Link to lesson.

A link to the lesson can be found here

What is a method?

- Methods are **verbs (actions)** of programming.
- Methods are a **sequence of instructions** grouped together to perform a certain task.
- This **grouped sequence of instructions** is called a **method definition** and is given a name called a **method name**.
    - EVERY method definition starts with the key word **Def**.
    - Followed by the **name of the method**.
    - Next, comes the **grouped sequence of instructions**.
    - The code finishes with the keyword **end**.

  Example 1 (method definition):

  ```
  #keyword and name of method
  def tie_my_shoes

  #grouped sequence of instructions
      puts "grab shoe laces"
      puts "twist and tie laces around"
      puts "end"

  #keyword
  end
  ```

- This will NOT execute anything as it is JUST a **method definition**.
- In order for the method definition to execute a value, we need to CALL the method by its name.

  ```
  tie_my_shoes
  ```

this will return:

```
    => grab shoe laces
    => twist and tie laces around
    => end
```

Method call (invoke)

- A key point to note is a method must be defined ONLY ONCE in the program.
- It can be called ANY NUMBER of times in the program.
- HOWEVER, a **method definition** DOES NOT mean it will be executed.

Why use methods?

- DRY - Do Not Repeat Yourself.
- It is VERY important in any programming language to keep the code DRY.
- Methods help facilitate the DRY approach.
- This is achieved by defining a group of instructions **ONLY ONCE** in a program.
- Then we can call the method any number of times.
- Therefore, we are NOT repeating ouselves as we are only defining the method ONCE, and then can call it with the method name **any number of times**.

Passing Parameters/Arguments to a method

- Methods can be **customised** by **passing parameters/arguments** to it.
- Parameters and arguments refer to the same thing.
- These arguments are **variables** and they have **different values passed** when called.
- Number of arguments in the definition MUST MATCH the arguments in the **method call**.
- Parameters/arguments are placed between parentheses (see example below).

```
#method definition with arguments/parameters in parentheses
 def cook (item, cooking_item)

 #grouped sequence of instructions with interpolation
     puts "fill a saucepan with water"
     puts "place a saucepan on the stove"
     puts "bring a saucepan to boil"
     puts "add #{item_name}
     puts "cook for #{cooking_time} minutes

 #keyword
 end
```

Method call **Example 1** with parameters:

```
    cook("rice", 10)
```

- NOTE the number of parameters/arguments in the method call MATCHES the number of parameters/arguments in the method definition.

This method call will return:

```
=> fill a saucepan with water
=> place saucepan on the stove
=> bring saucepan to boil
=> add rice
=> cook for 10 minutes
```

- In this manner, the SAME SET OF INSTRUCTIONS are customised by the **arguments or parameters** that we are **passing**.

Method call **Example 2** with parameters:

```
    cook("pasta", 15)
```

This method call will return:

```
    => fill a saucepan with water
    => place saucepan on the stove
    => bring saucepan to boil
    => add pasta
    => cook for 15 minutes
```

- This illustrates how you can CUSTOMISE methods by using parameters/arguments.

- To be CLEAR, the number of parameters/arguments in the method call MUST MATCH the number of parameters/arguments in the method definition.

- Otherwise an error will be returned.

Incorrect method call **Example 1** with parameters:

```
    cook("rice")
```

- This is because in our method definition we passed TWO parameters/arguments

  ```
  def cook (item, cooking_item)
  ```

- Therefore, we MUST call the same number of parameters/arguments - which is TWO in this example
  - The first parameter is:

    ```
    (item)
    ```

  - The second parameter is:

    ```
    (cooking_item)
    ```

- This a common mistake!

Default Parameters

- Default Parameters allows this common mistake to be avoided.
- While defining a method a **default value** can be assigned to the arguments/parameters.
- These named arguments are assigned as **key value pairs**. (Just like the way we define a **hash**).
- If NO ARGUMENT is passed, the default value is taken.
- IF an arguments IS PASSED, the default value is OVERWRITTEN with the passed argument.

  ```
  #method definition with named key value pairs
  def greeting (name: "Programmer", language: "Ruby")


  #grouped sequence of instructions with interpolation
      puts "hello #{name}! We heard you love the #{language} program.

  #keyword
  end
  ```

Call method with NO passed arguments:

```
greeting
```

This will return:

```
    => hello programmer! We heard you love the ruby program.
```

Call method with one passed argument:

```
greeting(name:"isaac")
```

This will return:

```
     => hello isaac! We heard you love the ruby program.
```

Call method with two passed argument:

```
greeting(name:"kaylee", language: "python")
```

This will return:

```
=> hello kaylee! We heard you love the python program.
```

Implicit return vs Explicit return

Arguments

Storing return values from a method to a variable