```python
"""
OOP - Object Oriented Programming

1. Class
   1. A class is a logical entity. (It doesn't exist in real world).
      It is like a blueprint for creating objects.

Syntax:
class <class_name>:
    attr1 = None
    attr2 = None

    def <method_name>(self):
        pass

    def <method_name>(self):
        pass


Naming Convention for class name:
1. Class name should start with either alphabet or an underscore.
2. Class name can contain alphabets, numbers or an underscore.
3. No Special characters are allowed inside class name.
4. Spaces are also not allowed inside class name

PEP8 Convention:
1. we use Upper Camel Case -
Ex: my class name - MyClassName

2. Object
   An Object is a Physical entity. (It exists in real world.)
   An Object has properties(attributes) and behaviours (methods).
Syntax:
<object_name> = <class_name>()
To Access the class properties and methods, we use .(dot) notation.

3. Inheritance
4. Abstraction
5. Encapsulation
6. Polymorphism
"""
class Student:
    name = None
    age = None
    email_id = None

    def print_name(self):
        print("My Name is ", self.name)

    def print_age(self):
        print("My Age is ", self.age)

    def print_email_id(self):
        print("My Email Address is: ", self.email_id)

    def print_details(self):
        self.print_name()
        self.print_age()
        self.print_email_id()
```

```
s1 = Student()
s1.name = "Rama"
s1.age = 20
s1.email_id = "rama@gmail.com"

s2 = Student()
s2.name = "Krishna"
s2.age = 25
s2.email_id = "krishna@gmail.com"

s1.print_name()
s2.print_name()

s1.print_age()
s2.print_age()

s1.print_email_id()
s2.print_email_id()

s1.print_details()
print()
s2.print_details()


"""
class Foo:
    attr1 = None
    attr2 = None

    def method1(myself):
        print(myself.attr1, myself.attr2)

f = Foo()
f.attr1 = "Value1 of Object1"
f.attr2 = "Value2 of Object1"

f.method1()
"""



"""
Constructor
A Constructor is used to initialize the values for attributes in a class.

Syntax:
class <class_name>:
    def __init__(self):
        body

Note:
1. Constructor is also a method of a class.
2. Constructor is automatically invoked when an object is created.
"""


class Student:
    def __init__(self, name, age, email_id):
        print("Constructor Invoked")
        self.name = name
        self.age = age
        self.email_id = email_id

    def print_name(self):
        print("My Name is ", self.name)

    def print_age(self):
        print("My Age is ", self.age)

    def print_email_id(self):
        print("My Email Address is: ", self.email_id)

    def print_details(self):
        self.print_name()
        self.print_age()
        self.print_email_id()


print("Before Objects Creation...")
s1 = Student("Rama", 20, "rama@gmail.com")
s2 = Student("Krishna", 25, "krishna@gmail.com")
print("After Objects Creation...")

s1.print_name()
s2.print_name()

s1.print_age()
s2.print_age()

s1.print_email_id()
s2.print_email_id()

s1.print_details()
print()
s2.print_details()


class Foo:
    def __init__(self, value1, value2):
        self.attr1 = value1
        self.attr2 = value2


    def print_attributes(self):
        print(self.attr1, self.attr2)


f = Foo("Value1", "Value2")
f.print_attributes()


# Is Instance
l1 = []
print(isinstance(l1, list)) # True
print(isinstance(l1, tuple)) # False
print(isinstance(f, Foo)) # True
print(isinstance(f, Student)) # False
print(isinstance(s1, Student)) # True
```