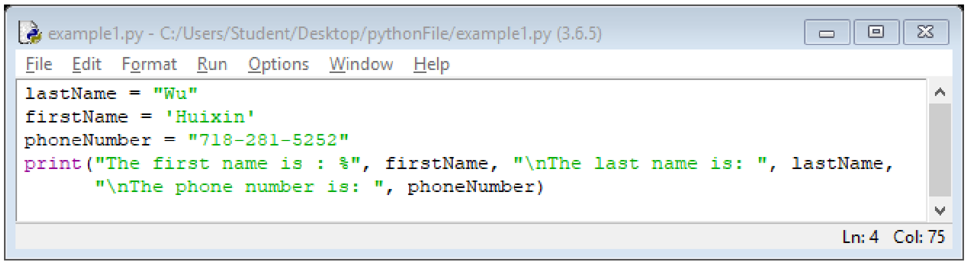
Strings

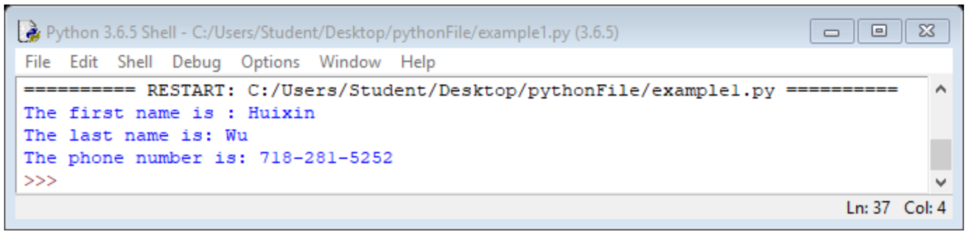
When programming, we usually call text a *string*. Think of a string as a collection of letters or characters. To create a string by putting single or double quotes around text because programming they need distinguish strings as value from variables. Also, we have to consistent in using single or double quotes, for example, if we start a string message with a single quote, we need to close the message with another single quote. If we start with a string with a single quote and we close it using a double quotes, when we run the Python program, the compiling will show us a syntax error.

Example) Use strings to create a complete message to display last name, first name, and phone number.


example1.py - C:/Users/Student/Desktop/pythonFile/example1.py (3.6.5)
File Edit Format Run Options Window Help
Ln: 4 Col: 75

```
lastName = "Wu"
firstName = 'Huixin'
phoneNumber = "718-281-5252"
print("The first name is : %", firstName, "\nThe last name is: ", lastName,
      "\nThe phone number is: ", phoneNumber)
```

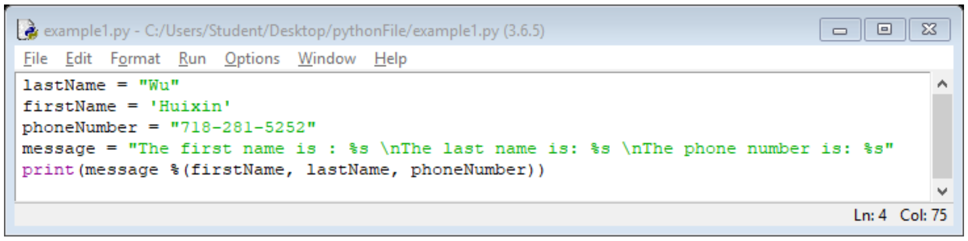
Result


Python 3.6.5 Shell - C:/Users/Student/Desktop/pythonFile/example1.py (3.6.5)
File Edit Shell Debug Options Window Help
Ln: 37 Col: 4

```
========== RESTART: C:/Users/Student/Desktop/pythonFile/example1.py ==========
The first name is : Huixin
The last name is: Wu
The phone number is: 718-281-5252
>>>
```

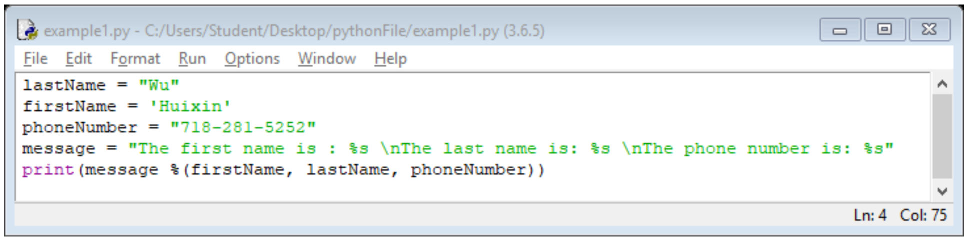
The comma symbol is used to concatenate to text with the variable.

We can also embed values in a string using **%s**, which is like a marker for a value that we want to add later.


example1.py - C:/Users/Student/Desktop/pythonFile/example1.py (3.6.5)
File Edit Format Run Options Window Help
Ln: 4 Col: 75

```
lastName = "Wu"
firstName = 'Huixin'
phoneNumber = "718-281-5252"
message = "The first name is : %s \nThe last name is: %s \nThe phone number is: %s"
print(message %(firstName, lastName, phoneNumber))
```

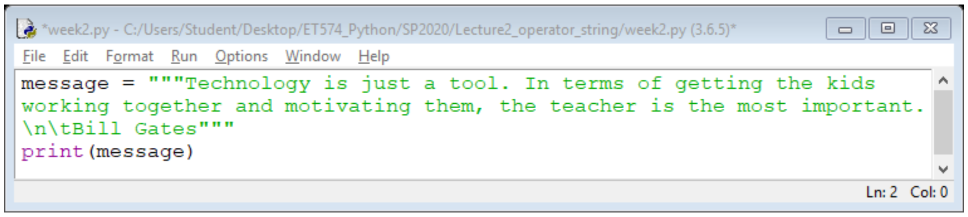
Result


example1.py - C:/Users/Student/Desktop/pythonFile/example1.py (3.6.5)
File Edit Format Run Options Window Help
Ln: 4 Col: 75

```
lastName = "Wu"
firstName = 'Huixin'
phoneNumber = "718-281-5252"
message = "The first name is : %s \nThe last name is: %s \nThe phone number is: %s"
print(message %(firstName, lastName, phoneNumber))
```

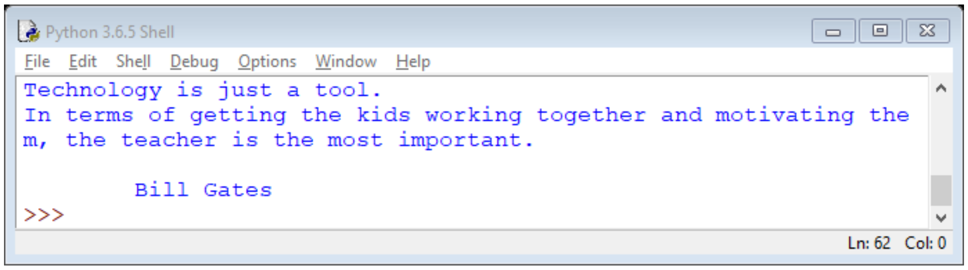
Multiline Strings

Triple double-quotes can use to create a multiline string to a variable


week2.py - C:/Users/Student/Desktop/ET574_Python/SP2020/Lecture2_operator_string/week2.py (3.6.5)
File Edit Format Run Options Window Help
Ln: 2 Col: 0

```
message = """Technology is just a tool. In terms of getting the kids
working together and motivating them, the teacher is the most important.
\n\tBill Gates"""
print(message)
```

Result



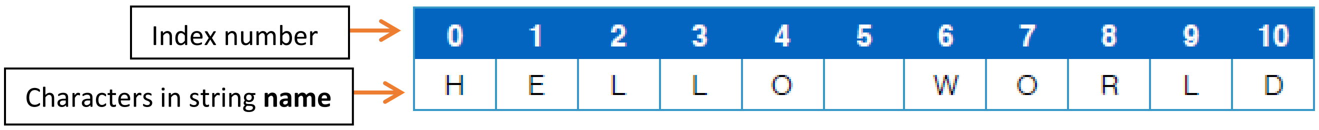
```
Technology is just a tool.
In terms of getting the kids working together and motivating the
m, the teacher is the most important.

        Bill Gates
>>>
```

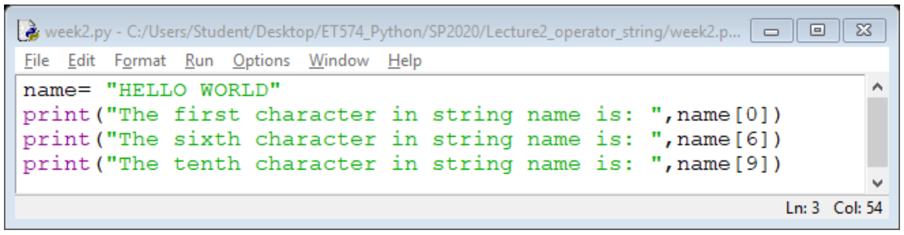
Strings are arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing Unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

The bracket operator `[]` can be used to read individual characters stored within a string. This requires an **index** value within the brackets (starting with 0) which indicates the character position in the string. We can think of string as an array of characters. So a string defined as the string **`name= “HELLO WORLD”`**



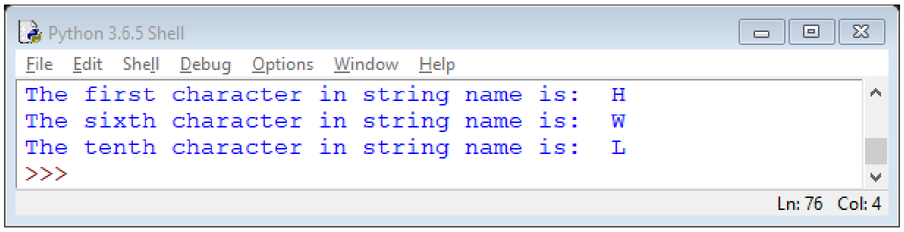
0	1	2	3	4	5	6	7	8	9	10
H	E	L	L	O		W	O	R	L	D



week2.py - C:/Users/Student/Desktop/ET574_Python/SP2020/Lecture2_operator_string/week2.p...
File Edit Format Run Options Window Help
Ln: 3 Col: 54

```
name= "HELLO WORLD"
print("The first character in string name is: ",name[0])
print("The sixth character in string name is: ",name[6])
print("The tenth character in string name is: ",name[9])
```

Result



```
The first character in string name is:  H
The sixth character in string name is:  W
The tenth character in string name is:  L
>>>
```

String slices

We can set a range of characters of a string using the slice syntax. The slice syntax specify the start index, follow by the symbol colon **:** , and then the end index, without including the end index. A segment of a string is called **slice.**

Example) Print a segment of string **strawberries** from the second to the fifth character.

```
fruit = " strawberries"
print(fruit[1:5])
```

Result

```
traw
>>>
```

We can specific a segment from the beginning of a string up to end-index.

Example) print a segment of string **strawberries** from the beginning to the seventh character.

```
fruit = "strawberries"
print(fruit[:7])
```

Result

```
strawbe
>>>
```

On the other hand, we can also specific a segment from a starting-index to the last character of a string

Example) print a segment of string `strawberries` from the fifth character to last character.

```
fruit = "strawberries"
print(fruit[5:])
```

Result

```
berries
>>>
```

String length

Python uses a built-in function ***Len()*** that returns the number of characters in a string.

Example) find the length of string `strawberries`

```
fruit = "strawberries"
fruit_length = len(fruit)
print("The fruit has ", fruit_length, "characters")
```

Result

```
The fruit has 12 characters
>>>
```

We can also use the ***Len()*** to find the last character of our string. For example, to find the last word of **`color = blue`**

```
color = "blue"
color_len = len(color)
color_last_index = color_len - 1
color_last_character = color[color_last_index]
print("The last character in string:", color, "is",
color_last_character)
```

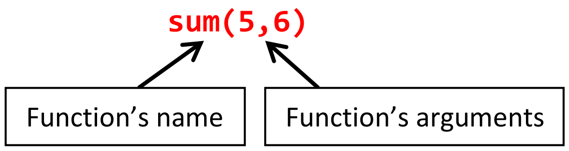
Result

```
The last character in string: blue is e
>>>
```

String Methods

Before we move on to how to use Python methods, let's review on some Python's terms which is function, classes/object, and methods. Almost everything in **Python** is an object, with its properties and methods. We are not going to have an intensive practice on the mentioned terms in this course, except for function, because they are standard features of Object Oriented Programming, OOP, which is the next level of computer programming.

Function: a function is a block of code to carry out a specific task and it is called by its name. All functions have arguments or not have arguments. The syntax of a function is as:



Classes: A **Class** is like an object constructor, or a "blueprint" for creating objects.

```
class MyClass:
  x = 5
```

Now we can use the class named **`MyClass`** to create objects. In this example, let's create an object named **`p1`** and print the value of **`x`**:

```
p1 = MyClass()
print(p1.x)
```

Object methods: Methods in objects are functions that belong to the object. A method is similar to a function—it takes arguments and returns a value—but the syntax is different.

Now that we now the basic terms in OOP, let's have a look at the built-in methods in Python for class **`string.`** Python has a set of built-in methods that we can use on strings:

- The **`strip`**`()` method removes any whitespace from the beginning or the end:

  ```
  a = "    Hello, World!"
  print(a.strip())
  ```
  Result
  ```
        Hello, World!
        >>>
  ```

- The **`lower`**`()` method returns the string in lower case

  ```
  a = "Hello, World!"
  print(a.lower())
  ```

  Result
  ```
        hello, world!
        >>>
  ```

- The **`upper`**`()` method returns the string in upper case

  ```
  a = "Hello, World!"
  print(a.upper())
  ```

Result
```
      HELLO, WORLD!
      >>>
```

- The **`replace()`** method replaces a string with another string

```
a = "Hello, World!"
print(a.replace("H", “J”))
```

Result
```
      Jello, World!
      >>>
```

- The **`split()`** method splits the string into substrings if it finds instances of the separator

```
a = "Hello World! Welcome to Python"
print(a.split("!"))
```

Result
```
      ['Hello World', ' Welcome to Python']
      >>>
```

- The **`find()`** method finds the character within the parenthesis

```
a = "Hello World! "
index = a.find("o")
print("index for letter o is:",index)
```

Result
```
      index for letter o is: 4
      >>>
```

We can also find a character of a string after an index:

```
a = "Hello World! "
index = a.find("o",5)
print("index for letter o is:",index)
```

Result
```
      index for letter o is: 7
      >>>
```

The **in** operator

Python uses membership operators **in** and **not in** is used to check if a certain phrase or character is present in a string. The word **in** is a Boolean operator that takes two strings and returns ***True*** if the first appears as a substring in the second.

```
msg = "Hello World! "
answer = "e" in msg
print("Is character e in string?",answer)
```

Result

```
      Is character e in string? True
      >>>
```

```
msg = "Hello World! "
answer = "e" not in msg
print("Is character e in string?",answer)
```

Result

```
      Is character e in string? False
      >>>
```

Concatenate Strings

To concatenate or combine two or more strings, we can use the + operator

```
a = "apple"
b = ", "
c = "grapes"
message = c+b+a+b+c
print(message)
```

Result

```
      grapes, apple, grapes
      >>>
```

String **format()**

Python cannot combine strings and integers or float value using the + operator. Instead, Python uses the **format()** method to combine string and numbers. The **format()** method takes the passed arguments, formats them, and places them in the string where the placeholders **{}** are:

```
age = 36
txt = "My name is John, and I am {}"
new_txt=txt.format(age)
print(new_txt)
```

Result

```
      My name is John, and I am 36
      >>>
```

We can also use multiple placeholder

```
price = 49.95
quantity = 3
itemno = 567
myorder = "I want {} pieces of item #{} for {} dollars."
final_order = myorder.format(quantity, itemno, price)
print(final_order)
```

Result

```
      I want 3 pieces of item #567 for 49.95 dollars.
```

You can use index numbers **`{0}`** to be sure the arguments are placed in the correct placeholders:

```
price = 49.95
quantity = 3
itemno = 567
myorder = "I want to pay {2} dollars for {0} pieces of item #{1}."
print(myorder.format(quantity, itemno, price))
```

Result

```
      I want to pay 49.95 dollars for 3 pieces of item #567.
```