Background and Objectives

Congratulations, you're now a vampire slayer. In this exercise, you will build yourself a guide to 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. Lucky for you, the vampire is slow and you still have time to choose another weapon.
3. The **only** way to slay the vampire is to choose a "stake"

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
- 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. The first valid weapons are a "stake" made of "wood" or "silver". The first valid weapons are a "stake" made of "wood", the vampire will disintegrate and you win. If the weapon is in silver, the vampire will explode and you also win.

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 ?