COMP1000 — Week 5: Nested Loops & Control Flow (Processing)

Greg Baker

26th August 2025

Today

- Build nested `for` loops (grid/table mental model).
- Put `if` statements *inside* loops to style/classify values.
- Learn two power moves: `continue` and `break`.
- Understand loop variable scope: `for (int loopCounter = ...)`.
- Think in frames: the screen updates after `draw()` finishes.

Administrative reminders

To do any assessments in iLearn (there are assessments coming up for COMP1000), you need to have already:

- Completed the **Academic Integrity Module**.
- Finished the **Safer Communities @ MQ Module**.

You can request not to do Safer Communities (e.g. from trauma) but approval can take a while.

If you can't sit next week

- If you can't sit next week, there is no need to submit a special consideration; you'll just sit it the next week.

Common syntax mistakes we saw in the practice exam

- **If statements need parentheses:** `if (x > 3) { ... }`.
- **Use >= not >==:** the latter won't compile.

Calling functions & arguments

- Call functions with parentheses: `fill(255, 0, 0);`.
- Values inside are the **arguments** passed to the function.
- Example: `line(x1, y1, x2, y2)` has four arguments.

Quick warm-up (2 mins)

- What does `i % 5` mean? $\rightarrow$ remainder after division by 5.
- True/false: `12 % 3 == 0` (true), `14 % 5 == 0` (false).
- If a number is a multiple of both 3 and 5, it's a multiple of 15.

Reminders about how the screen is drawn

- Processing runs `draw()` many times per second.
- Your drawing calls (`background`, `rect`, ...) take effect when `draw()` **finishes**.
- So: no mid-`draw()` updates. Drive changes with **time** (`millis()`, `frameCount`) or **input** (keys, mouse).

Anatomy of a `for` loop

```
1 for (int i = 0; i < 10; i++) {
2   // statement
3 }
```

- **Initializer:** `int i = 0` (runs once at the start).
- **Condition:** `i < 10` (checked every time; just before the statement has been run).
- **Afterthought:** `i++` (runs after each iteration; just after the statement).
- **Statement:** the body between { and }.

Odd but valid `for` loops

```
1  int i = 0;
2  for (; i < 5; i++) { println(i); }   // no initializer
3  for (int j = 0; ; j++) {  }          // no condition (infinite)
4  for (int k = 0; k < 5; ) { k++; }    // no afterthought
5  for ( ; ; ) { }                      // they are all optional
```

Missing pieces change how the loop behaves—use with care!

What's with k++?

- ++ after a variable means "after using this variable, increment it"
- ++ before a variable means "before using this variable, increment it"

These are the same program:

```
1  int i;
2  void draw() {
3     background(i++);
4     i %= 256;
5  }
```

```
1  int i;
2  void draw() {
3     background(i);
4     i = (i + 1) % 256;
5  }
```

And these are not quite the same, but you won't see a difference

```
1  int i;
2  void draw() {
3     background(i++);
4     i %= 256;
5  }
```

```
1  int i;
2  void draw() {
3     background(++i);
4     i %= 256;
5  }
```

These all draw the same thing

```
1  void draw() {
2    for (int i=0;
       i<20; i++) {
3      point(i,i);
4    }
5  }
```

```
1  void draw() {
2    for (int i=0;
       i<20; i=i+1) {
3      point(i,i);
4    }
5  }
```

```
1  void draw() {
2    for (int i=0;
       i<20; ++i) {
3      point(i,i);
4    }
5  }
```

```
1  void draw() {
2    for (int i=0;
       i<20; i+=1) {
3      point(i,i);
4    }
5  }
```

```
1  void draw() {
2    int i = 0;
3    for (;i<20;) {
4      point(i,i);
5      i += 1;
6    }
7  }
```

```
1  void draw() {
2   int i = 0;
3   boolean run = true;
4   for (;run;) {
5     point(i,i);
6     i += 1;
7     if (i == 20) {
8       run = false;
9     }
10  }
11 }
```

Hair-pulling exercise: common mistakes

```
1 int i = 0;
2 i = i++;
```

Hair-pulling exercise: common mistakes

```
1 int i = 0;
2 i = i++;
```

- The value of *i* now 0

Hair-pulling exercise: common mistakes

```
1 int i = 0;
2 i = i++;
```

- The value of *i* now 0
- (post-increment returns old value)

You don't have to increment by one

```
6  int stripeWidth = 40;
7  void draw() {
8    boolean hovered;
9    boolean dark = false;
```

- `width` is the width of the screen
- `left` will get incremented with a *stride* of `stripeWidth`

You don't have to increment by one

```
10    for (int left = 0; left < width; left += stripeWidth) {
11      int right = left + stripeWidth - 1;
12      hovered = mousePressed && mouseX >= left && mouseX <= right;
13      if (hovered) {
14        fill(255,0,0);
15      } else if (dark) {
16        fill(0,0,0);
17      } else {
18        fill(255,255,255);
19      }
20      rect(left, 0, stripeWidth, height);
21      dark = !dark;
22    }
23  }
```

Where FizzBuzz came from & why we still use it

FizzBuzz started as a counting game for kids: say the numbers, but say "Fizz" for multiples of 3, "Buzz" for multiples of 5, and "FizzBuzz" for multiples of both.

In programming, it's a classic beginner/interview task because it checks:

- Using % (modulo) to detect multiples.
- Ordering of conditions (`15` before `3` or `5`).
- Writing clean, readable control flow.

Why you care here: we'll push it into visuals (colour, layout) to tie logic to graphics in Processing.

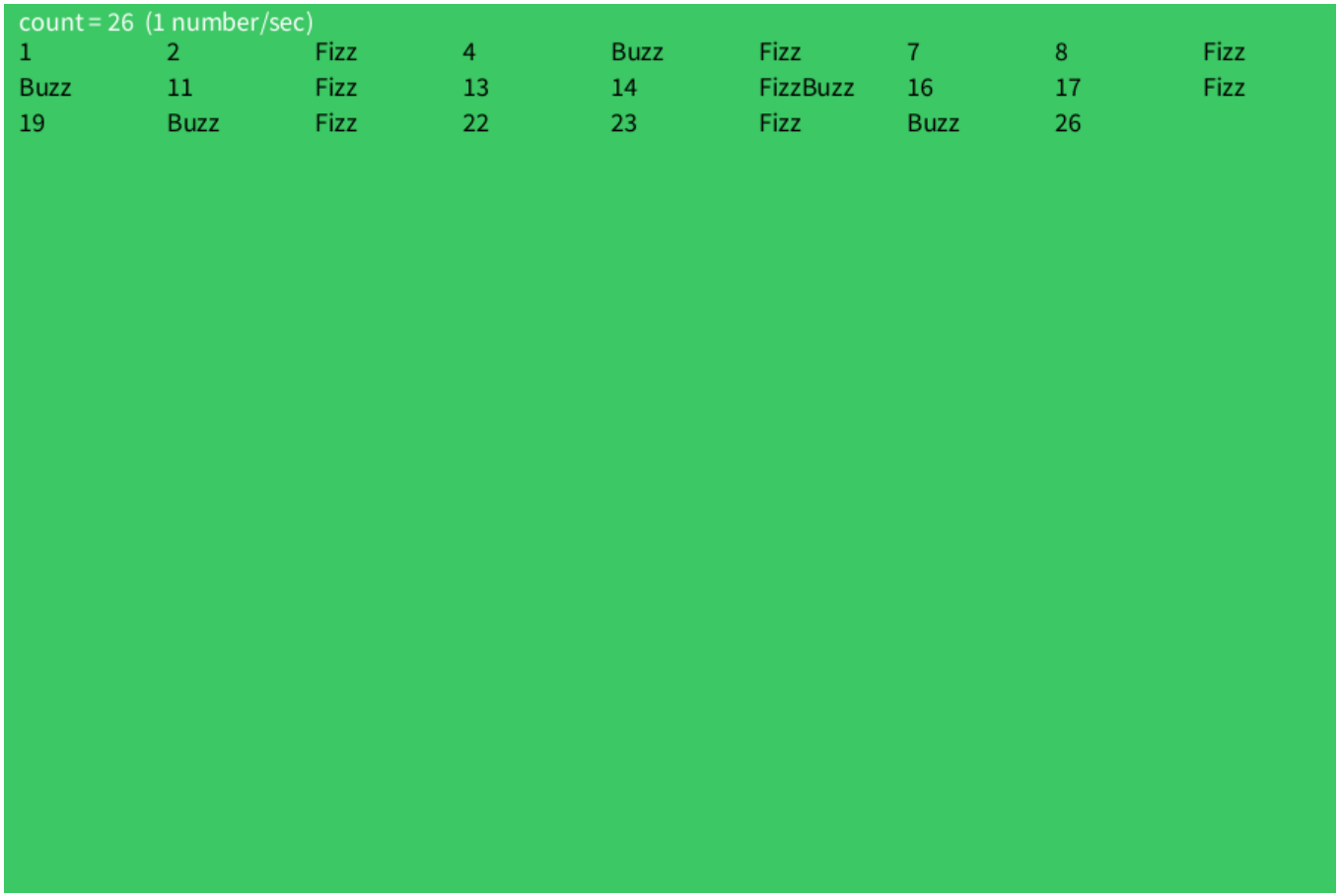
Program 1 — Time-Driven FizzBuzz (simple)

Goal: Colour the whole background based on the current count `currentCount`. Print all FizzBuzz outputs from `1..currentCount` in rows. No scrolling.

New ideas here:

- Derive `currentCount` from seconds since start: `(millis() - startMilliseconds)/1000`.
- Map a 1D index to grid positions using integer division/mod.
- Use `if/else if/else` to classify each value.
- Curly braces {} are optional after an `if` when there is only one statement.

Simple FizzBuzz — output



Simple FizzBuzz

```
1  int startMilliseconds;
2  int columnWidth = 100;  // horizontal spacing (px)
3  int rowHeight = 24;     // vertical spacing (px)
4  int columnCount, rowCount;
5
6  void setup() {
7    size(900, 600);
8    textSize(18);
9    textAlign(LEFT, TOP);
10   startMilliseconds = millis();
11
12   columnCount = width / columnWidth;
13   rowCount = height / rowHeight;
14 }
```

Simple FizzBuzz

```
16  void draw() {
17    // currentCount = seconds since start (1 per second), capped
        to what fits on screen
18    int elapsedSeconds = (millis() - startMilliseconds) / 1000;
19    int maxItems = columnCount * rowCount;
20    int currentCount = 1 + elapsedSeconds;
21    if (currentCount > maxItems) currentCount = maxItems;
22
23    // whole background colour based on currentCount
24    if (currentCount % 15 == 0)      background(220, 0, 220);
        // FizzBuzz
25    else if (currentCount % 3 == 0)  background(240, 60, 60);
        // Fizz
26    else if (currentCount % 5 == 0)  background(60, 90, 240);
        // Buzz
27    else                             background(60, 200, 100);
        // plain
```

What to notice

- `if` doesn't need braces if there is only one statement
- But it always needs parentheses around the boolean expression

Simple FizzBuzz

```
31    for (int currentNumber = 1; currentNumber <= currentCount;
        currentNumber++) {
32      int zeroBasedIndex = currentNumber - 1;
33      int columnIndex = zeroBasedIndex % columnCount;
34      int rowIndex = zeroBasedIndex / columnCount;
35      float xPosition = columnIndex * columnWidth + 10;
36      float yPosition = rowIndex * rowHeight + 10;
37
38      if (currentNumber % 15 == 0)
39        text("FizzBuzz", xPosition, yPosition);
40      else if (currentNumber % 3 == 0)
41        text("Fizz", xPosition, yPosition);
42      else if (currentNumber % 5 == 0)
43        text("Buzz", xPosition, yPosition);
44      else
45        text(currentNumber, xPosition, yPosition);
46    }
```

What to notice

- The loop variable `currentNumber` is **local** to the `for` loop.
- `zeroBasedIndex/columnCount` gives the row index; `zeroBasedIndex%columnCount` gives the column index.
- Condition order matters: check 15 before 3 or 5.
- We redraw everything each frame. That's fine at this size.

FizzBuzz Grid — output

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220
221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240

Program 2 — FizzBuzz Grid (nested loops)

Goal: Use a **loop inside a loop** to make a table of numbers and colour the cells by FizzBuzz rules.

New ideas here:

- Outer loop = rows, inner loop = columns.
- Compute `cellValue = rowIndex * columnCount + columnIndex + 1`.
- Style each cell via `if` conditions.

FizzBuzz Grid

```
1  int columnCount = 20, rowCount = 12;
2  float cellWidth, cellHeight;
3
4  void setup() {
5    size(900, 540);
6    textAlign(CENTER, CENTER);
7    textSize(12);
8    noStroke();
9    cellWidth = width / (float)columnCount;
10   cellHeight = height / (float)rowCount;
11 }
```

FizzBuzz Grid

```
13  void draw() {
14    background(245);
15    for (int rowIndex = 0; rowIndex < rowCount; rowIndex++) {
16      for (int columnIndex = 0; columnIndex < columnCount;
        columnIndex++) {
17        int cellValue = rowIndex * columnCount + columnIndex + 1;
          // 1..(rowCount*columnCount)
18        float xPosition = columnIndex * cellWidth;
19        float yPosition = rowIndex * cellHeight;
20        if (cellValue % 15 == 0)      fill(200, 80, 200);
21        else if (cellValue % 3 == 0)  fill(255, 120, 120);
22        else if (cellValue % 5 == 0)  fill(120, 140, 255);
23        else                          fill(230);
24        rect(xPosition, yPosition, cellWidth, cellHeight);
25        fill(30);
26        text(cellValue, xPosition + cellWidth/2, yPosition +
        cellHeight/2);
27      }
```

Reminder: floor notation

- $\lfloor x \rfloor$ means "the greatest integer $\leq x$".
- In code, use `floor(x)` for floating-point numbers.
- Casting with `int(x)` already floors positive values, so we don't need `floor()` here.

Mental model & quick checks

Grid indexing:

$$\text{cellValue} = \text{rowIndex} \times \text{columnCount} + \text{columnIndex} + 1$$

$$\Rightarrow \quad \text{rowIndex} = \left\lfloor \frac{\text{cellValue} - 1}{\text{columnCount}} \right\rfloor,$$

$$\text{columnIndex} = (\text{cellValue} - 1) \bmod \text{columnCount}$$

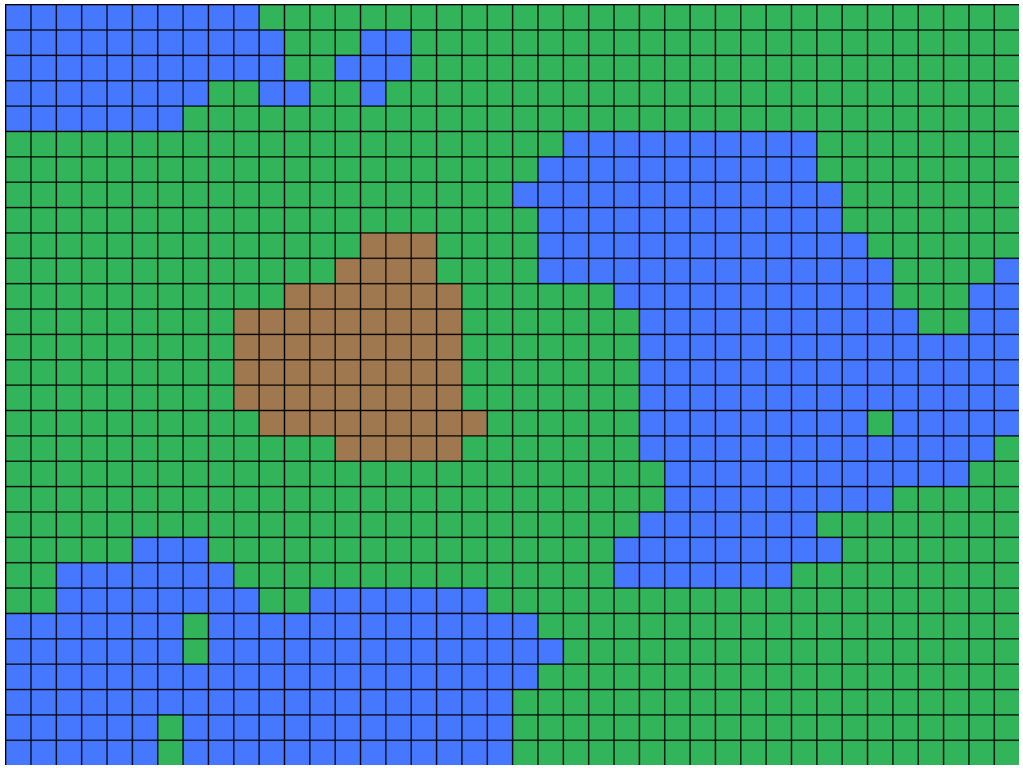
Ask the room: Where is cell value 73 when `columnCount = 20`? (Row 3, Column 12 — zero-based.)

Common mistakes:

- Swapping row/column (cells flip across the diagonal).
- Forgetting +1 in the cell value formula.

Terrain generation

Many computer games automatically generate terrain (land, water, hills). The key is a good random number generator



Perlin noise — smooth randomness for animation

- **Coherent** pseudo-random function $N(x, y, z)$: nearby inputs $\Rightarrow$ similar outputs.
- **Gradient noise**: random gradients on a lattice, smoothly interpolated.
- In Processing, `noise()` returns **0..1**, is **deterministic** for the same inputs.
- Use a third input (`z` or time) to animate

Why not `random()`?

- `random()` is uncorrelated per call (TV static).
- `noise()` is spatially/temporally correlated (natural clouds, terrain, motion).

How to use `noise()`

- Sampling scale: use fractions, e.g. `noise(x*0.07, y*0.07)` for larger features.
- Remap: `val = noise(..)*2 - 1` for $-1..1$.

Gotchas

- Integer division: `width/(float)cols`, not `width/cols`.
- Stepping by whole pixels (`x+=1`) gives high frequency; sample more finely for smoothness.

Drawing up a grid and populating it with Perlin noise

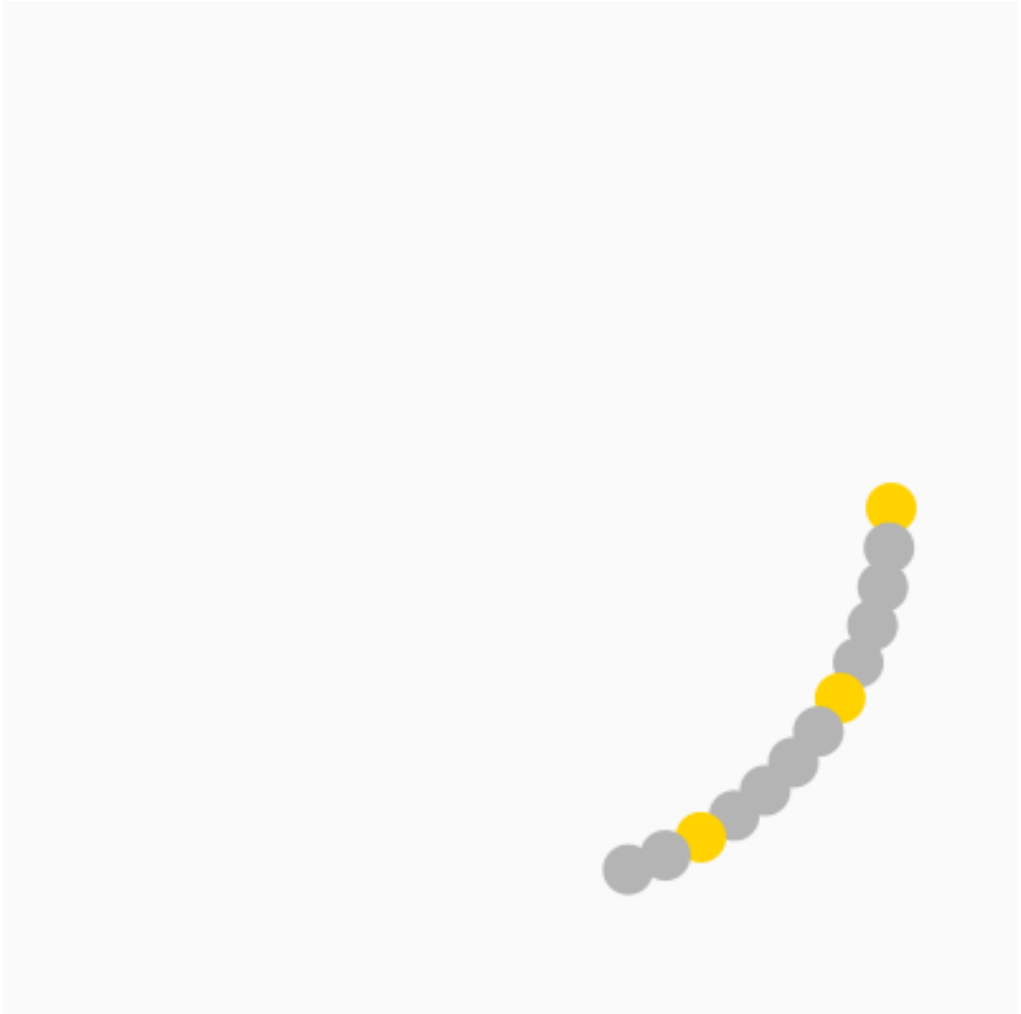
```
4  int cols = 40, rows = 30;
5  float z = 0;
6  void draw() {
7    z += 0.001; // animate noise over time
8    float cw = width / (float) cols;
9    float ch = height / (float) rows;
10   for (int r = 0; r < rows; r++) {
11     for (int c = 0; c < cols; c++) {
12       float n = noise(c*0.07, r*0.07, z); // 0..1
13       // Multi-way classification (order matters)
14       if (n < 0.45)          fill(70, 120, 255);  // water
15       else if (n < 0.65)     fill(50, 180, 90);   // plains
16       else if (n < 0.85)     fill(160, 120, 80);  // hills
17       else                   fill(230);           // peaks
18       rect(c*cw, r*ch, cw+1, ch+1); // +1 to hide gaps
19     }
20   }
21 }
```

`break` demo

Goal: Draw jewels around a clockface and stop when we reach the current second.

New idea: `break` exits the current loop immediately.

Clockface break — output



Clockface break

```
6  int totalTickCount = 60;
7  float circleRadius = 150;
8  float jewelDiameter = 20;
9  void draw() {
10   background(250);
11   translate(width/2, height/2);
12   for (int tickIndex = 0; tickIndex < totalTickCount;
       tickIndex++) {
13     if (tickIndex % 5 == 0) fill(255, 210, 0);
14     else fill(180);
15     float angle = TWO_PI * tickIndex / totalTickCount;
16     float xPosition = cos(angle) * circleRadius;
17     float yPosition = sin(angle) * circleRadius;
18     ellipse(xPosition, yPosition, jewelDiameter, jewelDiameter);
19     if (tickIndex == second()) {
20       break; // stop drawing more jewels this frame
21     }
22   }
```

What to notice

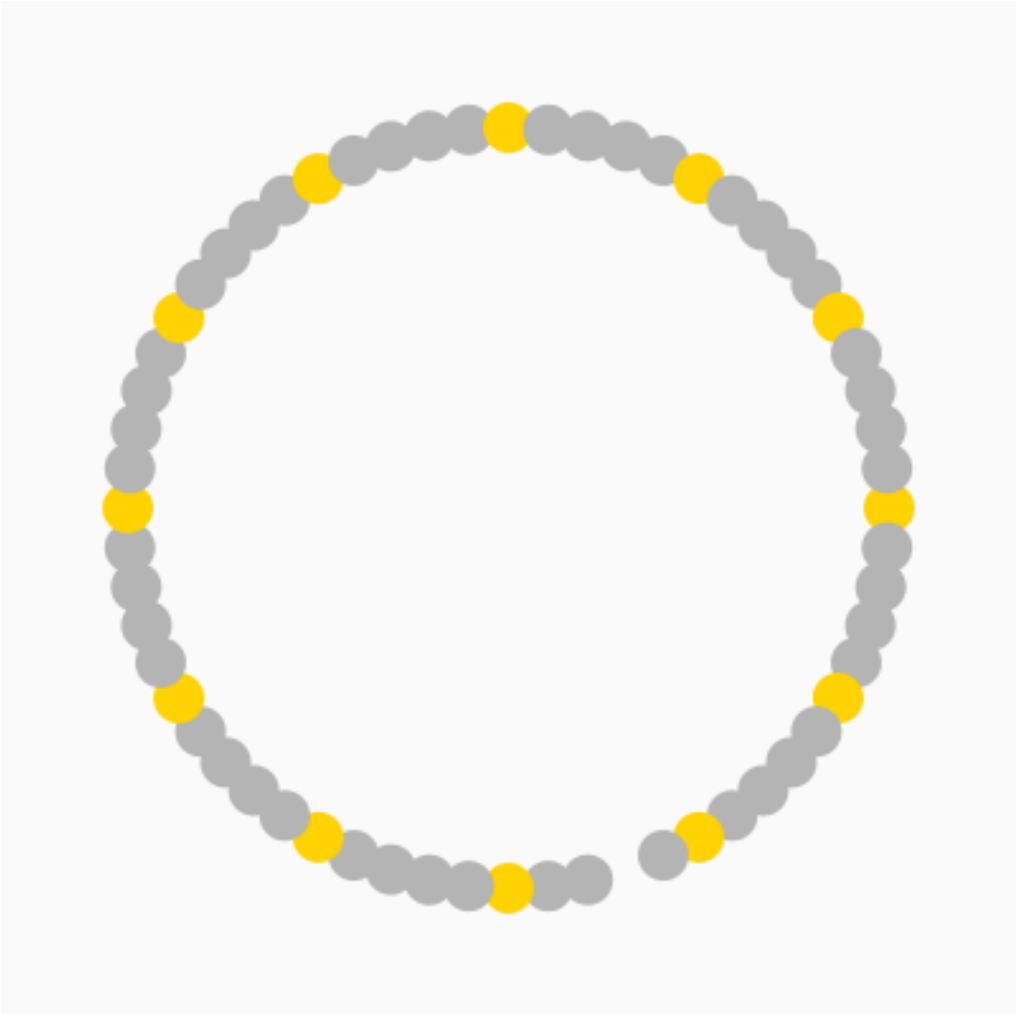
- `translate()` adjusts where (0,0) is: we moved that to the centre of the screen
- The loop stops once the jewel at the current second is drawn.
- Every fifth jewel is coloured differently.
- Could I have done this by adding an extra condition in the termination part of the `for` loop? Yes.

`continue` demo

Goal: Draw all jewels but skip the one at the current second.

New idea: `continue` skips the rest of the current iteration.

Clockface continue — output



Clockface continue

```
6  int totalTickCount = 60;
7  float circleRadius = 150;
8  float jewelDiameter = 20;
9  void draw() {
10   background(250);
11   translate(width/2, height/2);
12   for (int tickIndex = 0; tickIndex < totalTickCount;
       tickIndex++) {
13     if (tickIndex == second()) {
14       continue; // skip drawing the jewel at the current second
15     }
16     if (tickIndex % 5 == 0) fill(255, 210, 0);
17     else fill(180);
18     float angle = TWO_PI * tickIndex / totalTickCount;
19     float xPosition = cos(angle) * circleRadius;
20     float yPosition = sin(angle) * circleRadius;
21     ellipse(xPosition, yPosition, jewelDiameter, jewelDiameter);
22   }
```

What to notice

- The jewel at the current second is missing; everything else is drawn.

Labelled break

Useful when you need to exit an outer loop from inside an inner loop.

```
1  outer: for (int rowIndex = 0; rowIndex < rowCount; rowIndex++) {
2    for (int columnIndex = 0; columnIndex < columnCount;
       columnIndex++) {
3      if (shouldStop) {
4        break outer; // jumps out of both loops
5      }
6      ...
7    }
8  }
```

Loop variable scope — the one new scope today

Key rule: A variable declared in the `for` header exists *only inside the loop*.

```
1  // Example: loopCounter is local to the loop
2  for (int loopCounter = 0; loopCounter < 3; loopCounter++) {
3    println("inside: " + loopCounter);
4  }
5  // println(loopCounter); // <-- Uncommenting this gives:
     cannot find symbol 'loopCounter'
```

If you need it later, declare first:

```
1  int loopCounter;
2  for (loopCounter = 0; loopCounter < 3; loopCounter++) { /* ...
     */ }
3  println("after the loop, loopCounter = " + loopCounter); // OK
```

Common pitfalls (so you don't fall in)

- Checking `% 3` and `% 5` before `% 15` ⇒ wrong labels for 15, 30, ...
- Mutating loop counters inside the body ⇒ unpredictable loops. Example:

```
for (int index = 0; index < 5; index++)  index++;
```

- Mixing up row/column and using `columnCount` vs `rowCount` in formulas.

Computer science joke

There are only two hard problems in computer science: cache invalidation, naming things, and off-by-one errors.

Computer science joke

There are only two hard problems in computer science: cache invalidation, naming things, and off-by-one errors.

- Off-by-one errors: when you miscount one more than you expected
- We often start counting at 0

Bug 1: how many lines does this draw?

```
1  void draw() {
2    for (int x = 0; x <= 3; x++) { line(x*10, 0, x*10, height); }
3  }
```

Bug 2: Two problems

```
1  void setup() {
2    fill(255,0,0);
3  }
4
5  void draw() {
6   if (mouseX > width/2)
7     fill(0);
8     rect(0,0,50,50);
9  }
```

Programs that you could write now

- Tweak Simple FizzBuzz to use different words or colours for new rules.
- Fade the cells in FizzBuzz Grid so old numbers slowly disappear.
- Mark minutes on the clockface with squares or bigger jewels.
- Use a labelled `break` to exit nested loops when the mouse is pressed.