

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

### **V05 Elementare Datentypen - Teil 2 Zeichenketten (String)**

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

### **Wichtige Hinweise für Physiker:**

- Vom PA Physik: Als Nebenfach brauchen Sie 12 "benotete" CP.
  - Das Modul PRG1 hat aber nur 11 CP und PS2 ist unbenotet.
- Nach Anfrage folgende erfreuliche Mitteilung vom PA Physik (Frau Hölscher):
- "Lieber Herr Professor Krömker,  
das können wir gerne so machen. Wenn 11 CP benotet sind, können  
die 2 CP von PS2 unbenotet bleiben."

## Wichtige Hinweise für BA Informatik und Bioinformatik

- ▶ Bitte dringend zur BA-Prüfung anmelden!
- ▶ "Also die sogenannte „Zulassung zur Bachelor-Prüfung“.
- ▶ **Jeder soll das bitte sehr machen!!!**
- ▶ ... also bitte ins PA (Montag!), Formular holen ... ausfüllen ... und abgeben.
- ▶ Kostet euch nichts bringt dem Ifl aber Geld. Sorry.

## Aufpassen bitte!

- ▶ EPR 3 ist eine Aufgabe, die im Zweierteam zu erledigen ist.
- ▶ **Ihre Aufgaben**
  1. Eine Partner\*in Ihrer Übungsgruppe finden!  
Es gibt keine Ausnahmen! Ungerade Anzahl von Teilnehmern in der Übungsgruppe → Tutor ansprechen!
  2. Für die Abgabe die \_\_author\_\_-Variable richtig setzen.
  3. Dann **gemeinsam** die Aufgabe bearbeiten!

## Vorgehen (1)

- ▶ **Pair Programming** ist eine zentrale Technik aus dem eXtreme Programming (XP) – einem Beispiel für agile Techniken.
- ▶ Beim Pair Programming sitzen zwei Entwickler\*innen gleichberechtigt an einem Rechner und arbeiten gemeinsam an einer Aufgabe.
- ▶ Die zwei Entwickler nehmen unterschiedliche Rollen ein, welche oft mit „**Pilot**“ und „**Navigator**“ bezeichnet werden. Der „Pilot“ schreibt den Code, während der „Navigator“ die Korrektheit des Codes und des Lösungsansatzes überwacht und parallel über Verbesserungen am Design nachdenkt. Der Navigator kann sich zum Beispiel auch **Testfälle überlegen** (wenn das nicht schon vorher gemacht ist!)

## Vorgehen (2)

- ▶ Weil beide Entwickler gleichberechtigt sind, gibt es keine feste Aufgabenteilung. Deshalb wechselt der „Pilot“ z.B. jede Stunde zum „Navigator“ und der „Navigator“ wird zum „Piloten“.
- ▶ Auch in den Paaren werden die Partner gewechselt (Aber dazu später im Jahr.) So wird sichergestellt, dass jedes Teammitglied alle Teile des Projektes kennenlernt.
- ▶ Das funktioniert am besten, wenn die Programmierkenntnisse der Partner\*innen etwa gleich sind!
- ▶ " Das mache ich nicht!" – das ist ja viele zu aufwendig!
- ▶ VORSICHT! ... und über etwas zu reden, dass ich nicht wirklich kenne ist **auch blöd und unakademisch!**

## Ziele des Pair Programming

- **Steigerung der Software Qualität**
- Die Kontrollfunktion der/des zweiten ProgrammiererIn sollen "problematische" (z.B. trickreiche) Lösungen vermeiden.
- Das Pair Programming dient der Verbreitung von Wissen über den Quellcode.
- Durch das regelmäßige Rotieren der Partner wissen beide über den Quellcode bescheid: keine Expertenrollen.
- Bei uns zusätzlich: **schnelleres Erkennen einfacher Fehler** – fachliche Reflektion wird unterstützt.



Bild von: <http://www.it-agile.de/wissen/praktiken/pair-programming/>

## Vorteile des Pair Programming (1)

vergl. Wikipedia Paarprogrammierung

- Insgesamt weniger Fehler im Code: **ca. -15 %**
- Kleinere Programme: **ca. 20% kleiner**
- Höhere Disziplin
- Besserer Code: weniger Sackgassen → verständlicherer Code, höhere Qualität
- Belastbarer Flow: (aus der Psychologie), d.h. sehr hohe Motivation – Unterbrechungen werden auf diese Art besser abgewehrt.
- Freude an der Arbeit:  
90 % der Entwickler sprechen von einer erfreulicheren Arbeit.

## Vorteile des Pair Programming (2)

vergl. Wikipedia Paarprogrammierung

- Geringeres Risiko, weil das gesamte Projektteam Wissen über den Gesamtcode hat → geringeren Projektrisiko hinsichtlich der Mitarbeiterfluktuation und Mitarbeiterabwesenheiten.
- Wissensvermittlung: Jeder hat Wissen, das andere nicht haben. Paarprogrammierung ist ein **Anlass dieses Wissen zu teilen**.
- Teambildung: Die ProgrammiererInnen lernen sich gegenseitig schneller kennen, wodurch die Zusammenarbeit verbessert werden kann.
- Weniger Unterbrechungen: Paare werden seltener unterbrochen als jemand, der alleine arbeitet.

## Nachteile des Pair Programming

vergl. Wikipedia Paarprogrammierung

- Produktivität und Kosten: Geringere Geschwindigkeit bei der Programmierung: Ein Zweierteam schafft ca. 170% im Vergleich zu 200% bei zwei Personen → **die Produktivität pro Person reduziert sich also pro Programmierern um 15 %.**
- Befürworter der Paarprogrammierung behaupten, dass die Produktivität nicht sinke, sondern im Gegenteil sogar deutlich steige. Grund dafür sei, dass die gesteigerte technische und fachliche Qualität genau dort wo die Produktivität erhöhen, wo während der Softwareentwicklung am meisten meiste Zeit verbracht wird: Beim Fehlerfinden und -beheben, sowie beim Lesen von Code.
- Urheberrecht
- Haftung

## Rückblick auf Teil 1 der Elementaren Datentypen

- Gab es Schwierigkeiten mit den Übungsblättern?
- Themen vom letzten Montag
- Numerische Datentypen: **Integer**
- Boolesche Datentyp: **Bool**
- Datentyp `NoneType`

## Unser heutiges Lernziele

- *Ziel ist es, den Datentyp `String` als Programmierer\*in **beherrschen zu lernen** und ihre Eigenarten zu kennen.*
- *Strings als erster Sequenzdateintyp: Der Umgang mit der*
- **Indexierung und Slicing bei Sequenztypen**  
*Wissen und können ;-)*
- **Weitere Escape-Sequenzen** kennenlernen
- **... und allgemein die Nutzung behandeln**

## Übersicht

- Was ist das: Text – Schrift – Zeichen – Alphabet?
- Übliche Zeichensätze: ASCII, ISO/IEC 8859, Unicode
- Erste Programmiererfahrungen mit String: Literale und Operatoren
- Indexierung und Slicing bei Sequenztypen
- String-Funktionen und String-Methoden
- Spezielle Castings hin zu string und String-Kodierung
- Abschluss

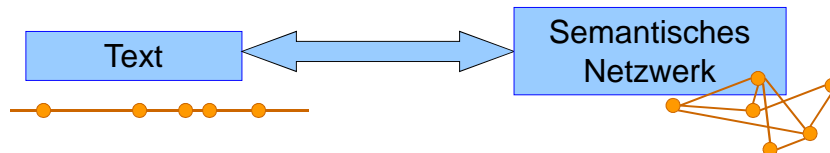
## Text

= geschriebene Sprache (im engeren Sinne)

kommt aus dem Lateinischen. **textus** =

- Gewebe
- Geflecht
- Stoff

## Etwas Hintergrund: Funktionale Verbindung zwischen Gewebe und Text



linear-temporäre Aneinanderreihung von Zeichen und Worten, wird mental  
in eine semantische Netzwerkrepräsentation transformiert

→ Text „verstrickt“ und „verflechtet“ Objekte und Akteure miteinander,

## Die Darstellung des Textes

benötigt eine **Schrift**, deren **Zeichen** wahlweise

- Phoneme ( Laute)
- Silben
- Wörter bzw. Begriffe

kodieren.



## Alphabetschrift

Alphabet (von gr.  $\alpha\lambda\phi\alpha\beta\eta\tau\omicron$  [alfǎwito] - Alpha & Beta)

Beispiele:

- Lateinische Schriften: Den småne skriften
- IPA-Lautschrift: [alfǎwito]
- Kyrillisch: !
- Griechische Schrift:  $\alpha\lambda\phi\alpha\beta\eta\tau\omicron$

## Wortbildschrift, Logogrammschrift

Beispiele:

- Hanzi (chinesisch)
- Hanja (koreanisch)
- Maja-Schrift
- Jurjen
- Tangut



## Sonderformen / Logogramme als Zeichen

- |                  |          |                 |
|------------------|----------|-----------------|
| ► Ziffernschrift | arabisch | 0 1 2 3 .. 9    |
|                  | römisch  | I II III IV ... |

- ▶ **Symbolik der Mathematik oder der Chemie**

$$P_K = \frac{\pi^2 \cdot EI}{S_K^2}$$

- ▶ **Piktogramme**



- ▶ **Emojis &**
- ▶ **Emoticons**

No	Code	Brow.	Chart	Apple	Goog	Twit	Org	IBM	Wind	Sams	GMail	SB	DCM	KDDI	Summary
1	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺				GRINNING FACE
2	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	GRINNING FACE WITH SMILING EYES

## Schriftsysteme der Welt



<http://de.wikipedia.org/wiki/Schrift>

## Begriff: Alphabet in der Informatik

ist in der Informatik weiter gefasst als in der Linguistik:

Unter **Alphabet** versteht man (z.B. nach DIN 44300) eine **total geordnete endliche Menge** (i.d.R. nichtleere endliche Menge) von unterscheidbaren Symbolen (Zeichen).

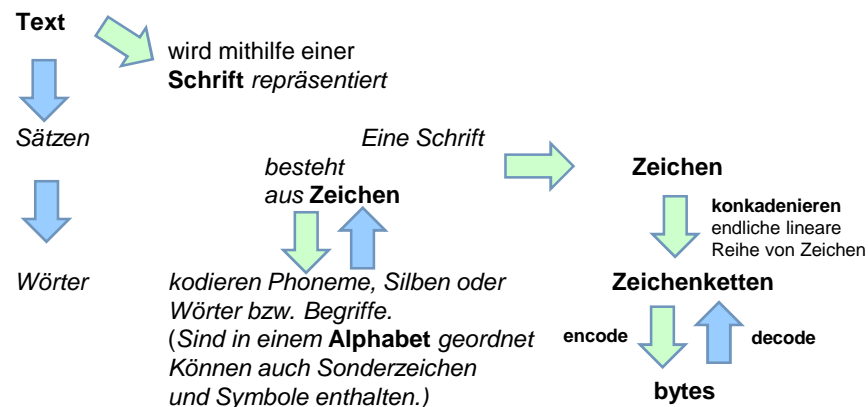
**Häufig symbolisiert durch**  $\Sigma$  (großes Sigma).

## Begriff: Zeichenreihe oder Zeichenkette

**Zeichenreihe (Zeichenkette)** = endliche lineare Reihe von Zeichen eines Alphabets

Übrigens: Auch die Zeichenreihe, die keine Zeichen enthält, ist ein Wort - das **leere Wort**. Es wird mit  $\epsilon$  bezeichnet.

## Zwischen - Zusammenfassung



23

Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing

Prof. Dr. Detlef Krömker

## Begriff: Zeichensatz (character set)

Die **Zuordnung** von alphanumerischen Zeichen (Buchstaben und Ziffern) sowie Sonderzeichen und Symbolen **zu einem Zahlencode**.

= Menge der verfügbaren Zeichen

**Achtung unterschiedliche Definitionen:** „Ein Zeichensatz ist weniger als ein **Zeichencode**, der zusätzlich noch eine definierte Abbildung eines Zeichens auf einen Binärvektor benötigt.“

Beispiele:

- **ASCII (traditionell bis heute sehr häufig verwendet)**
  - **American Standard Code for Information Interchange**
- **IBM EBCDIC (verliert immer mehr an Bedeutung)**

24

Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing

Prof. Dr. Detlef Krömker

## Zwischenruf: Zeichensatz    **Schriftart** (oder Glyph, engl. font, typeface) = **Formatierung**

Bekannte Schriftarten, hier  
**Schriftfamilie:**

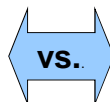
- Arial
- Times New Roman
- Frutiger light
- Courier New
- **Impact**
- Century Gothic
- usw.

- **Schriftart ist** also ein Attribut eines Zeichens
- Hier sind viele Eigenschaften zusammengefasst:
- Schriftfamilie mit den Proportionen und ggf. Serifen sowie dem Schnitt
- Weitere Attribute: Höhe und Laufweite, etc.

## Haupteigenschaften von Schriftfamilien (1)

Proportionale Schriften  
(veränderliche Typenbreite)

(Arial, Times, Frutiger, ..)



Nichtproportionale Schriften  
(fixe Typenbreite)

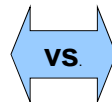
(Courier, Lucida Console, ...)

Fürs Programmieren!

## Haupteigenschaften von Schriftfamilien (2)

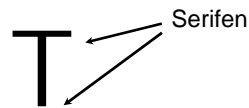
Grotesken-Schriften  
(ohne Serifen)

(Arial, Frutiger, Lucida  
Console, ...)



Antiqua-Schriften  
(mit Serifen)

(Times, Courier, ..)



## Weitere Eigenschaften von Schrift (3) Schriftschnitt

Arial normal (regular)

Arial kursiv (italic)

Arial fett (bold)

Arial fettkursiv (bold italic)

Arial extrafett (black, heavy)

Arial mager (light)

Arial eng (condensed, narrow)

Arial breit (extended)

OUTLINE

SCHATTIERT

### Variationsmöglichkeiten des Schriftbildes einer Schriftfamilie.

Der Begriff „**Schriftschnitt**“ stammt aus der Zeit, als das Schriftbild aller Buchstaben manuell von einem Stempelschneider in Stahl **geschnitten** werden musste. Durch Ausguss entstanden Lettern aus Blei um so die Schriften zu vervielfältigen.

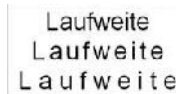
#### 4. Höhe

- ▶ Versalhöhe, die Höhe eines Großbuchstabens (Versal): A, B, C, D, E, ... Z
- ▶ Mittellänge (x-Höhe), die Höhe der meisten Kleinbuchstaben: a, c, e, m, n, o, r, s, u, v, w, x, z
- ▶ Oberlänge, sie ragt etwas über die Mittellänge hinaus: b, d, h, k, l
- ▶ Unterlänge, sie ragt unter die Grundlinie hinaus: g, j, p, q, y
- ▶ Ausnahmen: J, Q, f, i, t.



#### 5. Spationierung Laufweite und Kerning

- ▶ Die **Laufweite** bezeichnet den Abstand **zwischen** den Zeichen einer Schrift.



- ▶ **Kerning** ist das Unterschneiden

ohne

VAL  
VAL

mit

#### Internationale Zeichensätze

<b>ASCII</b> American Standard Code for Information Interchange	Einer der ältesten Computer- Zeichensätze – <b>7 Bit Code</b>	1963	Sehr weit verbreitet, (Programmiersprachen, Internet- Adressen, etc.)
<b>ISO/IEC 8859</b>	15 verschiedenen Kodierungen zur Abdeckung europäischer Sprachen sowie Arabisch, Hebräisch, Thailändisch und Türkisch <b>8 Bit Code</b>	1986	u.a. in Linux und MS Windows verwendet.
<b>Unicode ISO/IEC</b> 10646 – 1991	Internationaler Standard – (7) 8, 16 oder 32 Bit	1991	<b>Ist heute die dominierende Repräsentation</b>

## ASCII-Anekdote



31

Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing

Prof. Dr. Detlef Krömker

## ASCII-Anekdote



32

Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing

Prof. Dr. Detlef Krömker

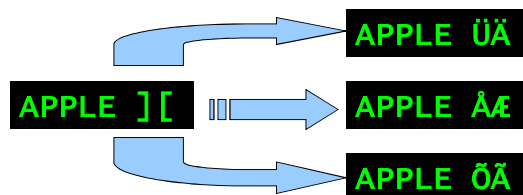


## ASCII genauer unter die Lupe genommen

Problem: ASCII enthält keine diakritischen Zeichen!

⇒ 1972 Einführung von ISO 646-IRV

⇒ Internationale Kompatibilitätsprobleme



## ISO IEC 8859 (fast nur noch historisch bedeutsam!) 8 Bit Code (inbes. Latin-1 und Latin-9 mit Umlauten)

-1	Latin-1	Westeuropäisch
-2	Latin-2	Osteuropäisch
-3	Latin-3	Südeuropäisch
-4	Latin-4	Baltisch
-5		Kyrillisch
-6		Arabisch
-7		Griechisch
-8		Hebräisch
-9	Latin-5	Türkisch
-10	Latin-6	Nordisch
-11		Thai
-13	Latin-7	Baltisch
-14	Latin-8	Keltisch
-15	Latin-9	Westeuropäisch
-16	Latin-10	Südosteuropäisch

Veränderung an 8 Positionen  
(u.a. €-Zeichen)

## Unicode (1)



- **Internationale Norm (privates Konsortium und ISO/IEC), aktuell ist Version 8.0**
  - Identisch mit Universal Character Set (UCS nach ISO 10646)
  - Ziel: Langfristig soll für jedes **sinntragende „Schriftzeichen“** oder **„Textelement“** (Symbole, Emoticons, ...) aller bekannten Schriftkulturen und Zeichensysteme ein digitaler Code festgelegt werden.
  - Es werden nur „abstrakte Zeichen“ (engl.: *characters*) kodiert, nicht dagegen die grafische Darstellung (Glyphen)
  - Unicode wird ständig um weitere Zeichen ergänzt (gepflegt und gewartet).

## Unicode (2)



- 1. Schritt:** Jedem Unicode-Zeichen wird **eindeutig** ein sogenannter **Codepunkt** (engl. code point) zugeordnet.
- der Coderaum des Unicode-Zeichensatzes umfasst dezimal 0 – 1.114.111 (hexadezimal: 0 – 10FFFF).
  - Dies sind 17 sogenannten „Ebenen“ à 65536 codepoints
  - Von diesen gut 1 Mio **code points** sind aktuell nur etwas mehr als 100.000 genutzt.
  - Benannt werden diese Codepunkte üblicherweise durch ihren Hexcode mit dem Prefix

Siehe: [unicode.org/charts/](https://unicode.org/charts/) oder [unicode-table.com](https://unicode-table.com)

**Python 3.5 benutzt Unicode 8.0.0**

## Unicode (3)

**2. Schritt:** Mit Hilfe eines **Unicode Transformation Formats (UTF)** werden Unicode-Zeichen auf Folgen von Bytes abgebildet.

(Es gibt sehr viele Encodings.) Üblich sind:

- ▶ **UTF-8** kodiert Zeichen mit variabler Byte-Anzahl (**1-4 Bytes**). Die Codepoints 0 bis 127, die dem ASCII-Zeichensatz entsprechen, werden in einem Byte kodiert.
- ▶ **UTF-16** ist das älteste Kodierungsverfahren, bei dem ein oder zwei 16-Bit-Einheiten (2 oder 4 Bytes) verwendet werden. ACHTUNG: Bytereihenfolge beachten: UTF-32BE und UTF-32LE
- ▶ **UTF-32** kodiert ein Zeichen immer in genau 32 Bit. Auch hier Byte-Reihenfolge beachten.

## Übersicht

- ▶ Was ist das: Text – Schrift – Zeichen – Alphabet?
- ▶ Übliche Zeichensätze: ASCII, ISO/IEC 8859, Unicode
- ▶ **Erste Programmiererfahrungen mit String: Literale und Operatoren**
- ▶ Indexierung und Slicing bei Sequenztypen
- ▶ String-Funktionen und String-Methoden
- ▶ Spezielle Castings hin zu string und String-Kodierung
- ▶ Abschluss

## Zeichenketten in Python

~~Python implementierte in der Version 2.X zwei verschiedene Basis-Typen für Zeichenketten (*strings*):~~

- ~~• **strings** (8-Bit, ein Oktett, ein Byte)~~
- ~~• **Unicode-Strings** (variable Codelänge von 8-32 Bits)~~

Ab Version 3.0 gibt es nur noch

**strings** (UTF-8 kodiert variable Codelänge von 8-32 Bits, intern 32 Bit)

Python kennt nicht den Datentyp eines **Zeichens (ein character)** : Ein einzelnes Zeichen, z.B. "a" wird als Einelementiger String repräsentiert

**Python-Programme** werden im Default in UTF-8 kodiert!  
(Erlaubt sind aber nur sehr wenige Zeichen außer in Strings.)

## String-Literale in Python

- ▶ Zeichenketten werden in (obenstehenden) "Anführungszeichen" geschrieben, gleichgültig ob einfache oder doppelte ".
- ▶ Hier erlebt wohl jede ProgrammiererIn einmal Überraschungen, weil Typographen, Schriftsetzer, Linguisten, sonstige Sprachkundige, ... , kurz "echte Besserwisser" hier ein „Chaos“ angerichtet haben.
- ▶ Der Python Interpreter akzeptiert nur

Gedrucktes Zeichen	Unicode Name	Code point
'	Apostrophe	0x0027 = 39
"	Quotation Mark	0x0022 = 34

Die Python Syntax  
Sagt schlicht  
single quote (') or  
double quote (") --  
„Anführungszeichen“

## Welches Chaos genau?



aus:  
de.wikipedia.org/wiki/Apostroph:

Arial Calibri Tahoma Times New Roman Linux Libertine

Typografisch korrekter (grün) und gerader (rot) **Apostroph** sowie Minutenzeichen (*Prime*, blau) zwischen Buchstaben l, i mit Akut-Akzen.

Es gibt noch diverse weitere Varianten, siehe z.B. in der Wikipedia: [de.wikipedia.org/wiki/Apostroph](http://de.wikipedia.org/wiki/Apostroph) oder [de.wikipedia.org/wiki/Anführungszeichen](http://de.wikipedia.org/wiki/Anführungszeichen)

Die Programmiersprachen-Editoren (z.B. Idle) machen das natürlich immer richtig, aber wenn man Code aus Word, PowerPoint, o.ä. Programme übernimmt, dann kommt es i.d.R. zu Problemen.

## String-Literale

1. '...' oder Standard
2. "..." oder nimmt man, wenn 'im String vorkommt
3. '''...''' oder der String darf mehrzeilig sein
4. """..."""

- ▶ Immer nur als Paar nutzen! - '...' ist unzulässig ... erlaubt ist aber: "hello" ' world' , das ist äquivalent zu "hello world"
- ▶ Zwischen den „Anführungszeichen“ können **alle** Unicode-Zeichen stehen
  - ▶ Eingabe durch die Tastatur, auch ß, ö, ... oder
  - ▶ durch eine sogenannte **ESCAPE-Sequenz**, das ist
    - \????... Backslash ist das Escape-Symbol (Fluchtsymbol oder Markierungszeichen), es folgen 1–n Zeichen

## Unicode Escape- (=Flucht) Sequenzen

Escape Sequence	Meaning	Notes
<code>\N{name}</code>	Character <i>name</i> in the Unicode database	1)
<code>\uxxxx</code>	Character with 16-bit hex value <i>xxxx</i>	exakt 4 Hex
<code>\Uxxxxxxxx</code>	Character with 32-bit hex value <i>xxxxxxxx</i>	exakt 8 Hex


1) Siehe z.B. <http://unicode.org/Public/UNIDATA/NamesList.txt>

**Achtung:** Es kann sein, dass Sie ein gültiges Unicode Zeichen eingegeben haben, Sie es aber trotzdem nicht drucken können, weil es dieses Zeichen in den ausgewählten Schriften (Fonts) nicht gibt.


Es gibt noch weitere Escape Sequenzen, **siehe Vertiefungs-Kurs**.

## Weitere zulässige Escape (Flucht) -Sequenzen in Python

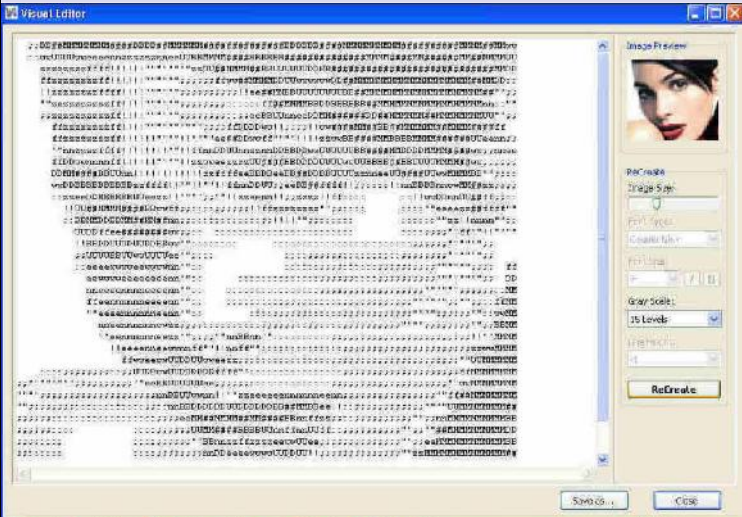
Escape Sequence	Meaning	Escape Sequence	Meaning
<code>\newline</code>	Backslash and newline ignored	<code>\n</code>	<b>ASCII Linefeed (LF)</b>
<code>\\</code>	Backslash (\)	<code>\r</code>	ASCII Carriage Return (CR)
<code>\'</code>	<b>Single quote (')</b>	<code>\t</code>	ASCII Horizontal Tab (TAB)
<code>\"</code>	<b>Double quote (")</b>	<code>\v</code>	ASCII Vertical Tab (VT)
<code>\a</code>	ASCII Bell (BEL)	<code>\ooo</code>	Character with octal value <i>ooo</i>
<code>\b</code>	ASCII Backspace (BS)	<code>\xhh</code>	Character with hex value <i>hh</i>
<code>\f</code>	ASCII Formfeed (FF)		



studiumdigitale  
megadigitale



GOETHE  
UNIVERSITÄT  
FRANKFURT AM MAIN



Damit wird dann auch sogenannte „ASCII-Kunst“ gemacht - von:

<http://www.bestshaware.net/img2/ascii-art-generator3-big.jpg>

45
Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing
Prof. Dr. Detlef Krömer



studiumdigitale  
megadigitale



GOETHE  
UNIVERSITÄT  
FRANKFURT AM MAIN

## Beispiele

```

>>> 'Hallo\nLeute' # a regular string with escape \n
'Hallo\nLeute'
>>> print('Hallo\nLeute') # makes a „line feed“ while printed
Hallo
Leute
>>> r'Hallo\nLeute' # the same string as raw string, notice \\
'Hallo\nLeute'
>>> print(r'Hallo\nLeute') # in a print statement
Hallo\nLeute
>>> print('Hallo\nLeute')
Hallo\nLeute
>>> type(r'Hallo\nLeute') # stored as a regular string
<class 'str'>
>>> r"" # two characters
""
>>> r" # not a valid string literal
SyntaxError: EOL while scanning string literal

```

46
Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing
Prof. Dr. Detlef Krömer

## String Operatoren

Die Operatoren + und \* haben für Strings S, T eine besondere Bedeutung

S + T: ist die Konkatenation (Verkettung, Hintereinanderfügen der Operanden)

n \* S; S \* n: Wiederholung (n ist Integer)

```
>>> a = 'bim-'
>>> b = 'bam, '
>>> a + b
'bim-bam, '
>>> c = a + b
>>> c * 3
'bim-bam, bim-bam, bim-bam, '
>>> 3 * c
'bim-bam, bim-bam, bim-bam, '
```

## Vergleichsoperatoren für Strings (1)

Die Vergleichsoperatoren

x > y,	x >= y,	x == y,	x is y (Objekt-Identität)
x < y,	x <= y,	x != y	x is not y

funktionieren **auch** für strings.

**Achtung:** <, >, <=, >= ordnen die **Zeichen** (einschließlich Ziffern, Leerzeichen, Satz- und Sonderzeichen) **nach dem Zahlenwert der dem Codepoint im Unicode entspricht**, so dass bspw. alle lateinischen Großbuchstaben vor dem kleinen „a“ eingeordnet werden; Ö,ö,Ü,ü,Ä,ä,ß stehen dagegen ganz hinten. Siehe Anhang zum Skript.



## Vergleichsoperatoren für Strings (2)

- Ansonsten wird **lexikographisch** verglichen, d.h. die Strings werden zunächst nach ihren Anfangsbuchstaben verglichen, dann bei gleichen Anfangsbuchstaben nach dem jeweils zweiten Buchstaben, usw.
- Ist ein Wort ganz in einem anderen als Anfangsteil enthalten (wie beispielsweise „kann“ in „kannst“), so ist das kürzere Wort kleiner.

## Übersicht

- Was ist das: Text – Schrift – Zeichen – Alphabet?
- Übliche Zeichensätze: ASCII, ISO/IEC 8859, Unicode
- Erste Programmiererfahrungen mit String: Literale und Operatoren
- **Indexierung und Slicing bei Sequenztypen**
- Escape-Sequenzen
- String-Funktionen und String-Methoden
- Spezielle Castings hin zu string und String-Kodierung
- Abschluss

## Indexierung

Konzeptionelles Modell des Strings: Zeichen stehen in „Zellen“. Diese Zellen sind nummeriert. Jedes Zeichen ist unter seinem eigenen **Index**  $i$  zugreifbar:  $S[i]$  in eckigen Klammern.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	E	P	R		i	s	t		t	o	l	l	
Index	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

Beispiel:

```
>>> S = 'EPR ist toll'
>>> S[0]
'E'
>>> S[-4]
't'
```

Zu negativen Indexwerten wird  $\text{len}(S)$  hinzuaddiert.

```
>>> len('EPR ist toll')
12
```

## Slicing (= Teilbereichbildung)

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	E	P	R		i	s	t		t	o	l	l	
Index	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

- ▶ Extrahiert zusammenhängende (Zellen-)bereiche.
- ▶ **Regel:  $S[i,j]$**   $S[i]$  ist das erste Element im Teilbereich  
 $S[j-1]$  ist das letzte Element im Teilbereich
- ▶  $S[1:5]$  geht von 1 bis 4 (**ausschließlich 5**)
- ▶ Bereichsgrenzen sind mit 0 und der Sequenzlänge  $\text{len}(\text{String})$  vorbelegt
- ▶  $S[1:]$  geht von Index 1 bis zum letzten Element
- ▶  $S[:-1]$  nimmt alles mit Ausnahme des letzten Elements
- ▶  $S[:]$  macht eine (flache) Kopie von  $S$

## Beispiele zum Slicing (= Teilbereichbildung)

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	E	P	R		i	s	t		t	o	l	l	
Index	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

```
>>> S = 'EPR ist toll'
>>> S[1:5]
'PR i'
>>> S[1:]
'PR ist toll'
>>> S[:-3]
'EPR ist t'
>>> S[: ]
'EPR ist toll'
```

## Indizes und Slicing-Indizes außerhalb der „Feldgrenzen“

Index			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Wert			E	P	R		i	s	t		t	o	l	l			
Index	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1			

**Indexwerte** außerhalb des Wertebereichs [0,11] oder [-12,-1] führen zu einem **"IndexError"**.

```
>>> S[-12]
'E'
>>> S[-13]
Traceback (most recent call last):
  File "<pyshell#204>", line 1, in ...
IndexError: string index out of range
```

Anders **beim Slicing**, dort werden die mit roten Pfeilen markierten Indexwerte auf  $\pm \text{len}(S)$  „gecastet“

```
>>> S[9:14]
'oll'
>>> S[-9:-12]
''
>>> S[-14:3]
'EPR'
```

## Slicing-Indizes mit strides Erweiterte Teilbereichsbildung $S[i:j:k]$

Index			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Wert			E	P	R		i	s	t		t	o	l	l			
Index			-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1			

- Das dritte Element k in  $S[i,j,k]$  gibt den **<stride>** („Schritt“) an:  
(Erhöhung / Erniedrigung des Index, bisher war das +1)
- Als erstes werden die zu großen j oder zu kleinen i auf die zulässigen Werte gecastet, dann in positive Werte gemäß der Grafik gewandelt, ergibt m 0, n len(S)
- wenn **<stride> > 0**: Ergebnis: S [m] bis S [n] mit <stride> Schritten;  
S [n] gehört nicht zum Slice; wenn m n ist das Ergebnis der leere String;
- wenn **<stride> < 0**: Ergebnis S [n] bis S [m]; S [m] gehört nicht zum Slice;  
wenn m n ist das Ergebnis der leere String.

## Beispiele zur erweiterte Teilbereichsbildung $S[i:j:k]$

Index			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Wert			E	P	R		i	s	t		t	o	l	l			
Index			-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1			

wenn **<stride> = 0**: ,ValueError: slice step cannot be zero'

```
>>> S[::2]
'ERittl'
>>> S[6:0:-1]
'tsi RP'
>>> S[6::-1]
'tsi RPE'
>>> S[0:6:-1]
''
```

```
>>> S[6:6:-1]
''
>>> S[-6:-12:-1]
'tsi RP'
```

## Beispiele: Erweiterte Teilbereichsbildung S[i:j:k]

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Wert	E	P	R		i	s	t		t	o	l	l	
Index	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

- S[::2] von Anfang bis Ende jedes zweite Element
- S[::-1] ergibt die Umkehrung der Sequenz S
- S[0:12:-1] ergibt den leeren String
- S[15:0:-1] ergibt die Umkehrung der Sequenz, bis auf S[0] gehört nicht dazu
- S[4::-2] holt jedes zweite Element von rechts nach links ab Position 4 bis 0
- S[-4:-12:-2] = 'ttiR'

## Übersicht

- Was ist das: Text – Schrift – Zeichen – Alphabet?
- Übliche Zeichensätze: ASCII, ISO/IEC 8859, Unicode
- Erste Programmiererfahrungen mit String: Literale und Operatoren
- Indexierung und Slicing bei Sequenztypen
- **String-Funktionen und String-Methoden**
- Spezielle Castings hin zu string und String-Kodierung
- Abschluss

## Funktionen

- Wir kennen bisher wenige Funktionen
  - sie liefern entweder einen Wert und können damit an jeder Stelle stehen, wo eine Variable stehen kann, z.B. `len(s)` oder
  - sie lösen ein bestimmtes Verhalten (eine Operation oder Funktion) aus, z.B. `print(s)`
- Schreibweise: `funktionsname(x,y,...)`, z.B. `len(s)`



Parameter

- Hier ist (leider) ein kleiner Vorgriff auf Funktionen und die Objektorientierung nötig.

59

Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing

Prof. Dr. Detlef Krömer

## Attribut-(Punkt-)schreibweise für Methoden

- Python ist im Kern eine objektorientierte Programmiersprache. Hier nennt man Funktionen Methoden (genauer **Funktionen implementieren Methoden**).
- Methoden werden mit der Attribut-(Punkt-)schreibweise aktiviert:  
`s.Methodenname()`
- Für jeden Operator und für jede built-in Funktion gibt es in Python die zugehörige Methode, die diese Operation implementiert:
- Operator `x + y` → Methode `__add__` → Aufruf `x.__add__(y)`
- Funktion `len(x)` → Methode `__len__` → Aufruf `x.__len__()`
- Dies sind (zunächst) einmal unterschiedliche Schreibweisen desselben.

60

Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing

Prof. Dr. Detlef Krömer

## Beispiele

```
>>> s = 'Detlef '
>>> t = 'Krömker'
>>> len(s)
7
>>> s+t
'Detlef Krömker'
>>> s.__len__()
7
>>> s.__add__(t)
'Detlef Krömker'
```

## Weitere String (und Byte) Operationen

### Operatoren

- x in S      Test auf Enthalten sein
- x not in S

### Funktionen

- len(S)      Länge des Strings
- min(S)      minimales Element
- max(S)      maximales Element

Es gibt noch zwei weitere Operationen, die wir im Zusammenhang mit der Iteration behandeln werden.

## Beispiele für Operationen

```
>>> S = 'Halligalli'
>>> 'll' in S
True
>>> 'lla' not in S
True
>>> len(S)
10
>>> max(S)
'l'
>>> min(S)
'H'
```

## Weitere String (und Byte) Methoden

- ▶ String-Methoden bieten weitergehende Möglichkeiten zur Textbearbeitung.
- ▶ Insgesamt weit **über 50 Methoden** u.a. zum Suchen, zum Aufteilen und Zusammenfügen, zum Formatieren.
- ▶ Einige der Funktionen heißen merkwürdig, z.B. `__add__(self, value, /)` Was heißt das: Dies sind Python-interna, die der Objektorientierung zuzurechnen sind ... wir benutzen diese Funktionen nicht in dieser Form.
- ▶ `help(str)`  
liefert die benötigten Infos.

**ABER: Textvergleiche sind allgemein schwer und manchmal tricky.**



## Beispiel für help ()

```
>>> help (str)
Help on class str in module builtins:

class str(object)
|   str(object='') -> str
|   str(bytes_or_buffer[, encoding[, errors]]) -> str
|
|   Create a new string object from the given object. If encoding or
|   errors is specified, then the object must expose a data buffer
|   that will be decoded using the given encoding and error handler.
|   Otherwise, returns the result of object.__str__() (if defined)
|   or repr(object).
|   encoding defaults to sys.getdefaultencoding().
|   errors defaults to 'strict'.
|
|   Methods defined here:
```

```
...   __add__(self, value, /)
|       Return self+value.
...
|   capitalize(...)
|       S.capitalize() -> str
|
|       Return a capitalized version of S, i.e. make the first character
|       have upper case and the rest lower case.
...
|   casefold(...)
|       S.casefold() -> str
|
|       Return a version of S suitable for caseless comparisons.
...
|   zfill(...)
|       S.zfill(width) -> str
|
|       Pad a numeric string S with zeros on the left, to fill a field
|       of the specified width. The string S is never truncated.
```

## Beispiel - Ziel: Unterschiedliche flektierte Wortformen sollen als gleiches Wort (Lexem) erkannt werden:

- **Lemmatisierung** (Grundform)
- **Stemming** (*Stammform-, Normalformenreduktion*)  
u.a. „Plätten“ von Umlauten → idealerweise Wortstamm

### Auch dazu gehören

- Abgleich von Schreibvarianten
- Groß- und Kleinschreibung
- Schreibfehler
- Entfernen von Stoppwörtern
- ggf. Auflösen von Abkürzungen
- ggf. Entfernen von Satzzeichen

Die Herausforderungen sind viel, viel größer. Im Ifl gibt es dazu

- **Prof. Dr. Alexander Mehler** Texttechnologie
- **JProf. Dr. Christian Chiarcos** Angewandte Computerlinguistik

## Zusammenfassung und Ausblick

- Tatsächlich waren das Details.
- Nicht immer geht es ums Rechnen im engeren Sinn.
- Machen Sie bitte das Quiz.

## Übersicht

- Was ist das: Text – Schrift – Zeichen – Alphabet?
- Übliche Zeichensätze: ASCII, ISO/IEC 8859, Unicode
- Erste Programmiererfahrungen mit String: Literale und Operatoren
- Indexierung und Slicing bei Sequenztypen
- String-Funktionen und String-Methoden
- **Spezielle Castings hin zu string und String-Kodierung**
- Abschluss

## Der Unterschied zwischen str(), repr() und ascii()

- Alle drei Funktionen erzeugen Werte vom **Typ String**.
- Alle drei Funktionen sind zumindest bei allen Basistypen anwendbar.
- **Was ist der Unterschied?**
- **str()** erzeugt einen String mit dem Ziel: **möglichst gute Lesbarkeit** für Menschen: Diese Funktion wird zum Beispiel bei print () ausgeführt und liefert damit „gut“ lesbare Ausgaben.
- **repr()** erzeugt einen String mit dem **Ziel auf Eindeutigkeit und Unmissverständlichkeit**. Es kann für ProgrammiererInnen bei der Fehlersuche hilfreich sein.
- **ascii()** erzeugt einen String **repr()** jedoch mit Unicode-escapes, wo nötig.

## Beispiel: str(), repr() und ascii()

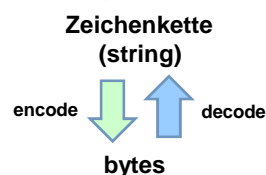
```
>>> import datetime
>>> now = datetime.datetime.now()
>>> str(now)
'2015-08-30 16:23:01.111462'
>>> repr(now)
'datetime.datetime(2015, 8, 30, 16, 23, 1, 111462)'
>>> ascii(now)
'datetime.datetime(2015, 8, 30, 16, 23, 1, 111462)'
>>> print(now)
2015-08-30 16:23:01.111462
>>> repr('Krömkker')
'"Krömkker"'
>>> len('Krömkker')
7
>>> len(repr('Krömkker'))
9
>>> ascii('Krömkker')
'"Kr\x66mker"'
```

- Ooooh, doch erhebliche Unterschiede.
- Für Programmieranfänger (einfache Datentypen) reicht immer das str().
- Kompliziertere Datentypen offenbaren ihre Geheimnisse für Programmierer mit repr oder ascii offenkundiger.

## Bytes

- “Humans use text. Computers speak bytes.”  
Slide 12 of PyCon 2014 talk “Character Encoding and Unicode in Python” ([slides](#), [video](#)).
- “Bytes are bytes; characters are an abstraction.” Mark Pilgrim, 2011  
[diveintopython3.net/strings.html](http://diveintopython3.net/strings.html)
- “Bytes is a built-in type for manipulating binary data.”  
<https://docs.python.org/3/library/stdtypes.html>
- Bytes sind **kein numerischer** Datentyp

Strings haben die Funktion `encode()`  
Bytes haben die Funktion `decode()`



## Beispiele

```
>>> s = 'alpha'
>>> b = b'beta'
>>> print(s,b) #s is a string, b a byte object
alpha b'beta'
>>> t = s.encode('utf-8') # the encoding method
>>> print(t)
b'alpha'
>>> t = 'Krömkker, # handling of non-ASCII character
>>> t.encode('utf-8')
b'Kr\xc3\xb6mkker'
>>> u = b.decode('utf-8') # the decode method
>>> print(u)
beta
>>> s + b # never mix up string and byte types
Traceback (most recent call last):
  File "<pyshell#104>", line 1, in <module>
    s + b
TypeError: Can't convert 'bytes' object to str implicitly
```

73

Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing

Prof. Dr. Detlef Krömkker

**Twelve characters, their code points, and their byte representation (in hex) in seven different encodings - (asterisks indicate that the character cannot be represented in that encoding)**

char.	code point	ascii	latin1	cp1252	cp437	gb2312	utf-8	utf-16le
A	U+0041	41	41	41	41	41	41	41 00
¿	U+00BF	*	BF	BF	A8	*	C2 BF	BF 00
Ã	U+00C3	*	C3	C3	*	*	C3 83	C3 00
á	U+00E1	*	E1	E1	A0	A8 A2	C3 A1	E1 00
Ω	U+03A9	*	*	*	EA	A6 B8	CE A9	A9 03
€	U+06BF	*	*	*	*	*	DA BF	BF 06
"	U+201C	*	*	93	*	A1 B0	E2 80 9C	1C 20
€	U+20AC	*	*	80	*	*	E2 82 AC	AC 20
ƒ	U+250C	*	*	*	DA	A9 B0	E2 94 8C	0C 25
气	U+6C14	*	*	*	*	C6 F8	E6 B0 94	14 6C
氣	U+6C23	*	*	*	*	*	E6 B0 A3	23 6C
Ⓢ	U+1D11E	*	*	*	*	*	F0 9D 84 9E	34 DB 1E DD

Aus: Luciano Ramalho: Fluent Python Published by O'Reilly Media, Inc., 2015

74

Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing

Prof. Dr. Detlef Krömkker

## Mit verschiedenen Encodings müssen wir leben:

Encoding defaults  
(von meinem Rechner)

```
>>> from sys import *
>>> import locale
>>> stdout.encoding
'cp1252'
>>> stdin.encoding
'cp1252'
>>> stderr.encoding
'cp1252'
>>> getdefaultencoding()
'utf-8'
>>> getfilesystemencoding()
'mbcs'
>>> locale.getpreferredencoding()
'cp1252'
```

- Wir können es kaum vermeiden:
- In einem realen System haben wir es immer mit verschiedenen Encodings zu tun.
- Cp1252:** Windows Codepage 1252, 8 Bit
- mbcs:** multi-byte character set  
In diesem Fall: In speziell für Windows: code pages 932 (Shift\_JIS), 936 (GBK), 949 (KS\_C\_5601-1987), and 950 (Big5), but **NOT** UTF-8.

75

Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing

Prof. Dr. Detlef Krömer

## Zusammenfassung

Best Practice

**The Unicode sandwich**



bytes → str Decode bytes on input,  
100% str process text only,  
str → bytes encode text on output.

76

Vorlesung PRG 1 – V4  
Elementare Datentypen – Float, String, Typing

Prof. Dr. Detlef Krömer

## Ausblick ... Was steht jetzt an?


















**Wie immer: Sie müssen üben, d.h. insbesondere die EPR-Übung machen!**

Auch zu dieser *Vorlesung* gibt es ein **Quiz**. Das sollten Sie unbedingt (sofort) absolvieren.

Erreichen Sie nicht 80% der Punkte, so sollten Sie gezielt die jeweiligen Kapitel nachlesen oder dieses Video noch einmal hochkonzentriert anschauen.

Und nicht vergessen: Das Quiz dann nach 2-3 Tagen noch einmal wiederholen.

ü , ð ñ ü ß ã ã @ µ ¶ !

	U+2603 SNOWMAN		U+2620 SKULL AND CROSSED BONES		U+1F30E EARTH GLOBE AMERICAS		U+1F30F EARTH GLOBE AFRICA
	U+1F40D SNAKE		U+1F41D HONEYBEE		U+1F426 BIRD		U+1F44C OK HAND SIGN
	U+1F44D THUMBS UP SIGN		U+1F47D EXTRATERRESTRIAL ALIEN		U+1F4A5 COLLISION SYMBOL		U+1F4A9 PILE OF POO
	U+1F601 GRINNING FACE WITH SMILING EYES		U+1F60E SMILING FACE WITH SUNGLASSES		U+1F61E DISAPPOINTED FACE		U+1F62D ANGRY FACE
	U+1F648 SEE NO EVIL MONKEY		U+1F649 HEAR NO EVIL MONKEY		U+1F64A SPEAK NO EVIL MONKEY		

Von: Ned Batchelder's: ["Pragmatic Unicode" talk](#) at US PyCon 2012.

## **Ausblick ... Nächsten Montag**

### **Programmflußkontrolle (2): Funktionen und Prozeduren**

**Am Freitag:**

**Datentyp Float**

**... und, danke für Ihre Aufmerksamkeit!**