p5.js Tutorial

Part A — Core p5.js Concepts

1) The p5.js “sketch lifecycle”

p5.js runs your program through a predictable lifecycle:

- `setup()` runs **once** at the start
- `draw()` runs **every frame** (default ~60 FPS)

```
function setup() {
  createCanvas(600, 400);
}

function draw() {
  background(220);
  // draw your frame here
}
```

Key idea: **animation = “redraw a new picture many times per second.”**

2) Canvas, coordinates, and units

- The canvas origin `(0, 0)` is **top-left**
- `x` increases to the right; `y` increases downward
- `width`, `height` are canvas dimensions (set by `createCanvas(w, h)`)

Useful debugging helpers:

```
point(mouseX, mouseY);
print(mouseX, mouseY);
```

3) Drawing primitives

Common shapes (all in pixels):

- `circle(x, y, d)`
- `rect(x, y, w, h)`
- `square(x, y, s)`
- `line(x1, y1, x2, y2)`

Defaults:

- `rect()` uses top-left corner positioning (unless you change rectMode)

4) Color + style state

p5.js uses a **state machine** style system:

- `fill(r, g, b)` sets shape fill color
- `stroke(r, g, b)` sets outline
- `noFill()` / `noStroke()` disable
- Values range 0–255 (by default)

```
background(220);
stroke(0);
fill(255, 0, 0);
circle(100, 100, 40);
```

5) Variables, scope, and animation patterns

- **Global variables** persist across frames (crucial for animation).
- **Local variables** exist only inside the function call.

Typical animation patterns:

- position update: `x += vx;`
- random walk: `y += random(-3, 3);`
- wrap/clamp: `x = constrain(x, 0, width);`

6) Conditionals: make visuals respond

Conditionals enable interaction / logic:

```
if (mouseX < width/2) {
  fill(255, 0, 0);
} else {
  fill(255);
}
square(100, 100, 30);
```

Common comparisons: `<`, `>`, `<=`, `>=`, `==`, `!=`

7) Loops: repetition for patterns

- `for` is best for known counts (grids, repeated shapes)
- `while` is best when you repeat until a condition becomes false

Grid template:

```
for (let r = 0; r < rows; r++) {
  for (let c = 0; c < cols; c++) {
    rect(c*w, r*h, w, h);
  }
}
```

8) Randomness and distance

- `random(a, b)` gives a float in `[a, b)`
- `dist(x1, y1, x2, y2)` gives Euclidean distance

Classic “hover hit-test” for a circle of radius `R`:

```
if (dist(cx, cy, mouseX, mouseY) <= R) { ... }
```

9) `map()` for smooth transitions

`map(value, inMin, inMax, outMin, outMax)` linearly maps one range to another.

Example: mouse controls grayscale:

```
let g = map(mouseX, 0, width, 0, 255);
background(g);
```

10) Arrays, objects, and events

Arrays: ordered lists, index starts at 0

Objects: named properties (`{x: 10, y: 20}`)

Events (run only when triggered):

- `mouseClicked()`
- `keyPressed()`

For “draw once” sketches, use `noLoop()` to stop `draw()` .

Part B — Midterm Practice

Q21 — Mouse click moves the circle

Goal: add `mouseClicked()` so the circle jumps to the mouse position.

```
let xPos;
let yPos;

function setup() {
  createCanvas(500, 400);
  xPos = width / 2;
  yPos = height / 2;
}

function draw() {
  background(220);
  circle(xPos, yPos, 40);
}
```

Q21 — Mouse click moves the circle

```
function mouseClicked() {
  xPos = mouseX;
  yPos = mouseY;
}
```

Why this works:

- `mouseClicked()` is an event callback.
- Update globals so `draw()` renders the new position every frame.

Q20 — Sum of odd positive numbers between a and b

Requirement: sum all **odd** and **positive** integers in `[a, b]`.

```
function oddsFromAToB(a, b) {
  let lo = min(a, b);
  let hi = max(a, b);

  let sum = 0;
  for (let i = lo; i <= hi; i++) {
    if (i > 0 && i % 2 !== 0) sum += i;
  }
  return sum;
}
```

Notes:

- Use `min/max` so your function is robust to `a > b`.
- `i % 2 !== 0` detects odd integers.

Q19 — Sum all numbers from a to b (inclusive)

```
function sumFromAToB(a, b) {
  let lo = min(a, b);
  let hi = max(a, b);

  let sum = 0;
  for (let i = lo; i <= hi; i++) sum += i;
  return sum;
}
```

Quick check:

- `sumFromAToB(1, 3) = 1 + 2 + 3 = 6`

Q18 — Find the smallest number in an array

Do not use built-in `max()` / `min()` on the array directly.

```
function smallest(arr) {
  let best = arr[0];
  for (let i = 1; i < arr.length; i++) {
    if (arr[i] < best) best = arr[i];
  }
  return best;
}
```

Pattern: “initialize with first element, then improve.”

Q17 — Square a number

```
function squared(n) {
  return n * n;
}
```

That’s it. Also acceptable: `return pow(n, 2);`
(but `n*n` is the simplest and fastest here).

Q16 — Implement `dividedRect(x, y, w, h)`

The prompt asks for a function with the same parameters as `rect()` that draws a rectangle with “divisions”.

A standard interpretation (commonly used in midterms): draw the rectangle and add a vertical + horizontal divider.

```
function dividedRect(x, y, w, h) {
  rect(x, y, w, h);
  line(x + w / 2, y, x + w / 2, y + h);
  line(x, y + h / 2, x + w, y + h / 2);
}
```

Q16 — Implement `dividedRect(x, y, w, h)`

Example usage:

```
function setup() {
  createCanvas(800, 400);
  background(255);
  dividedRect(50, 50, 300, 200);
  dividedRect(400, 80, 250, 250);
}
```

Q15 — Update the 3rd object in an array

“3rd element” means index `2`.

```
let positions = [
  { x: 20, y: 30 },
  { x: 100, y: 30 },
  { x: 40, y: 80 },
  { x: 120, y: 100 },
];

function setup() {
  createCanvas(400, 100);
  positions[2].x = 50;
}
```

Q14 — Create an object `circ` with x, y, radius

```
let circ;

function setup() {
  createCanvas(600, 400);
  circ = { x: 200, y: 150, radius: 30 };
}
```

You can then use: `circle(circ.x, circ.y, circ.radius * 2);`

Q13 — Display one random character exactly once

Key trick: do it in `setup()` or call `noLoop()`.

```
let arr = ["A", "B", "C", "D"];

function setup() {
  createCanvas(600, 400);
  background(220);

  textAlign(CENTER, CENTER);
  textSize(64);

  let i = floor(random(arr.length));
  text(arr[i], width / 2, height / 2);

  noLoop(); // ensure “exactly once”
}

function draw() {}
```

Q12 — Random walk along the y-axis

To move along the **y-axis**, the y-coordinate must change.

```
let y = 300;

function draw() {
  background(200);

  y += random(-3, 3);        // random step
  y = constrain(y, 0, height);

  circle(50, y, 20);         // x fixed, y changes
}
```

Tip: `constrain()` prevents drifting off-screen.

Q11 — Fill an array with 100 random numbers

```
let arr = [];

function setup() {
  createCanvas(600, 400);

  for (let i = 0; i < 100; i++) {
    arr.push(random(0, 100));
  }

  print(arr.length); // 100
}
```

Q10 — Replace the 2nd array item

Remember: “2nd item” is index `1`.

```
let arr = [10, 20, 30, 40, 50];

function setup() {
  createCanvas(600, 400);
  arr[1] = random(0, 100);
  print(arr);
}
```

Q9 — Background transitions from red to white using `map()`

Red = `(255, 0, 0)`
White = `(255, 255, 255)`

So: `r` stays 255; `g` and `b` increase from 0 to 255 as `mouseX` moves left→right.

```
let r, g, b;

function setup() {
  createCanvas(600, 400);
}

function draw() {
  r = 255;
  g = map(mouseX, 0, width, 0, 255);
  b = map(mouseX, 0, width, 0, 255);

  background(r, g, b);
}
```

Q8 — Map `i` to circle size (concentric circles)

We want 4 circles (i = 0..3) where size changes smoothly.

A typical solution: largest at `i=0`, smallest at `i=3`.

```
function setup() {
  createCanvas(200, 200);
  for (let i = 0; i < 4; i++) {
    fill(i * 50);
    let sz = map(i, 0, 3, 160, 40);
    circle(100, 100, sz);
  }
}
```

How to tune:

- Decrease 160 if it touches canvas edges.
- Increase 40 if the center circle is too small.

Q7 — What prints to the console?

```
let i = 5;

function setup() {
  createCanvas(600, 400);
  let num = map(i, 0, 10, 200, 0);
  console.log(num);
}
```

Reasoning:

- 0 maps to 200, 10 maps to 0 (decreasing line)
- 5 is halfway, so output is halfway between 200 and 0 → **100**

Q6 — Grid of rectangles (8 columns × 7 rows)

Canvas is 800×700, so each cell is: cell width = 800 / 8 = 100; cell height = 700 / 7 = 100

```
function setup() {
  createCanvas(800, 700);
  background(220);

  let cols = 8;
  let rows = 7;
  let w = width / cols;
  let h = height / rows;

  for (let r = 0; r < rows; r++) {
    for (let c = 0; c < cols; c++) {
      rect(c * w, r * h, w, h);
    }
  }
}
```

Q5 — Turn circle green when mouse is inside it

Given moving `x, y` and distance `d`:

```
let x = 300;
let y = 200;

function setup() {
  createCanvas(600, 400);
}
```

Q5 — Turn circle green when mouse is inside it

Given moving `x, y` and distance `d`:

```
function draw() {
  background(220);

  x = x + random(-3, 3);
  y = y + random(-3, 3);

  let d = dist(x, y, mouseX, mouseY);
  let radius = 10; // because diameter is 20

  if (d <= radius) fill(0, 255, 0);
  else fill(0);

  circle(x, y, 20);
}
```

Q4 — Circle centered at mouse position

Only one line is needed inside `draw()`:

```
let diam = 20;

function setup() {
  createCanvas(600, 400);
}

function draw() {
  background(220);
  circle(mouseX, mouseY, diam);
}
```

Q3 — While-loop: odd numbers 97 down to 3 (exactly once)

Because printing should happen once, do it in `setup()`.

```
function setup() {
  createCanvas(600, 400);

  let n = 97;
  while (n >= 3) {
    print(n);
    n -= 2;
  }

  noLoop();
}

function draw() {
  background(220);
}
```

Q2 — Random x each frame + conditional color

Specification:

- each frame: `x` random in `[0, width)`
- draw 30×30 square at x
- red if left half, white if right half

```
let x;

function setup() {
  createCanvas(600, 400);
}
```

Q2 — Random x each frame + conditional color

```
function draw() {
  background(220);

  x = random(0, width);

  if (x < width / 2) fill(255, 0, 0);
  else fill(255);

  square(x, height / 2 - 15, 30);
}
```

Q1 — Variable `size` grows by 10 each frame

Make `size` global so it persists across frames:

```javascript
let size = 15;

function setup() {
  createCanvas(600, 400);
}

function draw() {
  background(220);

  circle(width / 2, height / 2, size);

  size += 10;
}
```

Summary

Common exam pitfalls to avoid

- Confusing diameter vs radius (circle hit-tests)
- Forgetting array indices start at 0
- Updating variables inside `draw()` but declaring them inside `draw()` (they “reset” each frame)
- Using `draw()` for “print once” tasks (use `setup()` / `noLoop()`)

Suggested practice extensions

Pick any 3 questions and extend them:

- Add keyboard controls (`keyPressed`)
- Replace `random()` with `noise()` for smoother motion
- Turn solutions into reusable functions (e.g., `drawGrid(cols, rows)`)