ICS3U: Basic String Formatting

Most of our programs so far have dealt with the *content* of strings, but have not focused on how these strings *appear*. There are a number of methods available for formatting strings. Two of the more useful ones are `upper` and `lower`, which convert all alphabetic characters in a string to upper- or lowercase respectively.

```
>>> "This is a string!".upper()
'THIS IS A STRING!'
>>> "The ANSWER is 42.".lower()
'the answer is 42.'
```

There are other methods to convert between cases, including `capitalize`, `swapcase` and `title`. You can read about them in the official Python documentation if you are interested.

In many cases, data are entered with extra characters that are not relevant to our programs. For example, a text file might contain a list of integer values as follows.

```
10
25
32
```

While this is what we might see as humans, there may be non-printable characters on each line, such as those that indicate a new line. Thus, the actual contents of the file might be something like this.

```
10\r\n
25\r\n
32
```

The use of different line endings by various operating systems (`\r\n` for Windows, `\n` for Mac and Linux) can be problematic. Try opening a text file that was created using a Mac or Linux system on a Windows machine and you may see the numbers jumbled together. To remove extra characters from the ends (left, right, or both) of a string, Python provides three methods: `strip`, `lstrip` and `rstrip`.

```
>>> s = "**ab*cd**"
>>> s.strip("*")
'ab*cd'
>>> s.rstrip("*")
'**ab*cd'
>>> s.lstrip("*")
'ab*cd**'
```

Note that `strip` never removes characters from *inside* of a string – this is done using the `replace` method or some other technique. To be more precise, all three `strip` methods work their way from the outside toward the inside of a string. The instant that a character is found that is *not* one of the characters in the argument, it stops.

```
>>> s = "ABCABABCA*ACAB"
>>> s.rstrip("ABC")
'ABCABABCA*'
```

When called with no arguments, `strip` removes all whitespace characters from the beginning and end of a string.

```
>>> s = "   \tABC  \r\n"
>>> s.strip()
'ABC'
```

Now, consider the following code that calculates the tax (15%) and final cost of an item and displays a summary of the transaction.

```
unit_cost = float(input("What is the cost per unit? "))
quantity = int(input("How many units purchased? "))
pre_tax_cost = unit_cost * quantity
taxes = pre_tax_cost * 0.15
total_cost = pre_tax_cost + taxes
print("Unit cost:    ", unit_cost)
print("Quantity:     ", quantity)
print("Pre-tax cost: ", round(pre_tax_cost, 2))
print("Taxes:        ", round(taxes, 2))
print("Total cost:   ", round(total_cost, 2))
```

The output might look something like this.

```
What is the cost per unit? 12.95
How many units purchased? 3
Unit cost:     12.95
Quantity:      3
Pre-tax cost:  38.85
Taxes:         5.83
Total cost:    44.68
```

It would be nice if we could align all of the values along their decimal points, so that they appear in a vertical column. We can play around with adding spaces and tabs, but even these will be off-shifted if the numbers are sufficiently large. Instead, we can use the justification methods `center`, `ljust` and `rjust` to set the number of spaces in which to align a particular string. Each takes at least one argument, for the number of spaces in which to display the string, and an optional second argument for a "fill" character.

```
>>> "abcdef".rjust(10)
'    abcdef'
>>> "abcdef".center(10, "*")
'**abcdef**'
```

```
unit_cost = float(input("What is the cost per unit? "))
quantity = int(input("How many units purchased? "))
pre_tax_cost = unit_cost * quantity
taxes = pre_tax_cost * 0.15
total_cost = pre_tax_cost + taxes
print("Unit cost:    ", str(unit_cost).rjust(8))
print("Quantity:     ", str(quantity).rjust(8))
print("Pre-tax cost: ", str(round(pre_tax_cost, 2)).rjust(8))
print("Taxes:        ", str(round(taxes, 2)).rjust(8))
print("Total cost:   ", str(round(total_cost, 2)).rjust(8))
```

This produces the following output.

```
What is the cost per unit? 12.95
How many units purchased? 3
Unit cost:        12.95
Quantity:             3
Pre-tax cost:     38.85
Taxes:             5.83
Total cost:       44.68
```

There are other ways in which we can format strings that will allow for additional options. We will look at some of them in the next lesson.