

Warum bekommen Software-Design-Patterns und Prinzipien wenig Beachtung in Youtube Tutorials?

Die meisten Youtube-Tutorials im Bereich Python, richten sich an blutige Anfänger und erklären, überwiegend, Datenstrukturen, einfache Operationen und Funktionen.

Eventuell wird auch noch über Klassen gesprochen. Dennoch findet man auf Youtube auch Tutorials, die weitaus tiefer in die Materie reichen.

Hier möchte ich zwei, zufällig gewählte, Beispiele anführen, welche, zusammen immerhin über 9 Millionen Aufrufe haben.

The image shows a code editor window titled 'student.py' with a Python class definition. The class 'Student' has an '__init__' method that checks if 'name' is provided, raises a 'ValueError' if not, and then assigns 'name' and 'house' to 'self'. It also has a '__str__' method that returns a string representation and a 'house' property. Below the code is a terminal window showing the execution of 'python student.py', which outputs 'Name: Harry', 'House: Gryffindor', and 'Harry from Number Four, Privet Drive'. A video inset in the bottom right corner shows a man speaking.

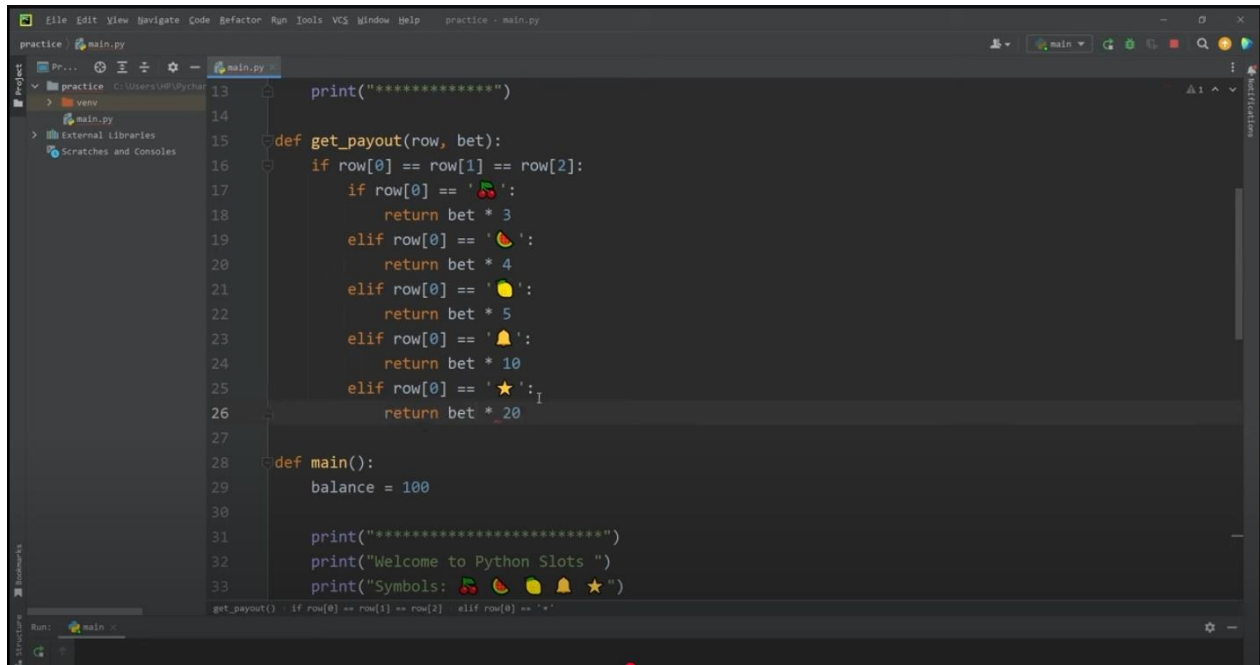
```
1 class Student:
2     def __init__(self, name, house):
3         if not name:
4             raise ValueError("Missing name")
5         self.name = name
6         self.house = house
7
8     def __str__(self):
9         return f"{self.name} from {self.house}"
10
11 @property
12 def house(self):
13     return self.house
14
```

```
TERMINAL
$ python student.py
Name: Harry
House: Gryffindor
Harry from Number Four, Privet Drive
```

Zum Beispiel der der *Harvard CS50's Introduction to Programming with Python - Full University Course* auf dem Kanal *freeCodeCamp.org*. In diesem Kurs, mit ein Dauer von knapp 16 Stunden, wird in Stunde 13 bei das Thema Klassenvererbung behandelt. Hier sieht man wie die Klasse *Student* mit einem if-statement initialisiert wird. Hier hätte man sich überlegen können, ob man die Ausnahme, welche hier für die Initialisierung verwendet wurde, nicht später einbaut und dabei gleichzeitig Design Patterns aufgreift und eine Builder Klasse erstellt.

Ein weiteres Beispiel ist das Video **Python Full Course for free (2024)** auf dem Kanal **Bro Code**. Hier wird in der fünften Stunden eine Slot Machine

programmiert. Dabei wird die Funktionen mit unnötig viel if-else Logik gefüttert und man sich durchaus für jeden if-Fall eine Funktionen hätte programmieren können.

A screenshot of a Python IDE (likely VS Code) showing a file named 'main.py'. The code implements a slot machine game. It starts with a 'print' statement for a separator. Then, a 'def get_payout(row, bet):' function is defined. Inside this function, there is a long chain of 'if' and 'elif' statements checking if the first three elements of the 'row' list are equal. If they are, it returns a payout based on the symbol: '🍒' (3x), '🍋' (4x), '🍌' (5x), '🍷' (10x), and '★' (20x). If none match, it returns 'bet * 20'. Below this, a 'def main():' function is defined, which sets 'balance = 100', prints a welcome message, and prints the symbols. At the bottom, there is a 'Run' button and a console output area showing the program's execution. The code is written in a dark-themed editor with syntax highlighting.

Fazit:

Das Ziel ist, Anfänger schnell zu einem Ergebnis zu bringen und eventuell Begeisterung für das Programmieren zu entfachen, nicht sauberen, wartbaren Code zu lehren.

Begriffe wie OCP (Open-Closed-Principle) oder DIP (Dependency Inversion Principle) klingen für Anfänger abstrakt und können eher abschreckend als begeistern.

Viele Tutorials zeigen, wie man ein Problem löst, aber nicht, wie man es richtig löst.

Oft wird einfach Code „hingeworfen“, der zwar funktioniert, aber nicht skalierbar oder gut strukturiert ist.

Konzepte wie Factory Pattern, Strategy Pattern oder Dependency Injection sind nicht so leicht verständlich wie eine einfache for-Schleife.

Viele Entwickler lernen Patterns erst, wenn sie in komplexeren Software-Projekten arbeiten.