Classes, Objects and Regular Expressions

Department of CE-AI/
CE-Big data

Unit no. 5 Classes, Objects and Regular Expressions

Class

A class in python are the user-defined blueprints that help us create an object.

For example: If we define a blueprint of human beings then it can have attributes like **Name, Age, Gender, Occupation** and functionalities like **Walk(), Eat(),** and **Sleep()**. Any human being must have these properties(data) and functionalities(methods) associated with them.

Creating a class

Creating a class is as easy as creating a function in Python. In function, we start with the def keyword while class definitions begin with the keyword class.
Following the keyword class, we have the class identifier(i.e. the name of the class we created) and then the : (colon) operator after the class name.
In the next indented lines(statement 1..n) are the members of the class. Also, the variables inside the class are known as attributes. The attributes can be accessed later on in the program using the dot(.) operator.

The syntax for creating a Class in Python is as follows:

```
class ClassName:
    # Statement 1
    # Statement 2
    .
    .
    # Statement n
```

Python Class Attributes and Methods

To make the most of Python Classes, we also need to add some functionality to the classes. We can make this possible with the help of attributes and methods.

Python Class Attributes

These are the variables that are defined inside the particular class and can be used by all the objects. These attributes can, later on, be called by using the class and attribute name with the dot(.) operator.

Example to define some attributes and access it.

```
class Subject:
    Course1 = 'Python'
    Course2 = 'C++'
    Course3 = 'Java'
# Accessing the values of the attributes
print(Subject.Course1)
print(Subject.Course3)
# Accessing through object instantiation.
obj= Subject()
print(obj.Course2)
```

Python Class Methods

Once we have defined the class attributes, we can even define some functions inside the particular class that can access the class attribute and can add more functionality to the existing code.

These defined functions inside a class are known as methods. We can define as many methods as we want inside a single class.

Note: When we define a method, it must at least pass a single parameter which is generally named as self. It refers to the current instance of the class.

We can call this parameter by any name, other than self if we want.

Syntax:

```
class Subject:
    def CourseDetails(self):
        print('Course Information')
```

Object

An object is called an instance of a class. The creation of an object (or an “instance” of a given class) in an object-oriented programming (OOP) language, is called instantiation.
Let's take an example to understand this concept better. Here, we have the blueprint of a car on the left side. This is known as the class, while on the right side, we have some of the models of the cars based on that blueprint, which is known as the objects(instances of the class) of the particular class.
Syntax for object:-

[object_name] = [class_name](arguments/parameters)

Creating an object in Python

Before creating an object, we should know that it contains such properties to distinguish it from others.

State - The state of an object is determined by the attributes of the object(i.e., the different items we have in the class, which the object inherits.).

Behavior - The behavior is represented by the methods of the object. It shows the difference and similarities of the functionality of an object to other objects.

Identity - Each and every object must be unique on its own. We can make this by giving it a unique name(like obj1, obj3, new_obj, etc.).

The syntax of creating an object of a particular class is as follows:

[object_name] = [class_name](arguments/parameters)

Here in comparison to the above example, the object_name is like the car models, namely, Hyundai, Ford, etc. The class_name is similar to the car blueprint. The **arguments** are just like the car's features, which can pass to some particular car models(objects).

Example

```
# Creating a class named Student
class Student:
  def __init__(self, name, age, hobby):
    self.name = name
    self.age = age
    self.hobby= hobby

  def new(self):
    print("Heyy my name is " + self.name + ". I Love " + self.hobby)

obj2 = Student("Vikram", 24, "coding")
obj2.new()
```

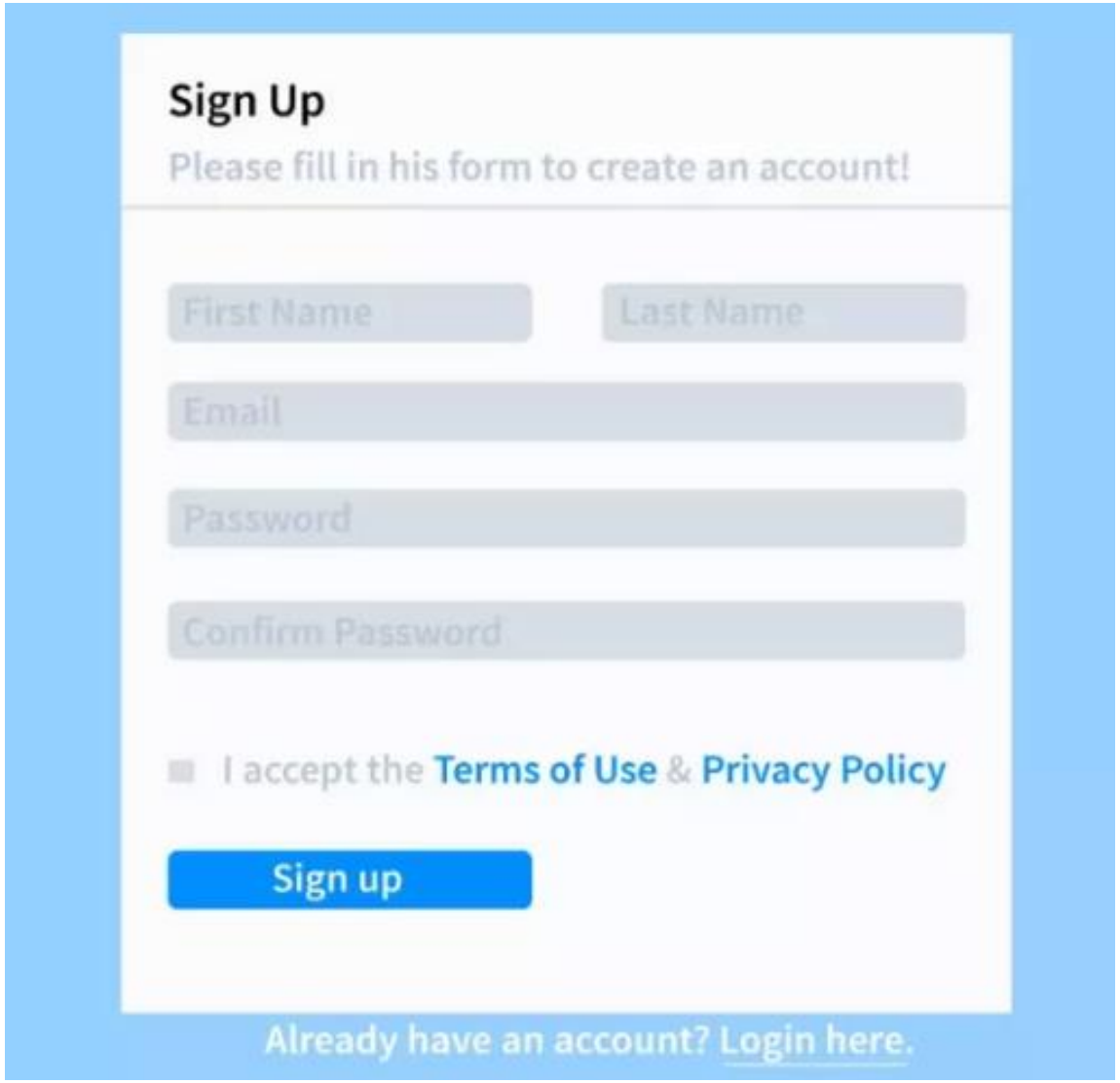
In this code, we have created a class named Student. we have defined two methods(_init_ & new). We have created an object ***"obj2"***, which is the instance of the class, and it accesses the function new(), while also passing three arguments. This way, the object can access the functions.

Difference Between Class and Object

Class	Object
A class is a blueprint for declaring and creating objects.	An object is a class instance that allows programmers to use variables and methods from inside the class.
Memory is not allocated to classes. Classes have no physical existence.	When objects are created, memory is allocated to them in the heap memory.
You can declare a class only once.	A class can be used to create many objects.
Class is a logical entity.	An object is a physical entity.
We cannot manipulate class as it is not available in memory.	Objects can be manipulated.
Class is created using the class keyword like class Dog{}	Objects are created through new keyword like Dog d = new Dog();. We can also create an object using the newInstance() method, clone() method, fatory method and using deserialization.
Example: Mobile is a class.	If Mobile is the class then iphone, redmi, blackberry, samsung are its objects which have different properties and behaviours.
Classes serve as blueprints for creating objects.	Objects represent specific instances created from a class.

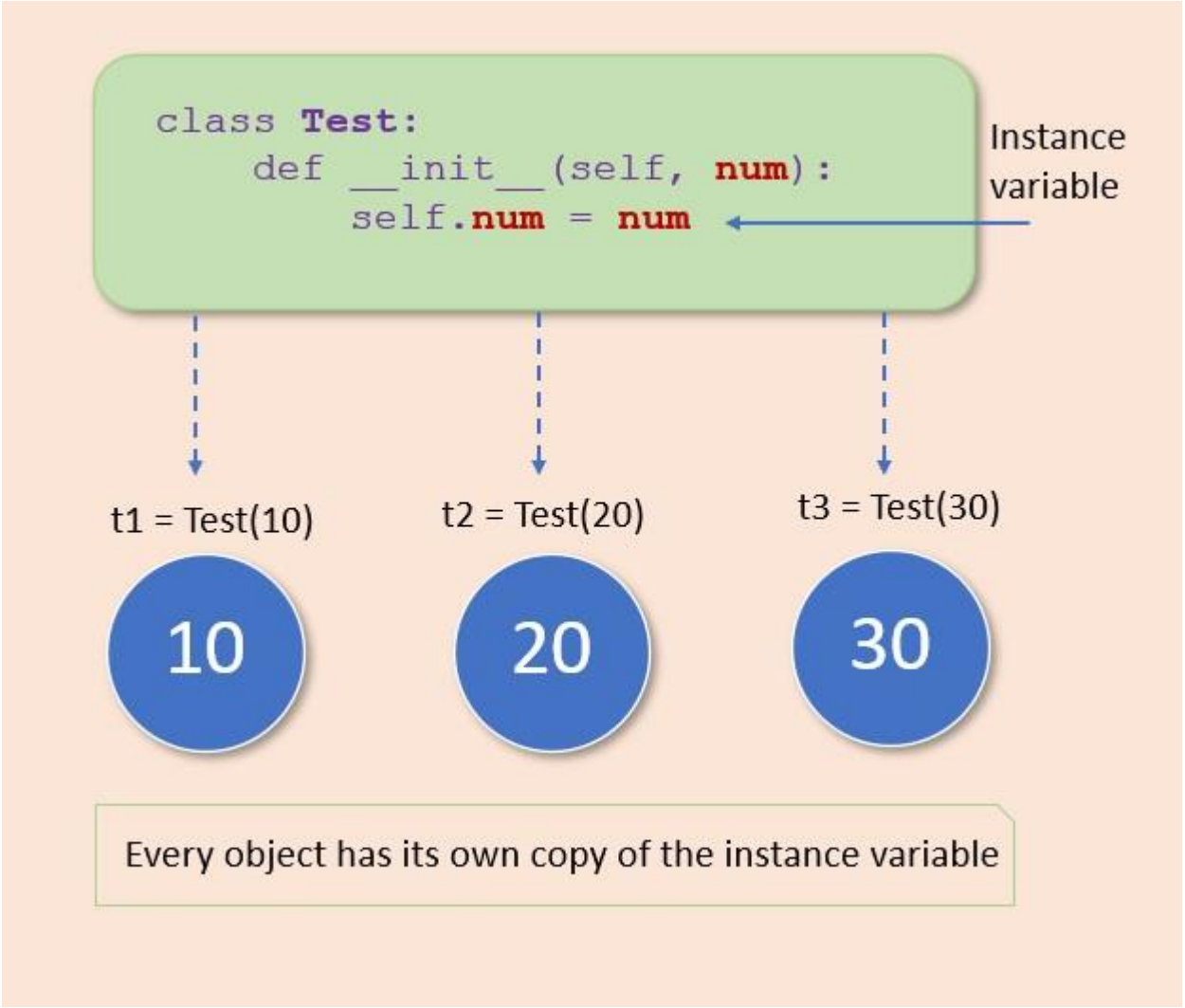
Example of Class and Object

This page contains many options, like First Name, Last Name, Email, and Password. This is an example of a blueprint. We can fill in various values and accordingly, various objects of this blueprint will be generated. When we fill these fields and click on **Sign Up**, a new object of this blueprint is created in the database.



Instance Variable in Python

If the value of a variable varies from object to object, then such variables are called instance variables. For every object, a separate copy of the instance variable will be created.



Access specifiers

Access specifiers or access modifiers in python programming are used to limit the access of class variables and class methods outside of class while implementing the concepts of inheritance.

There are three types of access specifiers or access modifiers

1). Public access modifier
2). Private access modifier
3). Protected access modifier

Public Access Modifier in Python

All the variables and methods (member functions) in python are by default public. Any instance variable in a class followed by the **'self' keyword** ie. **self.var_name** are public accessed.

```
#Syntax_public_access_modifiers
# defining class Student
class Student:
# constructor is defined
    def __init__(self, age, name):
        self.age = age # public Attribute
        self.name = name # public Attribute
# object creation
obj = Student(21,"python")
print(obj.age)
print(obj.name)
```

Private Access Modifier

Private members of a class (variables or methods) are those members which are only accessible inside the class. We cannot use private members outside of class.
It is also not possible to inherit the private members of any class (parent class) to derived class (child class). Any instance variable in a class followed by self keyword and the variable name starting with double underscore ie. **self.__varName** are the private accessed member of a class.

```
# Private_access_modifiers
class Student:
    def __init__(self, age, name):
        self.__age = age
    def __funName(self): self.y = 34
        print(self.y)
class Subject(Student):
        pass
obj = Student(21,"python")
obj1 = Subject
# calling by object reference of class Student
print(obj.__age)
print(obj.__funName())
# calling by object reference of class Subject
print(obj1.__age)
print(obj1.__funName())
```

Protected Access Modifier

Protected variables or we can say protected members of a class are restricted to be used only by the member functions and class members of the same class. And also it can be accessed or inherited by its derived class (child class). We can modify the values of protected variables of a class. The syntax we follow to make any variable **protected** is to write variable name followed by a single underscore (_) ie. **_varName.**

```
#Syntax_protected_access_modifiers
class Student:
  def __init__(self):
     self._name = "Python.com"
  def _funName(self):
      return "Method Here"
class Subject(Student):
  pass
obj = Student()
obj1 = Subject()
# calling by obj. ref. of Student class
print(obj._name)
# Python.com
print(obj._funName())
# Method Here
# calling by obj. ref. of Subject class
print(obj1._name)     # Python.com
print(obj1._funName()) # Method Here
```

Importance of self

The term “self” refers to the instance of the class that is currently being used. It is customary to use “self” as the first parameter in instance methods of a class. Whenever you call a method of an object created from a class, the object is automatically passed as the first argument using the “self” parameter. This enables you to modify the object’s properties and execute tasks unique to that particular instance.

__init__Method

__init__ is a special python method that runs whenever a new object is created. These types of functions are used to initialize the attributes of that class, E.g., if the class is Person, then the attributes will be name, age, etc.

Syntax

```
class Songs:
    def __init__(self, arguments):
        # function definition
```

Regular Expression in Python

Regular expression is a sequence of characters that forms a pattern which is mainly used to find or replace patterns in a string.
These are supported by many languages such as python, java, R etc.
Most common uses of regular expressions are:

- ✓ Finding patterns in a string or file.(Ex: find all the numbers present in a string)
- ✓ Replace a part of the string with another string.
- ✓ Search substring in string or file.
- ✓ Split string into substrings.
- ✓ Validate email format.

In python we have a built-in package called re to work with regular expressions.

Example

```
import re
text = '''Alan Turing was a pioneer of theoretical computer science and artificial intelligence. He was
born on 23 June 1912 in Maida Vale, London'''
res = re.search("^Alan.*London$",text)
if(res):
    print("We have a match!")
else:
    print("We don't have a match")
```

Output:
We have a match!

Python RegEx Expressions Functions

The ‘re’ package in python provides various functions to work with regular expressions.

S No	Function	Description
1	findall(pattern,string)	This matches all the occurrences of the pattern present in the string.
2	search(pattern,string)	This matches the pattern which is present at any position in the string. This will match the first occurrence of the pattern.
3	split(pattern,string)	This splits the string on the given pattern.
4	sub(pattern,rep_substring, string)	This replaces one or more matching pattern in the string with the given substring.

Meta Characters

S No	Meta character	Description
1	[ ](Square brackets)	This matches any single character in this bracket with the given string.
2	. (Period)	This matches all the characters except the newline. If we pass this as a pattern in the findall() function it will match with all the characters present in the string except newline characters.
3	^ (Carret)	This matches the given pattern at the start of the string.This is used to check if the string starts with a particular pattern or not.
4	$ (Dollar)	This matches the given pattern at the end of string. This is used to check if the string ends with a pattern or not.
5	* (Star)	This matches 0 or more occurrences of the pattern to its left.
6	+ (Plus)	This matches 1 or more occurrences of the pattern to its left.
7	? (Question mark)	This matches 0 or 1 occurrence of the pattern to its left.
8	{ } (Braces)	This matches the specified number of occurrences of pattern present in the braces.
9	(Alternation)	This works like ‘or’ condition. In this we can give two or more patterns. If the string contains at least one of the given patterns this will give a match.
10	( ) (Group)	This is used to group various regular expressions together and then find a match in the string.
11	\ (Backslash)	This is used to match special sequences or can be used as escape characters also.

Match() function

The match() function, located within Python's ‘re’ module, is purposefully designed to undertake pattern-matching operations exclusively at the beginning of a given string.

Syntax for match() function

re.match(pattern, string)

Now we will see the attributes and properties of re.match objects one by one. They are as follows:

match.group(): This returns the part of the string where the match was there.
match.start(): This returns the start position of the matching pattern in the string.
match.end(): This returns the end position of the matching pattern in the string.
match.span(): This returns a tuple which has start and end positions of matching pattern.
match.re: This returns the pattern object used for matching.
match.string: This returns the string given for matching.
Using r prefix before regex: This is used to convert the pattern to raw string.This means any special character will be treated as normal character. Ex: \ character will not be treated as an escape character if we use r before the pattern.

Search function

This is the same as match function but this function can search patterns irrespective of the position at which the pattern is present. The pattern can be present anywhere in the string. This function matches the first occurrence of the pattern.

Syntax for search function

Re.search(pattern, string)

Example

```
# import re module
import re
Substring ='string'
String1 ='''We are learning regex with geeksforgeeks
         regex is very useful for string matching.
          It is fast too.'''
String2 ='''string We are learning regex with geeksforgeeks
         regex is very useful for string matching.
          It is fast too.'''

# Use of re.search() Method
print(re.search(Substring, String1))
# Use of re.match() Method
print(re.match(Substring, String1))

# Use of re.search() Method
print(re.search(Substring, String2))
# Use of re.match() Method
print(re.match(Substring, String2))
```

Wildcards in Python

A wildcard is a symbol that can be used in place of or in addition to one or more characters. In computer programs, languages, search engines, including operating systems, wildcards are used to condense search criteria. The question mark (?) and the asterisk () are the most popular wildcards.

Types of wildcards

The Asterisk (∗)

The asterisk (*) or the character can be used to specify any number of characters. The asterisk * is typically used at the conclusion of the root word and when it is necessary to look for root words with a variety of possible ends.
For instance, if we use the word "game" as an example, the phrases "gamer" and "games" would appear in all search results. Depending on the search parameters and other words, there might be additional words in addition to these two.

The Question Mark (?)

The question mark or the character? denotes one. Any of the letters in the root word may be used with it. When a word contains several other spellings, the use of the question mark operator speeds up the process.

Instead of the question mark wildcard, the dot or. character is utilised for the single character representation.
Take the word "honour," for instance. It would indicate the result as honour while omitting honour in this context.

Wildcard Search in Python

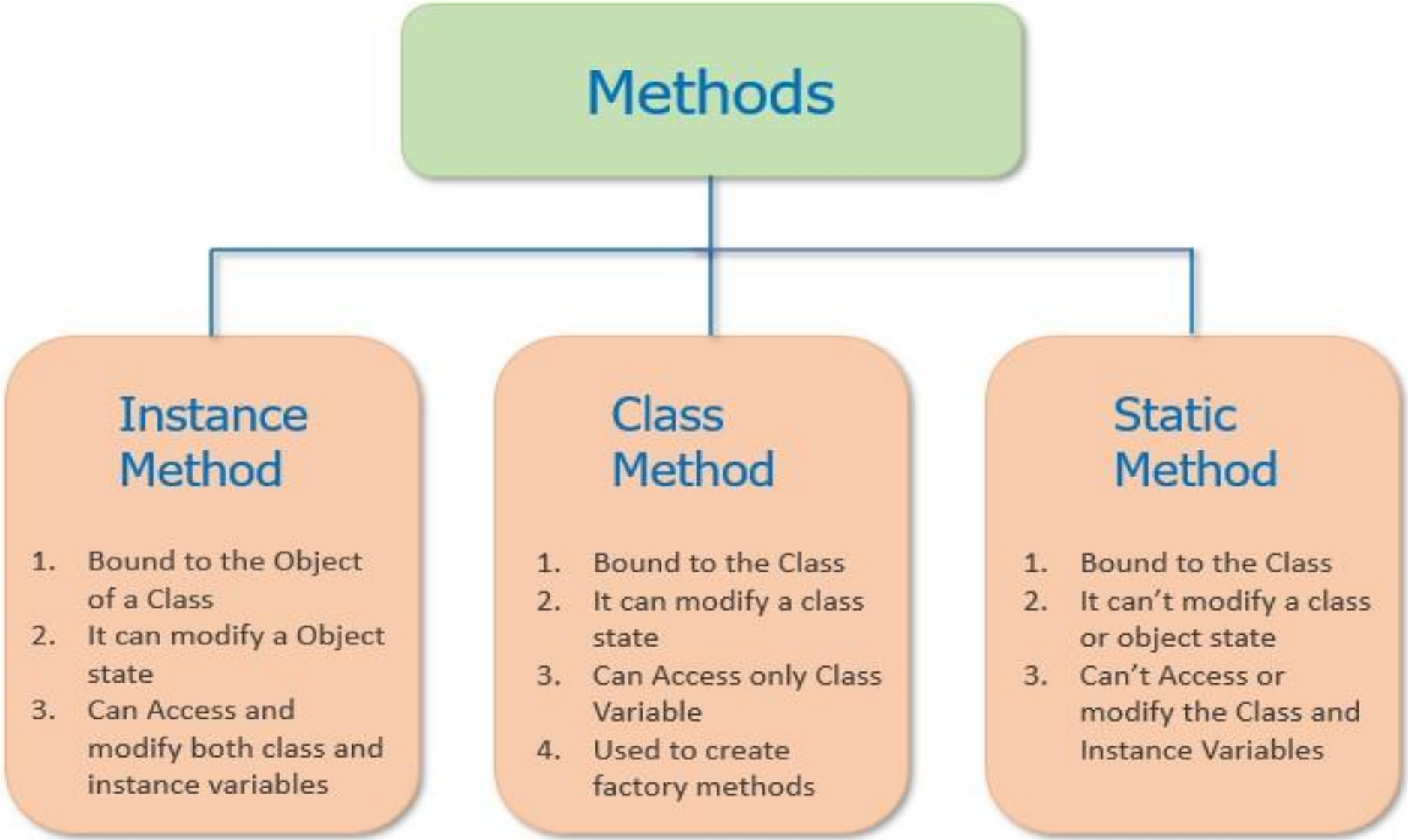
To use wildcard search in Python, the re library must be included into the programme. A library used to work with Regular Expressions in Python is called the re library, which is an acronym for the term Regular Expression.
To do the search, we will compile a list of words, and after that, we will employ the re library functions. With the aid of wildcards, we will locate a match with a correct word.

The Python code that follows conducts a wildcard search.

```
import re
str = re.compile('hel.o')
a = ['hello', 'welcome', 'to', 'java', 'point']
match_is = [string for string in a if re.match_is(str, string)]
print(match_is)
```

Python Class Method vs. Static Method vs. Instance Method

Instance method performs a set of actions on the data/value provided by the instance variables. If we use instance variables inside a method, such methods are called instance methods.
Class method is method that is called on the class itself, not on a specific object instance. Therefore, it belongs to a class level, and all class instances share a class method.
Static method is a general utility method that performs a task in isolation. This method doesn't have access to the instance and class variable.


Methods
Instance Method
1. Bound to the Object of a Class
2. It can modify a Object state
3. Can Access and modify both class and instance variables
Class Method
1. Bound to the Class
2. It can modify a class state
3. Can Access only Class Variable
4. Used to create factory methods
Static Method
1. Bound to the Class
2. It can't modify a class or object state
3. Can't Access or modify the Class and Instance Variables

Example

```
class Student:
    # class variables
    school_name = 'ABC School'
    # constructor
    def __init__(self, name, age):
        # instance variables
        self.name = name
        self.age = age
    # instance variables
    def show(self):
        print(self.name, self.age, Student.school_name)
    @classmethod
    def change_School(cls, name):
        cls.school_name = name
    @staticmethod
    def find_notes(subject_name):
        return ['chapter 1', 'chapter 2', 'chapter 3']
```

Method Call

```
# create object
jessa = Student('Jessa', 12)
# call instance method
jessa.show()
# call class method using the class
Student.change_School('XYZ School')
# call class method using the object
jessa.change_School('PQR School')
# call static method using the class
Student.find_notes('Math')
# call class method using the object
jessa.find_notes('Math')
```