

Modul: B-PRG1 Grundlagen der Programmierung 1 und Einführung in die Programmierung EPR

V04 Elementare Datentypen - Teil 1 Numerischer Datentyp: Integer und Boolescher Datentyp – Der Nichtstyp None

Prof. Dr. Detlef Krömker
Professur für Graphische Datenverarbeitung
Institut für Informatik
Fachbereich Informatik und Mathematik (12)

Wichtige Hinweise zu den Abgaben:

- Es gilt: Das Packen muss mit .zip erfolgen (**NICHT mit .7z**)
Das hatten wir schon gesagt (mehrmals)
→ leider keine Punkte für das Aufgabenblatt.
- Eine verschlüsselte .zip-Datei abgegeben.
→ kein Punktabzug, weil wir das nicht gesagt (verboten) hatten.
- **In Zukunft sind verschlüsselte Dateien verboten!**

Unser heutiges Lernziele

- *Elementare Datentypen sind solche, die von der Programmiersprache direkt unterstützt werden und häufig wertemäßig direkt auf Speicherzellen abgebildet sowie durch entsprechende Hardwarekomponenten effizient behandelt werden können.*
- *Ziel ist es, diese Datentypen als Programmierer*in beherrschen zu lernen und ihre Eigenarten zu kennen.*
- *Die Konzepte sind nicht sonderlich schwierig – Sie müssen sich allerdings durch Übung mit diesen Gegebenheiten vertraut machen.*
- **Der Umgang hiermit muss sicher sitzen ;-)**

Übersicht

- **Numerische Datentypen**
 - *Ganze Zahlen (Integer) - Repräsentationen*
 - *Literale = Notationen in Python*
 - *Ausdrücke (= Terme) - Operatoren und Operationen*
- **Boolescher Datentyp**
 - *Werte und Darstellung in Python*
 - *Literale = Notationen in Python - Operatoren*
 - *Ausdrücke (= Terme) - Operatoren und Operationen*
 - *Vergleichsoperatoren liefern den Booleschen Datentyp*
- **Der "Datentyp" None**
- **Zusammenfassung**

Grundsätzliches

- ▶ **Zahlen** und (Schrift-) **Zeichen** sind zweifellos elementare Datenstrukturen.
- ▶ Jede übliche Programmiersprache stellt diese Datentypen als elementare Datentypen zur Verfügung.
- ▶ Bei Formulierungen in Programmiersprachen versucht man dabei, die üblichen Notationen der Mathematik weitgehend beizubehalten, in der Regel die **angloamerikanischen** Konventionen.

Zahlen - Mathematische Grundlagen

- ▶ Es lohnt sich, die mathematischen Grundlagen kurz zu rekapitulieren, bevor wir die Rechnerrepräsentationen betrachten.
- ▶ Aus der Mathematik sind uns insbesondere folgende Zahlenmengen geläufig:

Die Menge der Natürlichen Zahlen	N oder \mathbb{N}
Die Menge der Ganzen Zahlen	Z oder \mathbb{Z}
Die Menge der Rationalen Zahlen	Q oder \mathbb{Q}
Die Menge der Reellen Zahlen	R oder \mathbb{R}
Die Menge der Komplexen Zahlen	C oder \mathbb{C}
(Die Menge der Quaternionen)	H oder \mathbb{H}
- ▶ Allgemein gilt, das $\mathbf{N \subset Z \subset Q \subset R \subset C \subset H}$

Die uns vertraute Dezimalschreibweise

hier beherrschen wir auch alle Grundrechenarten

- Wir schreiben Zahlen gewöhnlich als **vorzeichenbehaftete Dezimalzahl**, also in einem Stellenwertsystem (polyadisches System) zur Basis 10. Beispiele sind:

0 -42 1,414 3,141 -1,59 oder

- Wert **Z** einer Dezimalzahl hier in Zifferschreibweise $z_m z_{m-1} \dots z_1 z_0 \text{ , } z_{-1} z_{-2} \dots z_{-n}$ ist

$$Z = \sum_{i=-n}^m z_i \cdot 10^i$$

- Anstelle des Kommas als Dezimaltrenner (kennzeichnet die Stelle $(10^0, 10^{-1})$) verwendet man im angloamerikanischen Raum den Punkt
- diese Punkt-Schreibweise findet sich in allen (mir bekannten) Programmiersprachen wieder! Also:

0 -42 1.414 3.141 -1.59

Dualzahlen (Binärzahlen) sind die "natürliche Repräsentation"

- Die Dyadik (dyo, griech. = Zwei), also die Darstellung von Zahlen im Dualsystem (Binärsystem) wurde schon Ende des 17. Jahrhunderts von Leibniz entwickelt. Die Stellenbasis ist hier **2**.
- Die Ziffern z_i (0 oder 1) werden wie im Dezimalsystem ohne Trennzeichen hintereinander geschrieben Zifferschreibweise, ihr Stellenwert entspricht allerdings der zur Stelle passenden **Zweierpotenz** und **nicht** der Zehnerpotenz.
- Der Wert **Z** der Dualzahl ergibt sich durch Addition dieser Ziffern, welche vorher jeweils mit ihrem Stellenwert 2^i multipliziert werden:

$$z_m z_{m-1} \dots z_1 z_0 \quad Z = \sum_{i=0}^m z_i \cdot 2^i$$

Beispiel

Die Ziffernfolge **1101**, hat nicht (wie im Dezimalsystem) den Wert „Tausendeinhundertundeins“ dar, sondern den Wert Dreizehn, denn im **Dualsystem** berechnet sich der Wert durch

$$\begin{aligned}[1101]_2 &= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = [13]_{10} \\ 1101_{(2)} &= 13_{(10)} \\ 1101 (2) &= 13 (10)\end{aligned}$$

Wenn **Sie** hiermit nicht vertraut sind, müssen Sie üben ... **Auch das wandeln!**

Eine recht gute Abhandlung hierzu finden Sie hier:

<http://www.krucker.ch/skripten-uebungen/IAMSkript/IAMKap2.pdf>

Ausgabe: 1996/98(IMG. Krucker 2-1 Zahlendarstellungen Informatik und angewandte Mathematik Hochschule für Technik und Architektur Bern

Beispiele zur Wandlung: dezimal → dual

Wie häufig: Es gibt viele Möglichkeiten – hier durch

Natürliche Zahlen durch

Kettendivision

$$\begin{aligned}42 / 2 &= 21 \text{ Rest } 0 \\ 21 / 2 &= 10 \text{ Rest } 1 \\ 10 / 2 &= 5 \text{ Rest } 0 \\ 5 / 2 &= 2 \text{ Rest } 1 \\ 2 / 2 &= 1 \text{ Rest } 0 \\ 1 / 2 &= 0 \text{ Rest } 1\end{aligned}$$

↑
→ **101010**

Echt gebrochene Zahlen durch

Kettenmultiplikation

$$\begin{aligned}0.234375 * 2 &= 0,468750 \\ 0.46875 * 2 &= 0,93750 \\ 0.9375 * 2 &= 1,8750 \\ 0.875 * 2 &= 1,750 \\ 0.75 * 2 &= 1.5 \\ 0.5 * 2 &= 1\end{aligned}$$

↓
→ **0.001111**

Kettenmultiplikation
brauchen wir erst nächsten Montag!

Andere übliche Basen

Andere übliche Basen sind

- 8 (Oktalsystem) – Ziffern 0, ... , 7
- 16 (Hexadezimalsystem) – Ziffern 0, ... , 9, A, B, ..., F

Das sind Potenzen von 2 und lassen sich durch zusammenfassen wandeln.
Aber, jede natürliche (ganze) Zahl > 1 kann als Basis gewählt werden

Warum können amerikanische Informatiker (sogenannte Computer Scientists)
Weihnachten (25. Dezember) nicht von Halloween (31. Oktober) unterscheiden?

(Antwort: Weil $25(\text{dez}) == 31(\text{oct})$;-)

Leibnitz (* 1646; † 1716) -- Anekdote

Leibnitz sah das **Dualsystem** als ein besonders überzeugendes Sinnbild
des christlichen Glaubens an. So schrieb er an den chinesischen
Kaiser Kangxi:

„Zu Beginn des ersten Tages war die 1, das heißt Gott.

*Zu Beginn des zweiten Tages die 2, denn Himmel und Erde wurden
während des ersten geschaffen.*

*Schließlich zu Beginn des siebenten Tages war schon alles da; deshalb ist
der letzte Tag der vollkommenste und der Sabbat, denn an ihm ist alles
geschaffen und erfüllt, und deshalb schreibt sich die 7 111, also ohne
Null.*

**Und nur wenn man die Zahlen bloß mit 0 und 1 schreibt, erkennt man
die Vollkommenheit des siebenten Tages, der als heilig gilt, und von
dem noch bemerkenswert ist, dass seine Charaktere einen Bezug zur
Dreifaltigkeit haben.**"

Was müssen Sie können?

- Umrechnen vom Dual- ins Dezimalsystem
- Umrechnen vom Dezimal- ins Dualsystem
- Umrechnen vom Oktal, Hexadezimal ins Dezimalsystem
- Umrechnen vom Dezimal- ins Oktal, Hexadezimal
- Umkodieren Oktal \leftrightarrow Dual, Hexadezimal \leftrightarrow Dual
und auch: Hexadezimal \leftrightarrow Oktal
- auch Addieren, Subtrahieren, Multiplizieren im Dualsystem
Es gelten dieselben Regeln wie im Dezimalsystem, siehe Übung!

Da haben wir doch schon etwas geschafft. Jetzt:

Ganze Zahlen:

Positive und negative (nicht gebrochene) Zahlen

..., -4, -3, -2, -1, 0, 1, 2, 3, 4, ...

Kodierung ganzer Zahlen (also auch negativer Zahlen)

Lösung: Den Bereich der verarbeitbaren positiven Zahlen einschränken und die frei werdenden Kodeworte für negative Zahlen genutzt.

0. **Vorzeichenbehaftete Zahl:** 1. Ziffer wird als VZ interpretiert: meist 0: positiv, 1: negativ: (Nachteil: die Null hat zwei Repräsentationen, es gibt also -0 und +0 und die Arithmetik ist kompliziert: Addieren Subtrahieren ==> 4 Fälle.)
- 1.. **Einerkomplement**, hierbei wird für eine negative Zahl die Kodierung der entsprechenden positiven Binärzahl stellenweise invertiert (Nachteil: die Null hat zwei Repräsentationen, es gibt also -0 und +0 und die Arithmetik ist kompliziert)
2. **Zweierkomplement** (auch echtes Komplement, 2-Komplement). Nächste Folie Das am häufigsten genutzte Verfahren. Häufig durch Hardware unterstützt!
3. **Versatz: Benutzung eines fest gewählten Bias:** Der Wert einer Zahl e ergibt sich aus der (gespeicherten) nichtnegativen Binärzahl E durch Subtraktion eines festen Biaswertes B , also $e = E - B$.

Beispiel: Vergleich verschiedener Kodierungen

Speicherwort ist 4 Bit lang $\rightarrow 2^4 = 16$ verschiedene Kodewörter

Vorzeichenzahl (1 = negativ, 0 = positiv)

-7 ...	-3	-2	-1	0	+1	+2	+3 ...	+7
1111...	1011	1010	1001	0000	0001	0010	0011 ...	0111
				1000				

Einerkomplement (negative Zahlen durch Invertieren aller Ziffern)

-7 ...	-3	-2	-1	0	+1	+2	+3 ...	+7
1000 ...	1100	1101	1110	0000	0001	0010	0011 ...	0111
				1111				

Zweierkomplement

-8 ...	-3	-2	-1	0	+1	+2	+3 ...	+7
1000...	1101	1110	1111	0000	0001	0010	0011 ...	0111

Versatz (Bias = 7, d.h. +7) (auch Exzess- oder Offset-Code)

-7 ...	-3	-2	-1	0	+1	+2	+3 ...	+8
0000	0100	0101	0110 ...	0111	1000	1001	1010...	1111

Python 3.x ✓

Zweierkomplement

Voraussetzung ist eine feste Stellenzahl, sagen wir im folgenden Beispiel 8.

Negative Zahlen werden mit einer führenden **1** dargestellt und wie folgt kodiert:
Sämtliche Ziffern der entsprechenden positiven Zahl werden invertiert
(Einerkomplement). Zum Ergebnis wird 1 addiert.

Beispiel zur Umwandlung einer negativen Dezimalzahl, hier -4

- 1. Vorzeichen ignorieren und die Ziffer ins Binärsystem konvertieren:
 $4_{(10)} = 00000100_{(2)}$
- 2. Invertieren, da negativ: 11111011 (Einerkomplement)
- 3. Eins addieren: $11111011 + 00000001 = 11111100_{(2)} = -4_{(10)}$

Darstellungsbereich des Zweierkomplementes

Mit n Bits lassen sich Zahlen von $(-2^{n-1} \text{ bis } +2^{n-1} - 1)$ darstellen, also z.B.

- bei 32 Bit: $-2.147.483.648_{(10)}$ bis $+2.147.483.647_{(10)}$
- **Man benötigt eine feste Stellenzahl**
- **Null hat die Kodierung 00...0**
- **Code ist unsymmetrisch (etwas unschön!)**
- die Subtraktion zweier natürlichen Zahlen entspricht der Addition des Zweierkomplements des Subtrahenden: $x - y = x + (-y)$
 - ➔ Eine Subtraktions-Einheit braucht man nicht.
 - ➔ **Standard-Implementierung** in der Rechner-Hardware

Excess-x-Code

(Versatz: Benutzung eines fest gewählten Bias x)

Der **Wert einer Zahl e** ergibt sich aus der (gespeicherten) nichtnegativen Binärzahl E durch Subtraktion eines festen **Biaswertes B**, also $e = E - B$.

Meist wird für die Stellenzahl n $B = 2^{n-1} - 1$ gewählt → etwa gleichgroße Wertebereiche: auch Ex(c/z)ess- 2^{n-1} -Code genannt.

Höchstwertiges Bit = 0: → Zahl ist negativ

Höchstwertiges Bit = 1: → Zahl ist positiv

Null != der Kodierung 0, sondern gleich 2^{n-1} .

Null hat eine ein-eindeutige Kodierung.

Sehr einfach durchzuführen: Vergleich (Größer, Kleiner) und Differenz

Wird bei der **Kodierung des Exponenten in der IEEE 754 – Kodierung** gewählt, aber $E == 0$ und $E == 255$ stellen besondere Zahlen da.

Übersicht

- **Numerische Datentypen**
 - *Ganze Zahlen (Integer) - Repräsentationen*
 - **Literale = Notationen in Python**
 - *Ausdrücke (= Terme) - Operatoren und Operationen*
- **Boolescher Datentyp**
 - *Werte und Darstellung in Python*
 - *Literale = Notationen in Python - Operatoren*
 - *Ausdrücke (= Terme) - Operatoren und Operationen*
 - *Vergleichsoperatoren liefern den Booleschen Datentyp*
- **Zusammenfassung**

Integer in Python

- **Literale** (Syntax in EBNF-Notation, genauer kommt dies später):
- Sogenannte Non-Terminalsymbole werden ersetzt!

integer	::= decimalinteger octinteger hexinteger bininteger
decimalinteger	::= nonzerodigit digit*
nonzerodigit	::= "1"..."9"
digit	::= "0"..."9"
octinteger	::= "0" ("o" "O") octdigit+
hexinteger	::= "0" ("x" "X") hexdigit+
bininteger	::= "0" ("b" "B") bindigit+
octdigit	::= "0"..."7"
hexdigit	::= digit "a"..."f" "A"..."F"
bindigit	::= "0" "1"

Erweiterte BNF (Ersetzungsregeln)

::= wird ersetzt durch
| (vertical bar) definiert Alternativen
***** (star) keine, eine oder mehrere Wiederholungen
+ (plus) eine, oder mehrere
() (round brackets) Gruppierung
[] (square brackets) kein, ein oder mehrere Auftreten (optional)

Integer in Python (bis V3.5)

- **Literale** (Syntax in EBNF-Notation, präziser kommt dies etwas später):
- Sogenannte Non-Terminalsymbole werden ersetzt!

integer	::= decimalinteger octinteger hexinteger bininteger
decinteger	::= nonzerodigit (["_"] digit)* "0"+ (["_"] "0")*
nonzerodigit	::= "1"..."9"
digit	::= "0"..."9"
octinteger	::= "0" ("o" "O") octdigit+
hexinteger	::= "0" ("x" "X") hexdigit+
bininteger	::= "0" ("b" "B") bindigit+
octdigit	::= "0"..."7"
hexdigit	::= digit "a"..."f" "A"..."F"
bindigit	::= "0" "1"

Beispiele für integer

7	-2147483647	0o177	0b10011011
3	7922816251426433759	-0o377	0x100ABCDF
	79228162514264337593543950336476389275		0xdeadbeef
0	0000	-0b101	

```
01 # führende Nullen sind für Dezimalrepräsentationen
   # verboten
0b10211
12A
```

Python **integer** können (fast) beliebig groß (lang) werden.
(solange der Hauptspeicher ihres Rechners reicht ... ;-).

Negative Zahlen werden durch "kleine Ausdrücke" (mit unäre Ops: + - ~) repräsentiert. **Beachten Sie die Schreibweise negativer Binärzahlen!**

Achtung: ab Python 3.6 (Python lebt!)

Nach [PEP 515: Underscores in Numeric Literals](#).

Eine sinnvolle Ergänzung für das Schreiben langer Zahlen:

Anstelle des Punktes in der deutschen Dezimalschreibweise zum Tausender abtrennen, also: Anstatt 46.433.759 schreiben wir in Python:

```
46_433_759 # Underscores verändern den Wert nicht
```

Diese Schreibweise ist auch bei Float-Literalen nutzbar und z.B. bei Binärdaten sehr hilfreich:

```
0b_1001_1011 # Man erkennt sofort 8 Bit! . Underscore können
auch direkt nach dem base specifiers wie 0b stehen
```

Leider wird die EBNF-Notation deutlich unübersichtlicher, weil an vielen Stellen *Underscore* (=_) zugelassen werden muss.

Jetzt mit dem Rechner / dem Python Interpreter wandeln – int!

Bei **Integer** ist das echt toll:

Der Wert eines Integer wird durch `print()` als Dezimalzahl ausgegeben.

▸ Die Funktionen `hex()`, `oct()`, `bin()` liefern Strings.

```
>>> hex(254)
'0xfe'
>>> oct(254)
'0o376'
>>> bin(254)
'0b11111110'
>>> bin(-254)
'-0b11111110'
```

ACHTUNG

Dies sind nicht die exakten
Repräsentationen im
Hauptspeicher, sondern
"les- und interpretierbare"
Vorzeichenbehaftete Reps
(nicht Zweierkomplement!)

Nur am Rande: Ein weiteres Problem (und eine Entscheidung!)

Die **Byte-Reihenfolge (Byte order)**.

Im Speicher eines Rechners sind die Speicherzellen meist in Bytes organisiert und in der Regel auch so adressierbar

Byteorder ... das „NUXI-Problem“ papalapapp das "UNIX-Problem"

Für die Speicherung einer 32-Bit-Binärzahl, hier in Hex-Schreibweise
A4B3C2D1 im Hauptspeicher gibt es verschiedenen Möglichkeiten:

Natürlich ?

	Adresse	X + 3	X + 2	X + 1	X
Little Endian	Inhalt	A4	B3	C2	D1
Big Endian		D1	C2	B3	A4
Middle obso-		B3	A4	D1	C2
Endian let		C2	D1	A4	B3

Dies Problem gibt es in ähnlicher Weise natürlich auch bei Strings, weshalb anstelle von 'UNIX' 'NUXI' herauskommen könnte.

Eine Anekdote dazu:

Auch die Etymologie (Herkunft) dieser kuriosen Bezeichnungen „Endian“ ist nett:

Sie lehnen sich an den satirischen Roman *Gullivers Reisen* ("Gulliver's Travels") von Jonathan Swift (1726) an. Der Streit darüber, ob ein Ei am spitzen oder am dicken Ende aufzuschlagen sei, spaltete die Bewohner von Liliput in zwei verfeindete Lager.

die „Little-Endians“ und die „Big-Endians“

in der deutschen Übersetzung des Buches übrigens „Spitz-Ender“ und „Dick-Ender“ ... aber das sagt in der Fachsprache niemand ;-)

Wo findet man heute was?

- Little Endian: Intel-x86-Prozessoren und auch das Betriebssystem Windows
- **Big Endian**; Power PC (umschaltbar), Motorola-68000-Familie, MIPS Prozessoren, HP-UX
- **Kein Problem**, wenn Sie nur auf einem Rechner arbeiten oder keine Binärdaten austauschen, aber
- im Internet ist das **Big Endian** als **Network Byte Order** festgelegt. Die sogenannte *Host Byte Order* muss ggf. angepasst werden.

Zusammenfassung: Kodierung ganzer Zahlen - integer

- Als Basis benutzt man das **binäre Stellenwertsystem** der Zahlen.
- Umrechnung vom (gebräuchlichen) Dezimalsystem müssen Sie können.
- (Binär zu lange) Zahlen werden gern auch im Oktal (Basis 8) oder Hexadezimalsystem (Basis 16) angegeben.
- Negative Zahlen werden als Integer (Ganzzahl) meist durch das Zweierkomplement repräsentiert (aber auch durch Benutzung eines Bias ==> Floating Point, nächste Woche).
- Die Operationen auf Zahlen (Addition, Subtraktion, Multiplikation, etc.) werden durch die Hardware unterstützt.

Ausdrücke (= Terme) - Operatoren und Operationen

- Für Integer Operanden sind in Python ALLE Operatoren definiert, auch Bitweise-arbeitende Operatoren ($x \ll y$, $x \gg y$, $x \& y$, $x \wedge y$, $x | y$) siehe Programmierhandzettel 1.
- Hieraus können beliebige Ausdrücke (Terme) wie in der Mathematik üblich erzeugt werden.
- Auch Runde Klammern stehen zur Verfügung um die Auswertereihenfolge zu steuern.
- Das Ergebnis ist auch wieder ein Integer.

Operationen auf Zahlen

höchste Priorität

geordnet nach Vorrangregeln

Auszug aus Programmierhandzettel 1

geringste Priorität

Operation	Beschreibung
$+x$, $-x$, $\sim x$	Einstellige Operatoren Invertiere x bitweise (nur Integer)
$x ** y$	Exponential-Bildung x^y (Achtung: rechts-assoziativ)
$x * y$ x / y $x \% y$ $x // y$	Multiplikation (Wiederholung) Division Modulo (-Division) = Division mit Rest Restlose Division ²⁾
$x + y$ $x - y$	Addition (Konkatenation) Subtraktion
$x \ll y$, $x \gg y$	Bitweises „Schieben“ (nur bei Integer)
$x \& y$	Bitweises Und (nur bei Integer)
$x \wedge y$	Bitweises exklusives Oder (nur bei Integer)
$x y$	Bitweises Oder (nur bei Integer)

Klammerung

Wie in der Mathematik üblich, kann man die Auswertereihenfolge durch Klammerung (...) beeinflussen. Zugelassen sind allerdings **nur runde Klammern**. Diese können aber beliebig geschachtelt werden. Hierzu einige Beispiele:

```
>>> 1+2 * 3+4
11
```

```
>>> (1+2) * (3+4)
21
```

```
>>> ((1+2) * 3) + 4
13
```

Weitere Beispiele ("Spezialitäten"):

```
>>> ~4 # ~ ist der unäre Invertierungsoperator
-5
```

Ergebnis entspricht in der Repräsentation dem Einer-Komplement von 4, aber in Python als Zweier-Komplement bei der Ausgabe interpretiert, ergibt dies $-4-1 = -5$, oder

4	0 . . 0 0100	im Speicher
~4	1 . . 1 1011	im Speicher (Einer-Komp.)
	1 . . 1 1010	Eins subtrahiert
	0 . . 0 0101	Invertiert
-5		Ausgabe

```
>>> 2**3**2
512
```

** ist rechtsassoziativ, d.h. äquivalent zu $2 ** (3 ** 2)$

Übersicht

- **Numerische Datentypen**
 - Ganze Zahlen (*Integer*)
 - *Literale* = Notationen in Python - Operatoren
 - *Ausdrücke* (= *Terme*) - Operatoren und Operationen
- **Boolescher Datentyp**
 - *Werte und Darstellung in Python*
 - *Literale* = Notationen in Python - Operatoren
 - *Ausdrücke* (= *Terme*) - Operatoren und Operationen
 - *Vergleichsoperatoren liefern den Booleschen Datentyp*
- **Der "Datentyp" None**
- **Zusammenfassung**

Boolescher Datentyp (Boolean)

- Das kleinste Speicherelement eines Computers ist eine Speicherstelle, ein „Bit“, deren Wertebereich durch zwei Zustände 0 und 1 gegeben ist.
- Im Folgenden bezeichnen wir den Wert 0 als **False** und 1 als **True**. Oder auch in ihrer deutschen Übersetzung mit *Falsch* und *Wahr*.
- Der zugehörige Datentyp wird als „Boolesch“ (nach George Boole), oder Englisch „Boolean“ bezeichnet

Operatoren auf dem Datentyp Boolean

definiert sind u.a.: **and**, **or**, **not** (also ein vollständiger Operatorensetz), siehe auch DisMod.

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Boolescher Datentyp in Python (1)

- ▶ **Besonderheit in Python:** Jede Variable (jeder Ausdruck) kann boolesch „interpretiert“ werden und zwar:
 - ▶ **False** für numerische Datentypen, wenn der **Wert 0** vorliegt
bei String (sequentiellen) Datentypen die Länge 0 besitzt
 - ▶ **True** für numerische Datentypen, wenn ein **Wert 0** vorliegt
bei String (sequentiellen) Datentypen eine Länge 0 hat
- ▶ **Der Typ Boolean ist in Python ein Subtyp** (später sagen wir „Unterklasse“) **von Integer**.
- ▶ Das ist vielfach bequem, hat aber verschiedene Konsequenzen.

Boolescher Datentyp in Python (2)

Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 10:55:48) [MSC v.1600 32 bit (Intel)] Type "copyright", "credits" or "license()" for more

```
>>> a = False
```

```
>>> b = not a
```

```
>>> a and b
```

```
False
```

```
>>> a or b
```

```
True
```

```
>>> a + b
```

```
1
```

```
>>> a - b
```

```
-1
```

```
>>> c, d = -1, 0
```

```
>>> c and d
```

```
0
```

```
>>> c or d
```

```
-1
```

```
>>> c & d
```

```
0
```

```
>>> c | d
```

```
-1
```

Regeln hierfür
finden Sie im Skript!

Sehr Tricky!

Don't do that!

Genauer: Wann liefert eine coercion False?

- Die Konstanten None and False haben den Wahrheitswert False.
- Der Wert Null eines jeden numerischen Datentyps: 0, 0.0
- Leere Sequenzen oder Kollektionen:
"", oder für später (), [], {}, set(), range(0)
- Explizit wandeln erfolgt mit der Funktion: bool()

Beispiele: >>> bool("")

```
False
```

```
>>> bool(1)
```

```
True
```

Wichtig: Vergleichsoperatoren liefern den Boolesche Datentypen {True, False}

```
x >= y, x <= y
x == y, x != y
```

```
x is y, x is not y
```

```
x in s, x not in s
```

Test auf Identität

Tests auf „Enthalten sein“ in Sequenzen (kommt später)

Beachten Sie den Unterschied zwischen

$X == Y$ und $X \text{ is } Y$

Im ersten Fall wird verglichen ob X und Y den gleichen Wert haben, im zweiten Fall, ob sie dasselbe Objekt sind (also, die Identität gleich ist, d.h. in derselben Speicherzelle steht / unter derselben Adresse gespeichert ist).

Übersicht

- **Numerische Datentypen**
 - Ganze Zahlen (Integer)
 - Literale = Notationen in Python - Operatoren
 - Ausdrücke (= Terme) - Operatoren und Operationen
- **Boolescher Datentyp**
 - Werte und Darstellung in Python
 - Literale = Notationen in Python - Operatoren
 - Ausdrücke (= Terme) - Operatoren und Operationen
 - Vergleichsoperatoren liefern den Booleschen Datentyp
- **Der "Datentyp" None**
- **Zusammenfassung**

None – Der "Nichts-Typ"

Durch den Wert **None** wird in Python symbolisiert, dass diese Variable keinen Wert hat.

Dies kann hilfreich sein, wenn man eine Variable definieren, ihr aber erst späteren einen konkreten Wert zuweisen will; ein anderer Anwendungsfall wäre die Rückgabe eines **Ergebniswerts bei einer erfolglosen Suche**.

None ist ein **Singleton**, es gibt also stets nur eine Instanz dieses Typs "NoneType" und diese hat den Wert None.

Funktionen geben 'None' zurück, wenn am Ende kein 'return value' oder nur 'return' steht – also ähnlich 'void' in C oder Java

Viele Funktionsparameter haben 'None' als Defaultwert, der angenommen wird, wenn der Parameter nicht als Argument übergeben wird.

Wichtig: Vergleiche mit None

- vergleicht man **None** mit irgendeinem anderen Typ und Wert, so ergibt dies **immer False**.
- Achtung:

```
>>> 5 == None
False
```

```
>>> None is None
True
```

```
>>> 'Mist' == None
False
```

```
>>> None is not None
False
```

```
>>> None == None
True
```

PEP 8 - [Programming Recommendations](#): "Comparisons to singletons like None should always be done with **is** or **is not**, never the equality operators."

Warum `x is None` und nicht ~~`x == None`~~

- ▶ But does it really matter?
- ▶ `is None` ist etwas schneller als `== None`
- ▶ `is None` ist etwas sicherer (falls ein anderer Programmierer den `==`-Operator modifiziert haben sollte ... kommt aber wohl eher selten vor.)
- ▶ Wir übernehmen diese PEP 8 Programming Recommendation NICHT in unser Programmierhandbuch.

Juhuuu! .. Schon fertig für heut' !

Zusammenfassung – Das müssen Sie können!

Für natürliche Zahlen:

- Dualsystem \leftrightarrow Dezimalsystem
- Umkodieren Oktal \leftrightarrow Dual, Hexadezimal \leftrightarrow Dual
und auch: Hexadezimal \leftrightarrow Oktal
- Oktal oder Hexadezimal \leftrightarrow Dezimalsystem
- auch Addieren, Subtrahieren, Multiplizieren im Dualsystem

Für ganze Zahlen: Einerkomplement, Zweierkomplement,
Exzess-x-Kodierung

Literal-Schreibweisen und Integer-Operatoren und –Funktionen

Bedeutung und Benutzung von **None**

Fragen und (hoffentlich) Antworten

Zusammenfassung

Viele Details, ich weiß!

Als erstes das Quiz machen:

Q03: Zahlendarstellungen Integer und Bool

Jetzt müssen Sie üben, im Kopf und auf Papier (PRG1).
Und auch mit dem Interpreter (EPR).

Ausblick ... nächsten Freitag

Der Datentyp String

... und, danke für Ihre Aufmerksamkeit!