How to mock Peter’s backend

# Was ist ein Mock ?

**Ein Mock-Objekt ist in der Softwareentwicklung ein Programmteil, der zur Durchführung von Modultests als Platzhalter für echte Objekte verwendet wird. Solche Hilfsmittel werden umgangssprachlich auch *Mocks* genannt.**

Konkret auf die Ascona Umgebung bedeutet dies, dass die Backend Endpunkte **SolarLog**, **Relais** und **Temperatur Sensoren** durch Mock Objekte abgebildet werden. Diese Mocks liefern Zufallswerte in einem für den jeweiligen Endpunkt gültigen Wertebereich zurück.

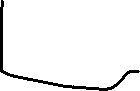
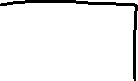
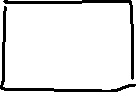
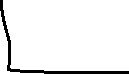
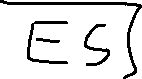
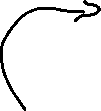
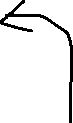
**Verbrauch = round(random.uniform(1000.00, 4500.00), 2)**

**Erzeugung = round(random.uniform(500.00, 15000.00), 2)**

**self.Bezug = round(random.uniform(1000.00, 2000.00), 2)**

RANDOM Werte wenn der Mock aktiviert ist

Echte SolarLog Werte über JSON-Schnittstelle



Vorteil:

* Wenn die physischen Endpunkte aus technischen Gründen nicht zur Verfügung stehen, kann man die Entladesteuerung mit den Mocks weiterentwickeln oder testen.
* Da die physischen Endpunkte (Schnittstellen) sich in der Regel nicht verändern, sind diese relativ stabil und nicht fehlerbehaftet.
* Die Entwicklung kann z.B. auch auf einem Windows PC durchgeführt werden, es ist kein Raspi notwendig.

# Lösungsansatz

Prinzipiell könnte man die Mock Funktionalität in der Global Variables Komponente GVS.PY implementieren und dann für den jeweiligen physikalischen Endpunkt über Flags aktivieren bzw. deaktivieren.

Ein eleganterer Ansatz ist das Mocking an der jeweiligen Endpunkt Funktion (Relais, Sensor, SolarLog) vorzunehmen und bei Bedarf ein- oder auszuschalten.

Dies lässt sich relativ elegant über Python Dekoratoren erreichen:

***In Python sind Dekoratoren eine spezielle Änderung der Python-Syntax, die es uns ermöglicht, Funktionen und Methoden (und möglicherweise Klassen in einer zukünftigen Version) bequemer zu ändern.***

***Ein Dekorator ist eine Funktion, die eine andere Funktion als Argument nimmt und das Verhalten der letzteren Funktion erweitert, ohne sie explizit zu ändern2.***

***(ChatGPT 😉)***

**def my\_decorator(decoFunc):**

**def wrapper():**

**print("Vor Aufruf der dekorierten Funktion")**

**decoFunc()**

**print("Nach Aufruf der dekorierten Funktion ")**

**return wrapper**

**@my\_decorator**

**def say\_hello():**

**print("Hello!")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**say\_hello()**

Ergebnis:

**Vor Aufruf der dekorierten Funktion**

**Hello!**

**Vor Aufruf der dekorierten Funktion**

Auf diese Art und Weise lassen sich ziemlich einfach in der Wrapper Funktion zusätzliche Funktionen für z.B. extensives Logging oder Ausführungszeitermittlung einschalten und dann später in Produktion durch Auskommentieren des Dekorators wieder ausschalten.

# SolarLog Dekorator

Hier die Implementierung des SolarLog Dekorators:

**def ReadSolarlogStub(fn): # wird als Decorator benutzt um künstliche Stromwerte**

**@wraps(fn) # zum Debuggen ohne Solaranlage zu simulieren**

**def wrapper(self):**

**logSolarLog = Logger().GetLogger("SolarLog") # Console/File Ausgaben**

**logSolarLog.log(logging.WARNING, "logSolarLog: Decorator is active!!!")**

**self.Verbrauch = round(random.uniform(1000.00, 4500.00), 2)**

**self.Erzeugung = round(random.uniform(500.00, 15000.00), 2)**

**self.Bezug = round(random.uniform(1000.00, 2000.00), 2)**

**self.SetLastUpdate()**

**return True**

**return wrapper**

**@dataclass**

**class SolarLog:**

**ipAddress: str**

**Verbrauch: float**

**Erzeugung: float**

**Bezug: float**

**lastUpdate: str**

**lastError: str**

**Code: int**

**def \_\_init\_\_(self, ipAddr: str) -> None:**

**self.ipAddress = ipAddr**

**@ReadSolarlogStub**

**def Read(self)->bool: # Daten vom Solar-Log einlesen und verarbeiten**

**try: # Update der JSON-Schnittstelle erfolgt alle 15 sec**

**self.lastUpdate = time.strftime("%Y.%m.%d %H:%M:%S")**

**self.Verbrauch = 0.0**

**self.Erzeugung = 0.0**

**self.Bezug = 0.0**

**req = urllib.request.Request('http://'+ self.ipAddress +'/getjp')**

**req.add\_header('Content-Type', 'application/json; charset=utf-8')**

**…. # Rest weggelassen**

Die Funktion ReadSolarLogStub generiert zufällige Werte für PV Anlage und gibt vorsichtshalber ein Warning aus, das der Dekorator aktiviert ist, um sicherzustellen, das in Produktion der Dekorator auskommentiert wird.

Anm.:

Das Statement **@wraps(fn)** das von der dekorierten Funktion der Name und der Kommentar-String ordnungsgemäß zur Verfügung steht.