ICS3U: Substrings

Related to strings are **substrings**. A substring is a smaller string made up of adjacent characters taken from a larger string. For instance, “cat” is a substring of “catatonic”, whereas “dog” is not a substring of “dropping”, even though “dropping” contains all of the letters in “dog”.

We can always for if a particular substring is contained in a string using `in`. If the substring is in the string, the result is `True`. Otherwise, it is `False`.

```
>>> s = "computer"
>>> "put" in s
True
>>> "opt" in s
False
```

The code below asks the user to enter a string, then determines if it contains a lower- or uppercase ‘a’.

```
s = input("Enter a string: ")
if "a" in s or "A" in s:
   print("The string contains an A.")
else:
   print("The string contains no A's.")
```

To count the number of occurrences of a substring in a string, use the `count` method. Unlike some of the earlier methods, like `isalpha`, `count` requires a substring as an argument. Only one substring can be checked at a time – attempting to use more than one argument will cause an error.

```
>>> s = "AbcAdeAfg"
>>> s.count("A")
3
>>> s.count("A", "B")
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: slice indices must be integers or None or have an __index__ method
```

Note that `count` will only count the number of non-overlapping substrings in a string. The code below begins counting from the left. Once the first “AA” substring is found, the remaining ‘A’ does not have a match, so the count is 1 rather than 2.

```
>>> s="AAA"
>>> s.count("AA")
1
```

Let’s modify the earlier program to not only test a string for the presence of ‘A’s, but to count them too.

```
s = input("Enter a string: ")
total_A = s.count("a") + s.count("A")
if total_A > 0:
   print("The string contains", total_A, "A's.")
else:
   print("The string contains no A's.")
```

To determine whether or not a string begins or ends with a specific substring, Python provides the `startswith` and `endswith` methods. These also require single substring arguments.

```
>>> s = "hello"
>>> s.startswith("he")
True
>>> s.endswith("zzz")
False
>>> s.endswith("O")
False
```

Remember that to the Python interpreter, an uppercase letter is not the same as its lowercase equivalent.

While `in` can tell us if a given substring exists in a string, sometimes we want to know *where* in the string that substring is located. There are a number of methods that will determine the location of a substring. The most direct way to find the location of a given substring is to use `find`. If a substring is found, the index of the *first* character of the substring is returned. If the substring does not exist in the string, a value of -1 is returned instead. This does not imply that the substring begins at the last character! It is just a value to inform you that the substring does not exist.

```
>>> s = "programming"
>>> s.find("ram")
4
>>> s.find("ewe")
-1
```

The `rfind` method does the same thing, but searches a string from right-to-left instead. This can be useful for finding the last occurrence of a substring. Note that the value returned is still the first (leftmost) character of the substring.

```
>>> s = "papaya"
>>> s.rfind("pa")
2
```

Both `find` and `rfind` can take arguments specifying starting and ending indices to check. Anything outside of these bounds is ignored and, as usual, the stop index is not included.

```
>>> s = "humuhumunukunukuapua'a"
>>> s.find("nuk")
8
>>> s.find("nuk", 9)
12
>>> s.find("nuk", 5, 10)
-1
```

The `index` method does essentially the same thing as `find`, but returns an error instead of -1 if the substring is not found. This is useful if you are writing a program that handles exceptions. It also has the added benefit that it is compatible with other sequence types, such as tuples and lists, for which there is no `find` method.

```
>>> s = "my fancy string"
>>> s.index("abc")
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    s.index("abc")
ValueError: substring not found
```

If you are determined to use `index` instead of `find` (and there are several good reasons to do so), a work-around to prevent this run-time error is to test for a substring's existence using `in` before attempting to determine its location.

```
string = input("Enter a string: ")
sub = input("Enter a substring: ")
if sub in string:
   i = string.index(sub)
   print("The substring begins at index", i)
else:
   print("The substring does not exist.")
```

There is a right-to-left equivalent of `index`, called `rindex`, which acts in a manner similar to `rfind`. Both `index` and `rindex` can take arguments specifying their starting and stopping indices.

One final method that is fairly useful is replace which, as its name suggests, replaces all occurrences of a substring with a new substring. A third, optional argument can be used to limit the number of replacements made. Note that this replacement is not permanent unless you explicitly assign the altered value to a variable. In the example below, notice how `s` remains untouched.

```
>>> s = "Canada"
>>> s.replace("a", "o")
'Conodo'
>>> s
'Canada'
```

In the following examples, we change the value of `s` by overwriting it with the new values.

```
>>> s = "Canada"
>>> s = s.replace("a", "o")
>>> s
'Conodo'
>>> s = s.replace("ono", "ede")
>>> s
'Cededo'
>>> s.replace("ed", "ax", 1)
'Caxedo'
```

The `replace` method can be used to remove characters from a string by replacing them with an empty string instead.

```
>>> s = "This string contains no i's"
>>> s = s.replace("i", "")
>>> s
"Ths strng contans no 's"
```