Regular Expressions (Regexes)

Introduction

In the following lessons we will learn how to create basic **Regular Expressions** in Python. Regular Expressions, also known as Regexes, are used to find different patterns of text. In general, regexes work by first specifying the rules for the set of possible patterns that you want to find and then making queries such as "Is this pattern found at the beginning of this string?" or “Is there a match for this pattern anywhere in this string?”. We will learn for example, how to write regular expressions to find phone numbers, names, and email addresses.

By the end this lesson you should be able to read and write basic regular expressions in Python and know how to apply them to get useful financial information from 10-Ks.

Raw Strings

Before we dive in and start creating our first regular expression, let's take a quick look at **Raw Strings**, since we will be using them to create our regexes.

In Python string literals are specified using either single quotes (`'`) or double quotes (`"`); and the backslash (`\`) character is used to escape characters that have a special meaning, such as a newline (`\n`) or tab (`\t`). Let's see a simple example:

```
In [1]: print('Hello\n\tWorld')
```

```
Hello
        World
```

We can clearly see that the `print()` function has replaced the `\n` with a new line, and the `\t` with a tab.

In some cases, however, you may want the `print()` function to interpret the string literally. This means that you don’t want characters preceded by a backslash (`\`) to be interpreted as special characters. In these cases, you can prefix the string literal with the letter `r`. Such strings are known as **Raw Strings** and treat backslashes (`\`) as literal characters. To see how this works, let's print the same string literal we had before but now as a raw string:

In [2]:
```python
print(r'Hello\n\tWorld')
```

```
Hello\n\tWorld
```

We can clearly see that by adding an `r` before the first quote of the string literal, both `\n` and `\t`, are no longer treated as special characters. It is important to note, that the `r` doesn't change the type of the string literal, but rather, it just changes how the string literal is interpreted. So, without the `r`, backslashes are used to escape characters and with the `r`, backslashes are treated as literal characters.

We will be using raw strings to create our regular expressions, because regular expressions themselves, also use the backslash character (`\`) to indicate their own special characters. Therefore, by using raw strings, we avoid the problem of Python interpreting the special characters in regexes in the wrong way.