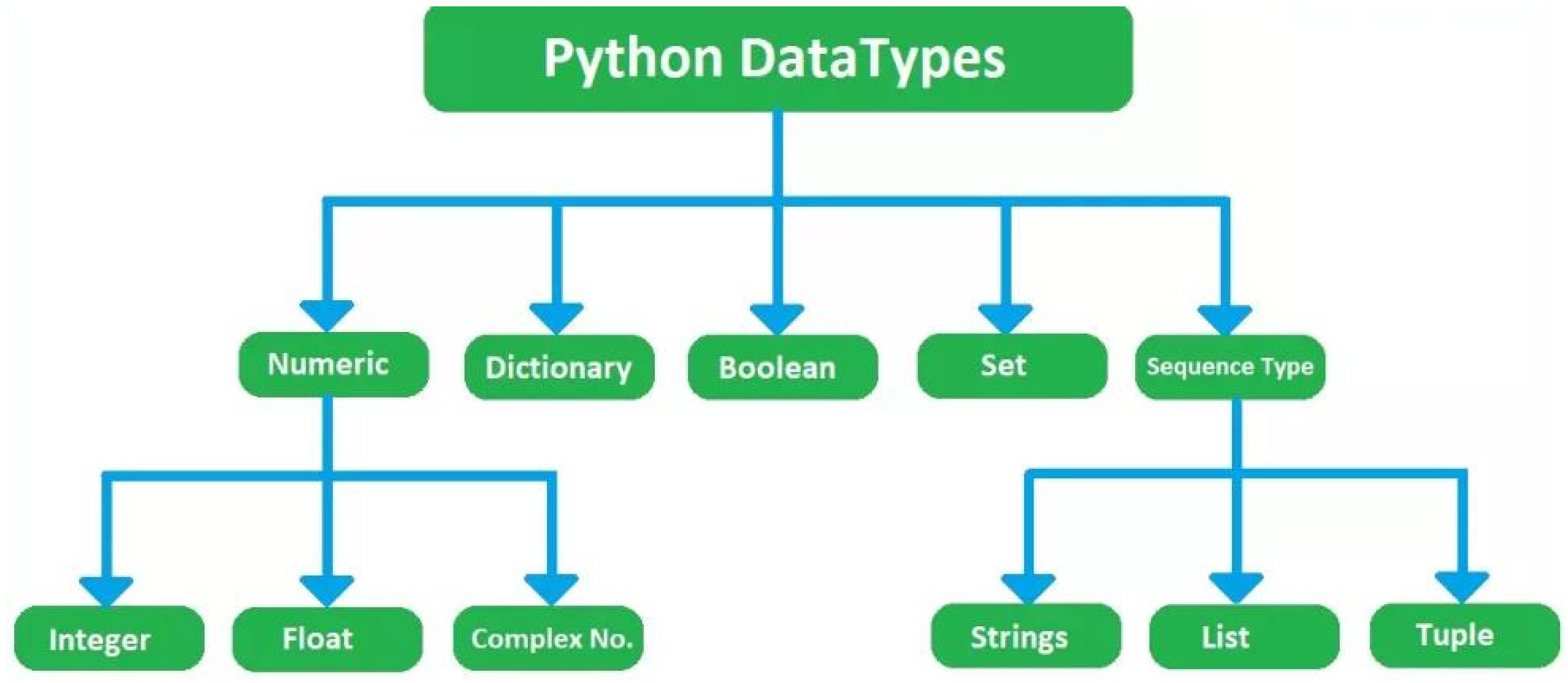
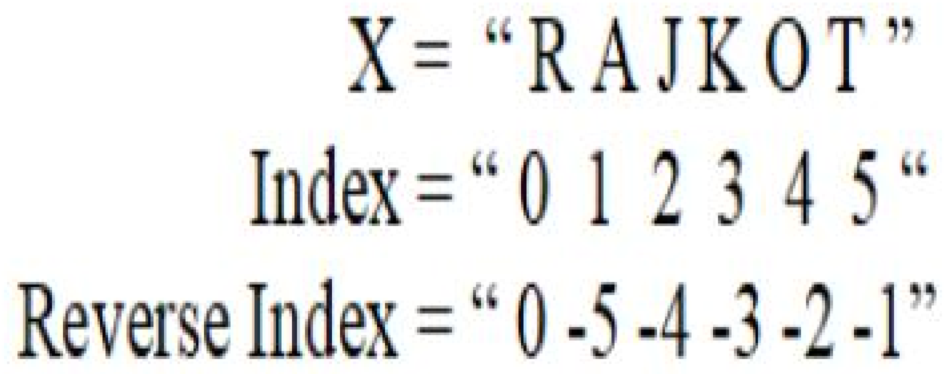
Data Types


Python DataTypes
Numeric
Dictionary
Boolean
Set
Sequence Type
Integer
Float
Complex No.
Strings
List
Tuple

String in Python

- String is **Ordered Sequence of Characters** such as “Rajkot”, ”City”, etc..
- Strings are **arrays of bytes** representing **Unicode** characters.
- String Can be represented as single , double quotes.
- String in python is **immutable**



X = “R A J K O T ”

Index = “ 0 1 2 3 4 5 “

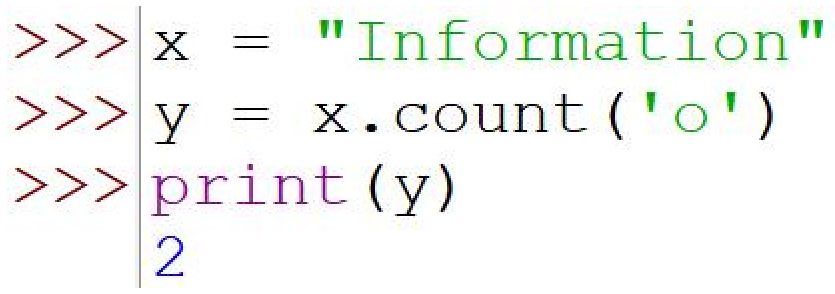
Reverse Index = “ 0 -5 -4 -3 -2 -1”

String functions in python

- Python has lots of built - in methods that you can use on strings, we are going to cover some frequently used methods for string like.

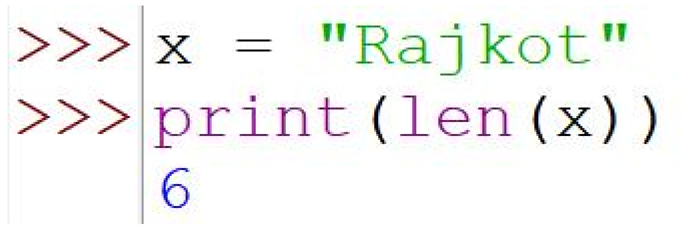
1) len()
2) Count()
3) title(), lower(), upper()
4) istitle(), islower(), isupper()
5) find(), rfind(), replace()
6) index(), rindex()
7) isalpha(), isalnum(), isdecimal(), isdigit()
8) strip(), lstrip(), rstrip()

count() method will returns the number of times a specified values occurs in a string



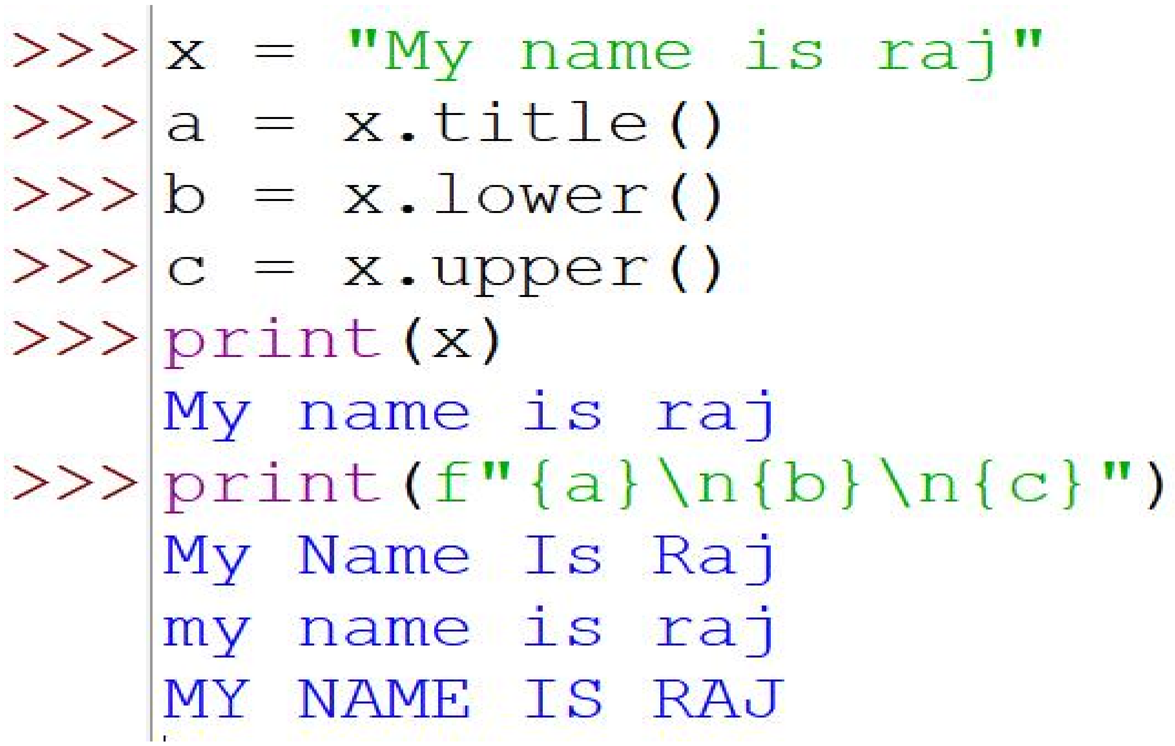
```
>>> x = "Information"
>>> y = x.count('o')
>>> print(y)
    2
```

len() is not the method of the string but can be used to get the length of the string



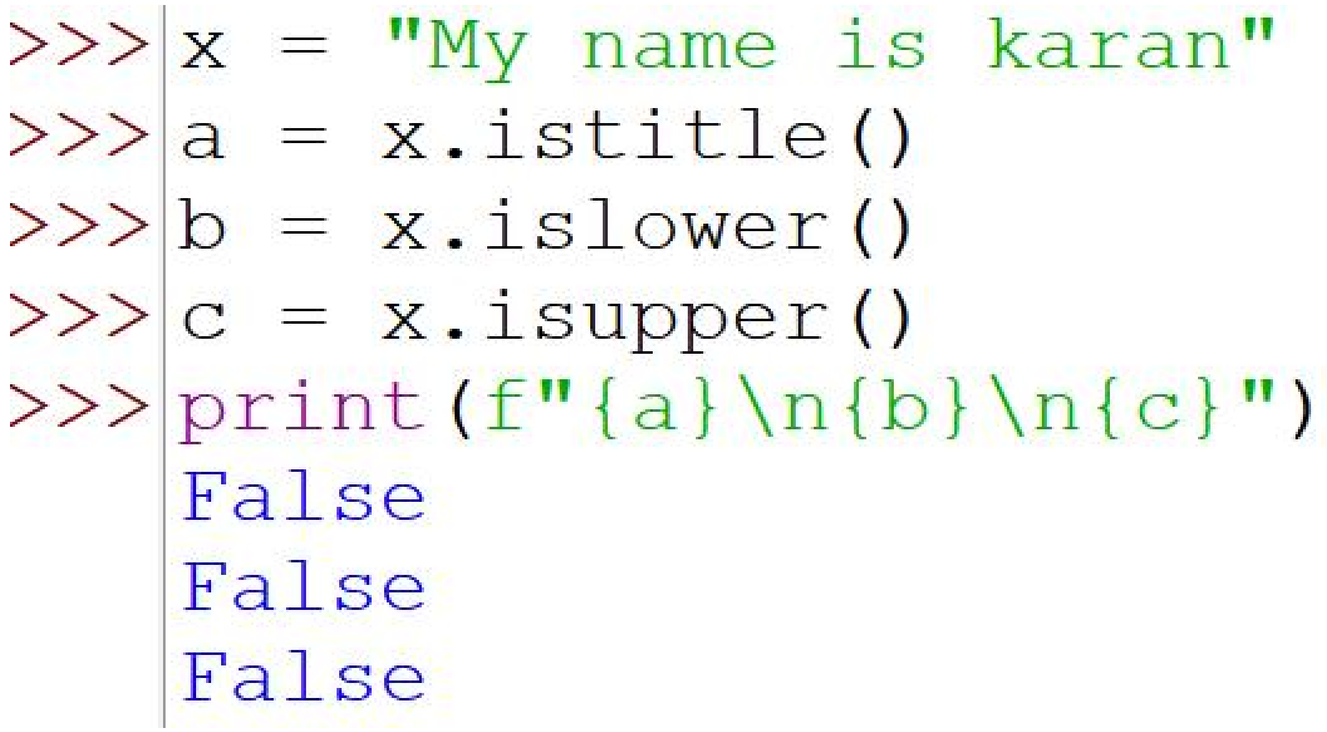
```
>>> x = "Rajkot"
>>> print(len(x))
    6
```

title(), lower(), upper() will returns capitalized, lower case, and upper case string respectively



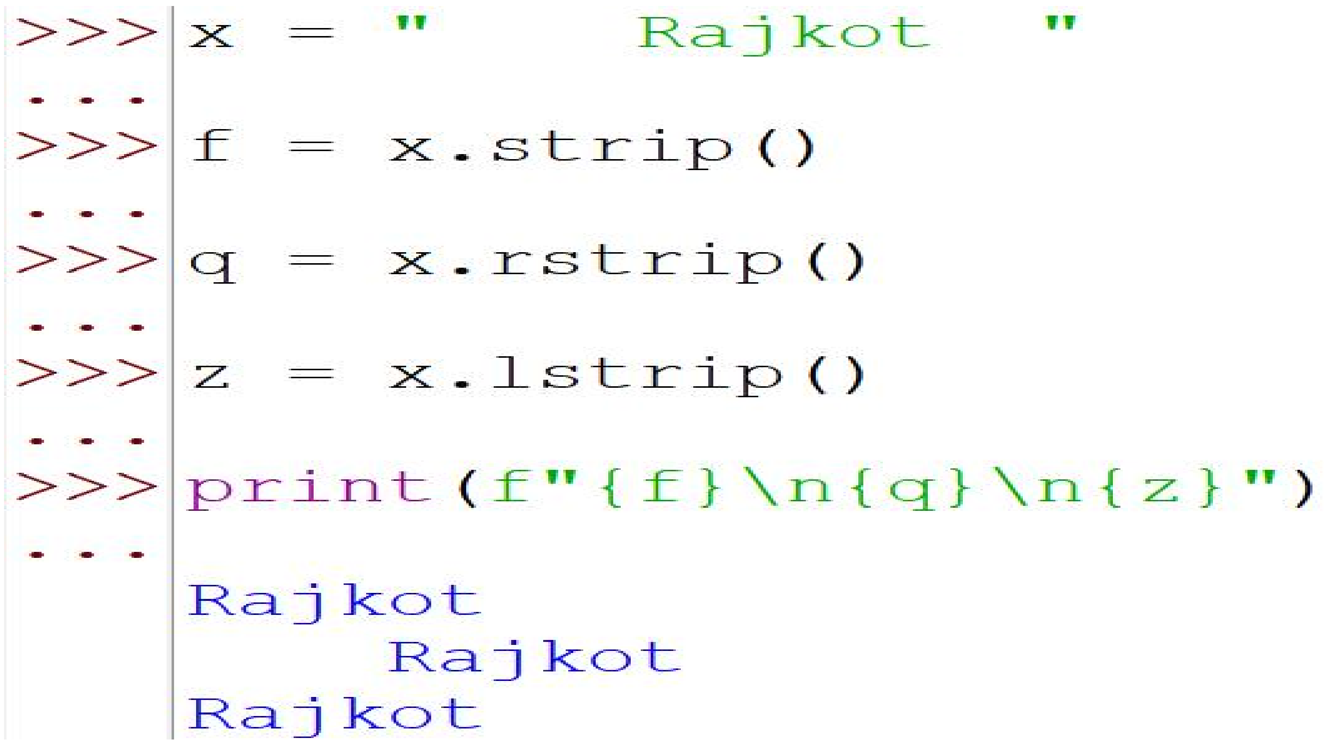
```
>>> x = "My name is raj"
>>> a = x.title()
>>> b = x.lower()
>>> c = x.upper()
>>> print(x)
    My name is raj
>>> print(f"{a}\n{b}\n{c}")
    My Name Is Raj
    my name is raj
    MY NAME IS RAJ
```

istitle(), islower(), isupper() will returns True if the given string is capitalized, lower case and upper case respectively.



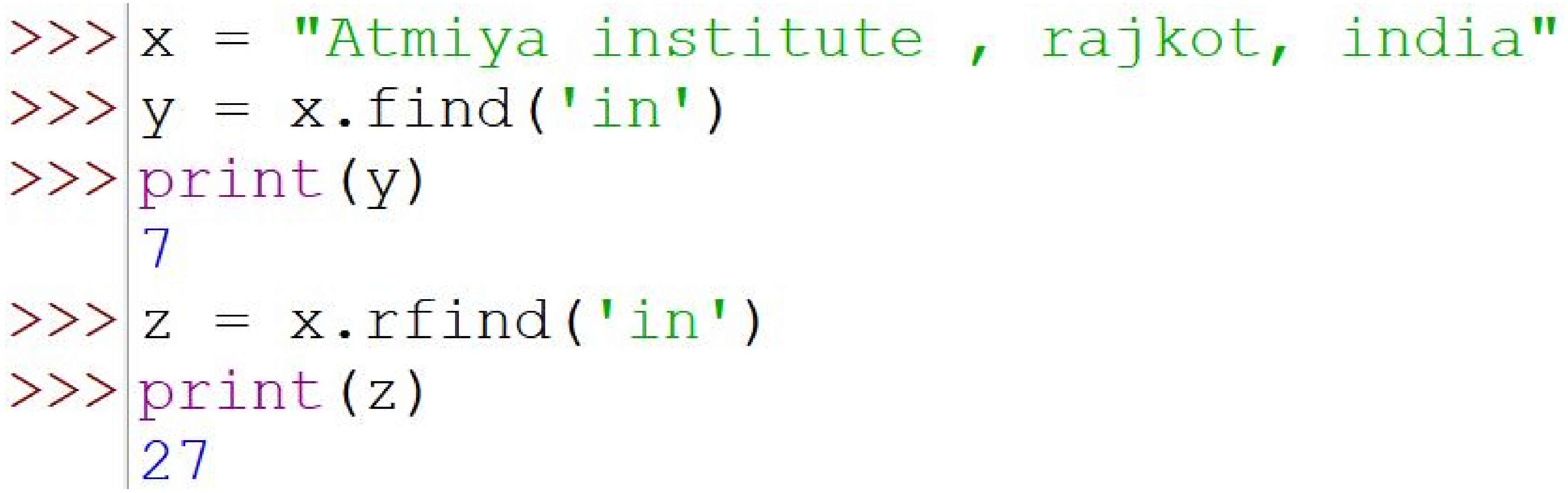
```
>>> x = "My name is karan"
>>> a = x.istitle()
>>> b = x.islower()
>>> c = x.isupper()
>>> print(f"{a}\n{b}\n{c}")
    False
    False
    False
```

strip() method will remove whitespace from both side of the string and returns the string.
rstrip() and lstrip() will remove whitespace from left and right side respectively



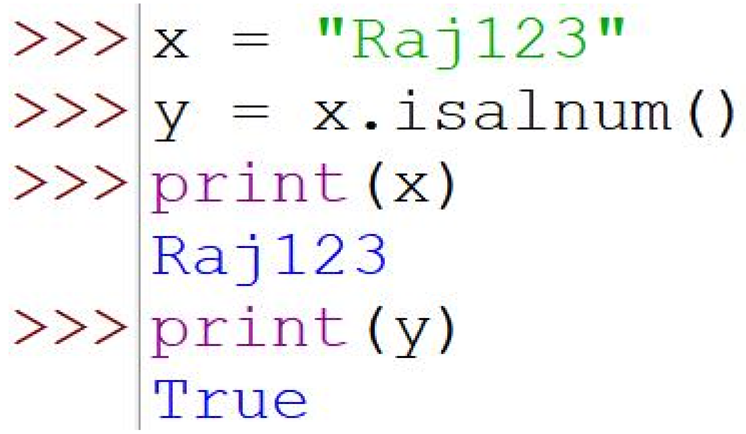
```
>>> x = "    Rajkot   "
...
>>> f = x.strip()
...
>>> q = x.rstrip()
...
>>> z = x.lstrip()
...
>>> print(f"{f}\n{q}\n{z}")
...
Rajkot
    Rajkot
Rajkot
```

find() method will search the string and returns the index at which they find the specified.
rfind() will start search of the string from the reverse and find the specified values



```
>>> x = "Atmiya institute , rajkot, india"
>>> y = x.find('in')
>>> print(y)
7
>>> z = x.rfind('in')
>>> print(z)
27
```

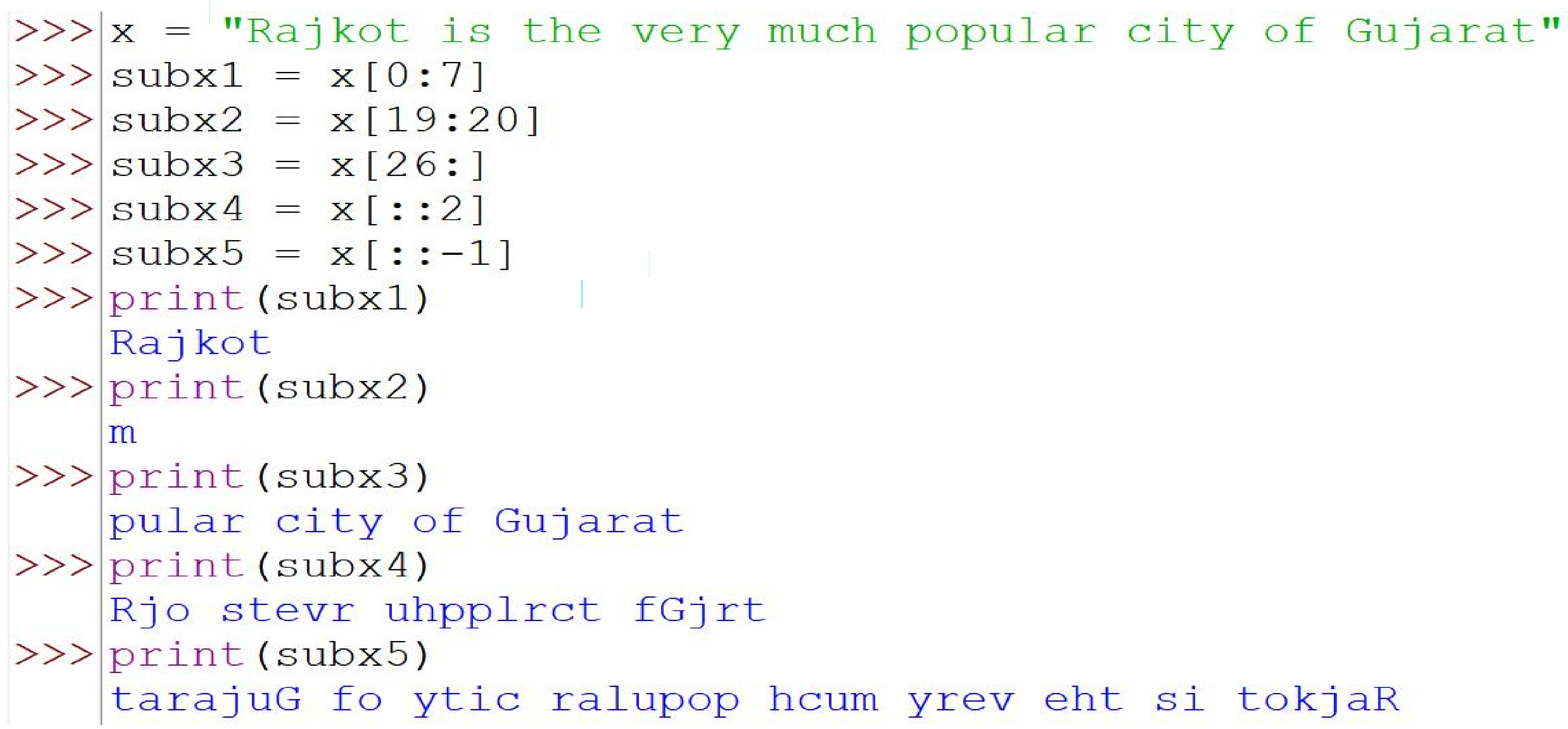
isalnum() method will return true if all the characters in the string are alphanumeric (i.e either alphabets or numeric).



```
>>> x = "Raj123"
>>> y = x.isalnum()
>>> print(x)
    Raj123
>>> print(y)
    True
```

String Slicing

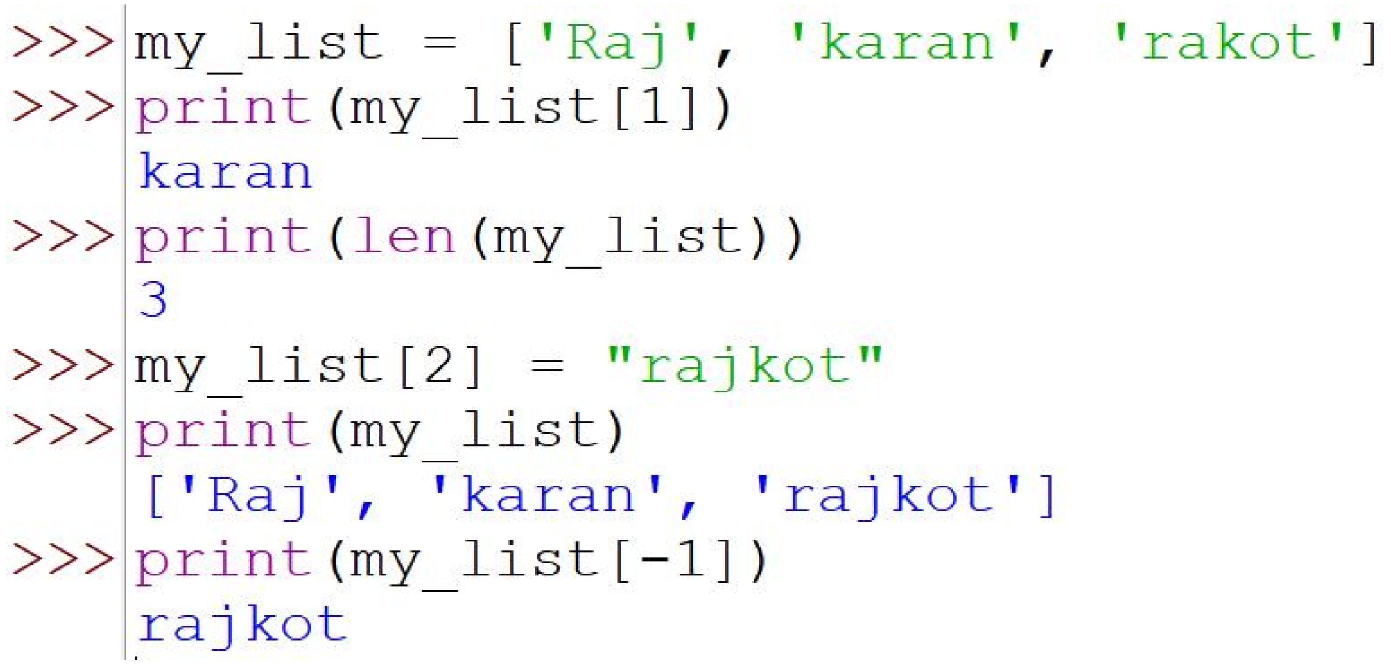
We can get the substring in python using string slicing, we can specify start index, end index(colon seperated) to slice the string



```
>>> x = "Rajkot is the very much popular city of Gujarat"
>>> subx1 = x[0:7]
>>> subx2 = x[19:20]
>>> subx3 = x[26:]
>>> subx4 = x[::2]
>>> subx5 = x[::-1]
>>> print(subx1)
    Rajkot
>>> print(subx2)
    m
>>> print(subx3)
    pular city of Gujarat
>>> print(subx4)
    Rjo stevr uhpplrct fGjrt
>>> print(subx5)
    tarajuG fo ytic ralupop hcum yrev eht si tokjaR
```

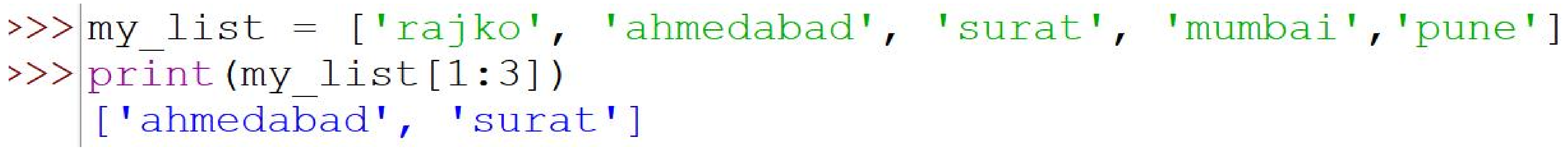
List

- List is mutable ordered sequence of objects, duplicate values are allowed iside list.
- List will be represented by square brackets [].
- It contain element of different type.
- Built in data structure, no import is needed



```
>>> my_list = ['Raj', 'karan', 'rakot']
>>> print(my_list[1])
karan
>>> print(len(my_list))
3
>>> my_list[2] = "rajkot"
>>> print(my_list)
['Raj', 'karan', 'rajkot']
>>> print(my_list[-1])
rajkot
```

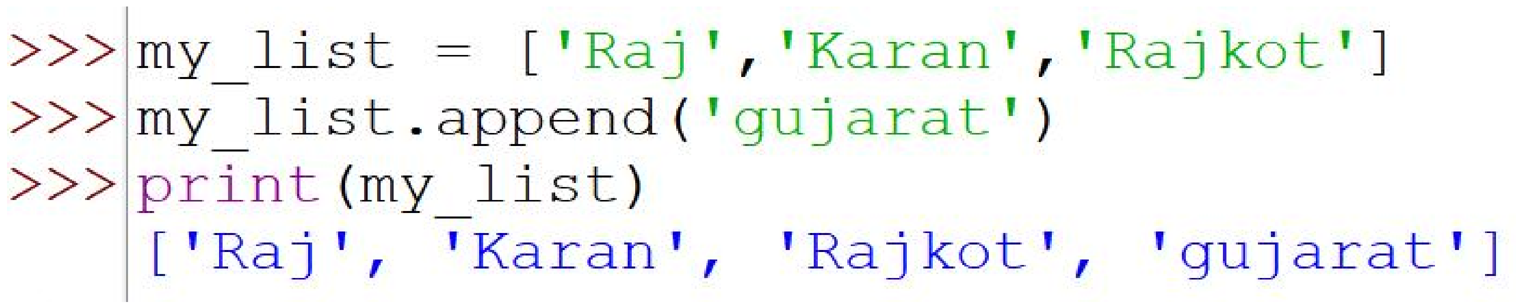
We can use slicing similar to string in order to get the sub list from the list



```
>>> my_list = ['rajko', 'ahmedabad', 'surat', 'mumbai','pune']
>>> print(my_list[1:3])
    ['ahmedabad', 'surat']
```

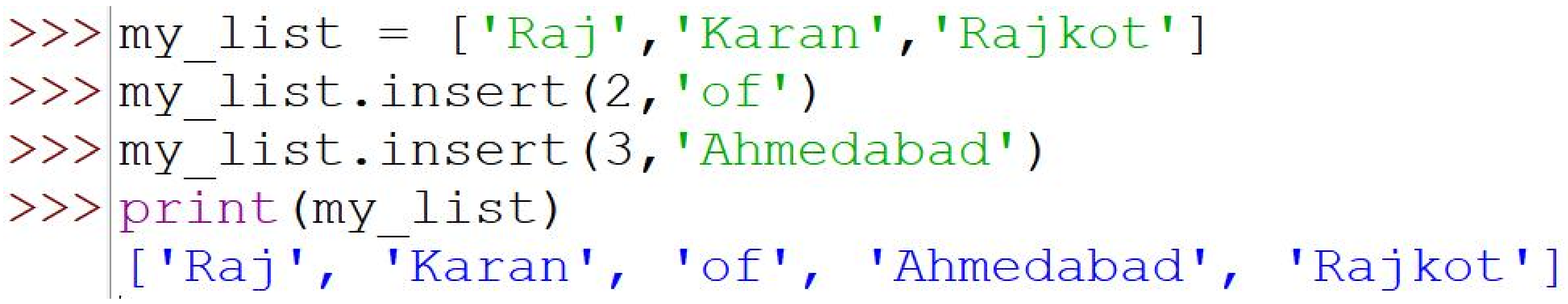
List Methods

append() method will add element at the end of the list.



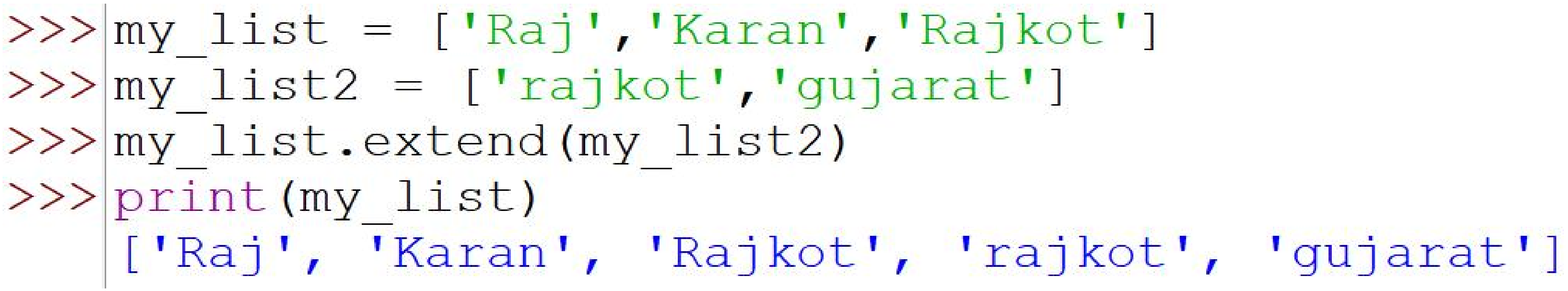
```
>>>my_list = ['Raj','Karan','Rajkot']
>>>my_list.append('gujarat')
>>>print(my_list)
   ['Raj', 'Karan', 'Rajkot', 'gujarat']
```

insert() method will add element at the specified index in the list.



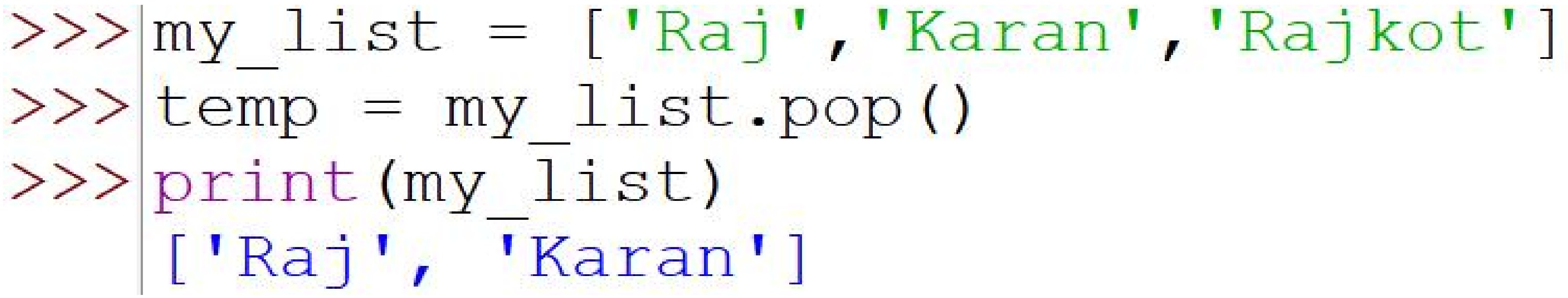
```
>>>my_list = ['Raj','Karan','Rajkot']
>>>my_list.insert(2,'of')
>>>my_list.insert(3,'Ahmedabad')
>>>print(my_list)
   ['Raj', 'Karan', 'of', 'Ahmedabad', 'Rajkot']
```

extend() method will add one data structure (List or any) to current list



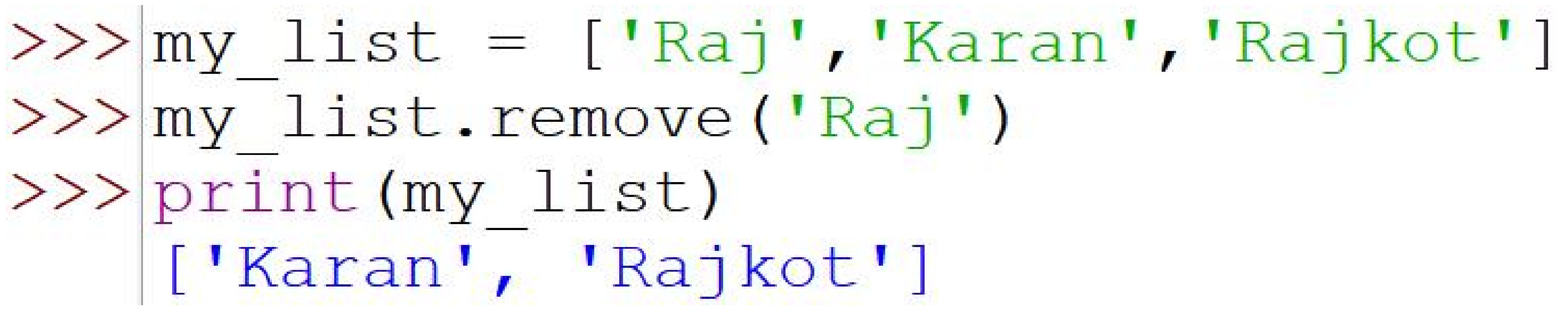
```
>>> my_list = ['Raj','Karan','Rajkot']
>>> my_list2 = ['rajkot','gujarat']
>>> my_list.extend(my_list2)
>>> print(my_list)
['Raj', 'Karan', 'Rajkot', 'rajkot', 'gujarat']
```

pop() method will remove the last element from the list and return it.



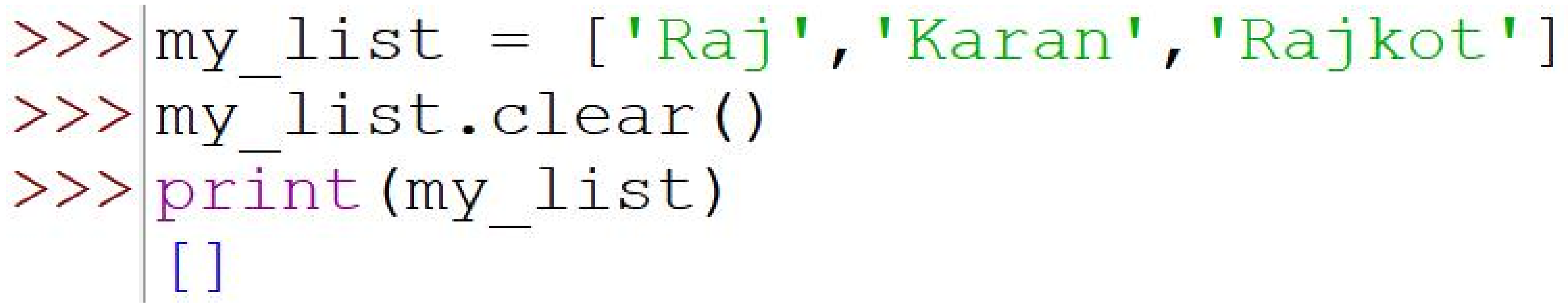
```
>>> my_list = ['Raj','Karan','Rajkot']
>>> temp = my_list.pop()
>>> print(my_list)
['Raj', 'Karan']
```

remove() method will remove first occurance of specified element



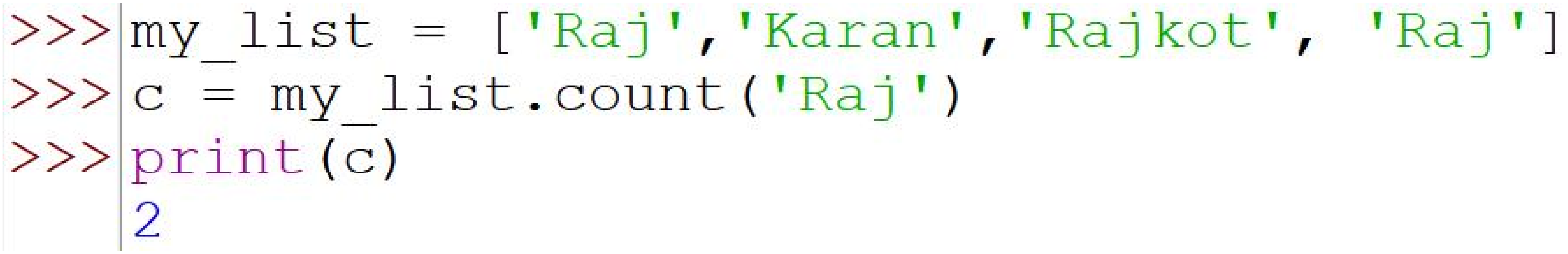
```
>>> my_list = ['Raj','Karan','Rajkot']
>>> my_list.remove('Raj')
>>> print(my_list)
    ['Karan', 'Rajkot']
```

clear() method will remove all the elements from the list



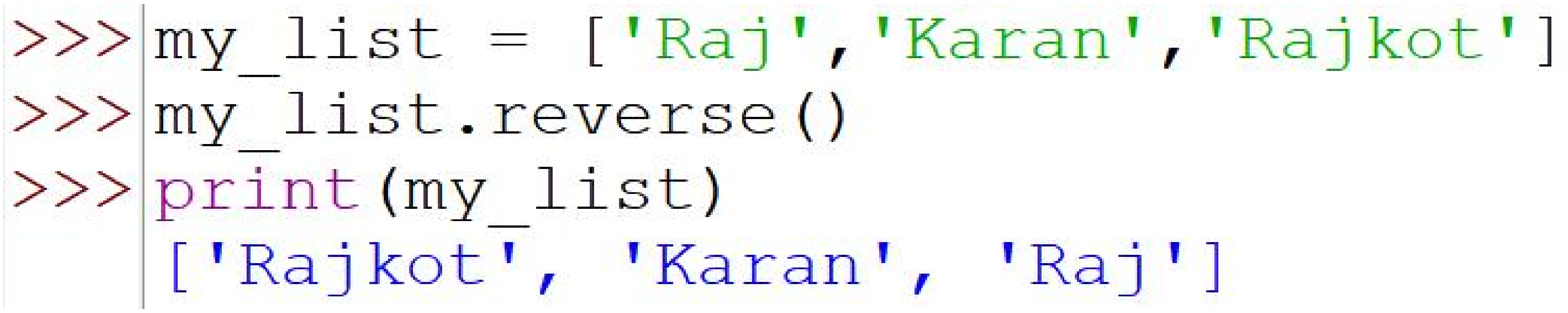
```
>>> my_list = ['Raj','Karan','Rajkot']
>>> my_list.clear()
>>> print(my_list)
    []
```

count() method will return the number of occurance of the specified elements



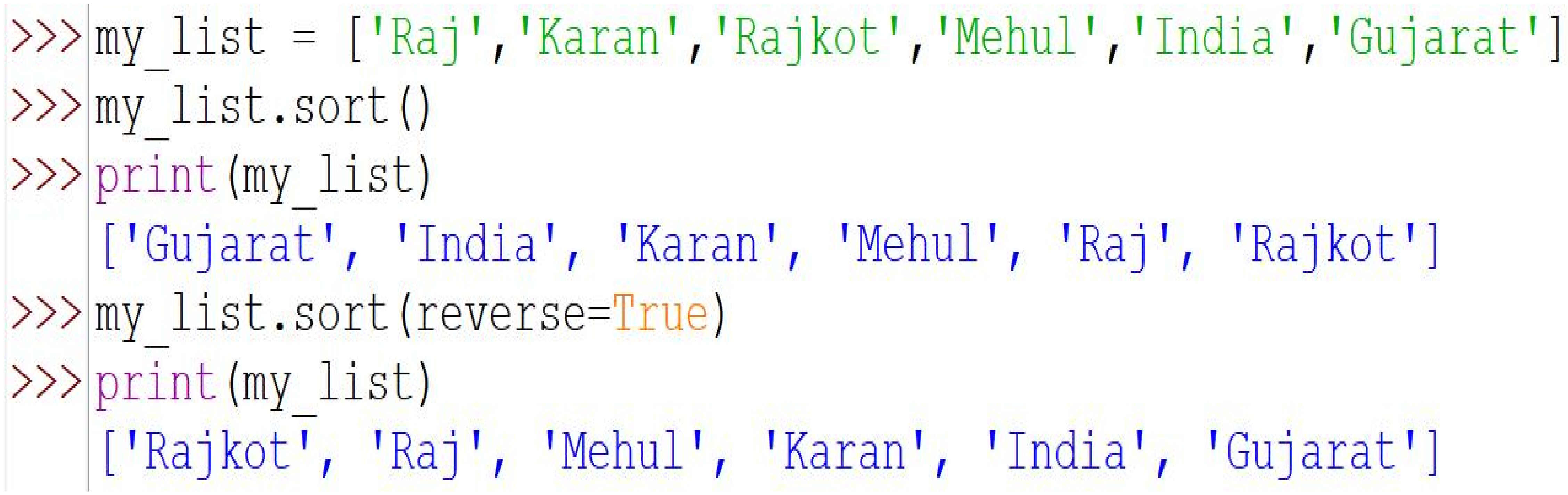
```
>>> my_list = ['Raj','Karan','Rajkot', 'Raj']
>>> c = my_list.count('Raj')
>>> print(c)
    2
```

reverse() method will reverse the element of the list



```
>>> my_list = ['Raj','Karan','Rajkot']
>>> my_list.reverse()
>>> print(my_list)
    ['Rajkot', 'Karan', 'Raj']
```

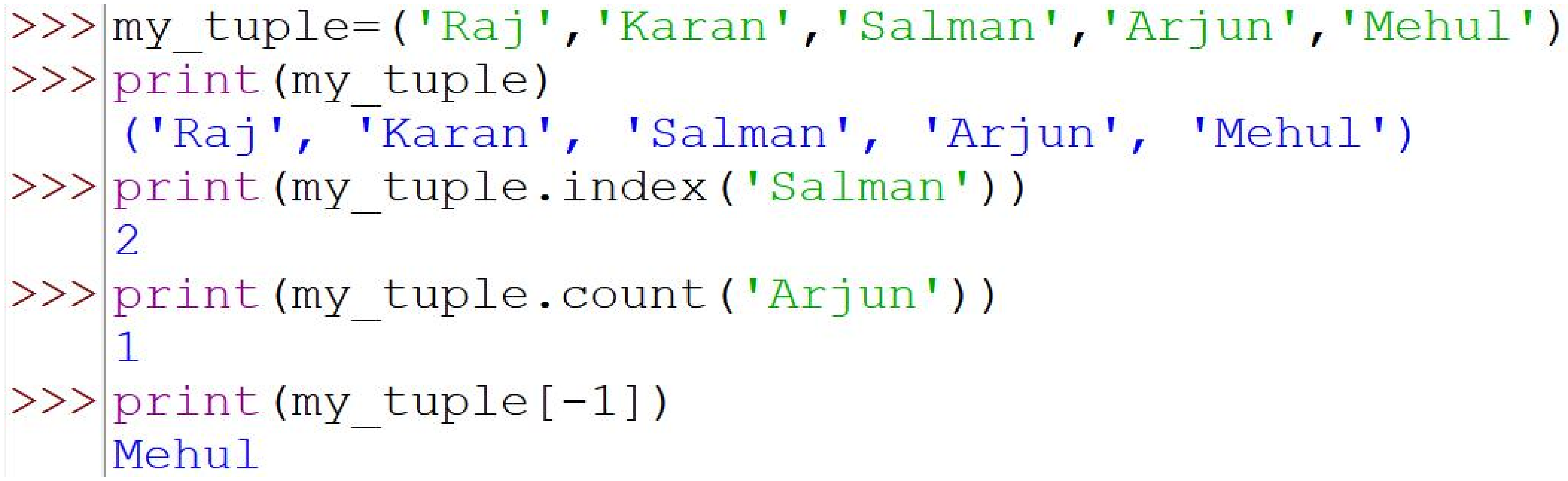
sort() method will sort the element in the list



```
>>> my_list = ['Raj','Karan','Rajkot','Mehul','India','Gujarat']
>>> my_list.sort()
>>> print(my_list)
['Gujarat', 'India', 'Karan', 'Mehul', 'Raj', 'Rajkot']
>>> my_list.sort(reverse=True)
>>> print(my_list)
['Rajkot', 'Raj', 'Mehul', 'Karan', 'India', 'Gujarat']
```

Tuple

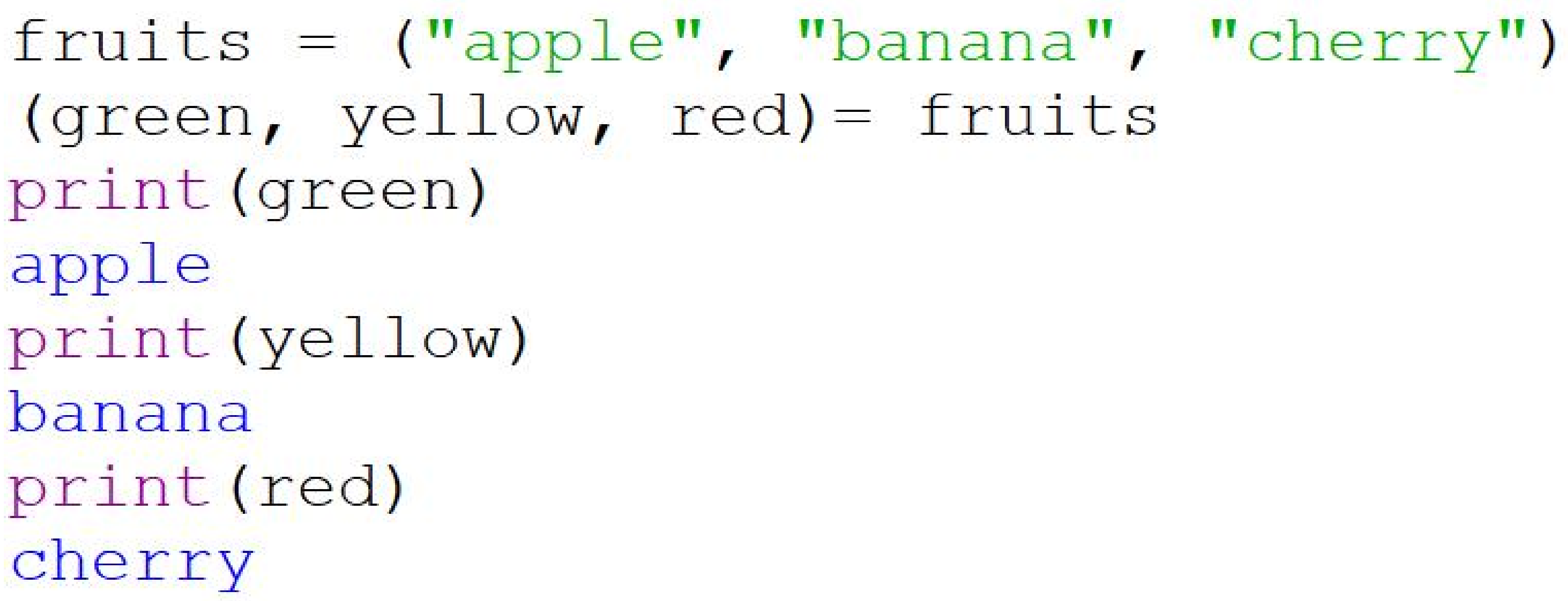
->Tuple is a immutable ordered sequence of objects, duplicate values are allowed inside list
-> Tuple will be represented by round brackets()
-> Tuple is similar to List but List is mutable wheras Tuple is immutable



```
>>> my_tuple=('Raj','Karan','Salman','Arjun','Mehul')
>>> print(my_tuple)
    ('Raj', 'Karan', 'Salman', 'Arjun', 'Mehul')
>>> print(my_tuple.index('Salman'))
    2
>>> print(my_tuple.count('Arjun'))
    1
>>> print(my_tuple[-1])
    Mehul
```

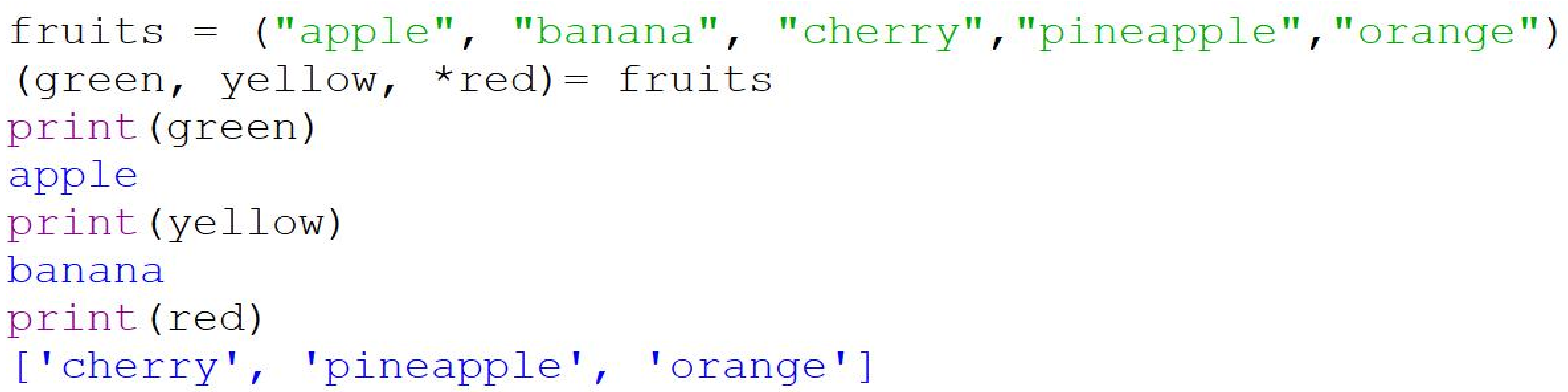
Unpacking Tuple

- When we create a tupe , we normally assign values to it. This is called the packing of tuple.
- But in Python , we are also allowed to extract the values back into variables. This is called “Unpacking”



```
fruits = ("apple", "banana", "cherry")
(green, yellow, red)= fruits
print(green)
apple
print(yellow)
banana
print(red)
cherry
```

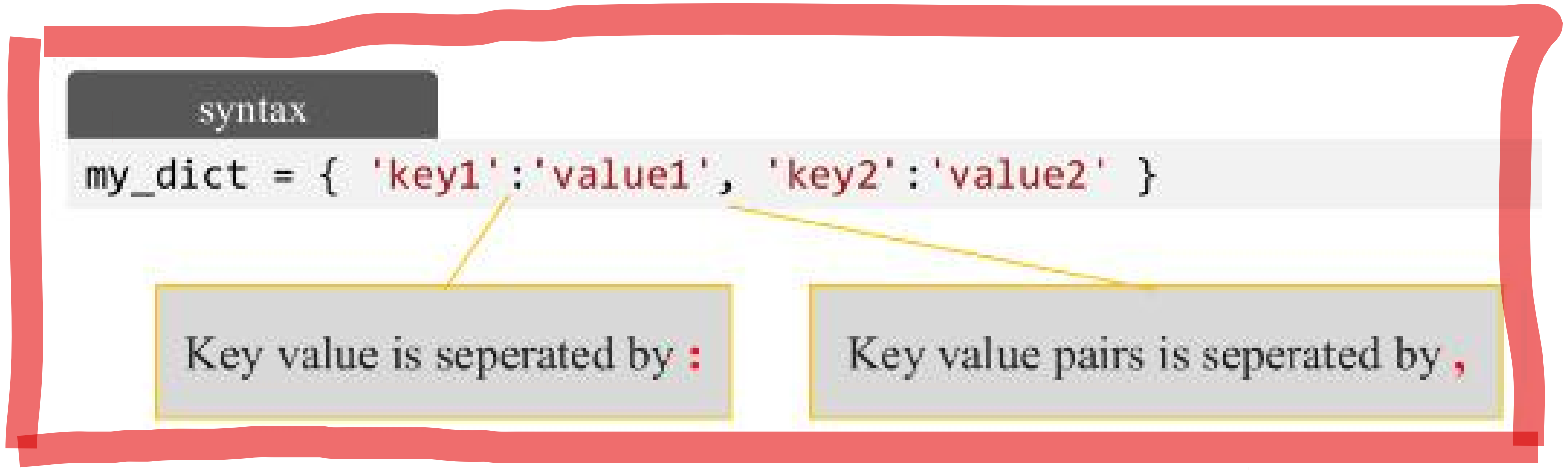
- **Using Asterisk***
- If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list:



```
fruits = ("apple", "banana", "cherry","pineapple","orange")
(green, yellow, *red)= fruits
print(green)
apple
print(yellow)
banana
print(red)
['cherry', 'pineapple', 'orange']
```

Dictionary

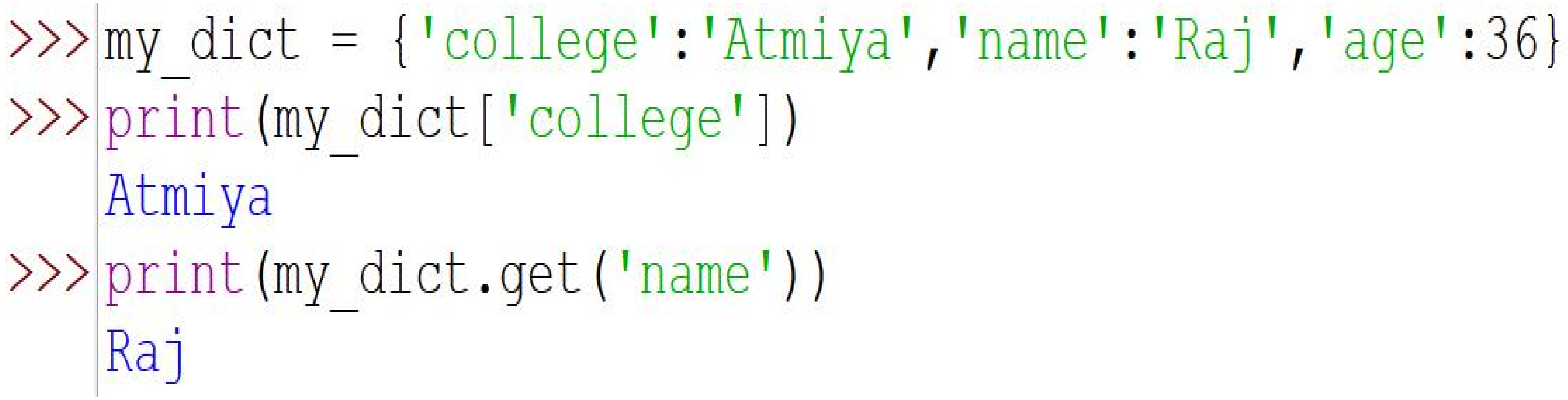
- Dictionary is a unordered collection of key value pair.
- Dictionary will be represented by curly brackets { }.
- Dictionary cannot have two items with the same key
- Dictionary is mutable.



```
my_dict = { 'key1':'value1', 'key2':'value2' }
```

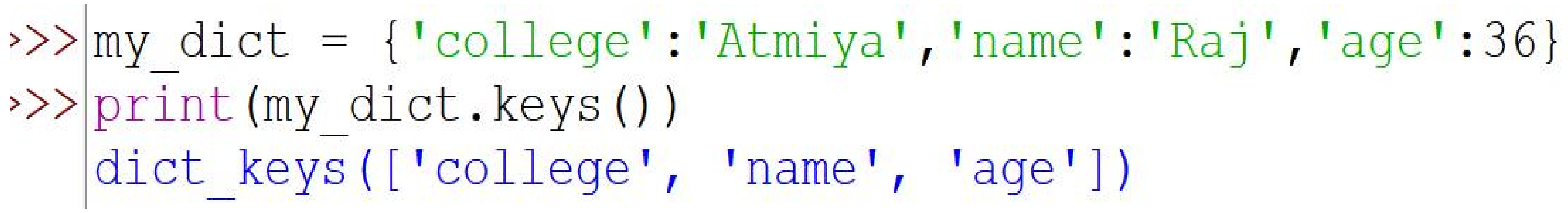
Key value is seperated by :
Key value pairs is seperated by ,

values can be accessed using key inside square brackets as well as using get() method



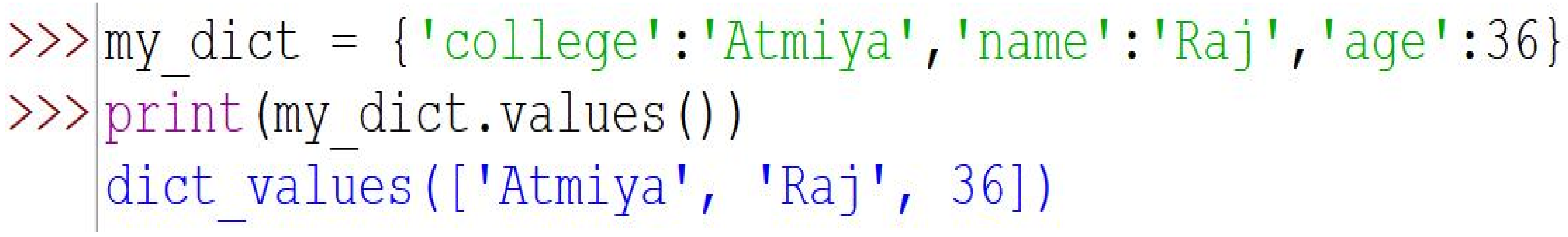
```
>>> my_dict = {'college':'Atmiya','name':'Raj','age':36}
>>> print(my_dict['college'])
Atmiya
>>> print(my_dict.get('name'))
Raj
```

keys() methods will return of all the keys associated with the Dictionary



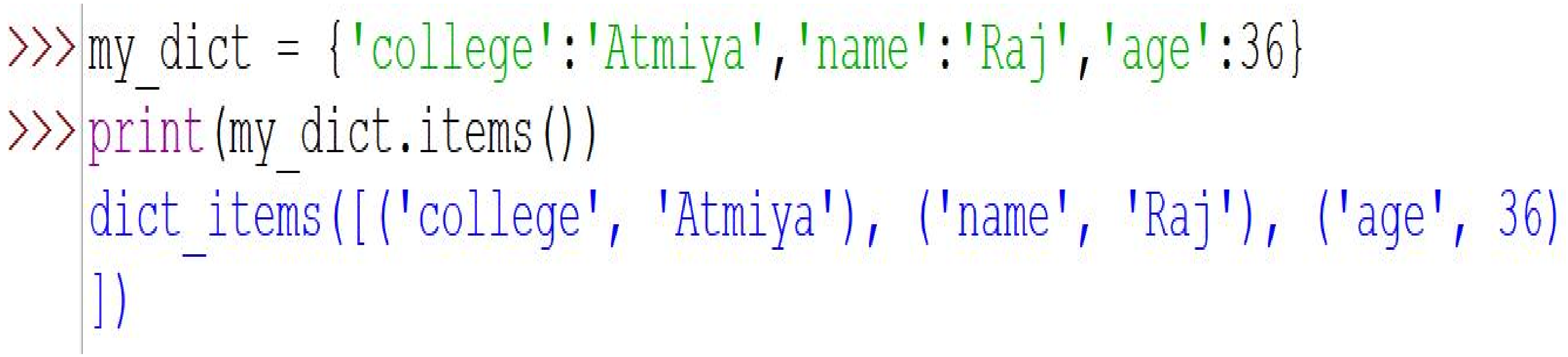
```
>>> my_dict = {'college':'Atmiya','name':'Raj','age':36}
>>> print(my_dict.keys())
    dict_keys(['college', 'name', 'age'])
```

values() methods will return of all the values associated with the Dictionary



```
>>> my_dict = {'college':'Atmiya','name':'Raj','age':36}
>>> print(my_dict.values())
    dict_values(['Atmiya', 'Raj', 36])
```

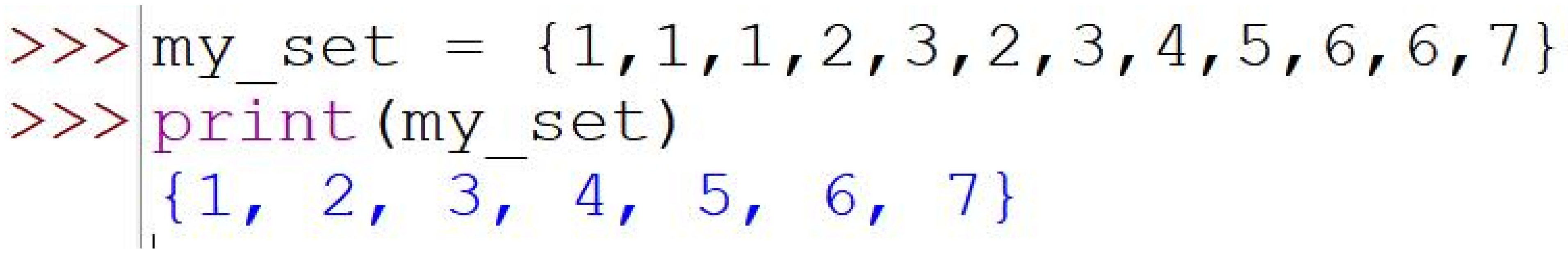
items() methods will return list of tuples for each key value pair associated with the dictionary



```
>>> my_dict = {'college':'Atmiya','name':'Raj','age':36}
>>> print(my_dict.items())
dict_items([('college', 'Atmiya'), ('name', 'Raj'), ('age', 36)
])
```

Set

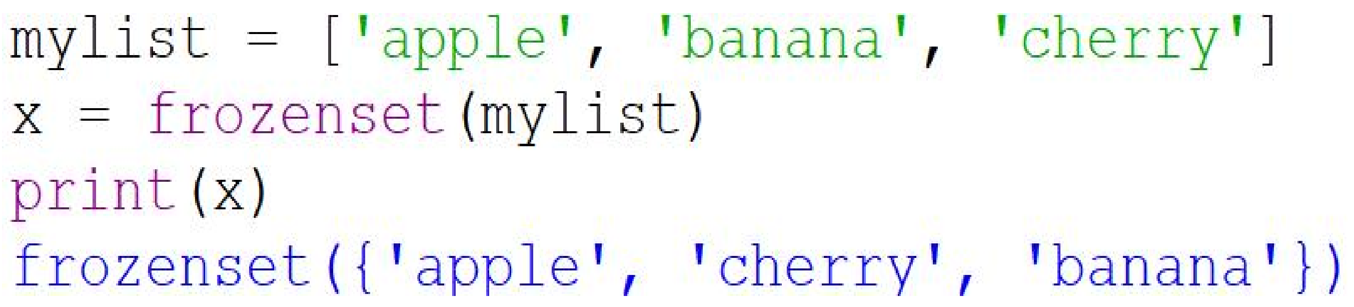
- Set is unordered collection of unique objects.
- Set will be represented by curly brackets { }
- Set has many in-built methods such as add(), clear(), copy(), pop(), remove() etc..
- Only difference between Set and List is that Set will have only unique elements and List can have duplicate elements.



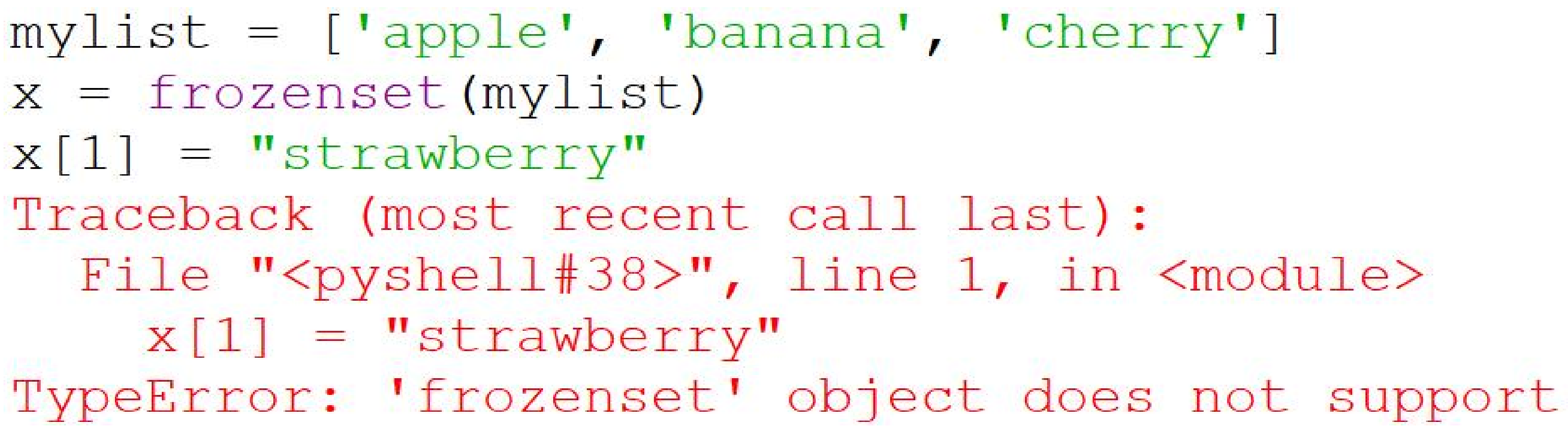
```
>>> my_set = {1,1,1,2,3,2,3,4,5,6,6,7}
>>> print(my_set)
{1, 2, 3, 4, 5, 6, 7}
```

Python frozenset()

- The frozensen() function returns an unchangable frozenset object (which is like a set object , only unchangable)
- Try to change the value of frozenset will cause the error



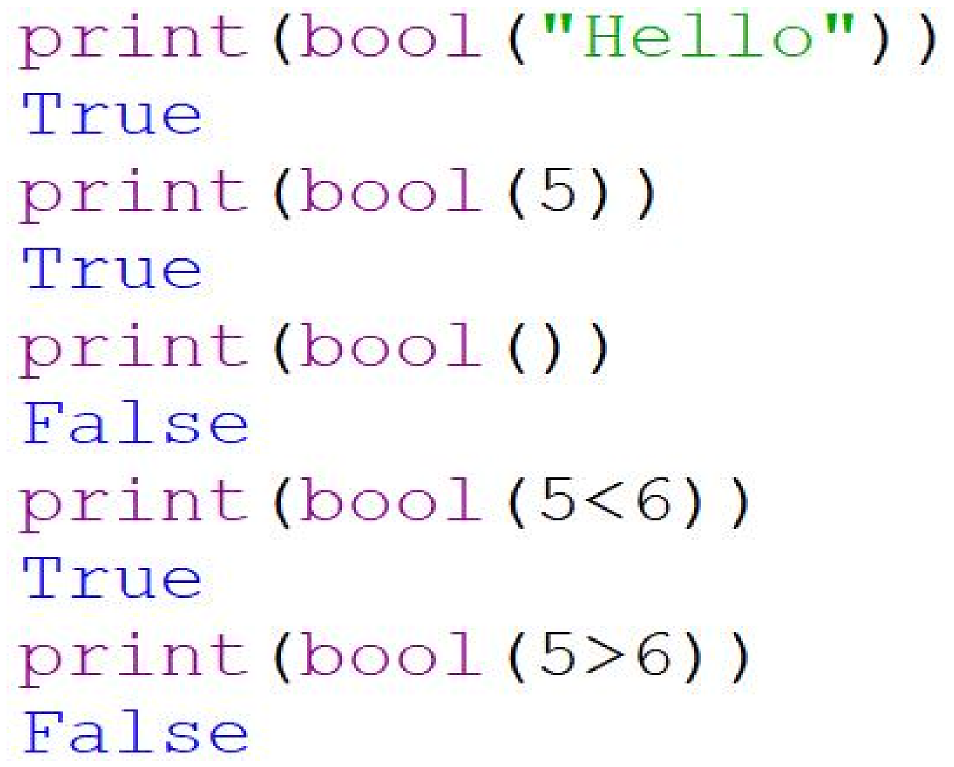
```
mylist = ['apple', 'banana', 'cherry']
x = frozenset(mylist)
print(x)
frozenset({'apple', 'cherry', 'banana'})
```



```
mylist = ['apple', 'banana', 'cherry']
x = frozenset(mylist)
x[1] = "strawberry"
Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    x[1] = "strawberry"
TypeError: 'frozenset' object does not support
```

Boolean

- Booleans represent one of two values: True or False.
- The bool() function allows you to evaluate any values, and gives you True or False in return



```
print(bool("Hello"))
True
print(bool(5))
True
print(bool())
False
print(bool(5<6))
True
print(bool(5>6))
False
```

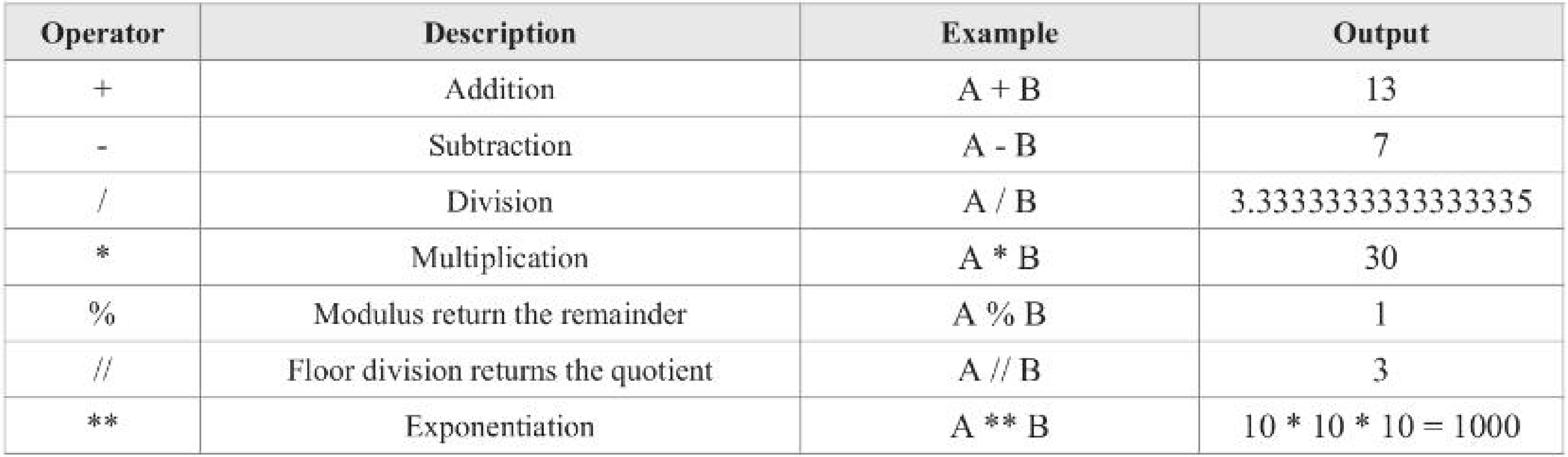
Operators in Python

-> We can use python operatos in the following group

1) Arithmetic Operators
2) Comparision Operators
3) Logical Operators
4) Identify Operators
5) Membership Operators
6) Bitwise Operators

Arithmetic Operators

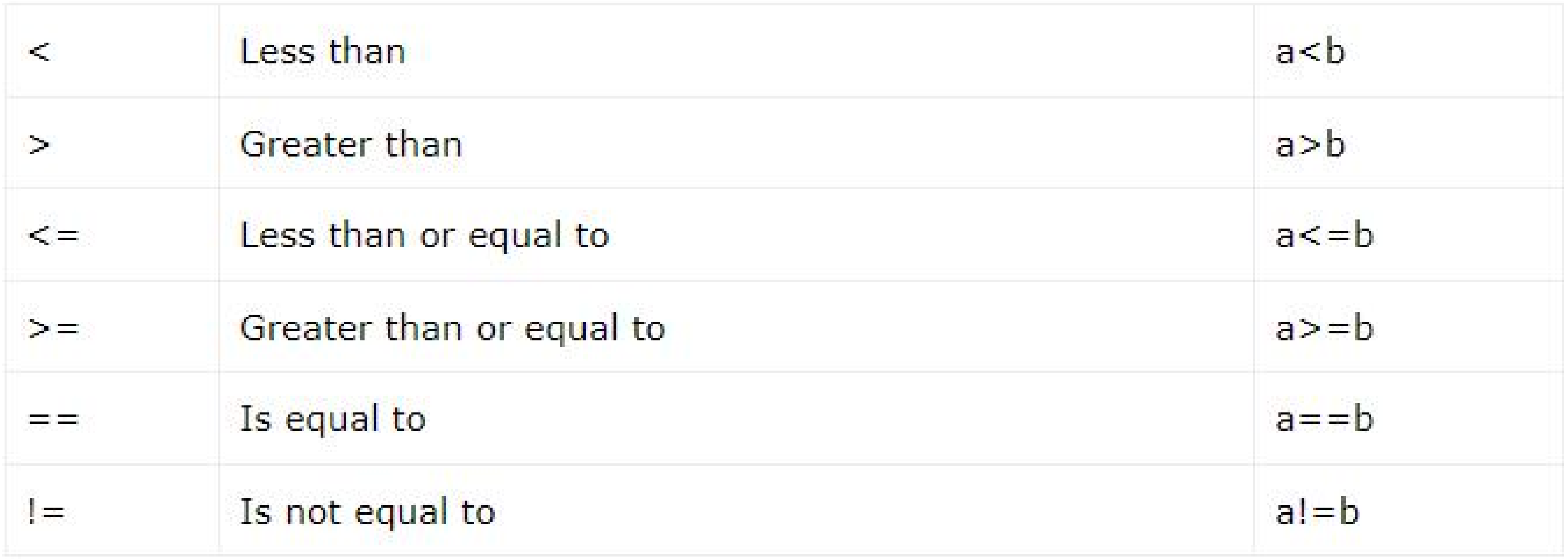
- Note : consider A = 10 and B = 3



Operator	Description	Example	Output
+	Addition	A + B	13
-	Subtraction	A - B	7
/	Division	A / B	3.3333333333333335
*	Multiplication	A * B	30
%	Modulus return the remainder	A % B	1
//	Floor division returns the quotient	A // B	3
**	Exponentiation	A ** B	10 * 10 * 10 = 1000

Comparision Operators

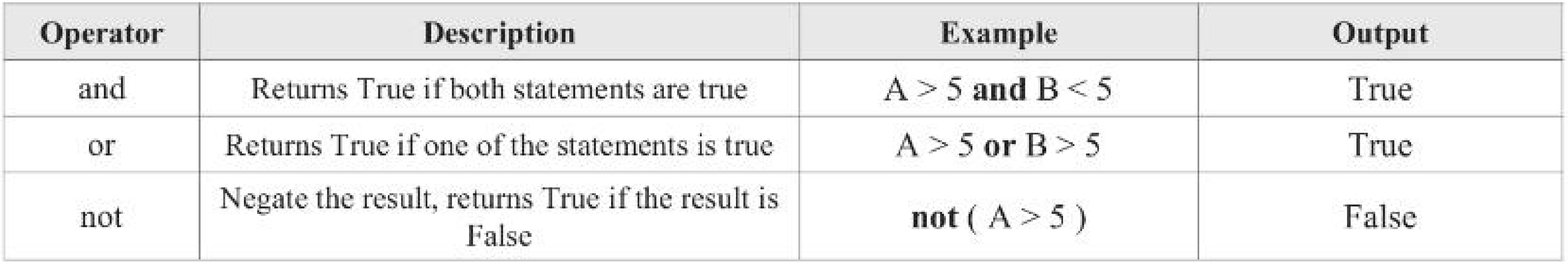
- Comparison operators are used to compare two values:



<	Less than	a<b
>	Greater than	a>b
<=	Less than or equal to	a<=b
>=	Greater than or equal to	a>=b
==	Is equal to	a==b
!=	Is not equal to	a!=b

Logical Operators

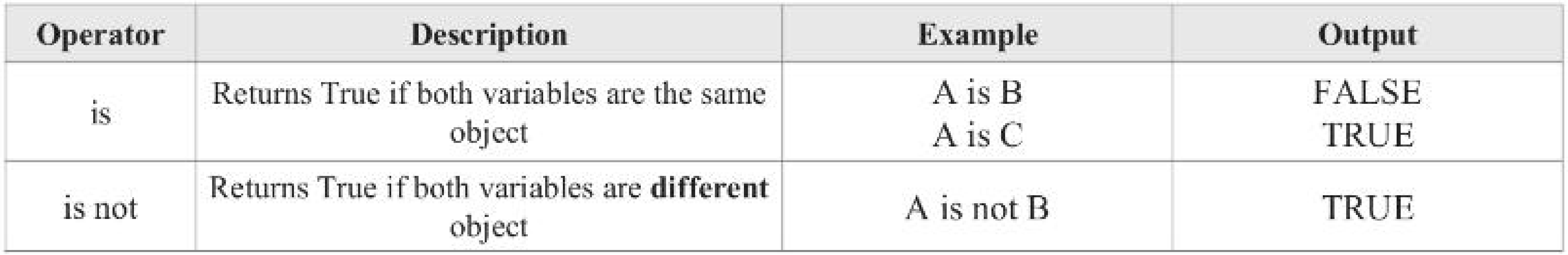
- Note : Consider A = 10 and B = 3



Operator	Description	Example	Output
and	Returns True if both statements are true	A > 5 **and** B < 5	True
or	Returns True if one of the statements is true	A > 5 **or** B > 5	True
not	Negate the result, returns True if the result is False	**not** (A > 5)	False

Identify Operators

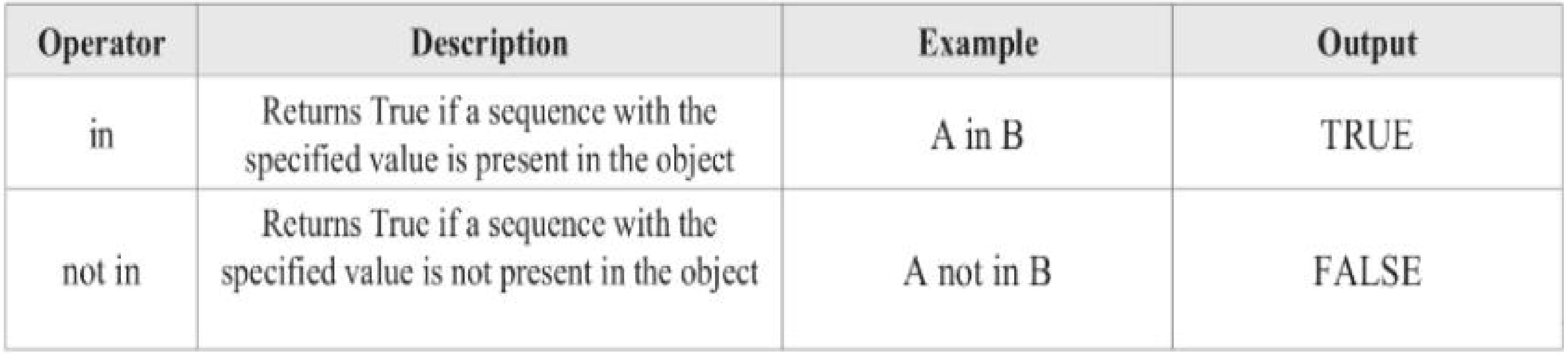
- Note : Consider A = [1,2] , B = [1,2] and C = A



Operator	Description	Example	Output
is	Returns True if both variables are the same object	A is B A is C	FALSE TRUE
is not	Returns True if both variables are **different** object	A is not B	TRUE

Members Operators

- Note : Consider A = 2 and B = [1,2,3]



Operator	Description	Example	Output
in	Returns True if a sequence with the specified value is present in the object	A in B	TRUE
not in	Returns True if a sequence with the specified value is not present in the object	A not in B	FALSE

Identifier and Reserved Words

- Identifier is a user-defined name given to a variable, function, class, module, etc.
- The identifier is a combination of character digits and an underscore.
- They are case-sensitive i.e., ‘num’ and ‘Num’ and ‘NUM’ are three different identifiers in python.
- It is a good programming practice to give meaningful names to identifiers to make the code understandable.

Rules for Naming Python Identifiers

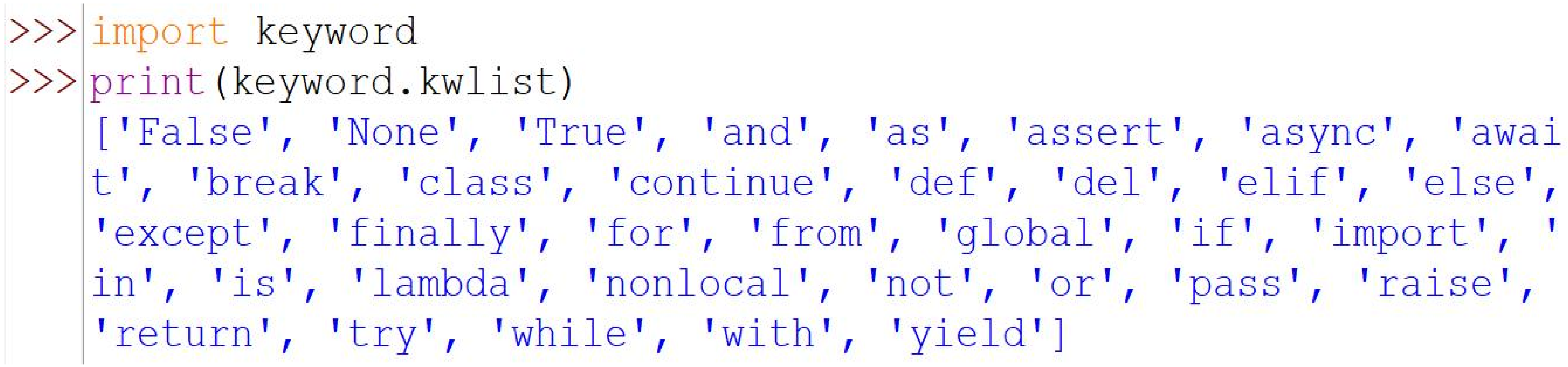
- It cannot be a reserved python keyword.
- It should not contain white space.
- It can be a combination of A-Z, a-z, 0-9, or underscore.
- It should start with an alphabet character or an underscore (_).
- It should not contain any special character other than an underscore (_).

Valid identifiers:

- var1
- _var1
- _1_var
- var_1

Invalid Identifiers

- !var1
- 1var
- 1_var
- var#1
- var 1



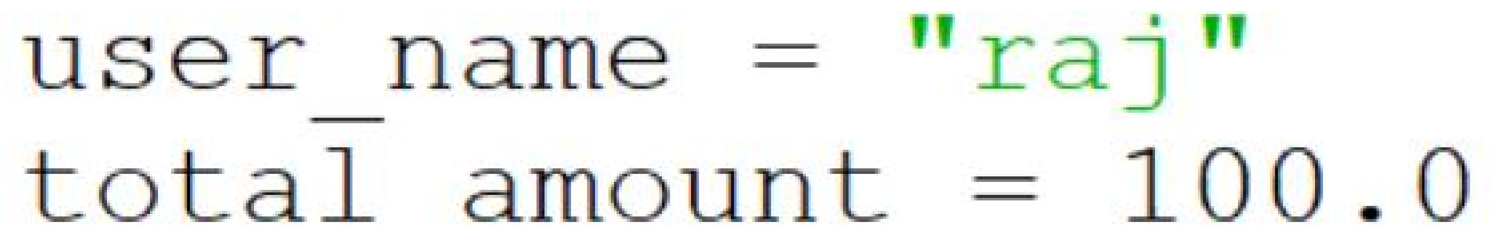
```
>>> import keyword
>>> print(keyword.kwlist)
    ['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'awai
    t', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else',
    'except', 'finally', 'for', 'from', 'global', 'if', 'import', '
    in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
    'return', 'try', 'while', 'with', 'yield']
```

Naming convention in python

1) Variables

-> Use Lowercase and Uppercase to seperate words

-> Example ‘my_variable’ , ‘user_name’

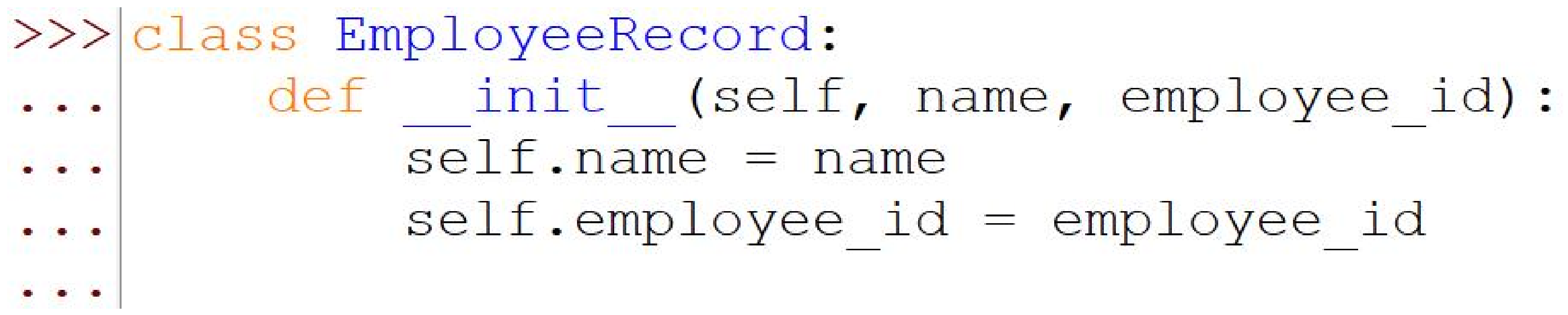


```
user_name = "raj"
total amount = 100.0
```

2) Classes

-> Use the CapWords (PascalCase) Convention.

-> Example: ‘MyClass’ , ‘EmployeeRecord’

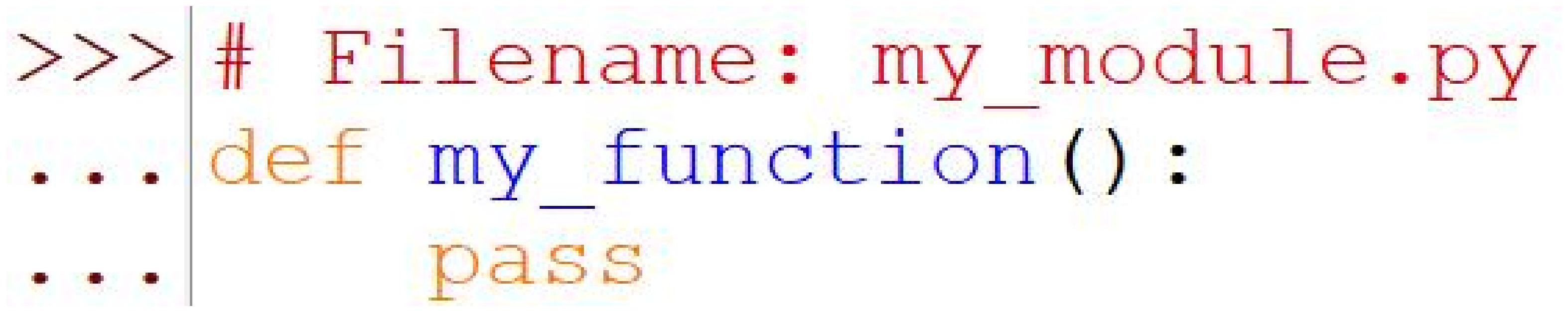


```
>>> class EmployeeRecord:
...     def __init__(self, name, employee_id):
...         self.name = name
...         self.employee_id = employee_id
...
```

3) Modules

-> Use short , lowercase names. Use underscores if it improves readablity

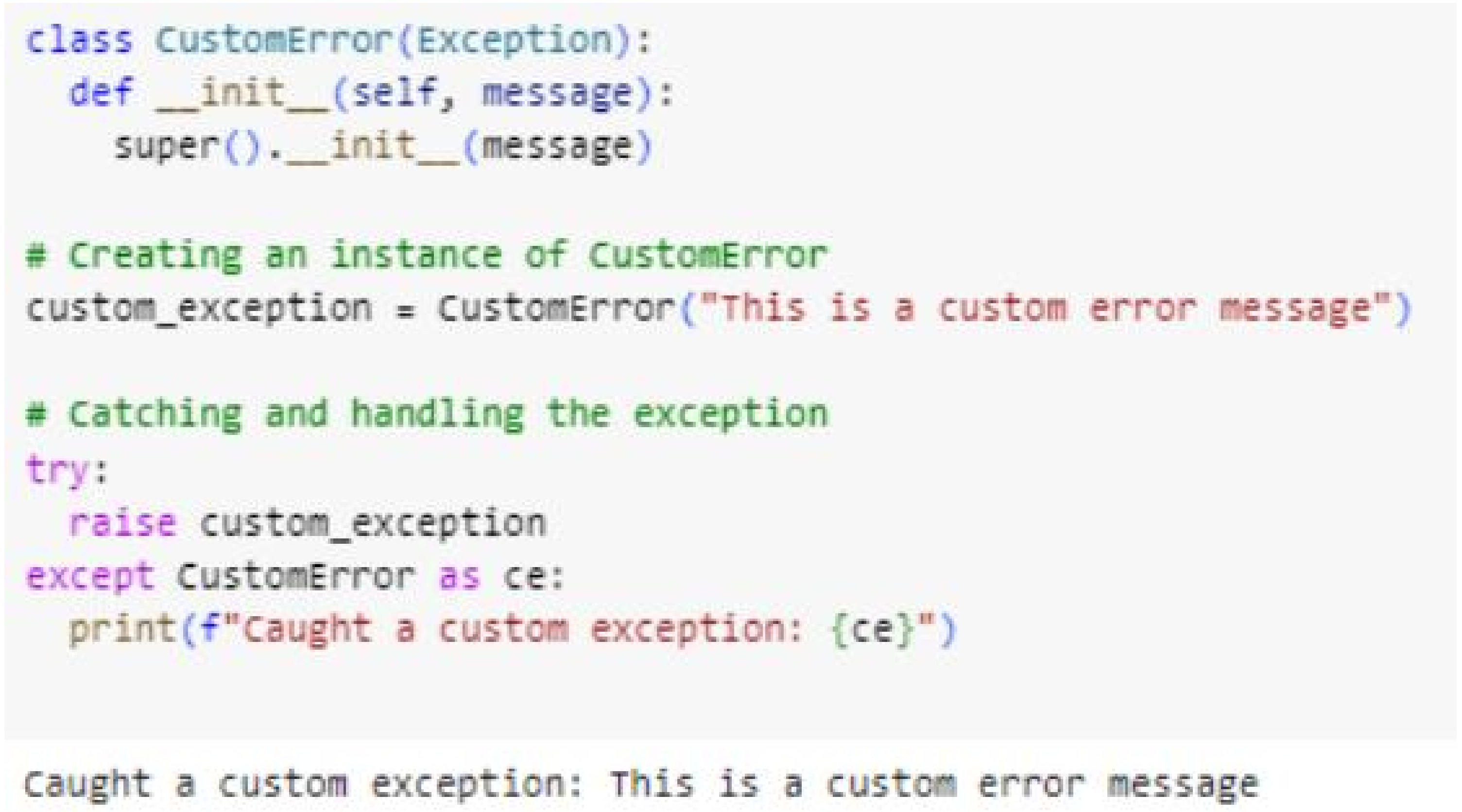
-> Example: ‘my_module.py’ , ‘data_processing’



```
>>> # Filename: my_module.py
... def my_function():
...     pass
```

4) Exception

-> Exception in Python names should end with “Error,” following the CapWords convention.



```
class CustomError(Exception):
  def __init__(self, message):
    super().__init__(message)

# Creating an instance of CustomError
custom_exception = CustomError("This is a custom error message")

# Catching and handling the exception
try:
  raise custom_exception
except CustomError as ce:
  print(f"Caught a custom exception: {ce}")
```

```
Caught a custom exception: This is a custom error message
```