PYTHON ACADEMY

O QUE É E COMO UTILIZAR @PROPERTY NO PYTHON

Nesse ebook você vai aprender o que é e como utilizar o @property no Python para deixar seu código mais elegante e fácil de entender

PYTHONACADEMY.COM.BR

Salve salve Pythonista!

O Python é uma linguagem de programação flexível e poderosa que oferece uma série de recursos avançados para os desenvolvedores.

Um desses recursos é o decorador `@property`, que permite transformar métodos de uma classe em propriedades, tornando o código mais elegante e fácil de entender.

Nesse artigo você vai aprender o que é e como utilizar o `@property` no Python.

Sem mais delongas, vamos nessa!

```
<div class="suggestions-header">
    <h2>Leia também</h2>
</div>
<div class="suggestions-body">
    <ul>
        <li>
            <a href="https://pythonacademy.com.br/blog/domine-decora-
tors-em-python" target="_blank">
                Domine Decoradores em Python
            </a>
        </li>
    </ul>
</div>
```

Ao criar classes em Python, é comum definir métodos que retornam ou modificam valores de atributos.

> *Se ainda não sabe sobre Classes em Python, leia nosso outro artigo sobre Classes no Python*

No entanto, esses métodos muitas vezes têm nomes diferentes dos atributos e são acessados como funções.

Para melhorar a legibilidade do código e fornecer uma sintaxe mais intuitiva, podemos usar o decorador `@property`.

Neste artigo, vamos explorar o que é o `@property` no Python e como utilizá-lo em nossas classes para criar propriedades que podem ser acessadas de uma forma mais intuitiva.

Propriedades em Python

Antes de mergulharmos no `@property`, é importante entender o conceito de propriedades em Python.

Uma propriedade é um atributo de classe que é calculado dinamicamente, em vez de ser armazenado em memória.

Quando um atributo é uma propriedade, ele é acessado e modificado como qualquer outro atributo, mas na verdade, por trás dos panos, um método customizado é chamado.

Ao utilizar propriedades, podemos ter controle sobre o acesso e modificação de atributos de uma classe, permitindo realizar validações, conversões de dados ou cálculos adicionais antes de retornar ou definir o valor.

O decorador `@property`

Em Python, o decorador `@property` é usado para transformar um método em uma propriedade de uma classe.

Ele permite que um método seja acessado como atributo, sem a necessidade de chamá-lo como uma função.

Vamos começar com um exemplo simples para ilustrar como o `@property` funciona.

Suponha que temos a classe `Retangulo` que representa um retângulo e possui os atributos `largura` e `altura`.

Para calcular a área do retângulo, poderíamos ter um método chamado `calcular_area`, como mostrado abaixo:

```
class Retangulo:
    def __init__(self, largura, altura):
        self.largura = largura
        self.altura = altura

    def calcular_area(self):
        return self.largura * self.altura
```

Neste caso, para calcular a área do retângulo, precisamos chamar o método `calcular_area()` explicitamente:

```
retangulo = Retangulo(5, 3)
area = retangulo.calcular_area()
print(area)
```

E a saída seria:

```
15
```

Agora, vamos utilizar o `@property` para transformar o método `calcular_area()` em uma propriedade da classe `Retangulo`:

```python
class Retangulo:
    def __init__(self, largura, altura):
        self.largura = largura
        self.altura = altura

    @property
    def area(self):
        return self.largura * self.altura
```

Note o uso do decorador `@property` antes do método `area()`. Agora, podemos acessar a área do retângulo como se fosse um atributo:

```python
retangulo = Retangulo(5, 3)
print(retangulo.area)  # Saída: 15
```

Agora, a chamada `retangulo.area` retorna o valor da área sem a necessidade de chamarmos explicitamente o método.

> 💡 *Estou construindo o **DevBook**, uma plataforma que usa IA para criar ebooks técnicos — com código formatado e exportação em PDF. Te convido a conhecer!*

Getters e Setters

Em muitos casos, queremos não apenas obter o valor calculado de uma propriedade, mas também definir seu valor.

Para isso, utilizamos os métodos `getter` e `setter`.

O método `getter` é responsável por retornar o valor da propriedade quando ela é acessada.

Utilizamos o decorador `@property` para definir o `getter`, como já vimos anteriormente.

O método `setter`, por sua vez, é usado para definir o valor da propriedade quando ela é modificada.

Para definir o `setter`, utilizamos o mesmo nome da propriedade seguido pelo decorador `@nomedapropriedade.setter`.

Vamos expandir nosso exemplo anterior para incluir um `setter` para modificar a largura do retângulo:

```
class Retangulo:
    def __init__(self, largura, altura):
        self._largura = largura
        self.altura = altura

    @property
    def largura(self):
        return self._largura

    @largura.setter
    def largura(self, nova_largura):
        if nova_largura > 0:
            self._largura = nova_largura
        else:
            raise ValueError("A largura deve ser maior que 0.")

    @property
    def area(self):
        return self.largura * self.altura
```

Neste exemplo, criamos um método `largura` para funcionar como o `getter` da propriedade `largura` e um método `largura.setter` para funcionar como o `setter`.

O método `setter` verifica se o novo valor da largura é maior que zero e, em caso positivo, atualiza o atributo `_largura`.

Caso contrário, lança uma exceção `ValueError` informando que a largura deve ser maior que zero.

Agora, podemos usar a propriedade `largura` para obter e modificar o valor da largura do retângulo:

```
retangulo = Retangulo(5, 3)
print(retangulo.largura)  # Saída: 5

retangulo.largura = 7
print(retangulo.largura)  # Saída: 7

retangulo.largura = -1  # Lança uma exceção ValueError
```

Podemos ver que, ao acessar a propriedade `largura`, o método `largura()` é chamado e retorna o valor atual da largura.

Da mesma forma, ao atribuir um novo valor à propriedade `largura`, o método `largura.setter` é executado e valida o novo valor antes de atualizar `_largura`.

Deletar propriedades

Além de obter e definir o valor de uma propriedade, também podemos excluí-la utilizando o método `deleter`.

Para definir o `deleter` de uma propriedade, utilizamos o decorador `@nomedapropriedade.deleter`.

Vamos adicionar um método `deleter` à classe `Retangulo` para excluir a propriedade `largura`:

```python
class Retangulo:
    def __init__(self, largura, altura):
        self._largura = largura
        self.altura = altura

    @property
    def largura(self):
        return self._largura

    @largura.setter
    def largura(self, nova_largura):
        if nova_largura > 0:
            self._largura = nova_largura
        else:
            raise ValueError("A largura deve ser maior que 0.")

    @largura.deleter
    def largura(self):
        del self._largura

    @property
    def area(self):
        return self.largura * self.altura
```

Agora podemos excluir a propriedade `largura` utilizando o comando `del`:

```python
retangulo = Retangulo(5, 3)
print(retangulo.largura)  # Saída: 5

del retangulo.largura

print(retangulo.largura)
```

Ao tentar acessar a propriedade na linha `print(retangulo.largura)` o seguinte erro será lançado:

```
AttributeError: 'Retangulo' object has no attribute '_largura'
```

Casos de uso do `@property`

O decorador `@property` é muito útil em situações em que precisamos controlar o acesso aos atributos de uma classe.

Aqui estão alguns exemplos de casos de uso comuns:

- Conversão de tipos: podemos usar `@property` para converter automaticamente tipos de dados. Por exemplo, podemos ter um atributo `data` que é armazenado como uma string e uma propriedade `data` que devolve o valor convertido em um objeto `datetime`.
- Verificação de validade: podemos adicionar validações em um `setter` para garantir que os atributos estão dentro dos limites aceitáveis. Por exemplo, podemos ter um atributo `idade` que precisa ser um número positivo e, caso contrário, levanta um erro.
- Uso de cache: podemos utilizar uma propriedade para fazer cache de um valor calculado, evitando recalcular a cada vez que a propriedade é acessada.
- Acesso a dados externos: podemos usar propriedades para acessar e atualizar dados em bancos de dados externos ou sistemas remotos. Dessa forma, podemos manter a interface do objeto consistente, independentemente de onde os dados são armazenados.

Conclusão

O decorador `@property` é uma ferramenta poderosa que nos permite transformar métodos em propriedades de uma classe, oferecendo um acesso mais intuitivo e controlado a essas propriedades.

Com a utilização do `@property`, podemos definir `getter`, `setter` e `deleter` para controlar o acesso e a modificação dos valores dos atributos.

Além de fornecer uma sintaxe mais elegante, as propriedades também nos permitem adicionar validações, conversões de dados e realizar cálculos adicionais antes de retornar ou definir o valor de um atributo.

Com esse conhecimento em mãos, você está pronto para utilizar o `@property` em suas classes Python e tornar seus códigos mais legíveis e eficientes.

Experimente utilizar propriedades em seus projetos e descubra como elas podem simplificar o acesso e a manipulação de atributos em suas classes.

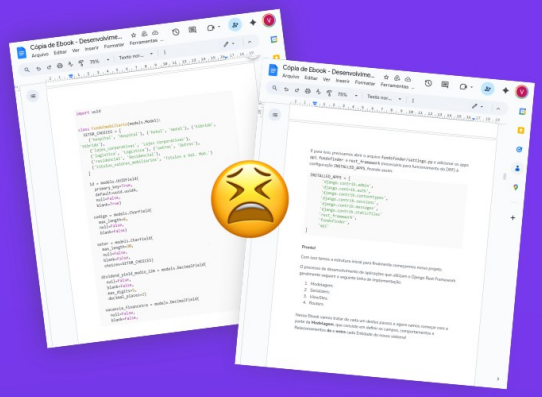
É isso por hoje! Nos vemos no próximo artigo 😉

Não se esqueça de conferir!

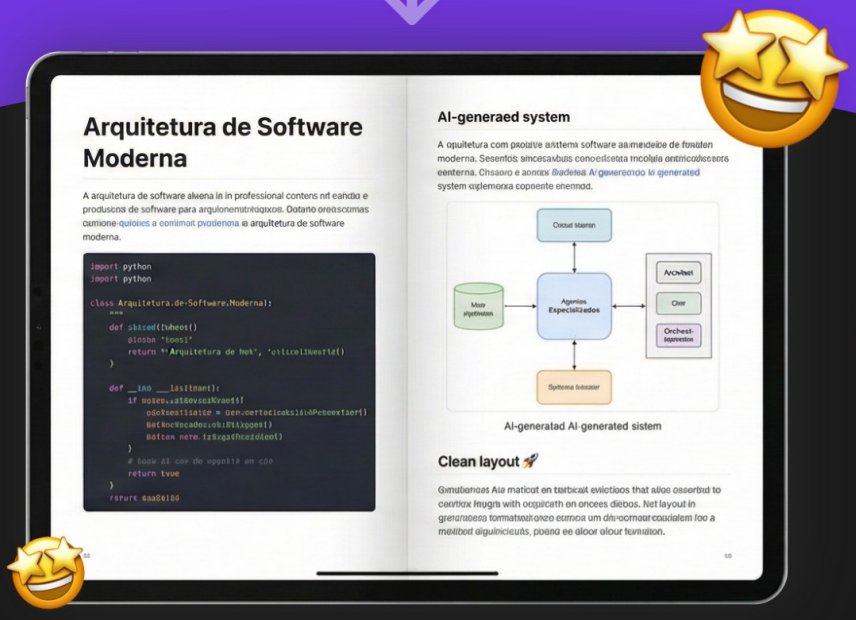
DevBook

Crie Ebooks técnicos em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código**!



Chega de formatar código no Google Docs



Syntax Highlight

Infográficos feitos por IA

Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado

Edite em Markdown em Tempo Real

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS