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'

E-mail: hasanein.y.m.alhrabi@uomus.edu.iq

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'.

E-mail: hasanein.y.m.alhrabi@uomus.edu.iq

```
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