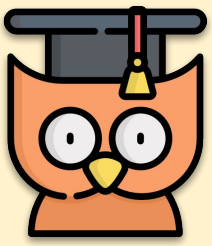
[the academy_of_code]

Grade 6

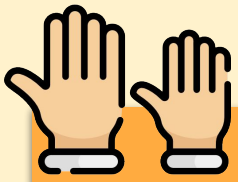
Unit 1

Lesson 1 - Let's get back into it!



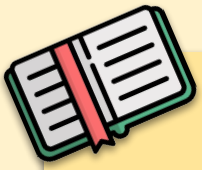
Learning Outcomes:

- Revise user-defined functions
- Go over collision detection
- Simple game mechanics



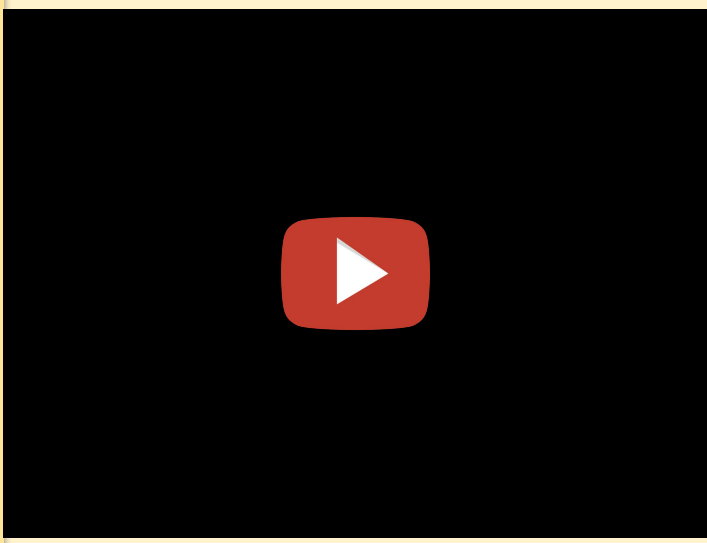
REMEMBER:

Put up your hand. We love to help!



Welcome Back!

It's probably been a while since you last coded in Processing, so in this lesson we're gonna use some of the concepts we learned last term to build a fun game! In this game, our player (white) must collect the green fruit. A red enemy also chases after our fruit, costing us a life every time it gets to the fruit before we do.



Here's what you will need to make the game

1. Make a **function** that draws a player and get it moving around the screen using `void keyPressed()` (either with 'WASD' or arrow keys).
2. Make a **function** that loops the player around the screen, i.e., when it goes off the right side, it appears back on the left side and vice versa.
3. Add in the green fruit at a random position and make a **function** to change its position if the player touches it.
4. Add an enemy which constantly moves towards the fruit and also causes the fruit to change position if it touches it.
5. Add in a score and lives system and display them on the screen. A life should be lost every time the enemy eats the fruit. Add a game over screen when you have no lives left.

Check the next pages for some help if you're not sure how to do a step.

Hints (Spoilers Ahead!)

1. Your player function should look something like this:

```
void drawPlayer() {
  //Write your code to
  //draw the Player here
}
```

Here's the code that makes the player go up:

```
void keyPressed() {
  if (key == 'w') {
    playerX = playerX - 5;
  }
}
```

Add 3 more if statements to make the player move in every other direction.

2. This is the code that loops the player off the right and left sides of the screen:

```
if (playerX-playerSize/2>width) {
  playerX = 0-playerSize/2;
}
if (playerX+playerSize/2<0) {
  playerX = width+playerSize/2;
}
```

Expert Tip
0 represents the left hand side of the screen. If we took this out the code would still run the same.

Add 2 more if statements to make the player loop off the top and bottom of the screen too.

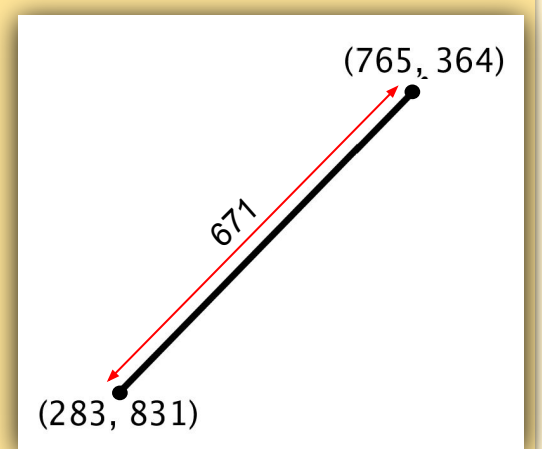
3. Here's the **pseudo-code** for the collision detection between Player and Fruit:

```
if (Distance between Player and Fruit is less than the sum of their two radii) {
    // change position of Fruit here
}
```

Note: You'll need to use the `dist()` function, which gives you the distance between two points. This is how it works:

`dist(x1, y1, x2, y2);` where (x1, y1) are the coordinates of the first point and (x2, y2) are the coordinates of the second point.

Thus, `dist(283, 831, 765, 364);` will return the number **671**



More Hints (More Spoilers Ahead!!)

4 There are a few different ways to code the movement for the enemy, depending on how you want the enemy in your game to move. This code makes the enemy move directly towards the fruit. Don't worry if you don't understand it, there's some complicated maths involved. You can also ignore this code entirely and write your own code.

```
void moveEnemy() {
  float h = dist(enemyX, enemyY, fruitX, fruitY);
  float dx = (fruitX - enemyX)/h;
  float dy = (fruitY - enemyY)/h;
  enemyX+=dx*enemySpeed;
  enemyY+=dy*enemySpeed;
}
```

Expert Tip

`x += speed;` is a shorter way of writing `x = x + speed;`

5 This is the code you'll need to add a game over screen:

```
void draw() {
  if (lives != 0) {
    //
    //All your game code goes here
    //
  } else {
    //
    //Your game over screen goes here
    //
  }
}
```

Expert Tip

!= means **not equal**. So in this code, as long as the number of lives is **not equal** to **0**, the game will run normally. **Otherwise**, the game is over.

Expert Tip

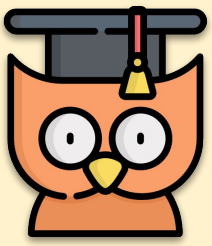
The last step can also be done using **booleans**, a type of variable which we can use to turn things on and off like a switch. You could have a boolean, for example, “**gameOver**”, which becomes **true** when you have 0 lives remaining. Then if “**gameOver**” is **true**, you’ll stop the game and draw your game over screen. **Otherwise**, the game will keep running.

Extra Challenges

If you’re finished the basic game, give these extra challenges a go:

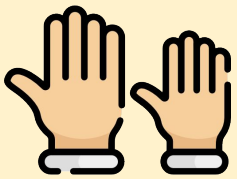
- Change your code to use a **boolean** instead of the given code for step 5
- Add a second fruit.
- Add a second enemy that follows the other fruit.
- Make the enemies move faster as your score increases.
- Add special kinds of fruit that give power-ups such as:
    - Extra life
    - Increased speed
    - Reduced enemy speed
    - Anything else you come up with!
- Improve the game’s looks by adding sprites. It can be any theme you want!

Lesson 2 - Introduction to Classes

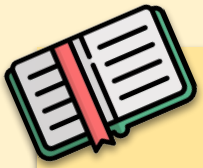


Learning Outcomes:

- Basic understanding of what classes and objects are
- How to use classes and why we should use them
- Converting last week's work to use classes



REMEMBER: If you have any questions, stay in your seat and put up your hand. We love to help!

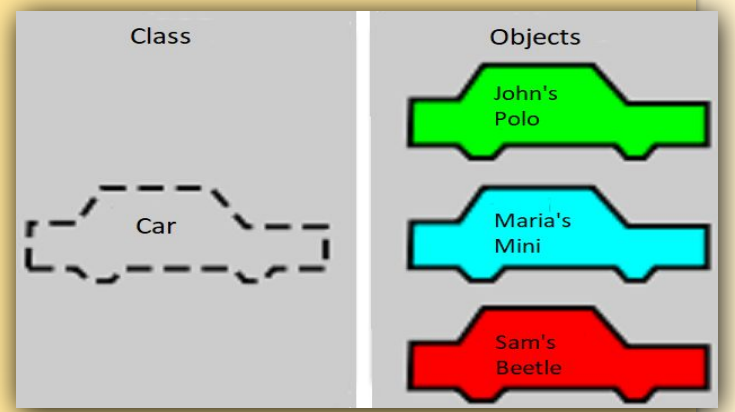


What is a class?

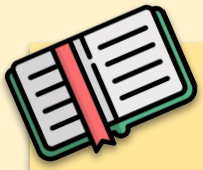
In the real world, you'll often find many individual objects of the same type. There may be millions of cars in existence, but they come in different makes, models, colours, engine sizes, etc. Each car was built from the same basic set of blueprints, however, each individual car will have a different set of **attributes**.

The most obvious example of an **attribute** you might notice is the **colour attribute**. This simply tells us the colour of the car and you can easily see how different cars can have different colours, but still be almost the same.

All of these cars are created from a basic set of blueprints. The blueprints tell us that every car needs four wheels, an engine, etc. This blueprint is called a **class** and all of our cars are created using this **class**. When we create an object (in this example a car) using a class, we say that the **object** is an **instance** of the **class**.



E.g. John's Polo is the **object** and the **object** is an **instance** of the car **class**. The **object** also has a colour **attribute** that is green.



Why should I use a Class?

This is a very good question, especially due to the fact that initially, classes can be a difficult topic to get your head around but it will all be worth it in the end, I promise! The short answer is that the use of classes will become essential as we start to work on more complex programs.

To understand why classes are so useful, let's say we want to produce some sort of animation that involves drawing hundreds or even thousands of cars, all with different attributes. Wouldn't it be easier to have a class that contains all of the attributes and methods (a fancy word for functions in a class) we need to draw a car, rather than a function to draw each car and then having to keep track of all the different variables for each car?

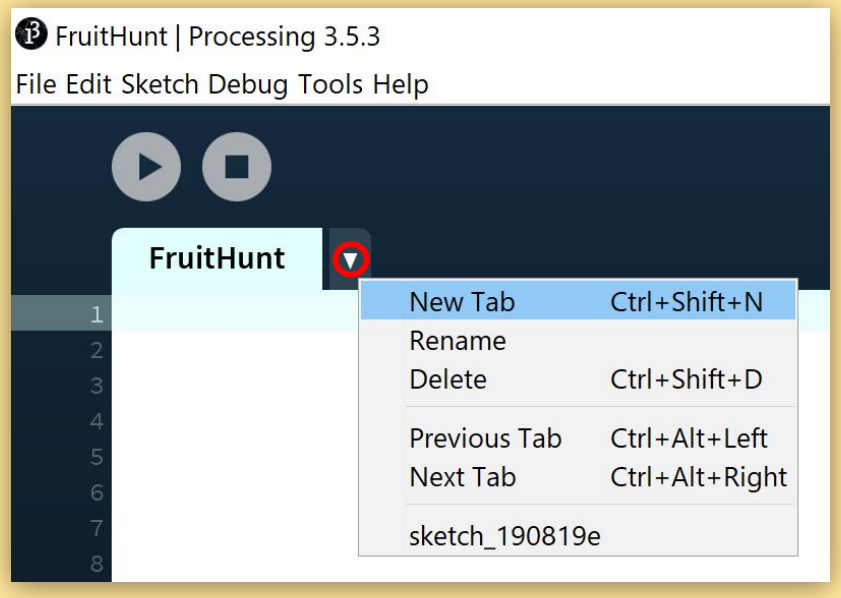
(Hint: The answer is yes!)

How do I use a Class?

To answer this question we're gonna re-make our game from last week using classes. Don't worry, you'll be able to copy and paste a lot of your code!

1. Make a new processing file and save it as **FruitHunt** (or whatever you like)

2. Create a new tab by clicking on the down arrow beside your file name and call the new tab **Player**:



How do I use a Class? (Continued)

3 This is the code in the **Player** tab. Here we have the definition of our **Player class**. We've set it up with a single **method**, `void drawPlayer()` and some **variables**. The Player class also contains a **constructor** that is used to create Player **objects** with different **attributes**, taken in as values. The class attributes are then set equal to these values. We access the class attributes using the keyword `this.` in front of the variable name.

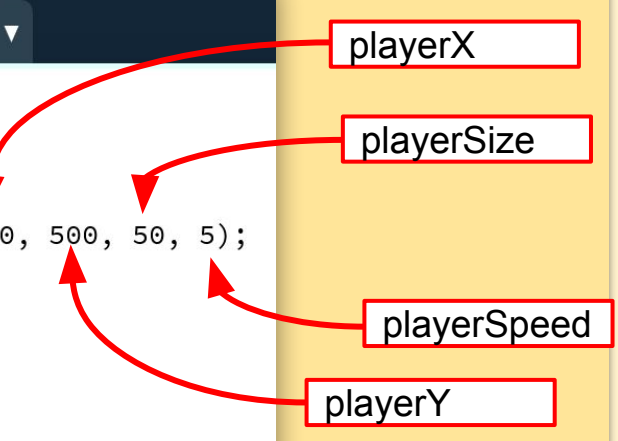
FruitHunt | Player

```
1  class Player {
2    float playerX;
3    float playerY;
4    float playerSize;
5    float playerSpeed;
6
7    Player(float playerX, float playerY, float playerSize, float playerSpeed) {
8      this.playerX = playerX;
9      this.playerY = playerY;
10     this.playerSize = playerSize;
11     this.playerSpeed = playerSpeed;
12   }
13
14   void drawPlayer() {
15     fill(255);
16     ellipse(playerX, playerY, playerSize, playerSize);
17   }
18 }
19
```

4 This is the code in our "main" file, which creates an **instance** of our Player class. Our player **object** must then be **initialized** by passing the parameters to the **constructor** in `void setup()` and then calling the `drawPlayer()` **method** in `void draw()`.

FruitHunt | Player

```
1  Player player;
2
3  void setup() {
4    size(1000, 1000);
5    player = new Player(500, 500, 50, 5);
6  }
7
8  void draw() {
9    player.drawPlayer();
10 }
11
```



Time to write your own code now!

1. Complete the Player class by copying your functions from your old code.

   Note: When using an object's variables **outside** the class, you must first use the object's name, just like when calling a method. For example, if you want to use `playerX` in your main file, you'll have to write `player.playerX`

2. Create a new tab for a **Fruit class** with all the fruit-related functions from your original code and create an **instance** of it in the 'main' file. Here's a template you can use to guide you:

FruitHunt | Enemy | Fruit | Player

```
class Fruit {
  float fruitX;
  float fruitY;
  float fruitSize;

  public Fruit(float x, float y, float s) {
    this.fruitX = x;
    this.fruitY = y;
    this.fruitSize = s;
  }

  void display() {
    //
    // Code that draws the fruit on the screen
    //
  }

  void move() {
    //
    // Code that moves the fruit to a random position
    //
  }
}
```

3. Create another tab with an **Enemy class** that contains all the enemy-related functions from your original code and create an **instance** of it in the 'main' file.

FruitHunt | Enemy | Fruit | Player

```
class Enemy {
  // declare your variables here

  public Enemy(float xPos, float yPos, float speed, float size) {
    // initialise your variables here
  }

  //
  // All your functions should go here
  //
}

```

Fruit Hunt with Classes (Continued)

4. Add in the remaining code to the main file (score, lives, game over screen, etc.)

5. Create a second **instance** of the Fruit class by declaring it and then initialising it in the main file. (You do **NOT** have to make a new class or any new variables, this is why classes are so useful!)

6. Create a second **instance** of the Enemy Class in a similar way so that the new enemy follows the new fruit.

   Note: You'll need to pass a **boolean** when **initialising** the enemy so that it knows which fruit to follow. You'll also have to edit the Enemy **constructor** and its movement **method**.

Extra Challenges

- Make the enemies move faster as your score increases.
- Add special kinds of fruit that give power-ups such as:
  - Extra life
  - Increased speed
  - Reduced enemy speed
  - Anything else you come up with!
- Improve the game's looks by adding sprites. It can be any theme you want!