Object-Oriented Programming Concepts

Yunke Mike Gan

August 8, 2021

1 Python Test/Implementation of Some OOP Concepts

Please find the code here.
Note: I meant to leave errors in the code to show some possible errors.

2 General OOP Questions

- What is OOP? / OOP Definition
- Why OOP? / OOP Pros
- Main Features of OOP
  Inheritance, Encapsulation, Polymorphism, Data Abstraction
- Difference between OOP and Structural Programming
- What is an object?
- What is a class?
- Difference between class and structure
- Difference between class and object

3 Inheritance

- What is Inheritance?
- Different types of Inheritance
- Difference between Multiple Inheritance and Multi-level Inheritance
- Hybrid Inheritance
- Hierarchical Inheritance
- Limitations of Inheritance
- Super Class / Parent Class / Base Class
- Sub Class / Child Class / Derived Class

4 Polymorphism

- What is Polymorphism?
- Static Polymorphism / Compile-time Polymorphism / Static Binding
- Dynamic Polymorphism / Run-time Polymorphism / Dynamic Binding
- Method Overloading
- Method Overriding
- Difference between Method Overloading and Method Overriding

5 Encapsulation

- What is Encapsulation?
- Access Specifier / Access Modifier
- Difference between Public/Private/Protected Access Modifiers

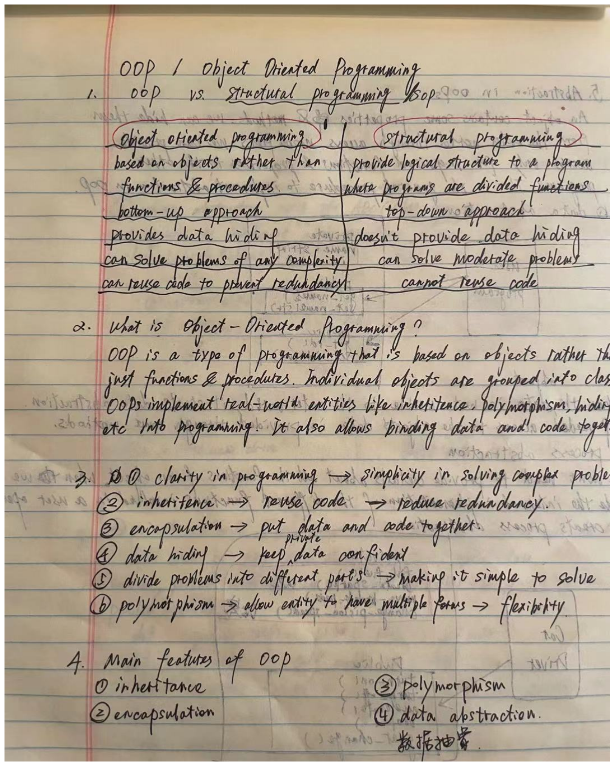
6 Data Abstraction

- What is Data Abstraction?
- Abstract Class
- Abstract Method / Abstract Function
- Interface
- Difference between Data Abstraction and Encapsulation
- Difference between Abstract Class and Interface

7 OOP Methods and Functions

- Virtual Functions / Overriding Functions
- Pure Virtual Functions / Abstract Functions
- Constructor
- Destructor
- Default/Parameterized/Copy/Static/Private Constructor
- Finalize Method
- Garbage Collection

8 OOP Manual Scripts



OOP / Object Oriented Programming

1. OOP vs. Structural programming / SOP

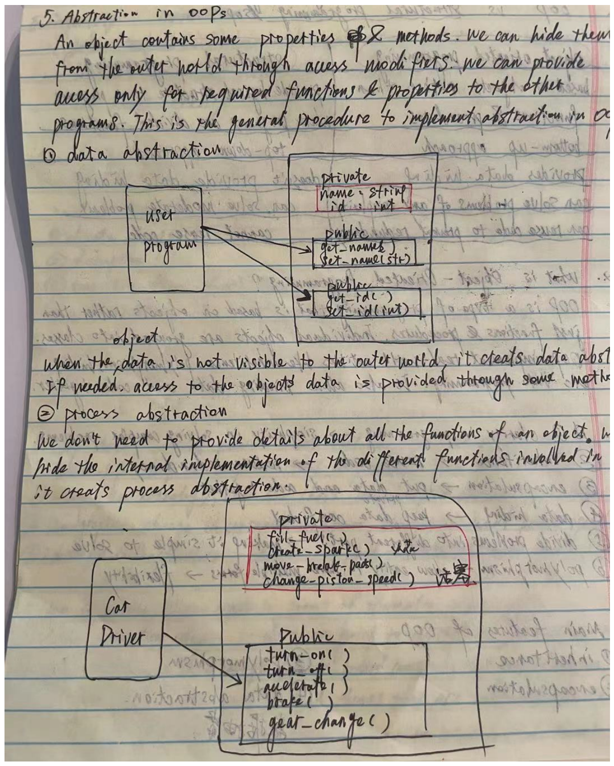
Object oriented programming	Structural programming
based on objects rather than functions & procedures	provide logical structure to a program where programs are divided functions
bottom-up approach	top-down approach
provides data hiding	doesn't provide data hiding
can solve problems of any complexity	can solve moderate problems
can reuse code to prevent redundancy	cannot reuse code

2. What is Object-Oriented Programming?

OOP is a type of programming that is based on objects rather th
just functions & procedures. Individual objects are grouped into clas
OOPs implement real-world entities like inheritence, polymorphism, hidin
etc into programming. It also allows binding data and code toget

3. ~~①~~ ① clarity in programming → simplicity in solving complex proble
② inheritence → reuse code → reduce redundancy.
③ encapsulation → put data and code together.
④ data hiding → keep $\overset{\text{private}}{\text{data}}$ confident
⑤ divide problems into different parts → making it simple to solve
⑥ polymorphism → allow entity to have multiple forms → flexibility.

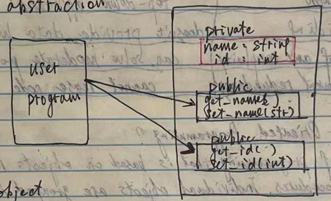
4. Main features of OOP
① inheritance
② encapsulation
③ polymorphism
④ data abstraction.
数据抽象.



5. Abstraction in OOPs

An object contains some properties ~~of~~ & methods. We can hide them
from the outer world through access modifiers. We can provide
access only for required functions & properties to the other
programs. This is the general procedure to implement abstraction in o

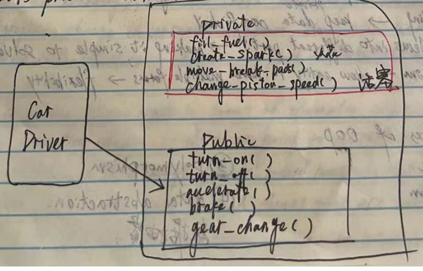
① data abstraction

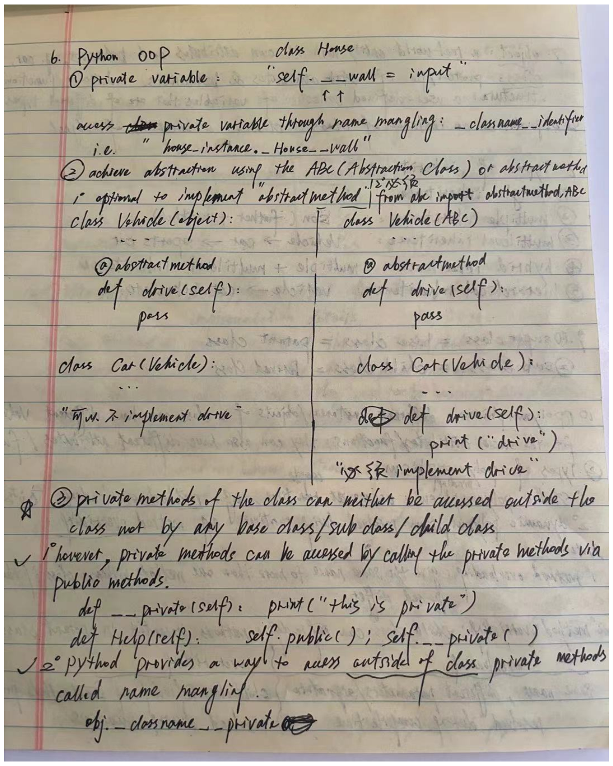


when the $\overset{\text{object}}{\text{data}}$ is not visible to the outer world, it creats data abst
If needed, access to the objects data is provided through some metho

② process abstraction

We don't need to provide details about all the functions of an object. W
hide the internal implementation of the different functions involved in
it creats process abstraction.





6. Python OOP

class House

① private variable : "self. __wall = input"

access ~~class~~ private variable through name mangling : _classname__identifier

i.e. " house_instance._House__wall"

② achieve abstraction using the ABC (Abstraction Class) or abstract method

1° optional to implement "abstractmethod" | 2° 必须 from abc import abstractmethod, ABC

```
class Vehicle(object):              |  class Vehicle(ABC)
    ...                             |      ...
    @abstractmethod                 |      @abstractmethod
    def drive(self):                |      def drive(self):
        pass                        |          pass

class Car(Vehicle):                 |  class Car(Vehicle):
    ...                             |      ...
"可以不 implement drive"             |      ~~def~~ def drive(self):
                                    |          print("drive")
                                    |  "必须 implement drive"
```

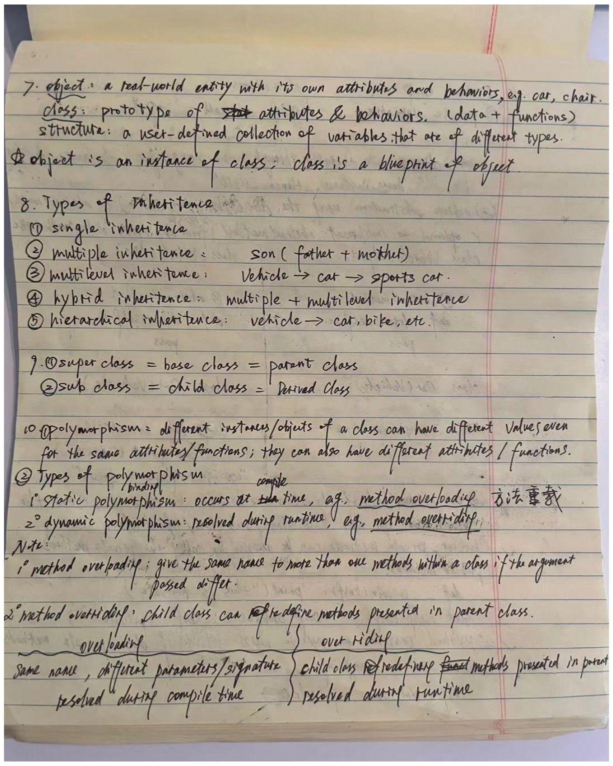
③ private methods of the class can neither be accessed outside the class nor by any base class / sub class / child class

✓ 1° however, private methods can be accessed by calling the private methods via public methods.

```
def __private(self): print("this is private")
def Help(self):      self.public() ; self.__private()
```

✓ 2° pythod provides a way to access outside of class private methods called name mangling.

obj._classname__private



7. object : a real-world entity with its own attributes and behaviors, e.g. car, chair.

class : prototype of ~~stat~~ attributes & behaviors. (data + functions)

structure : a user-defined collection of variables that are of different types.

object is an instance of class ; class is a blueprint of object

8. Types of Inheritence

① single inheritence

② multiple inheritence : son (father + mother)

③ multilevel inheritence : Vehicle → car → sports car.

④ hybrid inheritence : multiple + multilevel inheritence

⑤ hierarchical inheritence : vehicle → car, bike, etc.

9. ① super class = base class = parent class

② sub class = child class = Derived Class

10 ① polymorphism = different instances/objects of a class can have different values even for the same attributes/functions ; they can also have different attributes / functions.

② Types of polymorphism

1° static (binding) polymorphism : occurs at ~~the~~ compile time, e.g. method overloading 方法重载

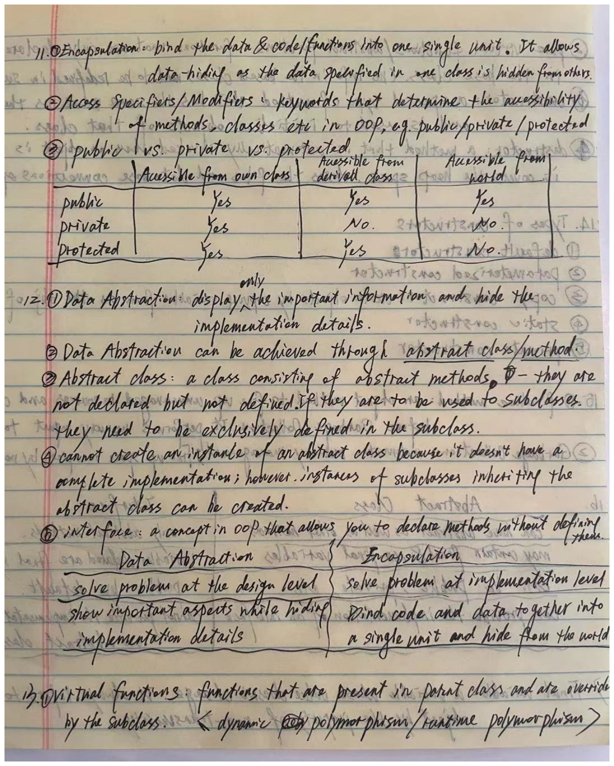
2° dynamic polymorphism : resolved during runtime, e.g. method overriding.

Note:

1° method overloading : give the same name to more than one methods within a class if the argument passed differ.

2° method overriding : child class can ~~ref~~ redefine methods presented in parent class.

overloading	overriding
same name, different parameters / signature	child class ~~ref~~ redefining ~~fun~~ methods presented in parent
resolved during compile time	resolved during runtime



11. ① Encapsulation: bind the data & code/functions into one single unit. It allows data-hiding as the data specified in one class is hidden from others.

② Access Specifiers/Modifiers: keywords that determine the accessibility of methods, classes etc in OOP, e.g. public/private/protected

③ public vs. private vs. protected

	Accessible from own class	Accessible from derived class	Accessible from world
public	Yes	Yes	Yes
private	Yes	No.	No.
protected	Yes	Yes	No.

12. ① Data Abstraction: display only the important information and hide the implementation details.

② Data Abstraction can be achieved through abstract class/method.

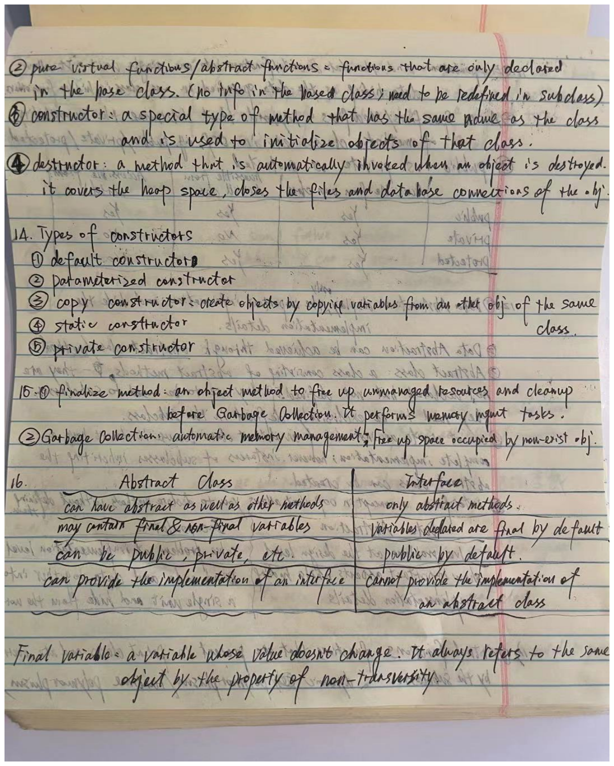
③ Abstract class: a class consisting of abstract methods. ~~Θ~~ - they are not declared but not defined. If they are to be used to subclasses. they need to be exclusively defined in the subclass.

④ cannot create an instance of an abstract class because it doesn't have a complete implementation; however. instances of subclasses inheriting the abstract class can be created.

⑤ interface: a concept in OOP that allows you to declare methods without defining them.

Data Abstraction	Encapsulation
solve problem at the design level	solve problem at implementation level
show important aspects while hiding implementation details	Bind code and data together into a single unit and hide from the world

13. ① virtual functions: functions that are present in parent class and are override by the subclass. < dynamic ~~poly~~ polymorphism/runtime polymorphism >



② pure virtual functions/abstract functions: functions that are only declared in the base class. (no info in the based class; need to be redefined in subclass)

③ constructor: a special type of method that has the same name as the class and is used to initialize objects of that class.

④ destructor: a method that is automatically invoked when an object is destroyed. it covers the heap space, closes the files and database connections of the obj.

14. Types of constructors
① default constructor
② parameterized constructor
③ copy constructor: create objects by copying variables from another obj of the same class.
④ static constructor
⑤ private constructor

15. ① finalize method: an object method to free up unmanaged resources and cleanup before Garbage Collection. It performs memory mgmt tasks.

② Garbage Collection: automatic memory management; free up space occupied by non-exist obj.

16.

Abstract Class	Interface
can have abstract as well as other methods	only abstract methods.
may contain final & non-final variables	variables declared are final by default
can be public, private, etc.	public by default.
can provide the implementation of an interface	cannot provide the implementation of an abstract class

Final variable: a variable whose value doesn't change. It always refers to the same object by the property of non-transversity.