Jawaharlal Nehru Technological University Hyderabad

Kukatpally, Hyderabad - 500 085, Telangana, India

Learning Objectives:

Python Sets , Built in Functions

Part 1, Session 9 , 7 Oct 22

Dr N V Ganapathi Raju
Professor and HOD of IT
Gokaraju Rangaraju Institute of Eng and Tech

Features of Python Sets

- Sets are a **mutable** collection of **unique** values
- Values are **unordered**
- Does **not support indexing**
- Highly useful to efficiently **remove duplicate values from a list or tuple**
- Perform common math operations like **unions and intersections**

Set Creation and Initialization

- To declare a set, type a sequence of items separated by commas, inside curly braces { }

  and assign it to a variable

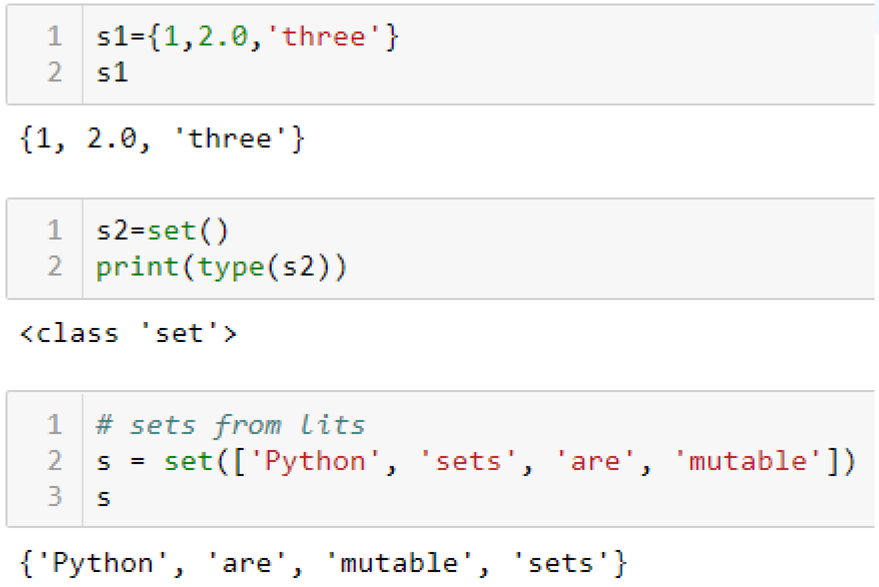
- Also by using **set()** built in function

- contain values of different types

- A set is mutable, but may not contain items like a list, set, or dictionary.

Set Creation and Initialization

```
s1={1,2.0,'three'}
print(s1)     # {1, 2.0, 'three'}


s2=set()
print(type(s2))  # <class 'set'>


# sets from lits
s3= set(['Python', ‘sets', 'are', 'mutable'])
print(s3)   #{'Python', 'are', 'mutable', ‘sets'}
```



```
s1={1,2.0,'three'}
s1
```

```
{1, 2.0, 'three'}
```

```
s2=set()
print(type(s2))
```

```
<class 'set'>
```

```
# sets from lits
s = set(['Python', 'sets', 'are', 'mutable'])
s
```

```
{'Python', 'are', 'mutable', 'sets'}
```

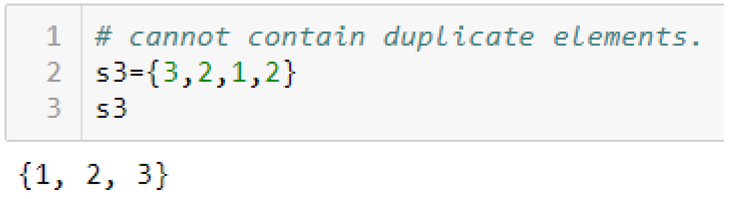
Imp points on sets

- since sets do not support indexing, they cannot be sliced

  s[:]

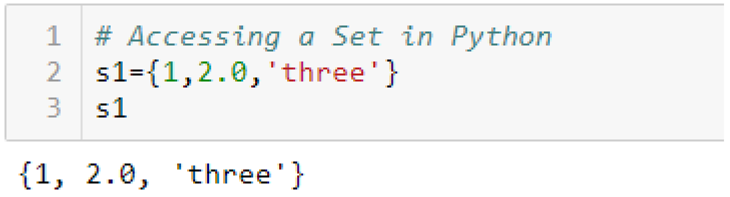
- Because a set isn't indexed, can't delete an element using its index.

```
# cannot contain duplicate elements.
s3={3,2,1,2}
print(s3)  # {1, 2, 3}

# Accessing a Set in Python
s1={1,2.0,'three'}
print(s1) # {1, 2.0, 'three'}
```



```
# cannot contain duplicate elements.
s3={3,2,1,2}
s3
```

```
{1, 2, 3}
```



```
# Accessing a Set in Python
s1={1,2.0,'three'}
s1
```

```
{1, 2.0, 'three'}
```

Adding elements

- Adding elements can ne done in two ways. 1. add()   2. update()

- To add or remove values from a set, Initialize it first

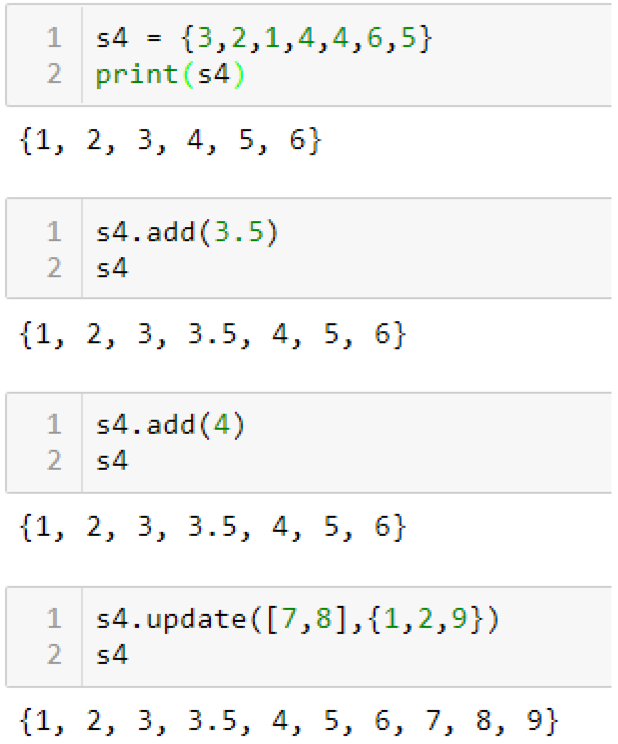
- To add single element using the add() method and multiple elements using the update() method.

- update() : can take tuples, lists, strings or other sets as its argument

  (duplicates are avoided)

Adding elements

```
s4 = {3,2,1,4,4,6,5}
print(s4)  # {1, 2, 3, 4, 5, 6}

s4.add(3.5)
print(s4)   # {1, 2, 3, 3.5, 4, 5, 6}

s4.add(4)
print(s4) # {1, 2, 3, 3.5, 4, 5, 6}

s4.update([7,8],{1,2,9})
print(s4)  # {1, 2, 3, 3.5, 4, 5, 6, 7, 8, 9}
```



```
s4 = {3,2,1,4,4,6,5}
print(s4)
```

```
{1, 2, 3, 4, 5, 6}
```

```
s4.add(3.5)
s4
```

```
{1, 2, 3, 3.5, 4, 5, 6}
```

```
s4.add(4)
s4
```

```
{1, 2, 3, 3.5, 4, 5, 6}
```

```
s4.update([7,8],{1,2,9})
s4
```

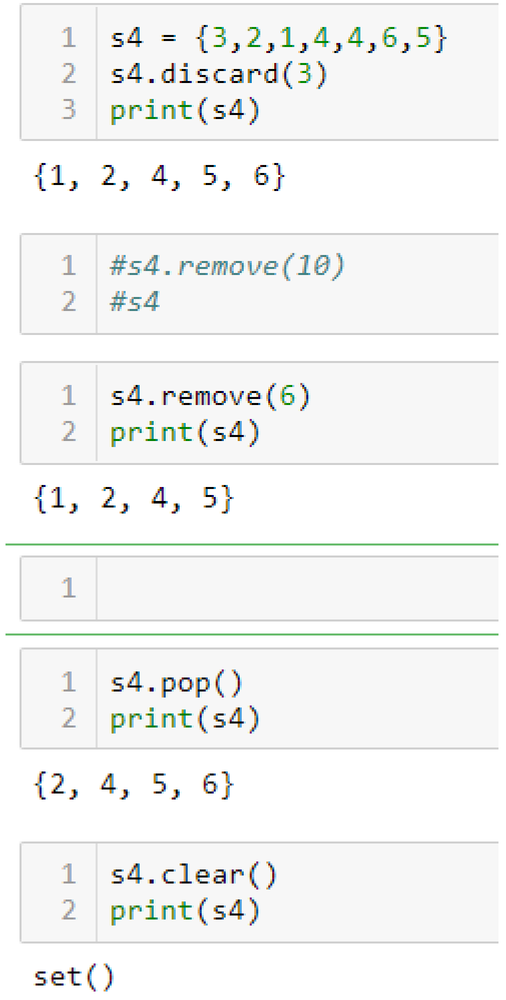
```
{1, 2, 3, 3.5, 4, 5, 6, 7, 8, 9}
```

Removing elements

- To remove an element from set 1. remove() 2 discard() 3 pop() 4 clear()
- Difference 1. remove() 2 discard()
  - while using discard() if the item does not exist in the set, it remains unchanged
  - remove() will raise an error in such condition.
- pop() : Remove and return an arbitrary value from a set
- clear(): remove all values from a set

Removing elements

```
s4 = {3,2,1,4,4,6,5}
s4.discard(3)
print(s4)

s4.remove(6)
print(s4)

s4.pop()
print(s4)

s4.clear()
print(s4)
```



```
s4 = {3,2,1,4,4,6,5}
s4.discard(3)
print(s4)
```

```
{1, 2, 4, 5, 6}
```

```
#s4.remove(10)
#s4
```

```
s4.remove(6)
print(s4)
```

```
{1, 2, 4, 5}
```

```
s4.pop()
print(s4)
```

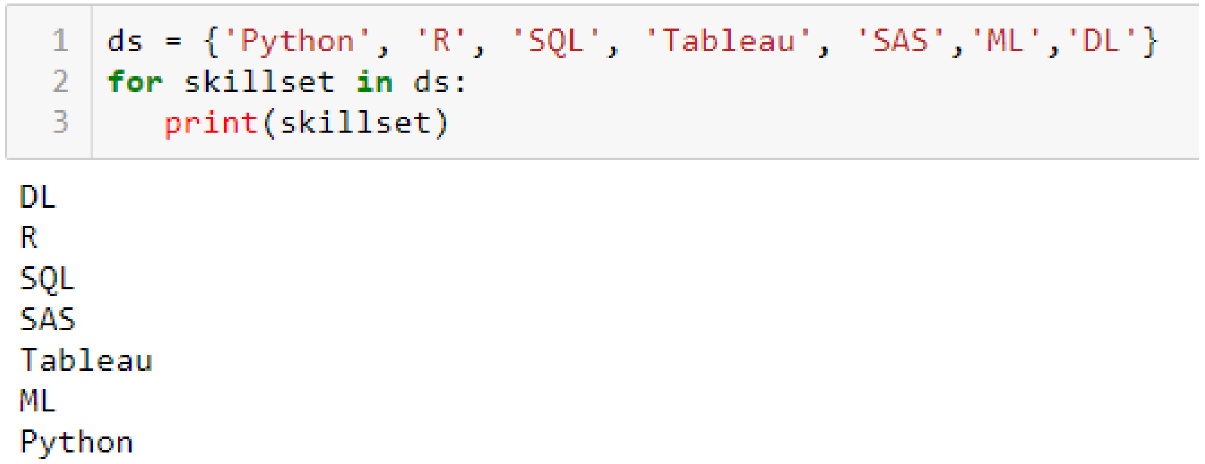
```
{2, 4, 5, 6}
```

```
s4.clear()
print(s4)
```

```
set()
```

Iterating over set using for loop

```
ds = {'Python', 'R', 'SQL', 'Tableau', 'SAS','ML','DL'}

for skillset in ds:

   print(skillset)
```



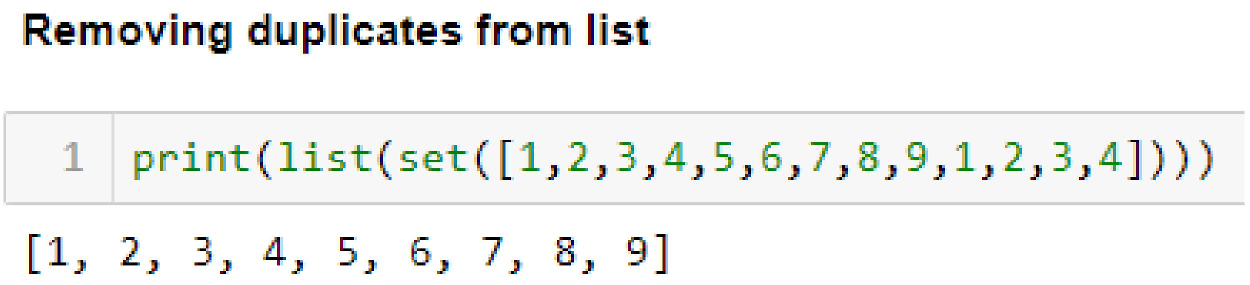
```
ds = {'Python', 'R', 'SQL', 'Tableau', 'SAS','ML','DL'}
for skillset in ds:
    print(skillset)
```

```
DL
R
SQL
SAS
Tableau
ML
Python
```

Removing duplicates

- Use a **set** to remove duplicates from a list.

```
print(list(set([1,2,3,4,5,6,7,8,9,1,2,3,4])))
```



Removing duplicates from list

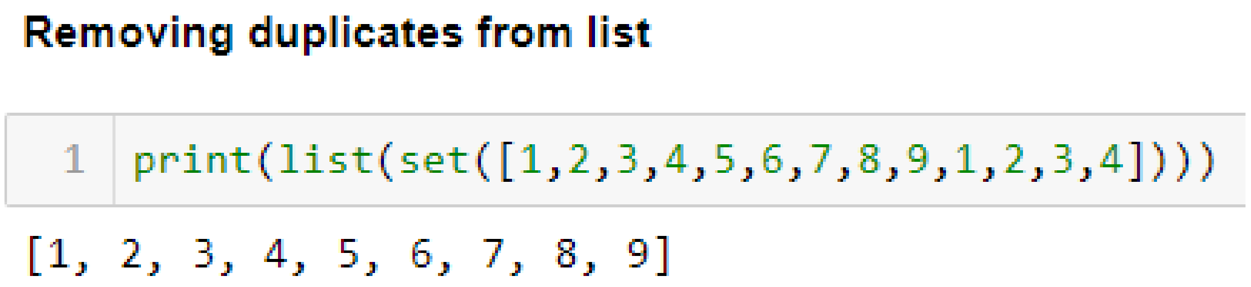
```
print(list(set([1,2,3,4,5,6,7,8,9,1,2,3,4])))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Removing duplicates

- Use a **set** to remove duplicates from a list.

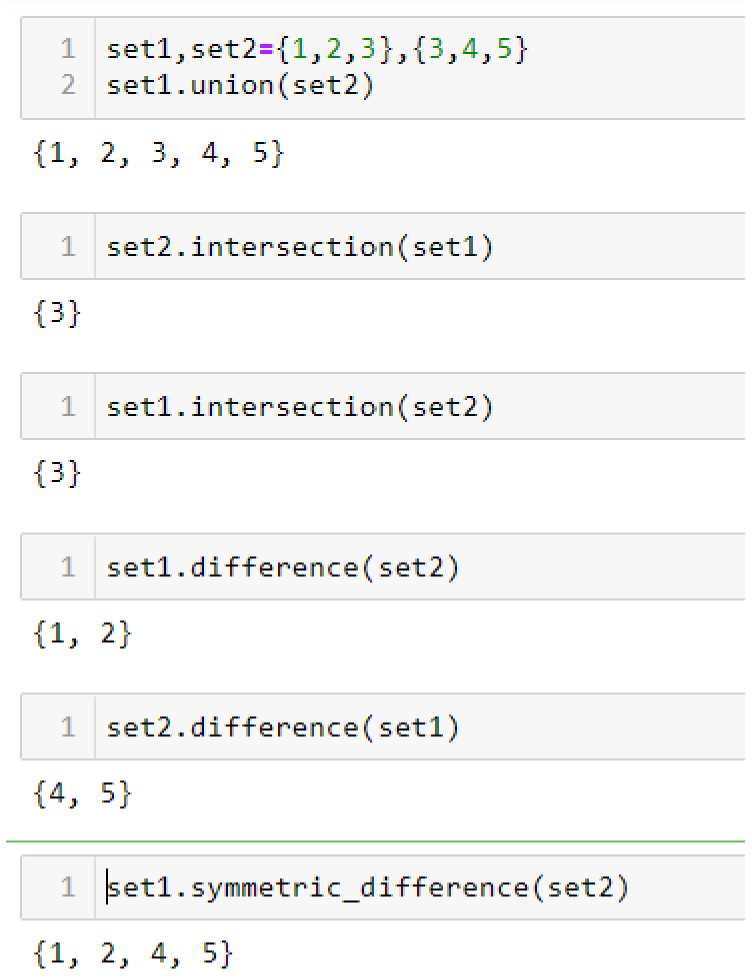
```
print(list(set([1,2,3,4,5,6,7,8,9,1,2,3,4])))
```


Removing duplicates from list

```
print(list(set([1,2,3,4,5,6,7,8,9,1,2,3,4])))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Sets Math Operations



```
set1,set2={1,2,3},{3,4,5}
set1.union(set2)
```

```
{1, 2, 3, 4, 5}
```

```
set2.intersection(set1)
```

```
{3}
```

```
set1.intersection(set2)
```

```
{3}
```

```
set1.difference(set2)
```

```
{1, 2}
```

```
set2.difference(set1)
```

```
{4, 5}
```

```
set1.symmetric_difference(set2)
```

```
{1, 2, 4, 5}
```

Set comprehension

- A set comprehension is like a list comprehension

  returns a **set**

```
s3 = {s for s in range(11) if s % 2}

print(s3)
```

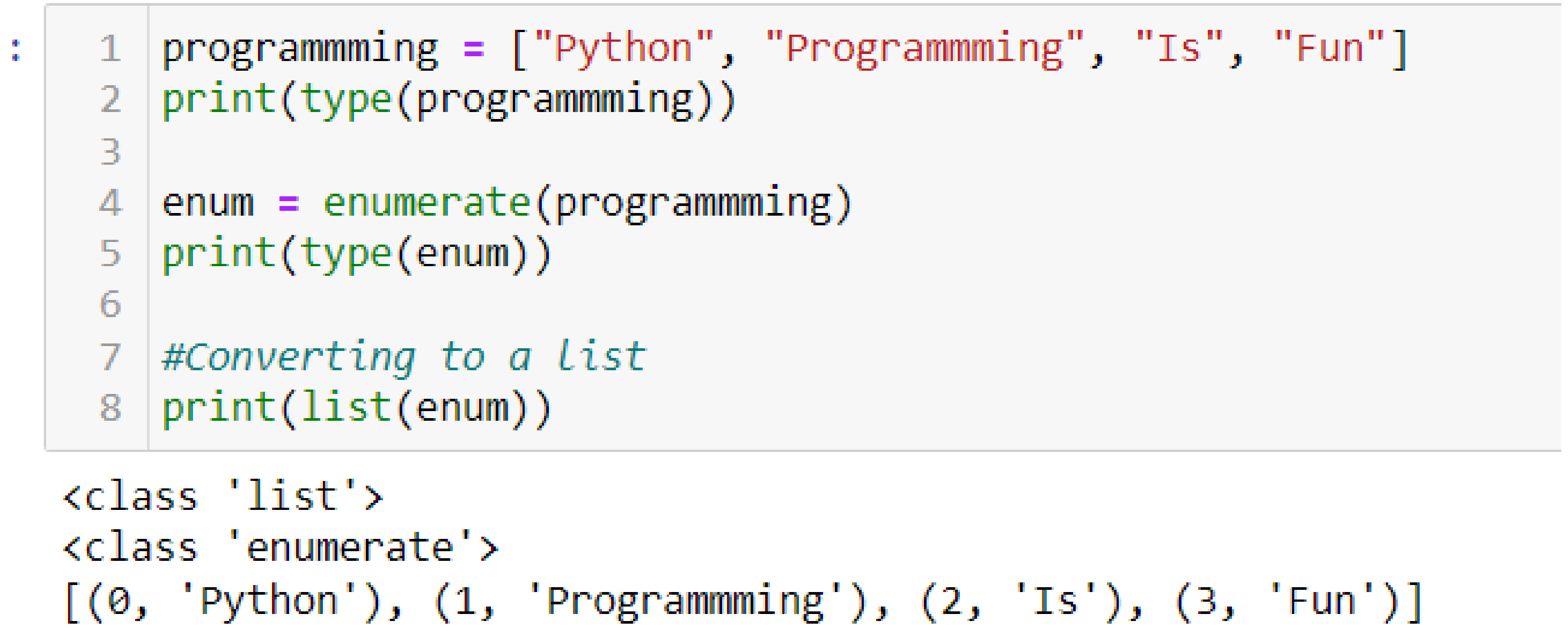
Built-In functions

- **The Python core library has three methods called**
  - enumerate()
  - zip()
  - map()
  - filter()
  - sorted()
  - reduce()

enumerate()

- An enumerator built-in-function ***adds a counter of iterable numbers*** to the provided data structure of integers, characters or strings and many more.
- The data structure might be any list, tuple, dictionary or sets.
- If the counter is not provided by the user, then it starts from 0 by default.
- Based on the number provided the enumerator function iterates.
- **Syntax:** **enumerate(iterable, start)**
- The return type of an enumerate function is an ***object*** type.
- So the enumerate function returns an object by adding the iterating counter value to it.
- You can also convert the enumerator object into a list(), tuple(), set() and many more.

enumerate



```python
programmming = ["Python", "Programmming", "Is", "Fun"]
print(type(programmming))

enum = enumerate(programmming)
print(type(enum))

#Converting to a list
print(list(enum))
```

```
<class 'list'>
<class 'enumerate'>
[(0, 'Python'), (1, 'Programmming'), (2, 'Is'), (3, 'Fun')]
```

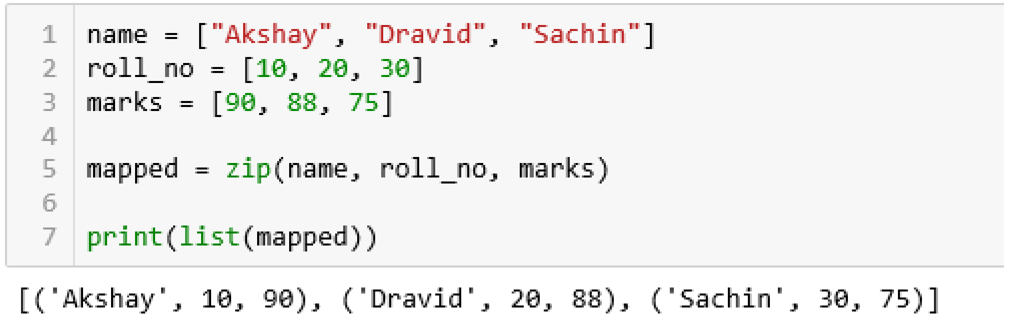
zip() built-in function

- zip() : function take iterables (can be zero or more), makes iterator that aggregates elements based on the iterables passed, and returns an iterator of tuples.

  zip(*iterables)

- The zip() function returns an iterator of tuples based on the iterable object.

```
name = ["Akshay", "Dravid", "Sachin"]
roll_no = [10, 20, 30]
marks = [90, 88, 75]

mapped = zip(name, roll_no, marks)

print(list(mapped))
```



```
name = ["Akshay", "Dravid", "Sachin"]
roll_no = [10, 20, 30]
marks = [90, 88, 75]

mapped = zip(name, roll_no, marks)

print(list(mapped))
```

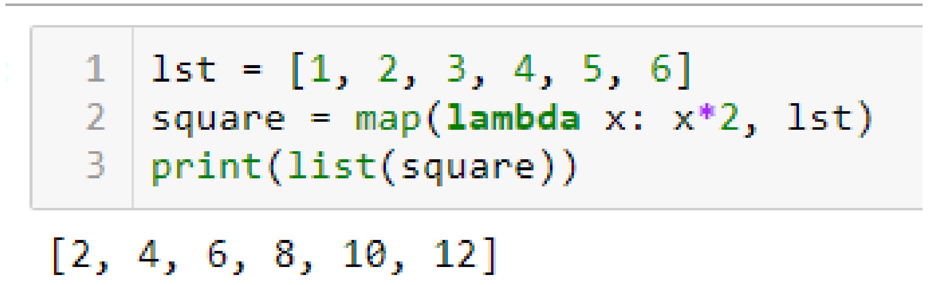
```
[('Akshay', 10, 90), ('Dravid', 20, 88), ('Sachin', 30, 75)]
```

map()

- The map() function expects two arguments: a function and a list.
- It takes that function and applies it on every item of the list and returns the modified list.

```
# Square each item of the list
def square(x):
    return x*2
lst = [1, 2, 3, 4, 5]
newlst = map(square, lst)
print(list(newlst))
```

```
lst = [1, 2, 3, 4, 5, 6]
square = map(lambda x: x*2, lst)
print(list(square))
```



```
lst = [1, 2, 3, 4, 5, 6]
square = map(lambda x: x*2, lst)
print(list(square))
```

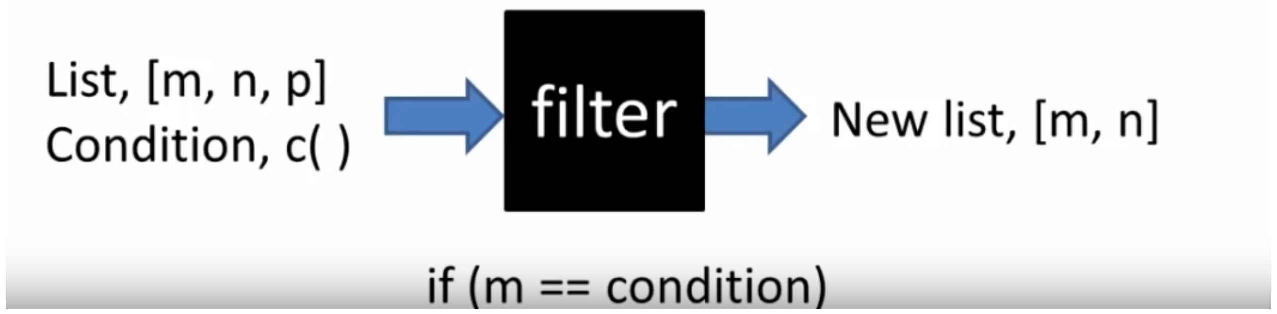
```
[2, 4, 6, 8, 10, 12]
```

filter()

- filter() function filters the given iterable with the help of a function that tests each element in the iterable to be true or not.

filter(fun, Iter)

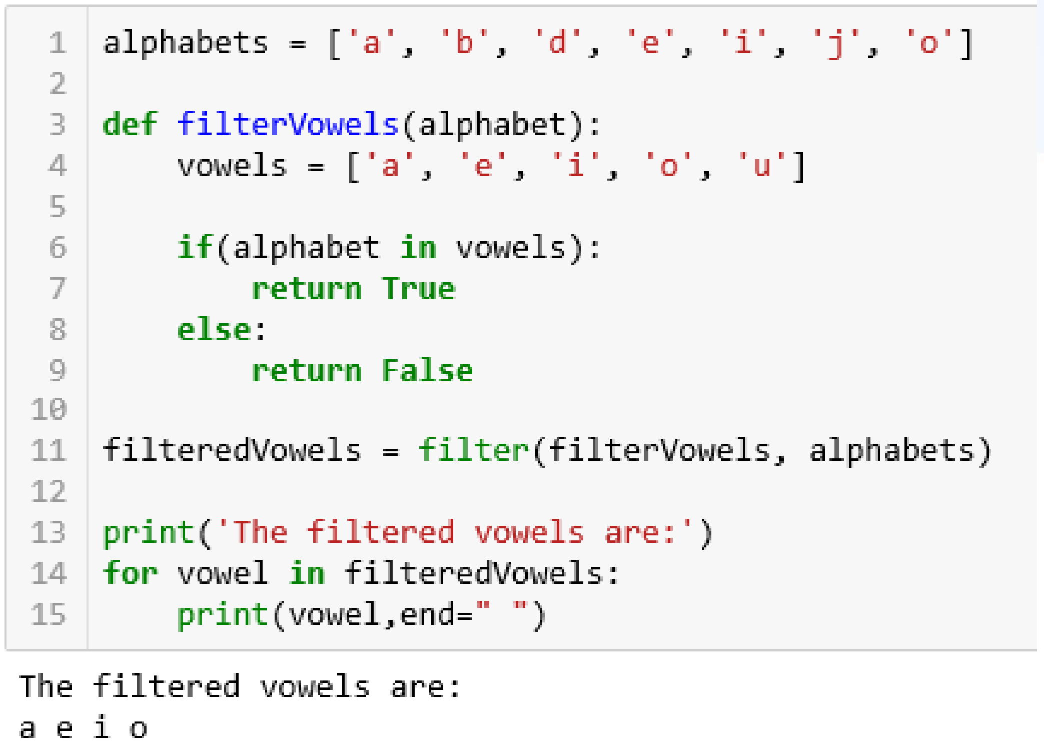
- **fun:** function that tests if each element of a sequence true or not.
- **Iter:** Iterable which needs to be filtered.



filter()

- **Function to filter out vowels from list**

```
alphabets = ['a', 'b', 'd', 'e', 'i', 'j', 'o']

def filterVowels(alphabet):
    vowels = ['a', 'e', 'i', 'o', 'u']

    if(alphabet in vowels):
        return True
    else:
        return False

filteredVowels = filter(filterVowels, alphabets)

print('The filtered vowels are:')
for vowel in filteredVowels:
    print(vowel,end=" ")
```



```
alphabets = ['a', 'b', 'd', 'e', 'i', 'j', 'o']

def filterVowels(alphabet):
    vowels = ['a', 'e', 'i', 'o', 'u']

    if(alphabet in vowels):
        return True
    else:
        return False

filteredVowels = filter(filterVowels, alphabets)

print('The filtered vowels are:')
for vowel in filteredVowels:
    print(vowel,end=" ")
```

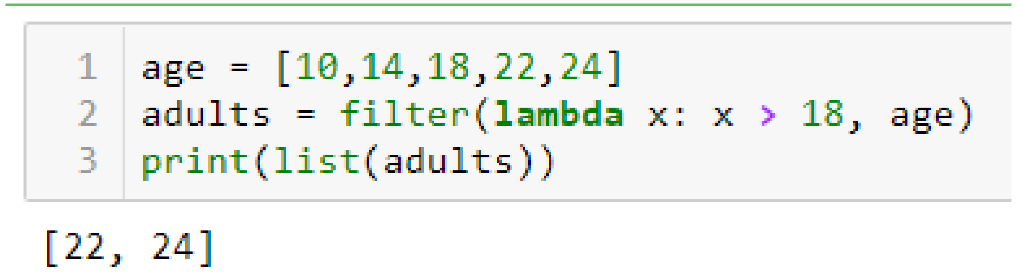
```
The filtered vowels are:
a e i o
```

filter()

- It takes a function and applies it to each item in the list to create a new list with only those items that cause the function to return True.

```
def checkAge(age):
    if age > 18:
        return True
    else:
        return False

lst = [10,14,18,22,24]
adults = filter(checkAge, lst)
print(list(adults))
```

```
age = [10,14,18,22,24]

adults = filter(lambda x: x > 18, age)

print(list(adults))
```



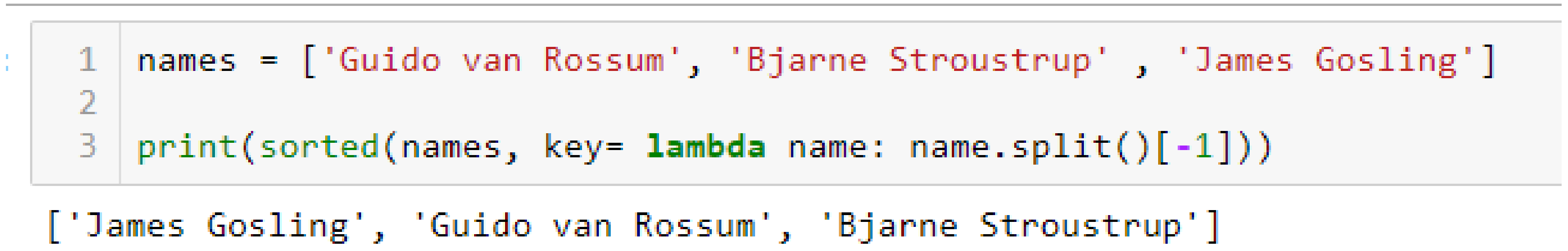
```
age = [10,14,18,22,24]
adults = filter(lambda x: x > 18, age)
print(list(adults))
```

```
[22, 24]
```

using built-in function sorted()

names = ['Guido van Rossum', 'Bjarne Stroustrup' , 'James Gosling']

print(sorted(names, key= lambda name: name.split()[-1])))



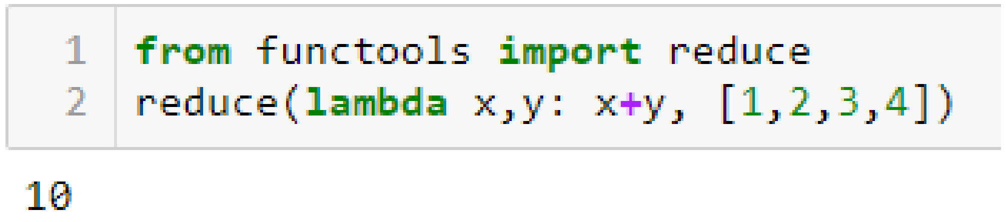
```
names = ['Guido van Rossum', 'Bjarne Stroustrup' , 'James Gosling']

print(sorted(names, key= lambda name: name.split()[-1]))
```

```
['James Gosling', 'Guido van Rossum', 'Bjarne Stroustrup']
```

reduce()

- The reduce(fun,seq) function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence.
- This function is defined in “functools” module.

```
from functools import reduce
reduce(lambda x,y: x+y, [1,2,3,4])
```



```
from functools import reduce
reduce(lambda x,y: x+y, [1,2,3,4])
```

```
10
```