(Part 1) Randomness

Randomness
“Maybe the greatest novelty here is the ability of the computer not only to follow any complex rule of organization but also to introduce an exactly calculated dose of randomness.”

— E.H. Gombrich

Georg Nees, *Gravel Stones* (1971)
Random functions

random()
randomSeed()
noise()
random(lo, hi)

• **Return** a random number at least as big as lo but smaller than hi.
• **Get a different answer every time!**
• **If no value for “lo” is given, it is assumed to be 0.**

```javascript
function setup() {
  frameRate(1);
}

function draw() {
  let rn = random(0, 7);
  print(rn);
}
```

5.9743951260834836
1.5229306022682347
0.49242281849654534
6.452274518075551
1.8180111798881362
Random Integers

\[ \text{int}(\text{random}(0, N)) \]

- Choose a random integer from the set 0, 1, ... \( N-1 \)
- The int() function always rounds down

```javascript
function setup() {
  frameRate(1);
}

function draw() {
  let rn = int(random(0, 7));
  print(rn);
}
```
Flipping a coin

Write a function that simulates flipping a fair coin.
Flip 8 coins at a time
Print the number of heads and tails

Heads: 3  Tails: 5
Flipping 8 coins at a time (slide 1 of 2)

```javascript
let heads;
let tails;
let th = 0;
let tt = 0;

function preload() {
    heads = loadImage("heads.png");
    tails = loadImage("tails.png");
}

function setup() {
    createCanvas(800, 100);
    heads.resize(0, 100);
    tails.resize(0, 100);
    flipCoins();
}
```
function flipCoins() {
    background(50);
    let x = 0.0;
    while (x < width) {
        if (random(1) < 0.5) {
            image(heads, x, 0);
            x += heads.width;
            th++;
        } else {
            image(tails, x, 0);
            x += tails.width;
            tt++;
        }
    }
    print("Heads: " + th + " Tails: " + tt);
}

function mousePressed() {
    flipCoins();
}
Flipping a biased coin

Write a function that simulates flipping a biased coin.
75% heads, 25% tails
function flipCoins() {
  background(50);
  let x = 0.0;
  while (x < width) {
    if (random(1) < 0.5) {
      image(heads, x, 0);
      x += heads.width;
      th++;
    } else {
      image(tails, x, 0);
      x += tails.width;
      tt++;
    }
  }
}

• Take the coinFlips function from last example
• Change the line
  
  “if (random(1) < 0.5) {“
• to
  
  “if (random(1) < 0.75) {“
• Everything else remains the same
• Over time 75% of the flips will be heads. Try it yourself.
One million biased coin flips
75% heads, 25% tails

```
let th = 0;
let tt = 0;

function setup() {
    millionBiasedCoinFlips();
}

function millionBiasedCoinFlips() {
    for (let i = 0; i < 1000000; i++) {
        if (random(1) < 0.75) {
            th++;
        } else {
            tt++;
        }
    }
    print("Heads: "+th+" Tails: "+tt);
}
```

Heads: 750152 Tails: 249848

https://openprocessing.org/sketch/1097760
10 PRINT CHR$(205.5+RND(1)); : GOTO 10
let y = 0;
let gridSize = 30;

function setup() {
    createCanvas(300, 300);
    background(220);
}

function mousePressed() {
    for (let i = 0; i < 10; i++) {
        if (random(1) < 0.5) {
            line(i * gridSize, y, i * gridSize + gridSize, y + gridSize);
        } else {
            line(i * gridSize + gridSize, y, i * gridSize, y + gridSize);
        }
    } 
    y += gridSize;
}
Random Grid of Slashes (slide 2 of 2)

After 1 mouse press

After 2 mouse presses

After 3 mouse presses
Other Resources for 10print

- 10print.org
  - A website about 10print
- Daniel Shiffman video
  - https://www.youtube.com/watch?v=bEyTZ5ZZxZs
- Example from previous years of CS106. Not used in W21.
  - Week6_TenPrintRandomVis
https://www.youtube.com/watch?v=1cUufMeOijg

Cloudflare lobby [photo by @mahtin]
Which of the following lines of code might we place in the blank below, giving a function that simulates flipping a coin?

```javascript
function flipCoin() {
    // Code goes here
}
```

(A) `return random(1) < 0.5;`
(B) `return int(random(2)) === 0;`
(C) `return random(-50, 50) > 0.0;`
(D) `return int(random(6)) % 2 === 0;`
(E) All of the above
Suppose we wished to simulate rolling a six-sided die. Which expression below would be best way to obtain the number rolled?

(A) \( \text{random}(6) \)
(B) \( \text{random}(7) \)
(C) \( \text{int}(\text{random}(6)) \)
(D) \( \text{int}(\text{random}(6)) + 1 \)
(E) \( \text{int}(\text{random}(7)) \)
A fair coin is flipped ten times. Which of the following sequences of flips is the least likely to occur?

(A) H H H H H H H H H H

(B) H T H H T H T T T T H

(C) H T H T H T H T H T

(D) T T T H T T T H T H

(E) These are all equally likely
Is this sequence of digits random?

07021798609437027705392171762931767523846748184676694051320005681271
4523656093733572713432337572800172323717872146940001324052430146540585
3.141592653589793238462643383279502884197169399375105820974944592307 81640628620899862803482534211706798214808651328230664709384460955058 22317253594081284811174502841027019385211055596446229489549303819644 28810975665933446128475648233786783165271201909145648566923460348610 45432664821339360726024914127372458700660631558817488152092096282925 40917153643678925903600113305305488204665213841469519415116094330572 70365759591953092186117381932611793105118548074462379962749567351885 75272489122793818301194912983367336244065664308602139494639522473719 07021798609437027705392171762931767523846748184676694051320005681271 45263560827785771342757789609173637178721468440901224953430146549585 37105079227968925892354201995611212902196086403441815981362977477130 99605187072113499999983729780499510597317328160963185950244594553469 08302642522308253344685035261931188171010003137838752886587533208381 42061717766914730359825349042875546873115956286388235378759375195778 18577805321712268066130019278766111959092164201989380952572010654858 63278865936153381827968230301952035301852968995773622599413891249721 77528347913151557485724245415069595082953311686172785588907509838175 4637464939319
Most random number generators are like the digits of π: completely deterministic, but hard to predict.

These are called **Pseudorandom Number Generators** (PRNGs).
randomSeed(seed)

**Reset** the internal state of p5’s PRNG based on the passed-in seed. A given seed will always produce the same sequence of answers to a given sequence of calls to `random()`.
randomSeed() always produces the same result

```javascript
function setup() {
    createCanvas(400, 400);
    background(220);

    randomSeed(1);

    print(random(50));
    print(random(50));
    print(random(50));
    print(random(50));
    print(random(50));
}
```

https://openprocessing.org/sketch/1104392
let seed;

function setup() {
  createCanvas(800, 800);
  // It doesn't really matter what seed you
  // start with here.
  seed = 83567345;
}

function keyPressed() {
  if (key === "a") {
    seed -= 1;
  } else if (key === 'd') {
    seed += 1;
  }
}
function draw() {
    background(255);
    strokeWeight(15);
    randomSeed(seed);

    for (let row = 0; row < 12; ++row) {
        for (let col = 0; col < 10; ++col) {
            let x = col * 40;
            let y = row * 40;

            if (random(1) < 0.5) {
                // Draw a line from NW to SE
                line(x, y, x + 40, y + 40);
            } else {
                // Draw a line from NE to SW
                line(x + 40, y, x, y + 40);
            }
        }
    }

    textSize(24);
    text("Seed: "+seed, 450, 200);
}
Pseudorandom number generators are a double-edged sword.
The good: we can always “replay” a sequence of pseudorandom numbers.
The bad: pseudorandom numbers *are not actually random.*
let count = 1;
let ballX;
let ballY;

function setup() {
    createCanvas(500, 500);
    noFill();
    frameRate(10);
}
function draw() {
    background(220);
    randomSeed(0);
    let x = 0;
    let y = height / 2;
    beginShape();
    for (let i = 1; i < width; i = i + 5) {
        if (i === count) {
            ballX = x;
            ballY = y;
        }
        x = i;
        y = random(height / 2 - 20, height / 2 + 20);
        vertex(x, y);
    }
    endShape();
    ellipse(ballX, ballY, 10, 10);
    count = (count + 5) % width;
}
let seed = 395345;
let mySlider;

function setup() {
    createCanvas(500, 500);
    createP(" ");
    mySlider = createSlider(1, 1000, 100);
}

function keyPressed() {
    seed = seed + 1;
}
function draw() {
    randomSeed(seed);
    background(220);

    let numCircles = int(mySlider.value());

    for (let i = 0; i < numCircles; i++) {
        let d = random(5, 100);
        let x = random(width);
        let y = random(height);
        fill(random(255), random(255), random(255),random(255));
        stroke(random(255), random(255), random(255),random(255));
        ellipse(x, y, d, d);
    }
}
Goals

- Understand how to use random() to generate unpredictable behaviour.

- Understand how to use randomSeed() to control the generation of pseudorandom numbers.

- Understand the difference between random numbers and pseudorandom numbers.
(Part 2)
Noise
CS 106 Winter 2021
noise()

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

• It was invented by Ken Perlin in the 1980s and been used since in graphical applications to produce procedural textures, natural motion, shapes, terrains etc.
1D noise()

• Always returns a number between 0-1

• For any given run of your program the same argument always returns the same result.
  • noise(6);
    • Returns a number between 0-1
  • Another call noise(6);
    • Returns the same number
Remember random()

- random(1) returns a number between 0 and 1
- Calling random(1) again returns a different number between 0-1
- random(6) returns a number between 0-6
noise(x) always returns the same number

let v;
function setup() {
  let start = 100;
  v = noise(start); // v is between 0 and 1
  print(v);
  v = noise(start); // v is same number as the v above
  print(v);
  v = noise(start); // v is same number as both v above
  print(v);
}
Varying the noise() argument
noise() can return similar or dissimilar numbers

let v1;
let v2;
let v3;
function setup() {
    let start = 10;

    v1 = noise(start); // returns a number between 0-1

    v2 = noise(start + 0.001); // returns a num close to v1
        // num is between 0-1 always

    v3 = noise(start + 1); // returns a dissimilar num
        // number is between 0-1 always

    print(v1, v2, v3);
}
// Let's draw a smooth line
function setup() {
  createCanvas(600, 200);
  background(220);
  noFill();
  let v = 10;
  let vInc = 0.05;
  let space = 5;
  let numPoints = width / space;

  beginShape();
  for (let i = 0; i < numPoints; i++) {
    vertex(i * space, height/2 + (noise(v) * 100));
    v = v + vInc;
  }
  endShape();
}
Modify the above code:

\[ \text{vlncc} = 0.001; \]

- The line is not straight. But it doesn’t vary much. It is very smooth.
Modify the above code:
vlnc = 1.0;

• The line varies a lot. It is not a smooth line.
Moving Mountains (1 of 2)

let vStart = 10;
let v;
let vInc = 0.01;
let xoff = 0;

function setup() {
    createCanvas(600, 200);
    fill(0);
}

https://openprocessing.org/sketch/1105435
function draw() {
    v = vStart + xoff;
    background(220);
    beginShape();
    for (let i = 0; i < width; i++) {
        vertex(i, height/2 + (noise(v) * 100));
        v = v + vInc;
    }
    vertex(width, height);
    vertex(0, height);
    endShape();
    xoff = xoff + .025;
}
Moving a ball along a noisy line

- Demo code:
  - “BallOnNoisyLine”
let count = 1;
let v;
let vInc = 0.01;
let ballX;
let ballY;

function setup() {
    createCanvas(500, 500);
    noFill();
}
function draw() {
    background(220);
    v = 1;
    beginShape();
    for (let i = 1; i < width; i++) {
        let x = i;
        let y = map(noise(v), 0, 1, 100, 400);
        vertex(x, y);
        v = v + vInc;
        if (i === count) {
            ballX = x;
            ballY = y;
        }
    }
    endShape();
    ellipse(ballX, ballY, 10, 10);
    count = (count + 1) % width;
}
Demo Code

- Demo code:
  - “Noise1DDirectManip”
Direct Manipulation

• Use mouseDragged() function

• Calculate movement of the mouse (left-right or right-left)

• Use mouse movement as Direct Manipulation
let dx = 0;
let inc = 0.01;
let start = 10;
let v;

function setup() {
    createCanvas(600, 200);
}

https://openprocessing.org/sketch/1105530
function draw() {
  background(220);
  strokeWeight(2);
  stroke(255, 0, 0);
  noFill();
  v = start - dx;

  beginShape();
  for (let x = 0; x < 600; x++) {
    let y = map(noise(v), 0, 1, 0, height);
    vertex(x, y);
    v += inc;
  }
  endShape();
}

function mouseDragged() {
  dx += (mouseX - pmouseX) * inc;
}
2D Noise

- Go through demo code:
  - “Noise2DDirectManip”
let tx;
let ty;

// Scaling factor for the noise() function. Try
// changing this number!
let sc = 100.0;

function setup(){
  createCanvas(300, 300);
}

https://openprocessing.org/sketch/1106083
function draw() {
    background(220);
    for (let y = 0; y < height; ++y) {
        for (let x = 0; x < width; ++x) {
            v = noise((x - tx) / sc, (y - ty) / sc);
            set( x, y, color( v * 256.0 ) );
        }
    }
    updatePixels();
}

function mouseDragged() {
    tx += mouseX - pmouseX;
    ty += mouseY - pmouseY;
}
Goals

• Be able to write short sketches that use the noise() function.

• Understand how noise() works in 1D and 2D, especially 1D.

• Understand the difference between random() and noise().
Which of these expressions is NOT guaranteed to return a number between 0 and 1?

(A) `random(100) / 100.0`
(B) `random(0, 1);`
(C) `noise(3);`
(D) `noise(0, 1);`
(E) They all return numbers between 0 and 1.
Assume we have the following two lines of code:
\[
\text{let } a = \text{noise}(99.0); \\
\text{let } b = \text{noise}(99.01);
\]
If noise(99) returns “0.5”, which is the most likely outcome returned by noise(99.01)?

(A) 1.49  
(B) 0.75  
(C) 0.51  
(D) 99.51  
(E) A number between 99 and 99.01
Assume we have the following line of code:

```javascript
let a = noise(99.0);
```

Which is the most likely value of “a”?

(A) \(-0.43\)

(B) \(99\)

(C) \(0.43\)

(D) \(A \text{ number between } 0-99\)
The following 3 clicker questions are about this code:

```javascript
createCanvas(400, 100);
let v = noise(10);
let x1 = 100 + (v * 100);
let x2 = x1 + 100;
line(x1, 50, x2, 50);
```

What might the value of “v” be?

(A) 10

(B) A number between 0-1

(C) A number between 0-10
The following 3 clicker questions are about this code:

```javascript
createCanvas(400, 100);
let v = noise(10);
let x1 = 100 + (v * 100);
let x2 = x1 + 100;
line(x1, 50, x2, 50);
```

What might the value of “x1” be?

(A) A number between 0-100
(B) A number between 100-200
(C) A number between 100.0-101.0
The following 3 clicker questions are about this code:

```javascript
createCanvas(400, 100);
let v = noise(10);
let x1 = 100 + (v * 100);
let x2 = x1 + 100;
line(x1, 50, x2, 50);
```

What might the value of “x2” be?

(A) A number exactly 100 larger than x1

(B) A number between 100-200

(C) A number between 200-300
Remember this ex from CS105
“Similar” code is needed this week
Let’s Review the code (next slide)
let shade = 0;

function setup() {
    createCanvas(100, 255);
    background(220);

    for (let y = 0; y <= height; y++) {
        stroke(shade);
        line(0, y, width, y);
        shade += 1;
    }
}
Dan Shiffman videos on noise()

• Dan Shiffman has several excellent videos on noise()
  • I.2: Introduction - Perlin Noise and p5.js Tutorial
    • https://www.youtube.com/watch?v=Qf4dIN99e2w
  • I.3: noise() vs random() - Perlin Noise and p5.js Tutorial
    • https://www.youtube.com/watch?v=YcdldZ1E9gU
  • I.4: Graphing 1D Perlin Noise - Perlin Noise and p5.js Tutorial
    • https://www.youtube.com/watch?v=y7sgcFhk6ZM
  • I.5: 2D Noise - Perlin Noise and p5.js Tutorial
    • https://www.youtube.com/watch?v=ikwNrFvnL3g