UNIVERSITY OF ENGINEERING AND TECHNOLOGY PESHAWAR, JALOZAI CAMPUS

Lab 8 Inheriting from Built-in Data Types

Lab Title: EE-271 "OOP & Data Structures Lab"

Time: 1 hour/ Task

---

1. The built-in `str` class allows you to create **strings** in Python. Strings are sequences of characters that you'll use in many situations, especially when working with textual data. From time to time, the standard functionalities of Python's `str` may be insufficient to fulfill your needs. So, you may want to create custom string-like classes that solve your specific problem.
   You'll typically find at least two reasons for creating custom string-like classes:
   - **Extending** the regular string by adding new functionality
   - **Modifying** the standard string's functionality
   - You need a string-like class that implements a new method to count the number of words in the underlying string.
   - In this example, your custom string will use the whitespace character as its default word separator. However, it should also allow you to provide a specific separator character. To code a class that fulfills these needs, you can do something like this:

```
class WordCountString(str):
    def words(self, separator=None):
        return len(self.split(separator))
```

   a. Run the following code.

```
sample_text = WordCountString(
    """Knowledge of Python can be an added advantage in
terms of skills, as it enables
      electrical engineers to leverage its versatility
for a wide range of tasks the
      scope of specialized software.""")

sample_text.words()
```

2. To learn how to modify the standard behavior of `str` in a custom string-like class, say that you need a string class that always prints its letters in uppercase. You can do this by overriding the `.__str__()` special method, which takes care of how string objects are printed.
   Here's an `UpperPrintString` class that behaves as you need:

```
class UpperPrintString(str):
```

```
    def __str__(self):
        return self.upper()
```

a. Run the following code.

```
sample_string = UpperPrintString("Hello, Pythonista!")
print(sample_string)
sample_string
```

Encouraging

1. Build a custom list class that will only accept integers and floats. If we attempt to append any other datatype, let's say for arguments sake a string, we will generate a Custom Error exception, that will helpfully inform the user of our class that *this* particular list can *only* accept integers and floats.

   We first begin by creating a IntFloatList custom class that inherits from the build in list class. This now means IntFloatList will act like a list in every respect, except when append is called.

Recommended Reading

1. https://realpython.com/inherit-python-str/
2. https://blog.finxter.com/how-to-count-the-number-of-words-in-a-string-in-python/
3. https://towardsdatascience.com/python-tricks-inheriting-from-built-in-data-types-f6cbeb8d88a5