Review

- Variables
- Variable types
- Integer division
- Drawing Images
- Conditionals: if - else if - else
- Motion simulation (today)

Simulated Motion (balldrop)

p = position
v = velocity
a = acceleration

- Constant acceleration (a)
  - assuming small time intervals (t=1)

$p_{i+1} = p_i + v_i$

$v_{i+1} = v_i + a$

Program Structure

- If code is to be executed only once
  - Put it in **`setup()`** not in **`draw()`**
  - Leave it in **`draw()`**, but call **`noLoop()`** in **`setup()`**
- Remove **`draw()`**?
  - All keyboard and mouse callbacks need the event loop
- Variable scope
  - variables are available/accessible only in the function where it is declared
- Global variables
  - declared outside of any function
  - available to all

```
int x, y;

void setup() {
}

void draw() {
}
```

Principals of Animation

- Think of each iteration of the **`draw()`** loop as drawing a new key frame
- In each frame, you animate an object by
  - Erasing the old canvas (**`background()`** call)
  - Drawing the object again with a new position
  - Updates if any
- Typical call sequence
  - new background
  - position = position + velocity
  - draw object
  - velocity = velocity + acceleration

Saving a Screen Shot

- **`save(filename);`**
- What if your sketch has animation or interaction?
  - you don't have a clear place in your code to put the **`save`** command
- Program the **`keyPressed`** interaction instead

```
void keyPressed() {
  if (key == 's') {
    save("screenshot.jpg");
  }
}
```

  - Screen shot will be now be saved whenever 's' is pressed

Expressions

- Collections of data values and variables related by operators and function calls, and grouped by parentheses.
- Expressions are automatically evaluated and replaced by the final evaluated value.
- Expressions can be assigned to variables using "="
  - Expression is always on right
  - Variable name is always on left

variable_name **=** *expression;*

Some Built-in Mathematical Functions

```
sin(x), cos(x), tan(x), asin(x), …
abs(x), exp(x), pow(x, y), log(x), sqrt(x), …
max(x1, x2), min(x1, x2), floor(x), ceil(x), …

dist(x1, y1, x2, y2)    -> distance between two points
norm(value, low, high)  -> normalizes a value to [0-1]
```

… and many more, all of which can be included in an expression.

Operators

+, –, *, / and …

```
i++;      equivalent to     i = i + 1;
i += 2;   equivalent to     i = i + 2;
i--;      equivalent to     i = i – 1;
i -= 3;   equivalent to     i = i - 3;
i *= 2;   equivalent to     i = i * 2;
i /= 4;   equivalent to     i = i / 4;

i % 3;    the remainder after i is divided by 3 (modulo)
```

Evaluating Expressions

```
1 + 2
pow(sin(x),2) + pow(cos(x),2) == 1.0
max(1, 2, 3) >= 2
floor(2.9) == ceil(1.8)
```

Iteration

Repetition of a program block

- Iterate when a block of code is to repeat multiple times.

Options

- The while-loop
- The for-loop

Iteration: while-loop

```
while (boolean_expression) {
  statements;
  // continue;
  // break;
}
```

- Statements are repeatedly executed as long as the boolean expression remains true;
- To break out of a while loop, call **`break;`**
  - usually in conjunction with an **`if`** statement
- To skip execution of statements and start another iteration, call **`continue;`**

Iteration: while-loop

```
while (boolean_expression) {
  statements;
  // continue;
  // break;
}
```

As a rule: never use continue or break. There is almost always a better way.

- Statements are repeatedly executed as long as the boolean expression remains true;
- To break out of a while loop, call **`break;`**
  - usually in conjunction with an **`if`** statement
- To skip execution of statements and start another iteration, call **`continue;`**

```
void setup() {
  size(500, 500);

  float diameter = 500.0;
  while (diameter > 1.0) {
   ellipse(250, 250, diameter, diameter);
   diameter = diameter * 0.9;
  }
}
```

What does this do?

```
void setup() {
  size(500, 500);

  float diameter = 500.0;
  while (true) {
   ellipse( 250, 250, diameter, diameter);
   diameter = diameter * 0.9;
   if (diameter <= 1.0) break;
  }
}
```

The Event Loop

- Although the **`draw()`** loop is certainly a loop, you should think of it as painting a particular still frame for a particular time step
- If you want anything repeated in this single frame, you will need a loop

Iteration: for-loop

```
for (initialization; continuation_test; increment) {
  statements;
  // continue;
  // break;
}
```

- Initialization, continuation test and increment commands are part of statements
- Known as a definite loop because you usually know exactly how many times it will iterate

Iteration: for-loop

```
for (initialization; continuation_test; increment) {
  statements;
  // continue;
  // break;
}
```

As a rule: never use continue or break. There is almost always a better way.

- Initialization, continuation test and increment commands are part of statements
- Known as a definite loop because you usually know exactly how many times it will iterate

```
for (int i = 0; i < 10; i++){
    print(i);
}
println();
```

```
for (int i = 0; i < 10; i++) {
    if (i % 2 == 1) continue;
    print(i);
}
println();
```

```
void setup() {
  size(500, 500);

  float diameter = 500;
  while (diameter > 1) {
   ellipse(250, 250, diameter, diameter);
   diameter = diameter - 10;
  }
}
```

```
void setup() {
  size(500, 500);

  for (float diameter = 500; diameter > 1; diameter -= 10) {
    ellipse(250, 250, diameter, diameter);
  }
}
```