Object Oriented Programming

Reference Document for Python Syntax

CONTENTS

CONTENTS ii
1. Classes and Objects 1
1.1. Declaring a Class 1
1.2. Creating an object of a Class 1
1.3. Constructor 1
1.4. Initializing and accessing instance variable 1
1.5. Parameterized Constructor 1
1.6. Declaring and invoking method 2
1.7. Access Specifier: Declaring and accessing of public and private variables 2
1.8. Example 2
2. Static variables and methods 3
2.1. Declaring static variable and method 3
2.2. Example 3
3. Aggregation 3
3.1. Syntax 3
3.2. Example 4
4. Composition 4
4.1. Syntax 4
4.2. Example 5
5. Association 5
5.1. Syntax 5
5.2. Example 6
6. Inheritance 6
6.1. Syntax 6
6.2. Using super() 6
6.3. Method Overriding 7
6.4. Example 7
6.5. Multiple Inheritance - Syntax 8
7. Abstract Class and Method 8
7.1. Syntax 8
7.2. Example 9

1. Classes and Objects

1.1. Declaring a Class

```
class ClassName:
    #block of statements
```

1.2. Creating an object of a Class

```
object_name =ClassName()
```

1.3. Constructor

```
class ClassName:
      def __init__(self):
          #block of statements
```

1.4. Initializing and accessing instance variable

```
class ClassName:
        def __init__(self):
            self.instance_variable_name1=None
            self.instance_variable_name2=None

            object_name=ClassName()
            object_name.instance_variable_name1=value
            object_name.instance_variable_name2=value
```

1.5. Parameterized Constructor

```
class ClassName:
       def __init__(self, variable):
           self.instance_variable=variable

object_name=ClassName(variable_value)
print(object_name.instance_variable)
```

1.6. Declaring and invoking method

```
class ClassName:
      def method_name(self,argument1):
          #block of statements
object_name=ClassName()
object_name.method_name(value1)
```

1.7. Access Specifier: Declaring and accessing of public and private variables

```
class ClassName:
      def __init__(self):
          self.public_instance_variable=None
          self.__private_instance_variable=None
      def get_private_instance_variable(self):
          return self. private_instance_variable

object_name=ClassName()
print(object_name.public_instance_variable)
print(object_name.get_private_instance_variable()
```

1.8. Example

```
class Employee:
    def __init__(self,emp_name):
        self.emp_name= None
        self.__salary=emp_name

    def display(self):
        print("Employee name:",self.emp_name,"salary: ",self.__salary)

employee=Employee(20000)
employee.emp_name= "john"
employee.display()
```

2. Static variables and methods

2.1. Declaring static variable and method

```
class ClassName:
      __static_variable_name=value

      @staticmethod
      def static_method_name():
            #block of statements
```

2.2. Example

```
class Demo:
    __static_variable_name=100

    @staticmethod
    def display():
        print(Demo.__static_variable_name)

Demo.display()
```

3. Aggregation

3.1. Syntax

```
class ClassName1:
      def __init__(self,variable):
            self.__instance_variable= variable

class ClassName2:
      def __init__(self,object):
            self.__instance_variable==object

object1=ClassName1(value)
object2=ClassName2(object1)
```

3.2. Example

```
class Employee:
    def __init__(self,emp_no,emp_name):
        self.__emp_no=emp_no
        self.__emp_name=emp_name
    def get_emp_no(self):
        return self.__emp_no
    def get_emp_name(self):
        return self.__emp_name

class Department:
    def __init__(self,dept_name,employee):
        self.__dept_name=dept_name
        self.__employee=employee
    def display(self):
        print("Department Name : ", self.__dept_name)
        print("Employee Number :", self.__employee.get_emp_no())
        print("Employee Name   :",self.__employee.get_emp_name())


employee=Employee(1001,"John")
department=Department("ETA",employee)
department.display()
```

4. Composition

4.1. Syntax

```
class ClassName1:
      def __init__(self,variable):
            self.__instance_variable=variable
       def method_name(self):
            #block of statements


class ClassName2:
      def __init__(self,variable1,variable2):
            self.__instance_variable=variable1
            self.__object=ClassName1(variable2)

     def method_name(self):
            self.__object.method_name()

object= ClassName2(value1,value2)
object.method_name()
```

4.2. Example

```
class University:
    def __init__(self,university_name,dept_name):
        self.__university_name= university_name
        self.__department=Department(dept_name)

    def display_details(self):
        print("University Name: ", self.__university_name)
        self.__department.display()

class Department:
    def __init__(self,dept_name):
        self.__dept_name=dept_name

    def display(self):
        print("Department Name: ", self.__dept_name)

university=University("VTU", "Computer Science")
university.display_details()
```

5. Association

5.1. Syntax

```
class ClassName1:
      def __init__(self,variable):
           self.instance_variable=variable
      def method_name(self):
           #block of statements


cass ClassName2:
      def __init__(self):
           #block of statements

      def method_name(self,object):
           object.method_name()
           print(object. instance_variable)
           #block of statements

object1=ClassName1(value)
object2=ClassName2()
object2.method_name(object1)
```

5.2. Example

```
class Cycle:
    def __init__(self,color):
        self.__color=color
    def display(self):
        print(self.__color)

class Employee:
    def __init__(self,emp_name):
        self.__emp_name=emp_name
    def display(self,cycle):
        print(self.__emp_name)
        cycle.display()

emp=Employee("John")
cycle=Cycle("Red")
emp.display(cycle)
```

6. Inheritance

6.1. Syntax

```
class Parent:
      def method_name1(self):
          #block of statements

class Child(Parent):
      def method_name2(self):
          #block of statements

child=Child()
child.method_name2()

parent=Parent()
parent.method_name1()
```

6.2. Using super()

```
class Parent:
    def __init__(self,variable1):
        self.__parent_instance_variable=variable1
    def get_parent_instance_variable(self):
        return self.__parent_instance_variable
```

```
class Child(Parent):
    def __init__(self,variable1,variable2):
        super().__init__(variable1)
        self.child_instance_variable=variable2
    def display(self):
        print(self.child_instance_variable)
        print(self.get_parent_instance_variable())

child=Child(value1,value2)
child.display()
```

6.3. Method Overriding

```
class Parent:
    def method_name(self):
        print("Parent method")

class Child(Parent):
    def method_name(self):
        super().method_name()
        print("Child method")

child=Child()
child.method_name()
```

6.4. Example

```
class Employee:
    def __init__(self,emp_id,emp_name):
        self.__emp_id=emp_id
        self.__emp_name=emp_name
    def display(self):
        print("Employee Id      : ", self.__emp_id)
        print("Employee Name    : ", self.__emp_name)

class PermanentEmployee(Employee):
    def
        __init__(self,emp_id,emp_name,salary,bonus):
        super().__init__(emp_id,emp_name)
        self.__salary=salary
        self.__bonus=bonus
    def display(self):
        super().display()
        print("Employee salary  : ", self.__salary+self.__bonus)

class TemperoryEmployee(Employee):
    def __init__(self,emp_id,emp_name,daily_wages,number_of_days):
        super().__init__(emp_id,emp_name)
```

```
        self.__daily_wages=daily_wages
        self.__number_of_days=number_of_days
    def display(self):
        super().display()
        print("Employee salary  : ", self.__daily_wages * self.__number_of_days)

emp1=PermanentEmployee(1000,"John",50000,12000)
emp1.display()
emp2=TemperoryEmployee("T101","James",1200,20)
emp2.display()
```

6.5. Multiple Inheritance - Syntax

```
class BaseClass1:
    def method_name1(self):
        #block of statements
class BaseClass2:
    def method_name2(self):
        #block of statements
class DerivedClass(BaseClass1, BaseClass2):
    def method_name3(self):
        #block of statements

object=DerivedClass()
object.method_name1()
object.method_name2()
object.method_name3()
```

7. Abstract Class and Method

7.1. Syntax

```
from abc import ABCMeta, abstractmethod
class BaseClass(metaclass=ABCMeta):
    @abstractmethod
    def method_name(self):
        pass

class DerivedClass(BaseClass):
    def __init__(self):
        pass

    def method_name(self):
        pass
```

```
object=DerivedClass()
object.method_name()
```

7.2. Example

```
from abc import ABCMeta, abstractmethod
class Employee(metaclass=ABCMeta):
    def __init__(self,emp_id,emp_name):
        self.__emp_id=emp_id
        self.__emp_name=emp_name
        self.__salary=None

    @abstractmethod
    def calculate_salary(self):
        pass

     def set_salary(self,salary):
        self.__salary=salary
    def get_salary(self):
        return self.__salary
    def display(self):
        print("Employee Id      : ", self.__emp_id)
        print("Employee Name    : ", self.__emp_name)

class PermanentEmployee(Employee):
    def __init__(self,emp_id,emp_name,basic_salary,bonus):
        super().__init__(emp_id,emp_name)
        self.__basic_salary=basic_salary
        self.__bonus=bonus

    def calculate_salary(self):
        salary=self.__basic_salary+self.__bonus
        self.set_salary(salary)

    def display(self):
        super().display()
        print("Employee salary  : ", self.get_salary())

class TemperoryEmployee(Employee):
    def __init__(self,emp_id,emp_name,daily_wages,number_of_days):
        super().__init__(emp_id,emp_name)
        self.__daily_wages=daily_wages
        self.__number_of_days=number_of_days

    def calculate_salary(self):
        salary=self.__daily_wages*self.__number_of_days
        self.set_salary(salary)

    def display(self):
        super().display()
        print("Employee salary  : ", self.get_salary())
```

```
emp1=PermanentEmployee(1000,"John",50000,12000)
emp1.calculate_salary()
emp1.display()
emp2=TemperoryEmployee("T101","James",1200,20)
emp2.calculate_salary()
emp2.display()
```