ষষ্ঠ অধ্যায়

অবজেক্ট অরিয়েন্টেড

tutorialspointbd.com

অবজেক্ট অরিয়েন্টেড পাইথন (OOP)

অবজেক্ট-ওরিয়েন্টেড প্রোগ্রামিং (OOP) হল একটি প্রোগ্রামিং প্যারাডাইম বা প্রোগ্রামিং শৈলী, যা বিশেষভাবে বড় এবং জটিল সফটওয়্যার সিস্টেম তৈরি করার জন্য কার্যকরী। এটি ডেটা এবং ফাংশনগুলিকে একত্রিত করার জন্য "অবজেক্ট" নামক মৌলিক ধারণা ব্যবহার করে, যার মধ্যে ডেটা (অ্যাট্রিবিউট) এবং কার্যকলাপ (মেথড) থাকে। OOP হল একধরনের অ্যাপ্লিকেশন ডিজাইন এবং ডেভেলপমেন্ট শৈলী, যা ডেটা এবং কোডের ম্যানিপুলেশন (কর্মসূচী) একত্রিত করতে সহায়তা করে। এটি মূলত ডেটার উপর ভিত্তি করে কাজ করে, যেখানে প্রতিটি ডেটা একটি অবজেক্ট হিসেবে আচরণ করে, এবং অবজেক্টগুলি একে অপরের সাথে মিথস্ক্রিয়া করতে পারে।

OOP এর জন্ম এবং ইতিহাস

OOP এর ধারণা প্রথম ১৯৬০-এর দশকে উন্নত হতে শুরু করে, যখন প্রোগ্রামিংয়ের প্রচলিত পদ্ধতি (যেমন ফাংশনাল প্রোগ্রামিং) অনেক বড় এবং জটিল সফটওয়্যার সিস্টেমের জন্য অপর্যাপ্ত হয়ে পড়েছিল। সেই সময় "Simula" নামক একটি প্রোগ্রামিং ভাষা উদ্ভাবিত হয়, যা অবজেক্ট-ভিত্তিক ধারণাকে প্রথম সিস্টেম atically প্রবর্তন করে। তারপর, ১৯৭০-এর দশকে C++ এবং ১৯৮০-এর দশকে জাভা প্রোগ্রামিং ভাষা OOP ধারণাকে আরও উন্নত এবং জনপ্রিয় করে তোলে।

OOP এর মৌলিক ধারণা

OOP এর মূল ভিত্তি হল অবজেক্ট এবং ক্লাস। অবজেক্ট এক ধরনের "ডেটা ইউনিট", যার মধ্যে ডেটা (অ্যাট্রিবিউট) এবং আচরণ (মেথড) থাকে। ক্লাস হল অবজেক্ট তৈরির একটি টেমপ্লেট বা ব্লুপ্রিন্ট। ক্লাস দ্বারা ডিফাইন করা কাঠামো অনুযায়ী অবজেক্ট তৈরি করা হয়।

ক্লাস এবং অবজেক্ট

- **ক্লাস:** ক্লাস একটি সাঁজানো কাঠামো, যা অবজেক্ট তৈরির জন্য প্রয়োজনীয় গঠন এবং আচরণ নির্ধারণ করে। এটি একটি ব্লুপ্রিন্ট যা অভ্যন্তরীণভাবে অবজেক্টের ভেরিয়েবল (অ্যাট্রিবিউট) এবং মেথডস সংরক্ষণ করে।

- **অবজেক্ট:** অবজেক্ট একটি কনক্রিট এক্সামপল বা ইনস্ট্যান্স ক্লাসের। এটি ক্লাসের আদলে তৈরি হয় এবং ক্লাসের ডেটা এবং আচরণ ধারণ করে।

ইনহেরিট্যান্স (Inheritance):

এক ক্লাস অন্য ক্লাসের বৈশিষ্ট্য (অ্যাট্রিবিউট এবং মেথড) উত্তরাধিকারসূত্রে গ্রহণ করতে পারে, এই পদ্ধতিকে বলা হয় ইনহেরিট্যান্স। এর মাধ্যমে কোড পুনঃব্যবহারযোগ্যতা সহজ হয় এবং প্রোগ্রামের মডুলারিটি বৃদ্ধি পায়।

এনক্যাপসুলেশন (Encapsulation):

একটি অবজেক্টের ডেটা এবং ফাংশনগুলো একত্রিত করা এবং বাহ্যিক অ্যাক্সেস নিয়ন্ত্রণ করা। এনক্যাপসুলেশন সাহায্যে অবজেক্টের অভ্যন্তরীণ ডেটা নিরাপদ থাকে এবং অপ্রয়োজনীয় তথ্য থেকে ব্যবহারকারীকে বিচ্ছিন্ন রাখা যায়।

অ্যাবস্ট্রাকশন (Abstraction):

অবজেক্টের ভিতরের জটিলতা লুকিয়ে রেখে শুধুমাত্র প্রাসঙ্গিক তথ্য ব্যবহারকারীকে উপস্থাপন করার প্রক্রিয়া। অ্যাবস্ট্রাকশন মূলত ডেটা এবং কার্যকলাপের সঠিক ধারণা ব্যবহারকারীর কাছে সরবরাহ করতে সহায়তা করে, যার ফলে কোড সহজ এবং বোঝার জন্য আরও সহজ হয়।

পলিমরফিজম (Polymorphism):

একাধিক আকারে বা ফর্মে একাধিক মেথড বা ফাংশন কার্যকর করার ক্ষমতা। এর মাধ্যমে একই নামের মেথড একাধিক ক্লাসে ভিন্নভাবে কাজ করতে পারে।

OOP এর উপকারিতা

- কোড পুনঃব্যবহারযোগ্যতা (Code Reusability): একবার তৈরি করা ক্লাস এবং অবজেক্ট গুলি পুনরায় বিভিন্ন প্রোগ্রামে ব্যবহার করা যায়। এটি ডেভেলপমেন্টের সময় এবং খরচ কমিয়ে আনে।

- মডুলারিটি (Modularity): OOP কোডকে ছোট ছোট অংশে বিভক্ত করতে সাহায্য করে। প্রতিটি অবজেক্ট নির্দিষ্ট একটি দায়িত্ব পালন করে, যার ফলে কোড বেশি সুসংহত এবং স্থিতিস্থাপক হয়ে ওঠে।

- রক্ষণাবেক্ষণ সহজ (Maintainability): যখন কোড মডুলার হয় এবং অবজেক্টের মধ্যে ডেটা এবং আচরণ একত্রিত থাকে, তখন প্রোগ্রামকে আপডেট বা রক্ষণাবেক্ষণ করা অনেক সহজ হয়। নতুন ফিচার যুক্ত করা বা বাগ ফিক্স করা সহজ হয়।

- ডেটা নিরাপত্তা (Data Security): এনক্যাপসুলেশন পদ্ধতির মাধ্যমে অবজেক্টের ডেটা নিরাপদ থাকে এবং শুধু নির্দিষ্ট মেথডের মাধ্যমে ডেটা অ্যাক্সেস করা যায়, যা নিরাপত্তা বৃদ্ধি করে।

- সমস্যা সমাধানে সহজ (Ease of Problem Solving): বাস্তব জীবনের জিনিসপত্রের মডেল (যেমন, গাড়ি, মানুষ, পশু ইত্যাদি) ব্যবহার করে প্রোগ্রামিং করা সহজ হয়। এতে সমস্যা সমাধান সহজ এবং আরও বাস্তবসম্মত হয়।

OOP এর চ্যালেঞ্জ

যদিও OOP অনেক সুবিধা প্রদান করে, তবুও কিছু চ্যালেঞ্জ রয়েছে, যেমন:

- **কমপ্লেক্সিটি:** OOP এর ধারণাগুলি, যেমন ইনহেরিট্যান্স, পলিমরফিজম, এবং অ্যাবস্ট্রাকশন, প্রাথমিকভাবে নতুন প্রোগ্রামারদের জন্য কঠিন হতে পারে। কোডকে সঠিকভাবে ডিজাইন করা এবং ক্লাসের মধ্যে সম্পর্ক সঠিকভাবে স্থাপন করা সময়সাপেক্ষ হতে পারে।

- **অতিরিক্ত মেমরি ব্যবহার:** OOP এর মধ্যে অবজেক্ট গুলি আলাদাভাবে সংরক্ষিত হয়, যার ফলে মেমরি ব্যবহারের পরিমাণ বেড়ে যেতে পারে। বড় প্রোগ্রামগুলির ক্ষেত্রে এটি একটি সমস্যা সৃষ্টি করতে পারে।

- **পারফরমেন্স:** কিছু অবজেক্ট-ওরিয়েন্টেড কনসেপ্ট যেমন ইনহেরিট্যান্স এবং পলিমরফিজম প্রোগ্রামের পারফরমেন্সে প্রভাব ফেলতে পারে, বিশেষত যখন অনেকগুলো অবজেক্ট একত্রে কাজ করছে।

OOP এর ভবিষ্যৎ

OOP এর ভবিষ্যৎ উজ্জ্বল, কারণ এটি সফটওয়্যার ডেভেলপমেন্টে এখনও ব্যাপকভাবে ব্যবহৃত হয়। আধুনিক প্রোগ্রামিং ভাষাগুলির মধ্যে (যেমন পাইথন, জাভা, সি++, সি#) OOP একটি কেন্দ্রীয় ভূমিকা পালন করছে। প্রযুক্তির অগ্রগতির সাথে সাথে OOP ধারণাগুলির নতুন প্রয়োগ দেখা যাবে, যেমন মেশিন লার্নিং এবং আর্টিফিশিয়াল ইনটেলিজেন্সের মতো আধুনিক অ্যাপ্লিকেশন ডেভেলপমেন্টে।

অবশেষে, অবজেক্ট-ওরিয়েন্টেড প্রোগ্রামিং প্রোগ্রামারদের আরো কার্যকরী, রক্ষণাবেক্ষণযোগ্য, এবং মডুলার কোড তৈরিতে সাহায্য করে, যা সফটওয়্যার ডেভেলপমেন্টের জগতে একটি অপরিহার্য অংশ হয়ে উঠেছে।

OOP এর মূল ধারণা

অবজেক্ট-ওরিয়েন্টেড প্রোগ্রামিং (OOP) এর মূল ধারণাগুলি সফটওয়্যার ডিজাইন এবং ডেভেলপমেন্টের জন্য একটি শক্তিশালী কাঠামো প্রদান করে। OOP এর মাধ্যমে আমরা বাস্তব বিশ্বের বস্তুর মতো (যেমন, গাড়ি, মানুষ, প্রাণী, ইত্যাদি) সফটওয়্যার অবজেক্ট তৈরি করতে পারি, যেখানে প্রতিটি অবজেক্টের নিজস্ব ডেটা এবং আচরণ থাকে। এর মাধ্যমে প্রোগ্রামিং আরও বাস্তবসম্মত এবং মডুলার হয়ে ওঠে। OOP এর মূল ধারণাগুলি হল এনক্যাপসুলেশন, অ্যাবস্ট্রাকশন, ইনহেরিট্যান্স, এবং পলিমরফিজম। আসুন, এই চারটি ধারণা বিস্তারিতভাবে বুঝে নেওয়া যাক:

এনক্যাপসুলেশন (Encapsulation)

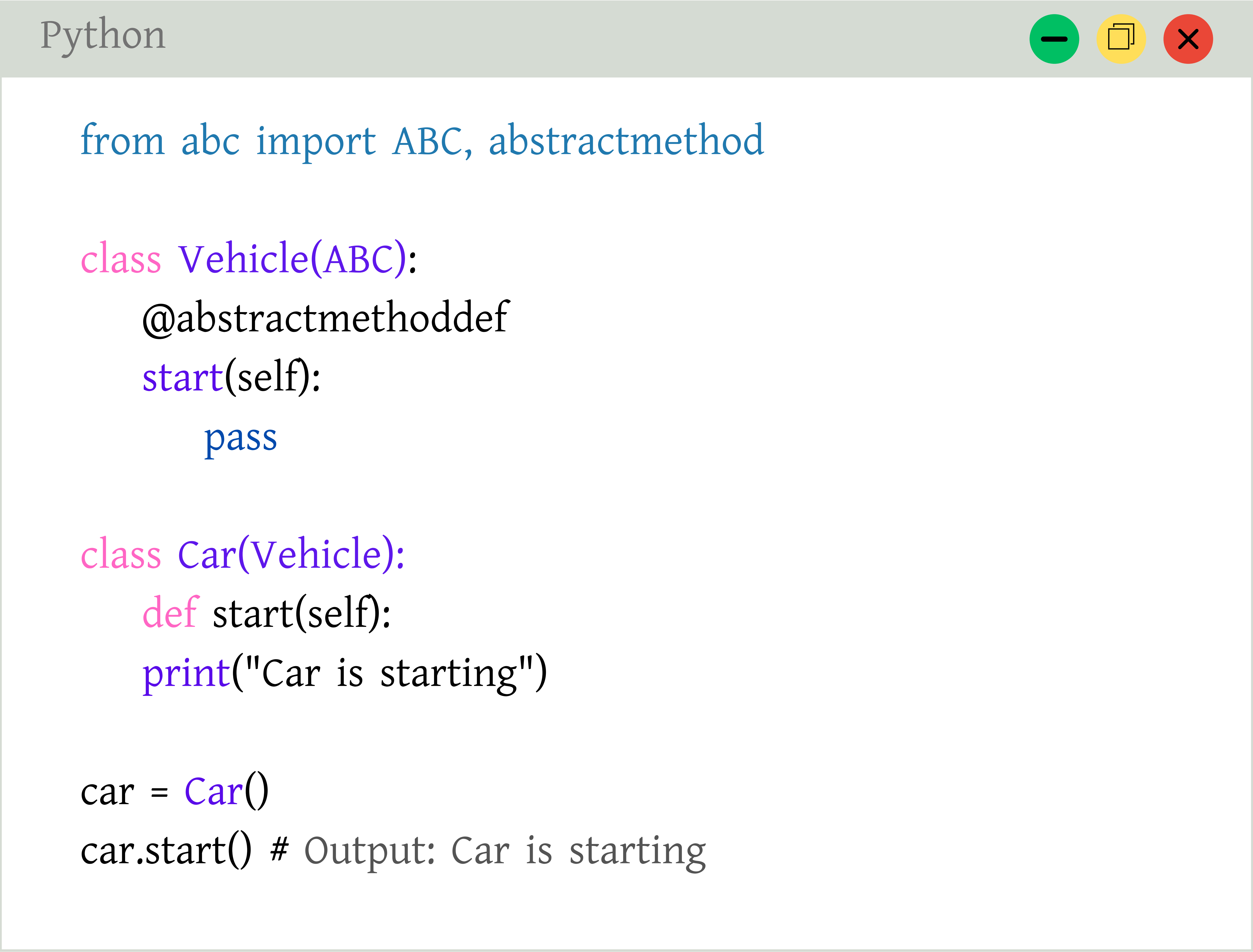
এনক্যাপসুলেশন হল ডেটা হাইডিং বা ডেটাকে একটি অবজেক্টের মধ্যে লুকিয়ে রাখা। এর মাধ্যমে অবজেক্টের অভ্যন্তরীণ ডেটা এবং কার্যক্রম (মেথড) একত্রিত হয় এবং বাহ্যিক কোড থেকে অ্যাক্সেস নিয়ন্ত্রণ করা হয়।

অ্যাবস্ট্রাকশন (Abstraction)

অ্যাবস্ট্রাকশন হল শুধুমাত্র গুরুত্বপূর্ণ তথ্য সরবরাহ করার প্রক্রিয়া এবং অপ্রয়োজনীয় বা জটিল তথ্য লুকিয়ে রাখা। এর মাধ্যমে ব্যবহারকারী শুধুমাত্র প্রয়োজনীয় ফাংশনালিটি দেখতে পায়, যেহেতু অ্যাবস্ট্রাকশন প্রযুক্তি কোডের অভ্যন্তরীণ কার্যকলাপকে লুকিয়ে রাখে।

- **অ্যাবস্ট্রাকশন এর উদ্দেশ্য:** এটি ব্যবহারকারীকে জটিলতা থেকে দূরে রাখে, যাতে তারা সহজভাবে সিস্টেমের সাথে ইন্টারঅ্যাক্ট করতে পারে। অ্যাবস্ট্রাকশন পদ্ধতির সাহায্যে আমরা কোডের বিভিন্ন অংশ আলাদা রাখতে পারি এবং পুনরায় ব্যবহারযোগ্য করে তুলতে পারি।

- **অ্যাবস্ট্রাক্ট ক্লাস এবং মেথড:** অ্যাবস্ট্রাক্ট ক্লাস একটি ক্লাস যা নিজে থেকে ইনস্ট্যান্স তৈরি করতে পারে না। এই ক্লাসটি শুধুমাত্র মেথডের সিগনেচার দেয় এবং নির্দিষ্ট সাবক্লাসে সেই মেথডের বাস্তবায়ন করতে হয়।



```
from abc import ABC, abstractmethod

class Vehicle(ABC):
    @abstractmethoddef
    start(self):
        pass

class Car(Vehicle):
    def start(self):
    print("Car is starting")

car = Car()
car.start() # Output: Car is starting
```

ইনহেরিট্যান্স (Inheritance)

ইনহেরিট্যান্স হল একটি প্রক্রিয়া যেখানে একটি ক্লাস অন্য ক্লাসের বৈশিষ্ট্য (অ্যাট্রিবিউট এবং মেথড) গ্রহণ করে। এইভাবে, আমরা কোড পুনঃব্যবহারযোগ্যতা অর্জন করতে পারি এবং নতুন ক্লাস তৈরির জন্য পূর্ববর্তী ক্লাসের বৈশিষ্ট্য গ্রহণ করতে পারি।

- **ইনহেরিট্যান্স এর উদ্দেশ্য:** ইনহেরিট্যান্স প্রোগ্রামিংয়ের পুনঃব্যবহারযোগ্যতা, কাস্টমাইজেশন এবং সহজ রক্ষণাবেক্ষণ নিশ্চিত করে। এর মাধ্যমে নতুন ক্লাস তৈরি করতে পুরানো ক্লাসের সমস্ত ফিচার ব্যবহার করা যায়।

- **বেস ক্লাস (Super Class):** যেই ক্লাস থেকে বৈশিষ্ট্যগুলি উত্তরাধিকার সূত্রে নেওয়া হয়, তাকে বলা হয় বেস ক্লাস।

- **ডেরিভড ক্লাস (Derived Class):** যেই ক্লাস বেস ক্লাস থেকে বৈশিষ্ট্য গ্রহণ করে, তাকে ডেরিভড ক্লাস বলা হয়।

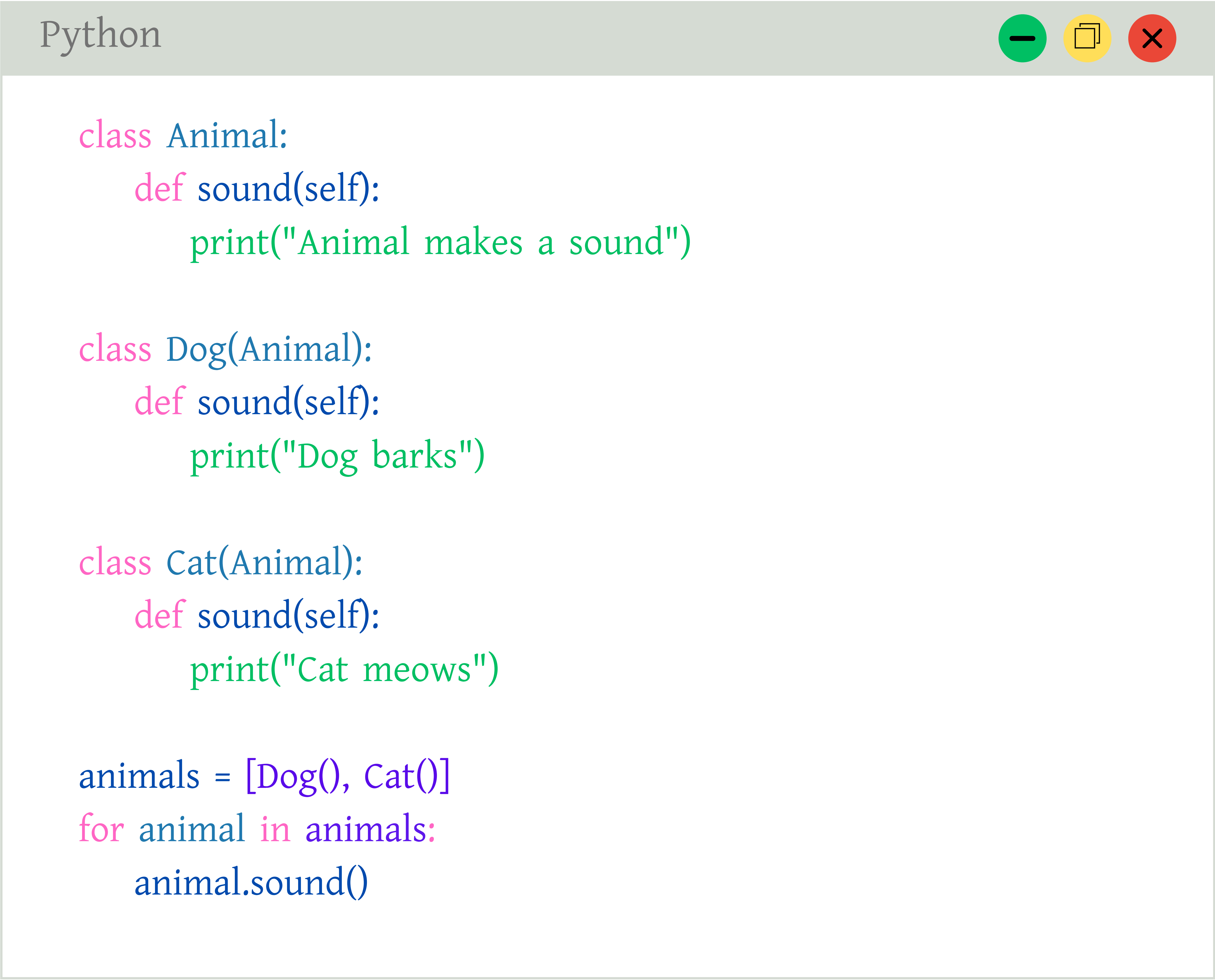
```python
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

dog = Dog()
dog.speak()  # Output: Dog barks
```

পলিমরফিজম (Polymorphism)

পলিমরফিজম হল একাধিক আকার বা ফর্মে একাধিক মেথড বা ফাংশন কার্যকর করার ক্ষমতা। এটি একই নামের মেথড বা ফাংশনকে বিভিন্ন ক্লাসে ভিন্নভাবে বাস্তবায়ন করার সুযোগ দেয়। এতে কোড আরও সাধারণ এবং মডুলার হয়।

- **মেথড ওভাররাইডিং (Method Overriding):** পলিমরফিজমের অন্যতম একটি দিক হল মেথড ওভাররাইডিং, যেখানে বেস ক্লাসের মেথড ডেরিভড ক্লাসে নতুনভাবে বাস্তবায়িত করা হয়।

- **মেথড ওভারলোডিং (Method Overloading):** একাধিক মেথডে একাধিক সিগনেচার থাকতে পারে, যেগুলি একসাথে এক ক্লাসের মধ্যে থাকে। (এটি কিছু ভাষায় সমর্থিত হলেও পাইথনে এটি সরাসরি সমর্থিত নয়।)



```python
class Animal:
    def sound(self):
        print("Animal makes a sound")

class Dog(Animal):
    def sound(self):
        print("Dog barks")

class Cat(Animal):
    def sound(self):
        print("Cat meows")

animals = [Dog(), Cat()]
for animal in animals:
    animal.sound()
```

আউটপুট:

```
Dog barks
Cat meows
```

সারাংশ

OOP এর মূল ধারণাগুলি হল:

- **এনক্যাপসুলেশন (Encapsulation):** ডেটা এবং মেথড একত্রিত করে, বাহ্যিক কোড থেকে নিরাপত্তা এবং কন্ট্রোল প্রদান করা।

- **অ্যাবস্ট্রাকশন (Abstraction):** গুরুত্বপূর্ণ তথ্য প্রদর্শন এবং অপ্রয়োজনীয় তথ্য লুকিয়ে রাখা।

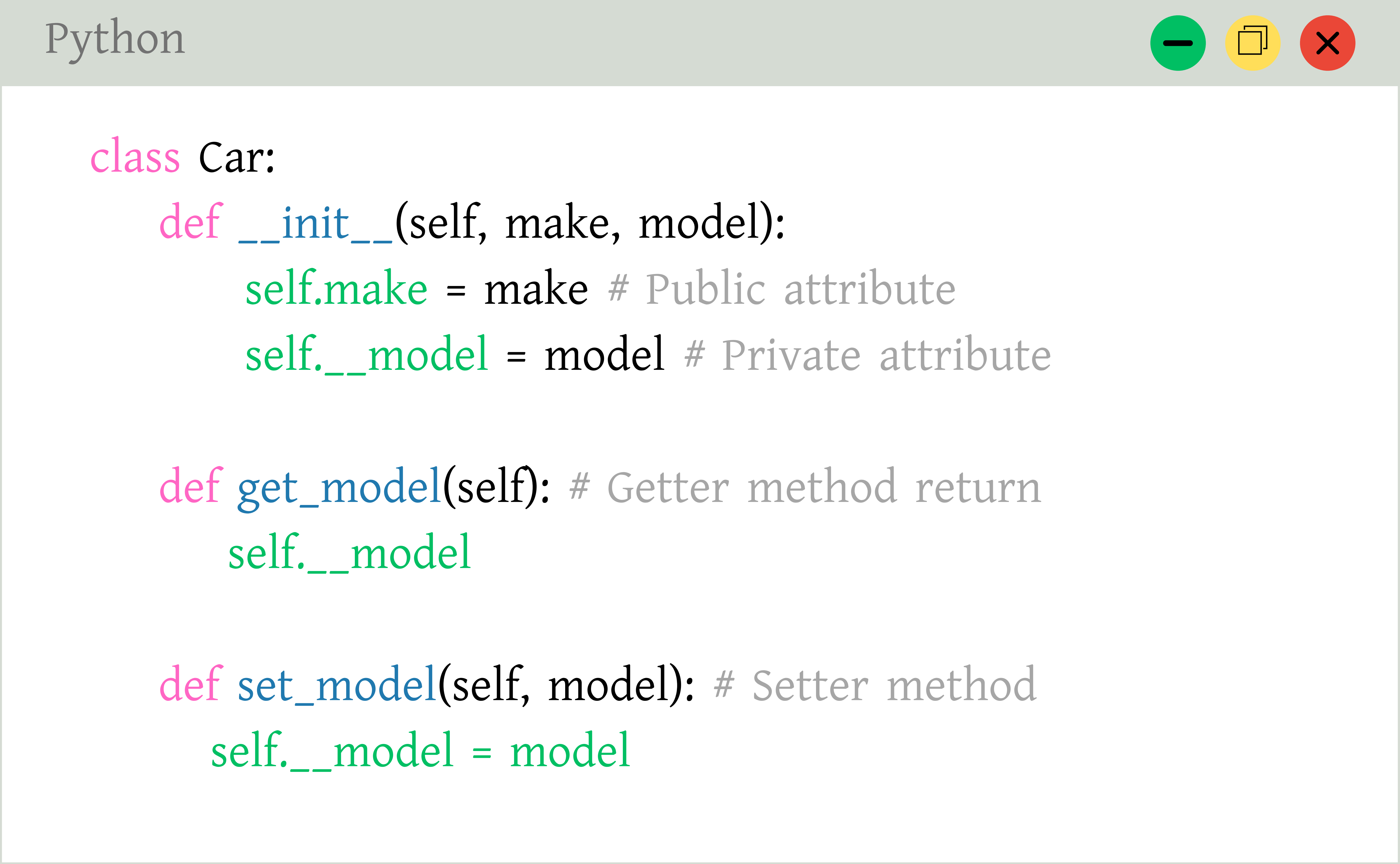
- **ইনহেরিট্যান্স (Inheritance):** এক ক্লাসের বৈশিষ্ট্য অন্য ক্লাস দ্বারা গ্রহণ করা।

- **পলিমরফিজম (Polymorphism):** একাধিক আকার বা ফর্মে একই নামের মেথড কাজ করতে সক্ষম।

- **এনক্যাপসুলেশন এর মূল উদ্দেশ্য:** এটি ডেটার নিরাপত্তা নিশ্চিত করে। যখন ডেটা এনক্যাপসুলেটেড হয়, তখন শুধুমাত্র নির্দিষ্ট মেথডের মাধ্যমে ডেটা অ্যাক্সেস বা পরিবর্তন করা সম্ভব। এতে ডেটা অবাঞ্ছিত পরিবর্তন থেকে সুরক্ষিত থাকে এবং সফটওয়্যারের স্থিতিশীলতা বজায় থাকে।

- **প্রাইভেট এবং পাবলিক অ্যাট্রিবিউট:** ক্লাসের ভিতরে ডেটাকে প্রাইভেট হিসেবে নির্ধারণ করা যায়, যাতে বাইরের কোড বা ক্লাসগুলি সরাসরি এই ডেটার সাথে কাজ করতে না পারে। এর পরিবর্তে, পাবলিক মেথড দিয়ে ডেটাকে অ্যাক্সেস করা হয়।

- **Getter এবং Setter মেথড:** getter মেথড ডেটাকে ফেরত দেয় এবং setter মেথড ডেটা সেট করে। এর মাধ্যমে ডেটা ম্যানিপুলেশনকে কন্ট্রোল করা হয়।



```
class Car:
    def __init__(self, make, model):
         self.make = make # Public attribute
         self.__model = model # Private attribute

    def get_model(self): # Getter method return
        self.__model

    def set_model(self, model): # Setter method
       self.__model = model
```