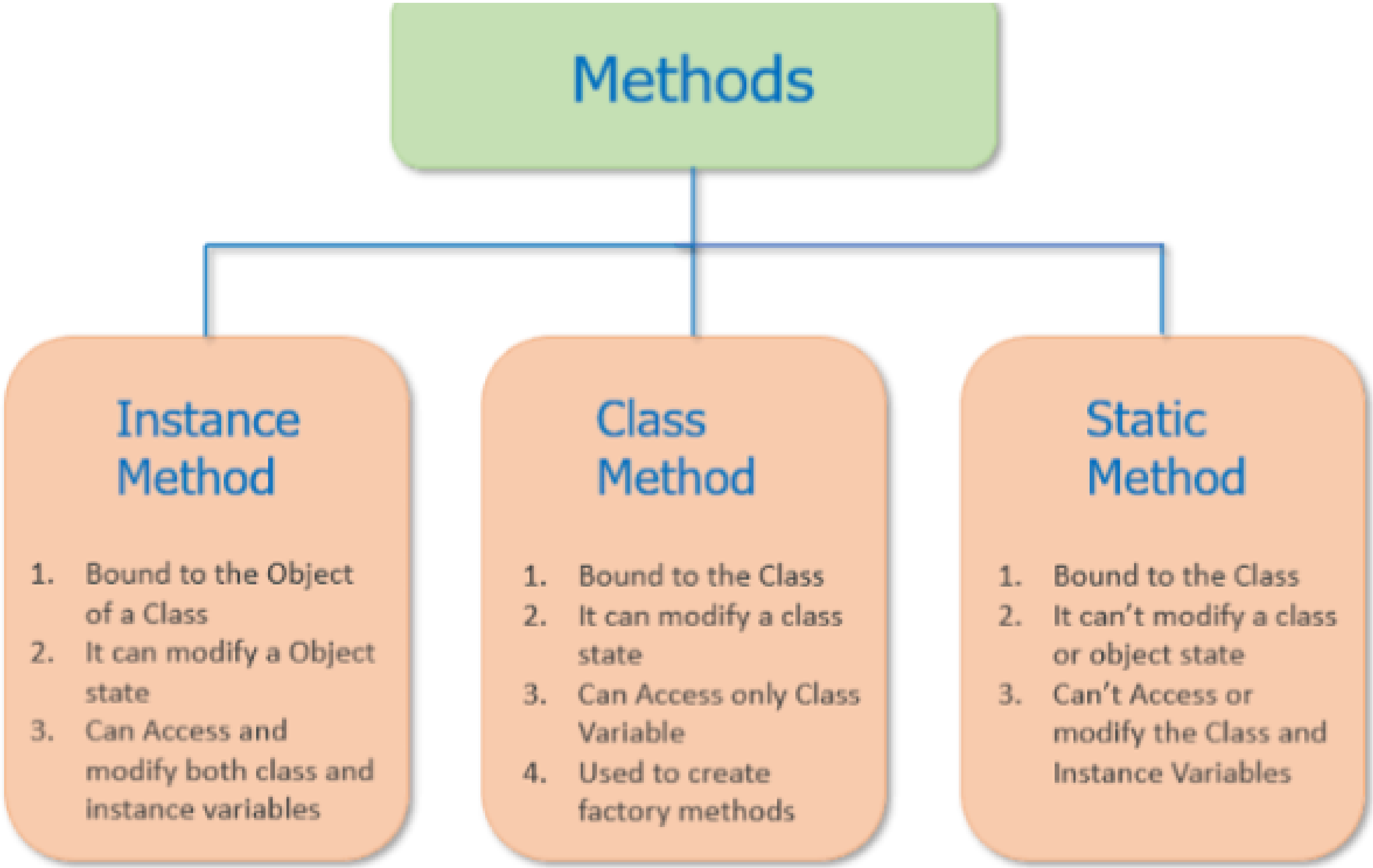
Python

Class Method
Vs
Static Method

python



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

classmethod() in Python

The **classmethod()** is an inbuilt function in Python, which returns a class method for a given function.;

> **Syntax:** classmethod(function)
>
> **Parameter :**This function accepts the function name as a parameter.
>
> **Return Type:**This function returns the converted class method.

Python classmethod()

The classmethod() methods are bound to a class rather than an object. Class methods can be called by both class and object. These methods can be called with a class or with an object.

Class Method vs Static Method

The basic difference between the class method vs Static method in Python and when to use the class method and static method in Python.

- A class method takes cls as the first parameter while a static method needs no specific parameters.
- A class method can access or modify the class state while a static method can't access or modify it.
- In general, static methods know nothing about the class state. They are utility-type methods that take some parameters and work upon those parameters. On the other hand class methods must have class as a parameter.
- We use @classmethod decorator in Python to create a class method and we use @staticmethod decorator to create a static method in Python.

Example of classmethod in Python

Create a simple classmethod

In this example, we are going to see how to create classmethod, for this we created a class with geeks name with member variable course and created a function purchase which prints the object. Now we passed the method geeks.purchase into classmethod which converts the methods to a class method and then we call the class function purchase without creating a function object.

Python3

```python
class geeks:
    course = 'DSA'

    def purchase(obj):
        print('Purchase course : ', obj.course)

geeks.purchase = classmethod(geeks.purchase)
geeks.purchase()
```

Output:

```
Purchase course :  DSA
```

Create class method using classmethod()

Created print_name classmethod before creating this line print_name() It can be called only with an object not with the class now this method can be called as classmethod print_name() method is called a class method.

Python3

```python
class Student:

    # create a variable
    name = "Geeksforgeeks"

    # create a function
    def print_name(obj):
        print("The name is : ", obj.name)

# create print_name classmethod
# before creating this line print_name()
# It can be called only with object not with class
Student.print_name = classmethod(Student.print_name)

# now this method can be called as classmethod
# print_name() method is called a class method
Student.print_name()
```

Output:

```
The name is :  Geeksforgeeks
```

Factory method using a Class method

Uses of classmethod() function are used in factory design patterns where we want to call many functions with the class name rather than an object

Python3

```python
# Python program to demonstrate
# use of a class method and static method.
from datetime import date

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # a class method to create a
    # Person object by birth year.
    @classmethod
    def fromBirthYear(cls, name, year):
        return cls(name, date.today().year - year)

    def display(self):
        print("Name : ", self.name, 'Age : ', self.age)

person = Person('mayank', 21)
person.display()
```

Output:

```
Name :  mayank Age :  21
```

Python @classmethod Decorator

The @classmethod decorator is a built-in function decorator which is an expression that gets evaluated after your function is defined. The result of that evaluation shadows your function definition. A class method receives the class as the implicit first argument, just like an instance method receives the instance.

Syntax of classmethod Decorator

```
class C(object):
  @classmethod
    def fun(cls, arg1, arg2, ...):
     ....
```

Where,

- **fun:** the function that needs to be converted into a class method
- **returns:** a class method for function.

Note:

- A class method is a method that is bound to the class and not the object of the class.
- They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
- It can modify a class state that would apply across all the instances of the class. For example, it can modify a class variable that would be applicable to all instances.

Example

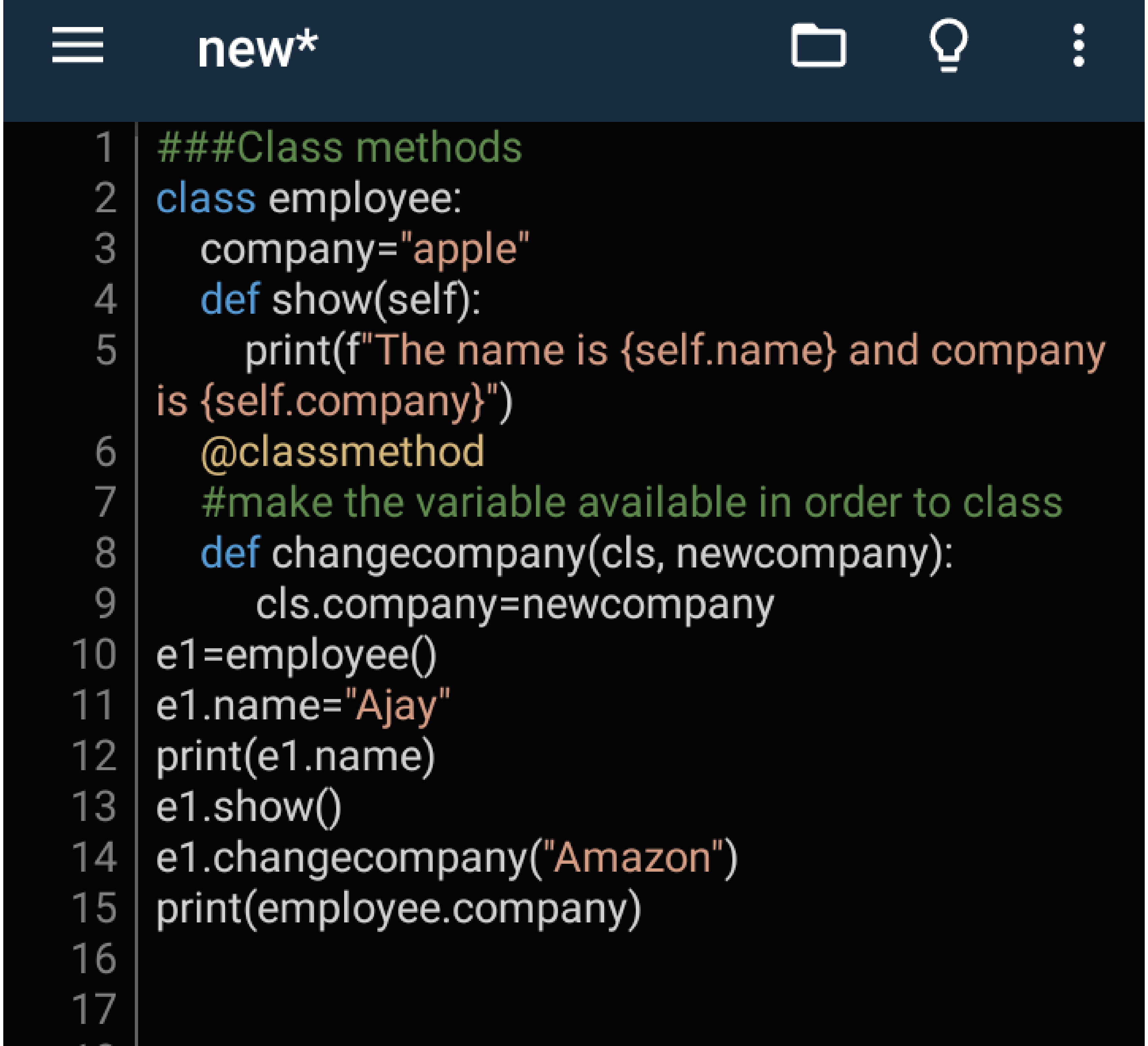
In the below example, we use a staticmethod() and classmethod() to check if a person is an adult or not.

Python3

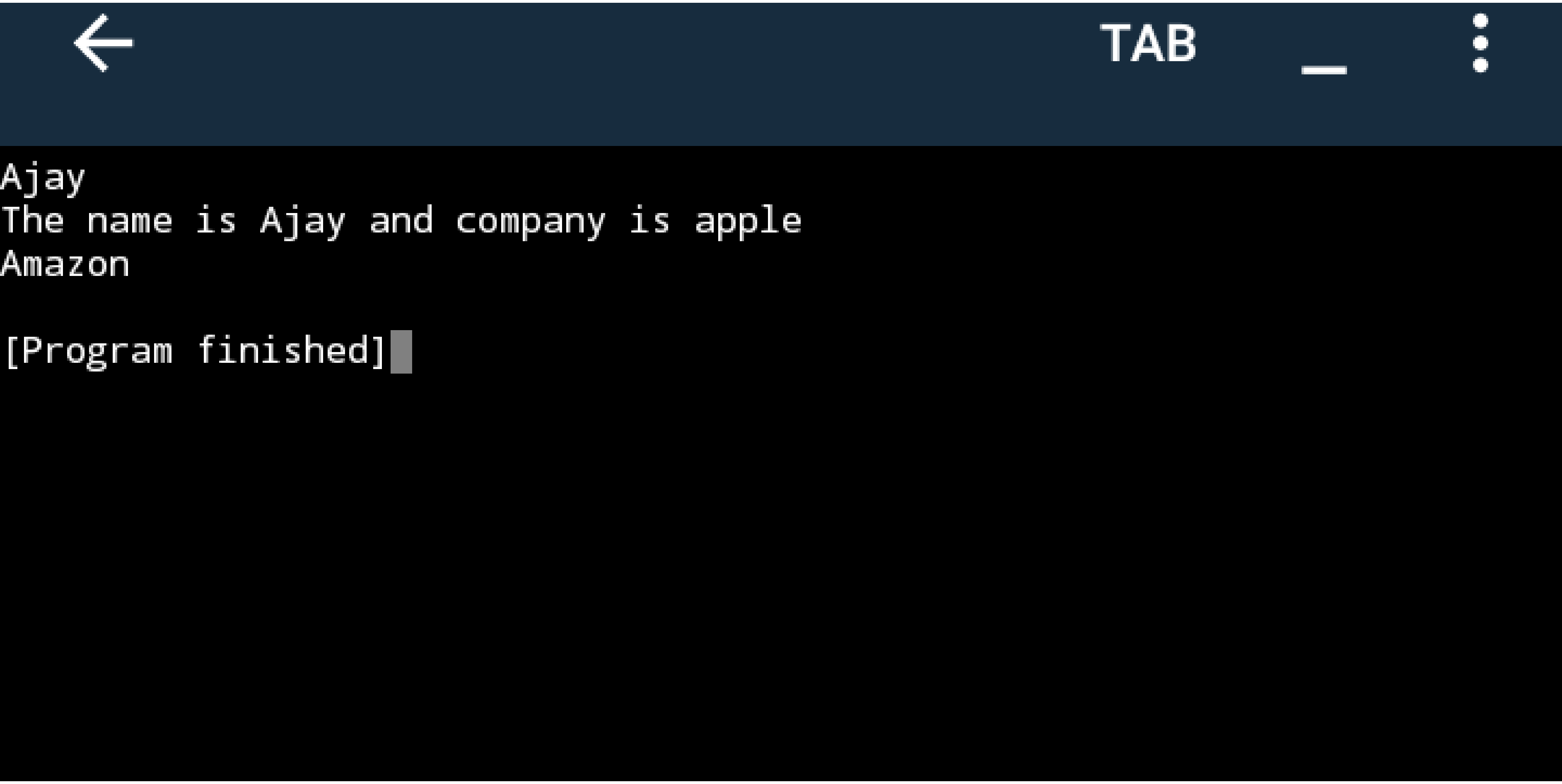
```
# Python program to demonstrate
# use of a class method and static method.
from datetime import date

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # a class method to create a
    # Person object by birth year.
    @classmethod
    def fromBirthYear(cls, name, year):
        return cls(name, date.today().year - year)

    # a static method to check if a
    # Person is adult or not.
    @staticmethod
    def isAdult(age):
        return age > 18

person1 = Person('mayank', 21)
person2 = Person.fromBirthYear('mayank', 1996)

print(person1.age)
print(person2.age)

# print the result
print(Person.isAdult(22))
```

Output:

```
21

25

True
```



```python
###Class methods
class employee:
    company="apple"
    def show(self):
        print(f"The name is {self.name} and company is {self.company}")
    @classmethod
    #make the variable available in order to class
    def changecompany(cls, newcompany):
         cls.company=newcompany
e1=employee()
e1.name="Ajay"
print(e1.name)
e1.show()
e1.changecompany("Amazon")
print(employee.company)
```



```
Ajay
The name is Ajay and company is apple
Amazon

[Program finished]
```

Static methods in Python

In actuality, you define utility methods or group functions that have some logical relationships in a class using static methods.

It is very similar to defining a regular function to define static methods inside of a class.

Static methods in Python, in comparison to instance methods, are not bound to an object. In other words, object state cannot be accessed or changed by static methods. Additionally, Python does not automatically give the self or cls parameters to static methods. Therefore, the state of the class cannot be accessed or changed by static methods.

Example

The @staticmethod decorator is used to define static methods –

```
class name_of_class:
   @staticmethod
   def name_of_static_method(parameters_list):
      pass
print ('static method defined')
```

Edit & Run

Output

The output of the above code is mentioned below –

```
static method defined
```

Calling a static method

Without having to create a class instance, a static method can be called directly from the class. Only static variables can be accessed by a static method; instance variables are not accessible.

Syntax

The following syntax is used for calling a static method −

```
Name_of_class.static_method_name()
```

Example

Following is an example for calling a static method using Name_of_class.static_method_name() and by using an object of the class –

Edit & Run

```
class Animal:
   @staticmethod
   def test(a):
      print('static method', a)

# calling a static method
Animal.test(12)
# calling using object
anm = Animal()
anm.test(12)
```

Output

Following is an output of the above code –

```
static method 12
static method 12
```

Example

Calling Static method from another method

We will now examine the process of calling a static method from another static method of the same class. Here, we'll distinguish between a static method and a class method:

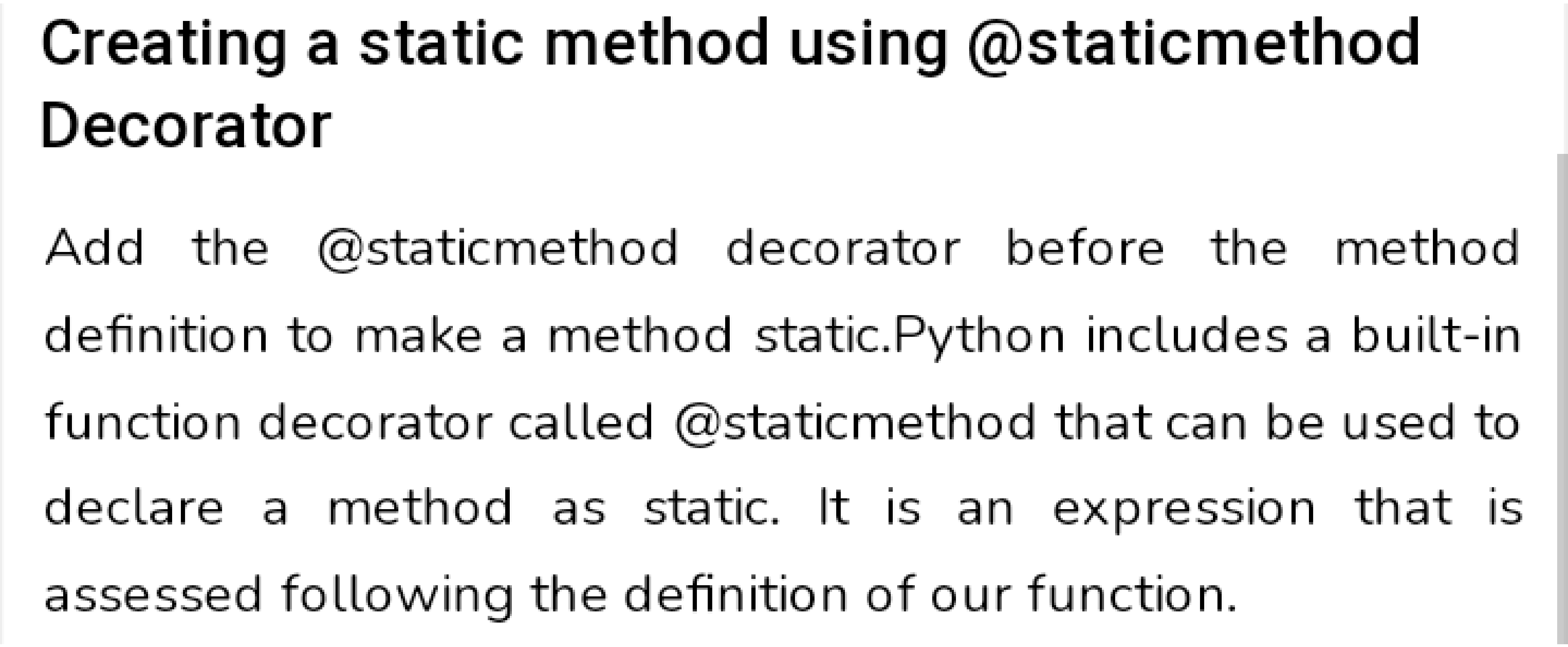
Edit & Run

```
class Animal :
   @staticmethod
   def first_static_method():
      print('first_static_method')
   @staticmethod
   def second_static_method() :
      Animal.first_static_method()
   @classmethod
   def class_method(cls) :
      cls.second_static_method()
# calling the class method
Animal.class_method()
```

Output

Following is an output of the above code –

```
first_static_method
```



Creating a static method using @staticmethod Decorator

Add the @staticmethod decorator before the method definition to make a method static.Python includes a built-in function decorator called @staticmethod that can be used to declare a method as static. It is an expression that is assessed following the definition of our function.

Example

Static methods are a particular type of method. There are times when you'll write code that is part of a class but doesn't use the actual object at all. It is a utility method and can function without an object (self parameter). Since it is static, we declare it that way. Additionally, we can call it from another class method.

As an illustration, let's develop a static method called “information()” that takes a “kingdom" and returns a list of all the requirements that need to be fulfilled for that kingdom –

Output

Following is an output of the above code −

```
Information collected lion
Information collected Pantheria
Information collected Animalia
```

The staticmethod() function

Some programs may define static methods the old-fashioned way by calling staticmethod() as a function rather than a decorator.

If you need to support previous versions of Python (2.2 and 2.3), then you should only define static methods using the staticmethod() function. The @staticmethod decorator is advised in all other cases.

Syntax

Following is the syntax of staticmethod() function –

```
staticmethod(function)
```

where,

The method you want to change to a static method is called function.It returns the static method that was transformed.

Example

Following is an example of staticmethod() function –

Edit & Run

```
class Animal:
    def test(a):
        print('static method', a)
# converting to the static method
Animal.test = staticmethod(Animal.test)
# calling the static method
Animal.test(5)
```

Output

Following is an output of the above code –

```
static method 5
```

Note – When you require a reference to a function from a class body but don't want the automatic conversion to an instance method, the staticmethod() approach can be useful.