```
"""
Inheritance:
To Inherit the attributes and methods from a parent class.

Use:
DRY - Don't Repeat Yourself

Types of Inheritance:
1. Single Inheritance
2. Hierarchical Inheritance
3. Multi Level Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance

Terms:
1. Base Class / Parent Class / Super Class:
   The Class from which the attributes and methods are being inherited.
2. Child Class / Sub Class / Derived Class:
   The Class to which the attributes and methods are being inherited.

Syntax:
class <child_class_name>(<parent_class1>, <parent_class2>):
    # Body

Rules:
Inheritance should only be used when there is a "is - a" relationship
for parent and child.
Ex: Lion is an animal
<child class> is a <parent class>
"""
class Foo:
    def __init__(self):
        self.attr1 = 10
        self.attr2 = 20

    def method1(self):
        print("Inside Method1")

    def method2(self):
        print("Inside Method2")


class Bar(Foo):
    pass


b = Bar()
print(b.attr1)
print(b.attr2)
b.method1()
b.method2()


class Parent:
    attr1 = 10
    attr2 = 20

    def method1(self):
        print("Inside Method1 of Parent Class")
```

```
class Child(Parent):
    attr3 = 30
    attr4 = 40

    def method2(self):
        print("Inside Method 2 of Child Class")


c = Child()
print(c.attr1)
print(c.attr2)
print(c.attr3)
print(c.attr4)

c.method1()
c.method2()


p = Parent()
print(p.attr1)
print(p.attr2)
p.method1()
# print(p.attr3)
# p.method2()


# Super Keyword
# It is used to invoke the parent class methods and attributes.
class Parent:
    def __init__(self, value1, value2):
        self.attr1 = value1
        self.attr2 = value2

    def method1(self):
        print("Inside Method1 of Parent")


class Child(Parent):
    def method2(self):
        print("Inside Method2 of Child")


c = Child(10, 20)
print(c.attr1)
print(c.attr2)


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

    def method1(self):
        print("Inside Method1 of Parent")


class Child(Parent):
```

```
    def __init__(self, value1, value2, value3, value4):
        super().__init__(value1, value2)
        self.attr3 = value3
        self.attr4 = value4

    def method2(self):
        super().method1()
        print("Inside Method2 of Child")


c = Child(100, 200, 300, 400)
print(c.attr1)
print(c.attr2)
print(c.attr3)
print(c.attr4)
c.method2()


# Types of Inheritance
# 1. Single Inheritance
class Person:
    def __init__(self, name, age, blood_group, mobile_number):
        self.name = name
        self.age = age
        self.blood_group = blood_group
        self.mobile_number = mobile_number


    def walk(self):
        print("Person will walk")

    def run(self):
        print("Person will eat...")


class Student(Person):
    def __init__(self, name, age, blood_group, mobile_number, regd_number,
standard, educational_institution):
        super().__init__(name, age, blood_group, mobile_number)
        self.registered_number = regd_number
        self.standard = standard
        self.educational_institution = educational_institution


    def read(self):
        print("Students will read....")

    def write_exam(self):
        print("Students write exams....")


student1 = Student("S1", 20, "A +ve", 838237323, "r12345", 10, "VTalent")
print(student1.name)
print(student1.age)
print(student1.blood_group)
print(student1.mobile_number)
print(student1.registered_number)
print(student1.standard)
print(student1.educational_institution)
```

```
student1.walk()
student1.run()
student1.read()
student1.write_exam()


# 2. Hierarchical Inheritance
class Person:
    def __init__(self, name, age, blood_group, mobile_number):
        self.name = name
        self.age = age
        self.blood_group = blood_group
        self.mobile_number = mobile_number


    def walk(self):
        print("Person will walk")

    def run(self):
        print("Person will eat...")


class Student(Person):
    def __init__(self, name, age, blood_group, mobile_number, regd_number,
standard, educational_institution):
        super().__init__(name, age, blood_group, mobile_number)
        self.registered_number = regd_number
        self.standard = standard
        self.educational_institution = educational_institution


    def read(self):
        print("Students will read....")

    def write_exam(self):
        print("Students write exams....")


class Teacher(Person):
    def __init__(self, name, age, blood_group, mobile_number, emp_id,
educational_institution):
        super().__init__(name, age, blood_group, mobile_number)
        self.emp_id = emp_id
        self.educational_institution = educational_institution


    def conduct_exams(self):
        print("Teachers conduct exams...")

    def evaluate_answers(self):
        print("Teachers evaluate answers...")

class Sportsman(Person):
    def __init__(self, name, age, blood_group, mobile_number, sport):
        super().__init__(name, age, blood_group, mobile_number)
        self.sport = sport
```

```
    def practice_sport(self):
        print(f"Practicing {self.sport}")


    def play(self):
        print(f"Play {self.sport}")


print("-----------------Hierarchical Inheritance-------------")
student1 = Student("S1", 20, "A +ve", 838237323, "r12345", 10, "VTalent")
print(student1.name)
print(student1.age)
print(student1.blood_group)
print(student1.mobile_number)
print(student1.registered_number)
print(student1.standard)
print(student1.educational_institution)

student1.walk()
student1.run()
student1.read()
student1.write_exam()


teacher = Teacher("T1", 30, "AB +ve", 9347920301, "E12345", "VTalent")
print(teacher.name)
print(teacher.age)
print(teacher.blood_group)
print(teacher.mobile_number)
print(teacher.emp_id)
print(teacher.educational_institution)

teacher.conduct_exams()
teacher.evaluate_answers()


sports_person = Sportsman("S1", 30, "B -ve", 938492931, "Cricket")
print(sports_person.name)
print(sports_person.age)
print(sports_person.blood_group)
print(sports_person.mobile_number)
print(sports_person.sport)


sports_person.practice_sport()
sports_person.play()


# Multi Level Inheritance
class Vehicle:
    def __init__(self, type):
        self.type = type

    def method1(self):
        print(self.type)
        print("Inside Vehicle Class")

class RoadWays(Vehicle):
    def __init__(self, vehicle):
```

```
        super().__init__("Roadways")
        self.vehicle = vehicle

    def method2(self):
        print(self.vehicle)
        print("Inside Roadways class")


class TwoWheeler(RoadWays):
    def __init__(self):
        super().__init__("Two Wheeler")

    def method3(self):
        print("Inside Two Wheeler class")


tw = TwoWheeler()
tw.method1()
tw.method2()
tw.method3()


# Multiple Inheritance
class Herbivores:
    def eat_plants(self):
        print("We eat plant food")

class Carnivores:
    def eat_meat(self):
        print("We eat meat")

class Omnivores(Herbivores, Carnivores):
    def eat_food(self):
        super().eat_plants()
        super().eat_meat()

o = Omnivores()
o.eat_food()


class Herbivores:
    def eat(self):
        print("We eat plants")


class Carnivores:
    def eat(self):
        print("We eat meat")

class Omnivores(Herbivores, Carnivores):
    def eat(self):
        super().eat()


o = Omnivores()
o.eat()
print(Omnivores.__mro__)

"""
```

```
MRO (Method Resolution Order)
Method Resolution Order (MRO) in Python defines the order in
which base classes are searched when a method or attribute is
accessed in an instance of a class, especially in the context of
multiple inheritance. It dictates the hierarchy of method and
attribute lookup, ensuring a consistent and predictable behavior.

Search Path:
When you call a method or access an attribute on an object,
Python follows the MRO to find the first occurrence of that method or
attribute in the class hierarchy. The search starts from the class
of the object itself, then proceeds through its parent classes
according to the MRO.
"""
class A:
    def method(self):
        print("Method from A")

class B:
    def method(self):
        print("Method from B")

class C(A, B):
    pass

class D(B, A):
    pass

print(C.__mro__)
# Output: (<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class
'object'>)

print(D.__mro__)
# Output: (<class '__main__.D'>, <class '__main__.B'>, <class '__main__.A'>, <class
'object'>)

c_instance = C()
c_instance.method()  # Calls method from A

d_instance = D()
d_instance.method()  # Calls method from B


# Hybrid Inheritance
class Vehicle:
    def __init__(self, travels_on, engine_cc):
        self.travels_on = travels_on
        self.engine_cc = engine_cc

    def drive(self):
        if self.travels_on == "Road":
            print("Drives on Road")
        elif self.travels_on == "Air":
            print("Flies in Air")
        elif self.travels_on == "Water":
            print("Sails on Water")

class RoadWays(Vehicle):
    def __init__(self, vehicle_type, engine_cc, n_tyres):
```

```
        super().__init__("Road", engine_cc)
        self.vehicle_type = vehicle_type
        self.n_tyres = n_tyres

    def print_wheels(self):
        print(f"{self.vehicle_type} has {self.n_tyres} wheels")

    def gear_up(self):
        print("Increase Speed....")

    def gear_down(self):
        print("Decrease Speed...")

class AirWays(Vehicle):
    def __init__(self, vehicle_type, engine_cc, wings, propellors):
        super().__init__("Air", engine_cc)
        self.vehicle_type = vehicle_type
        self.wings = wings
        self.propellors = propellors

    def onboard(self):
        if self.vehicle_type == "Passenger Plane":
            print("Passengers will be onboarded")
        elif self.vehicle_type == "Cargo Plane":
            print("Goods will be loaded")

    def take_off(self):
        print("Taking off..")

"""
class TwoWheeler(RoadWays):
    def __init__(self, vehicle):
        super().__init__("Two Wheeler")
        self.vehicle = vehicle

    def has_engine(self):
        if self.vehicle == "Bike":
            print("Bike has Engine")
        elif self.vehicle == "Bicycle":
            print("Bicycle has no Engine")
"""
class MotorBike(RoadWays):
    def __init__(self, engine_cc, kick_road):
        super().__init__("MotorBike", engine_cc,2)
        self.kick_road = kick_road


fz = MotorBike(350, "Kick Road")
print(fz.kick_road)
print(fz.vehicle_type)
print(fz.n_tyres)
print(fz.travels_on)
print(fz.engine_cc)

fz.print_wheels()
fz.gear_up()
fz.gear_down()
fz.drive()
```

```
indigo = AirWays("Passenger Plane", 800, 2, 4)
print(indigo.vehicle_type)
print(indigo.wings)
print(indigo.propellors)
print(indigo.travels_on)
print(indigo.engine_cc)

indigo.onboard()
indigo.take_off()
indigo.drive()


class Television:
    def __init__(self, display_type, screen_size):
        self.display_type = display_type
        self.screen_size = screen_size


    def display_stream(self):
        print("Playing...")

    def power_on(self):
        print("Television Powering On....")


class Computer:
    def __init__(self, os_type, os_version):
        self.os_type = os_type
        self.os_version = os_version

    def access_internet(self):
        print("Access the Internet")

    def power_on(self):
        print("Computer Powering On....")


class SmartTV(Television, Computer):
    def __init__(self, display_type, screen_size, os_type, os_version):
        Television.__init__(self, display_type, screen_size)
        Computer.__init__(self, os_type, os_version)

    def play_ott_videos(self):
        print("Play OTT Vides...")

    def power_on(self):
        Television.power_on(self)
        Computer.power_on(self)

class LED(SmartTV):
    def __init__(self, screen_size, os_type, os_version):
        super().__init__("LED", screen_size, os_type, os_version)

    def display_in_LED(self):
        print("Displaying in LED")

samsung_tv = LED(65, "Android", "9.1.0")
print(samsung_tv.display_type)
print(samsung_tv.screen_size)
```

```
print(samsung_tv.os_type)
print(samsung_tv.os_version)


samsung_tv = LED(34, "Android", "8.2.0")
print(samsung_tv.display_type)
print(samsung_tv.screen_size)
print(samsung_tv.os_type)
print(samsung_tv.os_version)
samsung_tv.power_on()


# MRO
class A:
    def method(self):
        print("Method inside class A")


    def method1(self):
        print("Method1 inside class A")


class B:
    def method(self):
        print("Method inside class B")

    def method2(self):
        print("Method2 inside class B")

class C(A, B):
    def method(self):
        print("Method inside C")
        super().method()
        A.method(self)
        B.method(self)

c = C()
c.method1()
c.method2()
c.method()
# print(C.__mro__)


class Foo:
    def __init__(self, a, b):
        self.a = a
        self.b = b

class Bar:
    def __init__(self, c, d):
        self.c = c
        self.d = d

class Baz(Foo, Bar):
    def __init__(self, a, b, c, d, e, f):
        """
        super().__init__(a, b)
        super().__init__(c, d)
        """
```

```
        Foo.__init__(self, a, b)
        Bar.__init__(self, c, d)
        self.e = e
        self.f = f


baz_obj = Baz(1, 2, 3, 4, 5, 6)
print(baz_obj.a, baz_obj.b, baz_obj.c, baz_obj.d, baz_obj.e, baz_obj.f)
```