706.088 INFORMATIK 1

ÜBERBLICK

- Datenstrukturen
- Zahlensysteme
- Bit-Operatoren

DATENSTRUKTUREN

DATENSTRUKTUREN

Dienen dem systematischen Ablegen und Aufrufen von Daten.

- Speicherung
- Organisation
- Effizienz
- regelt Art des Zugriffs

DATENSTRUKTUREN

BEISPIELE

- Tupel
- Array
- assoziatives Array (Dictionary)
- Warteschlange (FiFo)
- Stapelspeicher (LiFo)
- Graphen
- Bäume (Binärbaum)

ARRAY

```
a = [1,"b","III",4,5]
a[0]
a[2]
a[5]
```

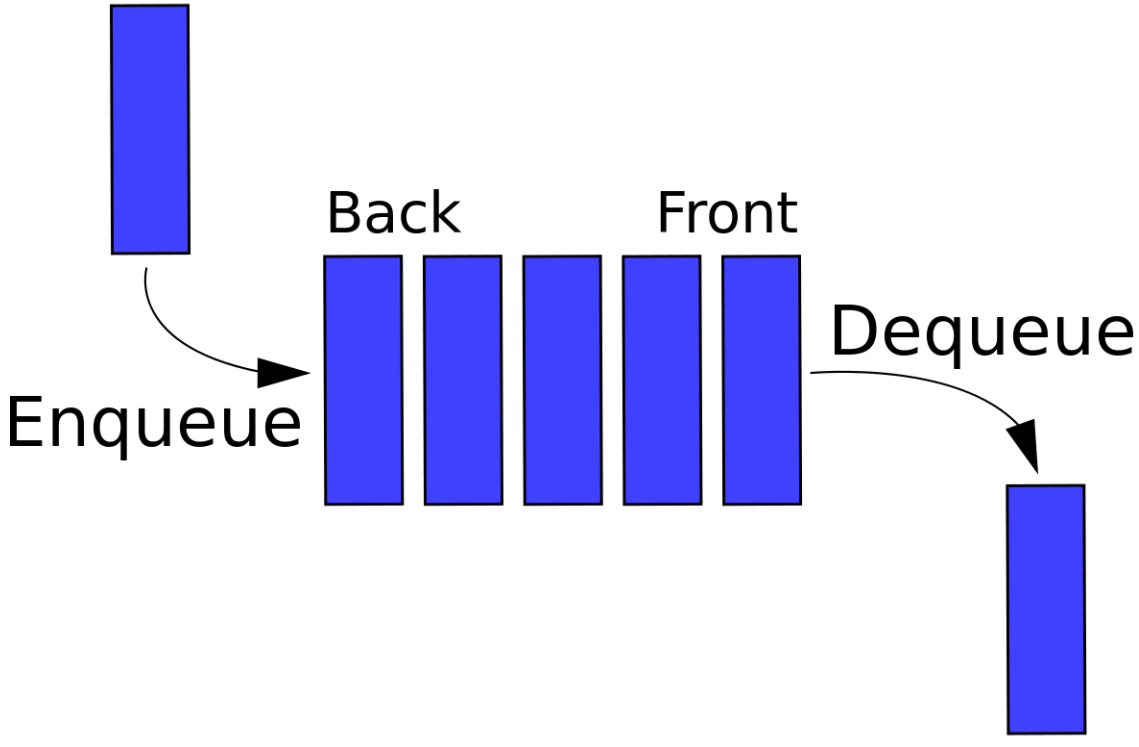
```
1
'III'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

DICTIONARY

```
d = {"element1": 1, "myelement": "python", "python": 3.5}
d['element1']
d['python']
d['myelement']
```

```
1
3.5
'python'
```

WARTESCHLANGE (FIFO)



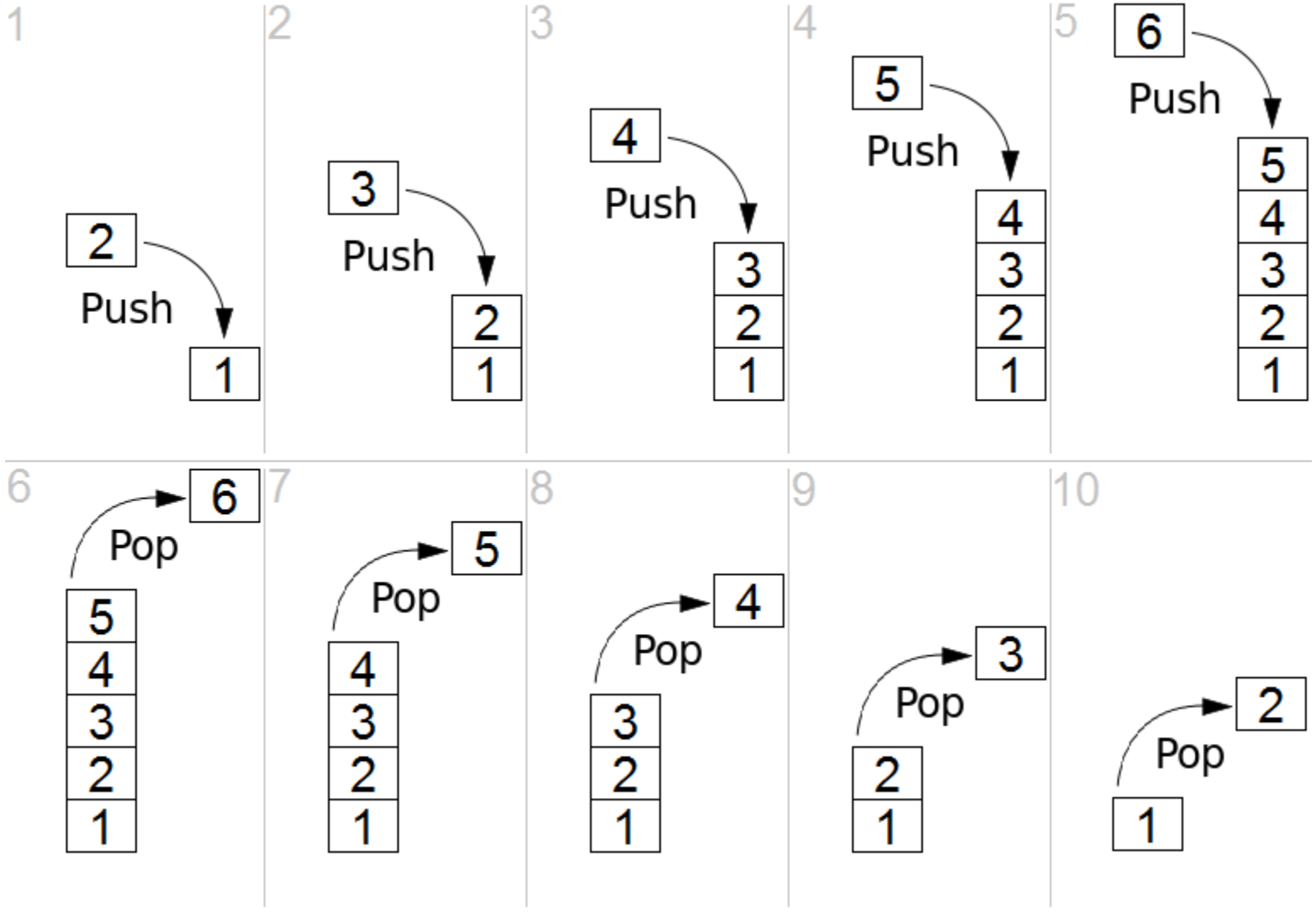
WARTESCHLANGE (FIFO)

```
import queue
q = queue.Queue()
q.put(1)
q.put(2)
q.put("last")

q.get()
q.empty()
q.get()
q.get()
q.empty()
```

```
1
False
2
'last'
True
```

STAPELSPEICHER (LIFO)



STAPELSPEICHER (LIFO)

```
import queue
q = queue.LifoQueue()
q.put(1)
q.put(2)
q.put("last")

q.get()
q.empty()
q.get()
q.get()
q.empty()
```

```
'last'
False
2
1
True
```

GRAPHEN

- bestehen aus **Kanten** und **Knoten**
- Eigenschaften:
  - gerichtete Graphen: Kanten haben Richtung
  - ungerichtete Graphen können in beide Richungen 'begangen' werden.
  - gewichtet: Kanten haben Gewicht
  - zyklisch: Weg von Knoten A zurück zu A ohne eine Kante mehrfach zu gehen

GRAPHENOPERATIONEN

- Hinzufügen eines Knotens (mit oder ohne Kanten)
- Entfernen des Knotens A, entfernt auch alle Kanten zu A
- Es gibt keine Kanten ohne Knoten an beiden Enden

BÄUME

Sonderform von Graphen

- Bäume: zusammenhängende, azyklische Graphen
  - gerichtet
  - ungerichtet
  - Binärbaum: maximal 2 Nachkommen pro Knoten

ZAHLENSYSTEME

DEZIMAL

› **Dezimal**-system: Basis 10 (std. Integer)

Zahlen	Präfix
0-9	"" (keiner)

```
>>> i = 123
>>> i
123
>>> f = int("99")
>>> f
99
>>>
```

Zahlen HEX Präfix

› **Hexadezimal**-system: Basis 16

Zahlen	Präfix
0-9, A-F	"0x"

```
>>> h = 0xA
>>> h
10
>>> h2 = 0xFF
>>> h2
255
```

OKTAL

› **Oktal**-system: Basis 8

Zahlen	Präfix
0-7	"0o" (Null-O)

```
>>> o = 0o7
>>> o
7
>>> o2 = 0o144
>>> o2
100
```

BINÄR

› **Binär**-system oder **Dual**-system: Basis 2

Zahlen	Präfix
0,1	"0b"

```
>>> b = 0b101
>>> b
5
>>> b2 = 0b111111
>>> b2
63
```

BINÄRSYSTEM

- Basis für Computer
- Reduktion auf 2 Zustände
  - 0: kein Strom, Spannung
  - 1: Strom, Spannung

BINÄRSYSTEM

- Einzelne Stelle heisst: Bit (**Bi**nary Digi**t**)
- rechteste Stelle: Least Significant Bit (LSB)
- linkeste Stelle: Most Significant Bit (MSB)

BINÄRSYSTEM

```
# Dezimal
i = 2*10**3 + 0*10**2 + 1*10**1 + 6*10**0 # 2016

# Binär
b = 1*2**3 + 1*2**2 + 0*2**1 + 1*2**0 # 13
```

BINÄRSYSTEM UMRECHNUNG (N=2)

- x(10) → x(n):
  - x/n ⇒ y, Rest z
  - z an Stelle 0
  - y ⇒ x wenn y != 0
  - von Vorne für Stelle 1, 2, …
  - wenn y = 0 fertig.

BINÄRSYSTEM UMRECHNUNG

25(10) → binär:

- 25/2 ⇒ 12, Rest 1
- 1 an Stelle 0 (LSB)
- 12/2 ⇒ 6, Rest 0
- 0 an Stelle 1
- 6/2 ⇒ 3, Rest 0
- 0 an Stelle 2
- …

- 3/2 ⇒ 1, Rest 1
- 1 an Stelle 3
- 1/2 ⇒ 0, Rest 1
- 1 an Stelle 4
- fertig: 11001

RECHNEN IM BINÄRSYSTEM

Grundregeln für Addition:

0 + 0	= 0
0 + 1	= 1
1 + 0	= 1
1 + 1	= 0 (1 Übertrag)
1 + 1 + 1	= 1 (1 Übertrag)

BINÄRE ADDITION

+	
0101101	= 45
0110110	= 54
1111	= Übertrag
1100011	= 99

BINÄRE SUBTRAKTION

Computer führt Addition auf Subtraktion zurück: möglich durch Zweier-Komplementbildung

- Darstellung von negativen Zahlen:
  - MSB trägt Information über Vorzeichen
- Limit an darstellbaren Zahlen:
  - `s`: verfügbaren Bits
  - Kleinste darstellbare Zahl: `-2**(s-1)`
  - Größte darstellbare Zahl: `2**(s-1)-1`

POSITIVE UND NEGATIVE BINÄRZAHLEN

bin	dec	bin	dec
0000	0	1000	-8
0001	1	1001	-7
0010	2	1010	-6
0011	3	1011	-5
0100	4	1100	-4
0101	5	1101	-3
0110	6	1110	-2
0111	7	1111	-1

ZWEIER-KOMPLEMENTDARSTELLUNG

- ist MSB 1, ist die Zahl negativ
- Schritt 1: alle Bits invertieren
- Schritt 2: zum Schluss 1 addieren

5(10) → -5	**0101(2)**	**-5(10) → 5**	**1011(2)**
Schritt 1: invertieren	1010		0100
Schritt 2: +1	1011		0101

SUBTRAKTION

6(10) - 2(10) → **6(10) + -2(10)**

Entspricht einer Addition mit dem Zweier-Komplement

- Was geschieht mit Überläufen?
  - werden verworfen

6(10)	0110	
-2(10)	1110	
	1 0100	= 4

3*2 MULTIPLIKATION 2018

kann **manchmal** durch Verschieben der Bits nach links durchgeführt werden

› Multiplikation mit 2**n: (Shiften um n Bits)

3*2		**20*8**	
3	011	**20**	010100
2 (2**1)	10	**8** (2**3)	001000
6	110	**160**	010100000

AUSNAHMEN MULTIPLIKATION

Bei Multiplikation mit Zahlen ungleich 2**n

```
13(10) *   5(10)
1101   * 101

 1101
+ 0000
+  1101
1000001  = 65
```

DIVISION

4//2 = 2 24//8 = 3

kann **manchmal** durch Verschieben der Bits nach rechts durchgeführt werden

- Division mit 2**n: (Shiften um n Bits)

4//2 = 2		**24//8 = 3**	
4	100	**24**	11000
2 (2**1)	10	**8** (2**3)	01000
2	10	**3**	11

AUSNAHMEN DIVISION

Bei Division mit Zahlen ungleich 2**n

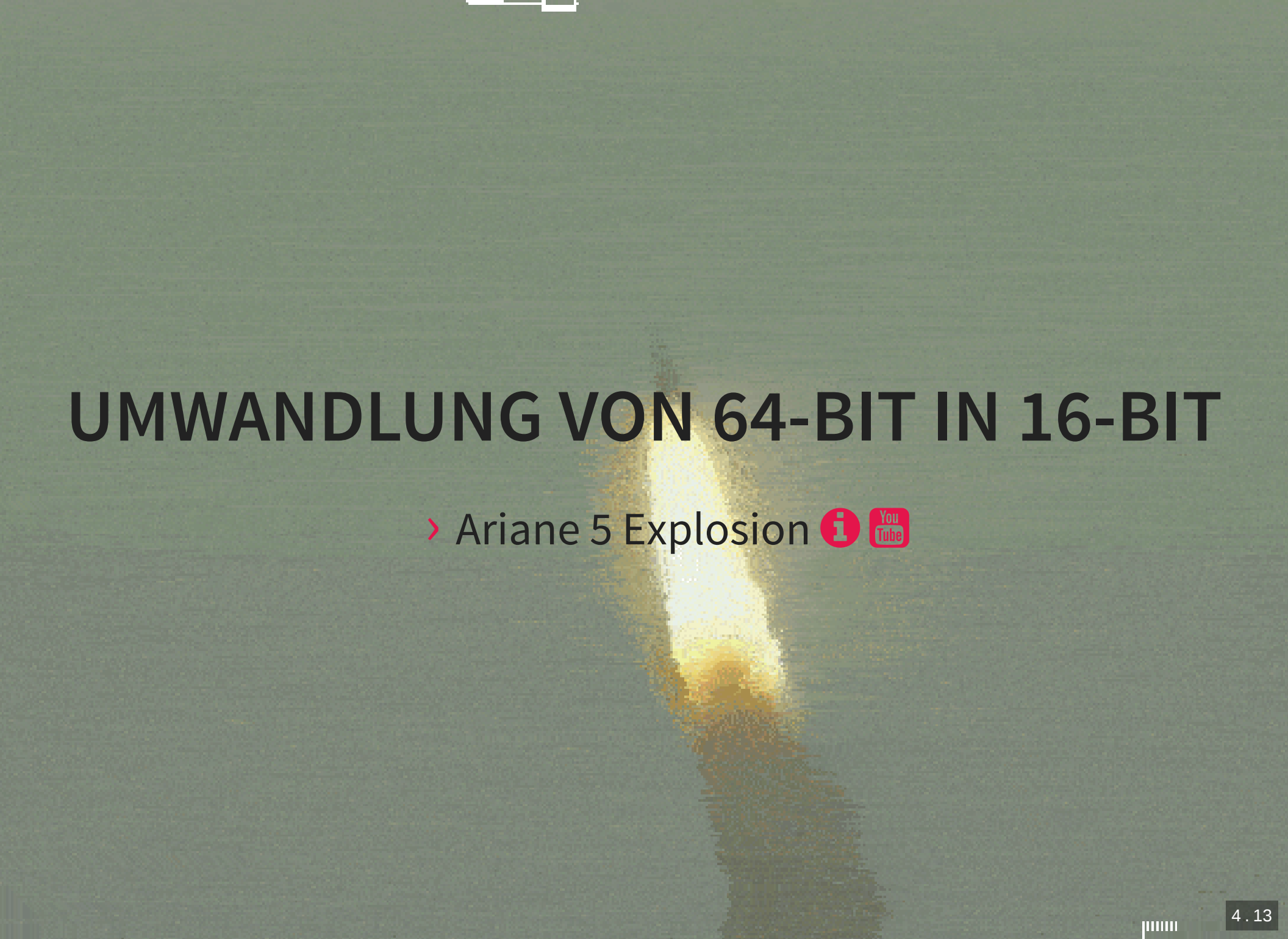
```
25(10) //   5(10)
   11001   // 101  = 101
   110               ^^^
  -101   >----------/ ||
---------             ||
   110                ||
 +1010  Komplement    ||
 +   1                ||
---------             ||
 10001                ||
     10 >-------------/|
     101               |
    -101 >-------------/
---------
     101
   +1010  Komplement
   +   1
---------
   10000 ---> kein Rest
```

LIMITIERUNG DER DARSTELLBAREN ZAHLEN

- Gängige Computer haben 32-bit oder 64-bit Architektur:
  - 32-bit: max positive Zahl `2**32 - 1`
  - 64-bit: max positive Zahl `2**64 - 1`
  - Python kann in Version 3 gut mit langen Zahlen umgehen.
  - Grundproblem bleibt generell für Computer bestehen.

```
>>> i = 0b11111111111111111111111111111111
>>> i
4294967295
>>> i.bit_length()
32
>>> j = i+1
>>> j.bit_length()
33
>>> bin(j)
'0b100000000000000000000000000000000'
>>> i64 = 0b1111111111111111111111111111111111111111111111111111111111111111
>>> i64.bit_length()
64
>>> i64
18446744073709551615
```

UMWANDLUNG VON 64-BIT IN 16-BIT



› Ariane 5 Explosion

BITOPERATOREN

BITOPERATOREN

Operatoren für Binärdarstellung

Operator	Zuweisung	Ergebnis
~x		bitweises Komplement (Einerkomplement) (= -x-1)
~x+1		Zweierkomplement (= -x)
x & y	x &= y	bitweises UND (AND)
x | y	x |= y	bitweises ODER (OR)
x ^ y	x ^= y	bitweises ausschließendes ODER (XOR)
x << n	x <<= n	shiften von x um n Bit nach links
x >> n	x >>= n	shiften von x um n Bit nach rechts

BITOPERATOREN

```
>>> a = 0b1001
>>> b = 0b0110
>>> bin(a | b)
'0b1111'
>>> bin(a & b)
'0b0'
>>> bin(a ^ b)
'0b1111'
>>> bin(~a)
'-0b1010'
>>> bin(~a & b)
'0b110'
```

BITOPERATOREN

```
>>> a = 0b1001
>>> b = 0b0110
>>> b >>= 1
>>> b
3
>>> bin(b)
'0b11'
>>> bin(a ^ b)
'0b1010'
>>> a << 2
36
>>> bin(a << 2)
'0b100100'
```

FRAGEN?

NÄCHSTES MAL

2016-11-23 16:00