Kurs ▮ Cours

12. Anonyme Funktionen
▮ Fonctions anonymes

Die Gliederung dieses Kurses folgt in groben Zügen dem Buch von Nancy Blachman: A Practical Approach.... Hinweis: Kapitel 12 lesen!
▮ L'articulation de ce cours correspond à peu près à celle du livre de Nancy Blachman: A Practical Approach....
Indication: Lire le chapitre 12.

Run mit WIN+*Mathematica* Version 5.2
▮ Testé avec *Mathematica* version 5.2+WIN

Anonyme Funktionen sind solche, die keinen Namen haben ....
Hier werden wir sehen, was es damit um sich hat!
▮ Les fonctions anonymes sont celles qui n'ont pas de nom...
Ici nous verrons ce qui en est d'elles!

12.1. "Function"
▮ "Function"

"Function" ist eine allgemeine Funktion ohne speziellen Namen, die gerade "unmittelbar" angewandt wird.
▮ "Function" est une fonction générale sans nom spécial, on peut l'utiliser "immédiatement".

Studiere: ▮ Etudie:

```
In[1]:= ??Function
```

```
Function[body] or body& is a pure function. The formal parameters are # (or #1), #2, etc.
   Function[x, body] is a pure function with a single formal parameter x. Function[
   {x1, x2, ... }, body] is a pure function with a list of formal parameters. Mehr...

Attributes[Function] = {HoldAll, Protected}
```

Beispiele: ■ Exemple:

```
In[2]:= Function[x, x^2][4]

Out[2]= 16

In[3]:= Function[x, x^2 - x][5]

Out[3]= 20

In[4]:= Function[x, x^2][{1, 2, 3, 4, 5, 6, 7}]

Out[4]= {1, 4, 9, 16, 25, 36, 49}
```

Einfach, nicht? ■ Simple, n'est-ce pas?

12.2. Anwendung auf Datenselektion ■ Application à la sélection de données

12.2.1. Mit Hilfe einer klassischen Funktion mit Name (hier "groes45") ■ A l'aide d'une fonction classique avec nom (ici "groes45")

Beispiel (erst Daten erzeugen, dann Funktion definieren, dann anwenden):
■ Exemple (d'abord générer des données, ensuite définir la fonction, puis appliquer):

```
In[5]:= data = Table[Random[Integer, {0, 100}], {14}]

Out[5]= {33, 89, 77, 26, 3, 33, 66, 53, 95, 45, 49, 69, 15, 90}

In[6]:= groes45[x_] := x > 45

In[7]:= Map[groes45,data]

Out[7]= {False, True, True, False, False, False,
          True, True, True, False, True, True, False, True}

In[8]:= groes45[data]

Out[8]= {33, 89, 77, 26, 3, 33, 66, 53, 95, 45, 49, 69, 15, 90} > 45
```

Anwendung zur Selektion:
■ Application à la sélection:

```
In[9]:= Select[data, groes45]

Out[9]= {89, 77, 66, 53, 95, 49, 69, 90}
```

12.2.2. Mit Hilfe einer anonymen Funktion
▮ A l'aide d'une fonction anonyme

Beispiel (erst ausprobieren):
▮ Exemple (essayer d'abord):

```
In[10]:= Function[x, x > 45][68]

Out[10]= True

In[11]:= Function[x, x > 45][24]

Out[11]= False
```

Anwendung zur Selektion:
▮ Application à la sélection:

```
In[12]:= Select[data, Function[x, x > 45]]

Out[12]= {89, 77, 66, 53, 95, 49, 69, 90}
```

12.2.3. Mit Hilfe der Abkürzung "#" für die Variable der anonymen Funktion
▮ A l'aide de l'abréviation "#" pour les variables de la fonction anonyme

```
In[13]:= Select[data, Function[# > 45]]

Out[13]= {89, 77, 66, 53, 95, 49, 69, 90}
```

12.2.4. Mit Hilfe der Abkürzung "( )&" für Function[ ]
▮ A l'aide de l'abréviation "( )&" pour Function[ ]

```
In[14]:= Select[data,  (# > 45)& ]

Out[14]= {89, 77, 66, 53, 95, 49, 69, 90}
```

12.3. Neuauflage einer früher definierten Funktion
▮ Nouvel emploi d'une fonction définie auparavant

Präzision 20 Stellen bei numerischen Rechnungen
▮ Précision de 20 unités lors de calculs numériques

Das Beispiel:
▮ L'exemple:

```
In[15]:= nZwanzig[x_] := N[x, 20]

In[16]:= $Post = nZwanzig

Out[16]= nZwanzig

In[17]:= Sin[1]

Out[17]= 0.84147098480789650665
```

Mit anonymer Funktion:
▮ Avec une fonction anonyme:

```
In[18]:= $Post := N[#, 40]&     (* vierzig *)

In[19]:= Sin[1]

Out[19]= 0.8414709848078965066525023216302989996226
```

12.4. Transformation von Wertepaaren in Regeln
▮ Transformation de paires de valeurs en règles

Aus zwei Listen {x1, x2, x3,... } und {y1, y2, y3,... } soll eine Liste {x1 - > y1, x2 -> y2, x3 -> y3,... } gemacht werden.
▮ Faire de deux listes {x1, x2, x3,... } et {y1, y2, y3,... } une seule {x1 - > y1, x2 -> y2, x3 -> y3,... }.

Listen erzeugen:
▮ Générer des listes de paires:

```
In[20]:= list1 = {x1, x2, x3, x4, x5, x6};
         list2 = {y1, y2, y3, y4, y5, y6};
```

Paarliste erzeugen:
Générer des listes de paires:

```
In[22]:= Transpose[{list1, list2}]

Out[22]= {{x1, y1}, {x2, y2}, {x3, y3}, {x4, y4}, {x5, y5}, {x6, y6}}
```

Anonyme Funktion (#[[1]] -> #[[2]])& auf die Liste setzen (mit "Map"):
■ Appliquer des fonctions anonymes (#[[1]] -> #[[2]])& à des listes (avec "Map"):

```
In[23]:= Map[(#[[1]] -> #[[2]])&, Transpose[{list1,
                                    list2}]]

Out[23]= {x1 → y1, x2 → y2, x3 → y3, x4 → y4, x5 → y5, x6 → y6}
```

12.5. Mehrere Argumente
■ Plusieurs arguments

Beispiel: ■ Exemple:

```
In[24]:= Function[{x, y}, x > y][1, 2]

Out[24]= False

In[25]:= Function[{x, y}, x > y][2, 1]

Out[25]= True

In[26]:= Function[{x, y}, x > y][u, 2]

Out[26]= u > 2.000000000000000000000000000000000000000
```

Was passiert, wenn die Anzahl der übergebenen Argumente nicht stimmt?
■ Que se passe-t-il quand le nombre des arguments transmis n'est pas correct?

```
In[27]:= Function[{x, y}, x > y][1]

         Function::fpct : Too many parameters in {x, y} to be filled from Function[{x, y}, x > y][1]. Mehr…

         Function::fpct : Too many parameters in {x, y} to be filled
            from Function[{x, y}, x > y][1.000000000000000000000000000000000000000]. Mehr…

Out[27]= Function[{x, y}, x > y][1.000000000000000000000000000000000000000]

In[28]:= Function[{x, y}, x > y][1, 2, 3]

Out[28]= False

In[29]:= Function[{x, y}, x > y][a, b, c]

Out[29]= a > b
```

Bei der Kurzschreibweise mit "&" und "#" kommt nun das Problem der Identifikation der verschiedenen Variablen. Die Variablen können dabei nummeriert werden. Beispiel:
■ Les abréviations "&" et "#" posent le problème de l'identification des différentes variables. On peut numéroter les variables. Exemple

In[30]:= **??#**

```
# represents the first argument supplied to a pure function. #n represents the nth argument.
```
Mehr...

```
Attributes[Slot] = {NHoldAll, Protected}
```

In[31]:= **?##**

```
## represents the sequence of arguments supplied to a pure function. ##n represents the
   sequence of arguments supplied to a pure function, starting with the nth argument.
```
Mehr...

In[32]:= **??&**

```
Function[body] or body& is a pure function. The formal parameters are # (or #1), #2, etc.
   Function[x, body] is a pure function with a single formal parameter x. Function[
   {x1, x2, ... }, body] is a pure function with a list of formal parameters.
```
Mehr...

```
Attributes[Function] = {HoldAll, Protected}
```

In[33]:= **?&&**

```
e1 && e2 && ...  is the logical AND function. It evaluates its arguments in order, giving
   False immediately if any of them are False, and True if they are all True.
```
Mehr...

In[34]:= **(#1 < #2)&[4, 6]**

Out[34]= True

Die anonyme Funktion ist jetzt (#1 < #2)&.

La fonction anonyme est maintenant (#1 < #2)&.

12.6. Daten filtern
Filtrer des données

Aufgabe
Problème

Ersetze eine Datenmenge nach einem "Filter" durch die Menge der gewichteten Mittelwerte je dreier sich folgender Elemente. Studiere dazu das folgende Beispiel!

Remplace un ensemble de données après un "Filtre" par l'ensemble des moyennes pondérées de trois éléments qui se suivent. Etudie l'exemple suivant!

Lösung in Einzelschritten:
Solution par étapes:

Zuerst Filter und Daten generieren:

Générer d'abord le filtre et les données:

```
In[35]:= filter = {0.1, 0.8, 0.1};
         datenF  = Table[Random[Integer, {0, 100}]/10,
                                        {10}]
```

```
Out[36]= {3.4000000000000000000000000000000000000000,
          8.8000000000000000000000000000000000000000,
          8.5000000000000000000000000000000000000000,
          1.8000000000000000000000000000000000000000,
          3.3000000000000000000000000000000000000000,
          0.9000000000000000000000000000000000000000,
          8.8000000000000000000000000000000000000000,
          7.0000000000000000000000000000000000000000,
          7.2000000000000000000000000000000000000000,
          6.3000000000000000000000000000000000000000}
```

Ordnen in Dreiergruppen, die dann gewichtet gemittelt werden können:

■ Ordonner en groupes de trois dont on peut faire ensuite la moyenne pondérée:

```
In[37]:= matrix = Partition[datenF, Length[filter], 1]
```

```
Out[37]= {{3.4000000000000000000000000000000000000000,
           8.8000000000000000000000000000000000000000,
           8.5000000000000000000000000000000000000000},
          {8.8000000000000000000000000000000000000000,
           8.5000000000000000000000000000000000000000,
           1.8000000000000000000000000000000000000000},
          {8.5000000000000000000000000000000000000000,
           1.8000000000000000000000000000000000000000,
           3.3000000000000000000000000000000000000000},
          {1.8000000000000000000000000000000000000000,
           3.3000000000000000000000000000000000000000,
           0.9000000000000000000000000000000000000000},
          {3.3000000000000000000000000000000000000000,
           0.9000000000000000000000000000000000000000,
           8.8000000000000000000000000000000000000000},
          {0.9000000000000000000000000000000000000000,
           8.8000000000000000000000000000000000000000,
           7.0000000000000000000000000000000000000000},
          {8.8000000000000000000000000000000000000000,
           7.0000000000000000000000000000000000000000,
           7.2000000000000000000000000000000000000000},
          {7.0000000000000000000000000000000000000000,
           7.2000000000000000000000000000000000000000,
           6.3000000000000000000000000000000000000000}}
```

```
In[38]:= % // MatrixForm
```

```
Out[38]//MatrixForm=
3.4000000000000000000000000000000000000000  8.8000000000000000000000000000000000000000
8.8000000000000000000000000000000000000000  8.5000000000000000000000000000000000000000
8.5000000000000000000000000000000000000000  1.8000000000000000000000000000000000000000
1.8000000000000000000000000000000000000000  3.3000000000000000000000000000000000000000
3.3000000000000000000000000000000000000000  0.9000000000000000000000000000000000000000
0.9000000000000000000000000000000000000000  8.8000000000000000000000000000000000000000
8.8000000000000000000000000000000000000000  7.0000000000000000000000000000000000000000
7.0000000000000000000000000000000000000000  7.2000000000000000000000000000000000000000
```

Die Gewichtung geschieht, indem man die erste Zahl einer Dreiergruppe mit der ersten Zahl des Filters multipliziert, die zweite Zahl des Filters mit der zweiten der Dreiergruppe etc. und das jeweils addiert. Diese Multiplikation mit nachfolgender Addition ist jeweils ein Skalarprodukt entsprechender Vektoren:
■ L pondération se fait en multipliant le 1er nombre d'un groupe de trois avec le 1er nombre du filtre, le 2e nombre du filtre avec le 2e nombre du groupe etc.. On en fait chaque fois l'addition. Cette multiplication suivie de l'addition est un produit scalaire des vecteurs correspondants:

```
In[39]:= Map[filter .#&, matrix]

Out[39]= {8.23, 7.86, 2.62, 2.91, 1.93, 7.83, 7.2, 7.09}
```

Lösung alles auf einmal:
■ Solution en un coup:

```
In[40]:= zusrollen[filterVar_List,
             dataVar_List]:= Map[filterVar . #&,
             Partition[dataVar, Length[filterVar], 1] ]

In[41]:= zusrollen[filter, datenF]

Out[41]= {8.23, 7.86, 2.62, 2.91, 1.93, 7.83, 7.2, 7.09}
```

"Putzmaschine" einsetzen
■ Employer la "machine de nettoyage"

```
In[42]:= (* Old Form: Remove["Global`@*"] *)

In[43]:= Remove["Global`*"]
```