Technische Universität Braunschweig

Programmieren 1 – Vorlesung #1

Arne Schmidt

Wiederholung

Begriffe

Algorithmen
- Berechnungs-vorschrift
- Eigenschaften

Paradigmen
- Imperativ
- Funktional
- Objektorientiert

Datentypen
- Char
- Boolean
- Integer

Programmiersprache
- Lexik
- Syntax
- Semantik

Sprachklassen
- Maschinensprache
- Maschinenorientiert
- Problemorientiert

Compiler

Interpreter

Kapitel 2 – Grundlagen der imperativen Programmierung

Python

Python ist eine Skriptsprache und kommt ohne Compiler aus.

Wir benötigen also nur einen Interpreter, um ein Pythonprogramm auszuführen. Über Konsole geht das mit:

```
python3 datei.py
```

Vorteile von Python

- Leicht zu lernen
- Einfach Code zu lesen
- Dynamisch Typisiert
- Schnelles Prototyping

Nachteile von Python

- Ziemlich langsam
- Teils hoher Speicherbedarf
- Testen ist aufwändiger
- Wechsel auf andere Sprachen kann schwieriger werden

Wir nutzen Python, um einfach die Konzepte der imperativen Programmierung zu erlernen.

Python Beispiel

```
def func(n):
    x = n
    f = 1
    while x > 0:
        f *= x
        x -= 1
    return f

print(func(10))
```

Schlüsselwörter

Kontrollstrukturen
(while, if, for, ...)

Lexik

Syntax und Semantik

```
def func(n):
    x = n
    f = 1
    while x > 0:
        f *= x
    x -= 1
    return f

print(func(10))
```

Dieser Code liefert etwas anderes!
Einrückung ist wichtig.
Mehr dazu später!

Kapitel 2.1 – Python Grundlagen

Pythons Lexik - Kommentare

Code muss kommentiert werden
- Außerhalb der Datei? Schwierig nachzuvollziehen!
- Innerhalb der Datei? Wird das mit interpretiert?
  => Interpreter soll bestimmte Zeile ignorieren!

Mit #Text können einzeilige Kommentare angegeben werden.

Mit """Text""" können mehrzeilige Kommentare angegeben werden (3 Anführungszeichen am Anfang und 3 am Ende).

```
# Das ist ein Kommentar!
```

```
"""
Dieser Kommentar
geht über
mehrere Zeilen!
"""
```

Pythons Lexik - Zeichen

Trennzeichen:
Leerzeichen, Zeilenendzeichen (ENTER-Taste), Tabulatorzeichen (TAB-Taste)

Operatoren:
+, -, *, **, /, %, ==, +=, -=, >=, <=, not, >>, <<, |, &, ^, ...

Interpunktion:
., (,), {, }, [,]

Syntax und Semantik lernen wir später!

Pythons Lexik - Schlüsselwörter

and	elif	if	return
as	else	lambda	True
assert	except	None	try
break	False	nonlocal	while
class	finally	not	with
continue	for	or	yield
def	from	pass	...
del	global	raise	

Pythons Lexik - Schlüsselwörter

and	**elif**	**if**	**return**
as	**else**	lambda	**True**
assert	except	None	try
break	**False**	nonlocal	**while**
class	finally	**not**	with
continue	**for**	**or**	yield
def	from	pass	**...**
del	global	raise	

Pythons Lexik - Bezeichner

Bezeichner:

- Namen für Variablen, Methoden, Klassen, ...
- Beliebig lang
- Starten mit Buchstaben oder Unterstrich
- Sind Case-Sensitive (Groß- und Kleinbuchstaben sind unterschiedlich)

Konvention:

- *Variablen und Funktionen*
  - Werden klein geschrieben
  - Mehrere Wörter werden mit Unterstrich getrennt
  - Beispiel: my_variable
- *Klassen* (siehe spätere Vorlesung zu Java)
  - CamelCase ohne Unterstriche (MyClass)
- *Konstanten*
  - Nur Großbuchstaben mit Unterstrichen (MY_CONSTANT)

Pythons Lexik - Datentypen

Datentypen:
- Boolean: True, False
- Ganzzahlig (Integer, int): 52, -9, 0, 387
- Fließkommazahlen (float): 0.3, -3.141, 9.4e+02
- Komplexe Zahlen: 3.4+8j
- Strings: “Hi! Dies ist ein String”

Die Typzuordnung passiert automatisch!
Variablen können bei Neuzuweisung ihren Typ wechseln.
Das nennt sich ***Dynamische Typisierung***.

Aber! Binäre Operatoren (+, -, ...) funktionieren im Allgemeinen nur auf gleichem Typ.

$x = \text{False}$ #x ist ein Boolean
$x = 34$ #x ist ein int
$x = 34.0$ #x ist ein float
$x =$ “34“ #x ist ein string

Kapitel 2.2 – Datentypen im Detail

Datentypen in verschiedenen Sprachen

In wieweit unterscheiden sich Datentypen in verschiedenen Sprachen?

Was muss man beachten?

Aus letzter Vorlesung:
Datentypen besitzen **Wertebereich** und **Operatoren.**

Oft ist ein Großteil an Operatoren in verschiedenen Sprachen gleich, nur die Lexik ist etwas anders.

Signifikant anders kann der Wertebereich sein!

Operatoren auf Datentypen

Für Operatoren definieren wir folgendes

Stelligkeit: Anzahl an *Operanden* (übergebene Werte)
Rückgabewert: Das Ergebnis des Operators
Priorität: Gibt die Auswertungsreihenfolge an

Generell

- Operatoren sind ein- oder zweistellig (vereinzelt gibt es auch dreistellige)
- Die Datentypen der Operanden und des Rückgabewertes sind vorgegeben.

Kapitel 2.2.1 – Boolean

Boolean – Unäre Operationen

Werte von Variablen sind wahr oder falsch (1 oder 0, True oder False)

Eine *unäre Operation* bekommt als Input eine Variable.
Für Boolean gibt es damit nur vier Möglichkeiten $f_0^1, f_1^1, f_2^1, f_3^1$:

b	$f_0^1(b)$	$f_1^1(b)$	$f_2^1(b)$	$f_3^1(b)$
0	0	1	0	1
1	0	0	1	1

Speziell $f_1^1(b)$ als Negation (not) ist von Interesse.
not True == False und not False == True

In anderen Sprachen wird not oft als ! geschrieben. Z.B. bei Java.

Boolean – Binäre Operationen

Eine *binäre Operation* bekommt als Input zwei Variablen.
Für Boolean gibt es damit 16 Möglichkeiten f_0^2, …, f_{15}^2 :

b_0	b_1	f_0^2	f_1^2	f_2^2	f_3^2	f_4^2	f_5^2	f_6^2	f_7^2	f_8^2	f_9^2	f_{10}^2	f_{11}^2	f_{12}^2	f_{13}^2	f_{14}^2	f_{15}^2
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

In anderen Sprachen wird and oft als && geschrieben. Z.B. bei Java.

b_0 & b_1 bzw b_0 and b_1
(logisches und bzw Konjunktion)

Boolean – Binäre Operationen

Eine *binäre Operation* bekommt als Input zwei Variablen.
Für Boolean gibt es damit 16 Möglichkeiten f_0^2, …, f_{15}^2 :

b_0	b_1	f_0^2	f_1^2	f_2^2	f_3^2	f_4^2	f_5^2	f_6^2	f_7^2	f_8^2	f_9^2	f_{10}^2	f_{11}^2	f_{12}^2	f_{13}^2	f_{14}^2	f_{15}^2
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

In anderen Sprachen wird or oft als || geschrieben. Z.B. bei Java.

$b_0 \mid b_1$ bzw b_0 or b_1 (logisches oder bzw. Disjunktion)

Boolean – Binäre Operationen

Eine *binäre Operation* bekommt als Input zwei Variablen.
Für Boolean gibt es damit 16 Möglichkeiten f_0^2, …, f_{15}^2 :

b_0	b_1	f_0^2	f_1^2	f_2^2	f_3^2	f_4^2	f_5^2	f_6^2	f_7^2	f_8^2	f_9^2	f_{10}^2	f_{11}^2	f_{12}^2	f_{13}^2	f_{14}^2	f_{15}^2
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

b_0 ^ b_1
(exklusives oder)

Boolean – Binäre Operationen

Eine *binäre Operation* bekommt als Input zwei Variablen.
Für Boolean gibt es damit 16 Möglichkeiten f_0^2, …, f_{15}^2 :

b_0	b_1	f_0^2	f_1^2	f_2^2	f_3^2	f_4^2	f_5^2	f_6^2	f_7^2	f_8^2	f_9^2	f_{10}^2	f_{11}^2	f_{12}^2	f_{13}^2	f_{14}^2	f_{15}^2
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

$b_0 == b_1$
(Gleichheit)

Boolean – Binäre Operationen

Eine *binäre Operation* bekommt als Input zwei Variablen.
Für Boolean gibt es damit 16 Möglichkeiten f_0^2, …, f_{15}^2 :

b_0	b_1	f_0^2	f_1^2	f_2^2	f_3^2	f_4^2	f_5^2	f_6^2	f_7^2	f_8^2	f_9^2	f_{10}^2	f_{11}^2	f_{12}^2	f_{13}^2	f_{14}^2	f_{15}^2
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

b_0 != b_1
(Ungleichheit)

Kapitel 2.2.2 – Integer

Integer - Operatoren

Integer in Python können beliebig groß sein.

In vielen anderen Sprachen ist der Wertebereich beschränkt.
Dazu in späteren Vorlesungen mehr!

Bezeichner	Zeichen	Rückgabetyp	Beispiel
Addition	+	Integer	3 + 4, Rückgabe 7
Subtraktion	-	Integer	3 - 9, Rückgabe -6
Multiplikation	*	Integer	2 * 6, Rückgabe 12
Division	/	Integer oder Float	7 / 2, Rückgabe 3.5
Integer Division	//	Integer	7 // 2, Rückgabe 3
Modulo	%	Integer	7 % 2, Rückgabe 1

Integer – Vergleichsoperatoren

Vergleichsoperatoren geben immer einen Boolean zurück.

Bezeichner	Zeichen	Beispiel
Gleichheit	==	25 == 5 * 5, Rückgabe True
Größer oder gleich	>=	42 >= 69, Rückgabe False
Größer	>	71 > 71, Rückgabe False
Ungleich	!=	99 != 100, Rückgabe True

Integer – Bitweise Operatoren / Binär Zahlen

Zahlen werden in der Regel als binäre Zahlen dargstellt. Damit können auf alle Bits Boolesche Operationen durchgeführt werden.

Beispiele:
8 entspricht $(1000)_2$
25 entspricht $(11001)_2$

Stelle (von rechts)	**n.**	**3.**	**2.**	**1.**	**0.**
Wertigkeit	2^n	8	4	2	1

Weitere Beispiele:

$(1001)_2$ enstpricht:	9	107 enstpricht:	$(1101011)_2$
$(11100)_2$ enstpricht:	28	46 enstpricht:	$(101110)_2$
$(110101)_2$ enstpricht:	53	99 enstpricht:	$(1100011)_2$

Integer – Bitweise Operatoren

Zahlen werden in der Regel als binäre Zahlen dargstellt. Damit können auf alle Bits Boolesche Operationen durchgeführt werden.

Beispiele:
8 entspricht $(1000)_2$
25 entspricht $(11001)_2$

Stelle (von rechts)	**n.**	**3.**	**2.**	**1.**	**0.**
Wertigkeit	2^n	8	4	2	1

Bezeichner	**Zeichen**	**Rückgabetyp**	**Beispiel**
Bitshift	<<, >>	Integer (abgerundet)	3 << 2, Rückgabe 12 (3* 2 * 2)
And	&	Integer	13 & 11, Rückgabe 9
Or	|	Integer	13 | 11, Rückgabe 15
Exclusive Or	^	Integer	13 ^ 11, Rückgabe 6

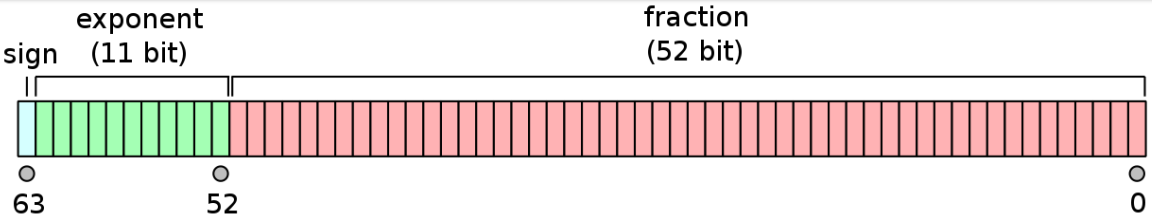
Kapitel 2.2.3 – Floats

Float (Fließkommazahlen)

Aufbau (nach IEEE 754, 64 bit):
Zahlen werden mit *double Precision* als 64-Bitzahl in der Form
$(-1)^{\text{sign}} \cdot (1.\text{fraction})_2 \cdot 2^{(\text{exponent})_2 - 1023}$ gespeichert.

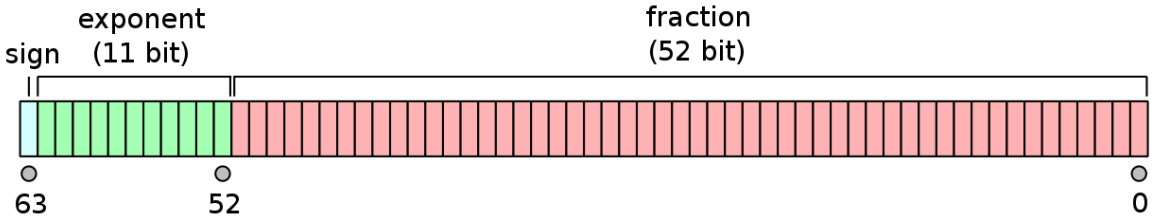
Sign – 1 Bit für das Vorzeichen (+/-) der Zahl
Exponent – 11 Bits für den bitshift
Fraction – 52 Bits für die signifikanten Bits

Wertebereich:
$2.2250738585072014 \cdot 10^{-308}$ bis $1.7976931348623157 \cdot 10^{+308}$



Float – Beispiel 1

Aufbau:
Zahlen werden mit *double Precision* als 64-Bitzahl in der Form
$(-1)^{\text{sign}} \cdot (1.\text{fraction})_2 \cdot 2^{(\text{exponent})_2 - 1023}$ gespeichert.



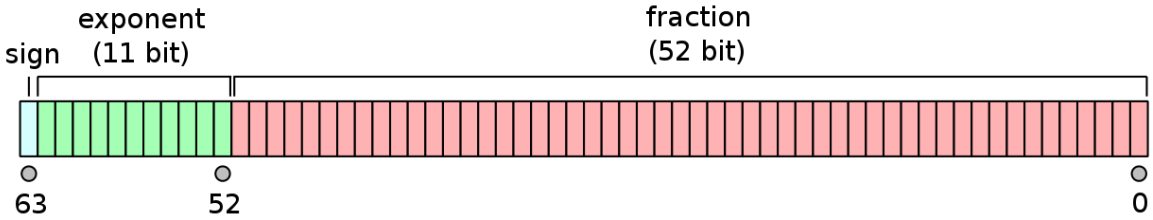
Betrachte:
0 | 10000001100 | 1100110…

Wert des Exponenten: 1036
Wert $(1.\text{fraction})_2$: $(1.110011)_2 = 1.796875$

Damit ist 0 | 10000001100 | 1100110… als Wert:
$(-1)^0 \cdot 1.796875 \cdot 2^{1036-1023} = 1.796875 \cdot 2^{13} = 14720$

Float – Beispiel 2

Aufbau:
Zahlen werden mit *double Precision* als 64-Bitzahl in der Form
$(-1)^{\text{sign}} \cdot (1.\text{fraction})_2 \cdot 2^{(\text{exponent})_2 - 1023}$ gespeichert.



Betrachte:
$4.1875 = (100.0011)_2 = (1.000011)_2 \cdot 2^2$

Damit ist 4.1875 als float:
0 | 10000000001 | 000011000…

Wert des Exponenten: $2+1023 = 1025 = (10000000001)_2$

Wert fraction: 000011000…

Float - Operatoren

Operatoren sind gleich mit Integer; nur bitweise Operatoren werden nicht unterstützt!

Im Gegensatz zu Integer muss man bei manchen Operationen sehr gut aufpassen! Beispiel: Vergleiche

In der Regel können Variablen und Werte mit folgenden Operatoren verglichen werden:

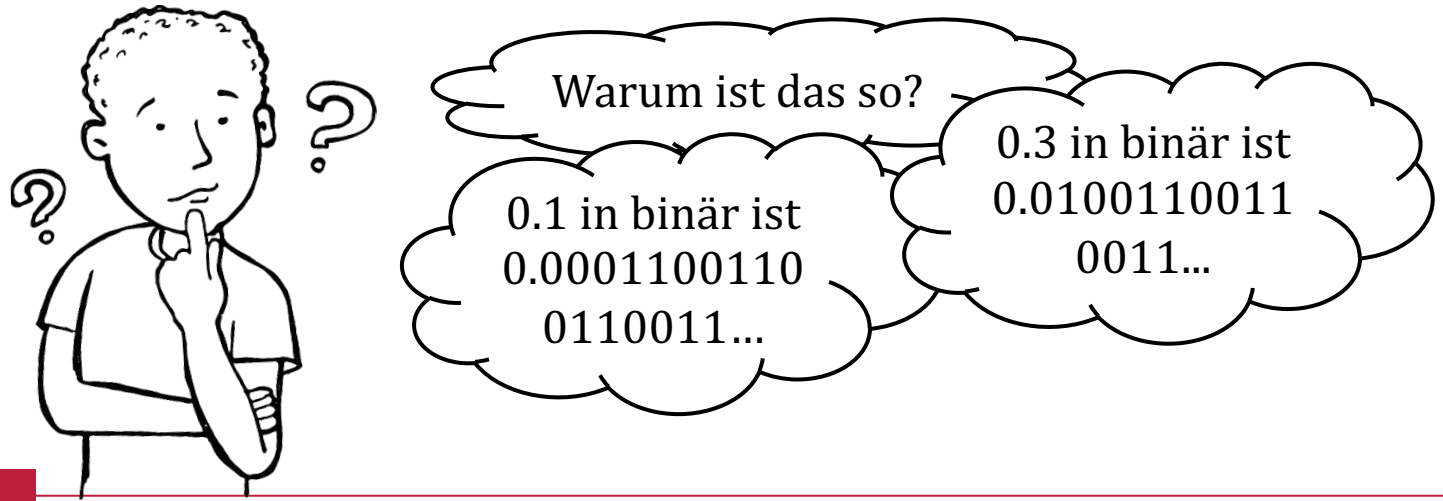
>= (Größer oder gleich)
> (größer)
< (kleiner
<= (kleiner oder gleich)
== (gleich)
!= (ungleich)

Float - Vergleichsoperatoren

Sei $x = 0.1$ ein float.

Welchen Wert hat der Test $3 * x == 0.3$?

Das Ergebnis ist False, denn $3 * x - 0.3 = 5.551115123125783e - 17$ mit double Precision



Float - Vergleichsoperatoren

0.1 in binär ist
0.0001100110
0110011…

0.3 in binär ist
0.0100110011
0011…

Irgendwo muss man abschneiden, man verliert also Informationen!

Also hat man (z.B. mit 8 signifikanten Stellen):

$$3 \cdot 1.10011001 \cdot 2^4 \rightarrow 1.00110010 \cdot 2^2$$
$$\neq 1.00110011 \cdot 2^2$$

Wie kann man das umgehen?
Bspw. mit einem Test wie $|3 * x - 0.3| < 10^{-12}$

Weitere Probleme mit Floats

```
a = 1000
while a != 0:
    a -= 0.001
```

Terminiert nie!

Addieren einer sehr großen Zahl A und einer sehr kleinen Zahl B kann als Ergebnis A sein!

Es existieren 0 und -0.

Addition ist nicht immer kumulativ, d.h. a + (b+c) != (a+b)+c

Große Ganzzahlen lassen sich nicht immer darstellen

Kapitel 2.2.4 – Strings

Strings – Definition und Operatoren

String:
Eine aneinandergereite Kette von Zeichen.
Üblicherweise wird eine Zeichenkette in Anführungszeichen gesetzt.
Beispiel “Dies ist ein String <(o.o<)”

Python bietet einfache Operatoren für Strings an.

+ (Konkatenation): Hängt den zweiten Operanden an den ersten
“Hallo” + “Du!” = “HalloDu!”

*** (Vervielfachen):** Hängt Kopien an den gegebenen String an
“Ente” * 5= “EnteEnteEnteEnteEnte”

in (Enthält): Prüft, ob Operand 1 in Operand 2 enthalten ist.
“teE” in “EnteEnte” = True, aber “tee” in “EnteEnte” = False

Nächste Woche

Nächste Woche