



# Object Attack

## Outstanding Objects!

Dot is so excited to see you back at Decomosphere! Today, you'll learn more about objects and how their properties can be kept out-of-sight from others. Let's jump right in!

### Who let the dogs out?

In object-oriented programming, **encapsulation** allows for the variables and functions of an object to be encapsulated, or kept private from the interference of other objects. Each object can have private and public properties.

### What's the difference between private and public?

While private properties (variables and functions) cannot be changed or used by other objects, public properties can. Let's take a look at an example with Dot's pet dog, Rocky!



**Rocky the Labrador**

#### Variables (attributes Rocky has)

**private** breed

**Why private?**  
We can't control  
Rocky's breed and  
age.

**private** age

**public** owner

**Why public?**  
We can control  
who owns Rocky

#### Functions (what Rocky does)

**private** bark

**private** sleep

**public** feed

**public** play

**Why private?**  
We can't control  
when Rocky  
barks or sleeps

**Why public?**  
We can control  
when to feed or  
play with Rocky

## Why do we use private properties?

In programming, it's important to make properties like breed, mood, hunger, and energy level private in order to **protect other objects from changing them**.

## How can we change private variables?

We can give our object certain **functions to change these private variables!**

For example, let's take a look at our public function **feed**:

**public def feed:**

Add 1 to hunger  
Add 1 to mood



Hunger-meter +1

Mood-meter +1



Since our feed function is public (we can decide when to feed Rocky), we can indirectly change our private variables (hunger and mood).

However, some private variables can never be changed.

For example, we cannot change Rocky's breed as a Labrador.

## Grid Attack!

In this action-packed game, you will be creating your own **player object** with its **private/public variables and functions** to attack an opponent!

### Materials

- 2 tokens for you and your friend (This will represent where you are on the board!)  
- 1 die 

### How to Play

- Define your player object's variables by creating a name, age, and health value. Each player starts the game with a health level of 5 (out of 5). Next to each variable and function, circle whether it is private or public! (*Think: Can these properties be directly changed or used by other objects?*)
- Take turns rolling the die with your friend. Use the **walk** function to move your token the number of steps you rolled in any direction(up, down, left, right). Keep track of your activity in your **activity list**.  
*(Ex. If you rolled a 6, you can choose to walk 3 steps left and 3 steps up).*

If you are on , use the **eat** function to add 1 point to your own health.



If you are at the **same location** as your opponent, use the **attack** function to subtract 2 points from your opponent's health.



- Winning the game: You win the game when your opponent's health status reaches 0.

## My Player Object

<p><b>Variables</b></p> <p><b>private/public Name</b> <small>(circle one)</small> _____</p> <p><b>private/public Age</b> <small>(circle one)</small> _____</p> <p><b>private/public Health</b> <small>(circle one)</small> 5 _____</p>	<p><b>Functions</b></p> <p> <b>private/public def Walk:</b> <small>(circle one)</small> Move the number of steps rolled</p> <p> <b>-2 private/public def Attack:</b> <small>(circle one)</small> Subtract 2 points from opponent's health</p> <p> <b>+1 private/public def Eat:</b> <small>(circle one)</small> Add 1 point to your own health</p>
--	---

## My Activity List

Function	Health	Function	Health
Ex. Walk left 3 steps	5	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____

