Mini Lecture 1

Postscript

07. November, 2022

Einführung

Was machen wir heute?

- Grundbegriffe

Was machen wir heute?

- Grundbegriffe
- Einige Basiselemente

Was machen wir heute?

- Grundbegriffe
- Einige Basiselemente
- Erste Interaktion mit einem PS-Interpreter

Was macht ihr heute?

- PS-Interpreter zum laufen bringen

Was macht ihr heute?

- PS-Interpreter zum laufen bringen
- Einfache Übungen im Interpreter

Notation

- Postscript arbeitet Stackbasiert

Notation

- Postscript arbeitet Stackbasiert
- Befehle daher in UPN angeben

Notation

- Postscript arbeitet Stackbasiert
- Befehle daher in UPN angeben
- Beispiel: `3 5 mul 27 add`

Notation

- Postscript arbeitet Stackbasiert
- Befehle daher in UPN angeben
- Beispiel: `3 5 mul 27 add`
- Ergebnisse werden auf den *operand stack* gelegt

Sprachkonzepte

- Erste Zeile einer PS-Datei: %!

Sprachkonzepte

- Erste Zeile einer PS-Datei: %!
- Comment: %

Sprachkonzepte

- Dictionary: Nestbar

Sprachkonzepte

- Dictionary: Nestbar
- Name: Folge von Zeichen, die nicht als Zahl interpretierbar sind

Sprachkonzepte

- Dictionary: Nestbar
- Name: Folge von Zeichen, die nicht als Zahl interpretierbar sind
- Number: Integers, Reals

Sprachkonzepte

- Dictionary: Nestbar
- Name: Folge von Zeichen, die nicht als Zahl interpretierbar sind
- Number: Integers, Reals
- String: `(This is a string)`

Sprachkonzepte

- Dictionary: Nestbar
- Name: Folge von Zeichen, die nicht als Zahl interpretierbar sind
- Number: Integers, Reals
- String: `(This is a string)`
- Array: `[3 5 mul]` $==$ `[15]`

Sprachkonzepte

- Procedure: Executable Array

Sprachkonzepte

- Procedure: Executable Array
- `{2 dup mul}` $\neq$ `{4}`

Sprachkonzepte

- Procedure: Executable Array
- `{2 dup mul}` $\neq$ `{4}`
- In der Regel genutzt um neue Operatoren zu definieren

Stacks

- **Stacks**: Operand Stack, Dictionary Stack (und weitere)

Stacks

- **Stacks**: Operand Stack, Dictionary Stack (und weitere)
- Operand Stack: Push operands, invoke operator

Stacks

- **Stacks**: Operand Stack, Dictionary Stack (und weitere)
- Operand Stack: Push operands, invoke operator
- `3 5 mul 27 add`

Stacks

- **Stacks**: Operand Stack, Dictionary Stack (und weitere)
- Operand Stack: Push operands, invoke operator
- `3 5 mul 27 add`
- Dictionary Stack: Hält eure Variablen (dictionaries)

Stacks

- **Stacks**: Operand Stack, Dictionary Stack (und weitere)
- Operand Stack: Push operands, invoke operator
- `3 5 mul 27 add`
- Dictionary Stack: Hält eure Variablen (dictionaries)
- `/sq {dup mul} def`

Operatoren

- `https://personal.math.ubc.ca/~cass/courses/ps.html`

Operatoren

- `https://personal.math.ubc.ca/~cass/courses/ps.html`
- PS matches operators greedily!

Operatoren

- `https://personal.math.ubc.ca/~cass/courses/ps.html`
- PS matches operators greedily!
- `/add {mul} def` überlädt `add`

Operatoren

- `https://personal.math.ubc.ca/~cass/courses/ps.html`
- PS matches operators greedily!
- `/add {mul} def` überlädt `add`
- Operatoren werden *zur Ausführung* durch Definition substituiert

Operatoren

- `https://personal.math.ubc.ca/~cass/courses/ps.html`
- PS matches operators greedily!
- `/add {mul} def` überlädt `add`
- Operatoren werden *zur Ausführung* durch Definition substituiert
- `bind` löst diese explizit auf

Typische Notation: Dokumentation

Typische Signatur-Notation eines Operators:

```
arg1 ... argn operator result
```

`arg1` bis `argn` werden entfernt
`result` auf den Stack gelegt

Übungen: 1

Was liegt nach Ausführung des Programms

```
8 3 sub
```

auf dem Operand Stack?

Übungen: 2

Was liegt nach Ausführung des Programms

```
/sub {mul} bind def
/mul {add} def
8 3 sub
```

auf dem Operand Stack?

Vergleiche und Conditionals

- `eq`, `ne`, `ge`, `gt`, `le`, `lt`
- `if`, `ifelse`
- `for`, `repeat`, `loop`

Stackmanipulation

- `pop`
- `index`
- `roll`
- `exch`
- `dup`

Übungen: 3

Gebt einen PS-Operator `sub0` mit Signatur

```
a b sub0 c
```

an, der $\max\{a - b, 0\}$ berechnet

Übungen: 4

Gebt einen PS-Operator `exp` mit Signatur

```
a b exp c
```

an, der a^b berechnet ($b \in \mathbf{N}$)

(verwendet nicht das eingebaute `exp`)

Übungen: 5

Gebt einen PS-Operator `invert` mit Signatur

```
any[0]...any[n] k invert any[0]...any[n] any[n-1]...any[n-k]
```

an, der die obersten k Elemente des Stacks invertiert ($k \in \mathbf{N}$)

Grundbegriffe

- Device Space
- User Space
- Transformation Matrix
- Path
- Current Path
- Clipping Path
- Graphics State

Nächste Minilecture

Was machen wir nächstes mal?

- Paths

Was machen wir nächstes mal?

- Paths
- Transformation Matrix

Was machen wir nächstes mal?

- Paths
- Transformation Matrix
- Visuellen Output erzeugen