Fatec
Rio Claro

Linguagem de Programação II

Prof. Orlando Saraiva Júnior
orlando.nascimento@fatec.sp.gov.br

Pilares do Paradigma

Fatec
Rio Claro

O paradigma é sustentado por quatro pilares principais:

Abstração: focar nas características essenciais do objeto, ignorando detalhes irrelevantes.

Encapsulamento: proteger os dados, controlando como são acessados ou modificados.

Herança: permitir que uma classe herde atributos e métodos de outra, facilitando o reuso de código.

Polimorfismo: possibilitar que diferentes objetos respondam de formas distintas a uma mesma mensagem/método.

Encapsulamento

Encapsulamento

Fatec
Rio Claro

Funções são um excelente exemplo de encapsulamento porque, ao chamar uma função, geralmente não nos importamos com o funcionamento interno dela.

Uma função bem escrita contém uma série de etapas que compõem uma única tarefa maior, com a qual nos importamos.

O nome da função deve descrever a ação que seu código incorpora.

Encapsulamento

Fatec
Rio Claro

```
def calculateAverage(numbersList):
    total = 0.0
    for number in numbersList:
        total = total + number
    nElements = len(numbersList)
    average = total / nElements
    return average
```

Encapsulamento

Fatec
Rio Claro

Depois de testar uma função como essa e verificar que ela funciona, você não precisa mais se preocupar com os detalhes da implementação.

Você só precisa saber qual(is) argumento(s) enviar para a função e o que ela retorna.

Encapsulamento com Objetos

Fatec Rio Claro

Ao contrário das variáveis usadas em funções comuns, as variáveis de instância em objetos persistem em diferentes chamadas de métodos.

Quando você está trabalhando dentro de uma classe (escrevendo o código dos métodos em uma classe), precisa se preocupar com a forma como os diferentes métodos da classe compartilham as variáveis de instância.

Encapsulamento com Objetos

Fatec
Rio Claro

Você considera a eficiência dos seus algoritmos. Você pensa em como a interface deve ser: quais métodos você deve fornecer, quais são os parâmetros para cada um e quais devem ser usados como valores padrão.

Em resumo, você se preocupa com o design e a implementação dos métodos.

Encapsulamento com Objetos

Fatec
Rio Claro

De fora, como programador cliente, você precisa conhecer a interface da classe. Você se preocupa com o que os métodos da classe fazem, quais argumentos devem ser passados e quais dados são retornados por cada método.

Uma classe, portanto, fornece encapsulamento por meio de:

Ocultar todos os detalhes da implementação em seus métodos e variáveis de instância

Fornecer todas as funcionalidades que um cliente precisa de um objeto por meio de sua interface (os métodos definidos na classe)

Objetos possuem seus dados

Fatec
Rio Claro

Na programação orientada a objetos, dizemos que os dados dentro de um objeto são propriedade do objeto. Programadores de POO geralmente concordam que, como um bom princípio de design, o código cliente deve se preocupar apenas com a interface de uma classe e não com a implementação dos métodos.

Objetos possuem seus dados

Fatec
Rio Claro

```
class Person():

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
```

Objetos possuem seus dados

Fatec
Rio Claro

Os valores das variáveis de instância

self.name e self.salary são definidos sempre que instanciamos novos objetos Person, assim:

```
oPerson1 = Person('Joe Schmoe', 90000)
oPerson2 = Person('Jane Smith', 99000)
```

Interpretações de Encapsulamento

Fatec Rio Claro

É aqui que as coisas ficam um pouco controversas. Diferentes programadores têm visões diferentes sobre a acessibilidade de uma variável de instância.

Python oferece uma interpretação flexível do encapsulamento, permitindo acesso direto a variáveis de instância usando uma sintaxe de ponto simples.

O código cliente pode acessar legalmente uma variável de instância de um objeto

<objeto>.<NomeDaVariávelDeInstancia>.

Interpretações de Encapsulamento

Fatec
Rio Claro

No entanto, uma interpretação estrita do encapsulamento afirma que o software cliente nunca deve ser capaz de recuperar ou alterar o valor de uma variável de instância diretamente.

Em vez disso, a única maneira de um cliente recuperar ou alterar um valor contido em um objeto é usar um método fornecido pela classe para esse propósito.

Interpretação Estrita com Getters e Setters

Fatec Rio Claro

A abordagem estrita para encapsulamento afirma que o código cliente nunca acessa uma variável de instância diretamente.

Se uma classe deseja permitir que o software cliente acesse as informações contidas em um objeto, a abordagem padrão é incluir um método getter e um método setter na classe.

Interpretação Estrita com Getters e Setters

Fatec Rio Claro

getter Um método que recupera dados de um objeto instanciado de uma classe.

setter Um método que atribui dados a um objeto instanciado de uma classe.

Os métodos *getter* e *setter* são projetados para permitir que os desenvolvedores de software cliente obtenham e definam dados em um objeto, sem a necessidade de conhecimento explícito da implementação de uma classe — especificamente, sem precisar saber ou usar o nome de qualquer variável de instância.

Tornando Variáveis de Instância Mais Privadas

Fatec Rio Claro

Em Python, todas as variáveis de instância são públicas (ou seja, podem ser acessadas por código externo à classe).

Algumas linguagens de POO permitem marcar explicitamente certas variáveis de instância como públicas ou privadas, mas Python não possui essas palavras-chave.

No entanto, existem duas maneiras pelas quais os programadores que desenvolvem classes em Python podem indicar que suas variáveis de instância e métodos devem ser privados.

Implicitamente Privada

Fatec
Rio Claro

Para marcar uma variável de instância como aquela que nunca deve ser acessada externamente, por convenção, você inicia o nome da sua variável de instância com um sublinhado à esquerda:

self._name

self._socialSecurityNumber

self._dontTouchThis

Variáveis de instância com nomes como esses devem representar dados privados, e o software cliente nunca deve tentar acessá-las diretamente.

O código ainda pode funcionar se as variáveis de instância forem acessadas, mas isso não é garantido.

Privado Mais Explicitamente

Fatec
Rio Claro

O Python permite um nível mais explícito de privatização. Para impedir que o software cliente acesse diretamente seus dados, crie um nome de variável de instância que comece com dois sublinhados.

O Python oferece essa funcionalidade realizando a manipulação de nomes. Nos bastidores, o Python altera qualquer nome que comece com dois sublinhados.

Prática

Fatec
Rio Claro

PersonGettersSettersAndDirectAccess

Analisar os códigos deste diretório

PrivatePersonWIthNameMangling

Analisar os códigos deste diretório

Decoradores e @property

Fatec
Rio Claro

Em um nível mais alto, um decorador é um método que recebe outro método como argumento e estende a maneira como o método original funciona. (Decoradores também podem ser funções que decoram funções ou métodos, mas vou me concentrar em métodos.)

Decoradores são um tópico avançado. No entanto, há um conjunto de decoradores internos que fornecem um meio-termo entre o acesso direto e o uso de getters e setters em uma classe.

Decoradores e @property

Fatec
Rio Claro

Em um nível mais alto, um decorador é um método que recebe outro método como argumento e estende a maneira como o método original funciona. (Decoradores também podem ser funções que decoram funções ou métodos, mas vou me concentrar em métodos.)

Decoradores são um tópico avançado. No entanto, há um conjunto de decoradores internos que fornecem um meio-termo entre o acesso direto e o uso de getters e setters em uma classe.

Decoradores e @property

Fatec
Rio Claro

Um decorador é escrito como uma linha que começa com o símbolo @ seguido por um nome de decorador e é colocado diretamente antes da instrução def de um método.

Ao escrever uma classe para usar decoradores de propriedade (@property), o desenvolvedor escreve um método *getter* e um método *setter* e adiciona um decorador interno distinto a cada um.

Prática

Fatec
Rio Claro

PropertyDecorator

Analisar os códigos deste diretório

Abstração e Encapsulamento

Fatec
Rio Claro

Enquanto o encapsulamento se refere à implementação, a abstração se refere à visão que o cliente tem de uma classe.

A abstração é extremamente comum em produtos de consumo. Muitas pessoas usam TVs, fornos de micro-ondas, carros e assim por diante todos os dias. Temos a interface de usuário que esses produtos nos oferecem.

Por meio de seus controles, eles fornecem uma abstração de sua funcionalidade. Você pressiona o pedal do acelerador em um carro para fazê-lo andar. Em um micro-ondas, você define um tempo e pressiona Iniciar para aquecer um alimento. Mas poucos de nós realmente sabemos como esses produtos funcionam internamente.

Prática

Fatec
Rio Claro

Stack

Analisar os códigos deste diretório

ValidatingData_ClubExample

Analisar os códigos deste diretório

Herança

Herança

Fatec
Rio Claro

Herança em POO é a capacidade de criar uma classe que se baseia (estende) em uma classe existente. Ao criar programas grandes, você frequentemente usará classes que fornecem recursos gerais muito úteis.

Caso deseje construir uma classe semelhante a uma classe já existente, mas que faça algumas coisas de forma.

Com herança, falamos sobre o relacionamento entre duas classes, normalmente chamadas de classe ***base*** e ***subclasse***.

Herança

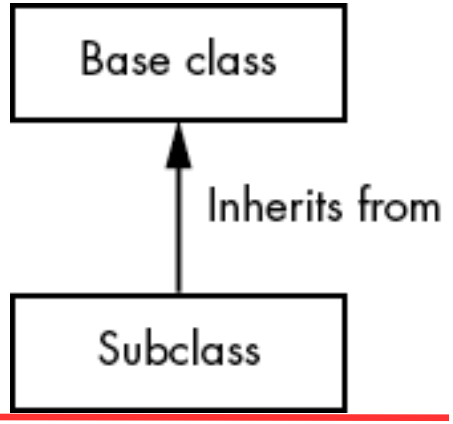
Fatec
Rio Claro

Classe base: A classe da qual herdamos; serve como ponto de partida para a subclasse.

Subclasse: A classe que herda; ela aprimora a classe base.

Embora esses sejam os termos mais comuns usados para descrever as duas classes em Python, você também pode ouvi-las sendo mencionadas de outras maneiras, como:

- Superclasse e subclasse
- Classe base e classe derivada
- Classe pai e classe filha



Implementando Herança

Fatec Rio Claro

A sintaxe da herança em Python é simples e elegante. A classe base não precisa saber que está sendo usada como classe base.

Apenas a subclasse precisa indicar que deseja herdar de uma classe base. Aqui está a sintaxe geral:

```
class <NomeDaClasseBase>():
# Métodos da ClasseBase

class <NomeDaSubClasse>(<NomeDaClasseBase>):
# Métodos da SubClasse
```

Prática

Fatec
Rio Claro

EmployeeManagerInheritance

Analisar os códigos deste diretório

Fatec
Rio Claro

Dúvidas

Prof. Orlando Saraiva Júnior
orlando.nascimento@fatec.sp.gov.br

Prática

Fatec
Rio Claro

Apresente um exemplo aplicável de Encapsulamento e herança.

Apresentar e debater com seus colegas se o exemplo pensado por você faz sentido.

Bibliografia

KALB, Irv. Object-Oriented Python. San Francisco: No Starch Press, 2022. 416 p.