Python - Basics

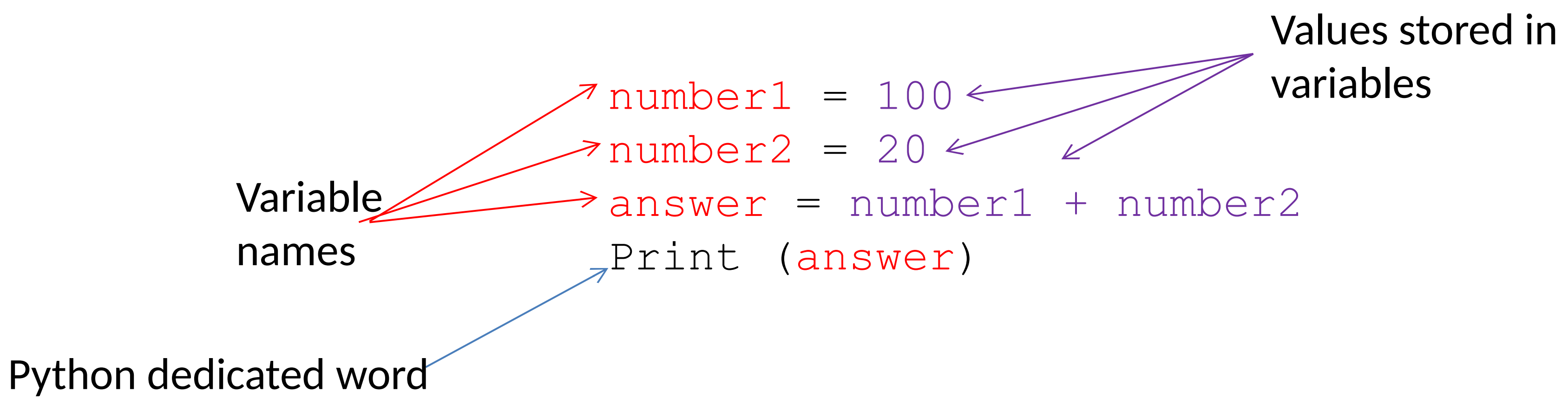
Storing and Manipulating Values

- **Variable**: A container for a value.
- Variables are created using assignment statement.
- The variable can be called whatever you want except Python dedicated word.

- The following statement creates a variable named `y` and stores `20` in it.

```
Y = 20
```

- Other examples are seen below.



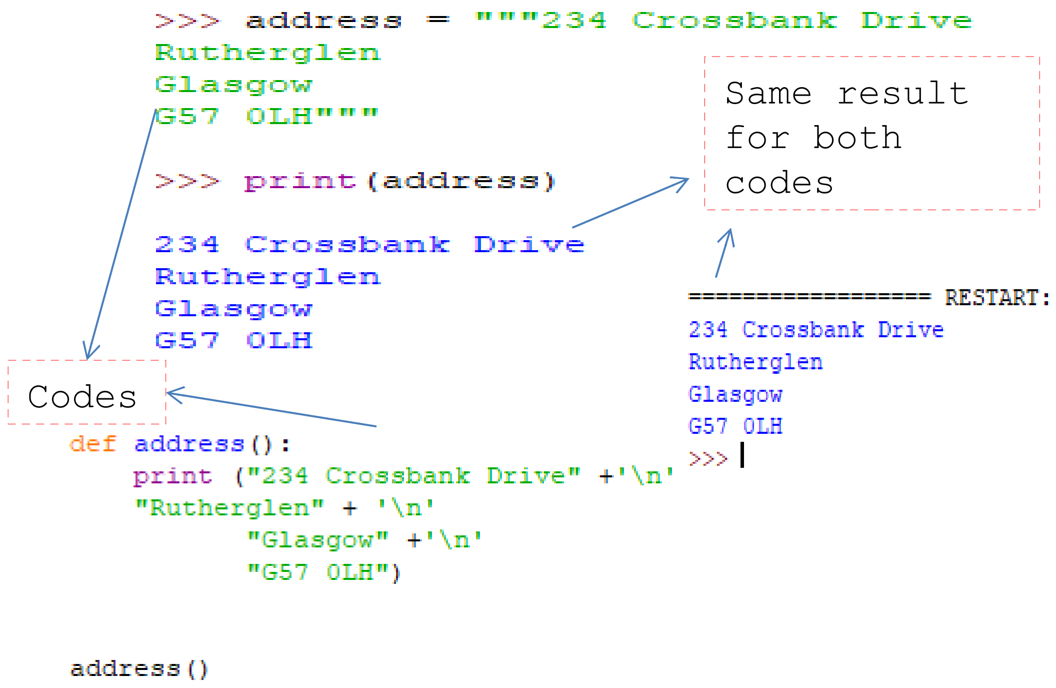
```
number1 = 100
number2 = 20
answer = number1 + number2
Print (answer)
```

Working with strings

- **String**: technical name for text. Text values need to be in speech marks (‘’) or (“”) but numbers do not.
- Be careful when inputting the following characters into strings as they have special meaning in Python and Python can get confuse if you use them in a string: “ ‘ \

Symbol	How to type this into a Python string
“	\“
‘	\‘
\	\\

- **Multiple-Line String**: If you want to put a string across multiple lines, you can either use the line break (`\n`) or you can enclose the entire thing in triple quotes. This will ensure that the formatting of the text remains the same.



```
>>> address = """234 Crossbank Drive
Rutherglen
Glasgow
G57 OLH"""
>>> print(address)

234 Crossbank Drive
Rutherglen
Glasgow
G57 OLH
```

```
def address():
    print ("234 Crossbank Drive" +'\n'
    "Rutherglen" + '\n'
           "Glasgow" +'\n'
           "G57 OLH")


address()
```

```
=================== RESTART:
234 Crossbank Drive
Rutherglen
Glasgow
G57 OLH
>>> |
```

Working with strings

- In Python, strings are an immutable iterable data type.
- **Immutable**: This means you cannot change an existing string (their state or content). Cannot be changed after it is created.
  - If you want to make a change you create a new string that is a variation on the original.

- Calling the same method with the same variable or value will guarantee the same output.

- **Iterable:** Anything that can be looped over, that appear on the right-side of the loop.

Other Python objects that are immutable:
`int, float, bool, unicode, tuple`

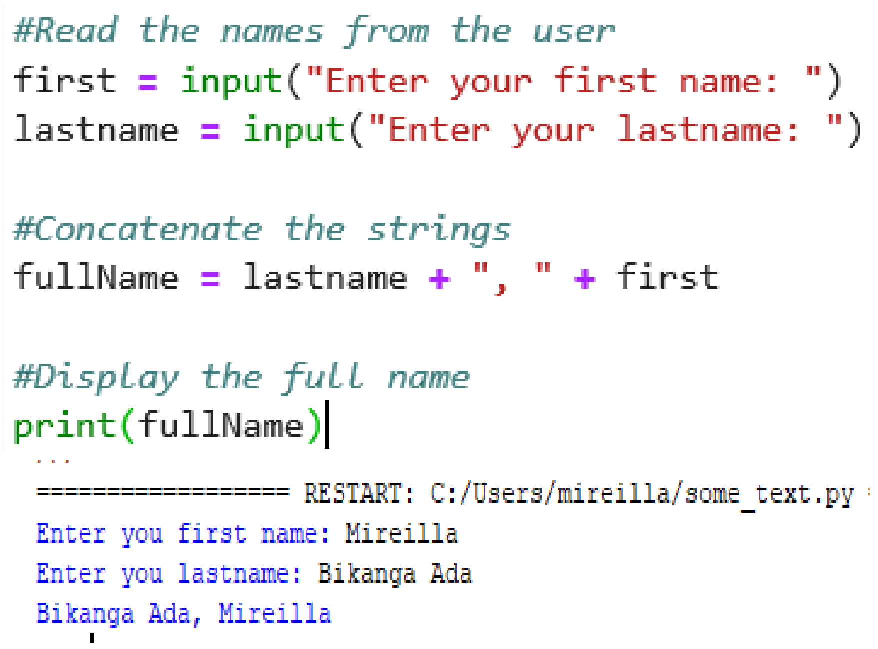
Other Python objects that are iterable:
`Lists, tuples, dictionaries`, and `sets`

Working with strings

- Strings can be manipulated with operators and passed to functions. Operations include
  - Concatenation
  - Computing the length of a string
  - Extracting individual characters from a string.

- **Concatenation**

Operator + can be used to concatenate the strings. Concatenation means joining together.



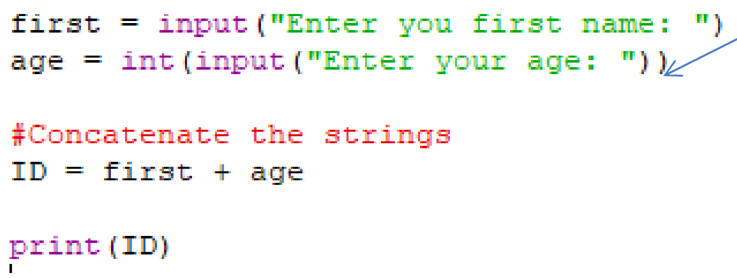
```
#Read the names from the user
first = input("Enter your first name: ")
lastname = input("Enter your lastname: ")

#Concatenate the strings
fullName = lastname + ", " + first

#Display the full name
print(fullName)
```

```
================== RESTART: C:/Users/mireilla/some_text.py
Enter you first name: Mireilla
Enter you lastname: Bikanga Ada
Bikanga Ada, Mireilla
```

Working with strings

!! String or integer as variable !!

If you define a variable as a string, even if it only contains numbers, you cannot later use that string as part of a calculation unless you convert it to a number. Same applies to defining a variable as integer or floating-point number. You will have to convert it as a string for concatenation to work.



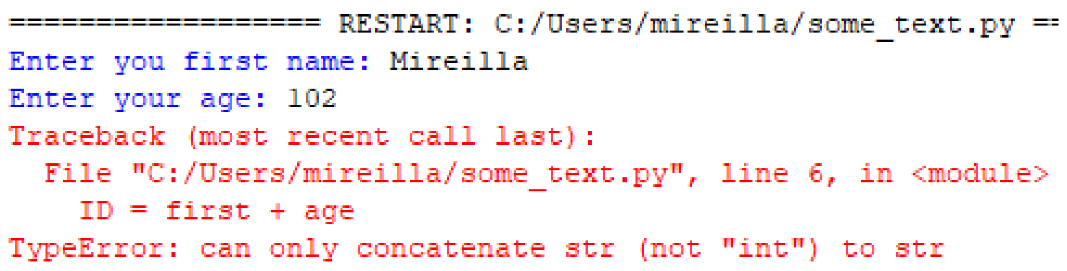
```
first = input("Enter you first name: ")
age = int(input("Enter your age: "))

#Concatenate the strings
ID = first + age

print(ID)
```

The example on the left throws an error. For this to work, convert the 2nd line into this:

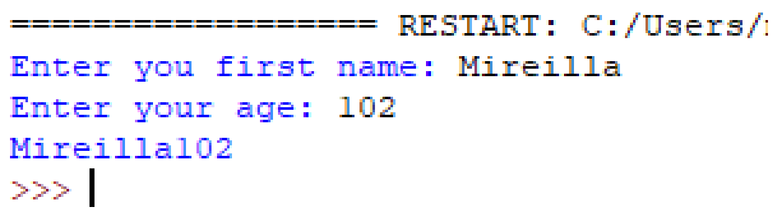
```
       age = str(int(input("Enter your age: ")))
```

or add this line after the 2nd line:

`age = str(age)` the you get the result below.



```
================== RESTART: C:/Users/mireilla/some_text.py ==
Enter you first name: Mireilla
Enter your age: 102
Traceback (most recent call last):
  File "C:/Users/mireilla/some_text.py", line 6, in <module>
    ID = first + age
TypeError: can only concatenate str (not "int") to str
```

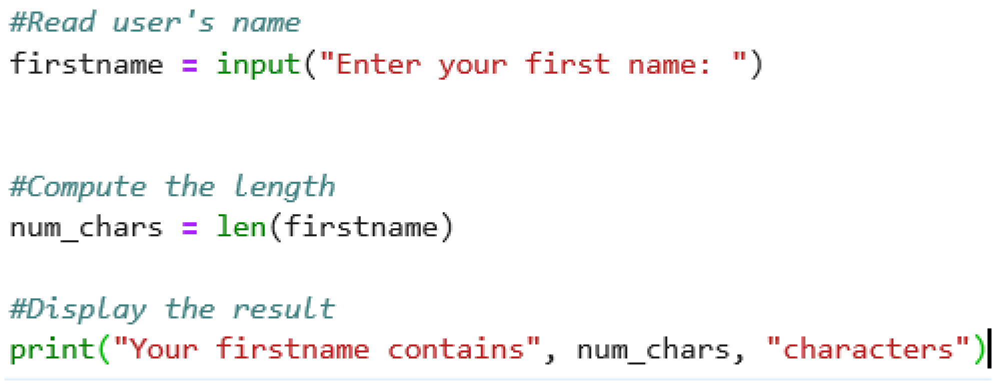


```
==================== RESTART: C:/Users/
Enter you first name: Mireilla
Enter your age: 102
Mireilla102
>>> |
```

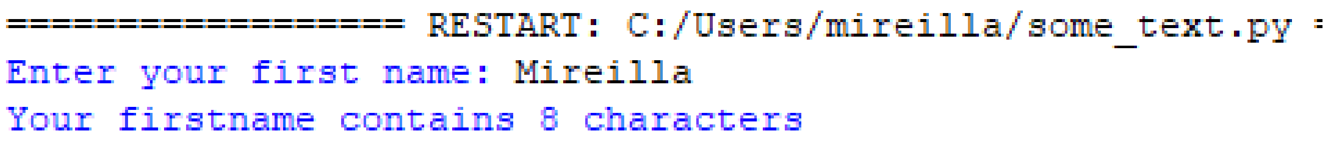
Working with strings

- **Finding the string’s length**

The number of characters in a string is referred to as a string’s length and is computed by calling the len() function.



```python
#Read user's name
firstname = input("Enter your first name: ")


#Compute the length
num_chars = len(firstname)

#Display the result
print("Your firstname contains", num_chars, "characters")
```

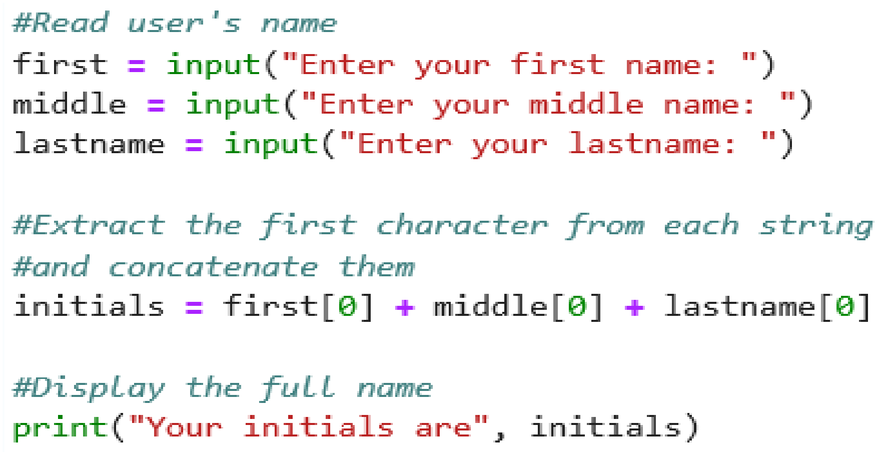


```
=================== RESTART: C:/Users/mireilla/some_text.py =
Enter your first name: Mireilla
Your firstname contains 8 characters
```

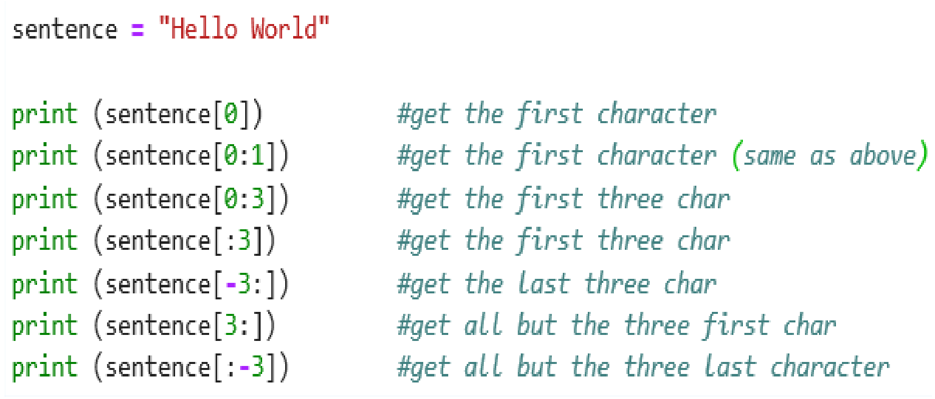
Working with strings

Slicing: taking a small piece of something bigger

- Each character in a string has a unique integer index. The first character in a string has index 0.
- The last character in a string has an index which is equal to the length of the string, minus one.
- A single character in a string is accessed by placing its index inside square brackets after the name of the variable containing the string.
- Consecutive characters can be accessed by including two indices, separated by a colon, inside the square bracket



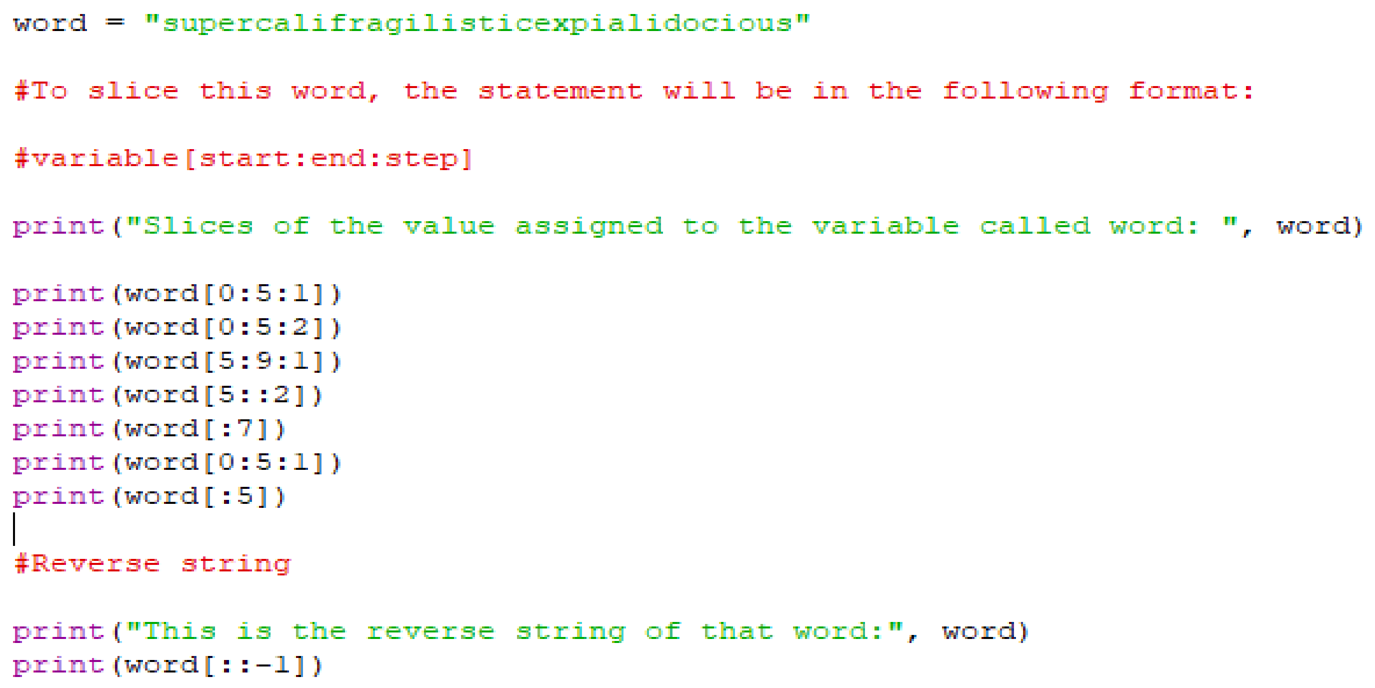
```
#Read user's name
first = input("Enter your first name: ")
middle = input("Enter your middle name: ")
lastname = input("Enter your lastname: ")

#Extract the first character from each string
#and concatenate them
initials = first[0] + middle[0] + lastname[0]

#Display the full name
print("Your initials are", initials)
```



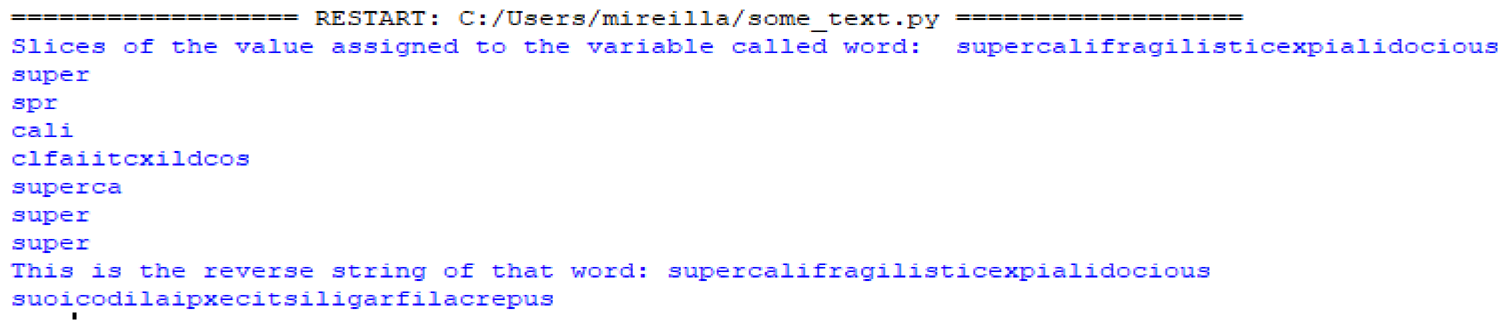
```
sentence = "Hello World"

print (sentence[0])         #get the first character
print (sentence[0:1])       #get the first character (same as above)
print (sentence[0:3])       #get the first three char
print (sentence[:3])        #get the first three char
print (sentence[-3:])       #get the last three char
print (sentence[3:])        #get all but the three first char
print (sentence[:-3])       #get all but the three last character
```

Working with strings

Slicing: taking a small piece of something bigger



```
word = "supercalifragilisticexpialidocious"

#To slice this word, the statement will be in the following format:

#variable[start:end:step]

print("Slices of the value assigned to the variable called word: ", word)

print(word[0:5:1])
print(word[0:5:2])
print(word[5:9:1])
print(word[5::2])
print(word[:7])
print(word[0:5:1])
print(word[:5])

#Reverse string

print("This is the reverse string of that word:", word)
print(word[::-1])
```



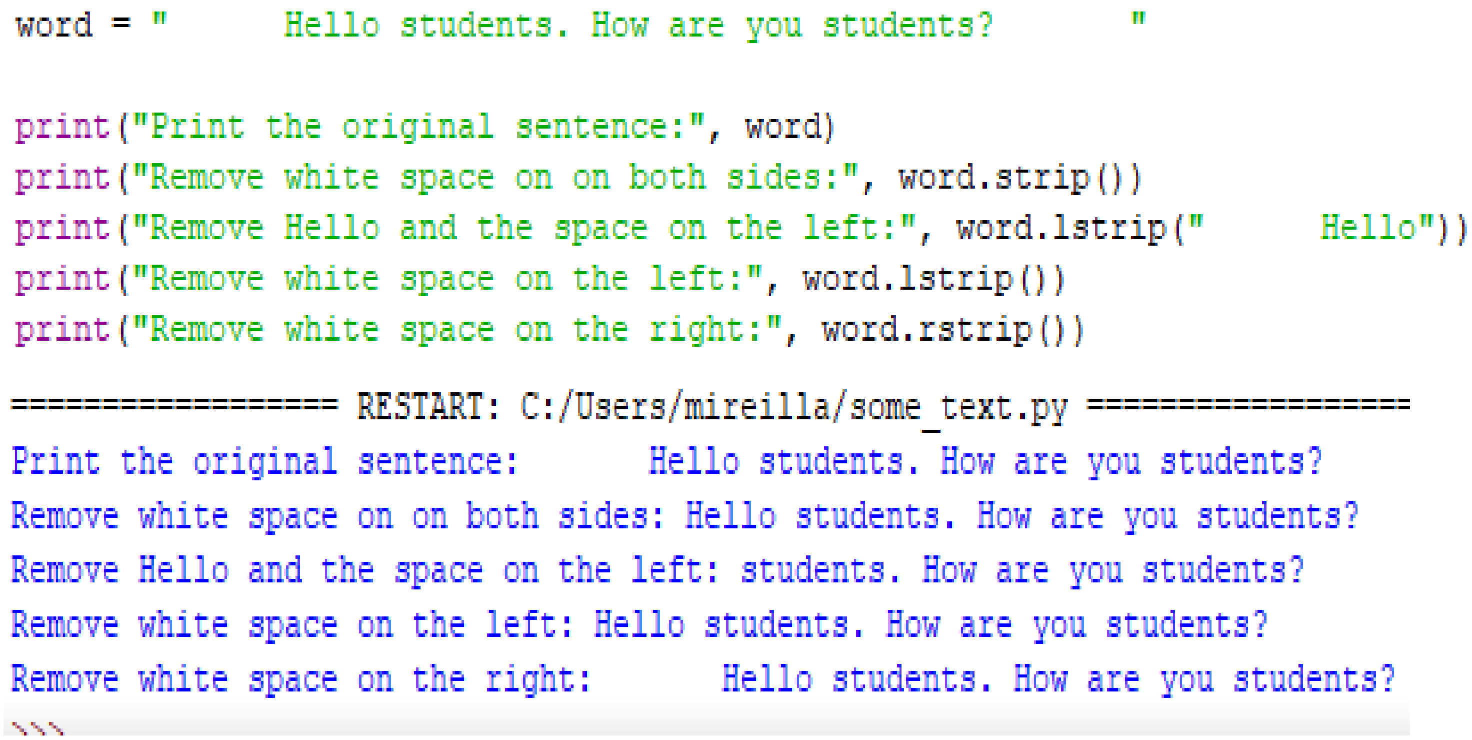
```
================== RESTART: C:/Users/mireilla/some_text.py ==================
Slices of the value assigned to the variable called word:  supercalifragilisticexpialidocious
super
spr
cali
clfaiitcxildcos
superca
super
super
This is the reverse string of that word: supercalifragilisticexpialidocious
suoicodilaipxecitsiligarfilacrepus
```

Working with strings

- Strip Strings:

Python strings have the strip(), lstrip(), rstrip() methods for removing any character from both end of the string.

If the character to be removed are not specified then white-space will be removed.



```
word = "      Hello students. How are you students?       "

print("Print the original sentence:", word)
print("Remove white space on on both sides:", word.strip())
print("Remove Hello and the space on the left:", word.lstrip("      Hello"))
print("Remove white space on the left:", word.lstrip())
print("Remove white space on the right:", word.rstrip())
```

```
================== RESTART: C:/Users/mireilla/some_text.py ==================
Print the original sentence:       Hello students. How are you students?
Remove white space on on both sides: Hello students. How are you students?
Remove Hello and the space on the left: students. How are you students?
Remove white space on the left: Hello students. How are you students?
Remove white space on the right:       Hello students. How are you students?
>>>
```

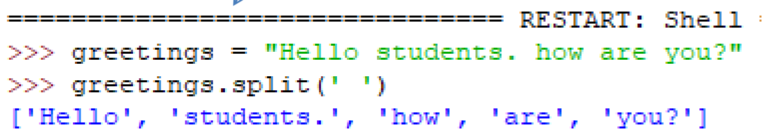
Working with strings

- Split Strings:

Split() method break the strings into a number of strings depending on the specified separator.

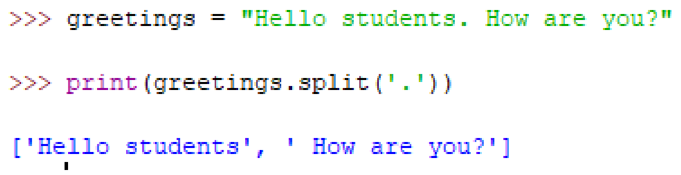
- Str.split(separator, maxsplit)

Example 1: The separator is white space ‵ ′. We get four strings.



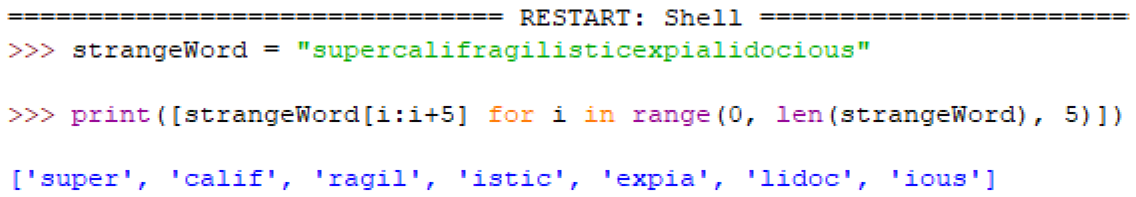
```
================================== RESTART: Shell
>>> greetings = "Hello students. how are you?"
>>> greetings.split(' ')
['Hello', 'students.', 'how', 'are', 'you?']
```

Example 2: The separator is the full stop ‵. ′. We get two strings.

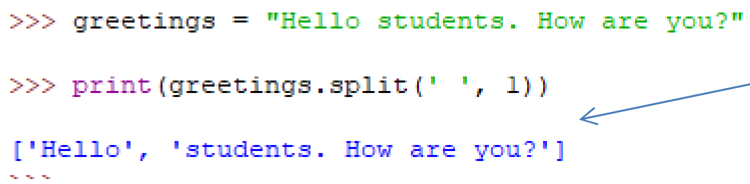


```
>>> greetings = "Hello students. How are you?"

>>> print(greetings.split('.'))

['Hello students', ' How are you?']
```

Example 3: we split after 5 characters. We get 7 strings.



```
================================== RESTART: Shell ========================
>>> strangeWord = "supercalifragilisticexpialidocious"

>>> print([strangeWord[i:i+5] for i in range(0, len(strangeWord), 5)])

['super', 'calif', 'ragil', 'istic', 'expia', 'lidoc', 'ious']
```



```
>>> greetings = "Hello students. How are you?"

>>> print(greetings.split(' ', 1))

['Hello', 'students. How are you?']
```

In this example, separator is white space‵ ′. However, we used maxsplit (1) which tells to the maximum number of times we want to split the string. We get two strings.

If we don’t use a maxsplit number, there is no limit to the number of splits performed like in example 1.

Working with strings

Lower, Upper case, Indentation

- Python is **case sensitive.** `text = str.lower(text)` changes the text to lower case. As Python is case sensitive, this changes the data input by the user into lower case so it is easier to check.

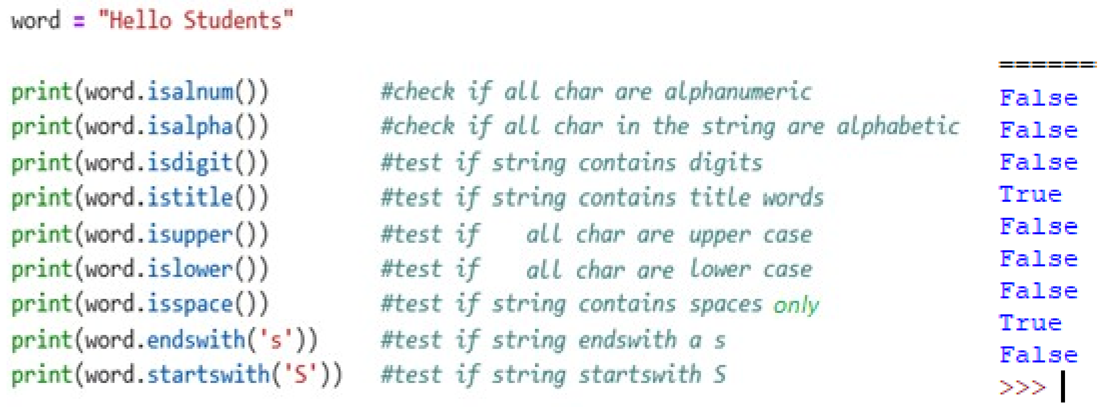
- `text = str.upper(text)` changes the text in upper case.
- `text = str.title(text)` transforms the text in a title.

- **Indentation** is important: It shows the lines that are dependent on others.

Working with strings

- **Testing**

A string can be tested for truth value. The return type will be in Boolean value (True or False)



```
word = "Hello Students"

print(word.isalnum())        #check if all char are alphanumeric
print(word.isalpha())        #check if all char in the string are alphabetic
print(word.isdigit())        #test if string contains digits
print(word.istitle())        #test if string contains title words
print(word.isupper())        #test if   all char are upper case
print(word.islower())        #test if   all char are lower case
print(word.isspace())        #test if string contains spaces only
print(word.endswith('s'))    #test if string endswith a s
print(word.startswith('S'))  #test if string startswith S
```

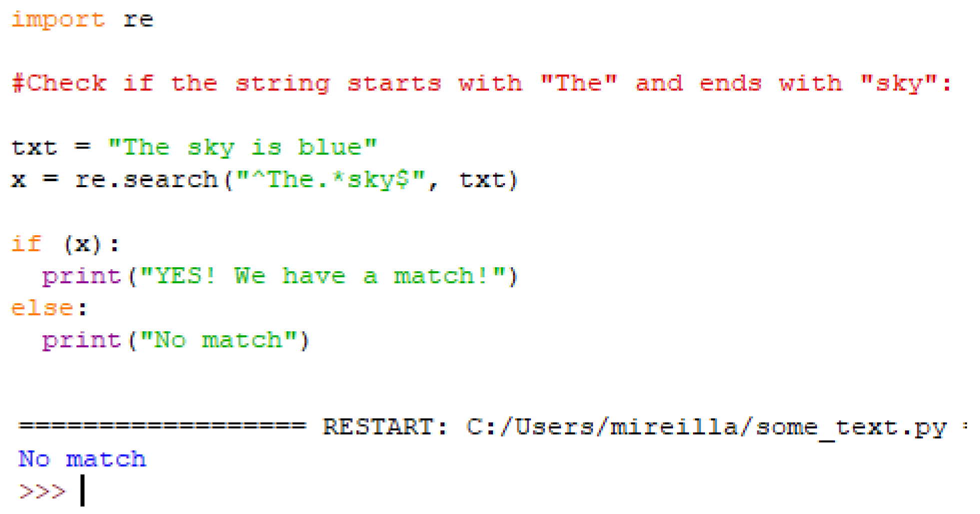
```
======
False
False
False
True
False
False
False
True
False
>>> |
```

- **Regular Expression (RegEx)**

is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

RegEx Module in Python is called `re`



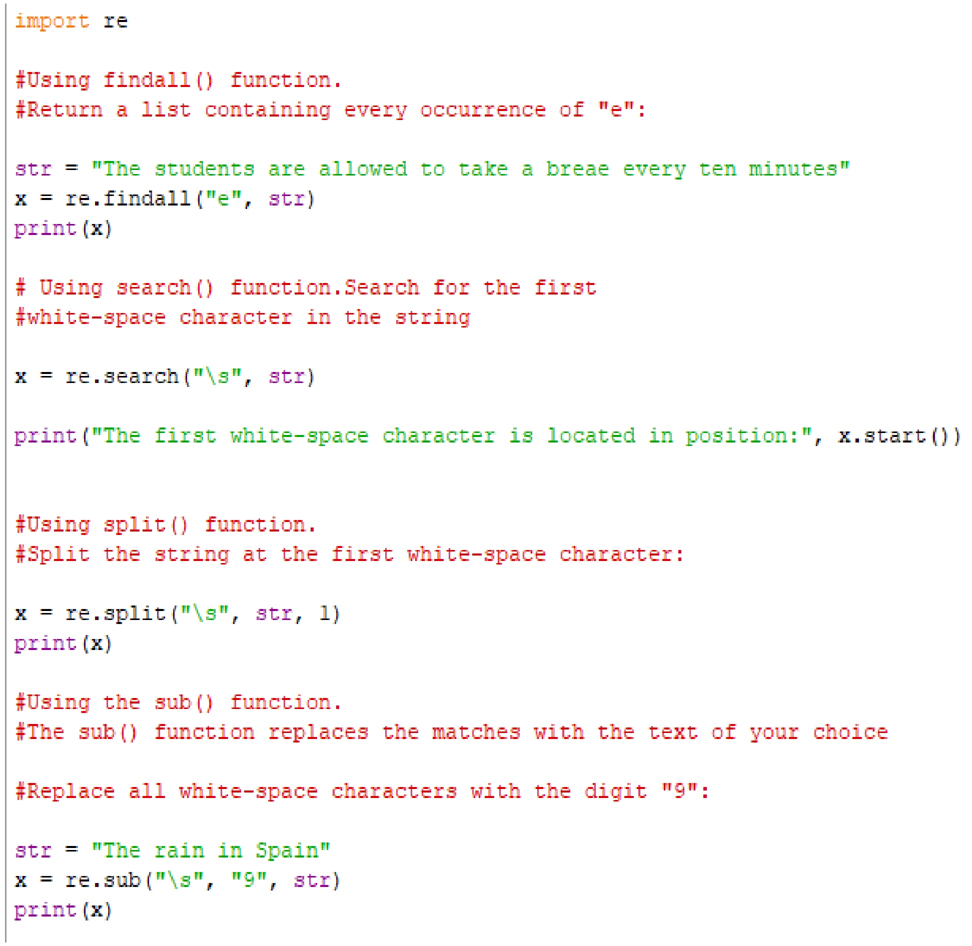
```
import re

#Check if the string starts with "The" and ends with "sky":

txt = "The sky is blue"
x = re.search("^The.*sky$", txt)

if (x):
  print("YES! We have a match!")
else:
  print("No match")


================== RESTART: C:/Users/mireilla/some_text.py
No match
>>> |
```

Working with strings

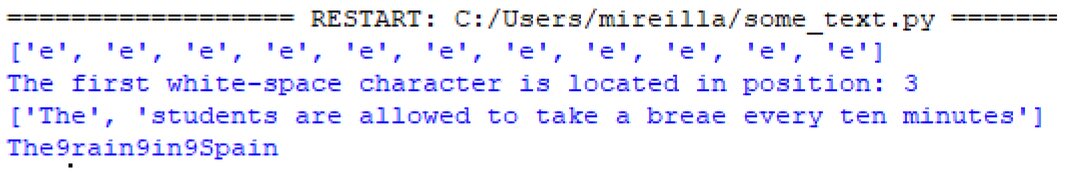
RegEx Functions

The re module offers a set of functions that allows us to search a string for a match

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string



```
import re

#Using findall() function.
#Return a list containing every occurrence of "e":

str = "The students are allowed to take a breae every ten minutes"
x = re.findall("e", str)
print(x)

# Using search() function.Search for the first
#white-space character in the string

x = re.search("\s", str)

print("The first white-space character is located in position:", x.start())


#Using split() function.
#Split the string at the first white-space character:

x = re.split("\s", str, 1)
print(x)

#Using the sub() function.
#The sub() function replaces the matches with the text of your choice

#Replace all white-space characters with the digit "9":

str = "The rain in Spain"
x = re.sub("\s", "9", str)
print(x)
```



```
================== RESTART: C:/Users/mireilla/some_text.py =======
['e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e']
The first white-space character is located in position: 3
['The', 'students are allowed to take a breae every ten minutes']
The9rain9in9Spain
```

Find more at: https://www.w3schools.com/python/python_regex.asp

Formatting

- `Integers`, `floating-point` numbers and `strings` can be formatted so that they occupy at least some minimum width.
- Python uses format specifiers, a sequence of characters that describe the formatting.

- Floating-point: `.7f` indicates that 7 digits should appear to the right of the decimal point.

- `%5.2f` format tells Python that the total of at least 5 spaces should be used to display a number, with 2 digits to the right of the decimal point.

- `“%d” % x` (value stored in x is formatted as a decimal (based 10) integer

- `“%f” % y` (value is stored in y is formatted as a floating point number)