

# The Magic of Self

## How Python inserts self into methods

**Sebastiaan Zeeff — Pyjamas Conf, 2021**

<https://sebastiaanzeeff.nl>

<https://www.linkedin.com/in/sebastiaanzeeff/>



# Who am I?

- **Sebastiaan Zeeff (35)**
- **Software Engineer & Codesmith @ Ordina Pythoners**
- **Python Discord**
- **Volunteer for EuroPython**



***"From day one, there was deep magic hiding in some places, designed to quietly help users."***

**— Guido van Rossum, on Twitter.**

# *Part one — What is the magic of self?*



```
1 class Guitar:
2     def __init__(self, name: str) -> None:
3         self.name = name
4
5     def play_note(self, note: str) -> None:
6         print(f"My {self.name} plays the note {note!r}")
7
```

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

```
1 class Guitar:
2     def __init__(self, name: str) -> None:
3         self.name = name
4
5     def play_note(self, note: str) -> None:
6         print(f"My {self.name} plays the note {note!r}")
7
```

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

# *Part two — Methods in Python*



```
1 class Guitar:
2     def __init__(self, name: str) -> None:
3         self.name = name
4
5     def play_note(self, note: str) -> None:
6         print(f"My {self.name} plays the note {note!r}")
7
```



- The function object is still (mostly) a **regular function object**.
- We've bound a **class attribute** to that function object.



# *Part three — Accessing methods*



```
1 class Guitar:
2     def __init__(self, name: str) -> None:
3         self.name = name
4
5     def play_note(self, note: str) -> None:
6         print(f"My {self.name} plays the note {note!r}")
7
```

```
>>> Guitar.play_note
<function Guitar.play_note at 0x00000v=dQw4w9WgXcQ>

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

>>>
```

# *Part four — Descriptors*



# The descriptor protocol

- Descriptors can customize attribute look-up, assignment, and deletion.
- They do that by implementing special ('dunder') methods:
  - The `__get__` method customizes look-up: 

```
warwick.play_note
```
  - The `__set__` method customizes assignment: 

```
warwick.play_note = "value"
```
  - The `__delete__` method customizes deletion: 

```
del warwick.play_note
```
- Functions are descriptors that implement a `__get__` method.

# *Part five — Implementing a descriptor*



```
1 class Guitar:
2     def __init__(self, name: str) -> None: ...
5     def play_note(self, note: str) -> None: ...
7
8     is_my_favourite = FavouriteDescriptor()
9
10 warwick = Guitar(name="Warwick Streamer")
11 print(warwick.is_my_favourite) # prints True
```

```
1 class Guitar:
2     def __init__(self, name: str) -> None: ...
5     def play_note(self, note: str) -> None: ...
7
8     is_my_favourite = FavouriteDescriptor()
9
10 fender = Guitar(name="Fender Jazz Bass Deluxe")
11 print(fender.is_my_favourite) # prints False
```

```
1 class Guitar:
2     def __init__(self, name: str) -> None: ...
5     def play_note(self, note: str) -> None: ...
7
8     is_my_favourite = FavouriteDescriptor()
9
10    fender = Guitar(name="Fender Jazz Bass Deluxe")
11    print(fender.is_my_favourite) # prints False
```

```
1 class FavouriteDescriptor:
2     def __get__(self, instance, owner) -> bool | FavouriteDescriptor:
3         if instance is None:
4             return self
5
6         return instance.name == "Warwick Streamer"
```



```
1 class Guitar:
2     def __init__(self, name: str) -> None: ...
5     def play_note(self, note: str) -> None: ...
7
8     is_my_favourite = FavouriteDescriptor()
9
10 fender = Guitar(name="Fender Jazz Bass Deluxe")
11 print(fender.is_my_favourite) # prints False
```

```
1 class FavouriteDescriptor:
2     def __get__(self, instance, owner) -> bool | FavouriteDescriptor:
3         if instance is None:
4             return self
5
6         return instance.name == "Warwick Streamer"
```

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

```
1 class Guitar:
2     def __init__(self, name: str) -> None: ...
5     def play_note(self, note: str) -> None: ...
7
8     is_my_favourite = FavouriteDescriptor()
9
10 fender = Guitar(name="Fender Jazz Bass Deluxe")
11 print(fender.is_my_favourite) # prints False
```

```
1 class FavouriteDescriptor:
2     def __get__(self, instance, owner) -> bool | FavouriteDescriptor:
3         if instance is None:
4             return self
5
6         return instance.name == "Warwick Streamer"
```

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

```
1 class Guitar:
2     def __init__(self, name: str) -> None: ...
5     def play_note(self, note: str) -> None: ...
7
8     is_my_favourite = FavouriteDescriptor()
9
10    fender = Guitar(name="Fender Jazz Bass Deluxe")
11    print(fender.is_my_favourite) # prints False
```

```
1 class FavouriteDescriptor:
2     def __get__(self, instance, owner) -> bool | FavouriteDescriptor:
3         if instance is None:
4             return self
5
6         return instance.name == "Warwick Streamer"
```

```
>>> Guitar.is_my_favourite
<FavouriteDescriptor object at 0x0000A1BFD314433>
```

# *Part six — The `__get__` method of a function*



```
1 # Pseudo-implementation of function's __get__
2 # Remember: self refers to the function itself!
3 class function:
4     def __get__(self, instance, owner):
5         if instance is None:
6             return self
7
8         return PyMethod(self, instance)
```

```
>>> Guitar.play_note
<function Guitar.play_note at 0x00000v=dQw4w9WgXcQ>

>>> Guitar.play_note.__get__(None, Guitar)
<function Guitar.play_note at 0x00000v=dQw4w9WgXcQ>

>>>
```

```
1 # Pseudo-implementation of function's __get__
2 # Remember: self refers to the function itself!
3 class function:
4     def __get__(self, instance, owner):
5         if instance is None:
6             return self
7
8     return PyMethod(self, instance)
```

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

# *Part seven — Other built-in descriptors*



# Other built-in descriptors

- The `classmethod` descriptor binds the class, not the instance, to the function.

```
1 class Guitar:
2     @classmethod
3     def print_class_name(cls) -> None:
4         print(f"The name of the class: {cls.__name__!r}")
5
6
7
8
```



# Other built-in descriptors

- The `classmethod` descriptor binds the class, not the instance, to the function.
- The `staticmethod` descriptor returns the function as-is, not binding anything.

```
1 class Guitar:
2     @staticmethod
3     def my_static_method() -> None:
4         print("I know no bounds!")
5
6
7
8
```

# Other built-in descriptors

- The `classmethod` descriptor binds the class, not the instance, to the function.
- The `staticmethod` descriptor returns the function as-is, not binding anything.
- The `property` descriptor allows you to easily create getters/setters/deleters.

```
1 class Guitar:
2     @property
3     def price(self) -> float:
4         return self._price
5
6     @price.setter
7     def price(self, new_price: float) -> None:
8         self._price = new_price
```

# *Summary*



# Summary

- Descriptors allow you to customize how attributes work.
- Python's functions implement the descriptor protocol to create bound methods.

# Summary

- Descriptors allow you to customize how attributes work.
- Python's functions implement the descriptor protocol to create bound methods.

## Not covered, but also important:

- The `__set__` and `__delete__` descriptor methods.
- The difference between data and non-data descriptors.
- The `__set_name__` method, often used in descriptors.

# *Epilogue: On magic*



*“I'm writing a book on magic”, I explain, and I'm asked, “Real magic?” By real magic people mean miracles, thaumaturgical acts, and supernatural powers. “No”, I answer: “Conjuring tricks, not real magic”. Real magic, in other words, refers to the magic that is not real, while the magic that is real, that can actually be done, is not real magic.”*

**— Lee Siegal in Net of Magic: Wonders and deceptions in India**



# *The Magic of Self*



## Sebastiaan Zeeff

- **Twitter:** @SebastiaanZeeff
- **LinkedIn:** <https://www.linkedin.com/in/sebastiaanzeeff/>
- **Website + slides:** <https://sebastiaanzeeff.nl>
- **Discord:** Sebastiaan#0008 @ <https://discord.com/invite/python>

*The duckies are licensed as CC BY-NC-SA 4.0 by Python Discord*