How Python inserts "self" into methods

An informal introduction to descriptors

Sebastiaan Zeeff

FOSDEM 2021 — Python Devroom

Introduction

- Sebastiaan Zeeff (34)

- Software Engineer
- Ordina Pythoneers
- New job (January 2021)

- Co-owner
- Online Community; 140,000 members
- Education, Events, Open Source
- Supporting the Python Ecosystem

The magic of self

```python
class Guitar:
    def __init__(self, name: str) -> None:
        self.name = name

    def play_note(self, note: str) -> None:
        print(f"My {self.name} plays the note {note!r}")

```

```
>>> warwick = Guitar(name="Warwick Streamer")
>>> warwick.name
'Warwick Streamer'
>>> warwick.play_note("C#")
My Warwick Streamer plays the note 'C#'
```

How *does* Python insert self?

```python
class Guitar:
    def __init__(self, name: str):
        self.name = name

    def play_note(self, note: str):
        print(f"My {self.name} plays a {note!r}")

print(Guitar.play_note)
<function Guitar.play_note at 0x0000025BC5D84F70>
```

- The function object itself is still a regular function object
- We've assigned a class attribute to that function object

```
class Guitar:
    def __init__(self, name: str):
        self.name = name

    def play_note(self, note: str):
        print(f"My {self.name} plays a {note!r}")

print(Guitar.play_note)
<function Guitar.play_note at 0x0000025BC5D84F70>

warwick = Guitar(name="Warwick Streamer")
print(warwick.play_note)
<bound method Guitar.play_note of Guitar(name="Warwick")>

warwick.play_note("C#")
```

- We've assigned a class attribute to that function object

Descriptors and Attribute Access

- Descriptor objects can customize attribute look-up, assignment, and deletion.

- This is done by implementing special ('dunder') methods:
  - The __get__ method customizes attribute look-up:
  - The __set__ method customizes attribute assignment:
  - The __delete__ method customizes attribute deletion:

```
warwick.play_note
```

```
warwick.play_note = 'value'
```

```
del warwick.play_note
```

- Functions are descriptors that implement __get__

```
class Guitar:
    def __init__(self, name: str):
        self.name = name

    def play_note(self, note: str):
        print(f"My {self.name} plays a {note!r}")
```

```
class Guitar:
    # def __init__(self, name: str): ...
    # def play_note(self, note: str): ...

    is_my_favourite = FavouriteDescriptor()

warwick = Guitar(name="Warwick Streamer")
print(warwick.is_my_favourite)  # Should print True
```

```
class FavouriteDescriptor:
    def __get__(self, instance, owner):
        if getattr(instance, "name", None) == "Warwick Streamer":
            return True

        return False
```

```
>>> warwick = Guitar(name="Warwick Streamer")
>>> print(warwick.is_my_favourite)
True
```

```
class Guitar:
    # def __init__(self, name: str): ...
    # def play_note(self, note: str): ...

    is_my_favourite = FavouriteDescriptor()

warwick = Guitar(name="Warwick Streamer")
print(warwick.is_my_favourite)  # Should return True
```

```
class FavouriteDescriptor:
    def __get__(self, instance, owner):
        if getattr(instance, "name", None) == "Warwick Streamer":
            return True

        return False
```

```
>>> fender = Guitar(name="Fender Jazz Bass")
>>> print(fender.is_my_favourite)
False
```

Functions also have a __get__ method

```
class Guitar:
    def __init__(self, name: str):
        self.name = name

    def play_note(self, note: str):
        print(f"My {self.name} plays a {note!r}")

warwick = Guitar(name="Warwick Streamer")
```

```
>>> Guitar.play_note.__get__(warwick, Guitar)
<bound method Guitar.play_note of Guitar(name="Warwick Streamer")>
```

Summary

- The descriptor protocol allows you customize how attributes work
- Functions implement the descriptor protocol to create bound methods
- You can do a lot of cool things with descriptors

Not covered, but important:

- Examples with __set__ and __delete__
- The specific difference between "data" and "non-data" descriptors
- How you can use __set_name__ in combination with descriptors

Thanks for listening!

How Python inserts "self" into methods

An informal introduction to descriptors

Sebastiaan Zeeff

FOSDEM 2021 — Python Devroom