



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Kapitel 03

CODE COVERAGE: C0 & C1
BERECHNEN

Lernziele

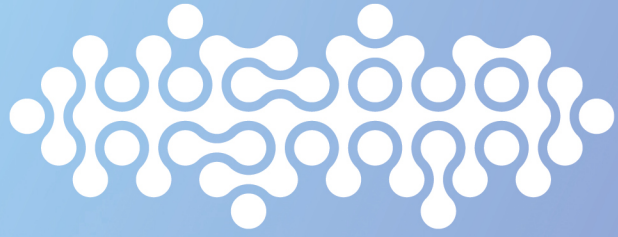
Am Ende dieser Einheit können Sie:

1. **C0** (Statement) und **C1** (Branch) Coverage von Hand **berechnen**
2. genau benennen, **was als Statement zählt** – und was nicht (Kommentare, Docstrings, Leerzeilen)
3. einen `pytest-cov` **-Bericht lesen** (Stmts, Miss, Branch, BrPart, Cover)
4. erklären, warum **100 % C0 $\neq$ 100 % C1** ist
5. die **Grenzen** von Coverage richtig einordnen

Fahrplan ($\approx$ 90 Minuten)

- **Warum Coverage?** — 10 min
- **Testtheorie kompakt** — 10 min
- **C0 berechnen** — 25 min
- **C1 berechnen** — 25 min
- **Subsumption** — 5 min
- **pytest-cov & CI** — 10 min
- **Grenzen & Zusammenfassung** — 5 min

*Roter Faden: Wir messen nicht nur Prozente – wir verstehen, **wie** sie zustande kommen.*



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Warum Coverage?

3

Die Lücke: „Tests grün“ ≠ „genug getestet“

Was die CI sagt:

-  Ruff: Stil ist sauber
-  Pyright: Typen passen
-  pytest: alle Tests grün

Was die CI NICHT sagt:

-  Wurde **jede Zeile** ausgeführt?
-  Wurde **jeder Branch** geprüft?
-  Welcher Code ist **ungetestet**?

Menschen schätzen schlecht, wie viel sie getestet haben. Coverage macht es messbar.

Coverage = objektive Metrik

Coverage = (vom Test ausgeführte Elemente / alle Elemente) × 100 %

- Eine **Zahl** statt Bauchgefühl (z. B. 72 %, 88 %, 100 %)
- Zeigt **konkrete Zeilennummern**, die ungetestet sind
- Sagt, **wo** noch Testfälle fehlen

⚠ *Coverage misst **Ausführung**, nicht Korrektheit. Hohe Coverage ≠ fehlerfreier Code – dazu später mehr.*

Testtheorie kompakt

- **Programm P** — der Code unter Test
- **Eingaberaum D** — alle möglichen Eingaben (meist unendlich)
- **Testsuite T** — endliche Teilmenge $T \subseteq D$, die wir wirklich ausführen
- **Modell $M(P)$** — Abstraktion des Programms (z. B. Kontrollflussgraph)
- **Kriterium C** — Regel, was „ausreichend getestet“ heißt

Kernidee: Coverage ist immer Coverage **eines Modells**. Anderes Modell → anderes Kriterium.

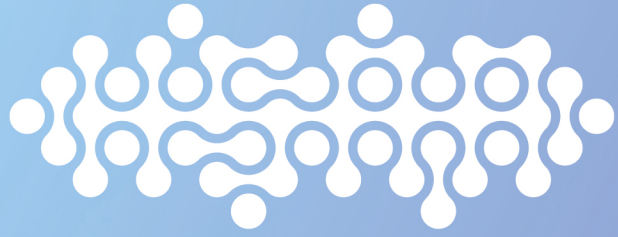
Das Modell: Kontrollflussgraph (CFG)

Ein **CFG** ist ein gerichteter Graph $G = (V, E)$:

- **Knoten (V)** = Anweisungen / Entscheidungen
- **Kanten (E)** = möglicher Kontrollfluss

C0 zählt **Knoten**, **C1** zählt **Kanten** aus Entscheidungen.

```
def classify(x):  
    if x > 0:  
        return "pos"  
    elif x < 0:  
        return "neg"  
    return "zero"
```



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Statement Coverage (C0)

3

C0 – Definition & Formel

Statement Coverage (C0): Jede ausführbare **Anweisung** mindestens einmal ausführen.

$$C0 = (\text{ausgeführte Statements} / \text{alle Statements}) \times 100 \%$$

- Modell: CFG, **Knoten = Statements**
- Schwächstes strukturelles Kriterium – nur die Basis
- **Aber:** Was genau ist „eine Anweisung“? Genau hier irren sich die meisten →

Was zählt als Statement?

Zählt NICHT:

- 🖐️ Kommentare (inline **und** Block)
- 🖐️ Leerzeilen
- 🖐️ Docstrings (Modul, Funktion, Klasse)
- 🖐️ Fortsetzungszeilen mehrzeiliger Statements

Zählt:

- ▶ `import` -Zeilen
- ▶ Zuweisungen / `return` / Aufrufe
- ▶ `def` - und `class` -Zeilen
- ▶ Bedingungen (`if`, `while`, ...)

Alle Zahlen auf den nächsten Folien stammen aus echten `coverage.py` -Läufen (`code_examples/coverage/`).

Beweis 1: Kommentare & Docstrings zählen nicht

1	"""Modul-Docstring."""	·	kein Statement
7	import math # Inline-Kommentar	▶	zählt (wegen import)
10	# Block-Kommentar	·	kein Statement
22	def addiere(a, b=2):	▶	zählt
26	"""Funktions-Docstring."""	·	kein Statement
27	# Kommentar im Körper	·	kein Statement
28	summe = a + b	▶	zählt

Legende: ▶ = Statement (zählt für C0) · = kein Statement

Ein Inline-Kommentar macht eine Zeile nicht „zählbar“ – der Code davor tut es.

Beweis 2: Signaturen laufen beim *Import*

```
17 def deko(fn):           ▶ zählt - „covered“ schon beim Import
42 def nie_aufgerufen():   ▶ zählt - „covered“ schon beim Import
43     """..."""           · kein Statement
44     wert = 1             x FEHLT (Funktion nie aufgerufen)
45     return wert         x FEHLT
```

- Die `def` -/ `class` -Zeile wird beim **Definieren** ausgeführt, nicht beim Aufruf.
- `nie_aufgerufen` wird **nie getestet** – trotzdem ist die `def` -Zeile abgedeckt, **nur der Körper fehlt**.

Beweis 3: Mehrzeilig = *ein* Statement

```
22 def addiere(          ▶ 1 Statement (die ganze Signatur)
23     a,                · gehört zu Zeile 22
24     b=2,              ·
25 ):                    ·
48 CONFIG = {            ▶ 1 Statement (das ganze Literal)
49     "a": 1,            ·
50     "b": 2,            ·
51 }
```

Ergebnis `was_zaeht.py` : 16 Statements, 14 ausgeführt, 2 fehlen

→ **$C0 = 14 / 16 = 88 \%$**



Mini-Übung: Wie viele Statements?

```
"""Begrüßung."""          # Zeile 1

def gruss(name):          # Zeile 3
    # baue die Nachricht
    text = f"Hallo {name}"
    return text            # Zeile 6
```

Frage an Sie: Wie viele ausführbare Statements zählt `coverage.py` hier?

→ **Lösung:** 3 — die Zeilen `def gruss`, `text = ...`, `return text`. Docstring (1) und Kommentar zählen **nicht**.

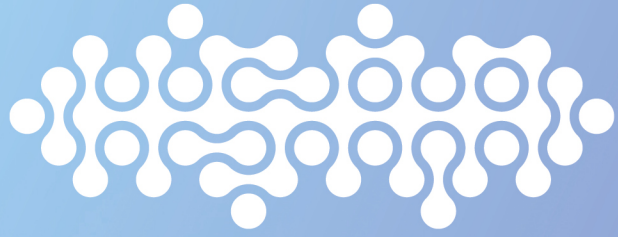
C0 berechnen: grade.py

```
def note(score):  
    """Doc."""  
    # Schutzklausel  
    if score < 0:  
        return "ungültig"  
    if score >= 90:  
        return "A"  
    return "B"
```

Tests: `note(95)→"A"`, `note(80)→"B"`

Name	Stmts	Miss	Cover	Missing
grade.py	6	1	83%	8

- 6 Statements, Zeile 8 nie ausgeführt
- **C0 = 5 / 6 = 83 %**
- Missing 8 = `return "ungültig"`
(kein Test mit `score < 0`)



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Branch Coverage (C1)

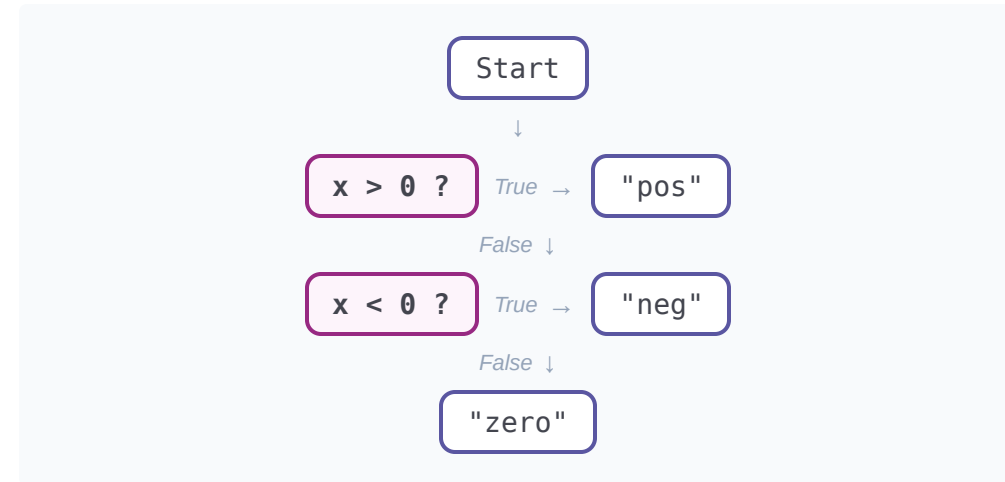
3

C1 – Definition & Formel

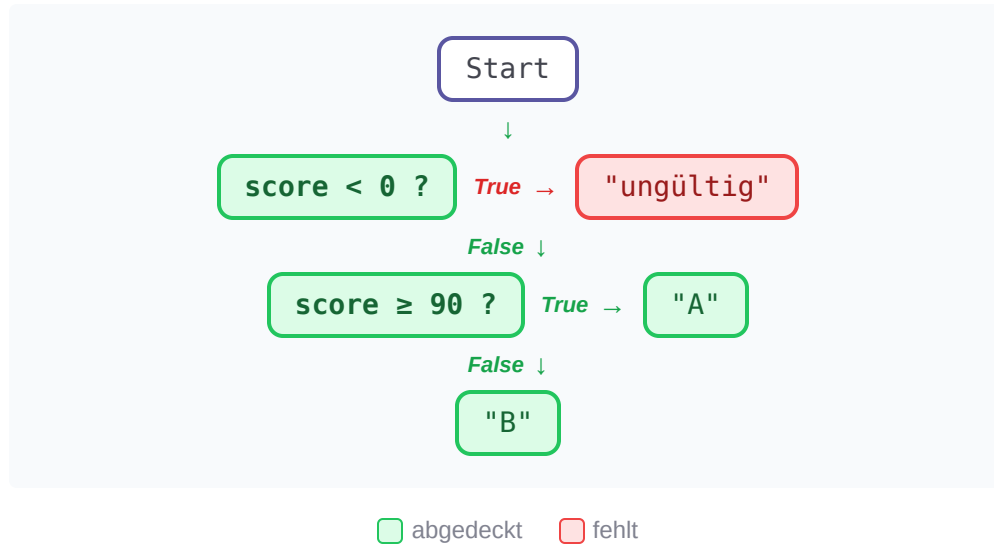
Branch Coverage (C1): Jedes **Ergebnis** (True und False) mindestens einmal.

C1 = (ausgeführte Branches / alle Branches) × 100 %

- Modell: CFG, **Kanten aus Entscheidungen**
- Jede einfache Entscheidung (`if` / `while`) hat **2** Branches (True/False)



C1 berechnen: `grade.py`



Entscheidung	True	False
<code>score < 0</code>	X	✓
<code>score ≥ 90</code>	✓	✓

- 4 Branches, 3 abgedeckt
- **$C1 = 3 / 4 = 75 \%$**
- Nur die True-Kante von `score < 0` fehlt (kein Test mit negativem Wert)

100 % C0 ≠ 100 % C1

```
def verarbeite(daten, pruefen=True):  
    if pruefen:  
        daten = daten.strip()  
    return daten.upper()
```

Test: nur `verarbeite(" hi ")` (Default `pruefen=True`)

- **Alle 4 Statements** laufen → **C0 = 100 %**
- `if pruefen` nur als **True** getestet → False-Kante fehlt
- **C1 = 1 / 2 = 50 %**
- `pytest-cov`: Missing 5->7

Merke: Volle Statement-Coverage kann ganze Branches übersehen. Deshalb ist C1 stärker.



Mini-Übung: Berechnen Sie C0 und C1

```
def rabatt(betrag, mitglied):  
    if betrag <= 0:  
        return 0  
    if mitglied:  
        return betrag * 0.9  
    return betrag
```

Test: nur `rabatt(100, True)`

→ **Lösung:** 6 Statements (`def` , 2× `if` , 3× `return`). Ausgeführt: 4 — `def` , beide `if` , `return betrag * 0.9` . Nicht ausgeführt: `return 0` , `return betrag` .

C0 = 4 / 6 = 67 % · **C1 = 2 / 4 = 50 %** (je nur eine Kante pro `if`).

Subsumption: C1 ist stärker als C0

C1 subsumiert C0:

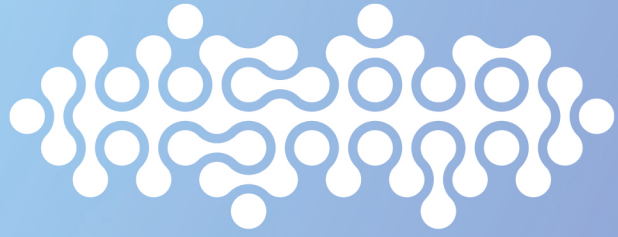
100 % C1 $\implies$ 100 % C0 (automatisch).

Beweis-Idee: Jede Kante ($v_1 \rightarrow v_2$) führt beide Knoten aus. Alle Kanten abgedeckt $\implies$ alle Knoten abgedeckt.

Umkehrung gilt NICHT (siehe vorige Folie).

Praxis: C1 statt C0+C1 messen. Ziel im Unit-Test: **70–80 %+ C1**.

Path Coverage
 $\supset$ subsumiert
Branch Coverage (C1)
 $\supset$
Statement Coverage (C0)
 $\supset$
Function Coverage



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

pytest-cov in der Praxis

3

Coverage messen mit pytest-cov

```
# Statement-Coverage + fehlende Zeilen
uv run pytest --cov=. --cov-report=term-missing

# zusätzlich Branch-Coverage (C1)
uv run pytest --cov=. --cov-branch --cov-report=term-missing
```

- `--cov=<paket>` — was gemessen wird
- `--cov-report=term-missing` — zeigt **fehlende Zeilennummern**
- `--cov-branch` — schaltet **Branch-Coverage** ein

Bericht lesen – und die %-Falle

Name	Stmts	Miss	Branch	BrPart	Cover	Missing
grade.py	6	1	4	1	80%	8

- **Stmts** 6 · **Miss** 1 (Zeile 8)
- **Branch** 4 · **BrPart** 1 (teilweise: nur eine Kante)
- **Cover 80 %** ist **kombiniert**:
 $(5 \text{ Stmts} + 3 \text{ Branches}) / (6 + 4) = 8/10$
- **Nicht** reines C0 (83 %) oder C1 (75 %)!

Coverage in der CI erzwingen

```
- name: Tests mit Coverage  
  run: uv run pytest --cov=. --cov-report=term-missing --cov-fail-under=70
```

- `--cov-fail-under=70` lässt die CI **fehlgeschlagen**, wenn Coverage < 70 %
- Verhindert, dass die Coverage unbemerkt **sinkt**
- Richtwerte: 70 % Minimum · 80 % gut · 90 % sehr gut · 100 % selten sinnvoll

Systematisch auf 100 % C1

1. **Alle Entscheidungspunkte** auflisten (`if`, `while`, `for`, ...)
2. Pro Entscheidung **beide Ergebnisse** erzeugen (True **und** False)
3. **Schleifen**: 0 / 1 / mehrere Durchläufe
4. **Alle Rückgabewege** (frühe Returns, Exceptions)
5. **Coverage erneut messen** und prüfen

Rückwärts denken: „Welche Eingabe macht diese Bedingung True? Welche False?“

Grenzen: 100 % ≠ fehlerfrei

```
def teile(a, b):  
    return a / b          # Bug: b == 0 nicht behandelt  
  
def test_teile():  
    assert teile(10, 2) is not None    # schwache Assertion
```

- 100 % Coverage – und der Bug bleibt: schwache Assertion, `b=0` ungetestet
- Coverage zeigt **Lücken**, garantiert aber **keine Korrektheit**
- **Ergänzen** mit Äquivalenzklassen & Grenzwerten

Zusammenfassung

- **C0** = Statements ausgeführt
- **C1** = Branch-Ergebnisse (T/F) ausgeführt
- Kommentare, Leerzeilen, **Docstrings zählen nicht**
- `def` / `class` zählen & laufen beim **Import**
- mehrzeilig = **ein** Statement
- **C1 subsumiert C0**; 100 % C0 $\neq$ 100 % C1
- `--cov-branch` : „Cover“ ist **kombiniert**
- 100 % Coverage $\neq$ fehlerfrei

Selbststudium & Übung

Projekte:

- [Selbststudium-Leitfaden](#)
- [Coverage-Konzepte Quiz](#)
- [CFG-Tracing-Übung](#)
- [Coverage-Detektiv-Übung](#)

Vorlesungen:

- [Lecture 6 – TDD & CI](#)
- [Lecture 7 – Testtheorie & Coverage](#)

Code zum Nachrechnen:

`code_examples/coverage/`



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Fragen?

DANKE