Funktionale Sprachen: LISP

- Input/Output:
  1. Standardstreams:

terminal-io	Benutzerkonsole
standard-input	Ein- (Tastatur) und
standard-output	Ausgabe im Toplevel-Loop

     weitere für Fehlerausgabe, Debugging
  2. Grundlegende Streamfunktionen:

     (streamp *stream*)

     (open-stream-p *stream*)

     (input-stream-p *stream*), (output-stream-p *stream*)

     (close *stream*)

     (open *dateiname parameter), (*with-open-file *…)*

     (make-string-input-stream *string),*

     (make-string-output-stream *string)*

Funktionale Sprachen: LISP

- Input/Output:
  3. Dateizugriff: nur über Verbindung der Datei mit Input- oder Output-Stream

     opt. Parameter steuert Richtung (:direction mit Werten :input, :output, :io)

     opt. Parameter steuert Fehlerbehandlung (:if-exists, :if-does-not-exist)

     if-exists

     :error, :overwrite, :append, :supersede, nil

     if-does-not-exist

     :error, :create

     nil kein Stream wird erzeugt, Streamvariable wird nil

Funktionale Sprachen: LISP

- Beispiel:

```
(with-open-file  (ifile "/home/usr/bsp.txt"
                    :direction :input              ;Stream zum Lesen
                    :if-does-not-exist NIL)        ;Fehlerbehandlung
          (when  (streamp  ifile)                  ;prüft, ob ifile Stream
                    (print  (read ifile))))        ;lies und drucke Inhalt
```

```
(with-open-file  (mystream   "hallo.txt"
                    :direction  :output
                    :if-does-not-exist  :create)
          (when  (streamp  mystream)
                    (print "Hallo, Leute!"   mystream)))
```

Funktionale Sprachen: LISP

- Wort/Zeile/Zeichen ein- und auslesen:
  - (prin1 ausdruck [stream]) ;wandelt Lisp-Objekt in darstellbare Form um, gibt sie an standard-output oder stream
  - (print ausdruck [stream]) ;wie prin1, nur mit Zeilenwechsel vor und Leerzeichen nach der Ausgabe
  - (read [stream]) ;liest von standard-input bzw. stream, gibt gelesenes als Lisp-Objekt zurück
  - (read-line [stream]) ;liest Zeile ein
  - (read-char [stream]) ;liest Zeichen ein
  - (read-from-string [string]) ;wie read, liest nur aus String

Funktionale Sprachen: LISP

- Formatierte Ausgabe:
  - (format stream control-string [argumente])
  - Ist stream NIL: Ausgabe in String, ist stream T:Ausgabe nach standard-output, bei Angabe streamname: Ausgabe in Output-Stream
  - control-string: Ausgabestring mit Steuerungsbefehlen zur Formatierung

  Formatierungsbefehle:
  - ~% Zeilenwechsel
  - ~& Zeilenwechsel, falls nicht bereits neue Zeile
  - ~A, ~S Stellvertreter für Variable, müssen als Argumente angegeben werden
  - ~D, ~nD Dezimalausgabe, n Mindest-Stellenanzahl
  - ~F Fließkommazahl
  - ~w,d F Fließkommazahl mit w Ausgabezeichen und d Nachkommastellen
  - ~~ stellt ~ selbst dar

Funktionale Sprachen: LISP

- Beispiele

(format T “ ~% Pi mit 2 Nachkommastellen = ~,2F“ 3.1416)
→ Pi mit 2 Nachkommastellen = 3.14

(setf nachname “Bond“ vorname “James“)
(format T “Mein Name ist ~A ~A“ vorname nachname)
→ Mein Name ist James Bond

Funktionale Sprachen: LISP

Formatierte Ausgabe in eine Datei:

```
(defun ausgabe (datei text)
    (let  ((stream  (open datei :direction :output)))
          (format stream “Das ist die heutige Ausgabe: ~%~%~S~%“
                                 text)
          (close stream)
    )
)
```

```
Aufruf>  (ausgabe “testdatei.txt“  ´(Dies ist eine Liste))
→ ??
Aufruf>  (ausgabe “testdatei.txt“ (length (list ´a ´b ´c ´d)))
→ ??
```

Funktionale Sprachen: LISP

unformatierte Ausgabe in eine Datei:

```
(setf stream (open “testdat.txt“ :direction :output))
(prin1 (+ 1 2 3 4) stream)
(close stream)
```

Lesen aus einer Datei:

```
(setf stream (open “testdat.txt“ :direction :input))
(setf a  (read stream))                    ;liest ein Wort
(print a)
(close stream)
```