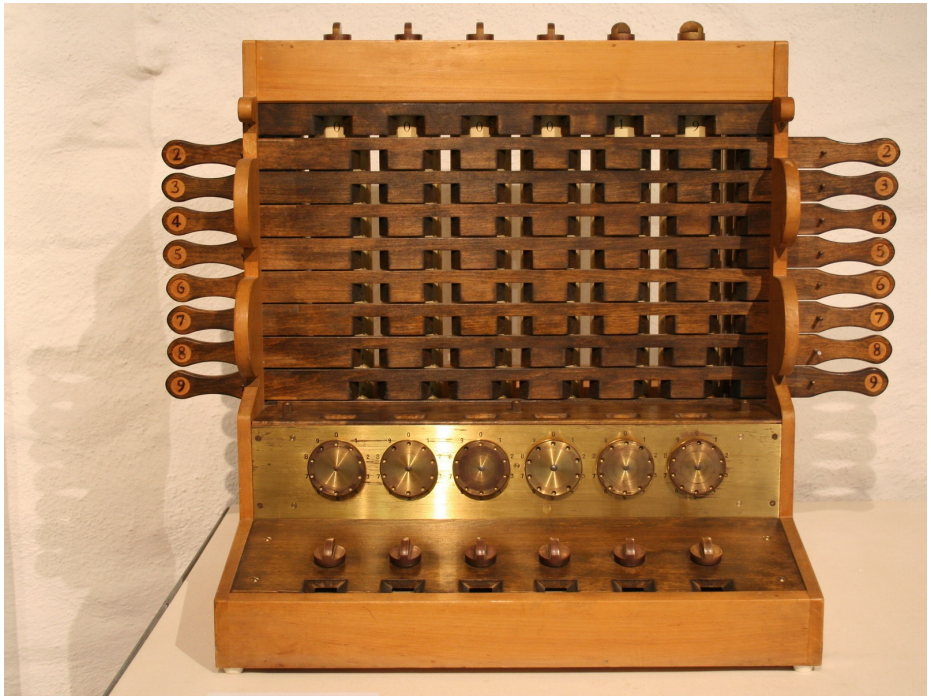
706.088 INFORMATIK 1

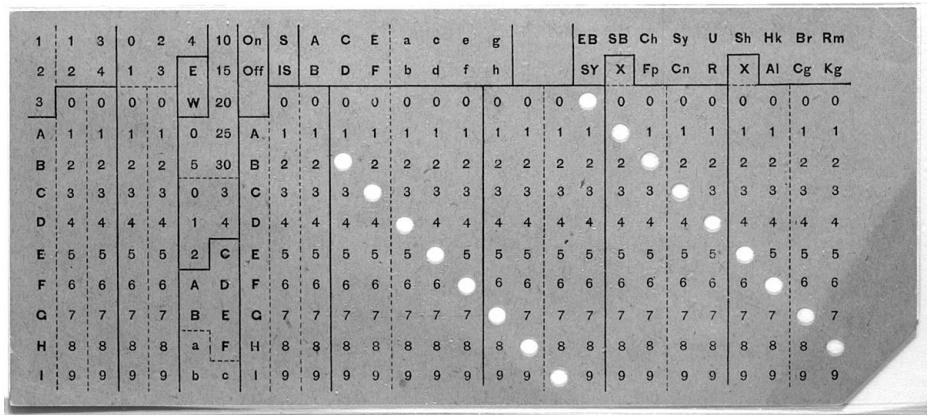
TUPEL, LISTEN,...; STRING SLICING,...; LIST COMPREHENSIONS, LAMBDA FUNCTIONS

WIEDERHOLUNG

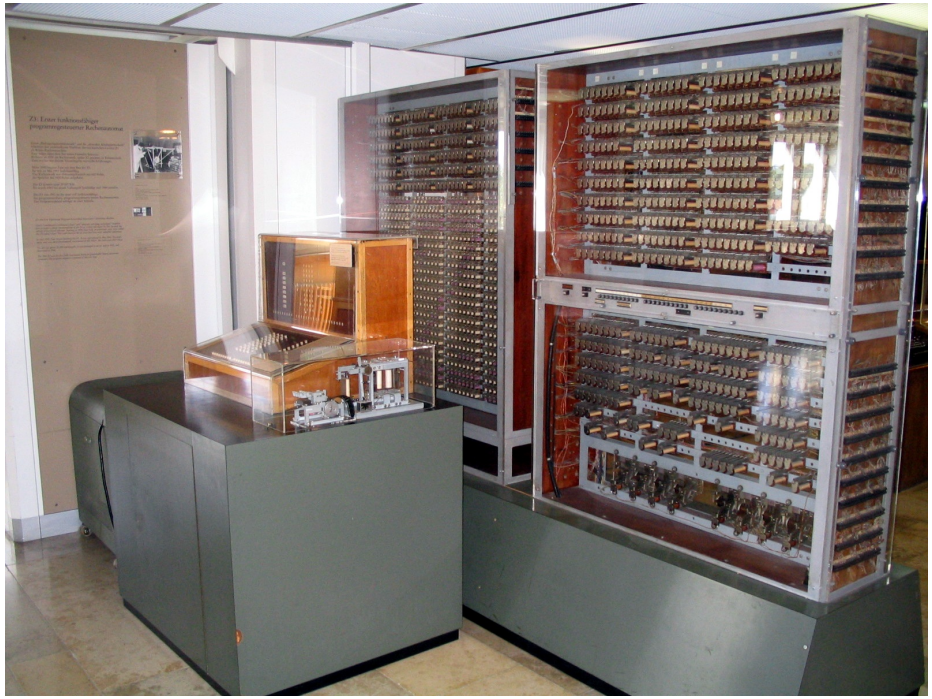
- Geschichte:
  - Mechanische, Elektronische Rechenmaschinen



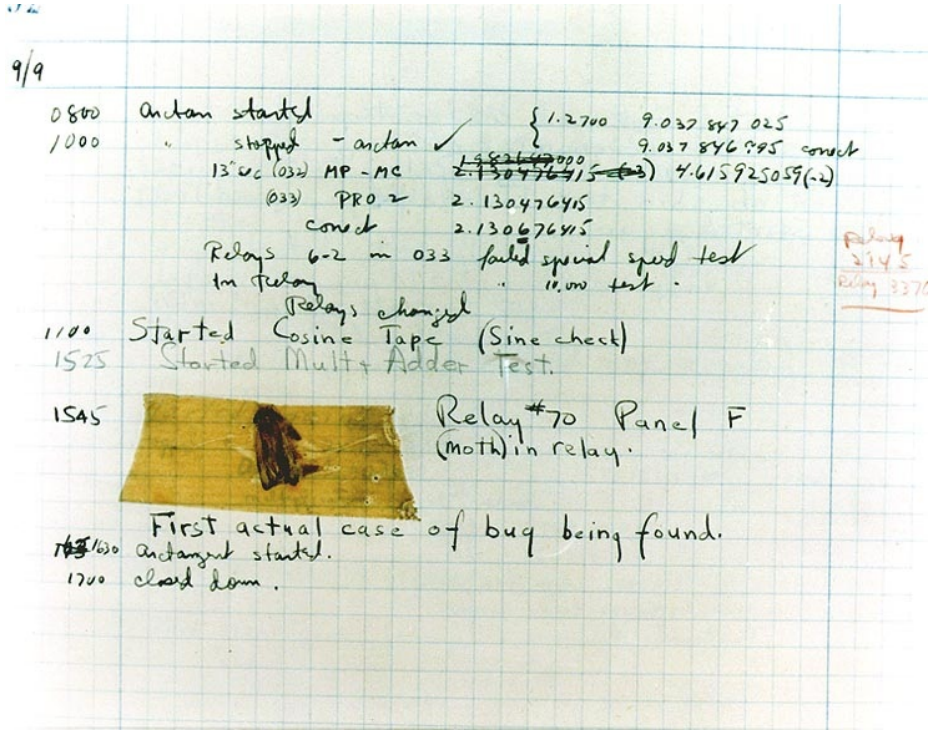
Schickard'sche Rechenmaschine (1623 n. Chr)
By Herbert Klaeren - Transferred from [1], CC BY-SA 3.0, Link



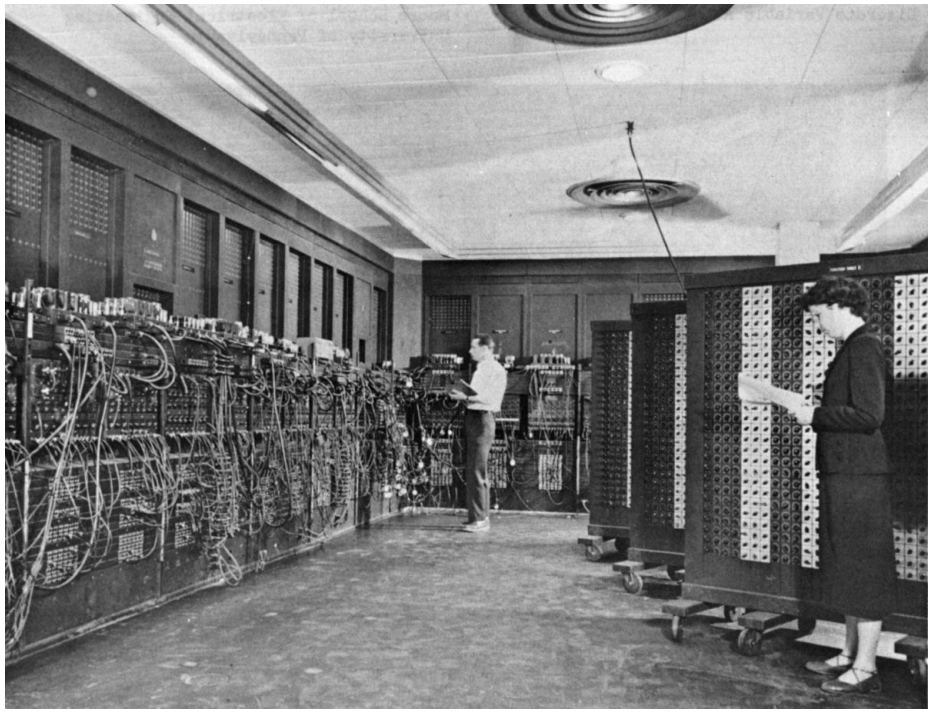
Lochkarte
By Unknown - Library of Congress http://memory.loc.gov/mss/mcc/023/0008.jpg, Public Domain, https://commons.wikimedia.org/w/index.php?curid=30538485



Z3 im Deutschen Museum
Von Venusianer aus der deutschsprachigen Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3632073



By Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988. - U.S. Naval Historical Center Online Library Photograph NH 96566-KN, Public Domain, https://commons.wikimedia.org/w/index.php?curid=165211



Eniac

Von Unbekannt - U.S. Army Photo, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=55124

Mailüfterl, techn. Museum Wien;
Von Dr. Bernd Gross - Eigenes Werk, CC-BY-SA 4.0, Link

ZAHLENSYSTEME

Basis: 10, 2 (`0b`), 8 (`0o`), 16 (`0x`)

Python default: 10

```
int(str(100),2)
```

```
4
```

```
>>> help(int)
int(x=0) -> integer
int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no
arguments are given.  If x is a number, return x.__int__().
For floating point numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a
string, bytes, or bytearray instance representing an integer
literal in the given base.  The literal can be preceded by
'+' or '-' and be surrounded by whitespace.  The base
defaults to 10. Valid bases are 0 and 2-36. Base 0 means to
interpret the base from the string as an integer literal.
>>> int('0b100', base=0)
4
```

```
int(str(0b100),0)
```

```
4
```

BITOPERATOREN

```
def ip_to_int(ip):
    '''converts classic IP representation to int IP'''
    ip_parts = []
    for element in ip.split('.'):
      ip_parts.append( int(element) )
    int_ip = ip_parts[0] << 24
    int_ip += ip_parts[1] << 16
    int_ip += ip_parts[2] << 8
    int_ip += ip_parts[3]
    return int_ip
```

```
def convert_ip(ip):
    '''converts an int IP to classic IP `WWW.XXX.YYY.ZZZ`'''
    classic_ip = [None]*4
    classic_ip[0] = (ip & (255 << 24)) >> 24
    classic_ip[1] = (ip & (255 << 16)) >> 16
    classic_ip[2] = (ip & (255 << 8)) >> 8
    classic_ip[3] = (ip & (255))
    str_ip = []
    for element in classic_ip:
      str_ip.append( str(element) )
    return ".".join(str_ip)
```

```
def is_same_subnet(ip1, ip2, sub=subnet_24):
    '''returns True if ip1 and ip2 are on the same subnet'''
    if (ip1 & sub) == (ip2 & sub):
        return True
    return False

def device_number(ip, sub=subnet_24):
    '''returns address of device on the subnet'''
    return ((ip & sub) ^ ip)
```

SEQUENTIELLE DATENTYPEN

`list, tuple, str, byte, bytearray`

LISTEN

`list`: listet beliebige Instanzen; veränderlich

TUPEL

`tuple`: listet beliebige Instanzen; **unveränderlich**

STRING

`str`: Text, Sequenz von Buchstaben,
auch Sonderzeichen; **unveränderlich**

BYTES

`bytes`: Binärdaten, Sequenz von Bytes (8-Bit);
unveränderlich

BYTEARRAY

`bytearray`: Binärdaten, Sequenz von Bytes (8-Bit); veränderlich

SLICING

```
s = "This is a test string"
s[0] # T
s[8] # a
s[0:3] # Thi
s[8:14] # a test
s[0:14:3] # Tss s
```

SLICING

```
s = "This is a test string"
s[-2] # n
s[-2:] # ng
s[-11::3] # tttn
s[:-5:] # This is a test s
```

SLICING

```
s = "This is a test string"
s[::-1] # gnirts tset a si sihT
s[:-5:-1] # gnir
s[-16::-1] # i sihT
s[-10::-2] # e  ish
s[-5:-12:-2] # t st
s[-13:12] # a te
```

LIST COMPREHENSIONS

Erstellen einer neuen Liste, aus bestehender Liste, nach bestimmten Regeln.

LIST COMPREHENSIONS

```
my_list = [1,2,3,4,5,6,7,8,9]
[x**2 for x in my_list]
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

LIST COMPREHENSIONS

```
my_list = [1,2,3,4,5,6,7,8,9]
[x**2 for x in my_list if x % 2 == 0]
```

```
[4, 16, 36, 64]
```

```python
def ip_to_int(ip):
    '''converts classic IP representation to int IP'''
    # ip_parts = []
    # for element in ip.split('.'):
    #   ip_parts.append(int(element))
    ip_parts = [int(x) for x in ip.split('.')]

    int_ip = ip_parts[0] << 24
    int_ip += ip_parts[1] << 16
    int_ip += ip_parts[2] << 8
    int_ip += ip_parts[3]
    return int_ip
```

```
def convert_ip(ip):
    '''converts an int IP to classic IP `WWW.XXX.YYY.ZZZ`'''
    classic_ip = [None]*4
    classic_ip[0] = (ip & (255 << 24)) >> 24
    classic_ip[1] = (ip & (255 << 16)) >> 16
    classic_ip[2] = (ip & (255 << 8)) >> 8
    classic_ip[3] = (ip & (255))
    # str_ip = []
    # for element in classic_ip:
    #   str_ip.append( str(element) )
    str_ip = [str(x) for x in classic_ip]

    return ".".join(str_ip)
```

```
my_list1 = ["A", "B", "C"]
my_list2 = ["D", "E", "F"]
[(a,b) for a in my_list1 for b in my_list2 ]
```

```
[('A', 'D'), ('A', 'E'), ('A', 'F'), ('B', 'D'),
 ('B', 'E'), ('B', 'F'), ('C', 'D'), ('C', 'E'),
 ('C', 'F')]
```

```
my_list1 = ["A", "B", "C"]
my_list2 = ["D", "E", "F"]
my_list3 = ["G", "H", "I"]
[(a,b,c) for a in my_list1 for b in my_list2 for c in my_list3]
```

```
[('A', 'D', 'G'), ('A', 'D', 'H'), ('A', 'D', 'I'), ('A', 'E', 'G'),
 ('A', 'E', 'H'), ('A', 'E', 'I'), ('A', 'F', 'G'), ('A', 'F', 'H'),
 ('A', 'F', 'I'), ('B', 'D', 'G'), ('B', 'D', 'H'), ('B', 'D', 'I'),
 ('B', 'E', 'G'), ('B', 'E', 'H'), ('B', 'E', 'I'), ('B', 'F', 'G'),
 ('B', 'F', 'H'), ('B', 'F', 'I'), ('C', 'D', 'G'), ('C', 'D', 'H'),
 ('C', 'D', 'I'), ('C', 'E', 'G'), ('C', 'E', 'H'), ('C', 'E', 'I'),
 ('C', 'F', 'G'), ('C', 'F', 'H'), ('C', 'F', 'I')]
```

LAMBDA FUNKTIONEN

ANONYME FUNKTIONEN

- müssen nicht benannt werden

```
lambda x: x**2
sq = lambda x: x**2
sq(3) # 9
(lambda x: x**2)(3) # 9
```

```
def convert_ip(ip):
    '''converts an int IP to classic IP WWW.XXX.YYY.ZZZ'''
    classic_ip = [None]*4
    classic_ip[0] = (ip & (255 << 24)) >> 24
    classic_ip[1] = (ip & (255 << 16)) >> 16
    classic_ip[2] = (ip & (255 << 8)) >> 8
    classic_ip[3] = (ip & (255))

    # str_ip = [str(x) for x in classic_ip]
    str_ip = list( map(lambda x: str(x), classic_ip) )

    return ".".join(str_ip)
```

FRAGEN?