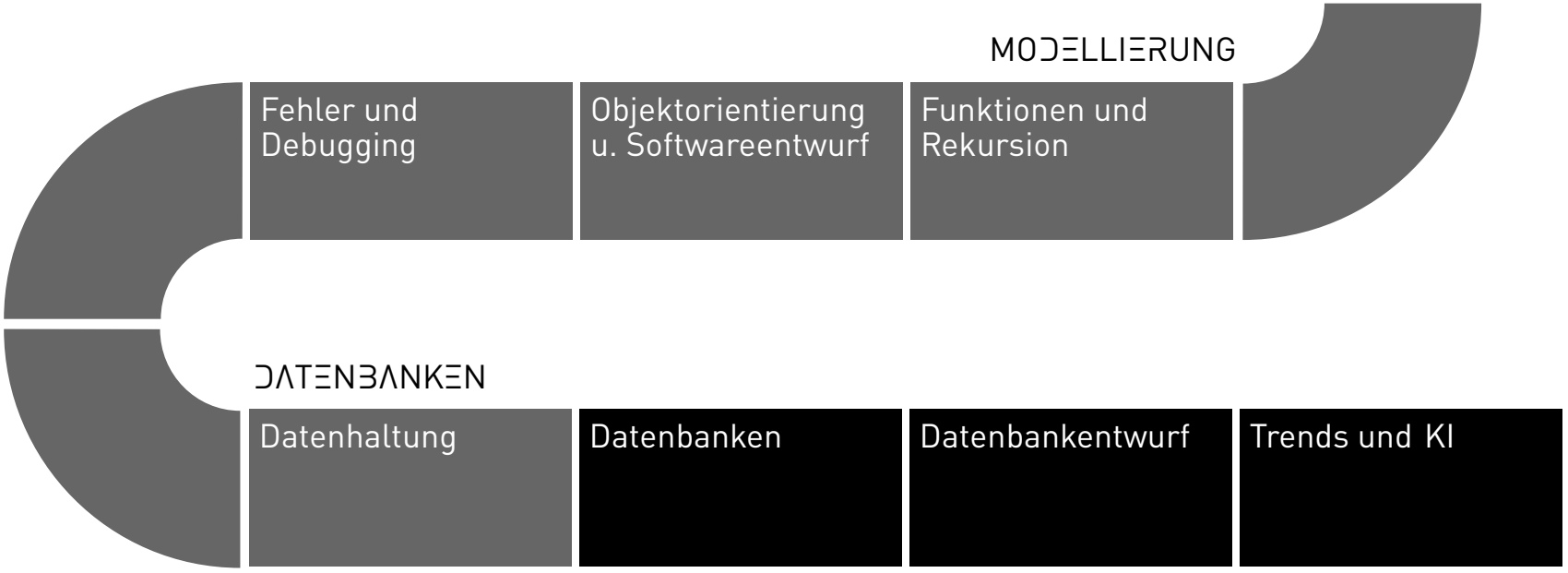
programmierung und datenbanken

Joern Ploennigs

Modularisierung

MIDJOURNEY: MODULAR BLOCKS, REF. PIET MONDRIAN

Ablauf


MODELLIERUNG
Fehler und Debugging
Objektorientierung u. Softwareentwurf
Funktionen und Rekursion
DATENBANKEN
Datenhaltung
Datenbanken
Datenbankentwurf
Trends und KI

Organisieren von Projekten über mehrere Dateien

Herausforderungen großer Programme:

- Dutzende Klassen mit hunderten Funktionen
- Viele tausende Zeilen Code
- Schwer Überblick zu behalten
- Versionskonflikte bei mehreren Programmierern

Lösung - Code aufteilen in .py Dateien:

- *Pro Klasse* eine Datei (bei Klassendefinitionen)
- *Pro Thema* eine Datei (z.B. Mathematik-Funktionen)
- *Pro Aufgabenbereich* eine Datei (z.B. Laden vs. Verarbeitung)

Die main()-Funktion als Standardfunktion

Woher weiß Python was ausgeführt werden soll?

- Eine bestimmte Datei als *Zentrum des Projektes* deklarieren
- Diese enthält eine spezielle Funktion namens `main()`
- Existiert in fast jeder Programmiersprache
- Gibt den *Startpunkt des Programms* an
- Kann keine oder dynamische Argumente erhalten (Kommandozeilenargumente)

Die main()-Funktion in Python

Verschiedene Ausführungsmöglichkeiten:

1. Als Skript in Kommandozeile/Gesamtprogramm
2. Importiert in interaktive Python-Konsole
3. Importiert in andere Python-Datei

Problem: In Fall 1 wollen wir das gesamte Skript, in Fall 2+3 nur Teile nutzen
Die Lösung - Variable `__name__`:

```python
def main():
    print("This is the main function")

if __name__ == "__main__":
    main()  # nur bei direktem Aufruf
```

Best Practice für main()-Funktion

Was gehört VOR die main()-Definition:

✅ Nur erlaubt:

- Funktionsdefinitionen
- Klassendefinitionen

Vorteile dieser Struktur:

- Programmfluss wird übersichtlicher
- Programm einfach modifizierbar
- Programm besser weiterverwendbar

❌ Vermeiden:

- Variablenbelegungen (globale Variablen)
- Funktionsaufrufe (Nebeneffekte)

Programmfluss über mehrere Dateien

- Jetzt haben wir einen zentralen Startpunkt.
- Wie rufen wir Code aus anderen Dateien auf?
- Jede Programmiersprache hat Befehle zum Laden externen Codes:
  - `import` und `include` (Python)
  - Spezielle "Header-Dateien" die Zusammenhänge zwischen Dateien definieren

Programmfluss in Python - import Statement

Strategie:

- Eine .py-Datei enthält `main()` (Fall 1)
- Alle anderen ohne `main()` oder mit `__name__` ignoriert (Fall 3)
- `import`-Statement lädt externe Dateien
- *Achtung:* Kompletter Code wird ausgeführt, auch Variablen und Funktionsaufrufe!

```python
import external_file

def main():
    print("This is the main function")
```

Variablen/Funktionen/Klassen aus `external_file.py` werden in Objekt `external_file` abgelegt

Import externer Bibliotheken (Packages)

Terminologie:

- *Module:* Importierte Skripte
- *Namespace:* Das entstehende Objekt (Typ: module)
- *Package:* Modul mit weiteren Untermodulen

Grundlagen der Projektstruktur

Allgemeine Struktur

```
main.py [enthält main()]
├── module1.py
├── module2.py
└── helpers.py
```

Code in `main.py`

```python
import module1
import module2
import helpers
```

Beispiel - Stadtprojekt

```
city.py
├── buildings.py
├── streets.py
└── geometry.py
```

Code in `main.py`

```python
import buildings
import streets
import geometry
```

Das import-Statement: Erweiterte Nutzungen

- Die Import-Funktionen von Python sind komplex und vielschichtig.
- Im Alltag trifft man aber meistens nur auf folgende zusätzliche Konstrukte:
  - `from-import`-Statement: Importiert ein Submodul oder ein Teil eines Moduls direkt, ohne den übergeordneten Namespace
  - `from-import-as`-Statement: Arbeitet genauso, benennt aber das importierte Objekt um

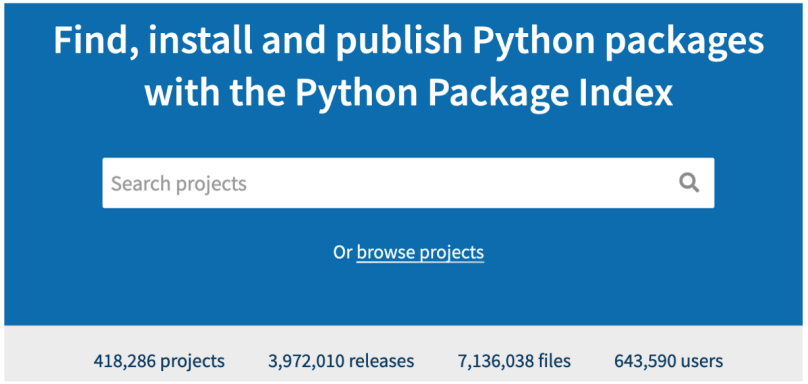
```
from external_file import external_function as ext_func
```

Import-Statements – Das Tor zur Welt

- Wir können nicht nur unsere eigenen Pakete importieren – sondern auch Pakete die von anderen Personen/Organisationen öffentlich gemacht wurden!
- Es gibt mehrere Möglichkeiten auf Pakete zuzugreifen:
- Paket ist in Python mitgeliefert
- Direkt als .py-Dateien / Ordner mit .py Dateien herunterladen
- Einen Package-Manager (pip, uv) benutzen (Besser!)

Packages - Die eigentliche Macht von Python

- Pythons Platzierung als populärste Programmiersprache der Welt basiert vor allem darauf, dass es sich zu einer Interface- und Pipeline-Sprache entwickelt hat.
- Fast jedes durch einen Computer lösbares Problem kann durch das geschickte Verschalten der richtigen Python-Packages gelöst werden.
- Die offizielle Anlaufstelle für Packages, der Python Package Index (PyPI), führt derzeit über 418.000 frei verfügbare Pakete.
- Diese können in den meisten Fällen mit einem einzigen Befehl/Klick installiert werden



Lesson Learned

Wie priotisiert man richtig?

“I have two kinds of problems, the urgent and the important. The urgent are not important, and the important are never urgent.”, Eisenhower

Lesson Learned - Priorisierung

- Beim Eisenhauer-Schreibtisch unterteilt man in dringend und wichtig
- Man bearbeitet zuerst die dringenden, wichtigen

fragen?

PROGRAMMIERUNG UND DATENBANKEN