Kampus Merdeka x MyEduSolve

Python Week #5

Team 3 - Data Science A

Introduction to Python

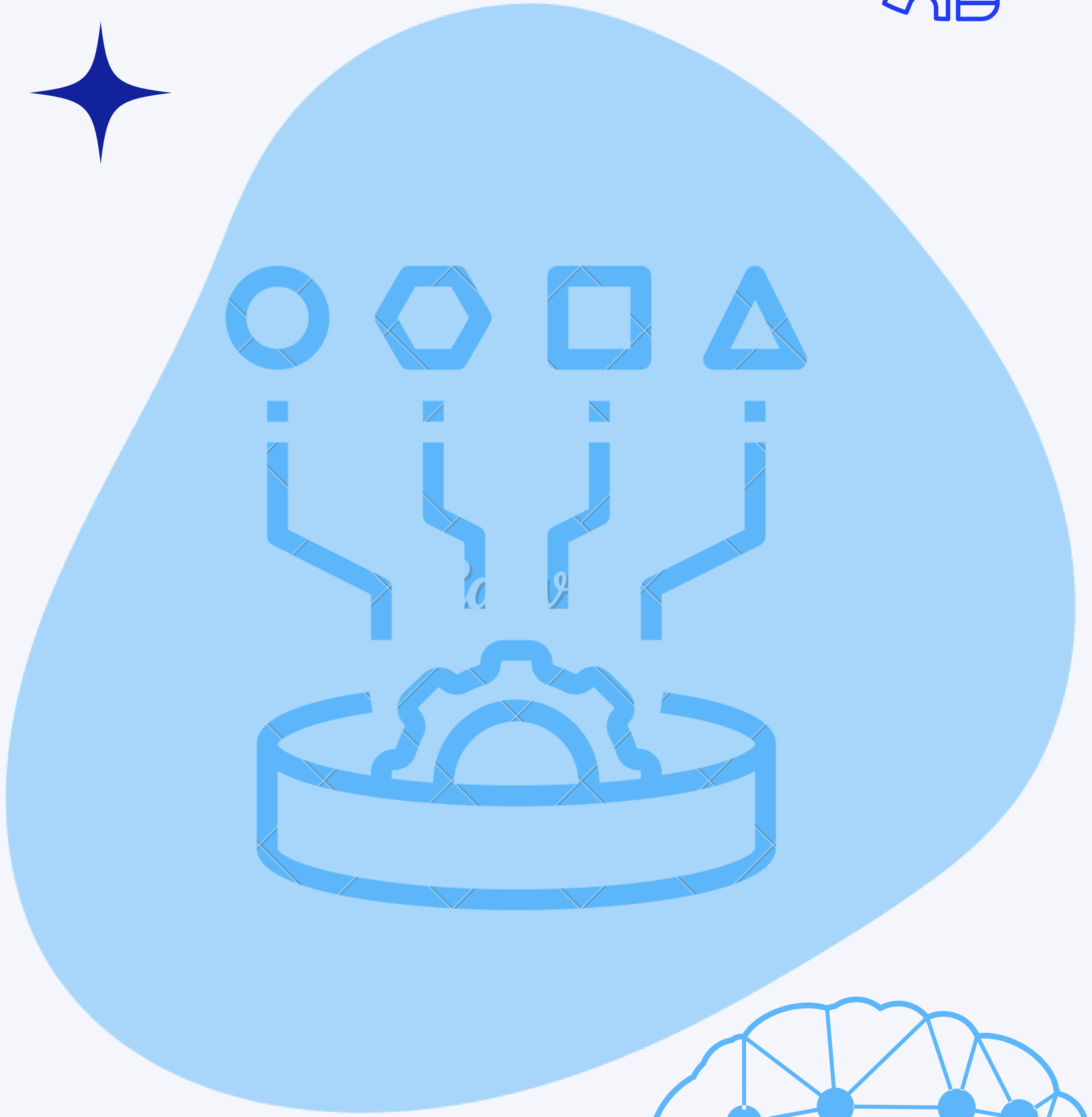
Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.
It is one of the best language used by data scientist for various data science projects/application.

Variable

A Python variable is a reserved memory location to store values.

- Variable can be declared by any name or even alphabets like a, aa, abc, etc.
- To create a variable, we just assign it a value and then start using it. Assignment is done with a single equals sign (=).

Variables can hold values, and every value has a data-type



Data Types

01 Integer

02 Float

03 String

04 Boolean

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data.

Data Types

Integer

is a data type for numeric objects in the form of positive and negative integers, for example 0, 1, 2, 3, and so on, as well as -1, -2, -3, and so on.

Float

is used to represent real numbers and is written with a decimal point dividing the integer and fractional parts, for example 2.5, 0.005.

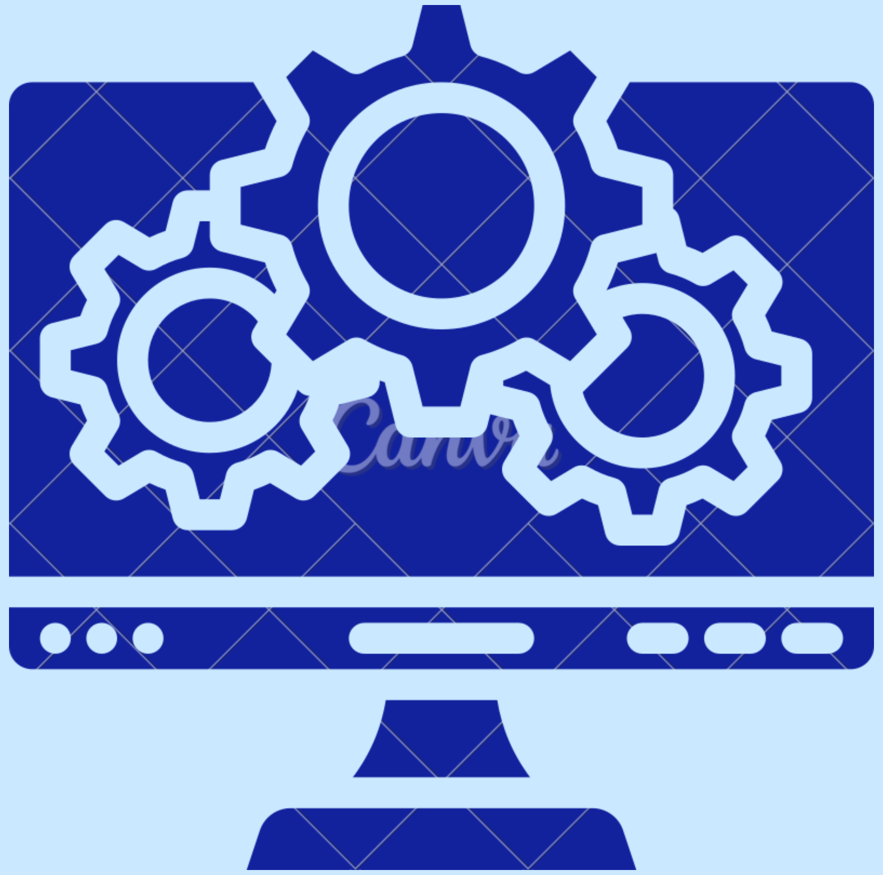
String

it contains sequence of characters. It represents by using single quotes, double quotes or triple quotes. It is denoted by the class str.

Boolean

has only two values, True and False. It is denoted by the class bool. This data type is used to check whether a condition is true or false

Data Types Operators



Assignment Operators

Comparison Operators

Logical Operators

Arithmetic Operators

Assignment Operators

An assignment statement evaluates the expression list and assigns the single resulting object to each of the target lists, from left to right.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x= x * 3

Comparison Operators

Python has six comparison operators, which are as follows:

- Less than (<)
- Less than or equal to (<=)
- Greater that (>)
- Greater than or equal to (>=)
- Equal to (==)
- Not equal to (!=)

These comparison operators compare two values and return a boolean value, either True or False. And we can use these comparison operators to compare both numbers and strings

Logical Operators

Logical operators are used on conditional statements. They perform Logical AND, Logical OR and Logical NOT operations. This operator supports short-circuit evaluation, which means that if the first argument is FALSE the second is never evaluated.

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operands is false	not x

Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations.

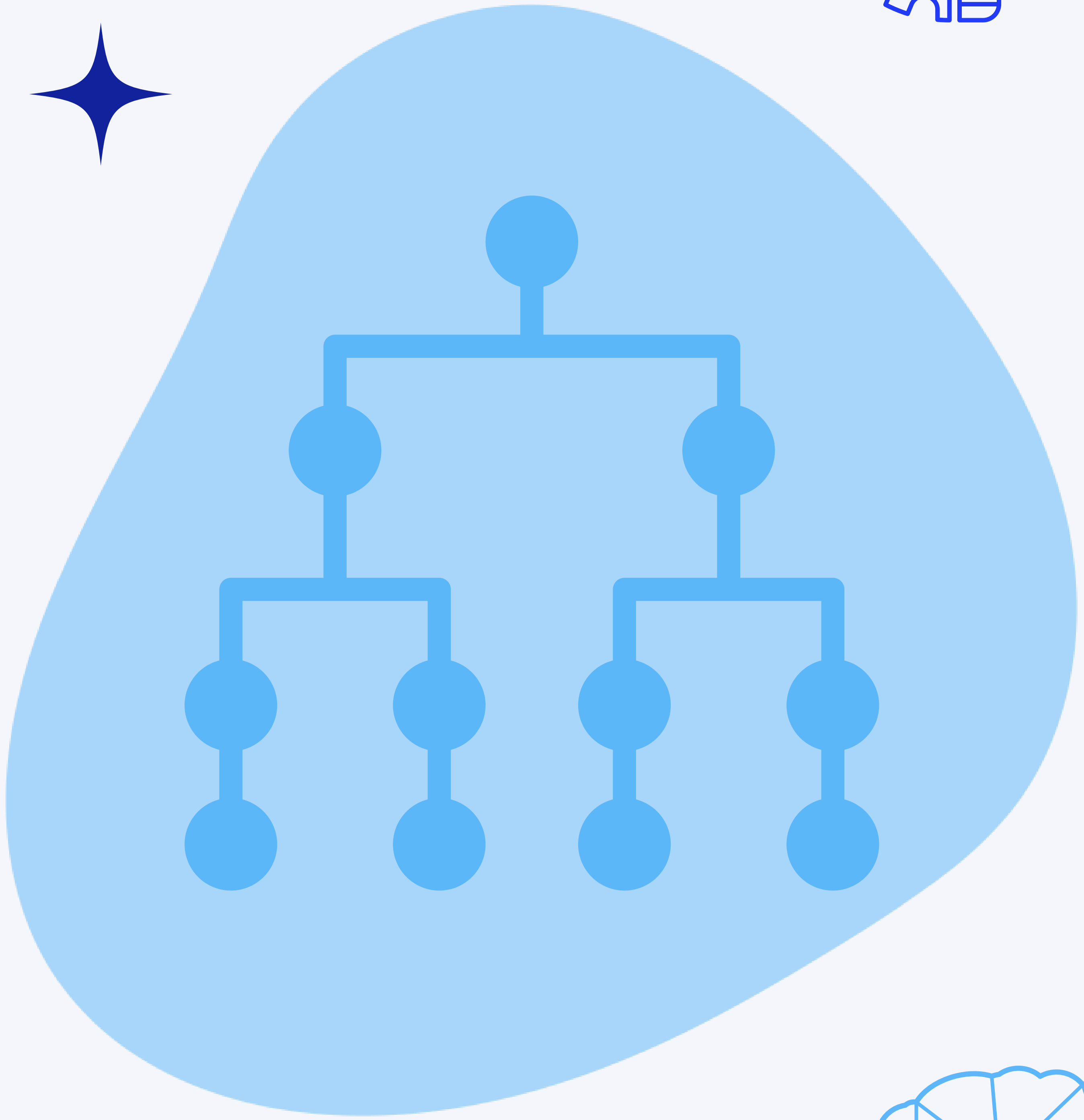
Operator	Name	Example
+	Addition	x + y
-	Substraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Data Structures

Data structures are a way of organizing data so that it can be accessed more efficiently depending upon the situation.

Python has implicit support for data structures which enable to store and access data.

These structures are called list, tuple, set, and dictionary.

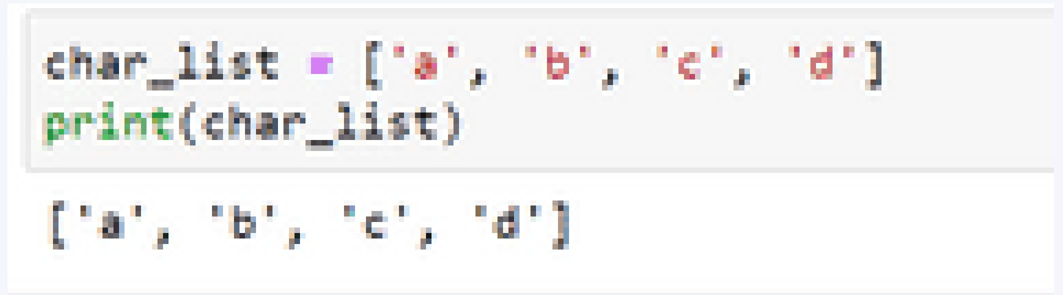


LIST

- List are used to store data of different data types in a sequential manner
- There are addresses assigned to every element of the list, which is called as Index.
- The index value starts from 0 and goes on until the last element called the **positive index**. And also **negative indexing** which starts from -1 enabling you to access elements from the last to first.

01 Creating a list

To create a list, use the square brackets and add elements into it accordingly.

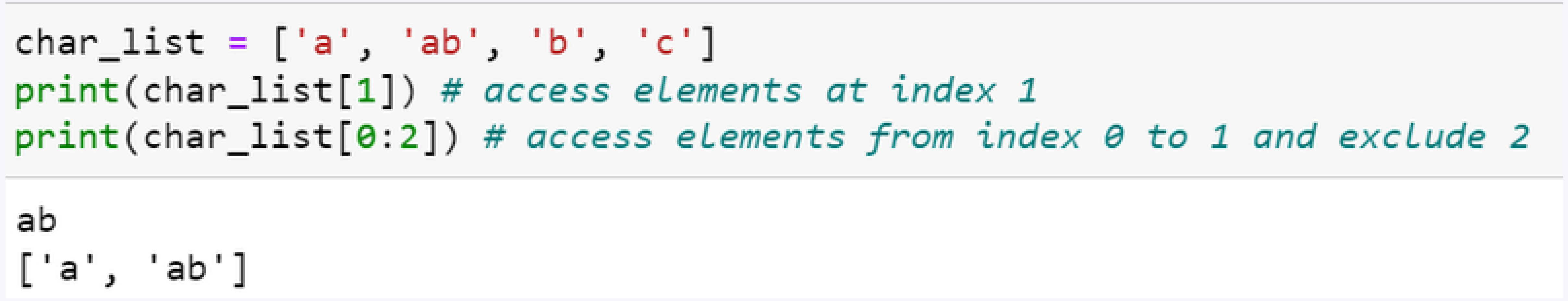


```
char_list = ['a', 'b', 'c', 'd']
print(char_list)
```

```
['a', 'b', 'c', 'd']
```

02 Accessing elements

To accessing elements in list, use the square brackets for slicing along with the index or indices to obtain value available at that index.



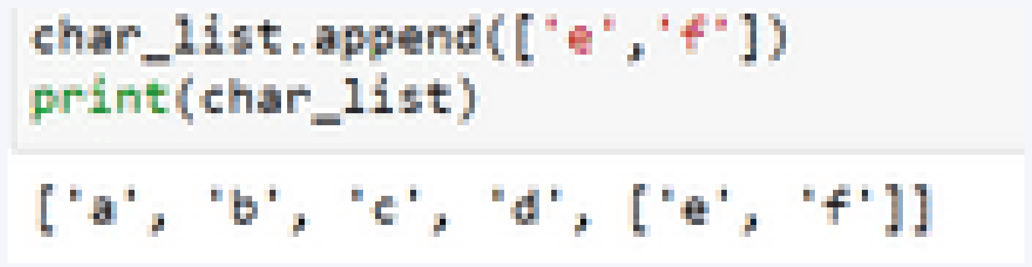
```python
char_list = ['a', 'ab', 'b', 'c']
print(char_list[1]) # access elements at index 1
print(char_list[0:2]) # access elements from index 0 to 1 and exclude 2
```

```
ab
['a', 'ab']
```

03 Adding elements

Adding the elements in the list can be achieved using the append(), extend() and insert() functions.

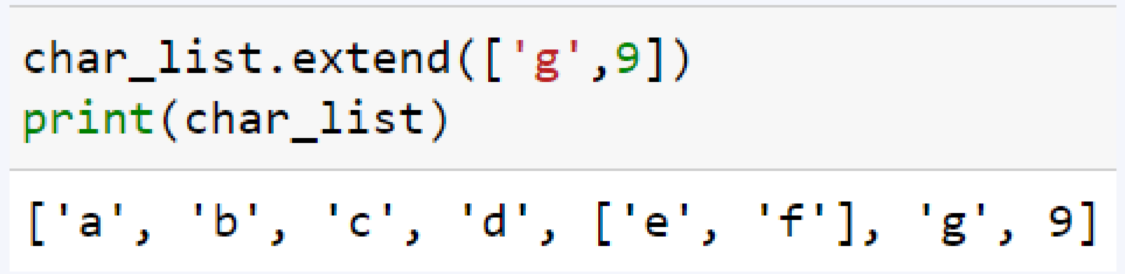
- The append() function adds all the elements passed to it as a single element.



```
char_list.append(['e','f'])
print(char_list)
```

```
['a', 'b', 'c', 'd', ['e', 'f']]
```

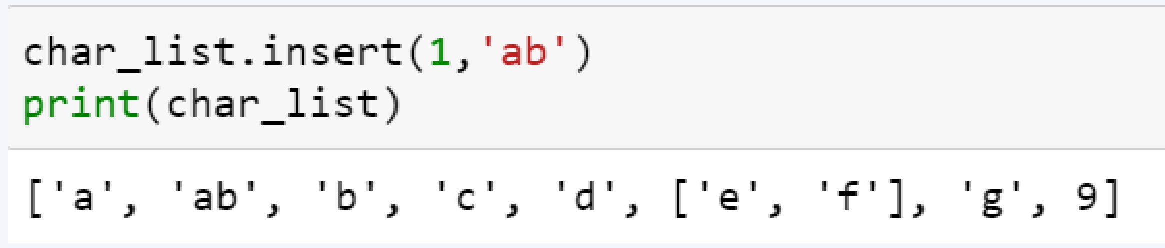
- The extend() function adds the elements one-by-one into the list.



```
char_list.extend(['g',9])
print(char_list)
```

```
['a', 'b', 'c', 'd', ['e', 'f'], 'g', 9]
```

- The insert() function adds the element passed to the index value and increase the size of the list too.



```
char_list.insert(1,'ab')
print(char_list)
```

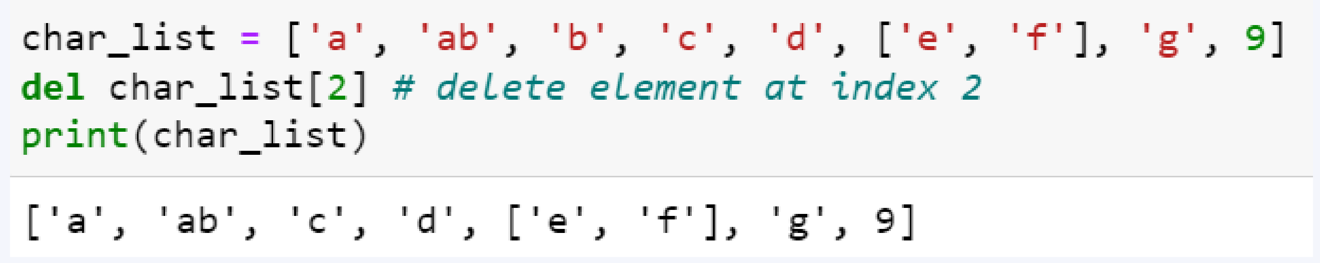
```
['a', 'ab', 'b', 'c', 'd', ['e', 'f'], 'g', 9]
```

04

Deleting elements

Deleting the elements in the list can be achieved using the del keyword, pop() and element() functions.

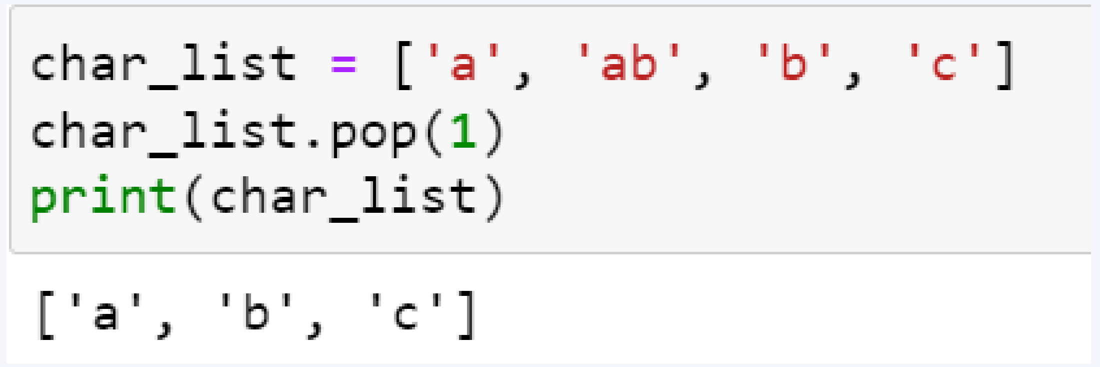
- To delete elements, use the *del keyword* which is built-in into Python but this does not return anything back to us



```
char_list = ['a', 'ab', 'b', 'c', 'd', ['e', 'f'], 'g', 9]
del char_list[2] # delete element at index 2
print(char_list)
```

```
['a', 'ab', 'c', 'd', ['e', 'f'], 'g', 9]
```

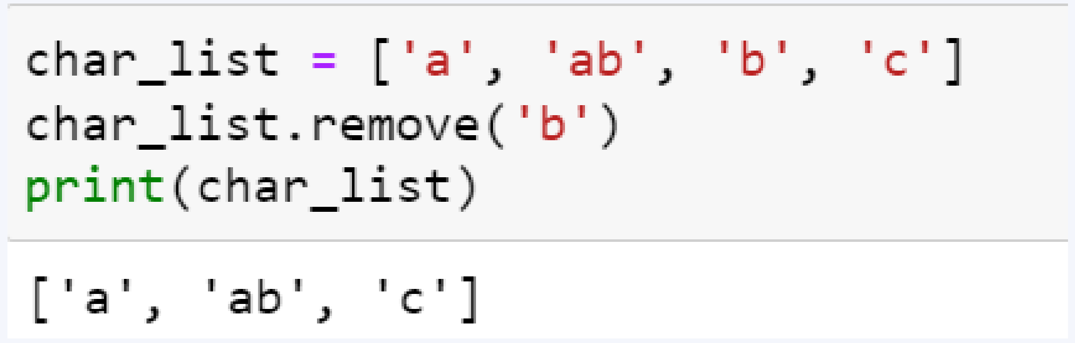
- To delete element that we want the element back, use the pop() function which takes the index value.



```
char_list = ['a', 'ab', 'b', 'c']
char_list.pop(1)
print(char_list)
```

```
['a', 'b', 'c']
```

- To remove an element by its value, you use the remove() function.



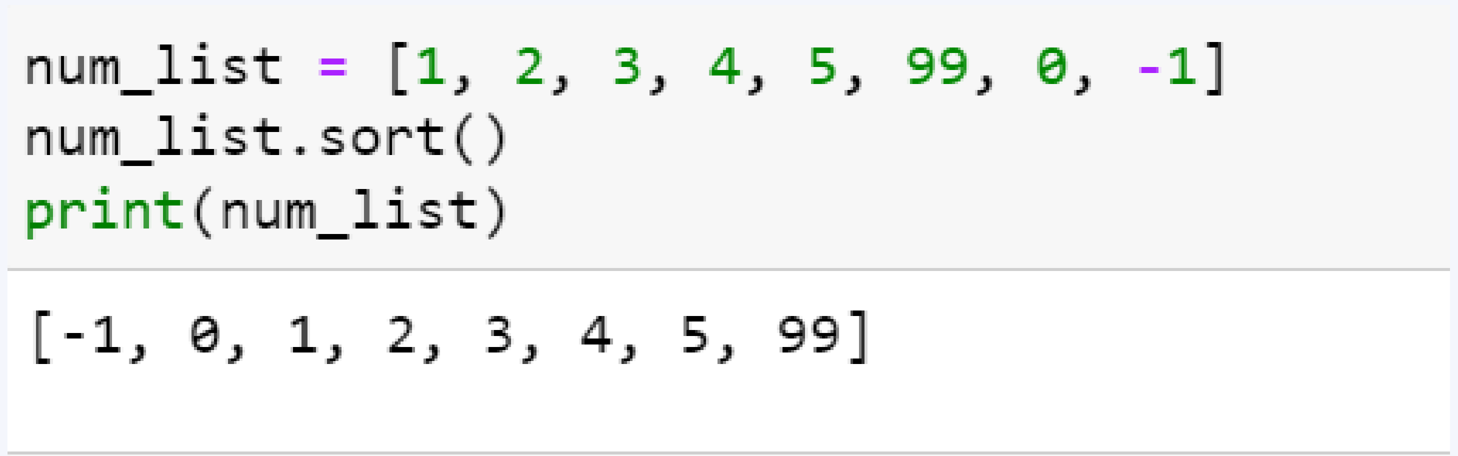
```
char_list = ['a', 'ab', 'b', 'c']
char_list.remove('b')
print(char_list)
```

```
['a', 'ab', 'c']
```

05 Sorting elements

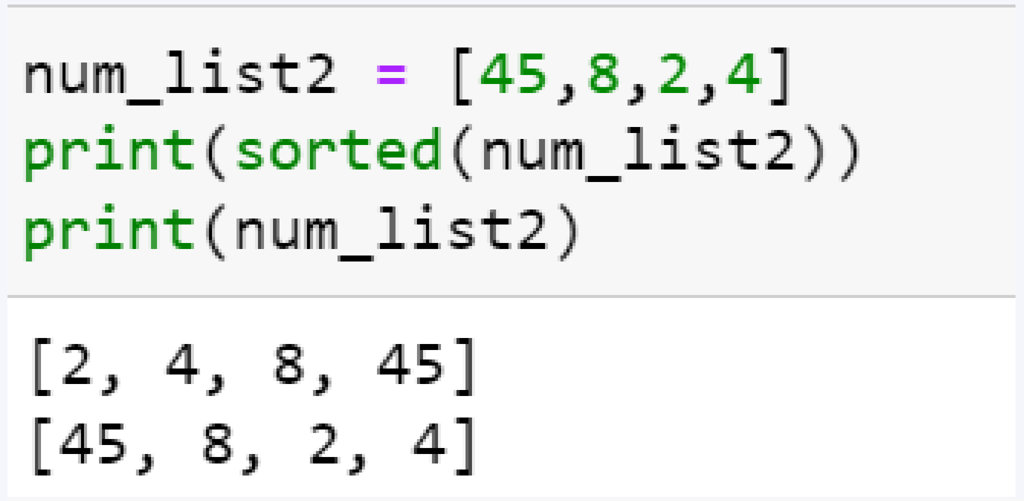
There are two basic function for sorting lists, sort and sorted. The **sort**() sorts the list in place, while the **sorted**() returns a new sorted list from the items in iterable.

Both functions have the same options, key and reverse. The **key** takes a function which will be used on each value in the list being sorted to determine the resulting order. The **reverse** option can reverse the comparison order.



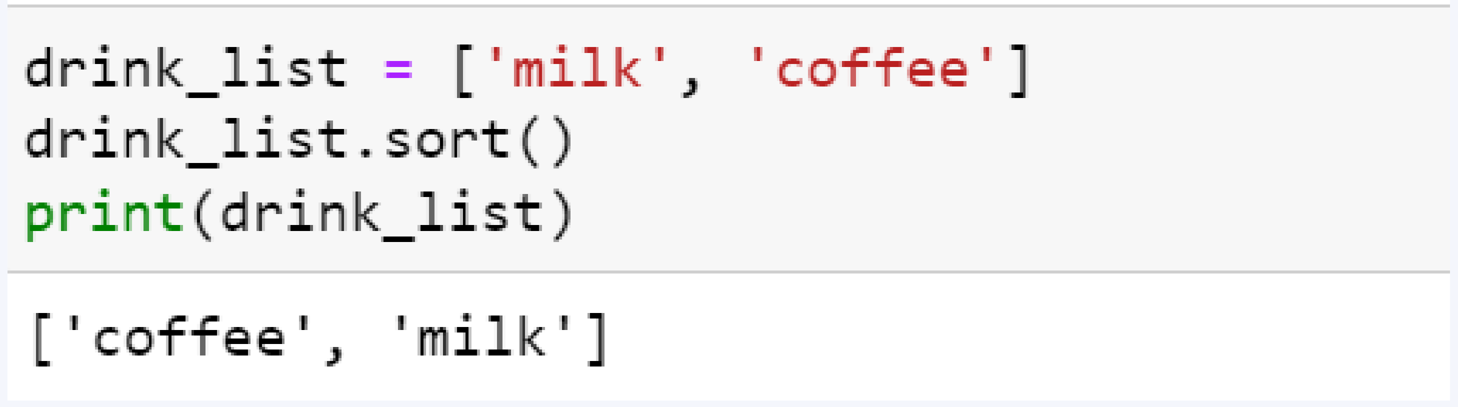
```
num_list = [1, 2, 3, 4, 5, 99, 0, -1]
num_list.sort()
print(num_list)
```

```
[-1, 0, 1, 2, 3, 4, 5, 99]
```



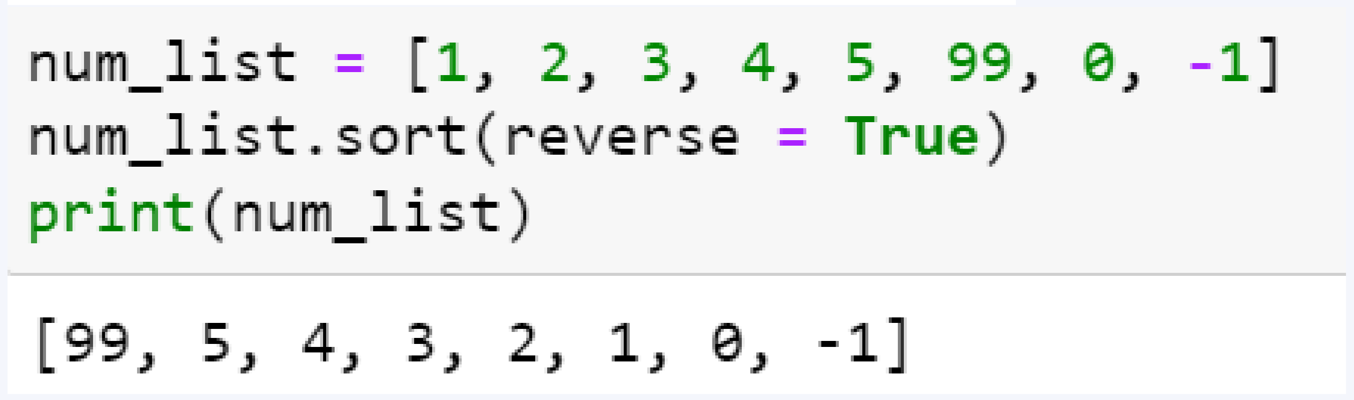
```
num_list2 = [45,8,2,4]
print(sorted(num_list2))
print(num_list2)
```

```
[2, 4, 8, 45]
[45, 8, 2, 4]
```



```
drink_list = ['milk', 'coffee']
drink_list.sort()
print(drink_list)
```

```
['coffee', 'milk']
```



```
num_list = [1, 2, 3, 4, 5, 99, 0, -1]
num_list.sort(reverse = True)
print(num_list)
```

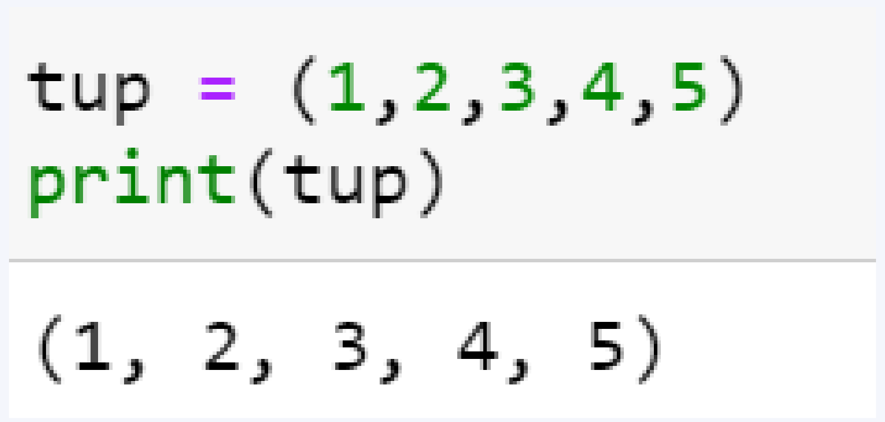
```
[99, 5, 4, 3, 2, 1, 0, -1]
```

TUPLE

Tuple are the same as lists are with the exception that the data once entered into the tuple can't be changed no matter what.

01 Creating a tuple

To create a tuple, use parenthesis or using the tuple() function.



```
tup = (1,2,3,4,5)
print(tup)
```

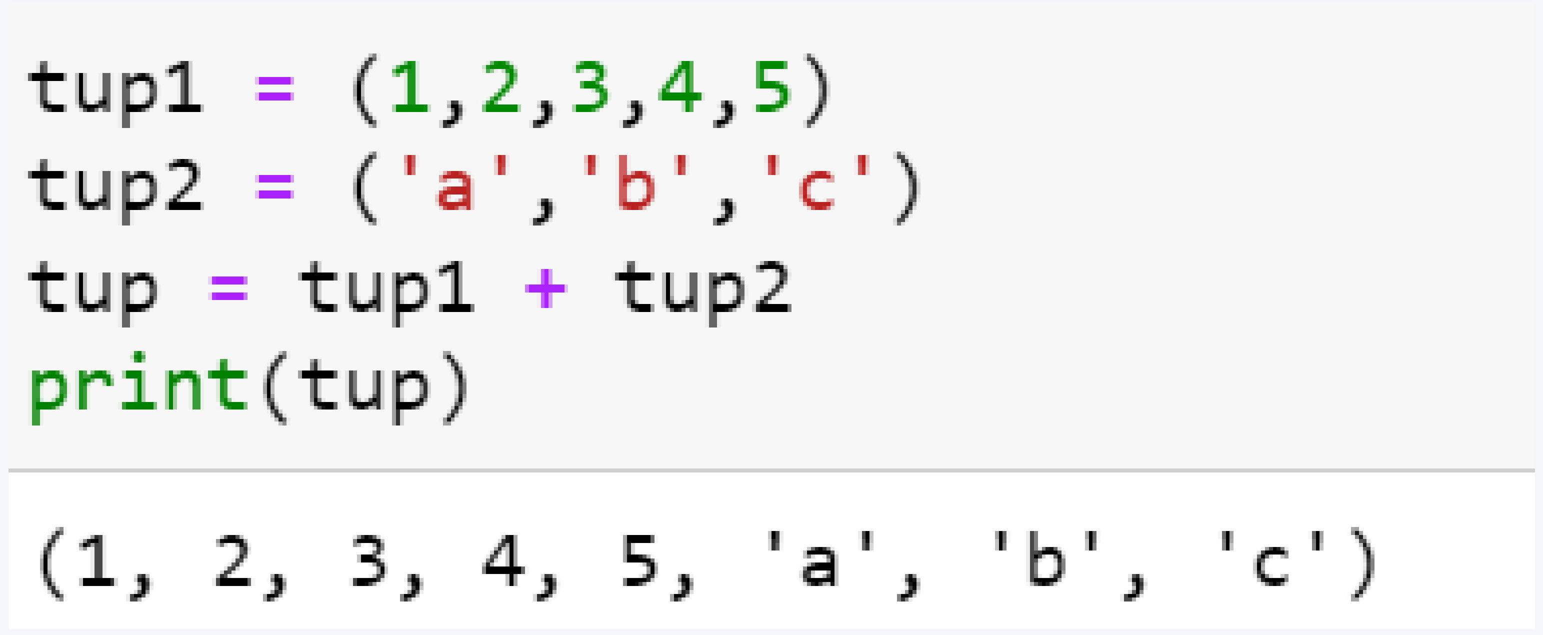
```
(1, 2, 3, 4, 5)
```

02 Accessing elements

Accessing elements is the same as it is for accessing values in lists.

03 Appending elements

To append the values, use the '+' operator which will take another tuple to be appended to it.



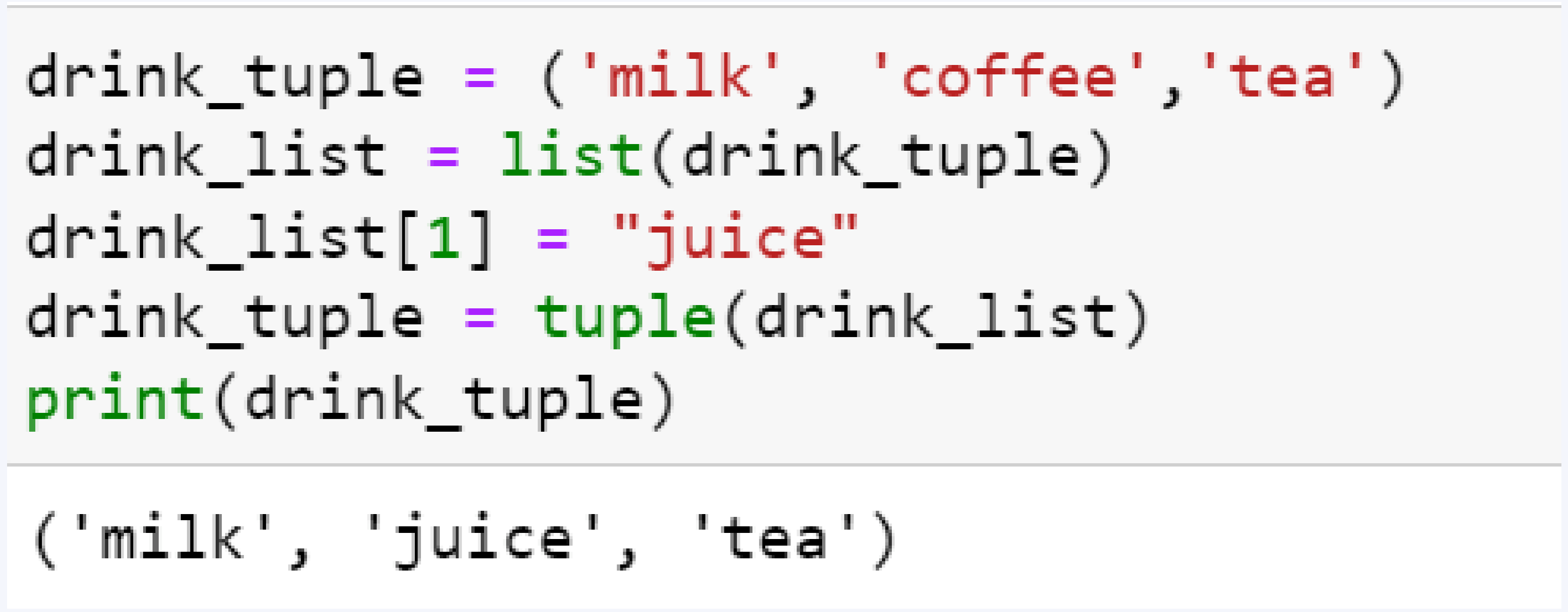
```
tup1 = (1,2,3,4,5)
tup2 = ('a','b','c')
tup = tup1 + tup2
print(tup)
```

```
(1, 2, 3, 4, 5, 'a', 'b', 'c')
```

04

Change tuple values (converting tuple to list)

Once a tuple is created, we cannot change its values. Tuples are unchangeable, or immutable.
But there is a workaround. We can convert the tuple into a list, change the list, and convert the list back into a tuple.



```
drink_tuple = ('milk', 'coffee','tea')
drink_list = list(drink_tuple)
drink_list[1] = "juice"
drink_tuple = tuple(drink_list)
print(drink_tuple)
```

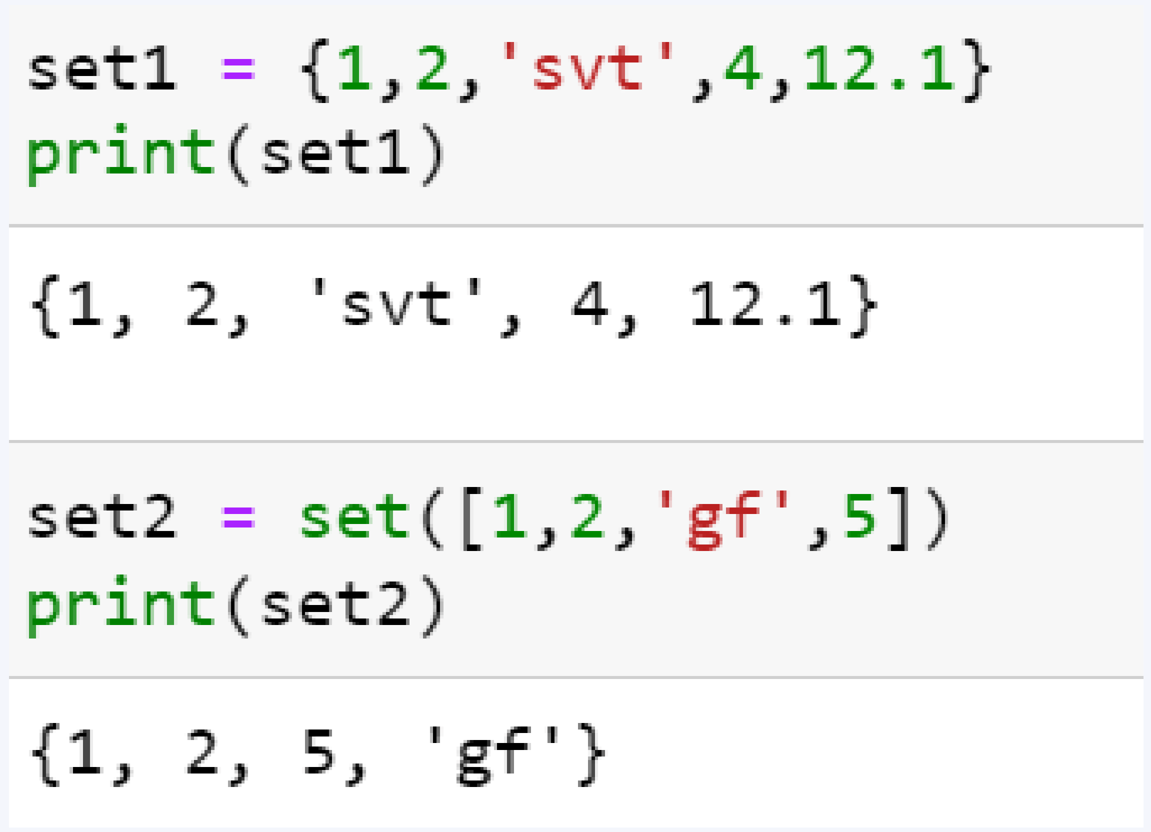
```
('milk', 'juice', 'tea')
```

SET

Set are a collection of unordered elements that are unique.

01 Creating a set

To create a set, use the flower braces, or we can create a set from list using set().



```
set1 = {1,2,'svt',4,12.1}
print(set1)
```

```
{1, 2, 'svt', 4, 12.1}
```

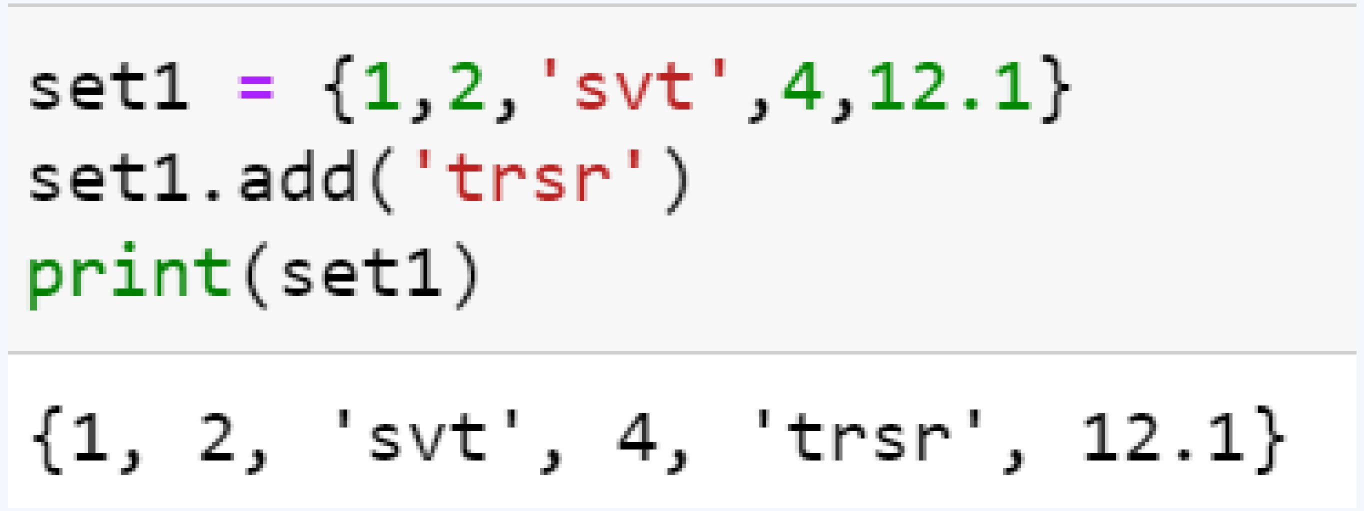
```
set2 = set([1,2,'gf',5])
print(set2)
```

```
{1, 2, 5, 'gf'}
```

02

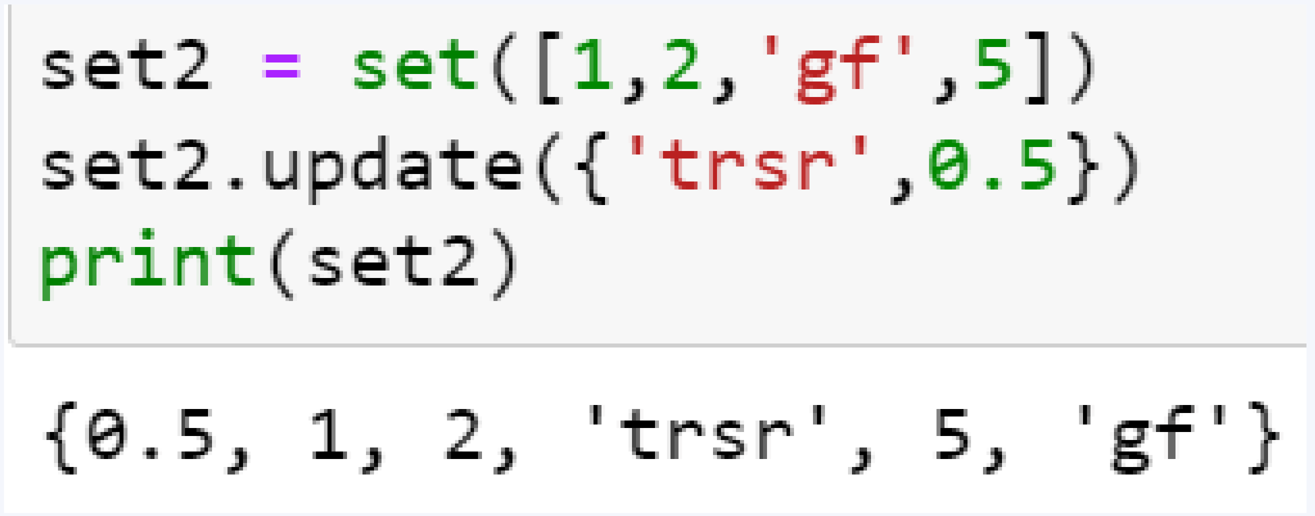
Adding elements

To add elements, use the add() or update() function and pass the value to it.



```
set1 = {1,2,'svt',4,12.1}
set1.add('trsr')
print(set1)
```

```
{1, 2, 'svt', 4, 'trsr', 12.1}
```



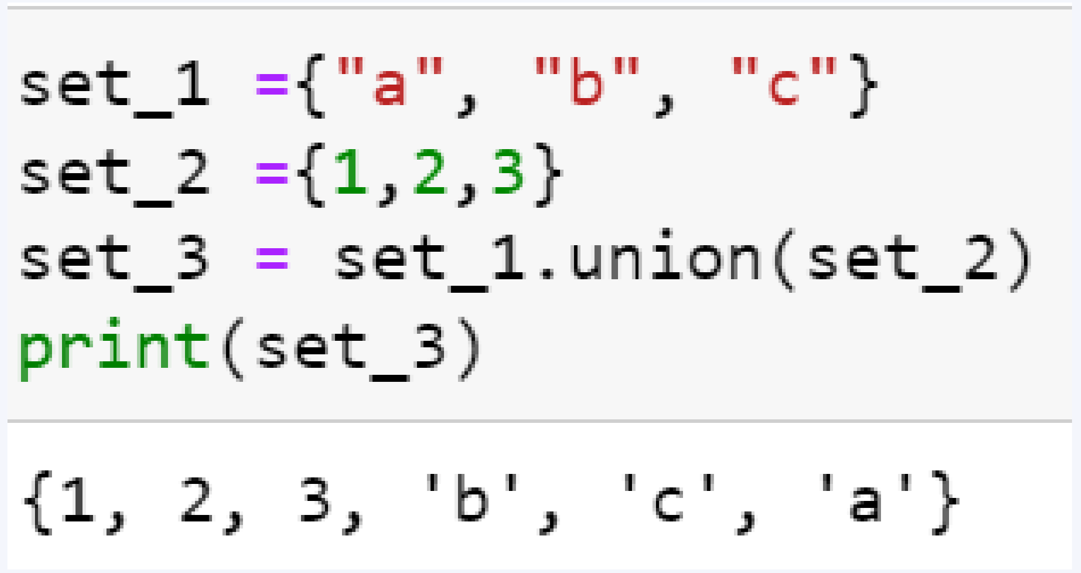
```
set2 = set([1,2,'gf',5])
set2.update({'trsr',0.5})
print(set2)
```

```
{0.5, 1, 2, 'trsr', 5, 'gf'}
```

03

Operations in set

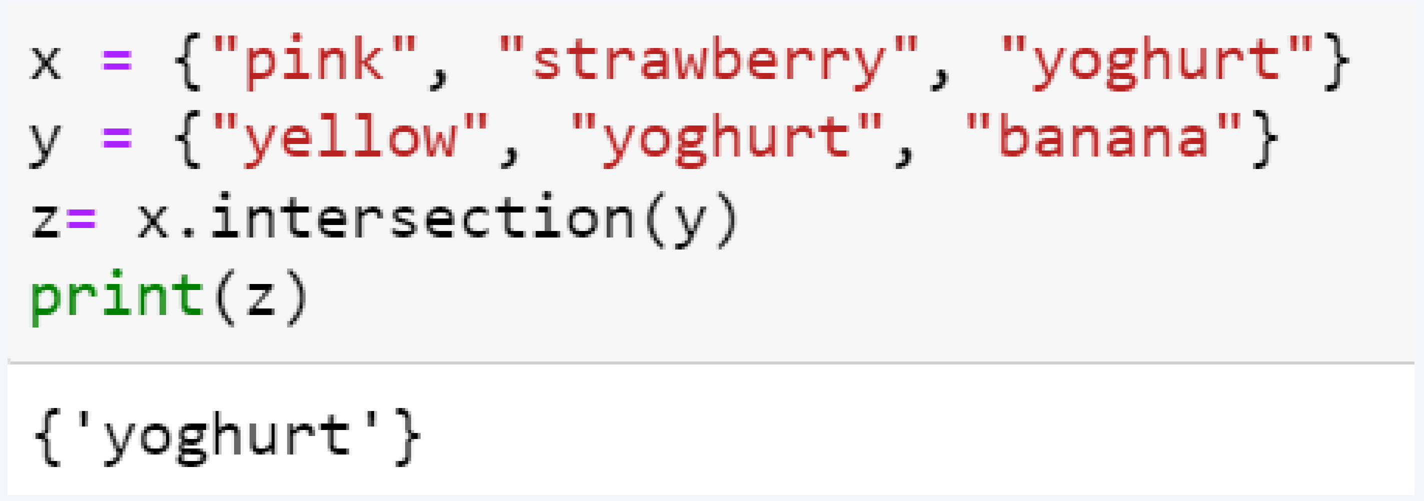
- The union() function combines the data present in both sets.



```
set_1 ={"a", "b", "c"}
set_2 ={1,2,3}
set_3 = set_1.union(set_2)
print(set_3)
```

```
{1, 2, 3, 'b', 'c', 'a'}
```

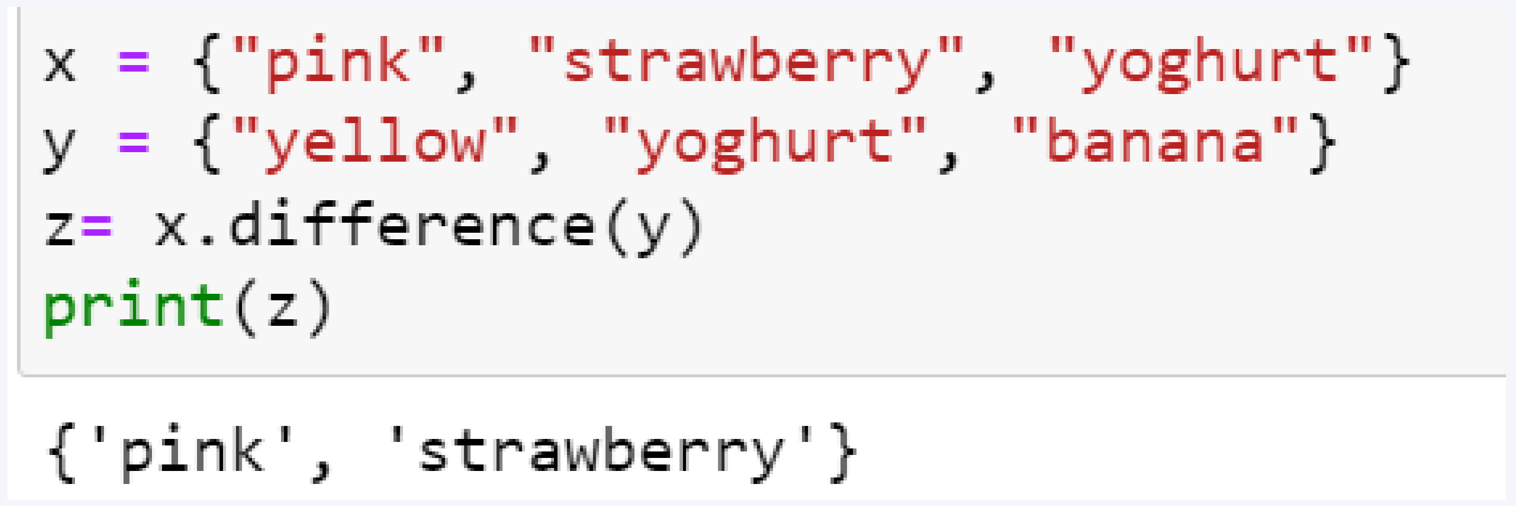
- The intersection() function finds the data present in both sets only.



```
x = {"pink", "strawberry", "yoghurt"}
y = {"yellow", "yoghurt", "banana"}
z= x.intersection(y)
print(z)
```

```
{'yoghurt'}
```

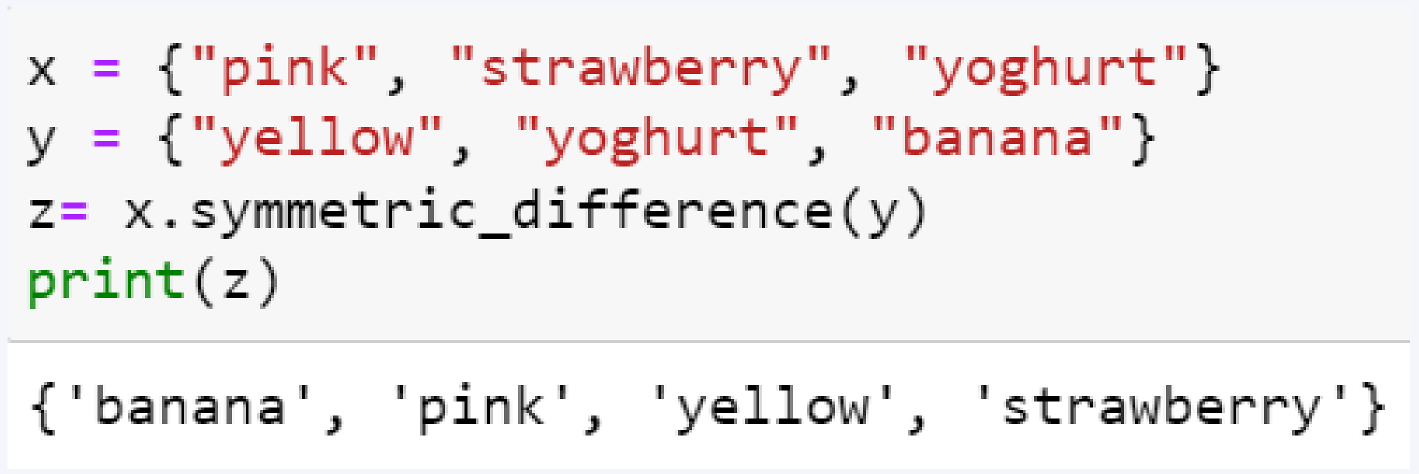
- The difference() function deletes the data present in both and outputs data present only in the set passed.



```
x = {"pink", "strawberry", "yoghurt"}
y = {"yellow", "yoghurt", "banana"}
z= x.difference(y)
print(z)
```

```
{'pink', 'strawberry'}
```

- The symmetric_difference() does the same as the difference() function but outputs the data which is remaining in both sets.



```
x = {"pink", "strawberry", "yoghurt"}
y = {"yellow", "yoghurt", "banana"}
z= x.symmetric_difference(y)
print(z)
```

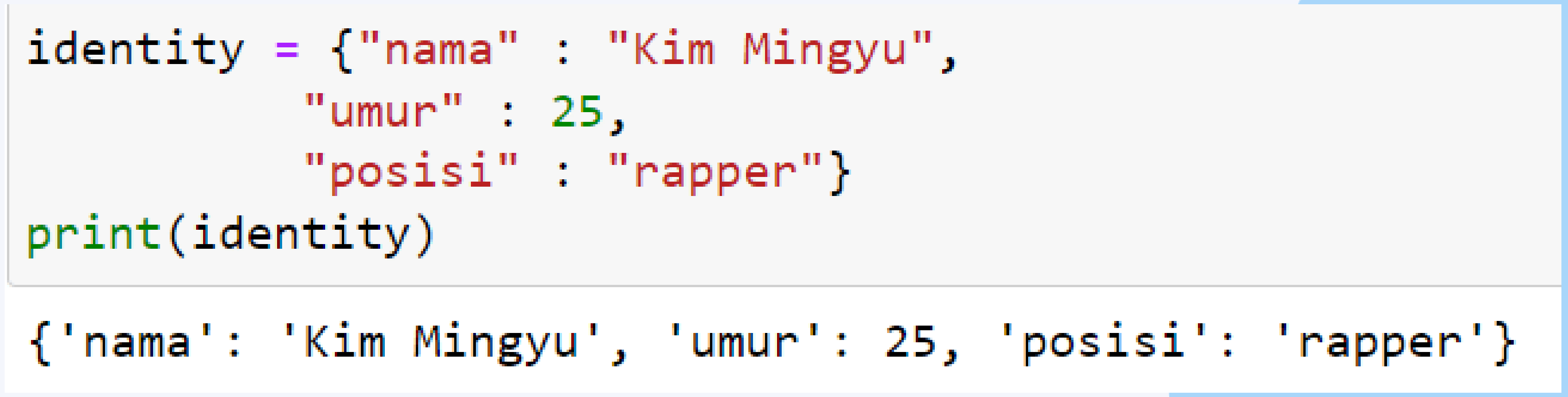
```
{'banana', 'pink', 'yellow', 'strawberry'}
```

DICTIONARY

Dictionary are used to store key-value pairs.

01 Creating a dictionary

Dictionaries can be created using the flower braces or using the dict() function.



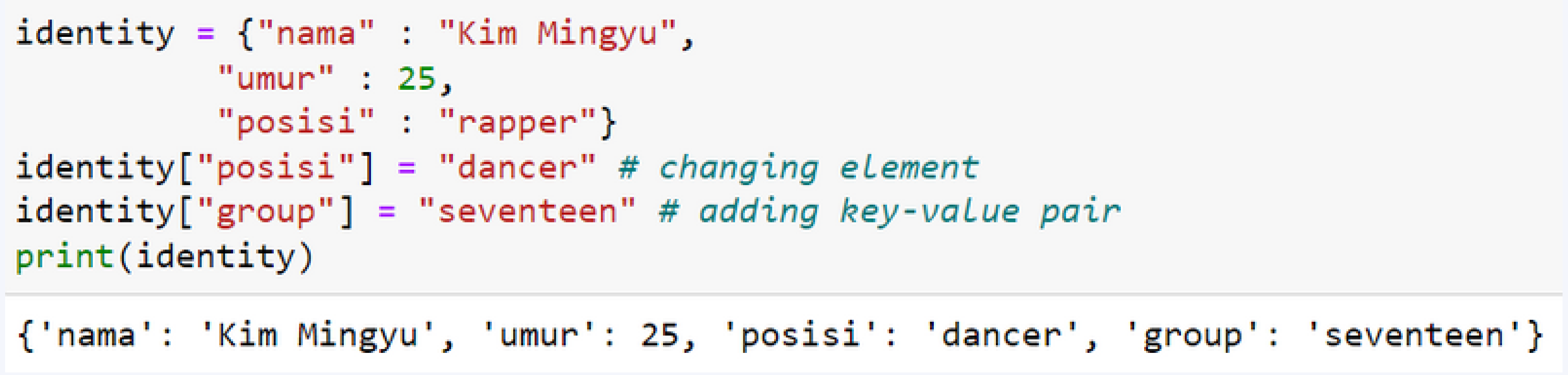
```
identity = {"nama" : "Kim Mingyu",
          "umur" : 25,
          "posisi" : "rapper"}
print(identity)
```

```
{'nama': 'Kim Mingyu', 'umur': 25, 'posisi': 'rapper'}
```

02

Changing and Adding key, value pairs

To change the values of the dictionary, use the keys. Firstly access the key, and then change the value accordingly.
To add values, just add another key-value pair as shown below.



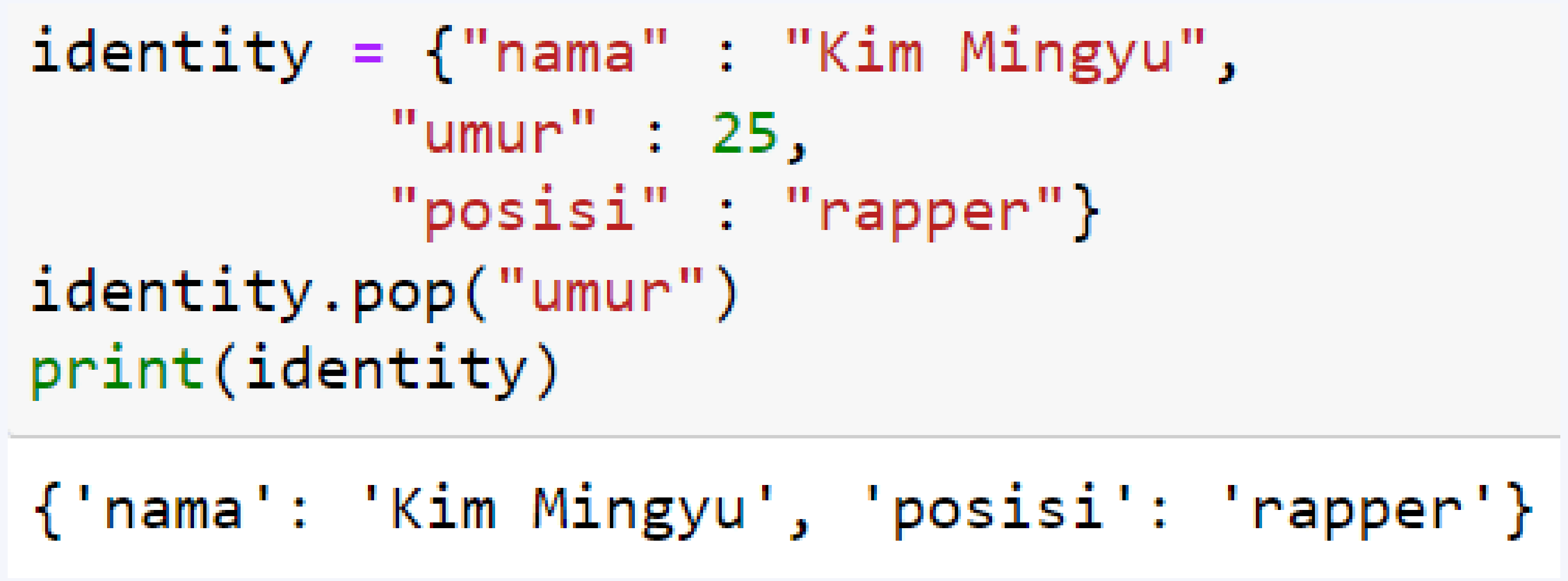
```
identity = {"nama" : "Kim Mingyu",
          "umur" : 25,
          "posisi" : "rapper"}
identity["posisi"] = "dancer" # changing element
identity["group"] = "seventeen" # adding key-value pair
print(identity)
```

```
{'nama': 'Kim Mingyu', 'umur': 25, 'posisi': 'dancer', 'group': 'seventeen'}
```

03

Deleting key, value pairs

To delete the values, use the pop() function which returns the value that has been deleted.



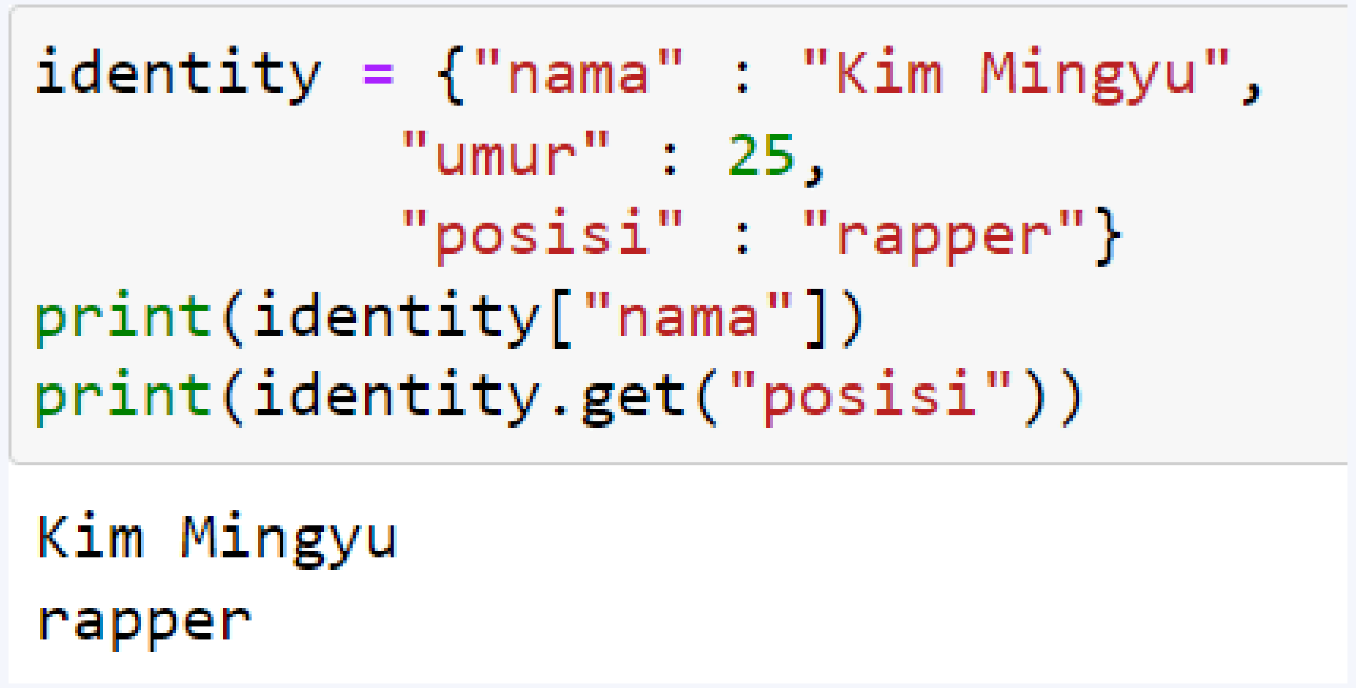
```
identity = {"nama" : "Kim Mingyu",
          "umur" : 25,
          "posisi" : "rapper"}
identity.pop("umur")
print(identity)
```

```
{'nama': 'Kim Mingyu', 'posisi': 'rapper'}
```

04

Accessing elements

To accessing the elements, we can access elements using the keys only. Use either the get() function or just pass the key values and will be retrieving the values.

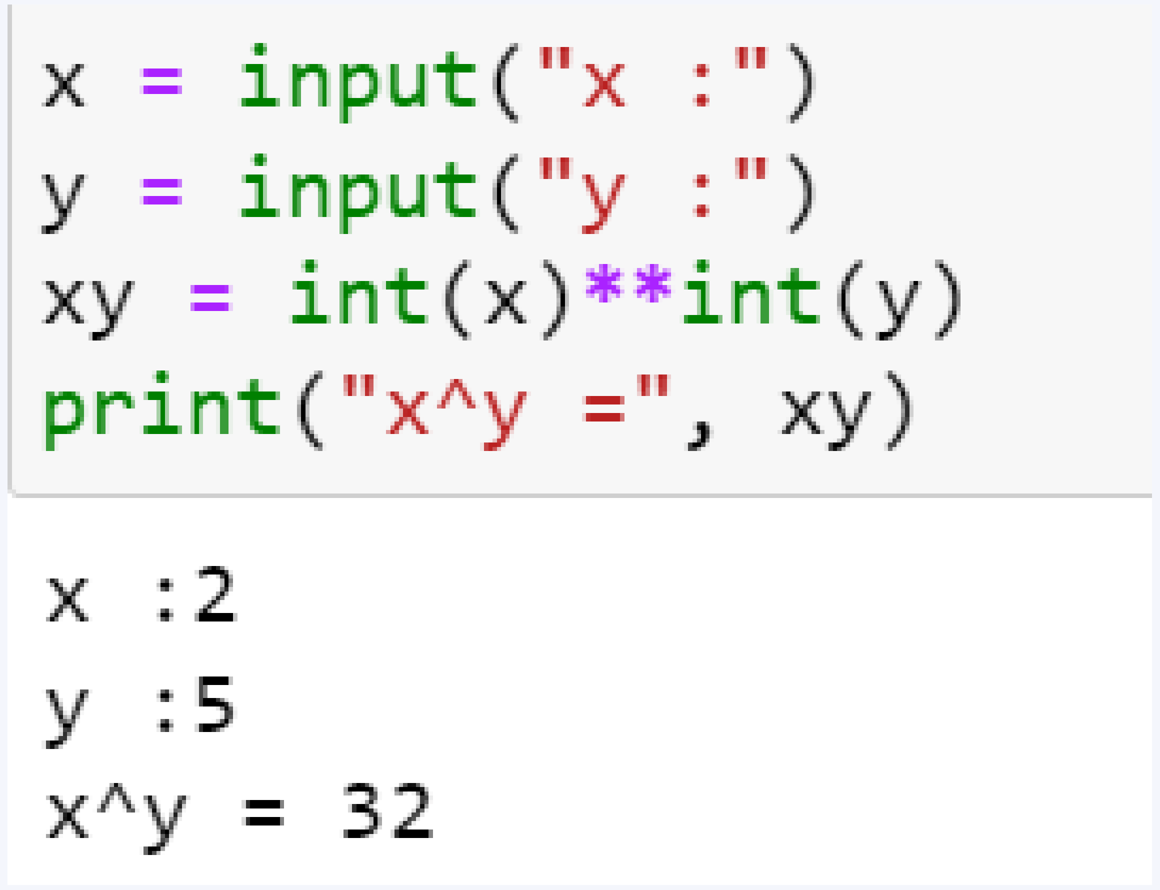


```
identity = {"nama" : "Kim Mingyu",
            "umur" : 25,
            "posisi" : "rapper"}
print(identity["nama"])
print(identity.get("posisi"))
```

```
Kim Mingyu
rapper
```

INPUT FUNCTION

The input() function takes input from the user and returns it.

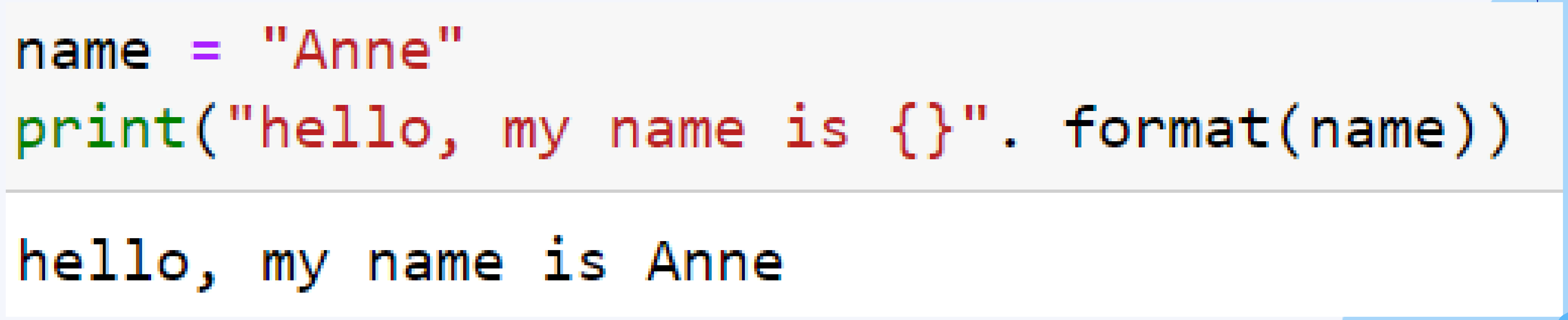


```
x = input("x :")
y = input("y :")
xy = int(x)**int(y)
print("x^y =", xy)
```

```
x :2
y :5
x^y = 32
```

FORMAT FUNCTION

The format() function is technique of the string category permits to try and do variable substitutions and data formatting. It enables to concatenate parts of a string at desired intervals through point data format.



```
name = "Anne"
print("hello, my name is {}". format(name))
```

```
hello, my name is Anne
```

Thank You