Informatik I: Einführung in die Programmierung

8. Objekte und Klassen zum Ersten

Albert-Ludwigs-Universität Freiburg

Peter Thiemann

20. November 2018

1 Objekte und Klassen

- Objekte
- Identität und Gleichheit
- Klassen für Records
- Klassendefinition
- Instanzenerzeugung
- Funktionen auf Records
- Geschachtelte Records
- Objekte anzeigen
- Entwurf mit Alternativen

1 Objekte und Klassen

- Objekte
- Identität und Gleichheit
- Klassen für Records
- Klassendefinition
- Instanzenerzeugung
- Funktionen auf Records
- Geschachtelte Records
- Objekte anzeigen
- Entwurf mit Alternativen

Objekte und Attribute

- Alle *Werte* in Python sind in Wirklichkeit *Objekte*.
- Damit ist gemeint, dass sie nicht nur aus reinen *Daten* bestehen, sondern auch assoziierte *Attribute* und *Methoden* haben, auf die mit der Punktnotation

  `ausdruck.attribut`

  zugegriffen werden kann:

Python-Interpreter

```
>>> x = complex(10, 3)
>>> x.real, x.imag
10.0 3.0
>>> "spam".index("a")
2
>>> (10 + 10).__neg__()
-20
```

Identität: `is` und `is not`

- Jedes Objekt besitzt eine Identität, die es von allen anderen Objekten unterscheidet.
- Die Operatoren `is` und `is not` testen die Identität:
- `x is y` ist `True`, wenn `x` und `y` dasselbe Objekt bezeichnen, und ansonsten `False` (`is not` umgekehrt):

Python-Interpreter

```
>>> x = ["ham", "spam", "jam"]
>>> y = ["ham", "spam", "jam"]
>>> z = y
>>> x is y, x is z, y is z
(False, False, True)
>>> x is not y, x is not z, y is not z
(True, True, False)
>>> del y[1]
>>> x, y, z
(['ham', 'spam', 'jam'], ['ham', 'jam'], ['ham', 'jam'])
```

1 Objekte und Klassen

- Objekte
- Identität und Gleichheit
- Klassen für Records
- Klassendefinition
- Instanzenerzeugung
- Funktionen auf Records
- Geschachtelte Records
- Objekte anzeigen
- Entwurf mit Alternativen

Identität vs. Gleichheit

- Außer Zahlen und Strings können auch Listen und Tupel auf Gleichheit getestet werden. Der Unterschied zum Identitätstest ist wichtig:

Python-Interpreter

```
>>> x = ["ham", "spam", "jam"]
>>> y = ["ham", "spam", "jam"]
>>> x == y, x is y
(True, False)
```

- Test auf *Gleichheit*: haben `x` und `y` den gleichen Typ, sind sie gleich lang und sind korrespondierende Elemente gleich? (die Definition ist rekursiv)
- Test auf *Identität*: bezeichnen `x` und `y` dasselbe Objekt?

Faustregel

Verwende in der Regel den Gleichheitstest.

Spezialfall `x is None`

Anmerkung zu None:

- Der Typ `NoneType` hat nur einen einzigen Wert: `None`. Daher ist es egal, ob ein Vergleich mit `None` per Gleichheit oder per Identität erfolgt.
- Es hat sich eingebürgert, Vergleiche mit `None` immer als
  `x is None` bzw. `x is not None`
  und nicht als
  `x == None` bzw. `x != None`
  zu schreiben.

Veränderlich oder unveränderlich?

Jetzt können wir auch genauer sagen, was es mit veränderlichen (*mutable*) und unveränderlichen (*immutable*) Datentypen auf sich hat:

- Instanzen von veränderlichen Datentypen können modifiziert werden. Vorsicht bei Zuweisungen wie `x = y`: Nachfolgende Operationen auf `x` beeinflussen auch `y`.
  - Beispiel: Listen (**`list`**)
- Instanzen von unveränderlichen Datentypen können nicht modifiziert werden. Daher sind Zuweisungen wie `x = y` völlig unkritisch:
  Da das durch `x` bezeichnete Objekt nicht verändert werden kann, besteht keine Gefahr für `y`.
  - Beispiele: Zahlen (**`int`**, **`float`**, **`complex`**), Strings (**`str`**), Tupel (**`tuple`**)

1 Objekte und Klassen

- Objekte
- Identität und Gleichheit
- Klassen für Records
- Klassendefinition
- Instanzenerzeugung
- Funktionen auf Records
- Geschachtelte Records
- Objekte anzeigen
- Entwurf mit Alternativen

Klassen beschreiben Objekte

- Bisher haben wir vorgefertigte Objekte verwendet,
- Jetzt beginnen wir selbst welche zu bauen!
- Dafür benötigen wir einen Bauplan.

Definition

Ein **Record** ist ein Objekt, das mehrere untergeordnete Objekte, die **Attribute**, enthält.

- alternativ: **Struct**; deutsch: Reihung, Struktur
- Objekte heißen auch **Instanzen**.
- Attribute heißen auch **Felder**.

Beispiel für ein Record: Ware

Beschreibung für Ware

Ein Händler beschreibt eine Ware durch den Namen und den Angebotspreis.

Schritt 1: Bezeichner und Datentypen

Ein Händler beschreibt eine Ware (`Article`) durch die Attribute

- `name : string`, den Namen und
- `price : int`, den Angebotspreis (in cent), immer ≥ 0.

1 Objekte und Klassen

- Objekte
- Identität und Gleichheit
- Klassen für Records
- Klassendefinition
- Instanzenerzeugung
- Funktionen auf Records
- Geschachtelte Records
- Objekte anzeigen
- Entwurf mit Alternativen

Definieren von Records und Klassen

Python-Interpreter

```
>>> class Article:
...     pass # nur notwendig für leere Klasse!
...
>>> Article
<class '__main__.Article'>
>>> int
<class 'int'>
```

- Neue Records und Klassen werden mit der `class`-Anweisung eingeführt (Konvention: CamelCase-Namen).
- Die `class`-Anweisung muss ausgeführt werden. Sie sollte nicht in einer bedingten Anweisung verborgen werden!
- Definiert einen neuen Typ `Article`.

1 Objekte und Klassen

- Objekte
- Identität und Gleichheit
- Klassen für Records
- Klassendefinition
- Instanzenerzeugung
- Funktionen auf Records
- Geschachtelte Records
- Objekte anzeigen
- Entwurf mit Alternativen

Erzeugung von Instanzen

- Jeder Aufruf der Klasse als Funktion erzeugt ein neue Instanz der Klasse.

Python-Interpreter

```
>>> class Article:
...     pass
...
>>> instance1 = Article()
>>> instance1
<__main__.Article object at 0x101ac51d0>
>>> instance2 = Article()
>>> instance1 is instance2
False
>>> instance1 == instance2
False
```

- Alle erzeugten Instanzen sind untereinander nicht-identisch und ungleich!

Instanzen sind dynamische Strukturen/Records

- Instanzen sind Records, die *dynamisch* neue Attribute erhalten können.

Python-Interpreter

```
>>> class Article:
...     pass
...
>>> phone = Article()
>>> phone.name = "Smartphone"
>>> phone.price = 49500
>>> phone.price * 0.19 / 1.19
7903.361344537815
```

- D.h. jede Instanz kann dynamisch neue Attribute erhalten – jede Instanz hat einen eigenen Namensraum, auf den die Punktnotation zugreift.
- Besser: gleiche Attribute für alle Instanzen einer Klasse!

Einfache Klasse: Article

Schritt 2: Klassengerüst

```
class Article:
    def __init__(self, name, price):
        self.name = name
        self.price = price
```

- Wenn eine `__init__` Funktion innerhalb der `class`-Anweisung definiert wird, dann wird sie automatisch beim Erzeugen einer Instanz aufgerufen.
- Die Klasse hat immer ein Argument weniger als die `__init__` Funktion!
- Der erste Parameter heißt immer `self` (Konvention) und enthält das neue Objekt.

Beispiel

Python-Interpreter

```
>>> class Article:
...     def __init__(self, name, price):
...         self.name = name
...         self.price = price
...
>>> phone = Article("Smartphone", 49500)
>>> phone.price * 0.19 / 1.19
7903.361344537815
```

1 Objekte und Klassen

- Objekte
- Identität und Gleichheit
- Klassen für Records
- Klassendefinition
- Instanzenerzeugung
- Funktionen auf Records
- Geschachtelte Records
- Objekte anzeigen
- Entwurf mit Alternativen

Beispiel

Angebotspreis

Der Händler will seine Preise am Black Friday um 25% herabsetzen. Der Angebotspreis soll dynamisch nur an der Kasse berechnet werden.

Schritt 1: Bezeichner und Datentypen

Der Händler braucht für die Kasse eine Funktion `sale_price`, die als Parameter

- `article : Article`, die Ware, und
- `discount : int`, den Rabattsatz (in Prozent zwischen 0 und 100)

erwartet und den Verkaufspreis `:int` (in Cent) berechnet.

Beispiel Angebotspreis

Schritt 2: Funktionsgerüst

```
def sale_price (
        article : Article,
        discount : int) -> int:
    # fill in
    return 0
```

- Neu: im Rumpf können wir die Attribute von `article` über die Punktnotation verwenden.

Beispiel Angebotspreis

Schritt 3: Beispiele

```
a1 = Article ("Mausefalle", 2000)
a2 = Article ("Promo␣Lutscher", 0)
a3 = Article ("Nougat", 2000)
sale_price (a1, 25) == 1500
sale_price (a1, 10) == 1900
sale_price (a3, 10) == 1900
sale_price (a2, 25) == 0
```

Beispiel Angebotspreis

Schritt 4: Funktionsdefinition

```
def sale_price (
        article : Article,
        discount : int) -> int:
    return article.price * (
        100 - discount) // 100
```

Bemerkung

Die Funktion funktioniert für jedes Objekt mit einem `price` Attribut.

1 Objekte und Klassen

- Objekte
- Identität und Gleichheit
- Klassen für Records
- Klassendefinition
- Instanzenerzeugung
- Funktionen auf Records
- Geschachtelte Records
- Objekte anzeigen
- Entwurf mit Alternativen

Beispiel

Terminplanung

Ein (Besprechungs-) Termin hat einen Titel, Teilnehmer, eine Anfangszeit und eine Endzeit. Eine (Uhr-) Zeit wird durch Stunde und Minute repräsentiert.

1. Wie lange dauert ein Termin?
2. Stehen zwei Termine in Konflikt?

Bemerkungen

- Zwei Klassen beteiligt: für Termin und für Zeit
- Frage 2 muss noch präzisiert werden

Terminplanung

Schritt 1: Bezeichner und Datentypen

Eine Zeit `Time` besteht aus

- einer Stundenzahl: `hour : int` zwischen 0 und 23
- einer Minutenzahl: `minute : int` zwischen 0 und 59.

Ein Termin `Appointment` hat

- einen Titel: `title : string`
- (mehrere) Teilnehmer: `participants : list` (of string)
- Anfangszeit: `start : Time`
- Endzeit: `end : Time` nicht vor `start`

Bemerkung

- Ein `Appointment`-Objekt enthält zwei `Time`-Objekte

Terminplanung

Schritt 2: Klassengerüst

```
class Time:
    def __init__(self, hour:int, minute:int):
        self.hour = hour
        self.minute = minute

class Appointment:
    def __init__(self, title:string
                     , partipants:list
                     , start:Time
                     , end:Time):
        self.title = title
        self.participants = participants
        self.start = start
        self.end = end
```

Funktionen der Terminplanung

Schritt 1: Bezeichner und Datentypen

Wie lange dauert ein Termin?

Die Funktion `duration` nimmt einen Termin `app : Appointment` und bestimmt seine Dauer in Minuten (`int`).

Schritt 2: Funktionsgerüst

```
def duration (app : Appointment) -> int:
    # fill in
    return 0
```

Funktionen der Terminplanung

Schritt 3: Beispiele

```
t1 = Time (12, 50)
t2 = Time (13, 10)
t3 = Time (10, 05)
t4 = Time (12, 45)
m1 = Appointment ("lunch", [], t1, t2)
m2 = Appointment ("lecture", [], t3, t4)
m3 = Appointment ("alarm", [], t4, t4)
duration(m1) == 20
duration(m2) == 160
duration(m3) == 0
```

Funktionen der Terminplanung

Schritt 4: Funktionsdefinition

```
def duration (app : Appointment) -> int:
    return difference (app.end, app.start)
```

Prinzip Wunschdenken

- Zur Erledigung der Aufgabe in `Appointment` benötigen wir eine Operation, die nur mit `Time` zu tun hat.
- **Wunschdenken** heißt, wir geben der gewünschten Funktion einen Namen und erstellen einen Vertrag für sie.
- Dann verwenden wir sie, bevor sie entworfen und implementiert ist.

Differenz von Zeiten

Schritt 1: Bezeichner und Datentypen

Bestimme die Differenz zweier Zeitangaben.
Die Funktion `difference` nimmt zwei Zeitangaben `t1, t2 : Time` und bestimmt die Differenz `t1 - t2` in Minuten (`int`). Dabei nehmen wir an, dass `t1 >= t2` ist.

Schritt 2: Funktionsgerüst

```
def difference (t1 : Time, t2 : Time) -> int:
    # fill in
    return 0
```

Differenz von Zeiten

Schritt 3: Beispiele

```
t1 = Time (12, 50)
t2 = Time (13, 10)
t3 = Time (10, 05)
t4 = Time (12, 45)
difference(t2, t1) == 20
difference(t4, t3) == 160
difference(t1, t1) == 0
```

Differenz von Zeiten

Schritt 4: Funktionsdefinition

```
def difference (t1 : Time, t2 : Time) -> int:
    return ((t1.hour - t2.hour) * 60
            + t1.minute - t2.minute)
```

In der Regel

- In Funktionen die Punktnotation nur zum Zugriff auf direkte Attribute verwenden.
- Also nicht tiefer als eine Ebene zugreifen.

Terminkonflikt

Präzisierung der Fragestellung

Stehen zwei Termine in Konflikt?

- Überschneiden sich zwei Termine zeitlich?
- Haben zwei Termine gemeinsame Teilnehmer?
- Konflikt, falls beides zutrifft!

Schritt 1: Bezeichner und Datentypen

Stehen zwei Termine in Konflikt?
Die Funktion `conflict` nimmt zwei Termine `a1, a2 : Appointment` und stellt fest, ob sie in Konflikt stehen (`bool`).

Terminkonflikt

Schritt 2: Funktionsgerüst

```
def conflict (a1 : Appointment,
              a2 : Appointment) -> bool:
    # fill in
    return False
```

Terminkonflikt

Schritt 3: Beispiele

```
t1 = Time (12, 00)
t2 = Time (12, 30)
t3 = Time (10, 05)
t4 = Time (12, 45)
ap = Appointment
a1 = ap ("lunch", ["jim", "jack"], t1, t2)
a2 = ap ("lecture", ["jeff", "jim"], t3, t4)
a3 = ap ("coffee", ["jack", "jill"], t2, t4)
#
conflict(a1, a2) and conflict (a2, a1)
not conflict(a1, a3)
not conflict(a2, a3)
```

Terminkonflikt

Schritt 4: Funktionsdefinition

```
def conflict (a1 : Appointment,
              a2 : Appointment) -> bool:
    time_ok = (before(a1.end, a2.start)
               or before(a2.end, a1.start))
    participants_ok = not (
        intersection (a1.partipants,
                      a2.partipants))
    return not (time_ok and partipants_ok)
```

Terminkonflikt

Wunschdenken

```
def before (t1 : Time, t2 : Time) -> bool:
    ''' check whether t1 is no later than t2 '''
    return False

def intersection (lst1 : list,
                  lst2 : list) -> list:
    ''' return the list of elements both in lst1 and lst
    return []
```


Weitere Ausführung selbst

- `before`: Bedingung auf den Attributen von `Time`-Objekten
- `intersection`: `for`-Schleife auf einer der Listen, Akkumulator für das Ergebnis

1 Objekte und Klassen

- Objekte
- Identität und Gleichheit
- Klassen für Records
- Klassendefinition
- Instanzenerzeugung
- Funktionen auf Records
- Geschachtelte Records
- Objekte anzeigen
- Entwurf mit Alternativen

Objekte anzeigen

Python-Interpreter

```
>>> class Article:
...     def __init__(self, name, price):
...         self.name = name
...         self.price = price
...
>>> phone = Article("Smartphone", 49500)
>>> phone
<__main__.Article object at 0x101ac51d0>
```

- Objekte werden nicht automatisch schön angezeigt.
- Absicht, damit ein Objekt “interne” Information enthalten kann.
- Ansatz: Schreibe eigene Druckfunktion

Objekte anzeigen

- Ein `Article` Objekt soll angezeigt werden.
- Dafür muss es in einen String umgewandelt werden.

Schritt 1: Bezeichner und Datentypen

Die Funktion `article_str` wandelt ein Objekt vom Typ `Article` in einen (informativen) String um.

Schritt 2: Funktionsgerüst

```
def article_str(art : Article):
    # fill in
    return ""
```

Objekte anzeigen

Schritt 3: Beispiele

```
a1 = Article("Phone", 49500)
a2 = Article("Hammer", 1300)
#
article_str (a1) == "Article('Phone', 49500)"
article_str (a2) == "Article('Hammer', 1300)"
```

Objekte anzeigen

Schritt 4: Funktionsdefinition

```
def article_str (art : Article):
    return ("Article(" + repr(art.name)
                  + ",␣" + repr(art.price) + ")")
```

- Die Funktion `repr(x)` erzeugt einen String aus dem Objekt `x`, sodass dieser String wieder von Python eingelesen werden kann.
- Für Zahlen liefern `str` und `repr` in der Regel das gleiche Ergebnis.

str und repr

Python-Interpreter

```
>>> string = "dead parrot"
>>> string
'dead parrot'
>>> print(string)
dead parrot
>>> str(string)
'dead parrot'
>>> repr(string)
"'dead parrot'"
```

Objekte bequemer anzeigen

Python-Interpreter

```
>>> Article.__str__ = article_str
>>> a1 = Article("Phone", 49500)
>>> str (a1)
'Article(’Phone’, 49500)'
>>> print(a1)
Article(’Phone’, 49500)
```

```
Article.__str__ = article_str
```

- Eine Klasse ist auch ein Objekt!
- Das `__str__` Attribut der Klasse ist “magisch”.
- Wenn es eine passende Funktion enthält, wird sie beim Umwandeln von Objekten der Klasse in einen String aufgerufen.

1 Objekte und Klassen

- Objekte
- Identität und Gleichheit
- Klassen für Records
- Klassendefinition
- Instanzenerzeugung
- Funktionen auf Records
- Geschachtelte Records
- Objekte anzeigen
- Entwurf mit Alternativen

Entwurf mit Alternativen

Spielkarten

Eine Spielkarte ist entweder

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

Schritt 1: Bezeichner und Datentypen

Eine Spielkarte hat eine von zwei Ausprägungen.

- Joker werden durch Objekte der Klasse `Joker` repräsentiert.
- Natürliche Karten durch Objekte der Klasse `Card` mit Attributen `suit` (Farbe) und `rank` (Wert).

Farbe ist ***C**lubs*, ***S**pades*, ***H**earts*, ***D**iamonds*
Wert ist 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, Kind, Ace

Entwurf mit Alternativen

Schritt 2: Klassengerüst

```
class Joker:
    pass # no attributes

class Card:
    def __init__(self, suit:string
                       , rank:string):
        self.suit = suit
        self.rank = rank
```

Beispiel: Rommé

Rommé Figuren erkennen

Ein Figur im Rommé ist entweder

- ein Satz (*set*): drei oder vier Karten gleichen Werts in verschiedenen Farben,
- eine Reihe (*run*): drei oder mehr Karten der gleichen Farbe mit aufsteigenden Werten

Eine Karte in einer Figur kann durch einen Joker ersetzt werden. Allerdings müssen mindestens zwei natürliche Karten vorhanden sein.
Erste Aufgabe: Erkenne einen Satz

Schritt 1: Bezeichner und Datentypen

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

Satz erkennen

Schritt 2: Funktionsgerüst

```
def is_set (cards):
    # initialization of acc
    for card in cards:
        # action on single card
    # finalization
    return True
```

- Liste `cards` verarbeiten: `for` Schleife mit Akkumulator
- Länge der Liste prüfen
- Anzahl der Joker prüfen
- auf gleichen Wert prüfen
- auf verschiedene Farbe prüfen

Satz erkennen

Schritt 3: Beispiele

```
c1 = Card ('C', 'Queen')
c2 = Card ('H', 'Queen')
c3 = Card ('S', 'Queen')
c4 = Card ('D', 'Queen')
c5 = Card ('D', 'King')
j1 = Joker ()

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

Satz erkennen

Schritt 4: Funktionsdefinition

```
def is_set (cards):
    if len (cards) < 3 or len (cards) > 4:
        return False
    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 rank and rank != card.rank:
                return False
            else:
                rank = card.rank
            if card.suit in suits:
                return False # repeated suit
            else:
                suits = suits + [card.suit]
    return nr_jokers <= len (cards) - 2
```

Satz erkennen / Klassentest

Schritt 4: Funktionsdefinition (Wunschdenken)

```
def is_joker (card):
    return type(card) is Joker
```

- Klassentest
- `type(x)` liefert immer das Klassenobjekt zum Wert in `x`
- Das Klassenobjekt ist eindeutig, daher kann es mit `is` verglichen werden.
- Verwendung im Gerüst, immer wenn ein Argument zu verschiedenen Klassen gehören kann.

Reihe erkennen

Schritt 1: Bezeichner und Datentypen

Die Funktion `is_run` nimmt als Eingabe eine Liste `cards` von Spielkarten und liefert `True` gdw `cards` eine Reihe ist.

Reihe erkennen

Schritt 2: Funktionsgerüst

```
def is_run (cards):
    # initialization of acc
    for card in cards:
        # action on single card
    # finalization
    return True
```

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

Reihe erkennen

Schritt 3: Beispiele

```
cq = Card ('C', 'Queen')
ck = Card ('C', 'King')
sa = Card ('S', 'Ace')
dq = Card ('D', 'Queen')
d10 = Card ('D', '10')
jj = Joker ()

not is_run ([cq, ck])
is_run ([cq, ck, sa])
is_run ([dq,ck,sa])
is_run ([d10,jj,dq])
is_run ([d10,jj,dq,ck])
not is_run ([s10, dq, ck])
not is_run ([d10, jj, jj])
```

Reihe erkennen

Schritt 3: Funktionsdefinition

```
def is_run (cards):
    if len (cards) < 3:      # check length of list
        return False
    else:
        # initialization of acc
        nr_jokers = 0        # count jokers
        current_rank = None # keep track of rank
        suit = None
    for card in cards:
        if current_rank:
            current_rank = next_rank (current_rank)
        # action on single card
        if is_joker (card):
            nr_jokers = nr_jokers + 1
        else:
            if not current_rank:
                current_rank = card.rank
            elif current_rank != card.rank:
                return False
            if not suit:
                suit = card.suit
            elif suit != card.suit:
                return False
    # finalization
    return nr_jokers <= len (cards) - 2
```

Reihe erkennen

Was noch fehlt …

- Wunschdenken: `next_rank`
- Maximalzahl von Jokern in einer Reihe?
- Maximalzahl von Jokern nebeneinander?
- Joker außerhalb der Reihe…

2 Zusammenfassung & Ausblick

Zusammenfassung & Ausblick

- Alle Werte in Python sind Objekte.
- Veränderliche Objekte besitzen eine Identität.
- Eine Klasse beschreibt Objekte/Instanzen.
- Ein Record ist ein Objekt, das untergeordnete Objekte enthält.
- Funktionsentwurf mit einfachen Records.
- Funktionsentwurf mit geschachtelten Records.
- Objekte anzeigen; das `__str__` Attribut.
- Entwurf mit Alternativen.
- Der Typtest geschieht durch Identitätstest gegen die Klasse.