Programmieren I (Python)

Christian Osendorfer

2023-10-06

Themen

- Wissen in **Datenstrukturen** darstellen
- **Iteration** und **Rekursion** als grundlegende **Berechnungsparadigmen**
- **Abstraktion** von Datentypen und Prozeduren
- **Organisation** und **Modularisierung** von Systemen mittels **Klassen** und **Methoden**
- **Algorithmen** (Suchen, Sortieren)
- (**Komplexität** von Algorithmen)

Was macht ein Computer ?

- Führt (einfache) Berechnungen aus
  - 1 Milliarde mal pro Sekunde!
- Speichert Ergebnisse
  - >> 1 Terabyte Speichervolumen
- Welche Berechnungen?
  - **built-in** (atomare Operationen)
  - definiert durch eine Programmiererin
- Ein Computer weiß nur das, was wir ihm sagen

Arten von Wissen

- **Deklaratives** Wissen
  - Sachwissen / “Knowing What”
    - *Mt. Kilimanjaro ist 5895m hoch*
- **Imperatives** Wissen
  - Anleitung / Rezept / “Knowing How”
    - *fliege nach Tanzania, fahre zum Basislager, …*

Mathematisches Beispiel

- **Wurzel** einer reellen positiven Zahl x ist y mit $y \times y = x$
- Rezept um **Wurzel von** x zu berechnen
  1. Rate eine Zahl g zwischen 0 und x
  2. Falls $g \times g$ **nahe genug** an x, dann beende den Vorgang, g ist die Lösung
  3. Andernfalls, berechne ein neues g' indem g und x/g gemittelt werden
  4. Gehe zu 2.

Numerisches Beispiel

Beispiel für $x = 16$

g	$g \times g$	x/g	$(g + x/g)/2$
3	9	16/3	4.17
4.17	123	3.837	4.0035
4.0035	16.0277	3.997	4.000002

Was ist ein *Rezept*?

- Eine Sequenz (einfacher) Berechnungsschritte
- Kontrollflußvorgaben, die bestimmen, wann ein Schritt ausgeführt wird
- Eine Möglichkeit die Sequenz von Schritten zu beenden

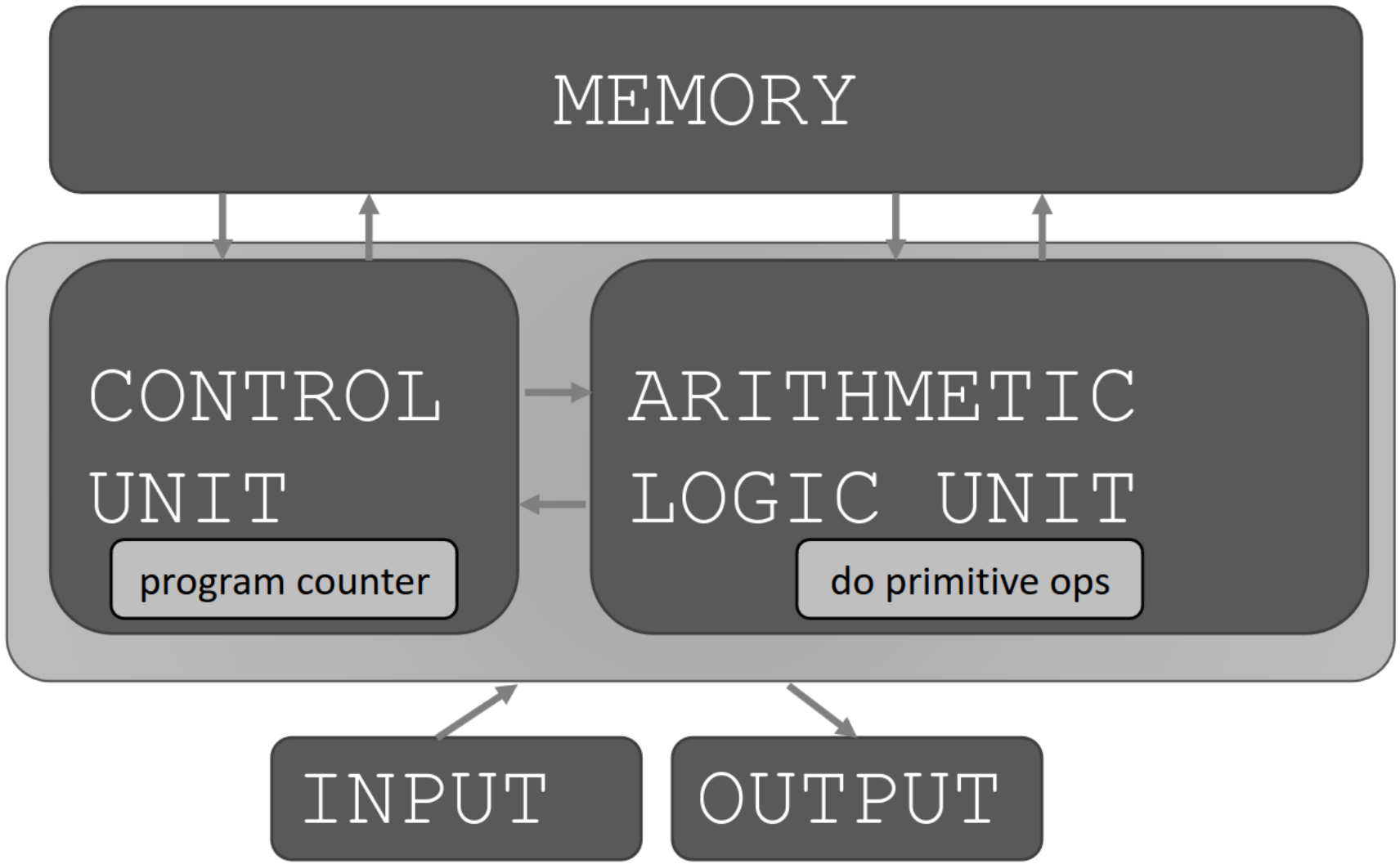
Algorithmus

Der Computer als eine Maschine

Wie bildet man einen Algorithmus auf einen mechanischen Prozeß ab?

- **Fixed Program Computer**
  - Taschenrechner
- **Stored Program Computer**
  - Eine Maschine speichert und führt Berechnungsschritte aus

Grundlegende Computerarchitektur



Ein (Von-Neumann) Computer

Stored Program Computer

- Sequenz von Befehlen, die im Rechner (Computer) gespeichert sind.
  - vordefinierte Menge an primitiven Befehlen/Operationen
    - Arithmetik und Logik
    - Einfache Tests
    - Verschiebebefehle
- Ein spezielles Program (**Interpreter** auf der *Control Unit*) führt jeden Befehl der Reihe nach aus
  - **Tests** werden benutzt um die Reihung der Befehle zu steuern
  - Beendet Ausführung

Primitive Operationen

- Alan Turing hat gezeigt dass man alles(?) berechnen kann mit nur 6 grundlegenden Operationen
- Moderne Programmiersprachen haben eine größere Anzahl an primitive Operationen
- Mittels Abstraktion kann man selber neue Operationen schaffen
- Programmiersprachen sind äquivalent.

Programmiersprachen

- Eine Programmiersprache stellt eine Menge an primitiven Operationen zur Verfügung
- **Expressions** (Ausdrücke) sind komplexe aber legale Kombinationen von (primitiven) Operationen
- **Statements** (Anweisungen) geben dem Rechner vor, etwas zu tun
- Expressions haben **values** (Werte)
- Statements haben eine **Semantik** (Bedeutung)

Aspekte einer Programmiersprache

- **Syntax**
  - *boy hugs cat / boy cat hugs*
- **Statische Semantik**
  - welche syntaktisch korrekten Zeichenketten haben eine Bedeutung
    - *I are hungry*
- **Semantik**
  - Die Bedeutung einer syntaktisch korrekten Zeichenkette ohne statischer semantischer Fehler
    - In Programmiersprachen gibt es immer genaue eine Bedeutung

Python Programme

- Ein Program ist eine Sequenz an **Definitionen** und **Statements**
  - Definitionen werden *evaluiert*
  - Statements werden vom Python Interpreter *ausgeführt*
- Programme manipulieren **Datenobjekte**
  - Datenobjekte haben einen **Typ** (type) der festlegt, welche Operationen ein Program mit ihnen machen kann
    - *Dodo ist ein Papagei, kann also fliegen, fressen, sprechen*
- Datenobjekte sind
  - skalare (oder: atomare/elementare) Datentypen
  - nicht-skalare Datentypen (mit interner Struktur)

Ausführung eines Python Programs

- Python-Programme werden sowohl **übersetzt** als auch **interpretiert**
- Bei Programmstart wird der Python-Code in *Byte-Code* **übersetzt**
  - Bytecode: Befehle für eine **virtuelle Maschine** (*PVM – Python Virtual Machine*)
  - Code unabhängig von realer Hardware
- Bytecode wird dann für einen bestimmten Maschinencode **interpretiert**

Ausführung eines Python Programs

Python Compile/Interpret

- Die **Python-Shell** ist eine REPL (Read, Evaluate, Print, Loop).
  - `python` oder `ipython`
    - (`pip install ipython`)

Skalare Objekttypen (in Python)

- `int`: ganze Zahlen (integer)
- `float`: reelle Zahlen (Fließkommazahlen)
- `bool`: boolsche Werte (`True` oder `False`)
- `NoneType`: spezieller Typ mit genau einem Wert: `None`
- `type()` ermittelt Objekttype eines Wertes: `type(4)`

Expression

- Kombiniere Datenobjekte und (primitive) Operatoren um eine Expression zu erzeugen
- Eine Expression hat einen Wert (value), der wiederum einen Typ hat
- `x+y` / `x-y` / `x*y` / `x/y` / `x%y` / `x**y`
  - falls `x` und `y` vom Typ `int` oder `float` sind
- **Call Expression** (*Funktionsaufruf*)
  - Operator kann auch eine Funktion sein
  - `add(3, 4)`

Namen

- Ein **Name** kann mit einem Wert verknüpft (bind) sein.
- Eine Möglichkeit einen Namen mit etwas zu verknüpfen ist durch ein **assignment statement**
  - `x = 7`
    - `x` ist der Name, die *Zahl* `7` der Wert
  - `x = 1 + 3/2 - 42`
  - `x = add(3, 4)`
  - `x = add(x, 3)`
    - Geht soetwas? Immer?

Variable

- Ein Name, der mit einem Wert verknüpft ist, heisst auch **Variable**
- Eine Variable kann erst verwendet werden, nachdem ein Wert mit dem Namen verknüpft wurde!
- Die Form eines Namen ist eingeschränkt (z.B. darf es kein **keyword** sein), siehe hier

Funktionen

- Eine Funktion ist eine Sequenz von Anweisungen die eine bestimmte Aufgabe erfüllt und einfach wiederverwendet werden kann.
- `add(3, 4)`
- Eine Funktion hat Eingaben (**arguments**) und eine(!) Ausgabe (**return value**)
- Ein Funktionsaufruf wendet den **Wert seines Operators** (*die Funktion*) auf die **Werte der Eingaben** (*Operanden*) an.

Funktionsdefinition

- In der Regel werden Funktionen in Python mit dem **`def` statement** definiert
- ```
  def <name>(<parameters>):
      return <return-expression>
  ```
- Zum Beispiel eine ganz besondere `add` Funktion
- ```
  def my_add(num1, num2):
      # Eine ganz besondere Funktion!
      # <- Das Kommentarsymbol
      x = num2*num2 # Einrückung ist in Python elementar !
      return num1 + num2 + x
  ```

Anatomie einer Funktionsdefinition

- Die erste Zeile einer Funktionsdefinition heisst **Signatur der Funktion** (**function signature**)
- Der Rest einer Funktionsdefinition ist der **Funktionskörper** (**function body**).
-

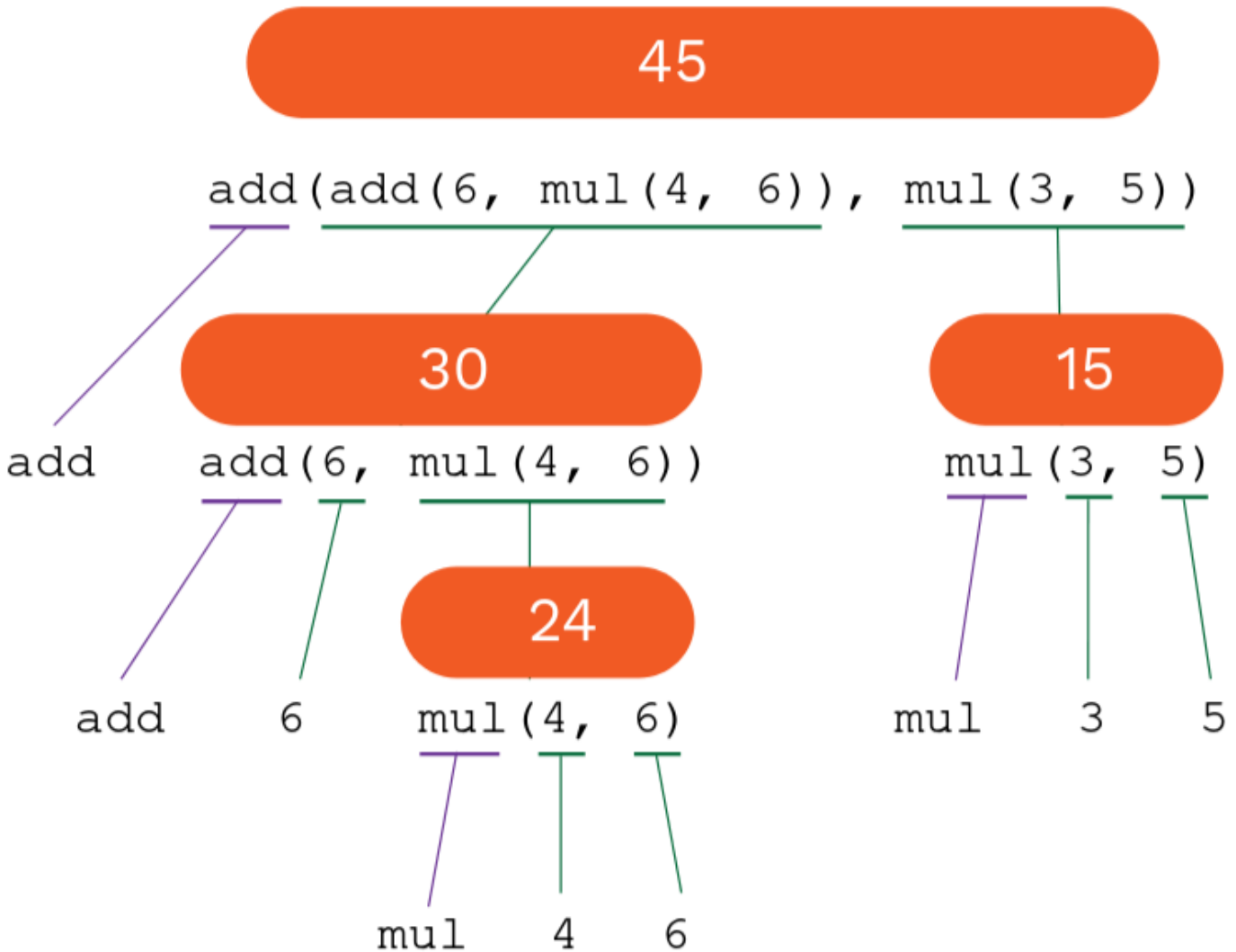
```
1   def <name>(<parameters>): # function signature: name and 'set of parameters'
2       statement1            # function body from here on
3       statement2
4       ...
5       statement_n
6       return <return-expression>
```

Ein erstes Python Program

```python
def my_add(num1, num2):
    x = num2 * num2
    return num1 + num2 + x

# Ausgabe auf die Konsole mittels 'print'
print("Enter first number:") # 'print' ist ein Funktionsaufruf! return value??
x = input() # Einlesen von der Konsole. return value ist eine Zeichenkette (string)!
x = int(x) # Typkonversion: Zeichenkette -> integer

print("Enter second number:")
y = input()
y = int(y)

result = my_add(x, y)
print("Result:", result)
```

- Ausführung mittels `python first_program.py` oder `python -i first_program.py`

Verschachtelte Funktionsaufrufe



Ein sogenannter **Expression Tree**

Zusammenfassung

- Ein Computerprogram besteht aus **statements** (Anweisungen an den Rechner), die **expressiones** (Ausdrücke) enthalten.
- Expressions beschreiben Berechnungen und evaluieren zu **values** (Werte).
- Werte *können* an **Namen** zugewiesen werden, um Wiederholung von Berechnungen zu vermeiden (**Variable**).
- Funktionen fassen eine Reihe von statements zusammen, die Eingaben auf *eine* Ausgabe **abbilden** (map).
- Ein `def` **statement** erzeugt ein **Funktionsobjekt** mit gewissen **Parametern** und einem **Funktionskörper** und stellt diesen Namen zur Verwendung in einem Program zur Verfügung.
- `print()`, `input()`, `int()`, `# Kommentar`, Einrückung in Python, REPL