Background and Objectives

Congratulations, you have been summoned by the higher beings and you're now Buffy, the vampire slayer.
In this exercise, you will build yourself a guide on how to slay a vampire, as told from a narrator's point of view.

This narrator should have the following behavior:

1. If you don't choose the right weapon, the narrator says "The vampire laughs at your ineffective weapon."
2. The only way to slay the vampire is to choose a `"stake"`.

This behavior should be implemented in the `vampire_weapon.rb` file. Once your `vampire_reaction` method is working, you can implement the logic in the `interface.rb` file to run the program. The interface should ask a user to choose a weapon to slay a vampire and display the vampire's reaction.

Once that works, try and implement a loop structure that will keep asking a user to choose a new weapon until the vampire is destroyed. You might need a `while` or `until` loop to do this. Scroll down to find a link with more information about these looping structures. We'll cover them in detail in tomorrow's lecture.

Let's make a comparison between the real world and the code world on this exercise.

Real world	Code world
Confront the vampire	Running `$ ruby lib/interface.rb` in the terminal
Choosing a weapon	Writing a string in the terminal and hitting Enter
Vampire reacts	Reading the vampire's reaction printed on the terminal with `puts`
Slaying the vampire	Typing `"stake"`, hitting Enter. The program should exit.

The objectives of this challenge are :

- Understand the flow of a program and learn how to "read" through your code, line by line
- Learn about conditional statements
- Learn about coding structures that modify your program flow: `if..elsif/else..end`,

`while/until..end`,.. They are control structures(we'll learn more about them tomorrow)
- Learn about boolean logic : AND (&&) / OR (||)

Specs

Vampire weapon

In the `lib/vampire_weapon.rb` file, you will find method definition of `vampire_reaction`. You can see that it takes one argument, `your_weapon` which is the weapon you use against the vampire. The method should return a `String`, the vampire's reaction will depend of which value is passed in `your_weapon`.

Enhanced Vampire Weapon

Now let's implement an enhanced version of the vampire's reaction. This time, you can choose a weapon and the material it's made of.
Here are some cases to consider:

- The vampire will disintegrate if you strike it with a `'stake'` made of `'wood'`.
- The vampire will explode if you hit it with any weapon made of silver
- Anything else will be ineffective against the vampire.

To implement these methods correctly, you will need to use multiple conditions. In Ruby, you can combine two conditions to create a new condition. If you need two conditions to be true at the same time to be true, you use the `&&` operator. Like so:

```
true && true   #=> true
true && false  #=> false
false && true  #=> false
false && false #=> false
```

Interactive Program

- Write the code for the interface that makes you confront the vampire.
- constraint: This program should "loop". The vampire should react and wait another round until you choose an effective weapon to destroy it. Use a `while..end` or `until..end` structure for that purpose.
- For the enhanced version, use the interface_enhanced.rb to implement your Programs. When the vampire explode, use the string 'The vampire has exploded!' and when he is disintegrated, use the string 'The vampire has been desintegrated!'.

Key learning points

- What's the usual flow of a program ?
- How do structures like `if..else..end` or `while..end` change this flow ?
- How do these structures work ?
- What's a conditional statement ? Which values can it take ? What's the difference between `=` and `==` ?
- Does a simple method call change the flow of your program ?