") )
149          pendulum.control += 50.0;  // add a motor force to the pendulum motor
150      else if ( keyboard.pressed("a") )
151          pendulum.control += -50.0; // add a motor force to the pendulum motor
```

In every call to `animate()`, we check for keyboard input and update control variables

Stencil Walkthrough

update_pendulum_state.js

```
1  function update_pendulum_state(numerical_integrator, pendulum, dt, gravity) {
2      // integrate pendulum state forward in time by dt
3      // please use names 'pendulum.angle', 'pendulum.angle_previous', etc. in else codeblock between line 28-30
4
5      if (typeof numerical_integrator === "undefined")
6          numerical_integrator = "none";
7
8      if (numerical_integrator === "euler") {
9
10     // STENCIL: a correct Euler integrator is REQUIRED for assignment
11
12     }
13     else if (numerical_integrator === "verlet") {
14
15     // STENCIL: basic Verlet integration
16
17     }
18     else if (numerical_integrator === "velocity verlet") {
19
20     // STENCIL: a correct velocity Verlet integrator is REQUIRED for assignment
21
22     }
23     else if (numerical_integrator === "runge-kutta") {
24
25     // STENCIL: Runge-Kutta 4 integrator
26     }
27     else {
28         pendulum.angle_previous = pendulum.angle;
29         pendulum.angle = (pendulum.angle+Math.PI/180)%(2*Math.PI);
30         pendulum.angle_dot = (pendulum.angle-pendulum.angle_previous)/dt;
31         numerical_integrator = "none";
32     }
33
34     return pendulum;
35 }
```

Feature stencils

Default rotation

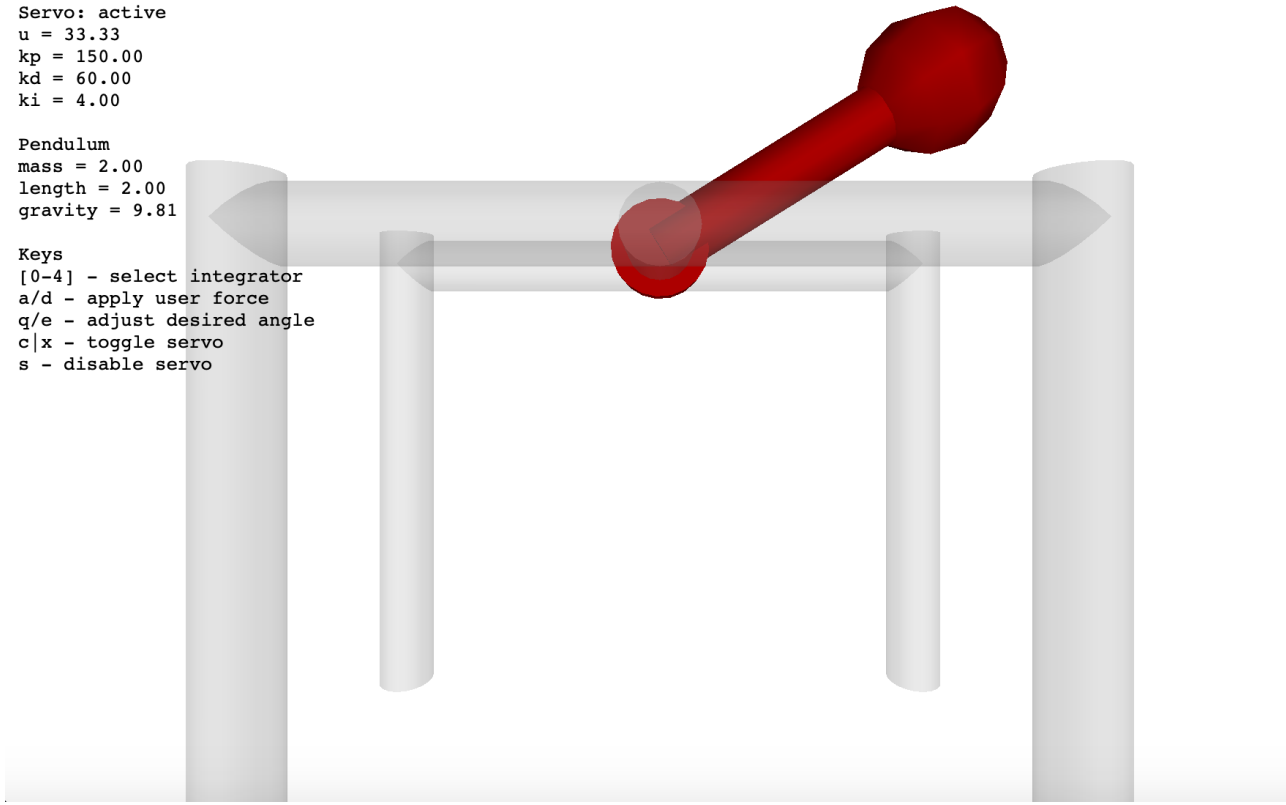
Pendularm Demo

```
System
t = 69.60 dt = 0.05
integrator = velocity verlet
x = -4.16
x_dot = 0.00
x_desired = -4.16

Servo: active
u = 33.33
kp = 150.00
kd = 60.00
ki = 4.00

Pendulum
mass = 2.00
length = 2.00
gravity = 9.81

Keys
[0-4] - select integrator
a/d - apply user force
q/e - adjust desired angle
c|x - toggle servo
s - disable servo
```



Coding Considerations

These concepts are optional, meant to help you on programming assignments

Concepts to consider for writing readable, easily debug-able code:

1. Use comments where complicated
2. Add whitespace for readability
3. Local variables to store indices/raw data
4. Helper functions that reduce code duplication

Using Comments

WITHOUT COMMENTS

```
var x = data;
var y = -1;
for (i=0; i<x.length; ++i){
   if (y<x[i]){
      y = x[i];
   }
}
```

WITH COMMENTS

```
// initialize data and min value so far
var x = data;
var y = -1;
// iterate over items in array x
for (i=0; i<x.length; ++i){
   // if current item in array is less than
   // min value so far
   if (y<x[i]){
      // update min value
      y = x[i];
   }
}
```

Using Whitespace

WITHOUT WHITESPACE

```
for (i=0; i<x.length; ++i){
   for (j=0; j<x[i].length; ++j){
      y = doStuff(i,j, x);
      doMoreStuff(y);
   }
}
```

WITH WHITESPACE AND COMMENTS

```
// iterate over every element in array x
for (i=0; i<x.length; ++i){
   for (j=0; j<x[i].length; ++j){

      // perform computation with current
      // position in x
      y = doStuff(i,j, x);

      // use result to do more stuff
      doMoreStuff(y);
   }
}
```

Local Variables for Temp Storage

COMPLICATED INDEX

Input: `G, node`

```
// index offset of neighbor
var offset = [0, 1];

// index into G at neighbor
G[node.i+offset[0]][node.j+offset[1]]
```

READABLE INDEX

Input: `G, node`

```
// index offset of neighbor
var offset = [0, 1];

// calculate indices and store in local var
var nbr_i = node.i+offset[0];
var nbr_j = node.j+offset[1];

// index into G at neighbor
G[nbr_i][nbr_j]
```

Helper Functions

DUPLICATED CODE

Input: `G, node`

```
//index into neighbors
nbr_u = G[node.i][node.j-1];
nbr_r = G[node.i+1][node.j];
nbr_d = G[node.i][node.j+1];
nbr_l = G[node.i-1][node.j];
…
//index into neighbors again
nbr_u = G[node.i][node.j-1];
nbr_r = G[node.i+1][node.j];
nbr_d = G[node.i][node.j+1];
nbr_l = G[node.i-1][node.j];
```

SINGLE FUNCTION, MULTIPLE CALLS

Input: `G, node`

```
function getNeighbors(node){
   nbr_u = G[node.i][node.j-1];
   nbr_r = G[node.i+1][node.j];
   nbr_d = G[node.i][node.j+1];
   nbr_l = G[node.i-1][node.j];

   return [nbr_u, nbr_r, nbr_d, nbr_l];
}
```