

Kurze Anleitung zum Einstieg in Julia

Herzlich Willkommen in den Hands-on Workshops! Dieses kurze Blatt soll Ihnen die grundsätzliche Bedienung eines Jupyter-Sheets (das ist die Dateiform unserer digitalen Übungsblätter und wir nennen sie oft auch einfach nur kurz "Sheets") erklären und in die Grundrechenarten in Julia einführen. Die Bearbeitung dieses Blattes ist erstmal nicht verpflichtend, die selben Aufgaben werden aber verpflichtender Teil des ersten Sheets zur Analysis I sein und werden benötigt um die kommenden Sheets erfolgreich bearbeiten zu können. Es werden keinerlei Vorkenntnisse im Programmieren vorausgesetzt. Weitere Anwendungen und Befehle in Julia, die wir im ganzen Semester benötigen werden (z.B. Schleifen, Funktionen, Plots) werden Ihnen dann ausführlich auf dem ersten Sheet zur Linearen Algebra I erklärt.

Inhaltsverzeichnis:

- Vorbereitung: Wie bearbeite und exportiere ich ein Jupyter Sheet
 - Grundlegende Rechenoperationen
-

Vorbereitung

Grundsätzliche Tipps zur Bearbeitung der Sheets: Wenn man links neben eine Zelle klickt und dann a bzw. b drückt, wird eine neue Zelle (obendrüber bzw. untendrunter) eingefügt. Alternativ kann man oben in der Leiste das + Symbol anklicken. Die neue Zelle ist dann automatisch eine "Code" Zelle, also zur Durchführung von Rechnungen mit Eingabe von Julia Befehlen geeignet. Wenn Sie in dieser Zelle nichts rechnen wollen, sondern erklärenden Text zu den Lösungen einfügen möchten, können Sie die Zelle im Reiter oben in eine "Markdown" Zelle umwandeln. Um den Befehl in einer Code Zeile auszuführen, muss Shift und Enter gedrückt werden. Um eine Markdown Zelle in "fertigen Text" umzuwandeln, drückt man ebenfalls Shift und Enter. Möchte man die Markdown-Zelle später noch einmal bearbeiten, kommt man durch Doppelklick darauf wieder in den Bearbeitungsmodus.

Aufgabe 1 ***

- a) Erzeugen Sie unter dieser grünen Zelle eine neue Code Zelle, in der Sie berechnen wieviele Minuten 1 Jahr hat (kein Schaltjahr).
*Hinweis: Die Grundrechenarten werden in Julia einfach mit +, -, *, / eingegeben*
Fehlerbehebung: Falls nach Drücken von Shift und Enter kein Ergebnis kommt, gehen Sie sicher, dass Julia überhaupt schon gestartet wurde, oben rechts sollte "Julia 1.4.2" stehen. Falls dort "No Kernel" steht, dann klicken Sie auf "No Kernel" und wählen Sie "Julia 1.4.2" in der Liste aus, ggf. mehrmals hintereinander.
- b) Erzeugen Sie darunter eine Markdown Zelle in der Sie einen Antwortsatz schreiben und die Rechnung kurz erläutern.

Ein Semikolon am Ende des Befehls (bzw. der Rechnung) führt dazu dass der Befehl ausgeführt wird, aber kein Ergebnis angezeigt wird. Wenn mehrere Befehle untereinander stehen, werden alle ausgeführt aber nur das Ergebnis des letzten angezeigt. Wenn auch das Ergebnis eines vorherigen Befehls aus der gleichen Zelle angezeigt werden soll, muss die entsprechende Rechnung in Klammern () gesetzt werden und der Befehl `println` davor geschrieben werden. Alternativ können mehrere Rechnungen mit Kommata getrennt in eine Zeile geschrieben werden, dann werden alle outputs in einem Tupel angeordnet angezeigt.

Kommentare innerhalb von Code-Zellen müssen mit `#` davor gekennzeichnet werden, damit Julia weiß, dass hier nichts zu rechnen ist. Wenn Sie Ihr Sheet als **PDF exportieren** wollen, ist unserer Erfahrung nach der beste Weg so: oben links unter *File* dann *Save and Export Notebook As...* auswählen, und zwar zunächst als HTML Export (nicht PDF!!). Dieser wird in Ihrem Download Ordner gespeichert und wenn Sie die Datei in einem Browser öffnen können Sie sie dort dann als PDF exportieren oder als PDF drucken (z.B. durch Drücken von Strg+P).

Aufgabe 2 ***

- a) Erzeugen Sie eine neue Code Zelle, in der Sie berechnest wieviele Minuten ein Nicht-Schaltjahr bzw ein Schaltjahr hat, sodass beide Ergebnisse angezeigt werden. Probieren Sie es einmal mit Kommata getrennt und einmal mit `println`. Schreiben Sie in die gleiche Zelle einen kurzen Kommentar neben die Rechnung, um zu kennzeichnen, welche Rechnung sich auf das Schaltjahr bezieht bzw. auf das Nicht-Schaltjahr.
- b) Exportieren Sie nun das (bis hierhin) bearbeitete Sheet als PDF.

Bevor wir nun richtig anfangen zu rechnen, laden wir zunächst das package `SymPy`. Mit `SymPy` können wir symbolisch rechnen, beispielsweise bekommen wir bei Rechnungen mit rationalen Zahlen oder Quadratwurzeln dann exakte Ergebnisse und keine gerundeten Fließkommazahlen. (Für Interessierte hier noch der Link zur [Dokumentation](#).)
Hinweis: Es kann ein paar Minuten dauern, bis `SymPy` fertig geladen ist, und Julia wieder bereit (*idle*) ist. Das Package muss nur einmal zu Beginn der Bearbeitung des sheets geladen werden.

```
In [1]: # bitte führe diese Zelle einmal aus (reinklicken und Shift+Enter)  
  
using SymPy
```

Grundlegende Rechenoperationen

Brüche können mit `/` eingegeben werden, dann rechnet Julia allerdings mit gerundeten Fließkommazahlen. Soll exakt gerechnet werden, dann müssen alle Brüche, die vorkommen mit `//` eingegeben werden.

Aufgabe 3 *** Berechnen Sie in der Code Zelle unten $\frac{5}{3} + \frac{17}{11}$. Vergleichen Sie die Eingabe der drei verschiedenen Befehle

- `5/3+17/11`
- `5//3+17//11`
- `5//3+17/11`

```
In [2]: # Lösung Aufgabe 3
```

In den folgenden Beispielen illustrieren wir, dass auch bei der Eingabe von Quadratwurzeln oder trigonometrischen Ausdrücken zwischen symbolischen Ausdrücken und näherungsweisen Rechnungen unterschieden werden kann:

```
In [2]: sqrt(2) # Quadratwurzel aus 2 (sqrt steht für das englische Wort squareroot)
```

```
Out[2]: 1.4142135623730951
```

```
In [3]: sympy.sqrt(2) # So geben wir Wurzel aus 2 an, wenn damit symbolisch gerechnet werden soll
```

```
Out[3]:  $\sqrt{2}$ 
```

```
In [4]: pi # Die Zahl pi
```

```
Out[4]:  $\pi = 3.1415926535897\dots$ 
```

In [6]: `sympy.pi` *# So geben wir pi ein, wenn damit symbolisch gerechnet werden soll.*

Out[6]: π

In [7]: `cos(2)` *# So erhalten wir ein gerundetes Ergebnis*

Out[7]: -0.4161468365471424

In [8]: `sympy.cos(2)` *# So geben wir cos(2) ein, wenn damit symbolisch gerechnet werden soll.*

Out[8]: $\cos(2)$

In [9]: `sympy.sin(sympy.pi)` *# Exaktes Ergebnis*

Out[9]: 0

In [10]: `sin(pi)` *# wenn man nicht symbolisch rechnet, erhält man die Ergebnisse nur näherungsweise.*

Out[10]: 1.2246467991473532e-16

In [11]: `sympy.sin(pi)` *# bei gemischten Ausdrücken rechnet Julia nicht symbolisch (nur wenn man es erzwingt).*

Out[11]: $1.22464679914735 \cdot 10^{-16}$

Freiwillige Aufgabe *** Recherchieren Sie im Internet, was Maschinengenauigkeit ist und welche Maschinengenauigkeit mit heutigen Computern erreicht wird. Sie werden dann merken, dass Julia hier mit $1.22... \cdot 10^{-16}$ ein innerhalb der Maschinengenauigkeit korrektes Ergebnis liefert.

Aufgabe 4 ***

- a) Berechne unten $\frac{5}{3} + \sqrt{17 + \sin(3)}$ einmal symbolisch und einmal näherungsweise. Welche der beiden Rechnungen hilft weiter, wenn wir in einer Anwendung wissen möchten ob der Ausdruck kleiner als 6 ist?
- b) Berechne unten $\cos\left(\frac{\pi}{2}\right)$ einmal symbolisch und einmal näherungsweise. Welche der beiden Rechnungen hilft weiter, um mit Sicherheit sagen zu können, dass hier eine Nullstelle vorliegt?

In [12]: `#a`

In [13]:

```
#a
```

In [14]:

```
#b
```

In [15]:

```
#b
```

Wir listen nun noch weitere nützliche Grundrechenarten auf:

- $x^{(1/n)}$ berechnet (näherungsweise) $\sqrt[n]{x}$ für eine Zahl x und eine natürliche Zahl n
- `factorial(n)` berechnet die Fakultät $n!$ einer natürlichen Zahl n
- `abs(x)` berechnet den Absolutbetrag $|x|$ einer Zahl x
- `n%m` berechnet den Rest von n geteilt durch m
- `sum(f(i) for i=1:n)` berechnet $\sum_{i=1}^n f(i)$, also beispielsweise `sum(i for i=1:10)` für $\sum_{i=1}^{10} i$

Aufgabe 5 ***

- a) Berechne unten $|13^7 - 13!|$.
- b) Berechne den Rest von 140 geteilt durch 13.
- c) Berechne die Summe der ersten 100 Quadratzahlen.

In [16]:

```
#a
```

In [17]:

```
#b
```

In [18]:

```
#c
```

Freiwillige Aufgabe *** Wenn man eine Maschine für sich rechnen lässt, sollte man sicherstellen, dass man sich auf das Ergebnis verlassen kann, d.h. dass der eingegebene Befehl für die entsprechende Anwendung ausgelegt ist. Geben Sie in Julia einmal 2^{64} und einmal 2.0^{64} ein. Sie werden sehen, dass im ersten Fall ein falsches Ergebnis und im zweiten Fall ein (näherungsweise) korrektes Ergebnis ausgegeben wird. Dies liegt an der Verwendung der verschiedenen Datentypen Int64 bzw. Float64 ein (im ersten Fall werden die ganzen Zahlen bit-weise in Darstellung zur Basis 2 dargestellt, und da nur 64 Bits zur Verfügung stehen ist die Zahl 2^{64} schlicht zu groß!). Wenn man ein genaues Ergebnis möchte, müsste man mit dem Datentyp BigInt rechnen. Geben Sie dazu $\text{big}(2)^{64}$ ein und vergleiche mit dem Ergebnis im Datentyp Float64. Lesen Sie sich [hier] (<https://einocs.julialang.org/en/v1/base/numbers/#Data-Formats>) in der Julia Dokumentation etwas in die verschiedenen Datentypen ein.
