# Strings

Strings are a data type in Python for dealing with text. Python has a number of powerful features for manipulating strings.

**Creating a string:** string is created by enclosing text in quotes. You can use either single quotes ' , or double quotes ". Triple-quote can be used for multi-line string. Here are some examples:

```
s = 'Hello'
t = "Hello"
m = """This is a long string that is
spread across two lines."""
```

**Input:** recall from first lecture that when getting numerical input we use an eval statement with the input statement, but when getting text, we do not use eval. The difference is illustrated below.

```
num = eval(input('Enter a number: '))
string = input('Enter a string: ')
```

**Empty string:** the empty sting '' is a string with nothing in it.

**Length:** to get the length of a string (how many character it has), we use the built-in function (len). For example, len('Hello') is 5.

**Concatenation and Repetition**

The operators + and * can be used on strings. This operation is called ***concatenation***. The * repeats a string a certain number of times. Here are some examples.

| Expression | Result |
|---|---|
| `'AB'+'cd'` | `'ABcd'` |
| `'A'+'7'+'B'` | `'A7B'` |
| `'Hi'*4` | `'HiHiHiHi'` |

**Example:** if we want to print a long row of dashes, we can do the following:

```
print('-'*75)
```

**Example:** the + operator can be used to build up a string, piece by piece. Here is an example that repeatedly asks the user to enter a letter and builds up a string consisting of only the vowels that the user entered.

```
s = ''
for i in range(10):
    t = input('Enter a letter: ')
    if t=='a' or t=='e' or t=='i' or t=='o' or t=='u':
        s = s + t
print(s)
```

**The in operator**

The in operator is used to tell if a string contains something. For example:

```
if 'a' in string:
    print('Your string contains the letter a.')
```

You can combine in with the not operator to tell if a string does not contain something:

```
if ';' not in string:
    print('Your string does not contain any semicolons.')
```

**Example:** in the previous section we had the long if condition.

```
if t=='a' or t=='e' or t=='i' or t=='o' or t=='u':
```

Using the in operator, we can replace that statement with the following:

```
if t in 'aeiou':
```

## Indexing

We will often want to pick out individual characters from a string. Python uses square brackets to do this. The table below gives some examples of indexing the string s=’Python’

| Statement | Result | Description |
|---|---|---|
| `s[0]` | `P` | first character of `s` |
| `s[1]` | `y` | second character of `s` |
| `s[-1]` | `n` | last character of `s` |
| `s[-2]` | `o` | second-to-last character of `s` |

- The first character of s is s[0], not s[1]. Remember that in programming, counting usually starts at 0, not 1.
- Negative indices count backwards from the end of the string.

**A common error:** suppose s = ‘Python’ and we try to do s[12]. There are only six characters in the string and Python will raise the following error message.

```
IndexError: string index out of range
```

You will see this message again. Remember that it happens when you try to read past the end of a string.

## Slices

A slice is used to pick out part of a string. It behaves like a combination of indexing and the range function. Below we have some examples with the string s = ‘abcdefghij’.

```
index:     0 1 2 3 4 5 6 7 8 9
letters:   a b c d e f g h i j
```

| Code | Result | Description |
|---|---|---|
| `s[2:5]` | `cde` | characters at indices 2, 3, 4 |
| `s[ :5]` | `abcde` | first five characters |
| `s[5: ]` | `fghij` | characters from index 5 to the end |
| `s[-2: ]` | `ij` | last two characters |
| `s[ : ]` | `abcdefghij` | entire string |
| `s[1:7:2]` | `bdf` | characters from index 1 to 6, by twos |
| `s[ : :-1]` | `jihgfedcba` | a negative step reverses the string |

- The basic structure is

  String name[starting location : ending location+1]

  Slice have the same quirk as the range function in that they do not include the ending location. For instance, in the example above, s[2:5] gives the characters in indices 2,3, and 4, but not the characters in index 5.
- We can leave either the starting or ending locations blank. If we have the starting location blank, it defaults to the start of the string. So s[ : 5] gives the first five characters of s. if we leave the ending location blank, it defaults to the end of the string. So s[5: ] will give all the characters from index 5 to the end. If we use negative indices, we can get the ending characters of the string. For instance, s[-2: ] gives the last two characters.
- There is an optional third argument, just like in the range statement, that can specify the step. For example, s[1:7:2] steps through the string by twos, selecting the characters at indices 1,3, and 5. The most useful step is -1, which steps backwards through the string, reversing the order of the characters.

## Changing Individual Characters of a String

Suppose we have a string called s and we wat to change the character at index 4 of s to ‘X’. it is tempting to try s[4] =’X’, but that unfortunately will not work. Python strings are ***immutable***, which means we can’t modify any part of them. If we want to change a character of s, we have to instead build a new string from s and reassign it to s. Here is code that will change the character at index 4 to 'X':

```
s = s[:4] + 'X' + s[5:]
```

The idea of this is we take all the characters up to index 4, then X, and then all of the characters after index 4.

## Looping

Very often we will want to scan through a string one character at a time. A for loop like the one below can be used to do that. It loops through a string called s, printing the string, character by character, each on a separate lint:

```
for i in range(len(s)):
    print (s[i])
```

In the range statement we have len(s) that returns how long s is. So, if s were 5 characters long, this would be like having range(5) and the loop variable I would run from 0 to 4. This means that s [i] will run through the characters of s. This way of looping is useful if we need to keep track of our location in the string during the loop.

If we don't need to keep track of our location, then there is a simpler type of loop we can use:

```
for c in s:
    print(c)
```

This loop will step through s, character by character, with c holding the current character. You can almost read this like an English sentence, "For every character c in s, print that character."

## String methods

String come with a ton of methods, function that return information about the string or return a new string that is modified version of the original. Here are some of the most useful ones:

| Method | Description |
|---|---|
| `lower()` | returns a string with every letter of the original in lowercase |
| `upper()` | returns a string with every letter of the original in uppercase |
| `replace(x,y)` | returns a string with every occurrence of `x` replaced by `y` |
| `count(x)` | counts the number of occurrences of `x` in the string |
| `index(x)` | returns the location of the first occurrence of `x` |
| `isalpha()` | returns `True` if every character of the string is a letter |

**Important note:** one very important note about lower, upper, and replace is that they do not change the original string. If you want to change a string, s, to all lowercase, it is not enough to just use s.lower(). You need to do the following.

```
s = s.lower()
```

Here are some examples of string methods in action:

| Statement | Description |
|---|---|
| `print(s.count(' '))` | prints the number of spaces in the string |
| `s = s.upper()` | changes the string to all caps |
| `s = s.replace('Hi','Hello')` | replaces each `'Hi'` in `s` with `'Hello'` |
| `print(s.index('a'))` | prints location of the first `'a'` in `s` |

isalpha: the **isalpha** method is used to tell if a character is a letter or not. It returns True if the character is letter and False otherwise. When used with an entire string, it will only return True if every character of the string is a letter. isalpha method can be used in if condition. Here is a simple example:

```
s = input('Enter a string')
```

```
if s[0].isalpha():
    print('Your string starts with a letter')

if not s.isalpha():
    print('Your string contains a non-letter.')
```

**A note about index:** if you try to find index of something that is not in a string, Python will raise an error. For instance, if s='abc' and you try s.index('z'), you will get an error. One way around this is to check first, like below:

```
if 'z' in s:
    location = s.index('z')
```

Other string methods: there are many more string methods. For instance, there are methods isdigit and isalnum, which are analogous to isalpha. Some other useful methods we will learn about later are join and split. To see a list of all the string method, type dir(str) into the Python shell. If you do this, you will see a bunch of names that start with ---. You can ignore them. To read Python's documentation for one of the methods, say the isdigit method, type help(str.isdigit).

## Escape characters

The backslash, \, is used to get certain special characters, called escape characters, into your string. There are a variety of escape characters, and here are the most useful ones:

- \n the newline character. It is used to advance to the next line. Here is an example:

```
print('Hi\n\nthere!')
```

```
Hi

There!
```

- \' for inserting apostrophes into string. Say you have the following string:

```
s = 'I can't go'
```

This will produce an error because the apostrophe will actually end the string. You can use \' to get around this:

```
s = 'I can\'t go'
```

Another option is to use double quotes for the string:

```
"s = I can't go"
```

- \” analogous to \’
- \\ this is used to get the backslash itself. For example:

  filename = ‘c:\\programs\\file.py’
- \t the tab character

**Exercises**

1. Write a program that asks the user to enter a string. The program should then print the following:
   a. The total number of characters in the string.
   b. The string repeated 10 times.
   c. The first character of the string (remember that string indices start at 0).
   d. The first three characters of the string.
   e. The last three characters of the string.
   f. The string backwards.
   g. The seventh character of the string if the string is long enough and a message otherwise.
   h. The string with its first and last characters removed.
   i. The string in all caps.
   j. The string with every a replaced with an e.
   k. The string with every letter replaced by a spac
2. A simple way to estimate the number of words in a string is to count the number of spaces in the string. Write a program that asks the user for a string and returns an estimate of how many words are in the string.
3. Write a program that asks the user to enter a word and prints out whether that word contains any vowels (a,e,i,o,u).
4. Write a program that asks the user to enter a string s and then converts s to lowercase, removes all the periods and commas from s, and prints the resulting string.
5. Write a program that asks the user to enter a word and determines whether the word is a palindrome or not. A palindrome is a word that reads the same backwards as forwards.
6. Write a program that asks the user to enter a string, then prints out each letter of the string doubled and on a separate line. For instance, if the user entered HEY, the output would be

```
HH
EE
YY
```

7. Write a program that asks the user to enter a word and then capitalizes every other letter of that word. So if the user enters rhinoceros, the program should print rHiNoCeRoS.
8. Write a program that asks the user to enter their name in lowercase and then capitalizes the first letter of each word of their name.