# Python Examples: super() in Single Inheritance

## 1. Order and OnlineOrder

class Order:  
 def \_\_init\_\_(self, order\_id):  
 self.order\_id = order\_id  
  
 def process\_payment(self):  
 print(f"Processing payment for Order #{self.order\_id}")  
  
class OnlineOrder(Order):  
 def \_\_init\_\_(self, order\_id, email):  
 super().\_\_init\_\_(order\_id)  
 self.email = email  
  
 def process\_payment(self):  
 super().process\_payment()  
 print(f"Sending confirmation email to {self.email}")  
  
order = OnlineOrder(101, "customer@example.com")  
order.process\_payment()

1. Order is the base class or a parent class with a method to process a payment.
2. OnlineOrder is the derived class or the child class that adds an email feature.
3. Super keyword is used to accesses the base class information to the child class .super().\_\_init\_\_(order\_id) initializes order\_id from the parent class.
4. super().process\_payment() calls the base class’s method.
5. Adds email confirmation logic in the child’s process\_payment().

## 2. Employee and Manager

class Employee:  
 def \_\_init\_\_(self, name, salary):  
 self.name = name  
 self.salary = salary  
  
 def display(self):  
 print(f"Name: {self.name}, Salary: ₹{self.salary}")  
  
class Manager(Employee):  
 def \_\_init\_\_(self, name, salary, department):  
 super().\_\_init\_\_(name, salary)  
 self.department = department  
  
 def display(self):  
 super().display()  
 print(f"Department: {self.department}")  
  
m = Manager("Shaik", 90000, "IT")  
m.display()

1. Employee defines basic employee details.
2. Manager adds an extra department field.
3. Uses super().\_\_init\_\_() to reuse initialization from Employee.
4. super().display() shows name and salary to display the output .
5. department display is seen on the child class which is added as an additional information to parent.

## 3. Vehicle and Car

class Vehicle:  
 def start(self):  
 print("Vehicle started")  
  
class Car(Vehicle):  
 def start(self):  
 super().start()  
 print("Car is ready to go")  
  
c = Car()  
c.start()

1. Vehicle has a start method.
2. Car inherits from Vehicle.
3. super().start() ensures base functionality is preserved.
4. Then adds car-specific startup message.
5. C = Car() creates an object which is accessed using . to display the output.

## 4. User Login System

class User:  
 def \_\_init\_\_(self, username):  
 self.username = username  
  
 def login(self):  
 print(f"{self.username} logged in")  
  
class Admin(User):  
 def login(self):  
 super().login()  
 print(f"{self.username} has admin privileges")  
  
a = Admin("admin\_user")  
a.login()

1. User has a login method with basic message.
2. Admin inherits from User which is considered to be the parent class.
3. Overrides login() to add additional message.
4. super().login() ensures original login message is printed.
5. Adds admin-specific privileges message.

## 5. Shape and Circle

class Shape:  
 def \_\_init\_\_(self):  
 print("This is a shape")  
  
 def area(self):  
 print("Area formula not defined")  
  
class Circle(Shape):  
 def \_\_init\_\_(self, radius):  
 super().\_\_init\_\_()  
 self.radius = radius  
  
 def area(self):  
 super().area()  
 print("Circle Area:", 3.14 \* self.radius \* self.radius)  
  
c = Circle(5)  
c.area()

1. Shape class provides a default area() method.
2. Circle inherits from Shape and adds a radius.
3. super().\_\_init\_\_() runs Shape constructor.
4. super().area() calls the base method before custom area logic.
5. Circle computes and prints area using its own formula.

## 6. Person and Student

class Person:  
 def \_\_init\_\_(self, name):  
 self.name = name  
  
 def show(self):  
 print(f"Name: {self.name}")  
  
class Student(Person):  
 def \_\_init\_\_(self, name, grade):  
 super().\_\_init\_\_(name)  
 self.grade = grade  
  
 def show(self):  
 super().show()  
 print(f"Grade: {self.grade}")  
  
s = Student("Ali", "A")  
s.show()

1. Person class holds basic personal info(name).
2. Student adds a new property: grade.
3. super().\_\_init\_\_(name) is used to initialize the name in the base class.
4. super().show() displays the name from Person.
5. Student's show() method then adds grade-specific output.

## 7. BankAccount and SavingsAccount

class BankAccount:  
 def \_\_init\_\_(self, balance):  
 self.balance = balance  
  
 def show\_balance(self):  
 print(f"Balance: ₹{self.balance}")  
  
class SavingsAccount(BankAccount):  
 def \_\_init\_\_(self, balance, interest):  
 super().\_\_init\_\_(balance)  
 self.interest = interest  
  
 def show\_balance(self):  
 super().show\_balance()  
 print(f"Interest Rate: {self.interest}%")  
  
acc = SavingsAccount(10000, 5)  
acc.show\_balance()

1. BankAccount contains a balance and a method to show it.
2. SavingsAccount extends it by adding interest rate.
3. super().\_\_init\_\_(balance) is used to reuse parent constructor.
4. super().show\_balance() prints the balance from base class.
5. Then prints the interest rate specific to SavingsAccount.

## 8. Product and ElectronicProduct

class Product:  
 def \_\_init\_\_(self, name):  
 self.name = name  
  
 def details(self):  
 print(f"Product: {self.name}")  
  
class ElectronicProduct(Product):  
 def \_\_init\_\_(self, name, warranty):  
 super().\_\_init\_\_(name)  
 self.warranty = warranty  
  
 def details(self):  
 super().details()  
 print(f"Warranty: {self.warranty} years")  
  
p = ElectronicProduct("Laptop", 2)  
p.details()

1. Product class defines a product name and method to display it.
2. ElectronicProduct adds a warranty attribute.
3. Uses super().\_\_init\_\_(name) to initialize name in parent class.
4. super().details() displays product info from Product.
5. Adds extra details about warranty in the child class.

## 9. Animal and Dog

class Animal:  
 def sound(self):  
 print("Animal sound")  
  
class Dog(Animal):  
 def sound(self):  
 super().sound()  
 print("Dog barks")  
  
d = Dog()  
d.sound()

1. Animal class has a sound() method with generic message.
2. Dog inherits from Animal and sound().
3. Calls super().sound() to include the base message.
4. Then prints dog-specific message (Dog barks).
5. Shows how super() can be used even when no constructor is involved.

## 10. Book and EBook

class Book:  
 def \_\_init\_\_(self, title):  
 self.title = title  
  
 def show(self):  
 print(f"Title: {self.title}")  
  
class EBook(Book):  
 def \_\_init\_\_(self, title, file\_size):  
 super().\_\_init\_\_(title)  
 self.file\_size = file\_size  
  
 def show(self):  
 super().show()  
 print(f"File Size: {self.file\_size} MB")  
  
eb = EBook("Python Guide", 5)  
eb.show()

1. Book holds a title and method to show it.
2. EBook adds file\_size attribute for digital content.
3. super().\_\_init\_\_(title) initializes the title via parent class.
4. super().show() displays title from Book.

5. Adds display of file size specific to EBook.