Abstraction

Abstraction is used to hide the internal functionality of the function. An abstraction is used to hide the irrelevant data/class in order to reduce the complexity.

A class that consists of one or more abstract method is called the abstract class. Abstract methods do not contain their implementation. Abstract class can be inherited by the subclass and abstract method gets its definition in the subclass.

Python provides the **abc** module to use the abstraction in the Python program. Python doesn't provide the abstract class itself. We need to import the abc module, which provides the base for defining **Abstract Base classes** (ABC). The ABC works by decorating methods of the base class as abstract. We use the ***@abstractmethod*** decorator to define an abstract method or if we don't provide the definition to the method, it automatically becomes the abstract method.

Example

```
from abc import ABC, abstractmethod
# abstract class
class Polygon(ABC):

   # abstract method
   @abstractmethod
   def sides(self):
      pass
   def display(self):
       print("non abstract method")

class Triangle(Polygon):
    def sides(self):
        print("Triangle has 3 sides")


t = Triangle()
t.sides()
t.display()
```

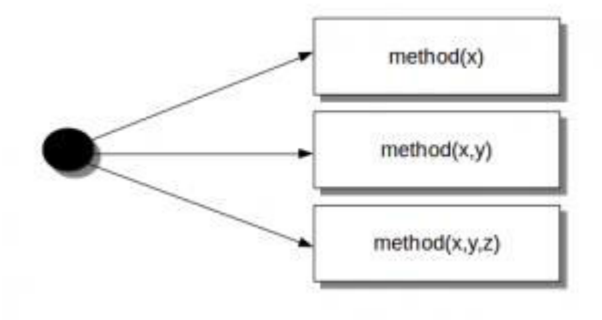
Output

```
Triangle has 3 sides
non abstract method
```

Polymorphism

Method overloading

Method Overloading is an example of Compile time polymorphism. In this, **more than one method of the same class shares the same method name having different signatures**(number,order and type of parameters). Method overloading is used to add more to the behavior of methods and there is no need of more than one class for method overloading.



Example

```
class Human:

    def sayHello(self, name=None):

        if name is not None:
            print('Hello ' + name)
        else:
            print('Hello ')


# Create instance
obj = Human()

# Call the method
obj.sayHello()

# Call the method with a parameter
obj.sayHello('Kiran')
```

Output

```
Hello
Hello Kiran
```

Example

```
def add(datatype, *args):

    # if datatype is int then,initialize answer as 0
    if datatype =='int':
        answer = 0

    # if datatype is str then, initialize answer as ''
    if datatype =='str':
        answer =''

        # Traverse through the arguments
        # This will do addition if the
        # arguments are int. Or concatenation
        # if the arguments are str
    for x in args:

        answer = answer + x
    print(answer)

# Integer
add('int', 5, 6)
# String
add('str', 'Hi ', 'Geeks')
```

Output

```
11
Hi Geeks
```

Method overriding

Method overriding is an example of run time polymorphism. Method overriding is a concept of object oriented programming that allows us to change the implementation of a function in the child class that is defined in the parent class.. In method overriding, inheritance always required as it is done between parent class(superclass) and child class(child class) methods.

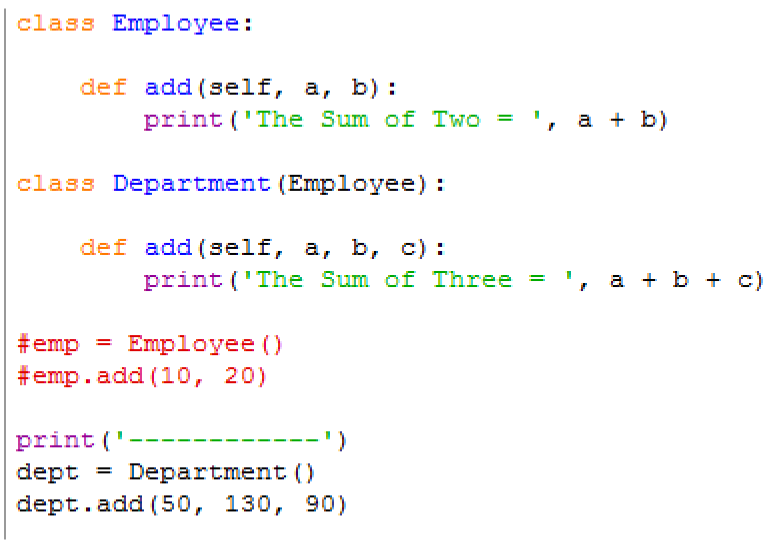
Example

```
class Rectangle():
        def __init__(self,length,breadth):
                self.length = length
                self.breadth = breadth
        def getArea(self):
                print(self.length*self.breadth," is area of rectangle")
class Square(Rectangle):
        def __init__(self,side):
                self.side = side
                #Rectangle.__init__(self,side,side)
        def getArea(self):
                print(self.side*self.side," is area of square")
s = Square(4)
s.getArea()

#r = Rectangle(2,4)
#r.getArea()
```

Output

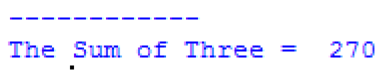
```
16  is area of square
```

Example



```
class Employee:

    def add(self, a, b):
        print('The Sum of Two = ', a + b)

class Department(Employee):

    def add(self, a, b, c):
        print('The Sum of Three = ', a + b + c)

#emp = Employee()
#emp.add(10, 20)

print('------------')
dept = Department()
dept.add(50, 130, 90)
```

Output



```
------------
The Sum of Three =  270
```

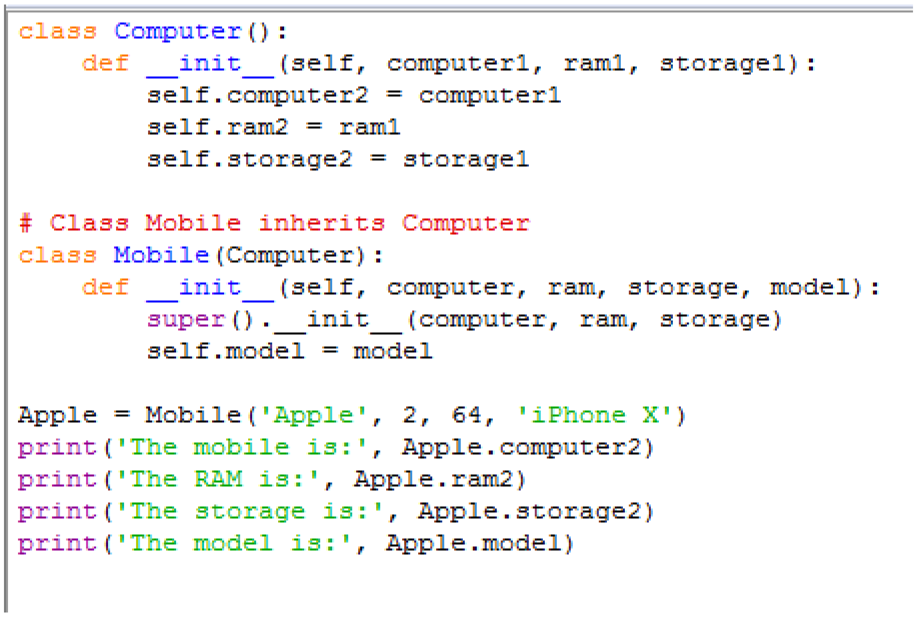
super() in Python

The super() function in Python makes class inheritance more manageable and extensible. The function returns a temporary object that allows reference to a parent class by the keyword super.

The super() function has two major use cases:

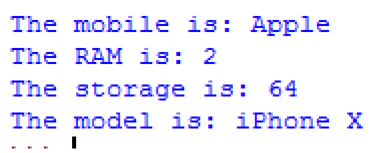
- To avoid the usage of the super (parent) class explicitly.
- To enable multiple inheritances.

Example



```
class Computer():
    def __init__(self, computer1, ram1, storage1):
        self.computer2 = computer1
        self.ram2 = ram1
        self.storage2 = storage1

# Class Mobile inherits Computer
class Mobile(Computer):
    def __init__(self, computer, ram, storage, model):
        super().__init__(computer, ram, storage)
        self.model = model

Apple = Mobile('Apple', 2, 64, 'iPhone X')
print('The mobile is:', Apple.computer2)
print('The RAM is:', Apple.ram2)
print('The storage is:', Apple.storage2)
print('The model is:', Apple.model)
```

Output



```
The mobile is: Apple
The RAM is: 2
The storage is: 64
The model is: iPhone X
```

Multiple inheritances with super()

```python
class Animal:
  def __init__(self, animalName):
    print(animalName, 'is an animal.');

# Mammal inherits Animal
class Mammal(Animal):
  def __init__(self, mammalName):
    print(mammalName, 'is a mammal.')
    super().__init__(mammalName)

# CannotFly inherits Mammal
class CannotFly(Mammal):
  def __init__(self, mammalThatCantFly):
    print(mammalThatCantFly, "cannot fly.")
    super().__init__(mammalThatCantFly)

# CannotSwim inherits Mammal
class CannotSwim(Mammal):
  def __init__(self, mammalThatCantSwim):
    print(mammalThatCantSwim, "cannot swim.")
    super().__init__(mammalThatCantSwim)

# Cat inherits CannotSwim and CannotFly
class Cat(CannotSwim, CannotFly):
  def __init__(self):
    print('I am a cat.');
    super().__init__('Cat')

# Driver code
cat = Cat()
print('')
bat = CannotSwim('Bat')
```

Output

```
I am a cat.
Cat cannot swim.
Cat cannot fly.
Cat is a mammal.
Cat is an animal.

Bat cannot swim.
Bat is a mammal.
Bat is an animal.
```

Session-9

Regular Expression

A RegEx, or Regular Expression, 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

Python has a built-in package called re, which can be used to work with Regular Expressions.

Import the re module:

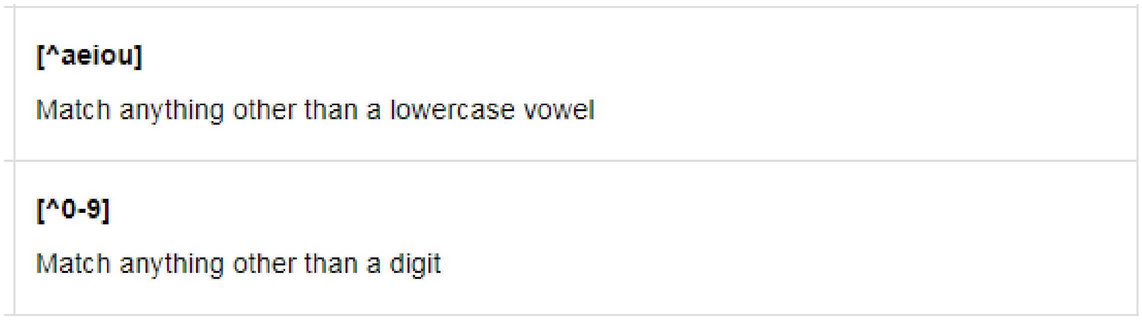
Regular Expression Patterns

Except for control characters, (+ ? . * ^ $ ( ) [ ] { } | \), all characters match themselves. You can escape a control character by preceding it with a backslash.

Sr.No.	Pattern & Description
1	**^** Matches beginning of line.
2	**$** Matches end of line.
3	**.** Matches any single character except newline. Using m option allows it to match newline as well.
4	**[...]** Matches any single character in brackets.
5	**[^...]** Matches any single character not in brackets

Character classes

[aeiou] Match any one lowercase vowel
[0-9] Match any digit; same as [0123456789]
[a-z] Match any lowercase ASCII letter
[A-Z] Match any uppercase ASCII letter
[a-zA-Z0-9] Match any of the above



[^aeiou] Match anything other than a lowercase vowel
[^0-9] Match anything other than a digit

Metacharacters Supported by the re Module

The following table briefly summarizes all the metacharacters supported by the re module. Some characters serve more than one purpose:

Regex	**Description**
\d	Matches any decimal digit; this is equivalent to the class [0-9]
\D	Matches any non-digit character; this is equivalent to the class [^0-9].
\s	Matches any whitespace character; this is equivalent to the class [\t\n\r\f\v].
\S	Matches any non-whitespace character; this is equivalent to the class [^ \t\n\r\f\v].
\w	Matches any alphanumeric character; this is equivalent to the class [a-zA-Z0-9_].
\W	Matches any non-alphanumeric character; this is equivalent to the class [^a-zA-Z0-9_].
\Z	Matches only at end of string

[..]	Match single character in brackets
[^..]	Match any single character not in brackets
.	Match any character except newline
$	Match the end of the string
*****	Match 0 or more repetitions
+	1 or more repetitions
{m}	Exactly m copies of the previous RE should be matched.
|	Match A or B. A|B
?	0 or 1 repetitions of the preceding RE
[a-z]	Any lowercase character
[A-Z]	Any uppercase character
[a-zA-Z]	Any character
[0-9]	Any digit

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

The findall() Function

Example

Print a list of all matches:

```
import re

txt = "A beautiful and colourful garden"
x = re.findall("ful", txt)
print(x)
```

Output

```
['ful', 'ful']
```

If no matches are found, an *empty list* is returned:

Example

Return an empty list if no match was found:

```
import re

txt = "The rain in beautiful"
x = re.findall("rose", txt)
print(x)
```

Output

```
[]
```

```
import re
s="sat mat pat cat rat"
l=re.findall(r"[spr]at",s)
print(l)
['sat', 'pat', 'rat']
```

```
import re
s="sat mat pat cat "
l=re.findall(r"[^spr]a",s)
print(l)
['ma', 'ca']
```

```
import re
s="sat mat pat cat 999989"
l=re.findall(r"\d{1,4}",s)
print(l)
['9999', '89']
```

```
import re
s="sat mat pat cat 999989 "
l=re.findall(r"\d{4}",s)
print(l)
```

['9999']

```
import re
s="sat mat pat cat 999989 8888"
l=re.findall(r"\b\d{4}\b",s)
print(l)
['8888']
```

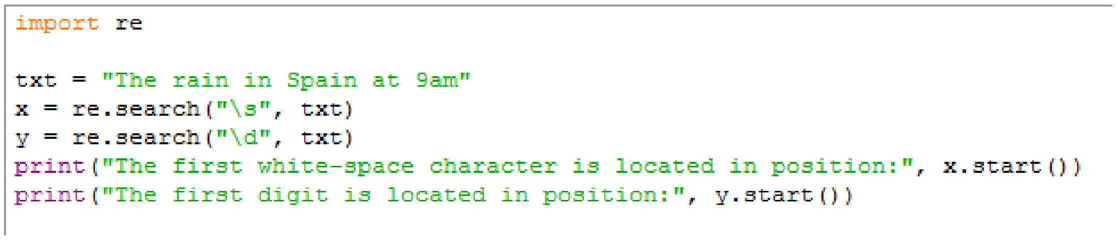
The search() Function

The search() function searches the string for a match, and returns a Match object if there is a match.If there is more than one match, only the first occurrence of the match will be returned.

Example

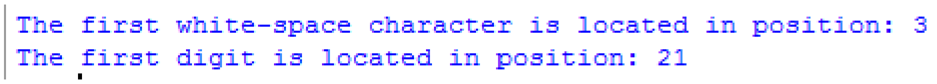
Search for the first white-space character   and digit in the string:

\s	Returns a match where the string contains a white space character.
\d	Returns a match where the string contains digits (numbers from 0-9)



```
import re

txt = "The rain in Spain at 9am"
x = re.search("\s", txt)
y = re.search("\d", txt)
print("The first white-space character is located in position:", x.start())
print("The first digit is located in position:", y.start())
```

Output



```
The first white-space character is located in position: 3
The first digit is located in position: 21
```

If no matches are found, the value None is returned:

Example

```
import re

txt = "The rain in Spain at 9am"
x = re.search("sun", txt)
print(x)
```

Output

```
None
```

Example:

Search the string to see if it starts with "The" and ends with "clever":[.* just means "0 or more of any character"]

```
import re
txt = "The boy is clever"
x = re.search("^The.*clever$", txt)

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

Output

```
YES! We have a match!
```

The split() Function

The split() function returns **a list** where the string has been split at each match:

Example

Split at each white-space character:

```
import re

txt = "Apple and mango are fruits"
x = re.split("\s", txt)
print(x)
```

Output

```
['Apple', 'and', 'mango', 'are', 'fruits']
```

Example

Split the string only at the first,second,third occurrences:

Example

```
import re

txt = "Apple and mango are fruits"
x = re.split("\s", txt, 1)
y = re.split("\s", txt, 2)
z = re.split("\s", txt, 3)
print(x)
print(y)
print(z)
```

Output

```
['Apple', 'and mango are fruits']
['Apple', 'and', 'mango are fruits']
['Apple', 'and', 'mango', 'are fruits']
```

The sub() Function

The sub() function replaces the matches with the text of your choice:

Example

Replace every white-space character with *:

```
import re
txt = "Rose and jasmine are flowers"
x = re.sub("\s", "*", txt)
print(x)
```

Output

```
Rose*and*jasmine*are*flowers
```

Example

Replace the first 2 occurrences:

Replace every white-space character with *:

```
import re
txt = "Rose and jasmine are flowers"
x = re.sub("\s", "*", txt)
v = re.sub("\s", "#", txt, 2)
print(x)
print(v)
```

Output

```
Rose*and*jasmine*are*flowers
Rose#and#jasmine are flowers
```

Match Object

A Match Object is an object containing information about the search and the result.

Note: If there is no match, the value None will be returned, instead of the Match Object.

Example

Do a search that will return a Match Object:

The Match object has properties and methods used to retrieve information about the search, and the result:

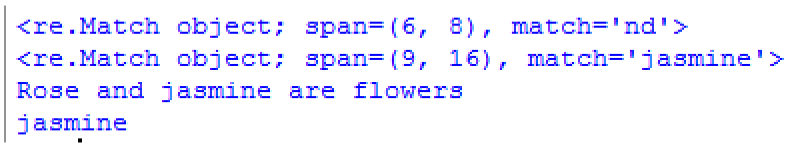
.span() returns a tuple containing the **start-, and end positions** of the match.

.string returns the **string** passed into the function

.group() returns the **part of the string** where there was a match

```
import re
txt = "Rose and jasmine are flowers"
x = re.search("nd", txt)
y = re.search(r"\bj\w+", txt)
print(x)
print(y)
print(y.string)
print(y.group())
```

Output



```
<re.Match object; span=(6, 8), match='nd'>
<re.Match object; span=(9, 16), match='jasmine'>
Rose and jasmine are flowers
jasmine
```

Email validation using re in python

Phone no

pincode