Ruby - Strings - Introduction

Numbers, boolean values and strings are some of the fundamental data types that we have explored in our previous challenges. In this set of tutorials, we turn our attention to the data type referred to as *String* or *Text* literals.

String data types are a sequence of characters, each of which occupies 1 byte of memory. Technically, you could represent the string using an array (or some collection) of characters, similar to that of classic languages like C. Any character outside the ASCII encoding set is restricted in C. How do we represent characters outside this range?

Before answering this question, let's learn about the different ways to represent strings,what they mean and their use cases.

Ruby provides 3 ways of including string literals into your source code.

1. **Single quoted strings**
   The easiest way of adding text is by surrounding it with single quote (apostrophe) symbols. However, characters like an apostrophe and a backslash within the string need to be escaped if they are present.

```
> 'Hello! Programmer. How\'s it going?'
```

1. **Double quoted strings**
   Double quoted strings are more flexible, and they allow special escape sequences, e.g. `\t`, `\n`, which don't work with single quoted strings. More importantly, they allow the embedding of arbitrary expressions. When a string is created, the expression in the string is evaluated, converted to a string and inserted into the text in place of the expression. This process is known as *interpolation*.

```
> "Hello! There is a tab \t here. Did you see?"
> "My name is Circle, and I love Pi. Pi is equal to #{Math::PI}"
```

1. **Here documents**
   This is helpful for putting large amounts of text without worrying about escape sequences or string evaluation. “Here documents” begin with `<<-`. These are followed immediately by an identifier or string that specifies the ending delimiter. (No space is allowed, to prevent ambiguity with the left-shift operator.)The text of the string literal begins on the next line and continues until the text of the delimiter appears on a line by itself. For example:

```
document = <<-HERE        # We begin with <<- followed by the ending delimiter HERE
This is a string literal.
It has two lines and abruptly ends with a newline...
HERE
```

(Example from *The Ruby Programming Language*)

In this introductory challenge, your task is to use each of the above three methods to return the text `Hello World and others!`