Introduction to Decorators

Power Up Your Python Code

Geir Arne Hjelle – geirarne@gmail.com

Pycon US April 20, 2023

Exercise 1

Write a decorator that prints `BEFORE` before calling the decorated function and `AFTER` afterward.

```
>>> @before_and_after
... def hi(name):
...     print(f"Hi, {name}!")
...
>>> hi("PyCon")
BEFORE
Hi, PyCon!
AFTER
```

You may submit your solution at `forms.gle/VhdS6cL5GddcbytN6` or click the link on GitHub: **gahjelle** > **decorators_tutorial** > **2023**

Exercise 2

Write a decorator that runs the decorated function twice and returns a 2-tuple with both return values.

```
>>> import random
>>> @do_twice
... def roll_dice():
...     return random.randint(1, 6)
...
>>> roll_dice()
(3, 1)
```

Exercise 3

Write a decorator that stores references to decorated functions in a dictionary.

```
>>> FUNCTIONS = {}
>>> @register
... def roll_dice():
...     return random.randint(1, 6)
...
>>> FUNCTIONS
{'roll_dice': <function __main__.roll_dice()>}
>>> FUNCTIONS["roll_dice"]()
2
```

Exercise 4

Write a decorator that repeatedly calls the decorated function as long as it raises an exception.

```
>>> @retry
... def only_roll_highs():
...     number = random.randint(1, 6)
...     if number < 5:
...         raise ValueError(number)
...     return number
...
>>> only_roll_highs()
# 5 or 6
```

Exercise 5

Rewrite `@retry` so that it only retries on specific, user-defined exceptions.

```
>>> @retry(ValueError)
... def calculation():
...     number = random.randint(-5, 5)
...     if abs(1 / number) > 0.2:
...         raise ValueError(number)
...     return number
...
>>> calculation()
# -5, 5, or ZeroDivisionError
```

Exercise 6

Adapt `@retry` so that it only tries a maximum of `max_retries` times.

```
>>> @retry(max_retries=3)
... def only_roll_highs():
...     number = random.randint(1, 6)
...     if number < 5:
...         raise ValueError(number)
...     return number
...
>>> only_roll_highs()
# 5, 6, or ValueError
```

Exercise 6

Adapt `@retry` so that it only tries a maximum of `max_retries` times.

```
>>> @retry(max_retries=3)
... def only_roll_highs():
...     number = random.randint(1, 6)
...     if number < 5:
...         raise ValueError(number)
...     return number
...
>>> only_roll_highs()
# 5, 6, or ValueError
```

Challenge: Can you make `@retry` count all retries across different function calls?

Exercise 7

Write a class-based `@Retry` decorator that keeps track of the number of retries across all function calls.

```
>>> @Retry
... def only_roll_highs():
...     # Same as before
...
>>> only_roll_highs()
Retry attempt 1
Retry attempt 2
6
>>> only_roll_highs()
Retry attempt 3
5
```

Exercise 8

Write a decorator that uses `.__annotations__` to enforce the type of arguments. For simplicity, only enforce keyword arguments:

```
>>> @enforce
... def adder(number_1: int, number_2: int):
...     return number_1 + number_2
...
>>> adder(number_1=23, number_2=10)
33
>>> adder(number_1=3.14, number_2=2.71)
TypeError: 'number_1' should be <class 'int'>
```

Other Resources

- Tutorial page:
  - github.com/gahjelle/decorators_tutorial
  - Material and video from the 2020 and 2021 courses
  - Code from 2023 will be added after the course
- Real Python article:
  - realpython.com/primer-on-python-decorators
- Real Python video course (paywall):
  - realpython.com/courses/python-decorators-101
- PEP 318: Decorators for Functions and Methods
  - www.python.org/dev/peps/pep-0318
- Awesome Python Decorator
  - github.com/lord63/awesome-python-decorator