

Informatik I: Einführung in die Programmierung

9. Alternativen und Pattern Matching

Albert-Ludwigs-Universität Freiburg



UNI
FREIBURG

Prof. Dr. Peter Thiemann

19. November 2025



UNI
FREIBURG

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Alternativen und Pattern Matching



UNI
FREIBURG

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Entwurf mit Alternativen

Spielkarten

Eine Spielkarte ist (alternativ) entweder

- ein Joker oder
- eine natürliche Karte mit einer Farbe und einem Rang.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Entwurf mit Alternativen



Spielkarten

Eine Spielkarte ist (alternativ) entweder

- ein Joker oder
- eine natürliche Karte mit einer Farbe und einem Rang.

Schritt 1: Bezeichner und Datentypen

Eine Spielkarte hat **eine von zwei Ausprägungen**.

- Joker werden durch Objekte der Klasse Joker repräsentiert.
- Natürliche Karten werden durch Objekte der Klasse FaceCard mit Attributen suit (Farbe) und rank (Rang) repräsentiert.

Farbe ist eine von '**Clubs**', '**Spades**', '**Hearts**', '**Diamonds**'.

Rang ist einer von 2, 3, 4, 5, 6, 7, 8, 9, 10, '**Jack**', '**Queen**', '**King**', '**Ace**'.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung



Farben

```
from typing import Literal
type Suit = Literal['Clubs', 'Spades', 'Hearts', 'Diamonds']
```

Der Typ Suit enthält genau die aufgelisteten Strings.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Farben

```
from typing import Literal
type Suit = Literal['Clubs', 'Spades', 'Hearts', 'Diamonds']
```

Der Typ Suit enthält genau die aufgelisteten Strings.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Ränge

```
type Rank = ( Literal[2,3,4,5,6,7,8,9,10]
              | Literal['Ace', 'King', 'Queen', 'Jack'])
```

Der Typ Rank enthält genau die aufgelisteten Werte (Mischung aus int und str).

Entwurf mit Alternativen



Schritt 2: Klassengerüst

```
@dataclass
class Joker:
    pass # no attributes

@dataclass
class FaceCard:
    suit: Suit
    rank: Rank

type Card = FaceCard | Joker
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

- Eine Karte Card kann **alternativ** FaceCard oder Joker sein.
- Das lässt sich ausdrücken durch einen **Union-Typ**: FaceCard | Joker.

Beispiel: Rommé



Figuren in Rommé erkennen

Ein Figur im Rommé ist entweder

- ein Satz (*set*): drei oder vier Karten gleichen Rangs in verschiedenen Farben,
- eine Reihe (*run*): mindestens drei Karten gleicher Farbe mit aufsteigenden Rängen

Eine Karte in einer Figur darf durch einen Joker ersetzt werden. Joker (dürfen nicht nebeneinander liegen und) dürfen nicht in der Überzahl sein.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Beispiel: Romm 



Figuren in Romm  erkennen

Ein Figur im Romm  ist entweder

- ein Satz (*set*): drei oder vier Karten gleichen Rangs in verschiedenen Farben,
- eine Reihe (*run*): mindestens drei Karten gleicher Farbe mit aufsteigenden R gen

Eine Karte in einer Figur darf durch einen Joker ersetzt werden. Joker (d rken nicht nebeneinander liegen und) d rken nicht in der 超erzahl sein.

Erste Aufgabe: Erkenne einen Satz

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufz hlungstypen

Zusammen-
fassung

Beispiel: Rommé



Figuren in Rommé erkennen

Ein Figur im Rommé ist entweder

- ein Satz (*set*): drei oder vier Karten gleichen Rangs in verschiedenen Farben,
- eine Reihe (*run*): mindestens drei Karten gleicher Farbe mit aufsteigenden Rängen

Eine Karte in einer Figur darf durch einen Joker ersetzt werden. Joker (dürfen nicht nebeneinander liegen und) dürfen nicht in der Überzahl sein.

Erste Aufgabe: Erkenne einen Satz

Schritt 1: Bezeichner und Datentypen

Die Funktion `is_rummy_set` nimmt als Argument eine Liste `cards` von Spielkarten und liefert `True` gdw. `cards` ein Satz ist.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Satz erkennen



Schritt 2: Funktionsgerüst

```
def is_rummy_set (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung



Satz erkennen

Schritt 2: Funktionsgerüst

```
def is_rummy_set (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

- Länge der Liste prüfen (drei oder vier)

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung



Satz erkennen

Schritt 2: Funktionsgerüst

```
def is_rummy_set (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

- Länge der Liste prüfen (drei oder vier)
- Liste cards verarbeiten: **for** Schleife mit Akkumulator

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Satz erkennen



Schritt 2: Funktionsgerüst

```
def is_rummy_set (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

- Länge der Liste prüfen (drei oder vier)
- Liste cards verarbeiten: **for** Schleife mit Akkumulator
- Anzahl der Joker prüfen (nicht in der Überzahl)

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Satz erkennen



Schritt 2: Funktionsgerüst

```
def is_rummy_set (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

- Länge der Liste prüfen (drei oder vier)
- Liste cards verarbeiten: **for** Schleife mit Akkumulator
- Anzahl der Joker prüfen (nicht in der Überzahl)
- Natürliche Karten auf gleichen Rang prüfen

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Satz erkennen



Schritt 2: Funktionsgerüst

```
def is_rummy_set (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

- Länge der Liste prüfen (drei oder vier)
- Liste cards verarbeiten: **for** Schleife mit Akkumulator
- Anzahl der Joker prüfen (nicht in der Überzahl)
- Natürliche Karten auf gleichen Rang prüfen
- Natürliche Karten auf unterschiedliche Farben prüfen

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Satz erkennen



Schritt 3: Beispiele

```
c1 = FaceCard ('Clubs', 'Queen')
c2 = FaceCard ('Hearts', 'Queen')
c3 = FaceCard ('Spades', 'Queen')
c4 = FaceCard ('Diamonds', 'Queen')
c5 = FaceCard ('Diamonds', 'King')
j1 = Joker ()

assert not is_rummy_set ([c1,c2])
assert is_rummy_set ([c1, c2, c3])
assert is_rummy_set ([c1, c2, j1])
assert is_rummy_set ([j1, c2, c3])
assert not is_rummy_set ([j1, c5, c4])
assert is_rummy_set ([c2, c3, c1, c4])
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Satz erkennen



Schritt 4: Funktionsdefinition

```
def is_rummy_set (cards : list[Card]) -> bool:
    if len (cards) < 3 or len (cards) > 4:
        return False
    common_rank = None      # common rank
    suits = []             # suits already seen
    nr_jokers = 0
    for card in cards:
        if is_joker (card):
            nr_jokers = nr_jokers + 1
        else:          # a natural card
            if not common_rank:
                common_rank = card.rank
            elif common_rank != card.rank:
                return False
            if card.suit in suits:
                return False # repeated suit
            else:
                suits = suits + [card.suit]
    return 2 * nr_jokers <= len (cards)
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Satz erkennen / is_joker



Schritt 1: Wunschdenken

Die Funktion `is_joker` nimmt als Argument eine Karte `card` und liefert `True` gdw. `card` ein Joker ist.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung



Satz erkennen / is_joker

Schritt 1: Wunschdenken

Die Funktion `is_joker` nimmt als Argument eine Karte `card` und liefert True gdw. `card` ein Joker ist.

Schritt 2: Funktionsgerüst

```
def is_joker (card : Card) -> bool:  
    ...
```

Zur Verdeutlichung expandieren wir die Definition von `Card`:

```
def is_joker (card : FaceCard | Joker) -> bool:  
    ...
```

Wir müssen eine Alternative bearbeiten. Dafür verwenden wir eine neue Anweisung — Pattern matching!

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Pattern Matching

■ Beim Entwurf mit Alternativen

- ein Argument kann aus einer von mehreren Klassen stammen
- erkennbar am Union-Typ

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Pattern Matching



- Beim Entwurf mit Alternativen
 - ein Argument kann aus einer von mehreren Klassen stammen
 - erkennbar am Union-Typ
- Funktionsgerüst für Alternative auf $x : T_1 \mid T_2 \mid \dots$
 - Pattern Matching gegen x : **match** x :
 - Ein Fall für jedes T_i : **case** $T_i()$:
 - Im Rumpf des **case** Zugriff auf die Attribute von T_i
 - mögliche Fehlerquelle: Verwechslung der Attribute!

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

- Beim Entwurf mit Alternativen
 - ein Argument kann aus einer von mehreren Klassen stammen
 - erkennbar am Union-Typ
- Funktionsgerüst für Alternative auf $x : T_1 \mid T_2 \mid \dots$
 - Pattern Matching gegen x : **match** x :
 - Ein Fall für jedes T_i : **case** $T_i()$:
 - Im Rumpf des **case** Zugriff auf die Attribute von T_i
 - mögliche Fehlerquelle: Verwechslung der Attribute!
- Noch besser:
 - Gleichzeitiger Test und Zugriff auf die Attribute
 - Schreibe die Fälle als **case** $T_i(a_1, \dots, a_n)$:
 - die a_i sind Variable für die Attribute sind.

Pattern Matching / Beispiel



Schritt 4: Funktionsdefinition (Wunschdenken)

```
def is_joker (card : FaceCard | Joker) -> bool:  
    match card:  
        case Joker():  
            return True  
        case FaceCard():  
            return False
```

- Die cases werden von oben nach unten abgearbeitet.
- Sie sollten alle möglichen Fälle des Typs abdecken. (Teilweise überprüft von vscode.)

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Syntax

```
match expr:  
    case pattern1 :  
        block1  
    case pattern2 :  
        block2  
    ...
```

- `match` und `case` sind Schlüsselworte.
- `expr` ist ein beliebiger Ausdruck.
- Jedes `pattern` beschreibt einen **Test** gegen den Wert von `expr`.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Beispiel: Kartenwerte in Romm 



F r die Punktabrechnung besitzt jede Spielkarte in Romm  einen Wert.

Rang	Wert in Punkten
Zwei bis Neun	Entsprechend dem Rang (2, 3, 4, 5, 6, 7, 8, 9)
Zehn, Bube, Dame, K�nig	10
Ass	11
Joker	20

Wert einer Karte in Romm 

Die Funktion `card_value` nimmt als Argument eine `Card` und liefert als Ergebnis ein `int` entsprechend dem Wert von `card`.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufz hlungstypen

Zusammen-
fassung

Beispiele



```
assert card_value (Joker()) == 20
assert card_value (FaceCard ('Hearts', 'Ace')) == 11
assert card_value (FaceCard ('Spades', 'Queen')) == 10
assert card_value (FaceCard ('Diamonds', 6)) == 6
```

```
def card_value (card: Card) -> int:  
    match card:  
        case Joker():  
            return ...  
        case FaceCard(suit, rank):  
            return ...
```

- Das Pattern `Joker()` passt nur, wenn `card` eine Instanz von `Joker` ist.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

```
def card_value (card: Card) -> int:  
    match card:  
        case Joker():  
            return ...  
        case FaceCard(suit, rank):  
            return ...
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

- Das Pattern `Joker()` passt nur, wenn `card` eine Instanz von `Joker` ist.
- Das Pattern `FaceCard(suit, rank)` passt, wenn `card` eine Instanz von `FaceCard` ist.
 - Im zugehörigen Block sind `suit` und `rank` an die entsprechenden Attribute von `card` gebunden.
 - Die Reihenfolge der Attribute entspricht der Reihenfolge in der Deklaration der Klasse `FaceCard` als `@dataclass`.

Gerüst + erstes Beispiel

```
def card_value (card: Card) -> int:  
    match card:  
        case Joker():  
            return 20  
        case FaceCard(_, rank):  
            return ...
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

- Das erste case gibt das Ergebnis für Joker vor.

Gerüst + erstes Beispiel

```
def card_value (card: Card) -> int:  
    match card:  
        case Joker():  
            return 20  
        case FaceCard(_, rank):  
            return ...
```

- Das erste `case` gibt das Ergebnis für `Joker` vor.
- Im zweiten `case` spielt die Farbe für die Bestimmung der Kartenwerts keine Rolle. Dort wird das **Wildcard-Pattern** `_` verwendet, das auf jeden beliebigen Wert passt und keine Variablenbindung vornimmt.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Gerüst + erstes Beispiel

```
def card_value (card: Card) -> int:  
    match card:  
        case Joker():  
            return 20  
        case FaceCard(_, rank):  
            return ...
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

- Das erste `case` gibt das Ergebnis für `Joker` vor.
- Im zweiten `case` spielt die Farbe für die Bestimmung der Kartenwerts keine Rolle. Dort wird das **Wildcard-Pattern** `_` verwendet, das auf jeden beliebigen Wert passt und keine Variablenbindung vornimmt.
- Zur Analyse des `rank`-Attributes können Pattern **geschachtelt** werden.

Gerüst + restliche Beispiele



```
match card:  
    case FaceCard(_, 'Ace'):  
        return 11  
    case FaceCard(_, 'Jack' | 'Queen' | 'King'):  
        return 10  
    case FaceCard(_, int(i)):  
        return i
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

- Das Literal-Pattern 'Ace' passt nur auf den String 'Ace'.

Gerüst + restliche Beispiele



```
match card:  
    case FaceCard(_, 'Ace'):  
        return 11  
    case FaceCard(_, 'Jack' | 'Queen' | 'King'):  
        return 10  
    case FaceCard(_, int(i)):  
        return i
```

- Das Literal-Pattern 'Ace' passt nur auf den String 'Ace'.
- Das Oder-Pattern 'Jack' | 'Queen' | 'King' passt auf einen der Strings 'Jack' oder 'Queen' oder 'King'.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Gerüst + restliche Beispiele



```
match card:  
    case FaceCard(_, 'Ace'):  
        return 11  
    case FaceCard(_, 'Jack' | 'Queen' | 'King'):  
        return 10  
    case FaceCard(_, int(i)):  
        return i
```

- Das Literal-Pattern '`Ace`' passt nur auf den String '`Ace`'.
- Das Oder-Pattern '`Jack`' | '`Queen`' | '`King`' passt auf einen der Strings '`Jack`' oder '`Queen`' oder '`King`'.
- Das Pattern `int(i)` passt, falls `card` eine Instanz von `int` ist und bindet die Zahl an `i`.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Beispiel: Rommé Satz erkennen



Schritt 4: Funktionsdefinition

```
def is_rummy_set(cards: list[Card]) -> bool:
    if len(cards) < 3 or len(cards) > 4:
        return False
    common_rank = None # common rank
    suits = [] # suits already seen
    nr_jokers = 0
    for card in cards:
        match card:
            case Joker():
                nr_jokers = nr_jokers + 1
            case FaceCard(suit, rank):
                if not common_rank:
                    common_rank = rank
                elif common_rank != rank:
                    return False
                if suit in suits:
                    return False
                suits = suits + [suit]
    return 2 * nr_jokers <= len(cards)
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

- Variable:
Das Pattern passt auf jeden Wert und weist ihn der Variable zu.
- Klassenname ($pattern_1, \dots, pattern_n$):
Das Pattern passt, wenn der Wert eine Instanz von Klassenname ist **und** alle Teilpattern $pattern_i$ auf das entsprechende Attribut der Instanz passen.
Es dürfen nicht mehr Pattern angegeben werden, als Attribute vorhanden sind. Ansonsten werden die ersten n Attribute geprüft.
- Konstante:
Das Pattern passt, wenn der Wert gleich der Konstante ist.
- weitere Möglichkeiten:
Listen von Patterns, Tupel von Patterns, ... Vgl. Dokumentation.

Semantik

```
match expr:  
    case pattern1 :  
        block1  
    case pattern2 :  
        block2  
    :  
    :
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

- Werte zuerst *expr* zu *v* aus.
- Dann prüfe ob *pattern₁* auf *v* passt; falls ja, führe *block₁* aus; Variablen im Pattern werden entsprechend *v* zugewiesen; danach nächste Anweisung nach dem **match**.
- Sonst prüfe *pattern₂* usw. bis das erste passende Pattern gefunden wird.
- Nächste Anweisung, falls kein Pattern passt.

Reihe erkennen



UNI
FREIBURG

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Schritt 1: Bezeichner und Datentypen

Die Funktion `is_rummy_run` nimmt als Argument eine Liste `cards : list[Card]` von Spielkarten und liefert `True` gdw. `cards` eine Reihe ist.

Reihe erkennen



Schritt 2: Funktionsgerüst

```
def is_rummy_run (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Reihe erkennen



Schritt 2: Funktionsgerüst

```
def is_rummy_run (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

- Länge der Liste prüfen

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Reihe erkennen



Schritt 2: Funktionsgerüst

```
def is_rummy_run (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

- Länge der Liste prüfen
- Liste verarbeiten: **for** Schleife mit Akkumulator

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Reihe erkennen



Schritt 2: Funktionsgerüst

```
def is_rummy_run (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

- Länge der Liste prüfen
- Liste verarbeiten: **for** Schleife mit Akkumulator
- Anzahl der Joker prüfen

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Reihe erkennen



Schritt 2: Funktionsgerüst

```
def is_rummy_run (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

- Länge der Liste prüfen
- Liste verarbeiten: **for** Schleife mit Akkumulator
- Anzahl der Joker prüfen
- Natürliche Karten auf gleiche Farbe prüfen

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Reihe erkennen



Schritt 2: Funktionsgerüst

```
def is_rummy_run (cards : list[Card]) -> bool:  
    # initialization of acc  
    for card in cards:  
        pass # action on single card  
    # finalization  
    return ...
```

- Länge der Liste prüfen
- Liste verarbeiten: **for** Schleife mit Akkumulator
- Anzahl der Joker prüfen
- Natürliche Karten auf gleiche Farbe prüfen
- Natürliche Karten auf aufsteigende Werte prüfen

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Reihe erkennen



Schritt 3: Beispiele

```
c2, cq = FaceCard ('Clubs', 2), FaceCard ('Clubs', 'Queen')
ck, ca = FaceCard ('Clubs', 'King'), FaceCard ('Clubs', 'Ace')
dq, d10 = FaceCard ('Diamonds', 'Queen'), FaceCard ('Diamonds', 10)
jj      = Joker ()

assert not is_rummy_run ([cq, ck])
assert is_rummy_run ([cq, ck, ca])
assert not is_rummy_run ([dq, ck, ca])
assert is_rummy_run ([d10,jj,dq])
assert not is_rummy_run ([d10,jj,dq,ck])
assert not is_rummy_run ([ck, ca, c2])
assert not is_rummy_run ([d10, jj, jj])
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Reihe erkennen

Schritt 3: Funktionsdefinition

```
def is_rummy_run (cards : list[Card]) -> bool:
    if len (cards) < 3:      # check length of list
        return False
    # initialization of accumulators
    nr_jokers = 0            # count jokers
    current_rank = None       # keep track of rank
    common_suit = None
    for card in cards:
        if current_rank:
            current_rank = next_rank (current_rank)
        # action on single card
        match card:
            case Joker():
                nr_jokers = nr_jokers + 1
            case FaceCard(suit, rank):
                if not current_rank:
                    current_rank = rank
                elif current_rank != rank:
                    return False
                if not common_suit:
                    common_suit = suit
                elif common_suit != suit:
                    return False
    # finalization
    return 2 * nr_jokers <= len (cards)
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Reihe erkennen



UNI
FREIBURG

Was noch fehlt ...

- Wunschdenken: `next_rank`
- Joker nebeneinander?
- Joker außerhalb der Reihe?
- ... Wer sagt eigentlich, dass `cards` nach Rang aufsteigend sortiert ist?

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung



Aufzählungstypen

Aufzählungstypen (Enumerations)



UNI
FREIBURG

- Das Rommé-Beispiel verwendet verschiedene Strings zur Modellierung der Farben und einiger Ränge.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching

Aufzählungstypen

Zusammen-
fassung

Aufzählungstypen (Enumerations)



- Das Rommé-Beispiel verwendet verschiedene Strings zur Modellierung der Farben und einiger Ränge.
- Prinzipiell könnten leicht illegale Karten erzeugt werden, aber praktisch wird das durch den Typchecker abgefangen:

```
illegal_card = FaceCard ('Ace', 'Ace') ## ??
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching

Aufzählungstypen

Zusammen-
fassung

Aufzählungstypen (Enumerations)



- Das Rommé-Beispiel verwendet verschiedene Strings zur Modellierung der Farben und einiger Ränge.
- Prinzipiell könnten leicht illegale Karten erzeugt werden, aber praktisch wird das durch den Typchecker abgefangen:

```
illegal_card = FaceCard ('Ace', 'Ace') ## ??
```
- Bei `suit` und `rank` handelt es sich jeweils um degenerierte Alternativen, wo nur Objekte aufgelistet werden.

Aufzählungstypen (Enumerations)



- Das Rommé-Beispiel verwendet verschiedene Strings zur Modellierung der Farben und einiger Ränge.
- Prinzipiell könnten leicht illegale Karten erzeugt werden, aber praktisch wird das durch den Typchecker abgefangen:

```
illegal_card = FaceCard ('Ace', 'Ace') ## ??
```
- Bei `suit` und `rank` handelt es sich jeweils um degenerierte Alternativen, wo nur Objekte aufgelistet werden.
- Wie können wir darauf die gewünschte Sortierung vorgeben?

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Aufzählungstypen (Enumerations)



- Das Rommé-Beispiel verwendet verschiedene Strings zur Modellierung der Farben und einiger Ränge.
- Prinzipiell könnten leicht illegale Karten erzeugt werden, aber praktisch wird das durch den Typchecker abgefangen:

```
illegal_card = FaceCard ('Ace', 'Ace') ## ??
```
- Bei `suit` und `rank` handelt es sich jeweils um degenerierte Alternativen, wo nur Objekte aufgelistet werden.
- Wie können wir darauf die gewünschte Sortierung vorgeben?
- Dies kann mit einem **Aufzählungstypen** (Enumeration) modelliert werden!

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching

Aufzählungstypen

Zusammen-
fassung

Definition von Farben als Enumeration



```
from enum import IntEnum
class Suit(IntEnum):
    CLUBS = 0
    SPADES = 1
    HEARTS = 2
    DIAMONDS = 3
```

- Definiert vier Objekte `Suit.CLUBS`, `Suit.SPADES`, `Suit.HEARTS` und `Suit.DIAMONDS`.
- Sie alle sind Instanzen der *Enumeration* `Suit`.
- Es gilt `Suit.CLUBS == Suit(0)`.
- Aber `Suit('Ace')` liefert einen Fehler!
- Alle Instanzen einer Enumeration besitzen Attribute `name` und `value` mit `Suit.CLUBS.name == 'CLUBS'` und `Suit.CLUBS.value == 0`.

Alternativen
und Pattern
Matching
Entwurf mit
Alternativen
Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Enumerations in Patterns



```
@dataclass
class FaceCard:
    suit: Suit
    rank: int | str
...
match card:
    # is this the ace of spades?
    case FaceCard (Suit.SPADES, 'Ace'):
        ...

```

- Das Pattern `Suit.SPADES` passt auf den entsprechenden Wert.
- (Erweiterung: Enumeration Rank für 1, ..., 10, Bube, Dame, König, Ass)

Iteration über Enumeration



Rommé Satz erzeugen

Gegeben einen Rang, erzeuge den Satz zu diesem Rang aus natürlichen Karten.

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Iteration über Enumeration



Rommé Satz erzeugen

Gegeben einen Rang, erzeuge den Satz zu diesem Rang aus natürlichen Karten.

Gerüst

```
def create_rummy_set (rank: Rank) -> list[FaceCard]:  
    # fill in  
    return ...
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Iteration über Enumeration

Rommé Satz erzeugen

Gegeben einen Rang, erzeuge den Satz zu diesem Rang aus natürlichen Karten.

Gerüst

```
def create_rummy_set (rank: Rank) -> list[FaceCard]:  
    # fill in  
    return ...
```

Implementierung

```
def create_rummy_set (rank: Rank) -> list[FaceCard]:  
    result = []  
    for s in Suit:      # iterate over all elements of Suit  
        result = result + [FaceCard (s, rank)]  
    return result
```

Alternativen
und Pattern
Matching

Entwurf mit
Alternativen

Pattern Matching
Aufzählungstypen

Zusammen-
fassung

Zusammenfassung

Zusammenfassung



UNI
FREIBURG

- Entwurf mit **Alternativen**.

Alternativen
und Pattern
Matching

Zusammen-
fassung

Zusammenfassung



UNI
FREIBURG

Alternativen
und Pattern
Matching

Zusammen-
fassung

- Entwurf mit **Alternativen**.
- Ein **Typtest** kann durch Identitätstest gegen die Klasse, `isinstance` oder Pattern Matching geschehen.

Zusammenfassung



UNI
FREIBURG

Alternativen
und Pattern
Matching

Zusammen-
fassung

- Entwurf mit **Alternativen**.
- Ein **Typtest** kann durch Identitätstest gegen die Klasse, `isinstance` oder Pattern Matching geschehen.
- Pattern Matching vereinigt alle Typtests, die Projektion der Attribute und die Fallunterscheidungen.

Zusammenfassung



- Entwurf mit **Alternativen**.
- Ein **Typtest** kann durch Identitätstest gegen die Klasse, `isinstance` oder Pattern Matching geschehen.
- Pattern Matching vereinigt alle Typtests, die Projektion der Attribute und die Fallunterscheidungen.
- Patterns können geschachtelt werden.

Zusammenfassung



- Entwurf mit **Alternativen**.
- Ein **Typtest** kann durch Identitätstest gegen die Klasse, `isinstance` oder Pattern Matching geschehen.
- Pattern Matching vereinigt alle Typtests, die Projektion der Attribute und die Fallunterscheidungen.
- Patterns können geschachtelt werden.
- Aufzählungstypen enthalten eine fest vordefinierte Anzahl von “frischen” Werten.

Zusammenfassung



- Entwurf mit **Alternativen**.
- Ein **Typtest** kann durch Identitätstest gegen die Klasse, `isinstance` oder Pattern Matching geschehen.
- Pattern Matching vereinigt alle Typtests, die Projektion der Attribute und die Fallunterscheidungen.
- Patterns können geschachtelt werden.
- Aufzählungstypen enthalten eine fest vordefinierte Anzahl von “frischen” Werten.
- Iteration über die Elemente eines Aufzählungstyps.