Teacher Guide: Introduction to Computer Science

Last Updated: February 3, 2016

Table of Contents

Summary: Introduction to Computer Science

With the right environment, learning the basics of formal syntax and typing code can be fun and intuitive for students as early as 3rd grade. Instead of block-based visual programming languages that hinder a student's proper understanding of code, CodeCombat introduces real coding from the very first level. By strengthening their typing, syntax and debugging skills, we empower students to feel capable of building real programs successfully.

In Course 1, students will learn the basic syntax of Python or Javascript, along with arguments, strings, variables and while loops.

CodeCombat Courses use object-oriented programming. This guide covers both Python and Javascript solutions, with Python code depicted in `red` and Javascript code depicted in `blue`.

Concepts Covered in this Course

Basic Syntax

Syntax is the basic spelling and grammar of a language, and must be carefully paid attention to in order for code to properly execute. For example, while Python and Javascript are used to do similar things in Course 1, the syntax for them is noticeably different, because they are different programming languages.

Arguments

An argument (also referred to as a parameter) is extra information passed into a method in order to modify what the method does. In both Python and Javascript, arguments are represented by code that is inside the parentheses after a method. In Course 1, arguments must be used to define enemies before the hero can attack them, and can also be used to move multiple times without writing new lines of code.

Strings

A string is type of programming data that represents text. In both Python and Javascript, strings are represented by text inside quotes. In Course 1, strings are used to identify objects for the hero to attack.

Variables

A variable is a symbol that represents data, and the value of the variable can change as you store new data in it. In Course 1, variables are used to first define an enemy, and then passed along as an argument to the attack method so that the hero can attack the right enemy.

While Loops

A while loop is used to repeat actions without the player needing to write the same lines of code over and over. In Python, the code that is looped must be indented underneath the while true statement. In Javascript, the code that is looped must be enclosed by curly brackets {}. In Course 1, while loops repeat forever, and are used to navigate mazes made up of identical paths, as well as attack objects that take a lot of hits to defeat (strong Doors, for example).

Level Overview & Solutions

1. Dungeons of Kithgard

Let's get started! To escape the dungeon, your hero has to move. You can tell them where to move by writing *code*.

Type your code into the editor to give your hero instructions. Your hero will read and execute these instructions for themself, so refer to the hero with:
Python: `self`
JavaScript: `this`

Now that you know how to refer to your hero, you can instruct them to move with `moveDown` and `moveRight` commands:

Python:
```
self.moveDown()
self.moveRight()
```
JavaScript:
```
this.moveDown();
this.moveRight();
```

To succeed at this level: move **right**, **down**, and **right** again to grab the gem!

You only need *three lines of code* to beat this level.

The code you write here is very similar to the code you might write to tell a computer how to do all kinds of things, from playing music to displaying a web page. You're taking your first steps towards being a coder!

Dungeons of Kithgard Solution

Python
```
# Move to the gem.
# Don't touch the walls!
# Type your code below.

self.moveRight()
self.moveDown()
self.moveRight()
```

Javascript

```
// Move to the gem.
// Don't touch the walls!
// Type your code below.

this.moveRight();
this.moveDown();
this.moveRight();
```

2. Gems in the Deep

Can you remember the lessons from the last level? This will be the same, but you will need to move a lot more. Remember,

Python: `self`
JavaScript: `this`

refers to you, the hero.

When you move, you only move as far as the next movement square (look for the small tiles on the ground), so you might have to `moveUp` twice in a row to get to the top of this level from the bottom.

Or you can pass a number as an **argument** to the movement command, to instruct your hero to move more than one space at a time.

For example, you can move up twice by typing:

Python: `self.moveUp(2)`
JavaScript: `this.moveUp(2);`

Gems in the Deep Solution

Python

```
# Grab all the gems using your movement commands.
self.moveRight()
self.moveDown()
self.moveUp()
self.moveUp()
self.moveRight()
```

Javascript

```
// Grab all the gems using your movement commands.
this.moveRight();
this.moveDown();
this.moveUp();
this.moveUp();
this.moveRight();
```

3. Shadow Guard

You don't have a weapon yet, so you can't fight the ogre munchkin who guards the path.
Instead, try moving up, behind the statue, so he doesn't see you. Then you can get the gem undetected.

Shadow Guard Solution

Python

```
# Stay out of sight of the ogre. Grab the gems.
self.moveRight()
self.moveUp()
self.moveRight()
self.moveDown()
self.moveRight()
```

Javascript

```
// Stay out of sight of the ogre. Grab the gems.
this.moveRight();
this.moveUp();
this.moveRight();
this.moveDown();
this.moveRight();
```

4. Enemy Mine

The floor is littered with Fire Traps, but there's a safe path through to the gem.

When you call a method like `moveRight()` you can sometimes give extra information to the method to modify what it does. This extra information is referred to as "arguments" or "parameters".

You can pass an argument to the `moveRight()` method like this: `moveRight(3)`. This tells `moveRight()` to make your hero move 3 spaces to the right instead of 1.

Enemy Mine Solution

Python

```
# Use arguments with move statements to move farther.
self.moveRight(3)
self.moveUp()
self.moveRight()
self.moveDown(3)
self.moveRight(2)
```

Javascript

```
// Use arguments with move statements to move farther.
this.moveRight(3);
this.moveUp();
this.moveRight();
this.moveDown(3);
this.moveRight(2);
```

5. True Names

Keep in mind a few things to beat this level:

1. You need to attack each ogre munchkin **twice** to defeat it.
2. Spell the names properly, with capitalization! `"Brak"` and `"Treg"`.
3. Put the names in quotes to make them into strings. Strings are a type of programming data. They represent text.
4. After you kill `"Brak"` and `"Treg"`, then move right to get the gem.
5. It's no problem if you die; you can always keep trying.

True Names Solution

Python

```
# Defend against Brak and Treg!
# You must attack small ogres twice.

self.moveRight()
self.attack("Brak")
self.attack("Brak")
self.moveRight()
self.attack("Treg")
self.attack("Treg")
self.moveRight()
self.moveRight()
```

Javascript

```
// Defend against Brak and Treg!
// You must attack small ogres twice.

this.moveRight();
this.attack("Brak");
this.attack("Brak");
this.moveRight();
this.attack("Treg");
this.attack("Treg");
this.moveRight();
this.moveRight();
```

6. Fire Dancing

Code normally executes in the order it's written. **Loops** allow you to repeat a block of code multiple times without having to re-type it.

How To Use while-true Loops

First, we start a loop with the `while` keyword. This tells your program **WHILE** something is true, repeat the **body** of the loop.

For now we want our loops to continue forever, so we'll use a **while-true loop**. Because `true` is always true!

Don't worry about this true stuff too much for now. We'll explain it more later. Just remember that a **while-true loop** is a loop that never ends.

This is how you code a **while-true loop**:

Python:

```
while True:
    self.moveRight()
    self.moveLeft()
self.say("This line is not inside the loop!")
```

JavaScript:

```
while(true) {
    this.moveRight(); this.moveLeft();
}
this.say("This line is not inside the loop!");
// Tip: the indentation (spaces at the start of the lines) is optional, but makes your code easier to read!
```

Tip: You can put whatever you want inside a while-true loop! But for this level, we only need to repeat two commands: `moveRight()` and `moveLeft()`!

Fire Dancing Solution

Python

```
# Code normally executes in the order it's written.
# Loops repeat a block of code multiple times.
# Use tab or 4 spaces to indent the move lines under the loop.

while True:
    self.moveRight()
    # Add a moveLeft command to the loop here
    self.moveLeft()
```

Javascript

```
// Code normally executes in the order it's written.
// Loops repeat a block of code multiple times.
```

```
while(true) {
    this.moveRight();
    // Add a moveLeft command to the loop here
    this.moveLeft();
}
```

7. Loop Da Loop

You can survive this level using one **while-true loop** containing just **4 commands**!
Make sure the commands you add are **inside** the **while-true loop**. Double check your indentation!

Loop Da Loop Solution

Python

```
# The code in this loop will repeat forever.

while True:
    # Move right
    self.moveRight()
    # Move up
    self.moveUp()
    # Move left
    self.moveLeft()
    # Move down
    self.moveDown()
```

Javascript

```
// The code in this loop will repeat forever.

while(true) {
    // Move right
    this.moveRight();
    // Move up
    this.moveUp();
    // Move left
    this.moveLeft();
    // Move down
    this.moveDown();
}
```

8. Haunted Kithmaze

Loops let you repeat the same code over and over. You can do this level in just four commands with a **while-true loop**.
Tip: the hallway needs **two movements to the right**, and then **two movements up**. From there, you can just let the **while-true loop** repeat to do the rest.
Make sure that your movement commands are **inside the loop** so that they repeat!

Haunted Kithmaze Solution

Python

```
# Loops are a better way of doing repetitive things.

while True:
    # Add commands in here to repeat.
    self.moveRight()
    self.moveRight()
    self.moveUp()
    self.moveUp()
```

Javascript

```
// Loops are a better way of doing repetitive things.

while(true) {
    // Add commands in here to repeat.
    this.moveRight();
    this.moveRight();
    this.moveUp();
    this.moveUp();
}
```

9. The Second Kithmaze

Carefully count how many movements you need inside your **while-true loop** to solve the maze!
Remember, you should only use one **while-true loop** per level, and make sure all your code is inside the loop.
Hover over the **while-true loop** documentation in the lower right to see an example.

The Second Kithmaze Solution

Python

```
# Use a while-true loop to navigate the maze!

while True:
    this.moveRight()
    this.moveUp()
    this.moveRight()
    this.moveDown()
```

Javascript

```
// Use a while-true loop to navigate the maze!

while (true) {
    this.moveRight();
    this.moveUp();
    this.moveRight();
    this.moveDown();
}
```

10. Dread Door

You can combine **while-true loops** and attack to easily kill things that take more than one hit. Like this door.

Python:

```
while True:
    self.attack("Door")
```

JavaScript:

```
while(true) {
    this.attack("Door");
}
```

You can attack the door by its name, which is `"Door"`.
With looping and attacking, you can do this level in just two lines of code.

Dread Door Solution

Python

```
# Attack the door!
# It will take many hits, so use a while-true loop.

while True:
    self.attack("Door")
```

Javascript

```
// Attack the door!
// It will take many hits, so use a while-true loop.

while (true) {
    this.attack("Door");
}
```

11. Cupboards of Kithgard

The ogre guards might be too much for you to handle. Maybe you'll find something useful in the `"Cupboard"`?

First, move close to the `"Cupboard"` (stand on the red X). It looks locked, so you'll have to attack it repeatedly using a **while-true loop** to break it open.

Cupboards of Kithgard Solution

Python

```
# There may be something around to help you!

# First, move to the Cupboard.
self.moveUp()
self.moveRight(2)
self.moveDown(2)

# Then, attack the "Cupboard" inside a while-true loop.
while True:
    self.attack("Cupboard")
```

Javascript

```
// There may be something around to help you!

// First, move to the Cupboard.
this.moveUp();
this.moveRight(2);
this.moveDown(2);

// Then, attack the "Cupboard" inside a while-true loop.
while (true) {
    this.attack("Cupboard");
}
```

12. Breakout

You'll need that soldier to protect you, so first attack the `"Weak Door"` to free her.
Then use a **while-true loop** to attack the `"Door"` while your new friend holds off the munchkins.

Breakout Solution

Python

```
# Free your ally, then clear a path to escape!

self.moveRight()
self.attack("Weak Door")
self.moveRight()
self.moveDown()
while True:
    self.attack("Door")
```

Javascript

```
// Free your ally, then clear a path to escape!
```

```
this.moveRight();
this.attack("Weak Door");
this.moveRight();
this.moveDown();
while (true) {
    this.attack("Door");
}
```

13. Known Enemy

Up until now, you have been doing three things:

1. Calling **methods** (commands like `moveRight`)
2. Passing **strings** (quoted pieces of text like `"Treg"`) as arguments to the methods
3. Using **while-true loops** to repeat your methods over and over.

Now you are learning how to use **variables**: symbols that represent data. The variable's value can **vary** as you store new data in it, which is why it's called a variable.

It's a pain to type the names of ogres multiple times, so in this level you use three variables to store the ogre names. Then when you go to attack, you can use the variable (`enemy1`) to represent the string that is stored in it (`"Kratt"`).

Declare variables like so:
Python: `enemy1 = "Kratt"`
JavaScript: `var enemy1 = "Kratt";`

When you use quotes: `"Kratt"`, you are making a **string**.

When you don't use quotes: `enemy1`, you are referencing the `enemy1` **variable**.

Known Enemy Solution

Python

```
# You can use a variable like a nametag.

enemy1 = "Kratt"
enemy2 = "Gert"
enemy3 = "Ursa"

self.attack(enemy1)
self.attack(enemy1)

self.attack(enemy2)
```

```
self.attack(enemy2)

self.attack(enemy3)
self.attack(enemy3)
```

Javascript

```
// You can use a variable like a nametag.

var enemy1 = "Kratt";
var enemy2 = "Gert";
var enemy3 = "Ursa";

this.attack(enemy1);
this.attack(enemy1);

this.attack(enemy2);
this.attack(enemy2);

this.attack(enemy3);
this.attack(enemy3);
```

14. Master of Names

Remember from the last level, **variables** are symbols that represent data. The variable's value can **vary** as you store new data in it, which is why it's called a variable.

Now instead of using the names of the enemies, you can use the `findNearestEnemy()` method to store references to the ogres in variables. You don't need to use their names.

When you call the `findNearestEnemy()` method, you **must store the result in a variable**, like `enemy3` (you can name it whatever you want). The variable will remember what the nearest enemy **was** when you called the `findNearestEnemy()` method, so make sure to call it when you see a nearby enemy.

Remember: when you use quotes, like `"Kratt"`, you are making a **string**. When you don't use quotes, like `enemy1`, you are referencing the `enemy1` **variable**.

Ogre munchkins still take two hits to defeat.

Master of Names Solution

Python

```
# Your hero doesn't know these enemy's names!
# The glasses give you the findNearestEnemy ability.

enemy1 = self.findNearestEnemy()
self.attack(enemy1)
self.attack(enemy1)

enemy2 = self.findNearestEnemy()
self.attack(enemy2)
self.attack(enemy2)

enemy3 = self.findNearestEnemy()
self.attack(enemy3)
self.attack(enemy3)
```

Javascript

```
// Your hero doesn't know these enemy's names!
// The glasses give you the findNearestEnemy ability.

var enemy1 = this.findNearestEnemy();
this.attack(enemy1);
this.attack(enemy1);

var enemy2 = this.findNearestEnemy();
this.attack(enemy2);
this.attack(enemy2);

var enemy3 = this.findNearestEnemy();
this.attack(enemy3);
this.attack(enemy3);
```

15. A Mayhem of Munchkins

In this level, you use a **while-true loop** to do two things:

First, use `findNearestEnemy()` to find an ogre. Remember to store the result in an `enemy` variable. Hover over the `findNearestEnemy()` method to see an example.

Then, `attack` using the `enemy` variable.

A Mayhem of Munchkins Solution

Python

```
# Inside a while-true loop, use findNearestEnemy() and attack!

while True:
    enemy = self.findNearestEnemy()
    self.attack(enemy)
    self.attack(enemy)
```

Javascript

```
// Inside a while-true loop, use findNearestEnemy() and attack!

while (true) {
    var enemy = this.findNearestEnemy();
    this.attack(enemy);
    this.attack(enemy);
}
```

16. The Gauntlet

With your powers of looping and variables, it should be no sweat to take down all these munchkins. In fact, with the **while-true loop**, you can do it in just five lines of code:

1. one to start the while-true loop,
2. one to move to where you can see an enemy,
3. one to store the nearest enemy into a variable,
4. and two to attack,
5. because munchkins take two hits with your current sword

The Gauntlet Solution

Python

```
# Use what you've learned to defeat the ogres.
# Remember: it takes two attacks to defeat an ogre munchkin!

while True:
    self.moveRight()
    enemy = self.findNearestEnemy()
    self.attack(enemy)
    self.attack(enemy)
```

Javascript

```
// Use what you've learned to defeat the ogres.
// Remember: it takes two attacks to defeat an ogre munchkin!

while (true) {
    this.moveRight();
    var enemy = this.findNearestEnemy();
    this.attack(enemy);
    this.attack(enemy);
}
```

17. The Final Kithmaze

This level combines **while-true loops** and **variables** to both solve a maze and attack enemies.

Now you see why you need variables, because you're actually going to vary the data in the variable. Inside your while-true loop, if you define an `enemy` variable, it will refer to each of the three ogre munchkins in the level as the loop repeats. Cool, huh?

Pay attention to where your while-true loop should repeat so that you don't move further than you need to.

Make sure that you call `findNearestEnemy()` when you can actually see the ogre munchkin with clear line of sight.

The Final Kithmaze Solution

Python

```
# Use a while-true loop to both move and attack.

while True:
    self.moveRight()
    self.moveUp()
    enemy = self.findNearestEnemy()
    self.attack(enemy)
    self.attack(enemy)
    self.moveRight()
    self.moveDown(2)
    self.moveUp()
```

Javascript

```
// Use a while-true loop to both move and attack.

while (true) {
    this.moveRight();
    this.moveUp();
    var enemy = this.findNearestEnemy();
    this.attack(enemy);
    this.attack(enemy);
    this.moveRight();
    this.moveDown(2);
    this.moveUp();
}
```

18. Kithgard Gates

When you use a builder's hammer, instead of the attack method, you get the `buildXY` method. `buildXY` takes three arguments, instead of one: `buildType`, `x`, and `y`. So you can decide what to build and where to build it.

- `buildType`: either the string "fence", to build fences, or the string "fire-trap", to build fire traps.
- `x`: the horizontal position at which to build. You can **hover over the map** to find coordinates.
- `y`: the vertical position at which to build. `x` and `y` are both in meters.

`buildXY("fence", x, y)` allows you to build a fence at a certain spot, like this:

Python: `self.buildXY("fence", 40, 20)`
JavaScript: `this.buildXY("fence", 40, 20);`

This level is **much easier to beat with `"fence"`** than with `"fire-trap"`. It's almost impossible to use fire traps to kill the ogres. If you want to try it, fine, but it took us fifteen minutes to figure it out, and we built the level.

You only need to build three fences to stop the ogres and escape the dungeon to the right.

Kithgard Gates Solution

Python
```
# Build 3 fences to keep the ogres at bay!

self.moveDown()
self.buildXY("fence", 36, 34)
self.buildXY("fence", 36, 30)
self.buildXY("fence", 36, 26)
self.moveRight(3)
```

Javascript
```
// Build 3 fences to keep the ogres at bay!

this.moveDown();
this.buildXY("fence", 36, 34);
this.buildXY("fence", 36, 30);
this.buildXY("fence", 36, 26);
this.moveRight(3);
```

19. Wakka Maul

Battle your friends, coworkers and classmates in this all out brawl through the Kithgard dungeons!

Break out allies, summon more units, and evade the enemy's advances!

The doors are labelled `"a","b","c","d","e","f","g","h","i","j"`. Use these strings to attack the specific door you want!

The human side can summon `soldier` and `archer` while the ogre side can summon `scout` and `thrower`. All either side needs to do is to say the unit name, and have enough gems, to summon the units. To summon units you'll want to say their name:

****Python****:

```
# If on the human side:
self.say("soldier")
# To summon a soldier for 20 gold.

self.say("archer")
# To summon an archer for 25 gold.

#If on the ogre side:
self.say("scout")
# To summon a scout for 18 gold.

self.say("thrower")
# To summon 2 throwers for 9 gold each.
```

JavaScript:

```
// If on the human side:
this.say("soldier");
// To summon a soldier for 20 gold.

this.say("archer");
// To summon an archer for 25 gold.

// If on the ogre side:
this.say("scout");
// To summon a scout for 18 gold.

this.say("thrower");
// To summon 2 throwers for 9 gold each.
```

Common Problems in this Course

While loops in Python require indentation
When students are first learning about while loops, it's sometimes challenging for them to understand that in order for something to be looped, it needs to be nested inside a while loop by indenting the line with four spaces or a tab.

This is not as much of an issue in JavaScript because JavaScript uses brackets `{}` to signify when loops occur, but it's still good practice to indent because it makes code easier to read.

Your hero needs to be told what to attack (using parameters)
When students use an attack function, they need to specify what will be attacked by putting a parameter inside the parentheses that comes after the "attack" function.

In Python, it looks like this:
```
self.attack(enemy)
```

This tells the hero to attack the variable named `enemy`.

In earlier levels, before variables are introduced, students attack enemies by name using strings (pieces of text in quotes):
```
self.attack("Treg")
```

Sometimes it takes more than one attack to defeat an enemy
Ogre munchkins each take two attack actions to defeat.

Weak doors (ones that are labeled "Weak Door") take one attack action to defeat, so don't use a while loop to attack them.

A strong door (labeled "Door") takes a lot of hits, so students should use a while loop to attack the door indefinitely.

You need line-of-sight to use findNearestEnemy()
If you can't see an enemy around a corner, then findNearestEnemy() will not return the enemy, so when you then attack the enemy variable you created with it, the attack will fail. Make sure that you move to within line of sight of the enemy before trying to attack it on levels like The Gauntlet and The Final Kithmaze.

If you don't update your enemy variables, you might keep attacking the first, dead enemy
In order to attack a second enemy, you need to call findNearestEnemy() again *after the first one is dead* so that you store the new enemy when the second enemy is the nearest one. Otherwise, you might keep attacking the first enemy, because you called findNearestEnemy() twice when the nearest enemy was still the same, first enemy. So do this:
```
enemy1 = self.findNearestEnemy()
self.attack(enemy1)
self.attack(enemy1)
enemy2 = self.findNearestEnemy()
self.attack(enemy2)
```

```
self.attack(enemy2)
```

And not this:

```
enemy1 = self.findNearestEnemy()
enemy2 = self.findNearestEnemy()  # Bug: this is the same as enemy1
self.attack(enemy1)
self.attack(enemy1)
self.attack(enemy2)
self.attack(enemy2)
```

Some levels limit how many code statements you can use

This is to encourage players to learn how to use while loops instead of repeating their code by hand. Sometimes it can be hard for a player to know what counts as a statement, though. It is not the same as a line of code, but rather it's each unique piece of information in the code the compiler sees. For example, this program is five statements:

```
while True:  # 1 for the while loop
    enemy = self.findNearestEnemy()  # 1 for the findNearestEnemy()
    self.attack(enemy)  # 1 for the attack
    self.moveRight(3)  # 1 for the moveRight and 1 for the number 3
```

Sometimes students will use extra statements they don't need with code like this:

```
self.moveRight(1)  # The 1 counts as an extra statement
```

Or they will not store the variable and call findNearestEnemy() an extra time:

```
self.attack(findNearestEnemy())  # 2 statements
self.attack(findNearestEnemy())  # 2 more statements
```

A level won't be finished until it's done under the required number of statements, so help students who are using too many statements understand concepts like while loops and variables before moving forward.