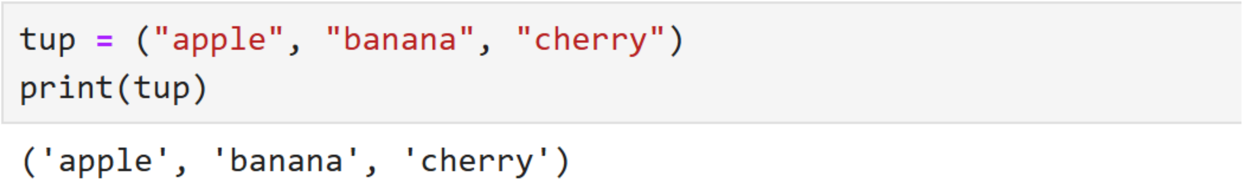
Day-2 (Advanced Python)

Tuple

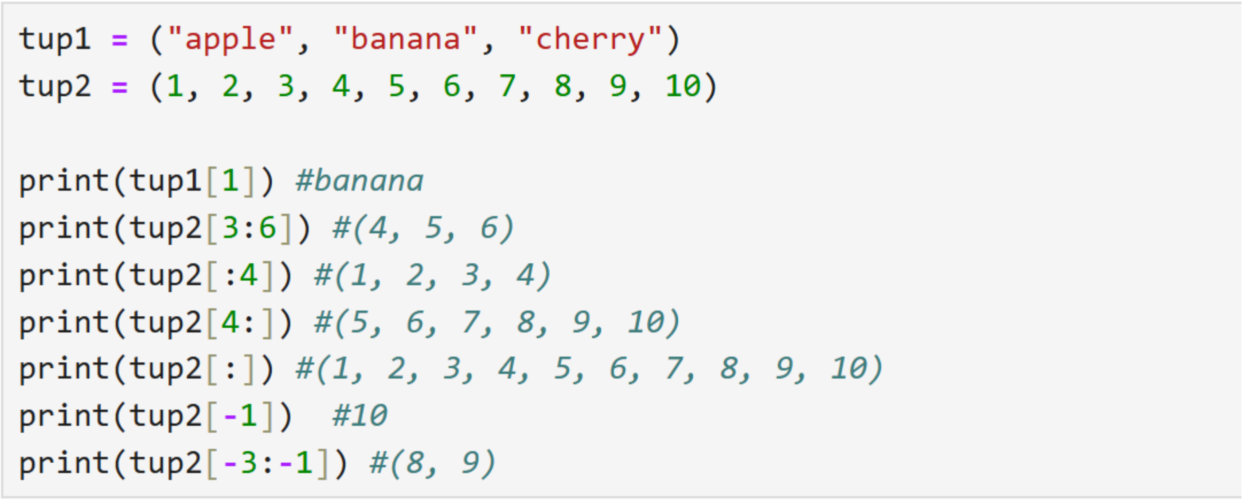
- A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.
- A tuple is an immutable sequence of Python objects.
- Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists.
- Tuples use parentheses, whereas lists use square brackets.



```
tup = ("apple", "banana", "cherry")
print(tup)
```

```
('apple', 'banana', 'cherry')
```

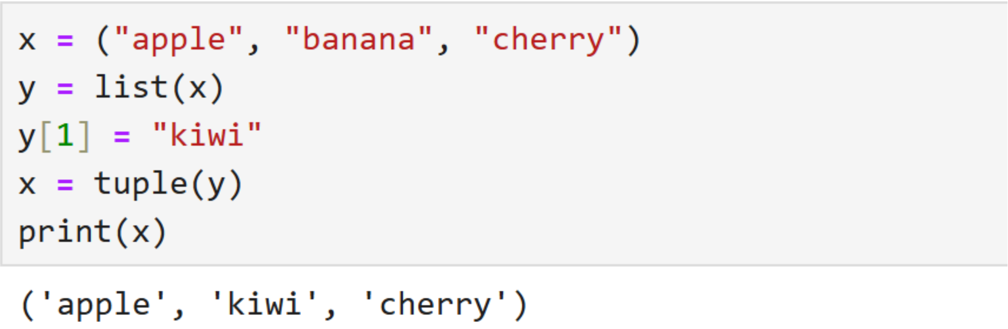
- You can **access tuple items by referring to the index number**, inside square brackets:



```
tup1 = ("apple", "banana", "cherry")
tup2 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

print(tup1[1]) #banana
print(tup2[3:6]) #(4, 5, 6)
print(tup2[:4]) #(1, 2, 3, 4)
print(tup2[4:]) #(5, 6, 7, 8, 9, 10)
print(tup2[:]) #(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print(tup2[-1])  #10
print(tup2[-3:-1]) #(8, 9)
```

How to Change Tuple Value

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

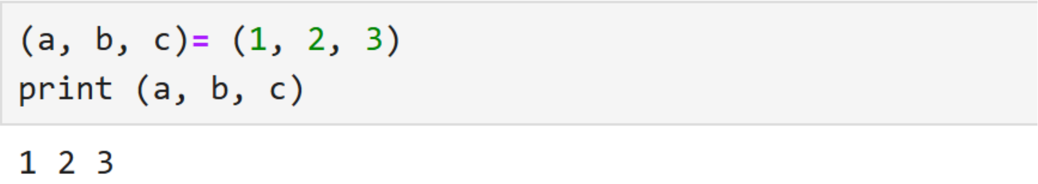


```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

```
('apple', 'kiwi', 'cherry')
```

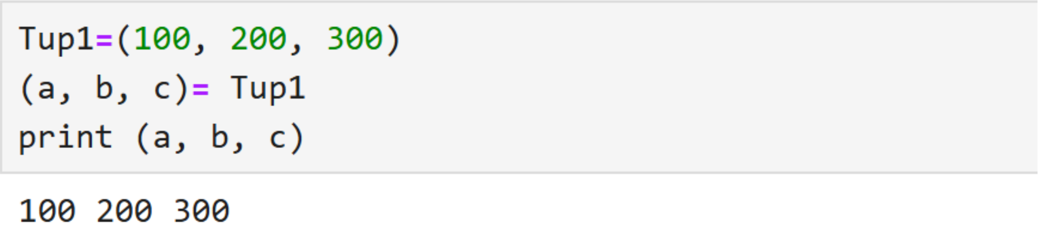
Tuple Assignment

It allows tuples of variables on the left hand side of the assignment operator to be assigned values from a tuple on the right hand side of the assignment operator. Each value is assigned to its respective variable.



```
(a, b, c)= (1, 2, 3)
print (a, b, c)
```

```
1 2 3
```

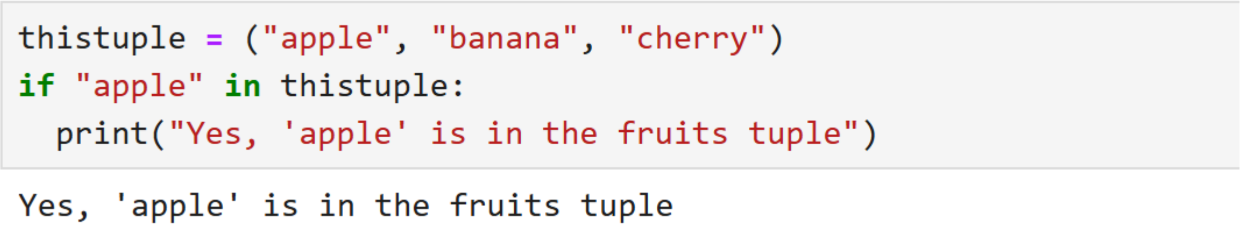


```
Tup1=(100, 200, 300)
(a, b, c)= Tup1
print (a, b, c)
```

```
100 200 300
```

Check if Item Exists In Tuple

To determine if a specified item is present in a tuple, use **in** keyword.

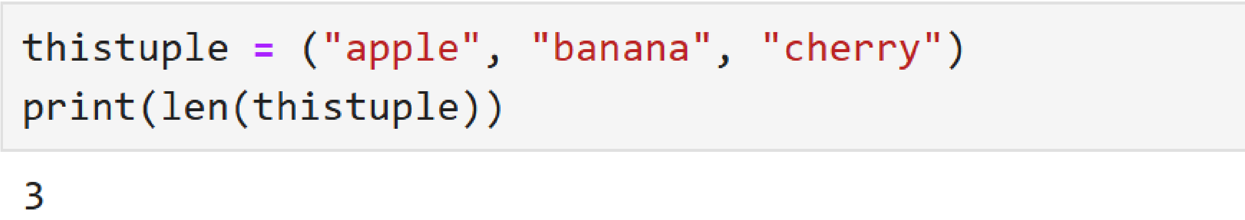


```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

```
Yes, 'apple' is in the fruits tuple
```

Tuple Length

To determine how many items a tuple has, use the len() method



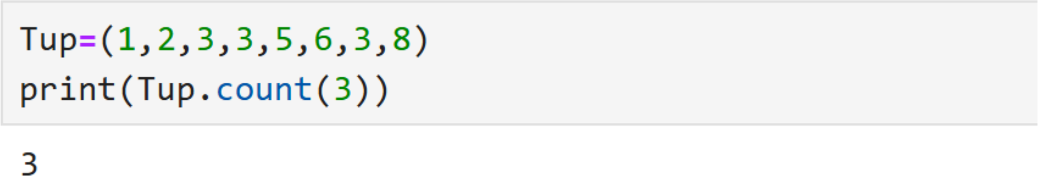
```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

```
3
```

Tuple Methods

Python has two built-in methods that you can use on tuples.

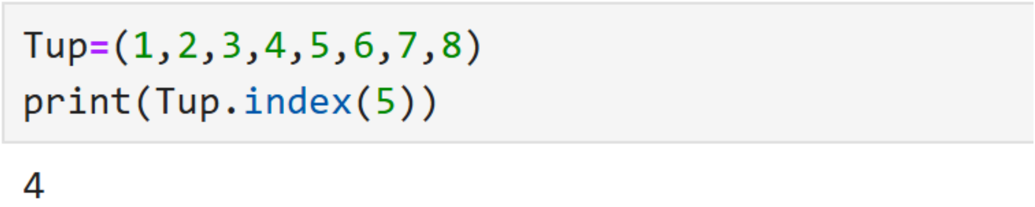
1. **count():** returns the number of times a specified value occurs in a tuple.



```
Tup=(1,2,3,3,5,6,3,8)
print(Tup.count(3))
```

```
3
```

2. **index():** Searches the tuple for a specified value and returns the position of where it was found.

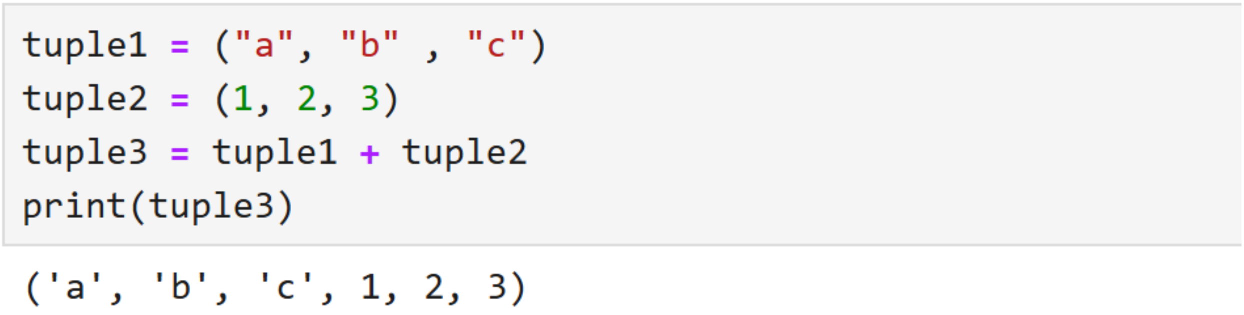


```
Tup=(1,2,3,4,5,6,7,8)
print(Tup.index(5))
```

```
4
```

Join Tuples

To join two or more tuples you can use the + operator:



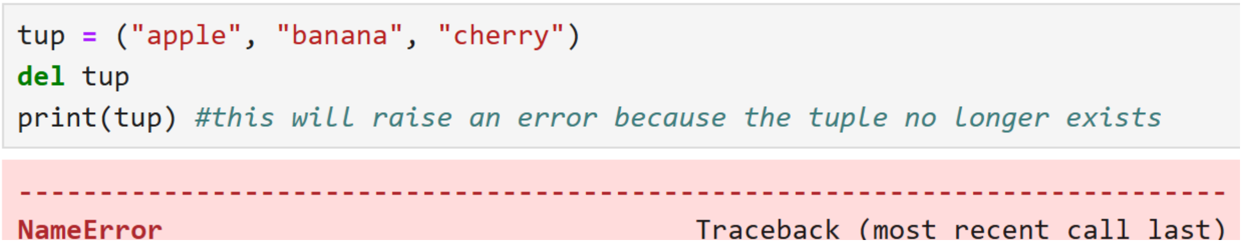
```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```

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

Remove Items in tuple

- Note: You cannot remove items in a tuple.
- Tuples are unchangeable, so you cannot remove items from it, but you can delete the tuple completely.

Example: The **del** keyword can delete the tuple completely:



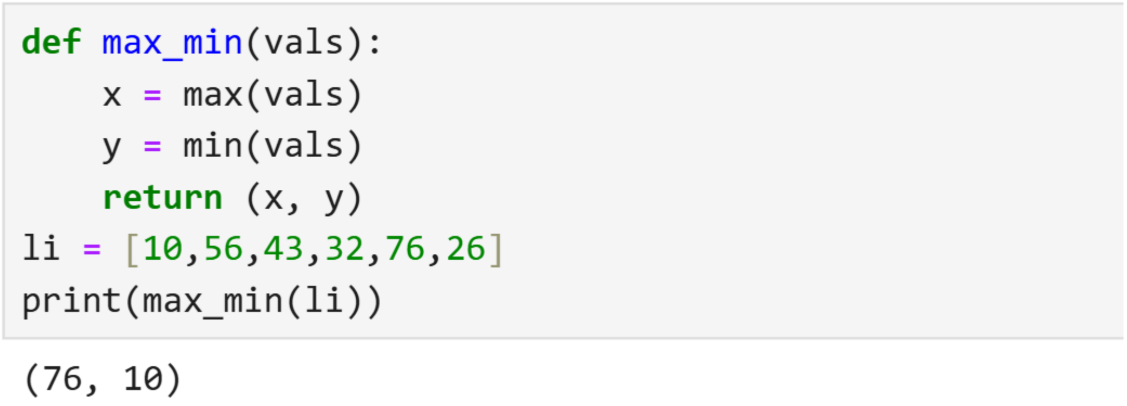
```
tup = ("apple", "banana", "cherry")
del tup
print(tup) #this will raise an error because the tuple no longer exists
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

Returning multiple values from function using tuple

A function can return only a single value. But when we need to return more than one value from a function, we group them as tuple and return.

Example:



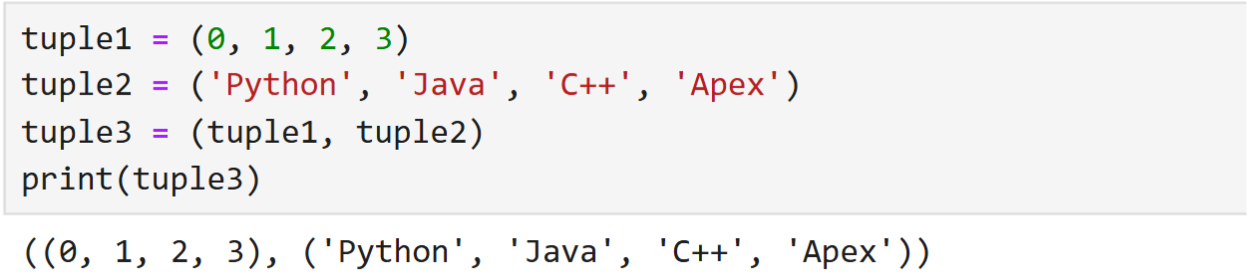
```python
def max_min(vals):
    x = max(vals)
    y = min(vals)
    return (x, y)
li = [10,56,43,32,76,26]
print(max_min(li))
```

```
(76, 10)
```

Nesting of Tuples

- Process of defining a tuple inside another tuple is called Nesting of tuple.

Example: Code for creating nested tuples

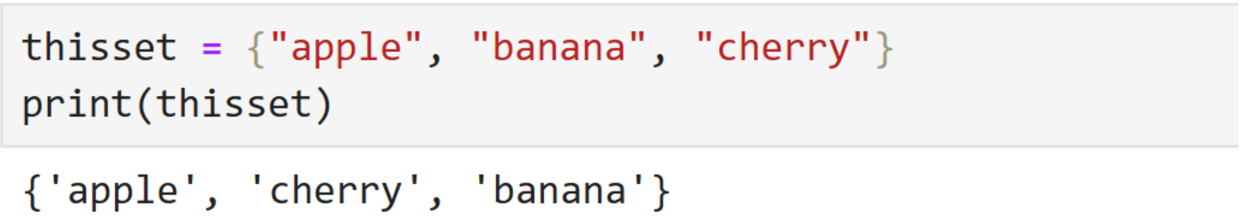


```python
tuple1 = (0, 1, 2, 3)
tuple2 = ('Python', 'Java', 'C++', 'Apex')
tuple3 = (tuple1, tuple2)
print(tuple3)
```

```
((0, 1, 2, 3), ('Python', 'Java', 'C++', 'Apex'))
```

Set

- A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets{ }.
- A set is mutable i.e. we can easily add or remove items from a set.
- Sets are the same as lists but with a difference that sets are lists with no duplicate entries and unordered.
- Once a set is created, you cannot change its items, but you can add new items.
- A set is created by placing all the elements inside curly brackets.



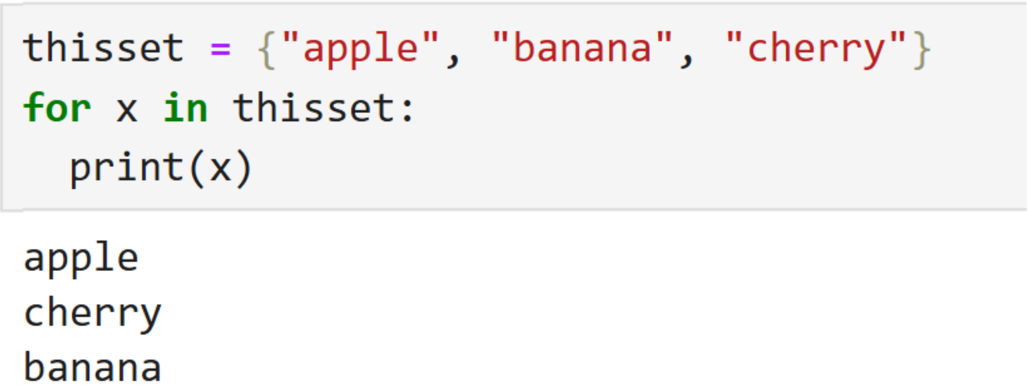
```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

```
{'apple', 'cherry', 'banana'}
```

Access Items in a Set

- You cannot access items in a set by referring to an index, since sets are unordered the items has no index.
- But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example: Loop through the set, and print the values:



```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

```
apple
cherry
banana
```

Set Methods

Method	Description	Example

add()	Adds an element to the set	*thisset = {"apple", "banana", "cherry"}*
		thisset.add('Mango')

clear()	Removes all the elements from the set	*thisset.clear()*

copy()	Returns a copy of the set	*thisset = {"apple", "banana", "cherry"}* *new_set = thisset.copy()*

difference()	Returns a set containing the difference between two or more sets	*set1 = {'Python', 'Java', 'C++'}*
		set2 = {'Python', '.Net', 'C'}
		set1.difference(set2)

difference_update()	Removes the items in this set that are also included in another, specified set	*set1 = {'Python', 'Java', 'C++'}*
		set2 = {'Python', '.Net', 'C'}
		set1.difference_update(set2)

discard()	Remove the specified item	*set1 = {'Python', 'Java', 'C++'}*
		set1.discard('C++')

intersection()	Returns a set, that is the intersection of two other sets	*set1 = {'Python', 'Java', 'C++'}*
		set2 = {'Python', '.Net', 'C'}
		set3 = set1.intersection(set2)

intersection_update()	Removes the items in this set that are not present in other, specified set(s)	*set1 = {'Python', 'Java', 'C++'}*
		set2 = {'Python', '.Net', 'C'}
		set1.intersection_update(set2)

isdisjoint()	returns True if none of the items are present in both sets, otherwise it returns False	set1 = {'Python', 'Java', 'C++'}
		set2 = {'Python', '.Net', 'C'}
		set1.isdisjoint(set2)

issubset()	Returns whether another set contains this set or not	*set1 = {'Python', 'Java', 'C++'}*
		set2 = {'Java', 'C++'}
		set2.issubset(set1)

issuperset()	Returns whether this set contains another set or not	set1 = {'Python', 'Java', 'C++'}
		set2 = {'Java', 'C++'}
		set1.issuperset(set2)

remove()	Removes the specified element. Throw error if element not present.	*set1 = {'Python', 'Java', 'C++'}*
		set1.remove('C++')

symmetric_difference()	This method returns a set that contains all items from both set, but not the items that	*set1 = {'Python', 'Java', 'C++'}*
		set2 = {'Python','.Net', 'C'}

	are present in both sets	*set3 = set1.symmetric_difference(set2)*

symmetric_difference_update()	Remove the items that are present in both sets, AND insert the items that is not present in both sets	*set1 = {'Python', 'Java', 'C++'}*
		set2 = {'Python','.Net', 'C'}
		set1.symmetric_difference_update(set2)

union()	Return a set containing the union of sets	*set1 = {'Python', 'Java', 'C++'}*
		set2 = {'Python','.Net', 'C'}
		set1.union(set2)

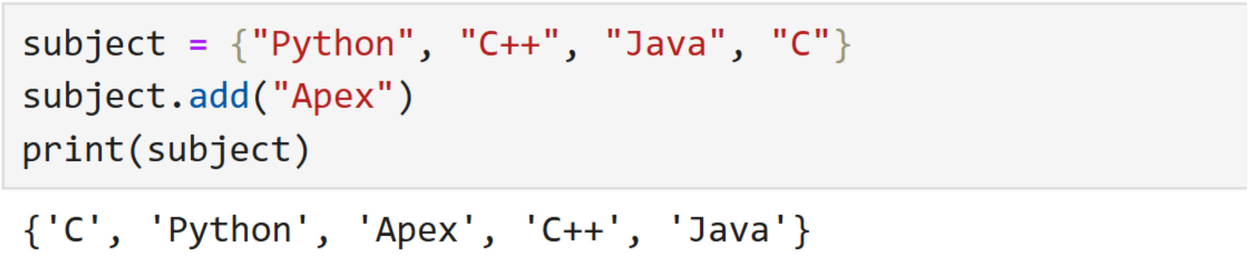
update()	Updates the current set, by adding items from another set (or any other iterable)	*set1 = {'Python', 'Java', 'C++'}*
		set2 = {'Python','.Net', 'C'}
		set1.update(set2)

pop()	Removes an element from the set	*set1 = {'Python', 'Java', 'C++', 'C', 'Apex'}*
		set1.pop()

Add Items to Set

- To add one item to a set use the add() method.
- To add more than one item to a set use the update() method.

Example: Add an item to a set, using the add() method.



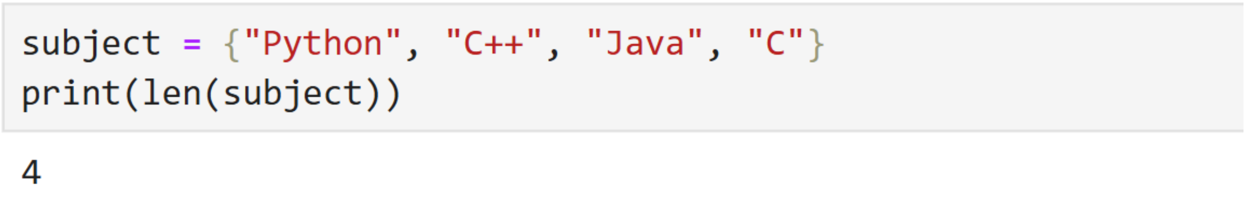
```
subject = {"Python", "C++", "Java", "C"}
subject.add("Apex")
print(subject)
```

```
{'C', 'Python', 'Apex', 'C++', 'Java'}
```

Get the Length of a Set

- To determine how many items a set has, use the len() method.

Example: Get the number of items in a set.



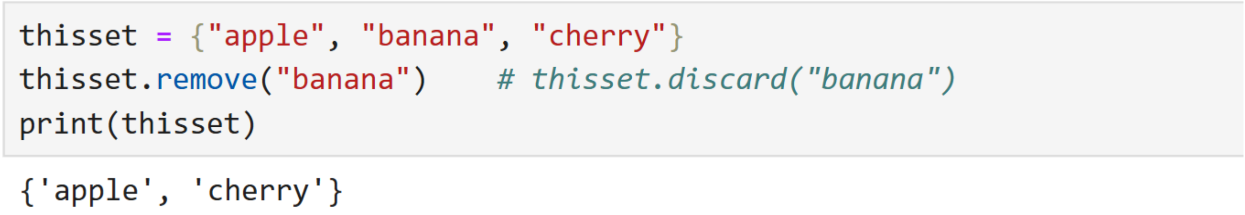
```
subject = {"Python", "C++", "Java", "C"}
print(len(subject))
```

```
4
```

Remove Item from a Set

- To remove an item in a set, use the remove(), or the discard() method.

Example: Remove "banana" by using the remove() method:



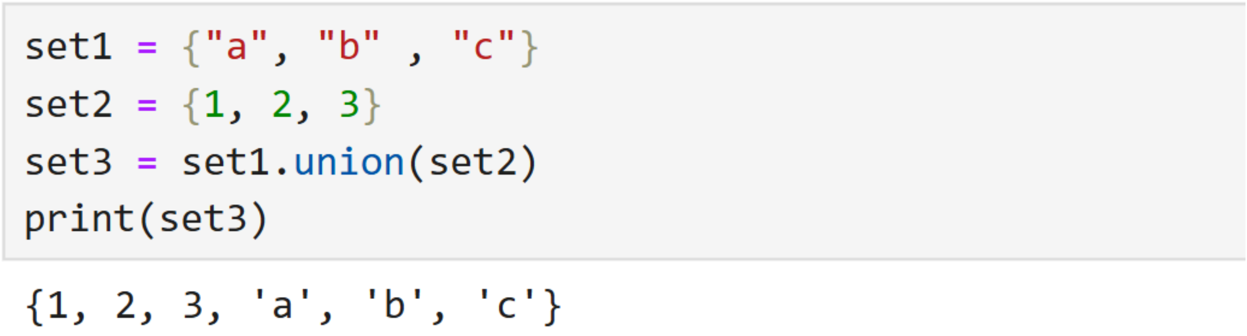
```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")    # thisset.discard("banana")
print(thisset)
```

```
{'apple', 'cherry'}
```

Join two Sets

You can use the **union()** method that returns a new set containing all items from both sets.

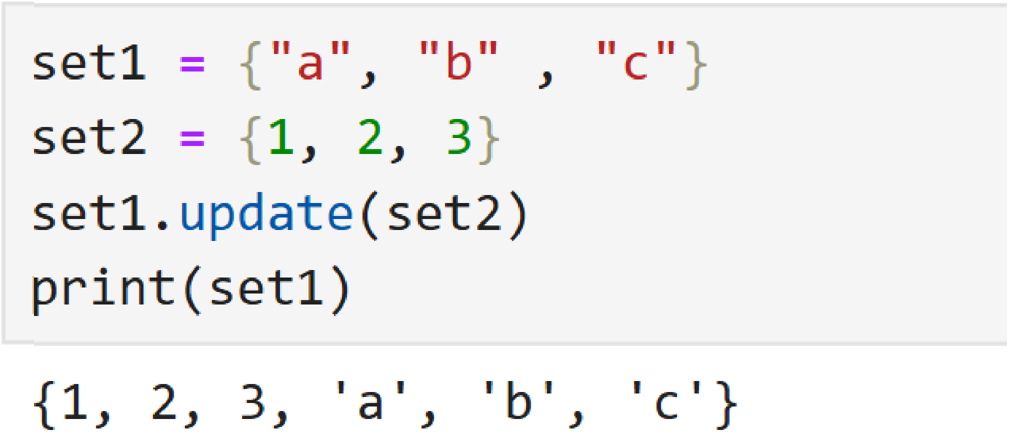
Example: The union() method returns a new set with all items from both sets



```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

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

Example: The **update()** method inserts the items in set2 into set



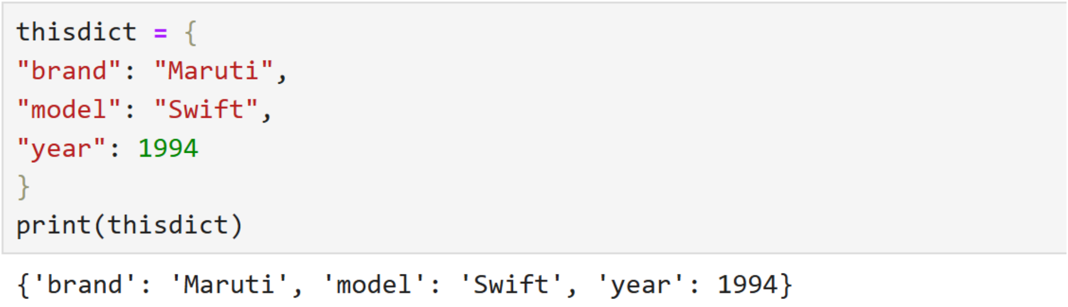
```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

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

Dictionary

- A dictionary is a collection, which is unordered, changeable and indexed.
- In Python dictionaries are written with curly brackets, and they have keys and values.
- Each key is separated from its value by a colon (:) and consecutive items are separated by commas.
- Entire items in the dictionary are enclosed in curly brackets ({}).

Create and print a dictionary

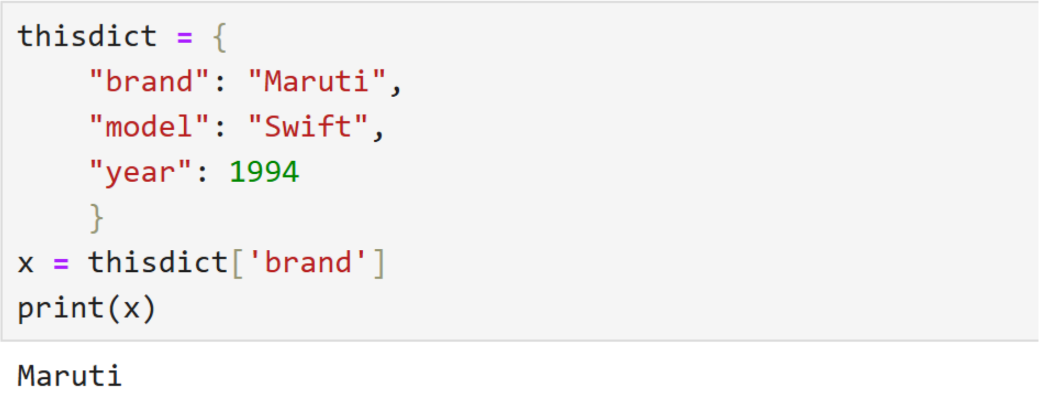


```
thisdict = {
"brand": "Maruti",
"model": "Swift",
"year": 1994
}
print(thisdict)
```

```
{'brand': 'Maruti', 'model': 'Swift', 'year': 1994}
```

Accessing Items in Dictionaries

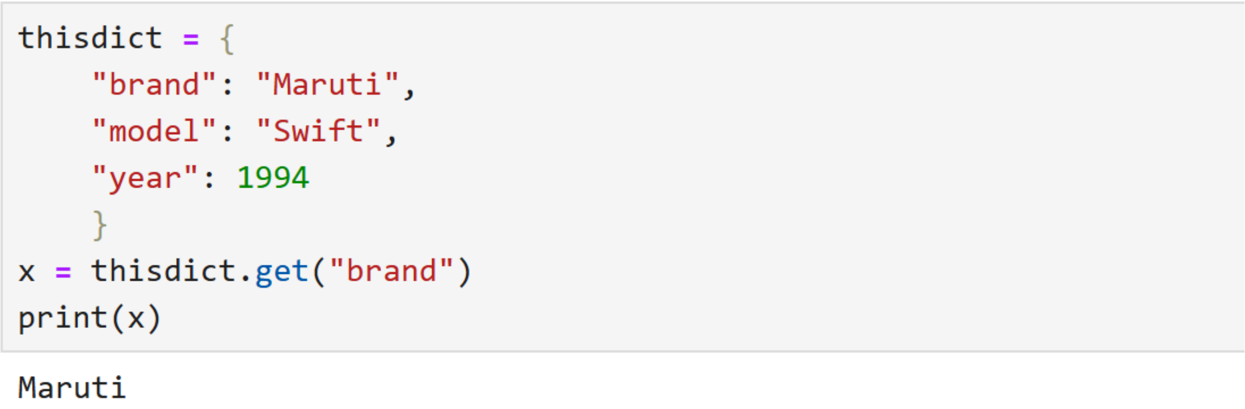
1. You can access the items of a dictionary by referring to its **key name, inside square brackets.**



```
thisdict = {
    "brand": "Maruti",
    "model": "Swift",
    "year": 1994
    }
x = thisdict['brand']
print(x)
```

```
Maruti
```

2. There is also a method called **get()** that will give you the same result.

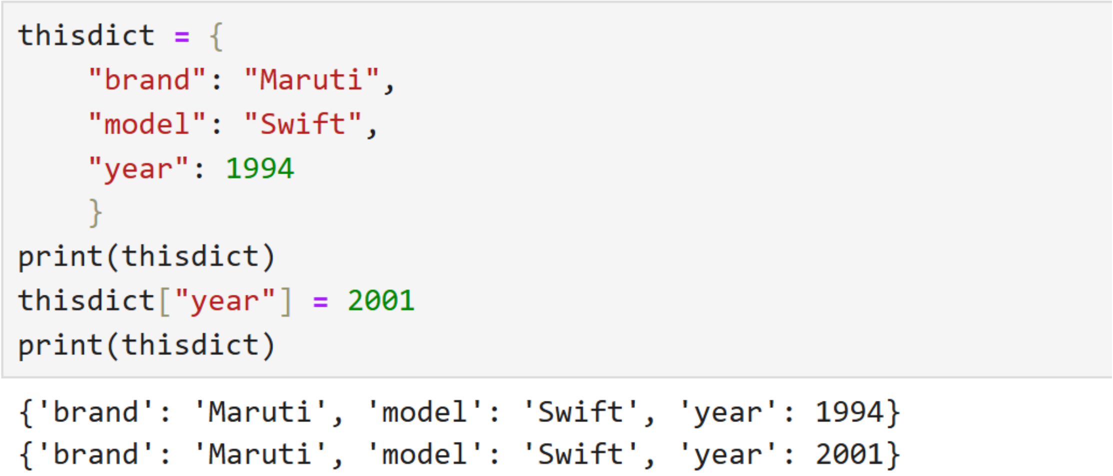


```
thisdict = {
    "brand": "Maruti",
    "model": "Swift",
    "year": 1994
    }
x = thisdict.get("brand")
print(x)
```

```
Maruti
```

Modifying an item in Dictionary

Example: Change the “year” to 2001.

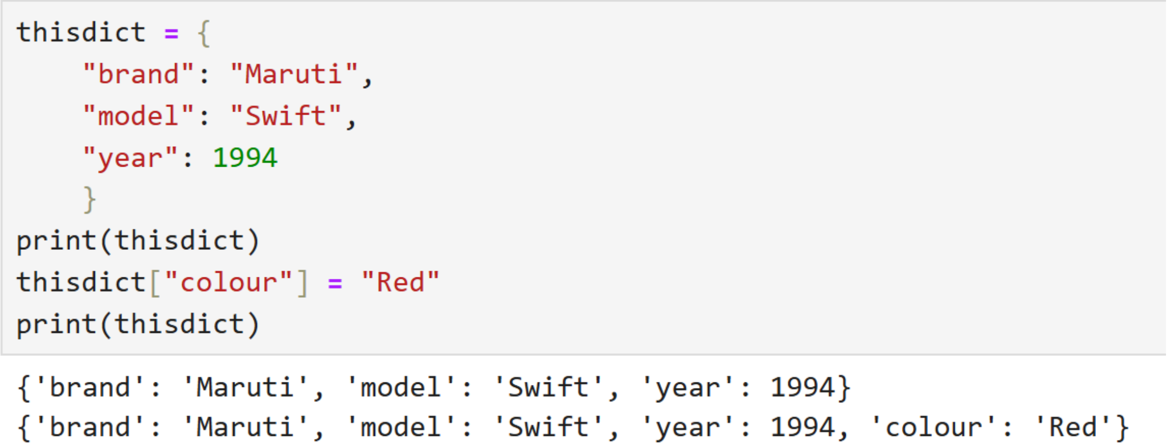


```
thisdict = {
    "brand": "Maruti",
    "model": "Swift",
    "year": 1994
    }
print(thisdict)
thisdict["year"] = 2001
print(thisdict)
```

```
{'brand': 'Maruti', 'model': 'Swift', 'year': 1994}
{'brand': 'Maruti', 'model': 'Swift', 'year': 2001}
```

Adding Items to Dictionary

Example:



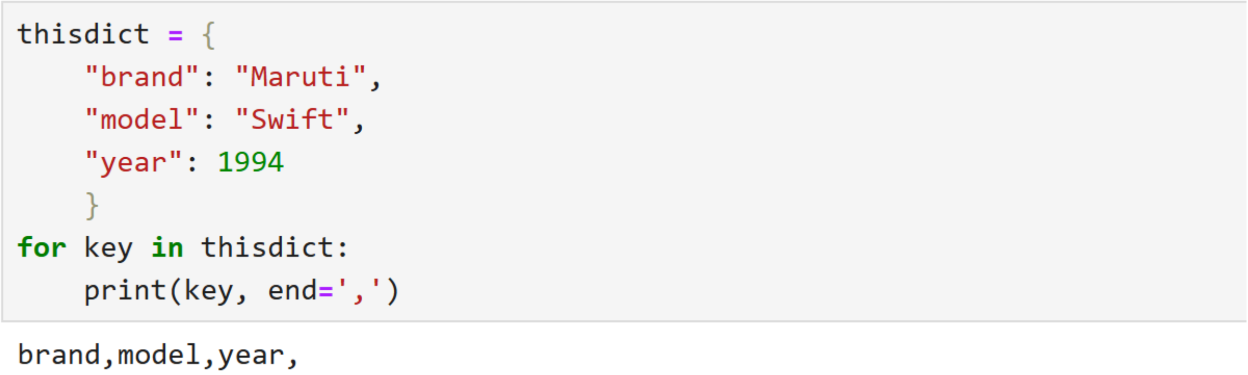
```
thisdict = {
    "brand": "Maruti",
    "model": "Swift",
    "year": 1994
    }
print(thisdict)
thisdict["colour"] = "Red"
print(thisdict)
```

```
{'brand': 'Maruti', 'model': 'Swift', 'year': 1994}
{'brand': 'Maruti', 'model': 'Swift', 'year': 1994, 'colour': 'Red'}
```

Loop Through a Dictionary

- You can loop through a dictionary by using a for loop.
- When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

Example: Print all key names in the dictionary, one by one.



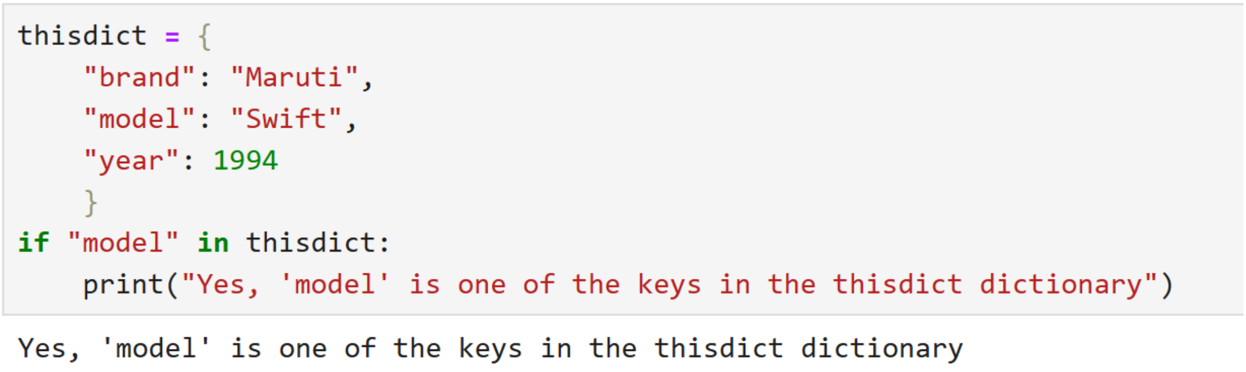
```
thisdict = {
    "brand": "Maruti",
    "model": "Swift",
    "year": 1994
    }
for key in thisdict:
    print(key, end=',')
```

```
brand,model,year,
```

Check if Key Exists

- To determine if a specified key is present in a dictionary use the **in** keyword.

Example: Check if "model" is present in the dictionary.



```
thisdict = {
    "brand": "Maruti",
    "model": "Swift",
    "year": 1994
    }
if "model" in thisdict:
    print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

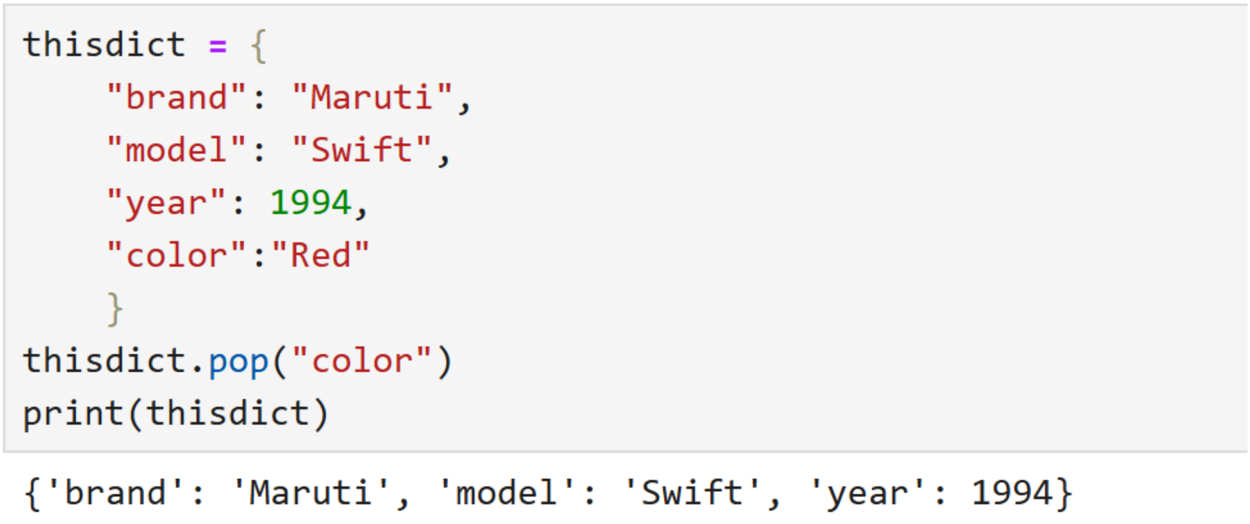
Removing Items from Dictionaries

There are following methods to remove items from a dictionary.

1. The **pop()** method removes the item with the specified key name.
2. The **del** keyword removes the item with the specified key name.
3. The del keyword can also delete the dictionary completely.
4. The **clear()** method empties the dictionary.

1. The pop() method removes the item with the specified key name.

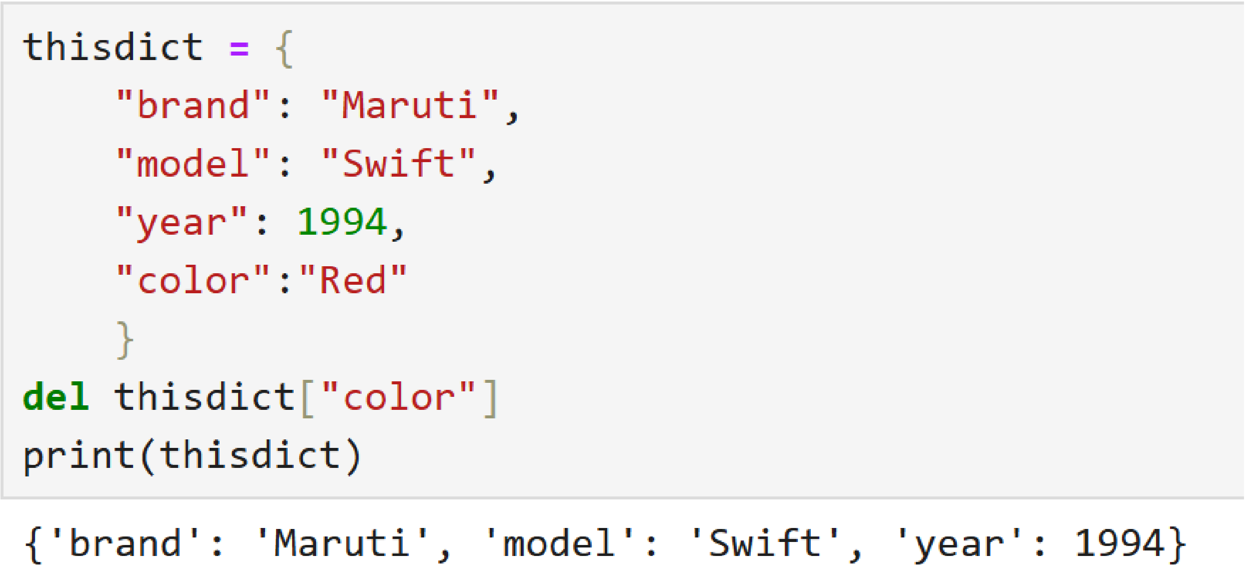
Example:



```
thisdict = {
    "brand": "Maruti",
    "model": "Swift",
    "year": 1994,
    "color":"Red"
    }
thisdict.pop("color")
print(thisdict)
```

```
{'brand': 'Maruti', 'model': 'Swift', 'year': 1994}
```

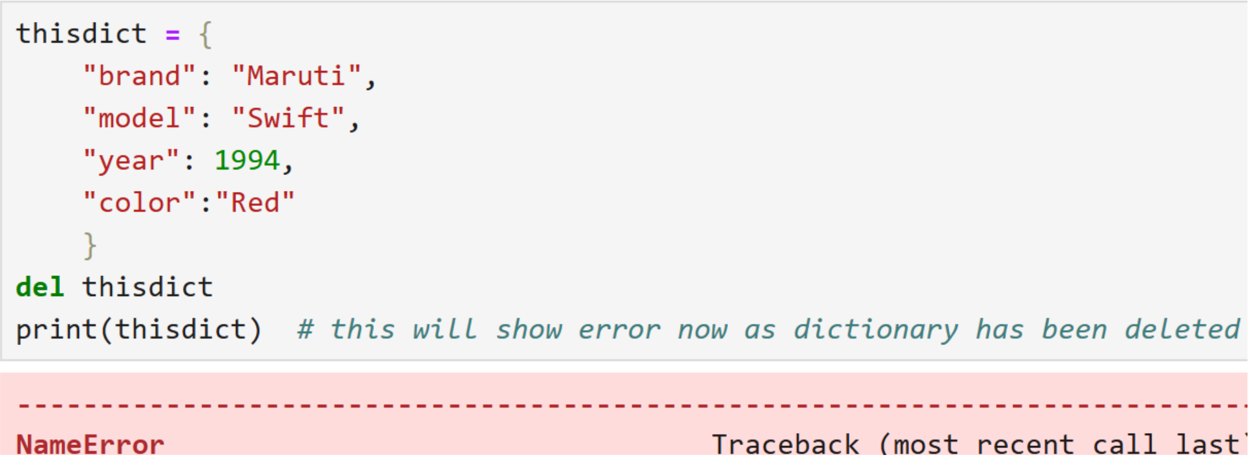
2. The del keyword removes the item with the specified key name.



```
thisdict = {
    "brand": "Maruti",
    "model": "Swift",
    "year": 1994,
    "color":"Red"
    }
del thisdict["color"]
print(thisdict)
```

```
{'brand': 'Maruti', 'model': 'Swift', 'year': 1994}
```

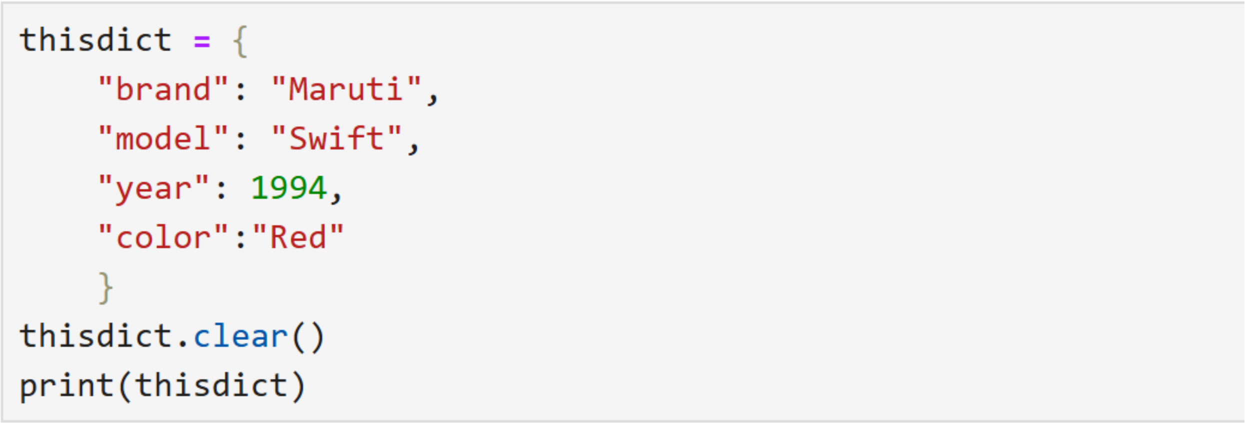
3. The del keyword can also delete the dictionary completely.



```
thisdict = {
    "brand": "Maruti",
    "model": "Swift",
    "year": 1994,
    "color":"Red"
    }
del thisdict
print(thisdict)  # this will show error now as dictionary has been deleted
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last
```

4. The clear() method will not delete but it empties the dictionary.



```
thisdict = {
    "brand": "Maruti",
    "model": "Swift",
    "year": 1994,
    "color":"Red"
    }
thisdict.clear()
print(thisdict)
```

```
{}
```

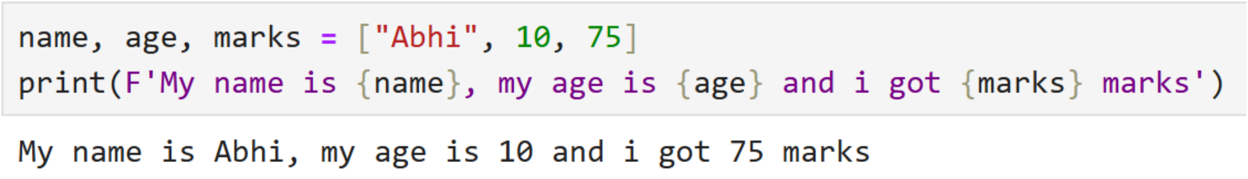
Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary **Syntax**: dictionary.clear( )
copy()	Returns a copy of the dictionary **Syntax**: x=dictionary.copy( )
fromkeys()	Returns a dictionary with the specified keys and value **Syntax**: x=dict.fromkeys(keys, value) **Example:** *x = (1,2,3,4)* *y = "value"* *my_dict = dict.fromkeys(x,y)*
get()	Returns the value of the specified key **Syntax**: dictionary.get(keyname)
items()	Returns a list containing a tuple for each key value pair **Syntax**: x = dictionary.items()

keys()	Returns a list containing the dictionary's keys **Syntax**: x = dictionary.keys()
pop()	Removes the element with the specified key **Syntax:** dictionary.pop(keyname)
popitem()	Removes the last inserted key-value pair **Syntax:** dictionary.popitem()
update()	Updates the dictionary with the specified key-value pairs **Syntax:** dictionary.update({key:value})
values()	Returns a list of all the values in the dictionary **Syntax:** dictionary.values()
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value. The value is optional. If the key exists, this parameter has no effect. **Syntax:** dictionary.setdefault(keyname, value) **Example:** *thisdict = {* *"brand": "Maruti",* *"model": "Swift",* *"year": 1994,* *"color":"Red"* *}* *thisdict.setdefault("color", "White")* *print(thisdict)*

Unpacking Sequences

- Sequence unpacking in python allows you to take objects in a collection and store them in variables for later use.
- A key feature of python is that any sequence can be unpacked into variables by assignment.



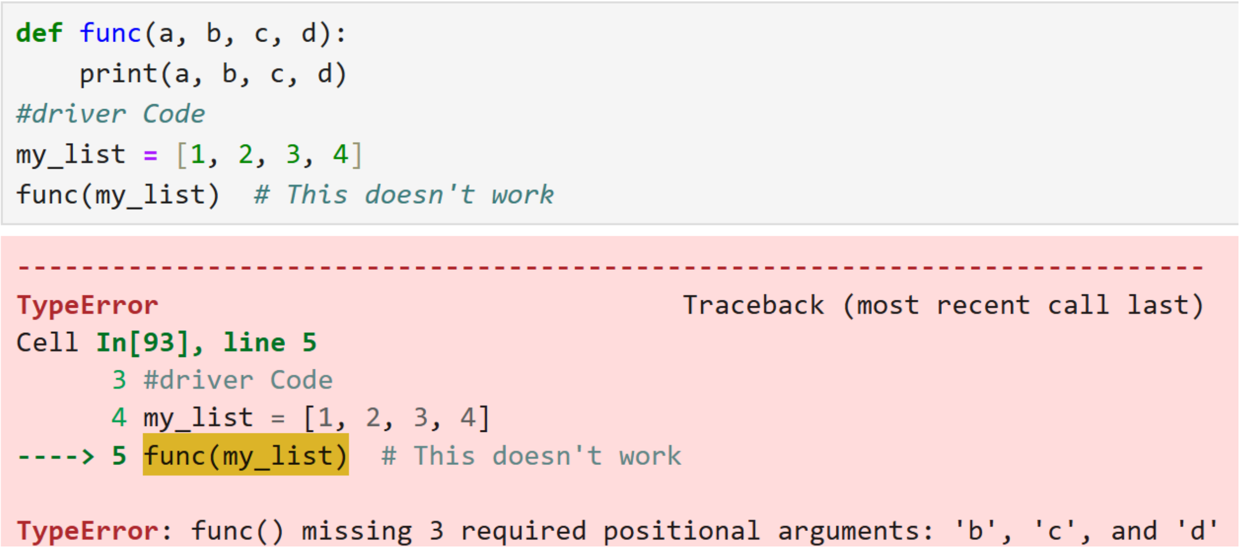
```
name, age, marks = ["Abhi", 10, 75]
print(F'My name is {name}, my age is {age} and i got {marks} marks')
```

```
My name is Abhi, my age is 10 and i got 75 marks
```

Need of Unpacking

Let we have a function that receives three arguments. Also we have a list of size 3 that has all arguments for the function. When we make call to this function and simply pass list to the function, the call doesn’t work.

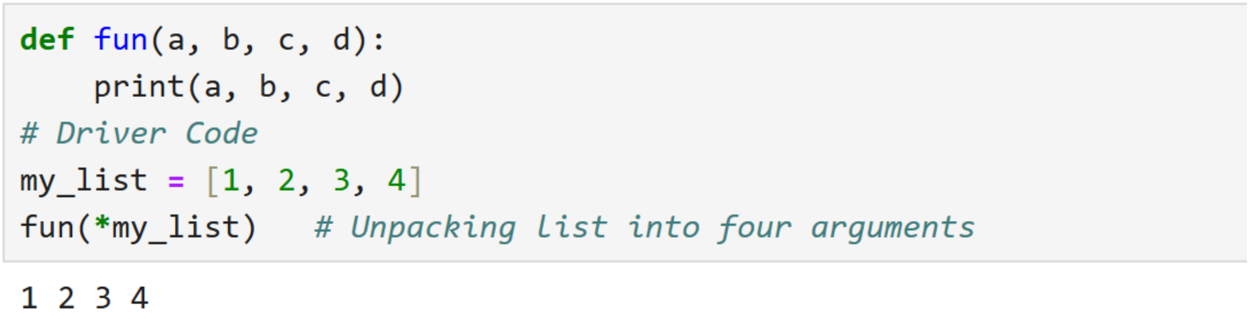
A sample function that takes 4 arguments and prints them.



```
def func(a, b, c, d):
    print(a, b, c, d)
#driver Code
my_list = [1, 2, 3, 4]
func(my_list)  # This doesn't work
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[93], line 5
      3 #driver Code
      4 my_list = [1, 2, 3, 4]
----> 5 func(my_list)  # This doesn't work

TypeError: func() missing 3 required positional arguments: 'b', 'c', and 'd'
```

Solution to above problem is Unpacking: We can use * to unpack the list so that all elements of it can be passed as different parameters.



```
def fun(a, b, c, d):
    print(a, b, c, d)
# Driver Code
my_list = [1, 2, 3, 4]
fun(*my_list)   # Unpacking list into four arguments
```

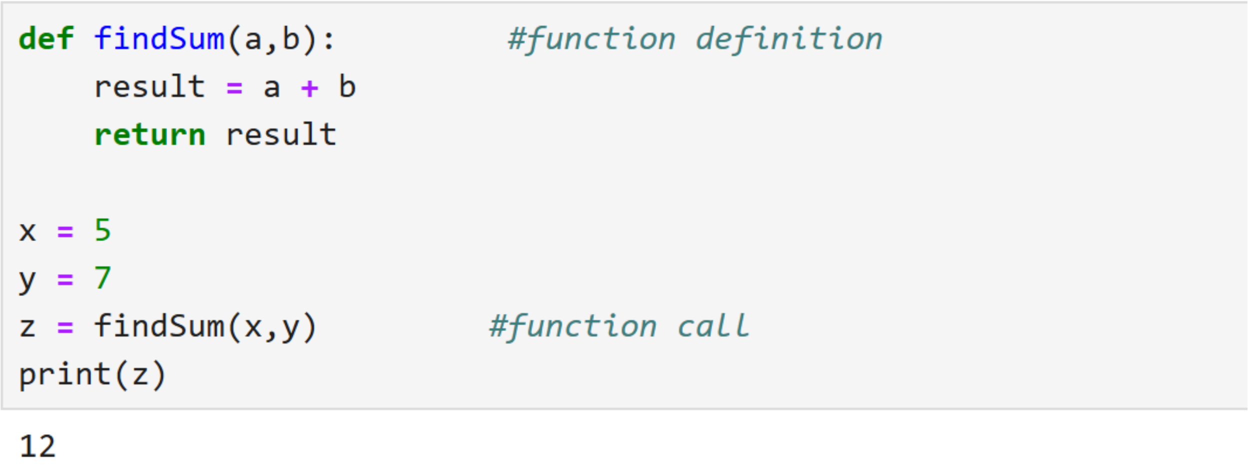
```
1 2 3 4
```

Function

- A function is a block of organized, reusable code that is used to perform a single, related action.
- Functions provide better modularity for your application and a high degree of code reusing.
- There are two types of functions in Python: Built-in functions and User defined functions
- Python gives many built in functions like print(), len() etc. But we can also create our own functions. These functions are called user defined functions.

Creating a Function

Function for addition of two numbers:



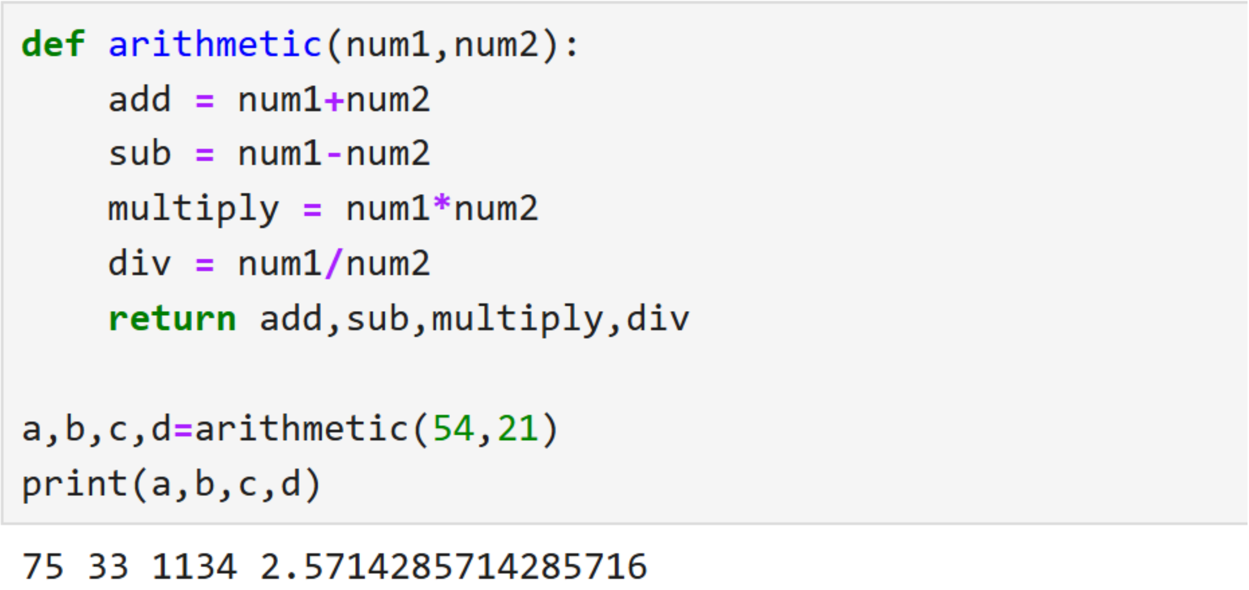
```
def findSum(a,b):          #function definition
    result = a + b
    return result

x = 5
y = 7
z = findSum(x,y)          #function call
print(z)
```

```
12
```

Note: a,b are called formal arguments while x, y are called actual arguments.

Return Multiple Values

You can also return multiple values from a function. Use the return statement by separating each expression by a comma.



```
def arithmetic(num1,num2):
    add = num1+num2
    sub = num1-num2
    multiply = num1*num2
    div = num1/num2
    return add,sub,multiply,div

a,b,c,d=arithmetic(54,21)
print(a,b,c,d)
```

```
75 33 1134 2.5714285714285716
```

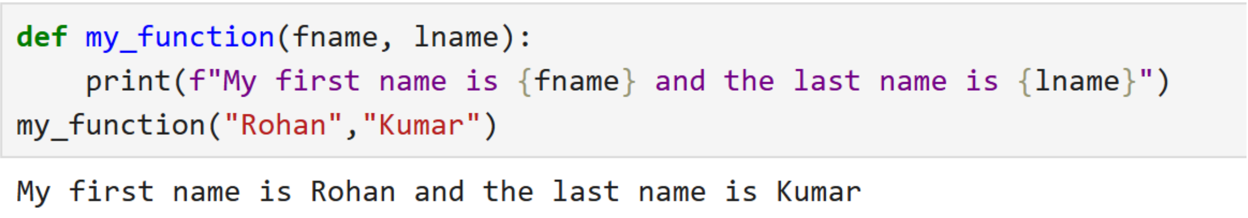
Arguments(or Parameters) in Function

There are the following types of arguments in Python.

1. positional arguments
2. keyword arguments
3. default arguments
4. Variable-length positional arguments
5. Variable-length keyword arguments

Positional Arguments

- During function call, values passed through arguments should be in the order of parameters in the function definition. This is called positional arguments.
- By default, a function must be called with the correct number of arguments.
- Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

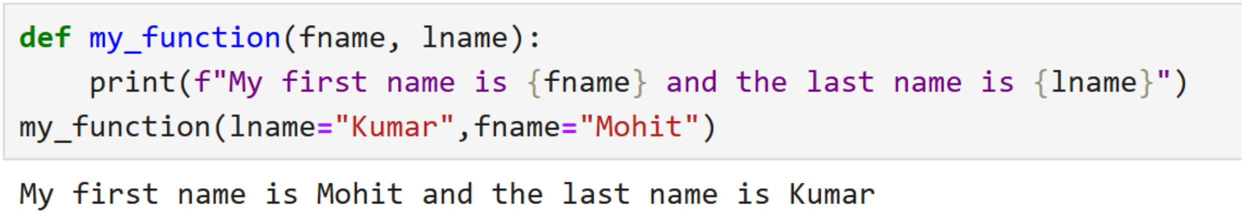


```
def my_function(fname, lname):
    print(f"My first name is {fname} and the last name is {lname}")
my_function("Rohan","Kumar")
```

```
My first name is Rohan and the last name is Kumar
```

keyword arguments

- We can also send arguments with the key = value syntax.
- This way the order of the arguments does not matter.
- Keyword arguments are related to the function calls
- The caller identifies the arguments by the parameter name.
- This allows to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

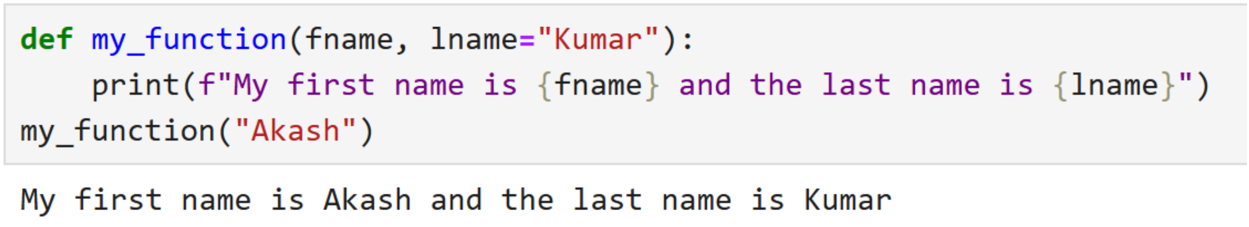


```python
def my_function(fname, lname):
    print(f"My first name is {fname} and the last name is {lname}")
my_function(lname="Kumar",fname="Mohit")
```

```
My first name is Mohit and the last name is Kumar
```

Default Arguments

- Default arguments are values that are provided while defining functions.
- The assignment operator = is used to assign a default value to the argument.
- Default arguments become optional during the function calls.
- If we provide value to the default arguments during function calls, it overrides the default value.
- Default arguments should follow non-default arguments.
- If we call the function without argument, it uses the default value.



```python
def my_function(fname, lname="Kumar"):
    print(f"My first name is {fname} and the last name is {lname}")
my_function("Akash")
```

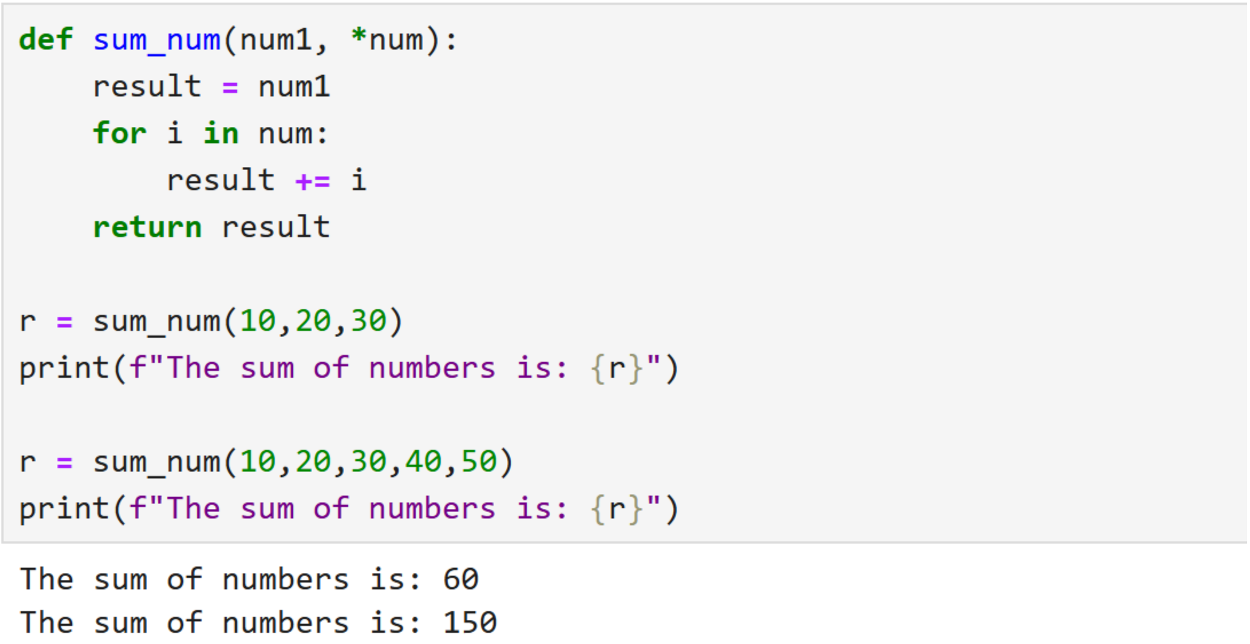
```
My first name is Akash and the last name is Kumar
```

Variable-length positional arguments

- Variable-length arguments are also known as Arbitrary arguments.
- If we don't know the number of arguments needed for the function in advance, we can use arbitrary arguments
- For arbitrary positional argument, an asterisk (*) is placed before a parameter in function definition which can hold non-keyword variable-length arguments.
- These arguments will be wrapped up in a tuple. Before the variable number of arguments, zero or more normal arguments may occur.

  Syntax:

```
def functionname(*var_args_tuple ):
        function_statements
        return [expression]
```

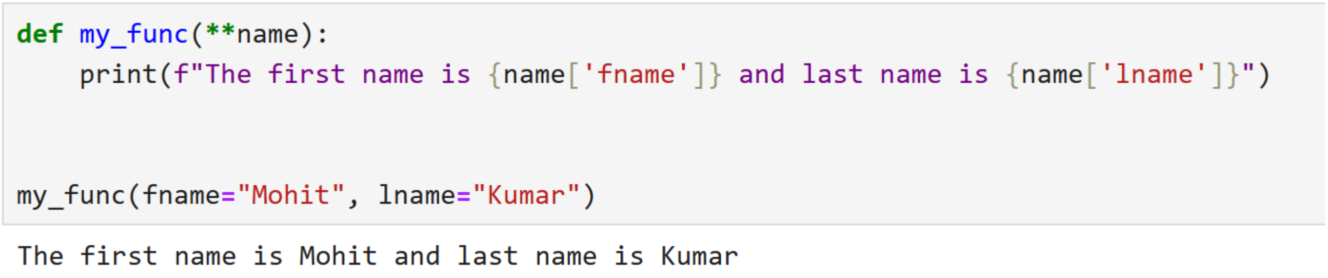


```
def sum_num(num1, *num):
    result = num1
    for i in num:
        result += i
    return result

r = sum_num(10,20,30)
print(f"The sum of numbers is: {r}")

r = sum_num(10,20,30,40,50)
print(f"The sum of numbers is: {r}")
```

```
The sum of numbers is: 60
The sum of numbers is: 150
```

Variable length Keyword Arguments

- If you do not know how many keyword arguments that will be passed into your function, add two asterisk (**) before the parameter name in the function definition.
- This way the function will receive a dictionary of arguments, and can access the items accordingly.



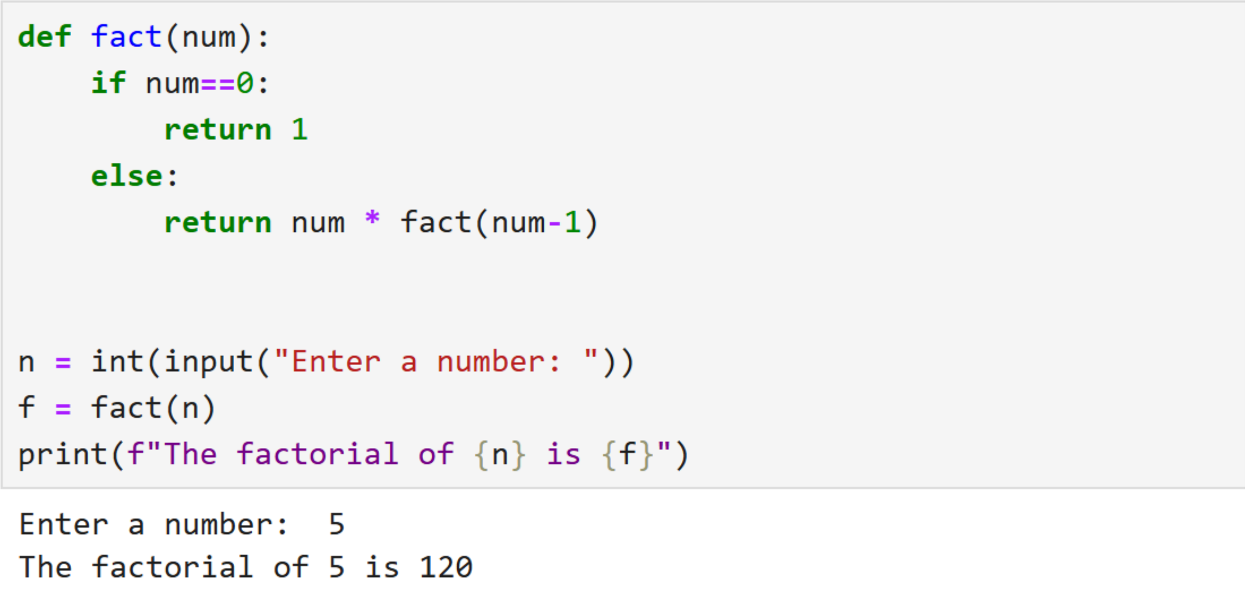
```
def my_func(**name):
    print(f"The first name is {name['fname']} and last name is {name['lname']}")


my_func(fname="Mohit", lname="Kumar")
```

```
The first name is Mohit and last name is Kumar
```

Recursion

- A recursive function is a function that calls itself, again and again.
- It means that the function will continue to call itself and repeat its behaviour until some condition is met to return a result.
- All recursive functions share a common structure made up of two parts: base case and recursive case

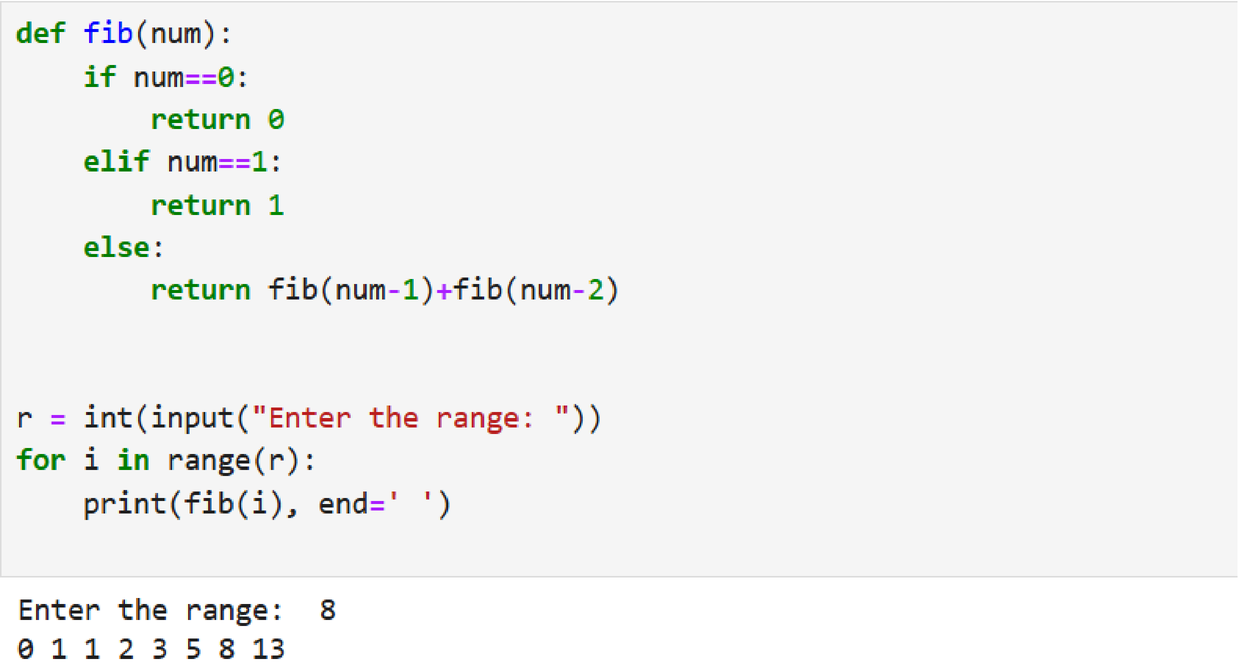
Example: Calculate the factorial of a number using recursive function.



```python
def fact(num):
    if num==0:
        return 1
    else:
        return num * fact(num-1)


n = int(input("Enter a number: "))
f = fact(n)
print(f"The factorial of {n} is {f}")
```

```
Enter a number:  5
The factorial of 5 is 120
```

Example: Print the fibonacci series using recursive function.



```
def fib(num):
    if num==0:
        return 0
    elif num==1:
        return 1
    else:
        return fib(num-1)+fib(num-2)


r = int(input("Enter the range: "))
for i in range(r):
    print(fib(i), end=' ')
```

```
Enter the range:  8
0 1 1 2 3 5 8 13
```

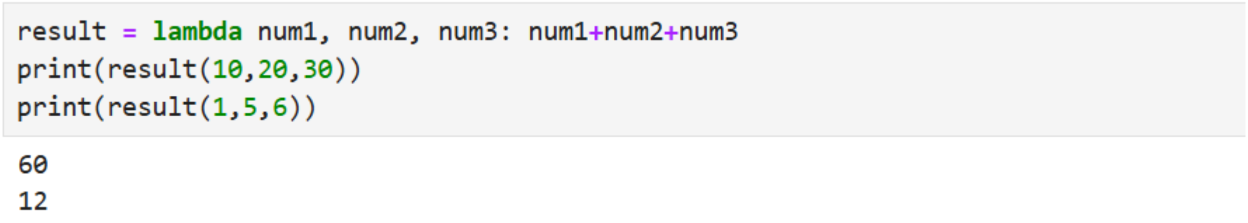
Anonymous/Lambda Function

- Sometimes we need to declare a function without any name. The nameless property function is called an anonymous function or lambda function.
- These functions are called anonymous because they are not declared in the standard manner by using the def keyword
- lambda keyword is used to declare the anonymous function.
- Lambda forms can take any number of arguments but return just one value in the form of an expression.
- Cannot access variables other than those in their parameter list and those in the global namespace.

  Syntax:

  lambda arg1 ,arg2,.....argn: expression

Example: Program to find the sum of three numbers.



```
result = lambda num1, num2, num3: num1+num2+num3
print(result(10,20,30))
print(result(1,5,6))
```

```
60
12
```

- We use lambda functions when we require a nameless function for a short period of time.
- Lambda functions are used along with built-in functions like filter(), map() etc.

filter() Function

The filter() function returns an iterator where the items are filtered through a function to test if the item is accepted or not.

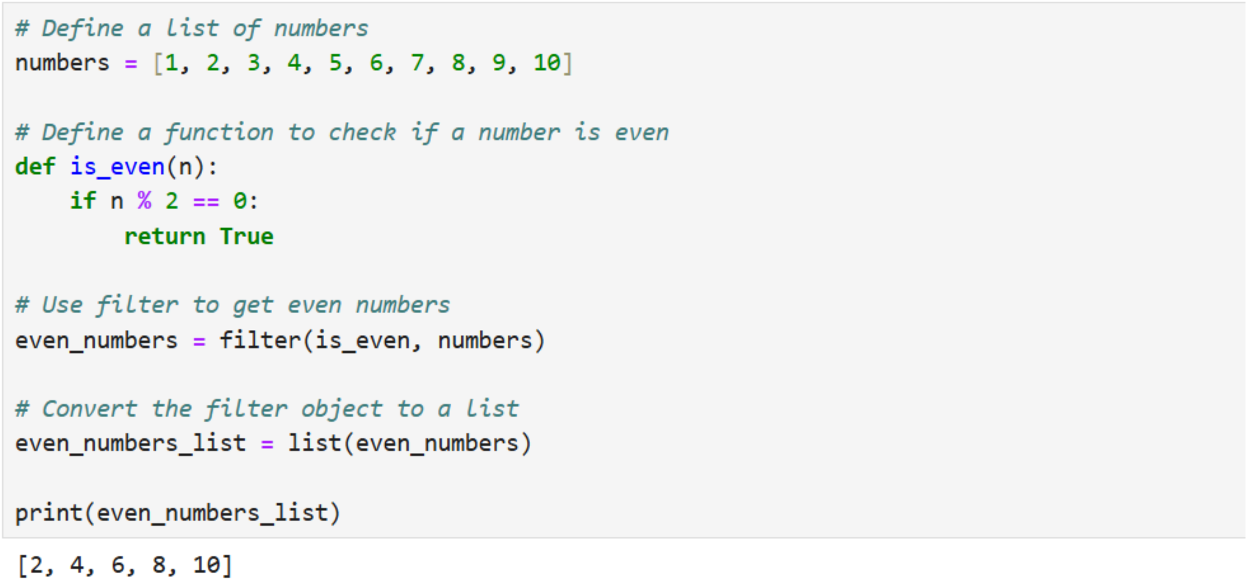
Syntax:

filter(function, sequence)

function – Function argument is responsible for performing condition checking.

sequence – Sequence argument can be anything like list, tuple, string

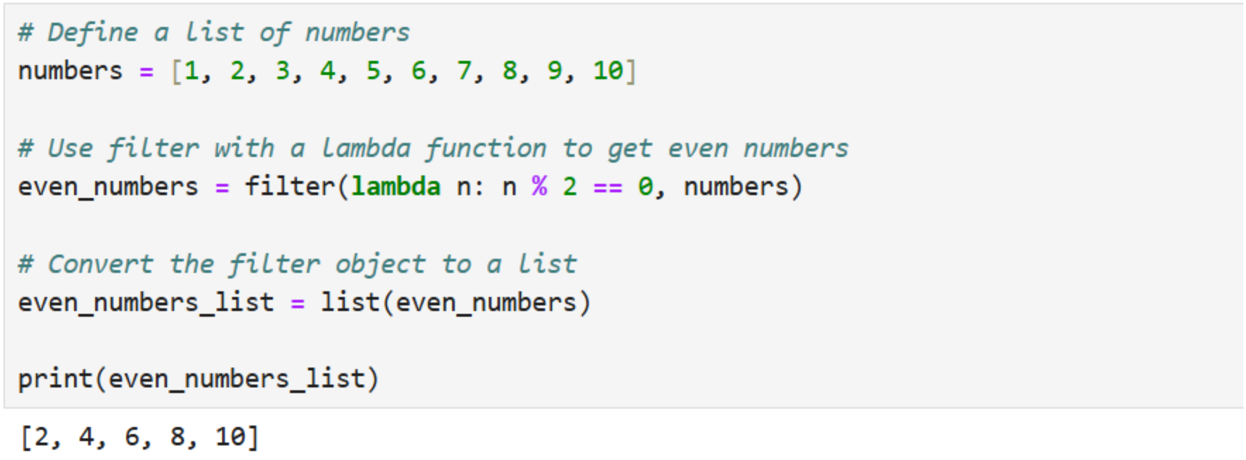
Example 1: Find even numbers from a list.



```
# Define a list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Define a function to check if a number is even
def is_even(n):
    if n % 2 == 0:
        return True

# Use filter to get even numbers
even_numbers = filter(is_even, numbers)

# Convert the filter object to a list
even_numbers_list = list(even_numbers)

print(even_numbers_list)
```

```
[2, 4, 6, 8, 10]
```

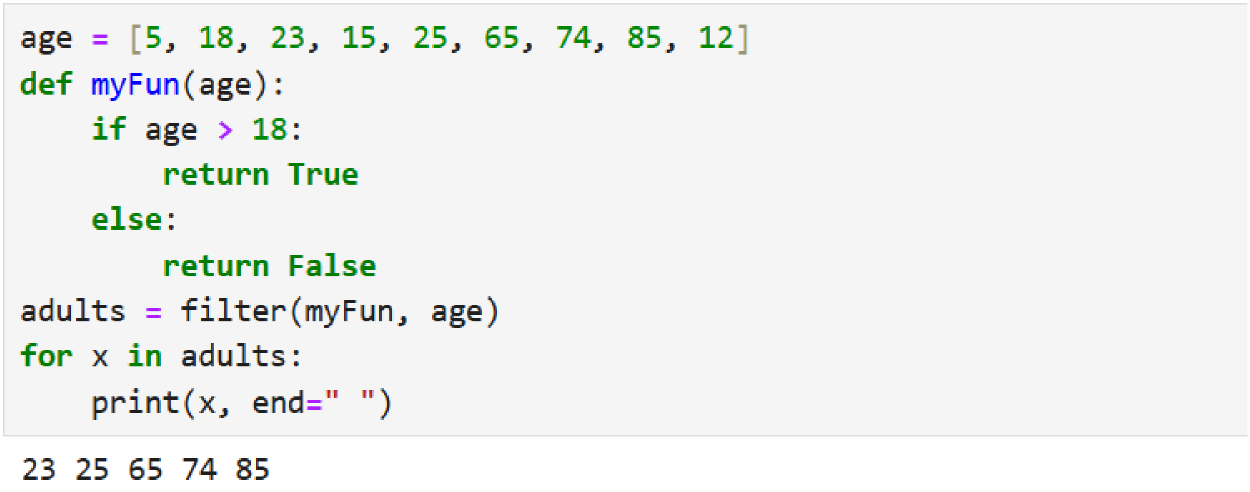
Example 2: Above problem using lambda function



```
# Define a list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Use filter with a lambda function to get even numbers
even_numbers = filter(lambda n: n % 2 == 0, numbers)

# Convert the filter object to a list
even_numbers_list = list(even_numbers)

print(even_numbers_list)
```

```
[2, 4, 6, 8, 10]
```

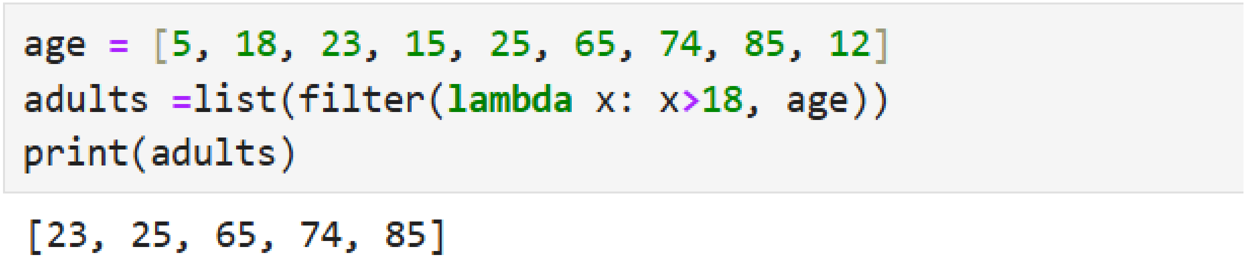
Example 3:



```
age = [5, 18, 23, 15, 25, 65, 74, 85, 12]
def myFun(age):
    if age > 18:
        return True
    else:
        return False
adults = filter(myFun, age)
for x in adults:
    print(x, end=" ")
```

```
23 25 65 74 85
```

Example 4: Same as example 3 but using lambda function



```
age = [5, 18, 23, 15, 25, 65, 74, 85, 12]
adults =list(filter(lambda x: x>18, age))
print(adults)
```

```
[23, 25, 65, 74, 85]
```

map() Function

- The map() function is used to apply some functionality for every element present in the given sequence and generate a new series with a required modification.

- The map() function in Python takes in a function and a list.
- The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

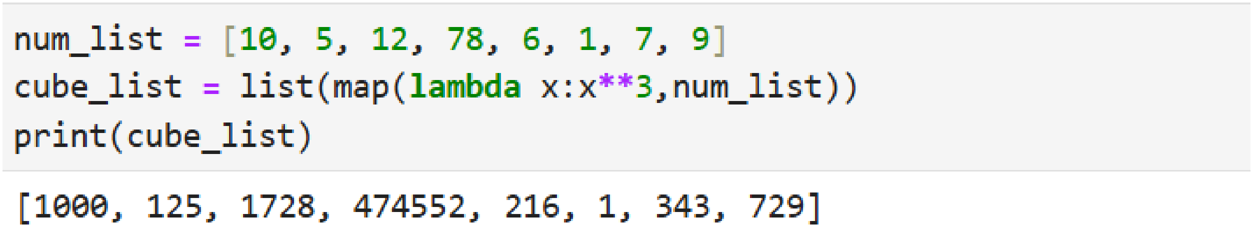
Syntax:

map(function, sequence)

function – function argument responsible for applied on each element of the sequence

sequence – Sequence argument can be anything like list, tuple, string

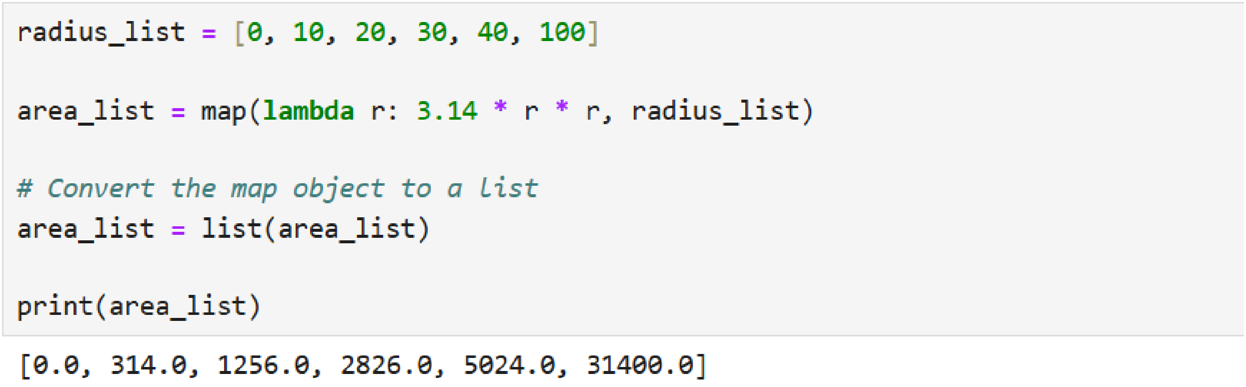
Example: Program to create a new list of cubes of each item of an existing list of Integers using map().



```
num_list = [10, 5, 12, 78, 6, 1, 7, 9]
cube_list = list(map(lambda x:x**3,num_list))
print(cube_list)
```

```
[1000, 125, 1728, 474552, 216, 1, 343, 729]
```

Example: Finding Area of circles whose radii are in a list.



```
radius_list = [0, 10, 20, 30, 40, 100]

area_list = map(lambda r: 3.14 * r * r, radius_list)

# Convert the map object to a list
area_list = list(area_list)

print(area_list)
```

```
[0.0, 314.0, 1256.0, 2826.0, 5024.0, 31400.0]
```

reduce() Function

- The reduce() function is used to minimize sequence elements into a single value by applying the specified condition.
- The reduce() function is present in the functools module. Hence, we need to import it using the import statement before using it.

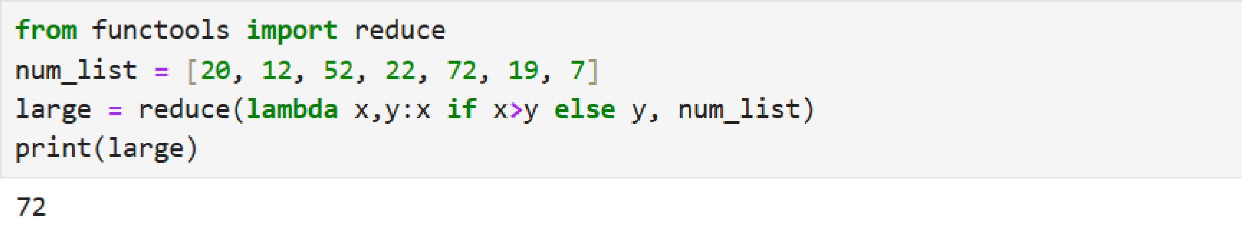
Syntax:

reduce(function, sequence)

function – function argument responsible for applied on each element of the sequence

sequence – Sequence argument can be anything like list, tuple, string

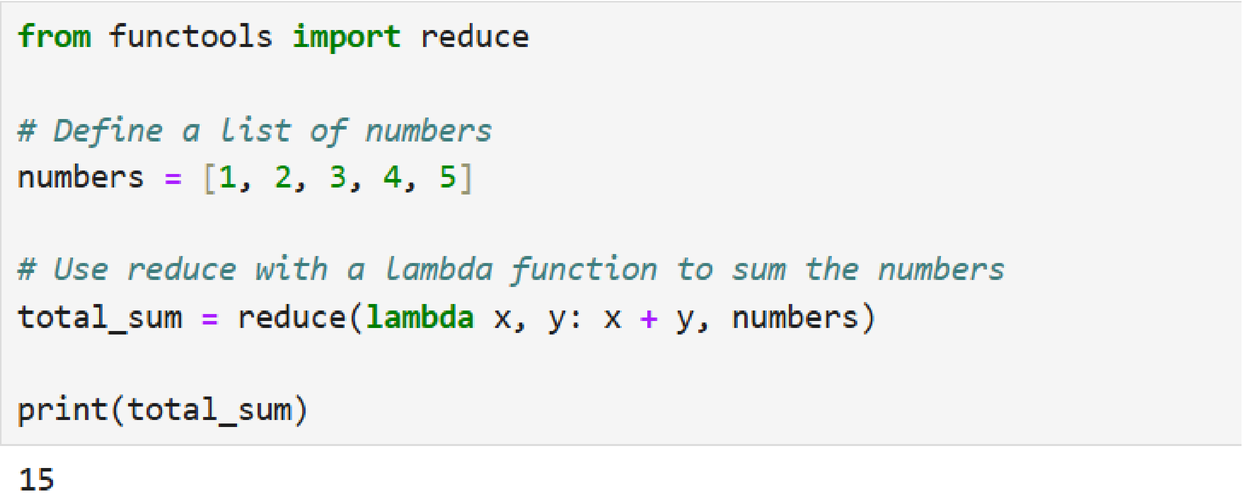
Example: Find the largest item from a given list.



```
from functools import reduce
num_list = [20, 12, 52, 22, 72, 19, 7]
large = reduce(lambda x,y:x if x>y else y, num_list)
print(large)
```

```
72
```

Example: Summing All Elements in a List



```
from functools import reduce

# Define a list of numbers
numbers = [1, 2, 3, 4, 5]

# Use reduce with a lambda function to sum the numbers
total_sum = reduce(lambda x, y: x + y, numbers)

print(total_sum)
```

```
15
```