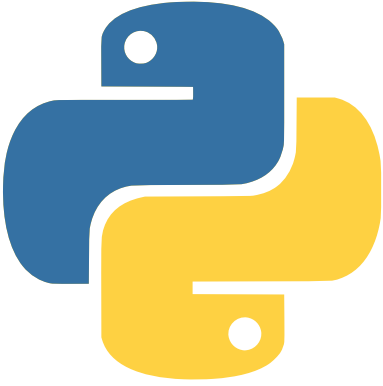
Universität Stuttgart
Projekt digit@L – BOOST. SKILLS. SUPPORT.



Programmierkurs Python

Der Datentyp float für Gleitkomma-Zahlen

Dominik Göddeke

IANS SimTech

Stiftung Innovation in der Hochschullehre

Der Datentyp float für Gleitkomma-Zahlen

Motivation

- Computer kann nur **endliche Anzahl** von Bits (0/1) speichern

Motivation

- Computer kann nur **endliche Anzahl** von Bits (0/1) speichern
- Kein Problem für ganze Zahlen, immer exakt binär darstellbar

Motivation

- Computer kann nur **endliche Anzahl** von Bits (0/1) speichern
- Kein Problem für ganze Zahlen, immer exakt binär darstellbar
  - Mehr in Kürze

Motivation

- Computer kann nur **endliche Anzahl** von Bits (0/1) speichern
- Kein Problem für ganze Zahlen, immer exakt binär darstellbar
  - Mehr in Kürze
  - In Python: Größe nur durch zur Verfügung stehenden Speicherplatz limitiert

Motivation

- Computer kann nur **endliche Anzahl** von Bits (0/1) speichern
- Kein Problem für ganze Zahlen, immer exakt binär darstellbar
  - Mehr in Kürze
  - In Python: Größe nur durch zur Verfügung stehenden Speicherplatz limitiert
- Rationale Zahlen wie $1/3 = 0.333333333333333\ldots$

Motivation

- Computer kann nur **endliche Anzahl** von Bits (0/1) speichern
- Kein Problem für ganze Zahlen, immer exakt binär darstellbar
  - Mehr in Kürze
  - In Python: Größe nur durch zur Verfügung stehenden Speicherplatz limitiert
- Rationale Zahlen wie $1/3 = 0.333333333333333\ldots$
  - Repräsentation als Bruch, Tupel von Zähler und Nenner, jeweils ganze Zahl

Motivation

- Computer kann nur **endliche Anzahl** von Bits (0/1) speichern
- Kein Problem für ganze Zahlen, immer exakt binär darstellbar
  - Mehr in Kürze
  - In Python: Größe nur durch zur Verfügung stehenden Speicherplatz limitiert
- Rationale Zahlen wie $1/3 = 0.333333333333333\ldots$
  - Repräsentation als Bruch, Tupel von Zähler und Nenner, jeweils ganze Zahl
  - Prinzipiell möglich, aber ineffizient $\rightarrow$ gleich

Motivation

- Computer kann nur **endliche Anzahl** von Bits (0/1) speichern
- Kein Problem für ganze Zahlen, immer exakt binär darstellbar
  - Mehr in Kürze
  - In Python: Größe nur durch zur Verfügung stehenden Speicherplatz limitiert
- Rationale Zahlen wie $1/3 = 0.333333333333333\ldots$
  - Repräsentation als Bruch, Tupel von Zähler und Nenner, jeweils ganze Zahl
  - Prinzipiell möglich, aber ineffizient $\rightarrow$ gleich
- Irrationale Zahlen wie π oder e

Motivation

- Computer kann nur **endliche Anzahl** von Bits (0/1) speichern
- Kein Problem für ganze Zahlen, immer exakt binär darstellbar
  - Mehr in Kürze
  - In Python: Größe nur durch zur Verfügung stehenden Speicherplatz limitiert
- Rationale Zahlen wie $1/3 = 0.333333333333333\ldots$
  - Repräsentation als Bruch, Tupel von Zähler und Nenner, jeweils ganze Zahl
  - Prinzipiell möglich, aber ineffizient $\rightarrow$ gleich
- Irrationale Zahlen wie π oder e
  - Grundsätzlich keine exakte Repräsentation mit endlich vielen Nachkommastellen

Motivation

- Computer kann nur **endliche Anzahl** von Bits (0/1) speichern
- Kein Problem für ganze Zahlen, immer exakt binär darstellbar
  - Mehr in Kürze
  - In Python: Größe nur durch zur Verfügung stehenden Speicherplatz limitiert
- Rationale Zahlen wie $1/3 = 0.333333333333333\ldots$
  - Repräsentation als Bruch, Tupel von Zähler und Nenner, jeweils ganze Zahl
  - Prinzipiell möglich, aber ineffizient $\rightarrow$ gleich
- Irrationale Zahlen wie π oder e
  - Grundsätzlich keine exakte Repräsentation mit endlich vielen Nachkommastellen
  - Vermutlich bekannt aus der Schulmathematik

Motivation

- Computer kann nur **endliche Anzahl** von Bits (0/1) speichern
- Kein Problem für ganze Zahlen, immer exakt binär darstellbar
  - Mehr in Kürze
  - In Python: Größe nur durch zur Verfügung stehenden Speicherplatz limitiert
- Rationale Zahlen wie $1/3 = 0.333333333333333\ldots$
  - Repräsentation als Bruch, Tupel von Zähler und Nenner, jeweils ganze Zahl
  - Prinzipiell möglich, aber ineffizient $\rightarrow$ gleich
- Irrationale Zahlen wie π oder e
  - Grundsätzlich keine exakte Repräsentation mit endlich vielen Nachkommastellen
  - Vermutlich bekannt aus der Schulmathematik
- **Umgang mit solchen Zahlen?**

Exkurs
Zahldarstellung in verschiedenen Basen

Exkurs: Zahldarstellung in verschiedenen Basen

- Zahlen im **Zehnersystem** („zur Basis 10“)

$$1234 = 1234_{10} = 4 \cdot 10^0 + 3 \cdot 10^1 + 2 \cdot 10^2 + 1 \cdot 10^3$$

Exkurs: Zahldarstellung in verschiedenen Basen

- Zahlen im **Zehnersystem** („zur Basis 10“)

$$1234 = 1234_{10} = 4 \cdot 10^0 + 3 \cdot 10^1 + 2 \cdot 10^2 + 1 \cdot 10^3$$

- 4 „Nuller“, 3 „Zehner“, 2 „Hunderter“ und 1 „Tausender“

Exkurs: Zahldarstellung in verschiedenen Basen

- Zahlen im **Zehnersystem** („zur Basis 10“)

$$1234 = 1234_{10} = 4 \cdot 10^0 + 3 \cdot 10^1 + 2 \cdot 10^2 + 1 \cdot 10^3$$

- 4 „Nuller“, 3 „Zehner“, 2 „Hunderter“ und 1 „Tausender“
- Berücksichtigung des Vorzeichens

$$1234 = 1234_{10} = \mathsf{sign}(1234) \cdot 4 \cdot 10^0 + 3 \cdot 10^1 + 2 \cdot 10^2 + 1 \cdot 10^3$$

Exkurs: Zahldarstellung in verschiedenen Basen

- Zahlen im **Zehnersystem** („zur Basis 10“)

$$1234 = 1234_{10} = 4 \cdot 10^0 + 3 \cdot 10^1 + 2 \cdot 10^2 + 1 \cdot 10^3$$

- 4 „Nuller“, 3 „Zehner“, 2 „Hunderter“ und 1 „Tausender“
- Berücksichtigung des Vorzeichens

$$1234 = 1234_{10} = \mathrm{sign}(1234) \cdot 4 \cdot 10^0 + 3 \cdot 10^1 + 2 \cdot 10^2 + 1 \cdot 10^3$$

- Üblichen Definition des Vorzeichens

$$\mathrm{sign}(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Exkurs: Zahldarstellung in verschiedenen Basen

- Im Computer: **binäre Repräsentation**, d.h. zu Zweierpotenzen statt Zehnerpotenzen

$$
\begin{aligned}
1234_{10} &= 2 + 16 + 64 + 128 + 1024 && (1)\\
&= 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 && (2)\\
&\quad + 1 \cdot 2^6 + 1 \cdot 2^7 + 0 \cdot 2^8 + 0 \cdot 2^9 + 1 \cdot 2^{10} + 0 \cdot 2^{11} && (3)
\end{aligned}
$$

Exkurs: Zahldarstellung in verschiedenen Basen

- Im Computer: **binäre Repräsentation**, d.h. zu Zweierpotenzen statt Zehnerpotenzen

$$
\begin{aligned}
1234_{10} &= 2 + 16 + 64 + 128 + 1024 && (1)\\
&= 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 && (2)\\
&\quad + 1 \cdot 2^6 + 1 \cdot 2^7 + 0 \cdot 2^8 + 0 \cdot 2^9 + 1 \cdot 2^{10} + 0 \cdot 2^{11} && (3)
\end{aligned}
$$

- Vorteil: nur ein Bit pro Vorfaktor (Koeffizient) vor einer Potenz nötig, und Potenzen sind (wie im Zehnersystem) implizit

Exkurs: Zahldarstellung in verschiedenen Basen

- Im Computer: **binäre Repräsentation**, d.h. zu Zweierpotenzen statt Zehnerpotenzen

$$1234_{10} = 2 + 16 + 64 + 128 + 1024 \tag{1}$$
$$= 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 \tag{2}$$
$$+ 1 \cdot 2^6 + 1 \cdot 2^7 + 0 \cdot 2^8 + 0 \cdot 2^9 + 1 \cdot 2^{10} + 0 \cdot 2^{11} \tag{3}$$

- Vorteil: nur ein Bit pro Vorfaktor (Koeffizient) vor einer Potenz nötig, und Potenzen sind (wie im Zehnersystem) implizit
- Weniger Speicherbedarf geht nicht

Exkurs: Zahldarstellung in verschiedenen Basen

- Im Computer: **binäre Repräsentation**, d.h. zu Zweierpotenzen statt Zehnerpotenzen

$$
\begin{aligned}
1234_{10} &= 2 + 16 + 64 + 128 + 1024 && (1)\\
&= 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 && (2)\\
&\quad + 1 \cdot 2^6 + 1 \cdot 2^7 + 0 \cdot 2^8 + 0 \cdot 2^9 + 1 \cdot 2^{10} + 0 \cdot 2^{11} && (3)
\end{aligned}
$$

- Vorteil: nur ein Bit pro Vorfaktor (Koeffizient) vor einer Potenz nötig, und Potenzen sind (wie im Zehnersystem) implizit
- Weniger Speicherbedarf geht nicht
- Ablesen der Koeffizienten mit zunehmender Größe der zugehörigen Potenz liefert **Binärdarstellung**

$$1234_{10} = 010011010010_2 \quad (4)$$

Exkurs: Zahldarstellung in verschiedenen Basen

- Im Computer: **binäre Repräsentation**, d.h. zu Zweierpotenzen statt Zehnerpotenzen

$$
\begin{aligned}
1234_{10} &= 2 + 16 + 64 + 128 + 1024 && (1)\\
&= 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 && (2)\\
&\quad + 1 \cdot 2^6 + 1 \cdot 2^7 + 0 \cdot 2^8 + 0 \cdot 2^9 + 1 \cdot 2^{10} + 0 \cdot 2^{11} && (3)
\end{aligned}
$$

- Vorteil: nur ein Bit pro Vorfaktor (Koeffizient) vor einer Potenz nötig, und Potenzen sind (wie im Zehnersystem) implizit
- Weniger Speicherbedarf geht nicht
- Ablesen der Koeffizienten mit zunehmender Größe der zugehörigen Potenz liefert **Binärdarstellung**

$$1234_{10} = 010011010010_2 \quad (4)$$

- Vorzeichen, noch ein Bit, Willkür ob 1=minus und 0=plus oder umgekehrt

Exkurs: Zahldarstellung in verschiedenen Basen

- **Länge** der Repräsentation einer Zahl: Anzahl Bits (Anzahl Nullen und Einsen), plus Vorzeichen-Bit

Exkurs: Zahldarstellung in verschiedenen Basen

- **Länge** der Repräsentation einer Zahl: Anzahl Bits (Anzahl Nullen und Einsen), plus Vorzeichen-Bit
- Analog für das Zehnersystem, Koeffizienten sind Zahlen und nicht Bits

Exkurs: Zahldarstellung in verschiedenen Basen

- **Länge** der Repräsentation einer Zahl: Anzahl Bits (Anzahl Nullen und Einsen), plus Vorzeichen-Bit
- Analog für das Zehnersystem, Koeffizienten sind Zahlen und nicht Bits
- Klar: Länge der Repräsentation einer Zahl in beiden Zahlsystemen abhängig von der Größe der Zahl

Exkurs: Zahldarstellung in verschiedenen Basen

- **Länge** der Repräsentation einer Zahl: Anzahl Bits (Anzahl Nullen und Einsen), plus Vorzeichen-Bit
- Analog für das Zehnersystem, Koeffizienten sind Zahlen und nicht Bits
- Klar: Länge der Repräsentation einer Zahl in beiden Zahlsystemen abhängig von der Größe der Zahl
- Im Jupyter: Ausdeklinieren der Idee einer Fixkomma-Repräsentation

Exkurs: Zahldarstellung in verschiedenen Basen

- **Länge** der Repräsentation einer Zahl: Anzahl Bits (Anzahl Nullen und Einsen), plus Vorzeichen-Bit
- Analog für das Zehnersystem, Koeffizienten sind Zahlen und nicht Bits
- Klar: Länge der Repräsentation einer Zahl in beiden Zahlsystemen abhängig von der Größe der Zahl
- Im Jupyter: Ausdeklinieren der Idee einer Fixkomma-Repräsentation
  - Feste Anzahl Vor- und Nachkommastellen

Exkurs: Zahldarstellung in verschiedenen Basen

- **Länge** der Repräsentation einer Zahl: Anzahl Bits (Anzahl Nullen und Einsen), plus Vorzeichen-Bit
- Analog für das Zehnersystem, Koeffizienten sind Zahlen und nicht Bits
- Klar: Länge der Repräsentation einer Zahl in beiden Zahlsystemen abhängig von der Größe der Zahl
- Im Jupyter: Ausdeklinieren der Idee einer Fixkomma-Repräsentation
  - Feste Anzahl Vor- und Nachkommastellen
  - Probleme bei Zahlen unterschiedlicher Größenordnung: Länge explodiert

Exkurs: Zahldarstellung in verschiedenen Basen

- **Länge** der Repräsentation einer Zahl: Anzahl Bits (Anzahl Nullen und Einsen), plus Vorzeichen-Bit
- Analog für das Zehnersystem, Koeffizienten sind Zahlen und nicht Bits
- Klar: Länge der Repräsentation einer Zahl in beiden Zahlsystemen abhängig von der Größe der Zahl
- Im Jupyter: Ausdeklinieren der Idee einer Fixkomma-Repräsentation
  - Feste Anzahl Vor- und Nachkommastellen
  - Probleme bei Zahlen unterschiedlicher Größenordnung: Länge explodiert
  - Effizienz-Problem: Hardware (ALUs) nicht konstruierbar für beliebige Längen

Ausweg:
Gleitkomma-Zahlen

Gleitkomma-Zahlen

- Etablierter Ausweg: **Gleitkomma-Zahlen** (floating point numbers)

Gleitkomma-Zahlen

- Etablierter Ausweg: **Gleitkomma-Zahlen** (floating point numbers)
- Repräsentation einer Gleitkomma-Zahl x

$$x = s \cdot (m_0, m_1 \cdots m_M) \cdot b^{e_0 e_1 \ldots e_E}$$

Gleitkomma-Zahlen

- Etablierter Ausweg: **Gleitkomma-Zahlen** (floating point numbers)
- Repräsentation einer Gleitkomma-Zahl x

$$x = s \cdot (m_0, m_1 \cdots m_M) \cdot b^{e_0 e_1 \ldots e_E}$$

- b: **Basis** der Darstellung

Gleitkomma-Zahlen

- Etablierter Ausweg: **Gleitkomma-Zahlen** (floating point numbers)
- Repräsentation einer Gleitkomma-Zahl x

$$x = s \cdot (m_0, m_1 \cdots m_M) \cdot b^{e_0 e_1 \ldots e_E}$$

- b: **Basis** der Darstellung
  - In der Hardware-Realität $b = 2$, im normalen Leben $b = 10$

Gleitkomma-Zahlen

- Etablierter Ausweg: **Gleitkomma-Zahlen** (floating point numbers)
- Repräsentation einer Gleitkomma-Zahl x

$$x = s \cdot (m_0, m_1 \cdots m_M) \cdot b^{e_0 e_1 \ldots e_E}$$

- b: **Basis** der Darstellung
  - In der Hardware-Realität $b = 2$, im normalen Leben $b = 10$
- m_i: Koeffizienten der **Mantisse**, e_i Koeffizienten des **Exponents**, in der gegebenen Basis

Gleitkomma-Zahlen

- Etablierter Ausweg: **Gleitkomma-Zahlen** (floating point numbers)
- Repräsentation einer Gleitkomma-Zahl x

$$x = s \cdot (m_0, m_1 \cdots m_M) \cdot b^{e_0 e_1 \ldots e_E}$$

- b: **Basis** der Darstellung
  - In der Hardware-Realität $b = 2$, im normalen Leben $b = 10$
- m_i: Koeffizienten der **Mantisse**, e_i Koeffizienten des **Exponents**, in der gegebenen Basis
  - Für $b = 2$: $m_i, e_i \in \{0, 1\}$, für $b = 10$: $m_i, e_i \in \{0, 1, \ldots, 9\}$

Gleitkomma-Zahlen

- Etablierter Ausweg: **Gleitkomma-Zahlen** (floating point numbers)
- Repräsentation einer Gleitkomma-Zahl x

$$x = s \cdot (m_0, m_1 \cdots m_M) \cdot b^{e_0 e_1 \ldots e_E}$$

- b: **Basis** der Darstellung
  - In der Hardware-Realität $b = 2$, im normalen Leben $b = 10$
- m_i: Koeffizienten der **Mantisse**, e_i Koeffizienten des **Exponents**, in der gegebenen Basis
  - Für $b = 2$: $m_i, e_i \in \{0, 1\}$, für $b = 10$: $m_i, e_i \in \{0, 1, \ldots, 9\}$
- s: Sonderrolle, definiert das **Vorzeichen**

Gleitkomma-Zahlen

- **Eindeutigkeit** der Darstellung: Normierung der Mantisse auf $1 \leq m < b$

Gleitkomma-Zahlen

- **Eindeutigkeit** der Darstellung: Normierung der Mantisse auf $1 \leq m < b$
- Beispiele für $b = 10$: $1.2 = 1.2 \cdot 10^0$, $0.99 = 9.9 \cdot 10^{-1}$ oder $1234.56 = 1.23456 \cdot 10^3$

Gleitkomma-Zahlen

- **Eindeutigkeit** der Darstellung: Normierung der Mantisse auf $1 \leq m < b$
- Beispiele für $b = 10$: $1.2 = 1.2 \cdot 10^0$, $0.99 = 9.9 \cdot 10^{-1}$ oder $1234.56 = 1.23456 \cdot 10^3$
- Lichtgeschwindigkeit im Vakuum: $c = 299\ 792\ 458 = 2.99792458 \cdot 10^8$

Gleitkomma-Zahlen

- **Eindeutigkeit** der Darstellung: Normierung der Mantisse auf $1 \leq m < b$
- Beispiele für $b = 10$: $1.2 = 1.2 \cdot 10^0$, $0.99 = 9.9 \cdot 10^{-1}$ oder $1234.56 = 1.23456 \cdot 10^3$
- Lichtgeschwindigkeit im Vakuum: $c = 299\ 792\ 458 = 2.99792458 \cdot 10^8$
- **Beispiel in Python**

Gleitkomma-Zahlen

- **Eindeutigkeit** der Darstellung: Normierung der Mantisse auf $1 \leq m < b$
- Beispiele für $b = 10$: $1.2 = 1.2 \cdot 10^0$, $0.99 = 9.9 \cdot 10^{-1}$ oder $1234.56 = 1.23456 \cdot 10^3$
- Lichtgeschwindigkeit im Vakuum: $c = 299\ 792\ 458 = 2.99792458 \cdot 10^8$
- **Beispiel in Python**
  - Format-Spezifizierer `:e` liefert normierte Gleitkomma-Darstellung in der Basis 10

Gleitkomma-Zahlen

- **Eindeutigkeit** der Darstellung: Normierung der Mantisse auf $1 \leq m < b$
- Beispiele für $b = 10$: $1.2 = 1.2 \cdot 10^0$, $0.99 = 9.9 \cdot 10^{-1}$ oder $1234.56 = 1.23456 \cdot 10^3$
- Lichtgeschwindigkeit im Vakuum: $c = 299\ 792\ 458 = 2.99792458 \cdot 10^8$
- **Beispiel in Python**
  - Format-Spezifizierer `:e` liefert normierte Gleitkomma-Darstellung in der Basis 10
  - Unabhängig davon, dass unter der Haube das Binärsystem verwendet wird

Gleitkomma-Zahlen

- Für reproduzierbare Ergebnisse auf allen Rechnerarchitekturen: Standardisierung durch IEEE (Institute of Electrical and Electronics Engineers)

Gleitkomma-Zahlen

- Für reproduzierbare Ergebnisse auf allen Rechnerarchitekturen: Standardisierung durch IEEE (Institute of Electrical and Electronics Engineers)
- **Verbindliche Industrienorm**

Gleitkomma-Zahlen

- Für reproduzierbare Ergebnisse auf allen Rechnerarchitekturen: Standardisierung durch IEEE (Institute of Electrical and Electronics Engineers)
- **Verbindliche Industrienorm**
- Auch: ALU in Hardware schneller bei fester Operandenlänge

Gleitkomma-Zahlen

- Für reproduzierbare Ergebnisse auf allen Rechnerarchitekturen: Standardisierung durch IEEE (Institute of Electrical and Electronics Engineers)
- **Verbindliche Industrienorm**
- Auch: ALU in Hardware schneller bei fester Operandenlänge
- In Python: Datentyp `float`

Gleitkomma-Zahlen

- Für reproduzierbare Ergebnisse auf allen Rechnerarchitekturen: Standardisierung durch IEEE (Institute of Electrical and Electronics Engineers)
- **Verbindliche Industrienorm**
- Auch: ALU in Hardware schneller bei fester Operandenlänge
- In Python: Datentyp `float`
  - Basis immer $b = 2$

Gleitkomma-Zahlen

- Für reproduzierbare Ergebnisse auf allen Rechnerarchitekturen: Standardisierung durch IEEE (Institute of Electrical and Electronics Engineers)
- **Verbindliche Industrienorm**
- Auch: ALU in Hardware schneller bei fester Operandenlänge
- In Python: Datentyp `float`
  - Basis immer $b = 2$
  - Mantisse hat (in der Binärdarstellung) 52 Stellen

Gleitkomma-Zahlen

- Für reproduzierbare Ergebnisse auf allen Rechnerarchitekturen: Standardisierung durch IEEE (Institute of Electrical and Electronics Engineers)
- **Verbindliche Industrienorm**
- Auch: ALU in Hardware schneller bei fester Operandenlänge
- In Python: Datentyp `float`
  - Basis immer $b = 2$
  - Mantisse hat (in der Binärdarstellung) 52 Stellen
  - Exponent repräsentiert durch 11 Bits (plus Bias-Magie)

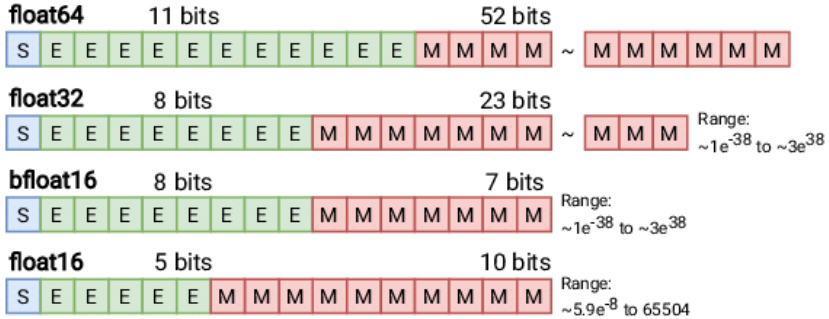
Gleitkomma-Zahlen

- Für reproduzierbare Ergebnisse auf allen Rechnerarchitekturen: Standardisierung durch IEEE (Institute of Electrical and Electronics Engineers)
- **Verbindliche Industrienorm**
- Auch: ALU in Hardware schneller bei fester Operandenlänge
- In Python: Datentyp `float`
  - Basis immer $b = 2$
  - Mantisse hat (in der Binärdarstellung) 52 Stellen
  - Exponent repräsentiert durch 11 Bits (plus Bias-Magie)
  - Sign-Bit von 0 repräsentiert positive Zahlen, eines von 1 negative

Gleitkomma-Zahlen

- Für reproduzierbare Ergebnisse auf allen Rechnerarchitekturen: Standardisierung durch IEEE (Institute of Electrical and Electronics Engineers)
- **Verbindliche Industrienorm**
- Auch: ALU in Hardware schneller bei fester Operandenlänge
- In Python: Datentyp `float`
  - Basis immer $b = 2$
  - Mantisse hat (in der Binärdarstellung) 52 Stellen
  - Exponent repräsentiert durch 11 Bits (plus Bias-Magie)
  - Sign-Bit von 0 repräsentiert positive Zahlen, eines von 1 negative
  - Speicheraufwand für eine Gleitkommazahl in diesem Format damit immer **8 Byte** (= 64 Bits)

Gleitkomma-Zahlen



Akash Haridas, Vadlamani Nagabhushana Rao, Yuki Minamoto, „Deep Neural Networks to Correct Sub-Precision Errors in CFD“, Abbildung 1, https://www.researchgate.net/publication/358490870_Deep_Neural_Networks_to_Correct_Sub-Precision_Errors_in_CFD, CC-BY

Arbeiten mit Gleitkomma-Zahlen in Python

Rundungsregeln und Maschinengenauigkeit für Gleitkomma-Zahlen

Rundungsregeln und Maschinengenauigkeit

- Konsequenz der festen Speichergröße für Gleitkomma-Zahlen

Rundungsregeln und Maschinengenauigkeit

- Konsequenz der festen Speichergröße für Gleitkomma-Zahlen
  - Existenz einer **größten und kleinsten darstellbare positiven Zahl**

Rundungsregeln und Maschinengenauigkeit

- Konsequenz der festen Speichergröße für Gleitkomma-Zahlen
  - Existenz einer **größten und kleinsten darstellbare positiven Zahl**
  - Analog für negative Zahlen

Rundungsregeln und Maschinengenauigkeit

- Konsequenz der festen Speichergröße für Gleitkomma-Zahlen
  - Existenz einer **größten und kleinsten darstellbare positiven Zahl**
  - Analog für negative Zahlen
  - **Demo in Jupyter**: Ausrechnen der Zahlen durch Einsetzen der Längen von Mantisse und Exponenten

Rundungsregeln und Maschinengenauigkeit

- Konsequenz der festen Speichergröße für Gleitkomma-Zahlen
  - Existenz einer **größten und kleinsten darstellbare positiven Zahl**
  - Analog für negative Zahlen
  - **Demo in Jupyter**: Ausrechnen der Zahlen durch Einsetzen der Längen von Mantisse und Exponenten
- Konsequenz des Sign-Bits

Rundungsregeln und Maschinengenauigkeit

- Konsequenz der festen Speichergröße für Gleitkomma-Zahlen
  - Existenz einer **größten und kleinsten darstellbare positiven Zahl**
  - Analog für negative Zahlen
  - **Demo in Jupyter**: Ausrechnen der Zahlen durch Einsetzen der Längen von Mantisse und Exponenten
- Konsequenz des Sign-Bits
  - **zwei Darstellungen der Null**, nämlich `+0.0` und `-0.0`

Rundungsregeln und Maschinengenauigkeit

- Spannende Zahl: `eps=2.220446049250313e-16`, **Maschinengenauigkeit**

Rundungsregeln und Maschinengenauigkeit

- Spannende Zahl: `eps=2.220446049250313e-16`, **Maschinengenauigkeit**
- Begleitet uns durch diesen Kurs

Rundungsregeln und Maschinengenauigkeit

- Spannende Zahl: `eps=2.220446049250313e-16`, **Maschinengenauigkeit**
- Begleitet uns durch diesen Kurs
- Zunächst formal argumentiert, **beweisbar innerhalb des IEEE-Gleitkomma-Systems**

Rundungsregeln und Maschinengenauigkeit

- Spannende Zahl: `eps=2.220446049250313e-16`, **Maschinengenauigkeit**
- Begleitet uns durch diesen Kurs
- Zunächst formal argumentiert, **beweisbar innerhalb des IEEE-Gleitkomma-Systems**
  - Zu jeder reellen Zahl $x \in [x_{min}, x_{max}] \subset \mathbb{R}^+$ bzw. dem entsprechend negativen Bereich existiert eine „gerundete“ Gleitkomma-Zahl $\mathrm{rd}(x)$, so dass der relative Fehler $\frac{|x-\mathrm{rd}(x)|}{|x|}$ durch `eps` beschränkt ist:

$$|x - \mathrm{rd}(x)| \leq |x| \,\mathtt{eps}, \quad \text{für alle } x \in \mathbb{R}$$

Rundungsregeln und Maschinengenauigkeit

- Spannende Zahl: `eps=2.220446049250313e-16`, **Maschinengenauigkeit**
- Begleitet uns durch diesen Kurs
- Zunächst formal argumentiert, **beweisbar innerhalb des IEEE-Gleitkomma-Systems**
  - Zu jeder reellen Zahl $x \in [x_{min}, x_{max}] \subset \mathbb{R}^+$ bzw. dem entsprechend negativen Bereich existiert eine „gerundete“ Gleitkomma-Zahl $\mathrm{rd}(x)$, so dass der relative Fehler $\frac{|x-\mathrm{rd}(x)|}{|x|}$ durch `eps` beschränkt ist:

$$|x - \mathrm{rd}(x)| \leq |x| \,\mathtt{eps}, \quad \text{für alle } x \in \mathbb{R}$$

  - x_{min} und x_{max}: kleinste und größte positive darstellbare Gleitkomma-Zahlen

Rundungsregeln und Maschinengenauigkeit

- Zugegebenermaßen **sehr abstrakt**

Rundungsregeln und Maschinengenauigkeit

- Zugegebenermaßen **sehr abstrakt**
- Aber auch sehr wichtig, bei angestrebter Strategie der Ignorierung kleinteiliger Details der Rundungsregeln

Rundungsregeln und Maschinengenauigkeit

- Zugegebenermaßen **sehr abstrakt**
- Aber auch sehr wichtig, bei angestrebter Strategie der Ignorierung kleinteiliger Details der Rundungsregeln
- Auf **unserer Reiseflughöhe**

Rundungsregeln und Maschinengenauigkeit

- Zugegebenermaßen **sehr abstrakt**
- Aber auch sehr wichtig, bei angestrebter Strategie der Ignorierung kleinteiliger Details der Rundungsregeln
- Auf **unserer Reiseflughöhe**
  - Wir wissen nicht, auf welche Gleitkomma-Zahl gerundet wird

Rundungsregeln und Maschinengenauigkeit

- Zugegebenermaßen **sehr abstrakt**
- Aber auch sehr wichtig, bei angestrebter Strategie der Ignorierung kleinteiliger Details der Rundungsregeln
- Auf **unserer Reiseflughöhe**
  - Wir wissen nicht, auf welche Gleitkomma-Zahl gerundet wird
  - Wir wissen, dass „vernünftig“ gerundet wird

Rundungsregeln und Maschinengenauigkeit

- Zugegebenermaßen **sehr abstrakt**
- Aber auch sehr wichtig, bei angestrebter Strategie der Ignorierung kleinteiliger Details der Rundungsregeln
- Auf **unserer Reiseflughöhe**
  - Wir wissen nicht, auf welche Gleitkomma-Zahl gerundet wird
  - Wir wissen, dass „vernünftig“ gerundet wird
  - In dem Sinne dass wir eine Grenze für den dabei auftretenden relativen Fehler kennen, nämlich die Maschinengenauigkeit

Rundungsregeln und Maschinengenauigkeit

- Zugegebenermaßen **sehr abstrakt**
- Aber auch sehr wichtig, bei angestrebter Strategie der Ignorierung kleinteiliger Details der Rundungsregeln
- Auf **unserer Reiseflughöhe**
  - Wir wissen nicht, auf welche Gleitkomma-Zahl gerundet wird
  - Wir wissen, dass „vernünftig“ gerundet wird
  - In dem Sinne dass wir eine Grenze für den dabei auftretenden relativen Fehler kennen, nämlich die Maschinengenauigkeit
- Diskussion der Konsequenzen im nächsten Video

Impressum, Danksagung und Quellen

Stiftung Innovation in der Hochschullehre

Gefördert durch die Stiftung Innovation in der Hochschullehre im Rahmen des Projekts digit@L, https://stiftung-hochschullehre.de

SimTech

Gefördert mit Mitteln der Deutschen Forschungsgemeinschaft (EXC 2075 - 390740016) im Rahmen der Exzellenzstrategie

---

Autor: Dominik Göddeke, IANS, Universität Stuttgart

ZOERR

Veröffentlicht auf dem Zentralen OER Repositorium Baden-Württemberg, https://www.zoerr.de