Week 4: Strings and String Manipulation

Strings

Because most programs define and gather some sort of data and then do something useful with it, it helps to classify different types of data. The first data type we'll look at is the string. Strings are quite simple at first glance, but you can use them in many different ways.

A *string* is a series of characters. Anything inside quotes is considered a string in Python, and you can use single or double quotes around your strings like this:

"This is a string."

'This is also a string.'

Changing Case in a String with Methods

One of the simplest tasks you can do with strings is change the case of the words in a string. Look at the following code, and try to determine what's happening:

```
name = "al mustagbal"
print(name.title())
```

Al mustagbal

In this example, the variable name refers to the lowercase string " al mustagbal ". The method title() appears after the variable in the print() call. A *method* is an action that Python can perform on a piece of data. The dot (.) after name in name.title() tells Python to make the title() method act on the variable name. Every method is followed by a set of parentheses, because methods often need additional information to do their work. That information is provided inside the parentheses. The title() function doesn't need any additional information, so its parentheses are empty.

For example, you can change a string to all uppercase or all lowercase letters like this:

```
name = "al mustagbal"
print(name.upper())
print(name.lower())
```

AL MUSTAGBAL

al mustagbal

The lower() method is particularly useful for storing data. You typically won't want to trust the capitalization that your users provide, so you'll convert strings to lowercase before storing them. Then when you want to display the information, you'll use the case that makes the most sense for each string.

Using Variables in Strings

In some situations, you'll want to use a variable's value inside a string. For example, you might want to use two variables to represent a first name and a last name, respectively, and then combine those values to display someone's full name:

```
first_name = "AlMustagbal"
last_name = "University"
full_name = f"{first_name} {last_name}" #1
print(full_name)
```

AlMustagbal University

To insert a variable's value into a string, place the letter f immediately before the opening quotation mark 1. Put braces around the name or names of any variable you want to use inside the string. Python will replace each variable with its value when the string is displayed.
These strings are called ***f-strings***. The ***f*** is for *format*, because Python formats the string by replacing the name of any variable in braces with its value. The output from the previous code is:

```
AlMustagbal University
```

Adding Whitespace to Strings with Tabs or Newlines
In programming, *whitespace* refers to any nonprinting characters, such as spaces, tabs, and end-of-line symbols. You can use whitespace to organize your output so it's easier for users to read.
To add a tab to your text, use the character combination **\t**:

```python
print("Python")
print("\tPython")
```

```
Python
	Python
```

To add a newline in a string, use the character combination **\n**:

```python
print("Languages:\nPython\nC++\nJava")
```

```
Languages:
Python
C++
Java
```

You can also combine tabs and newlines in a single string. The string **"\n\t"** tells Python to move to a new line, and start the next line with a tab. The following example shows how you can use a one-line string to generate four lines of output:

```python
print("Languages:\n\tPython\n\tC++\n\tJava")
```

```
Languages:
	Python
	C++
	Java
```

```python
language = ' python '
print(language.rstrip())
print(language.lstrip())
```

```
' python'
'python '
```

Removing Prefixes

When working with strings, another common task is to remove a prefix. Consider a URL with the common prefix *https://*. We want to remove this prefix, so we can focus on just the part of the URL that users need to enter into an address bar. Here's how to do that:

```
url = 'https://uomus.edu.iq/'
print(url)
print(url.removeprefix('https://'))
```

```
https://uomus.edu.iq/
uomus.edu.iq/
```

Basic String Operations

Python provides several ways to access the individual characters in a string. Strings also have methods that allow you to perform operations on them.

Iterating over a String with the for Loop

One of the easiest ways to access the individual characters in a string is to use the for loop.
Here is the general format:

for *variable* in *string:*
statement
statement
etc.

We say that the loop iterates over the characters in the string. Here is an example:

```
name = 'AlMustagbal'
for ch in name:
  print(ch)
```

```
A
l
M
u
s
t
a
g
b
a
l
```

This program asks the user to enter a string. It then uses a for loop to iterate over the string, counting the number of times that the letter T (uppercase or lowercase) appears.

```
# Create a variable to use to hold the count.
# The variable must start with 0.
count = 0
# Get a string from the user.
my_string = input('Enter a sentence: ')
```

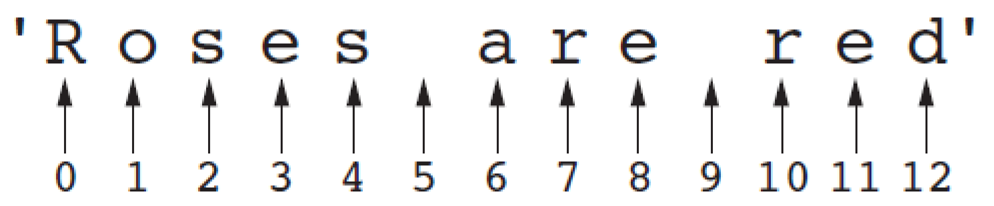
```
# Count the Ts.
for ch in my_string:
   if ch == 'T' or ch == 't':
      count += 1
# Print the result.
print('The letter T appears', count, 'times.')
```

```
Enter a sentence:  Today we sold twenty-two toys
The letter T appears 5 times.
```

Indexing

Another way that you can access the individual characters in a string is with an index. Each character in a string has an index that specifies its position in the string. Indexing starts at 0, so, the index of the first character is 0, the index of the second character is 1, and so forth. The index of the last character in a string is 1 less than the number of characters in the string. Figure shows the indexes for each character in the string 'Roses are red'. The string has 13 characters, so the character indexes range from 0 through 12.

Figure String indexes

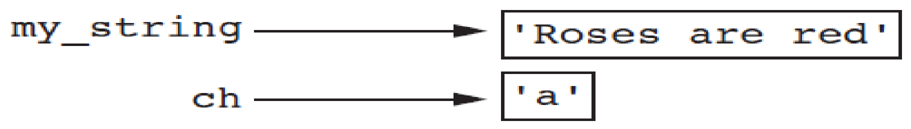


You can use an index to retrieve a copy of an individual character in a string, as shown here:

my_string = 'Roses are red'
ch = my_string[6]

The expression my_string[6] in the second statement returns a copy of the character at index 6 in my_string. After this statement executes, ch will reference 'a' as shown in Figure

Figure Getting a copy of a character from a string



Here is another example:
my_string = 'Roses are red'
print(my_string[0], my_string[6], my_string[10])
This code will print the following: R a r

```
my_string = 'Roses are red'
print(my_string[0], my_string[6], my_string[10])
```

R a r

```
my_string = 'Roses are red'
print(my_string[-1], my_string[-2], my_string[-13])
```

d e R

String Concatenation

A common operation that performed on strings is *concatenation*, or appending one string to the end of another string. You have seen examples in earlier chapters that use the + operator to concatenate strings. The + operator produces a string that is the combination of the two strings used as its operands. The following interactive session demonstrates:

```
message = 'Hello ' + 'world'
print(message)
```

Hello world

```
name = 'Al Mustagbal'
print('The name is', name)
name = name + ' University'
print('Now the name is', name)
```

The name is Al Mustagbal
Now the name is Al Mustagbal University

```
full_name = 'Al Mustagbal University'
middle_name = full_name[3:12]
print(middle_name)
```

Mustagbal

```
numbers = '0123456789'
print(numbers[0:10:2])
```

02468

```
letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
print(letters[0:26:2])
```

ACEGIKMOQSUWY