Tutorium 14 - 02.02.2024

Decorator, Testing

Decorator

- **Design-Pattern**, oft auch **Wrapper** genannt
- Verpackt ein Objekt um **zusätzliche Funktionalität** zu bieten
  - Funktionen sind auch Objekte
  - eine Klasse ist ein Objekt
- Oft einfach **syntax sugar**

Beispiel - execute_two_times

```python
def execute_two_times(fn: Callable[..., Any]) -> Callable[..., Any]:
    def wrapper(*args, **kwargs)
        fn(*args, **kwargs)
        fn(*args, **kwargs)
        return wrapper
    return wrapper

@execute_two_times()
def print_two_times(msg: str):
    print(msg)

print_two_times("hello")    # hello
                            # hello
```

Beispiel - execute_by

```python
def execute_by(n: int):
    def wrapper(fn):
        def wrapped_fn(*args, **kwargs):
            for _ in range(0, n):
                fn(*args, **kwargs)
            return wrapped_fn
        return wrapped_fn
    return wrapper

@execute_by(10)
def print_ten_times(msg: str):
    print(msg)

print_ten_times("hello")    # hello
                            # hello
                            # ... (10 mal)
```

Beispiel - CommandExecutor

```
class CommandExecutor[R]:

    def __init__(self):
        self.__commands: dict[str, Callable[..., R]] = {}
```

Beispiel - run

```
    def run(self, name: str, *args, **kwargs) -> list[R]:
        results : list[R] = []
        for command_name, command in self.__commands.items():
            if command_name == name:
                results += [command(*args, **kwargs)]
        return results
```

Beispiel - register

```
    def register(self, cmd: Callable[..., R]) -> Callable[..., R]:
        self.__commands[cmd.__name__] = cmd
        return cmd
```

Beispiel - CommandExecutor

```python
class CommandExecutor[R]:
    def __init__(self):
        self.__commands: dict[str, Callable[..., R]] = {}

    def run(self, name: str, *args, **kwargs) -> list[R]:
        results : list[R] = []
        for command_name, command in self.__commands.items():
            if command_name == name:
                results += [command(*args, **kwargs)]
        return results

    def register(self, cmd: Callable[..., R]) -> Callable[..., R]:
        self.__commands[cmd.__name__] = cmd
        return cmd
```

Beispiel - How to use

```python
app = CommandExecutor[str]()

@app.register
def hello_world() -> str:
    return 'hello_world'

@app.register
def divide(a: int, b: int) -> str:
    if b == 0:
        return "tried to divide by zero"
    return str(a / b)

print(app.run('hello_world'))
print(app.run('divide', 5, 0))
print(app.run('divide', 10, 2))
```

Decorator in der Klausur

- Waren noch nie Bestandteil der Klausur
- Mut zur Lücke
- Kann euch natürlich nichts versprechen

Testing mit `pytest`