ICS3U: Slicing Strings

We saw how we can reference a single character in a string, `s`, by providing an index, such as `s[0]`. It is also possible to obtain a sequence of characters from a string using a technique called **slicing**. To specify a slice, we must provide both the starting and ending indices separated by a colon. Look at the four examples below and see if you can match the indices to the letters in the word 'Programming'.

```
>>> s = "Programming"
>>> s[2:6]
'ogra'
>>> s[:5]
'Progr'
>>> s[3:]
'gramming'
>>> s[:]
'Programming'
```

In the first example we obtain all characters from index 2 up until, but not including, index 6. In this sense, slices operate similar to `range` – the last index is not included in the slice. This is why we get 'ogra' instead of 'ogram'. The other examples illustrate some shorthand notation for slices. If the first index is omitted, as in the second example, the slice begins at the start of the string. If the last index is omitted, as in the third example, then the slice continues until the end of the string. If both indices are omitted, then the slice is the entire string. This may not seem very useful, but we will see a possible application of this later when dealing with lists.

A third parameter can be specified that acts as a step value, in the same way manner as `range` or `randrange`.

```
>>> s[2:10:2]
'ormi'
>>> s[::-1]
'gnimmargorP'
```

The first example begins at index 2 and counts every other index until (but not including) index 10. The second example begins at the start of the string and finishes at the end of the string, but traverses the indices *backward*. This is particularly useful for reversing a string. Here is a short program that asks the user to enter a string and displays it backward.

```
string = input("Enter something: ")
rev_string = string[::-1]
print("Your string, reversed, is", rev_string)
```

Strings in Python are **immutable**. This means that once a string has been created, it cannot be modified. This may not appear to be the case, because we can do things like the following.

```
>>> name = "Jon"
>>> name += " Garvin"
>>> name
'Jon Garvin'
```

It might appear that we have modified the value of `name` by appending additional characters to it, but remember that `+=` is an assignment operator. Behind the scenes, Python is creating a temporary variable that has the original value of `name`, appending the new characters, then *overwriting* the original value of `name` with the new data. It is always possible to overwrite an existing variable.

To see that strings are immutable, we can try to modify one of the existing characters by assigning it a new value via indexing.

```
>>> name[0] = "D"
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

The error message is telling us that we cannot alter the value of the character in index zero because individual characters cannot be changed for a `str`. Not all sequence types are immutable. As we will see later, lists are **mutable** and the values of individual elements can be changed at any time. For now, what this means is that if we need to change one or more characters in a string, we will need to "rebuild" them using indexing, slicing and concatenation.

```
>>> name = "D" + name[1:]
>>> name
"Don Garvin"
>>> name = name[:3] + name[7:]
>>> name
"Donvin"
```

Below is a program that allows a user to display portions of a string. It uses the `get_integer` function that we created earlier.

```
import math

def get_integer(low=-math.inf, high=math.inf, prompt="Enter a value: "):
   integer = int(input(prompt))
   while integer < low or integer > high:
      integer = int(input("Invalid value, try again: "))
   return integer

string = input("Enter something: ")
start = get_integer(low=0, high=len(string), prompt="Enter the start index: ")
end = get_integer(low=start, high=len(string), prompt="Enter the end index: ")
new_string = string[start:end]
print("The characters from index", start, "to", end, "are '" + new_string + "'.")
```

The value of `high` is set to `len(string)` to ensure that the user does not select an index that is out of bounds. The value of `low` in the second call to `get_integer` is set to `start`, so that the ending index is not smaller than the starting index. Once the values of `start` and `end` are obtained, the slice `[start:end]` is displayed to the screen.