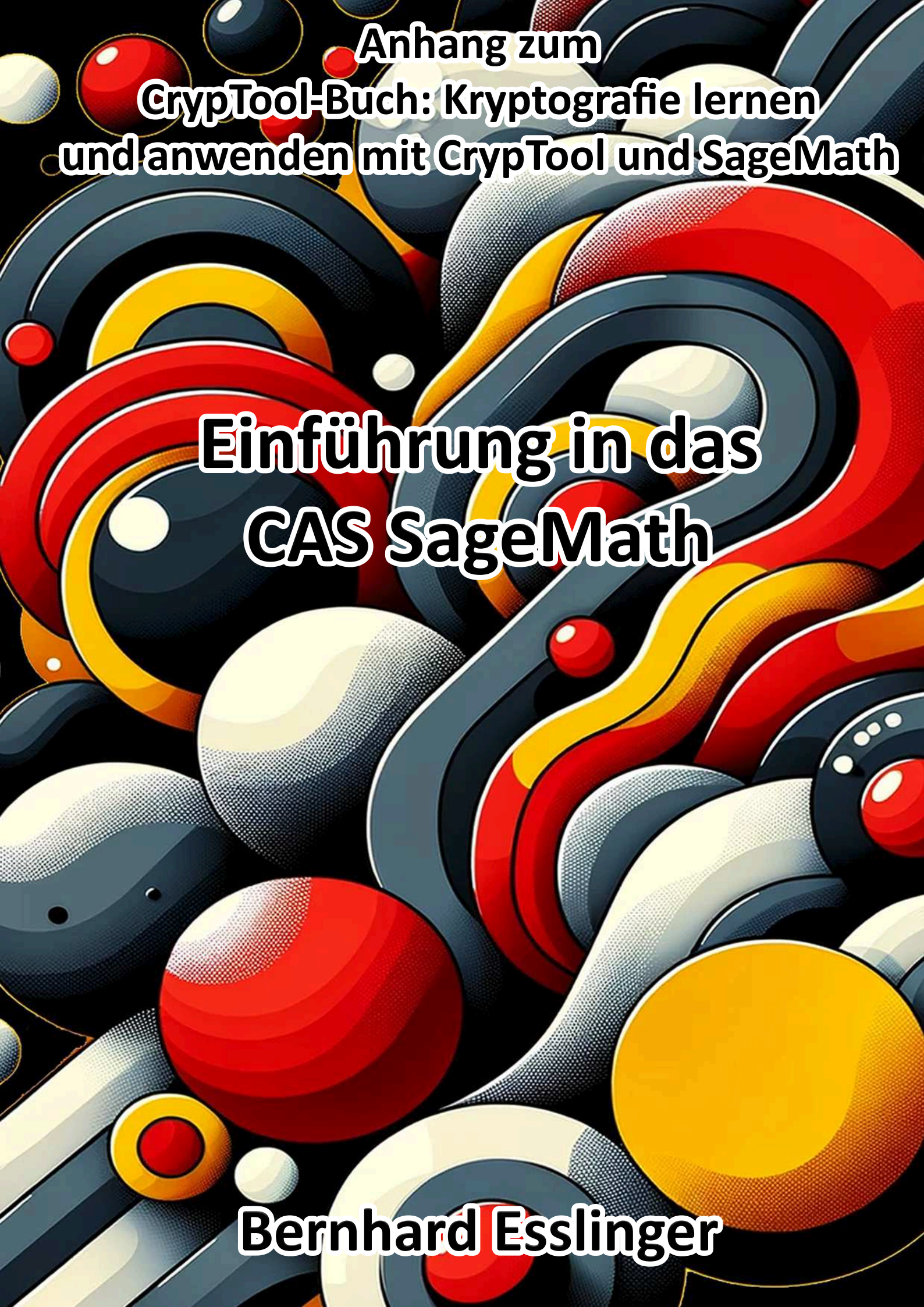




**Anhang zum
CrypTool-Buch: Kryptografie lernen
und anwenden mit CrypTool und SageMath**

Einführung in das CAS SageMath

Bernhard Esslinger

Anhang zum “CrypTool-Buch”:
Kryptografie lernen und anwenden mit CrypTool und SageMath

Einführung in das CAS SageMath

Bernhard Esslinger

<https://www.cryptool.org>

28. Juni 2024 (13:24:00)

Dies ist ein freies Dokument, d. h. Inhalte des Dokuments können kopiert und verbreitet werden, auch zu kommerziellen Zwecken. Im Gegenzug sind Autor, Titel und die CrypTool-Webseite (<https://www.cryptool.org>) zu nennen. Selbstverständlich darf aus dem CrypTool-Buch, genau wie aus jedem anderen Werk auch, zitiert werden. Darüber hinaus unterliegt dieses Dokument der GNU-Lizenz für freie Dokumentation. Dies betrifft auch den Code für die SageMath- und OpenSSL-Beispiele in diesem Dokument.

Copyright © 1998–2024 Bernhard Esslinger and the CrypTool Team.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation (FSF). A copy of the license is included in the section entitled “[GNU Free Documentation License](#)”.

Zum Referenzieren per Bib(La)T_EX:

```
@article{Esslinger:ctb_2024_app_sage_de,  
  editor    = {Bernhard Esslinger},  
  title     = {Anhang zum CrypTool-Buch: Kryptografie lernen  
              und anwenden mit CrypTool und SageMath:  
              Einführung in das CAS SageMath},  
  publisher = {CrypTool-Projekt},  
  year      = {2024}  
}
```

Schriftsatz-Software: L^AT_EX

Versionsverwaltungs-Software: Git

Unterstützung bei der Übersetzung: DeepL-Übersetzer von DeepL.com und ChatGPT 4 von openai.com

Unterstützung beim Generieren von Tikz-Grafik: Claude 3.5 Sonnet von claude.ai

Inhaltsverzeichnis

1	Einführung in das CAS SageMath	5
1.1	Die drei üblichen SageMath-Benutzerschnittstellen	6
1.2	Beispiele für in SageMath eingebaute mathematische Funktionen	8
1.3	Hilfe beim Benutzen von SageMath	12
1.3.1	Hilfe von Webseiten	12
1.3.2	Hilfe per <code>help()</code> , <code>?</code> , <code>??</code> oder <code>search_src</code>	14
1.3.3	Nutzen der Tab-Vervollständigung (Tab-Completion)	15
1.3.4	Die vollständige Befehlsliste: Der SageMath-Index	16
1.4	Bedienen des Jupyter-Notebooks	17
1.5	Der Kernel eines Jupyter-Notebooks	21
1.6	Programmieren mit SageMath in der Sage-Konsole oder per Start eines Sage-Skripts im Terminal	22
1.7	SageMath und $\text{\LaTeX}$	24
1.7.1	$\text{\LaTeX}$ und SageMath auf der Konsole	25
1.7.2	$\text{\LaTeX}$ und SageMath im Jupyter-Notebook	26
1.7.3	Im $\text{\LaTeX}$ -Dokument den von <code>latex()</code> erzeugten Sage-Befehl nutzen	31
1.7.4	Im $\text{\LaTeX}$ -Dokument von <code>SageTeX()</code> bearbeiteten Sage-Code nutzen	33
1.8	SageMath mit Jupyter und <code>interact</code>	35
1.8.1	Ein typisches <code>interact()</code> -Beispiel	35
1.8.2	Technisches – was sind Decorators	36
1.8.3	Interact-Beispiele ohne Grafik	37
1.8.4	Interact-Beispiele mit Grafik	38
1.9	SageMath mit Jupyter und Matplotlib <code>interactive_output</code>	40
1.10	Weitere <code>interact</code> SageMath-Beispiele zur Kryptografie	41
1.11	Der Verlauf der MTW-Punkteverteilungs-Kurven mit SageMath	42
1.12	Professionellere Sage-Programme	43
1.13	Weitere Hinweise zu SageMath in diesem Buch	44
	Dank	44
2	Verzeichnisse der Abbildungen, Tabellen, Code-Beispiele, etc.	45
2.1	Abbildungsverzeichnis	45
2.2	Tabellenverzeichnis	45
2.3	Verzeichnis der SageMath-Programmbeispiele	46
3	Gesamtliteraturverzeichnis	47
4	Index	48

1 Einführung in das CAS SageMath

(Bernhard Esslinger. Letzter Update: Juni 2024)

Dieses Kapitel, das als Anhang zum CrypTool-Buch konzipiert wurde, beschreibt vor allem das Ökosystem von SageMath, also wie benutzt man SageMath in verschiedenen Umgebungen und wie spielen diese zusammen. Einsteiger sind von den vielen Varianten oft verwirrt – hier hilft dieses Kapitel mit Beispielen und einem systematischen Überblick, der in den Tabellen 1 und 2 zusammengefasst wird.

Das CrypTool-Buch enthält zahlreiche mit SageMath erstellte Programmbeispiele. Dieser Anhang kann aber auch eigenständig als Einführung in SageMath gelesen werden. SageMath ist ein Computer-Algebra-System (CAS) wie Mathematica, Maple oder MATLAB. SageMath wird für Lehre, Studium und Forschung eingesetzt. Es kombiniert viele weitere hochwertige Open-Source-Pakete¹ und liefert den Zugang zu deren Funktionalität über ein gemeinsames, auf der Programmiersprache Python basierendes Interface.²

SageMath ist kostenlos und kann von folgender Webseite herunter geladen werden:

<https://www.sagemath.org>

Verwendung findet SageMath als mächtiger Taschenrechner; als Tool für das Mathematikstudium; als Programmier-Umgebung, um Algorithmen zu prototypen und um Forschung im Bereich der algorithmischen Aspekte der Mathematik zu betreiben. Statt mathematische Aufgaben mit SageMath zu programmieren, kann man genauso gut auch Python und Bibliotheken benutzen. Der Vorteil von SageMath ist, dass etliche mathematische Konstrukte schon fertig dabei sind und man auch symbolische Mathematik machen kann.³

Einen schnellen Einstieg bieten z. B. die Referenzen in Fußnote 4.⁴ Zum Weiterlesen seien in Fußnote 5 beispielhaft vier gute Dokumente zu SageMath genannt.⁵

Auch beim Studium der Kryptologie können SageMath-Module genutzt werden.⁶ Kryptografie-Einführungen, die SageMath nutzen, finden sich in dieser Fußnote⁷. In Abschnitt 2.8 des CrypTool-Buchs sind SageMath-Beispiele für viele klassische Krypto-Verfahren enthalten.

Diese Einführung zeigt vor allem anhand von Beispielen, was SageMath leisten kann, aber es zeigt auch die Umgebungen, also wie man SageMath im Terminal und im Notebook verwendet; wie die von SageMath mitgelieferten Bedienungs-Controls mit denen von Jupyter und Matplotlib zusammen hängen; oder wie LaTeX

¹ Normalerweise installiert man nur die Binaries von SageMath. Compiliert man es selbst, erhält man einen Eindruck von der Größe von SageMath: Die heruntergeladenen Sourcen von SageMath 4.1 brauchten zur Compilierung auf einem durchschnittlichen Linux-PC rund 5 h (inklusive aller Bibliotheken). Danach nahm es 1,8 GB Plattenplatz ein. Bei SageMath 9.0 war das bz2-File rund 2 GB, und ausgepackt > 7 GB groß. Die Compilierung von SageMath 10.3 dauerte bei mir unter Ubuntu 22.04 rund 40 min – anhand der Anweisungen von https://sagemanifolds.obspm.fr/install_ubuntu.html.

² Es gibt auch ein Interface für die Sprache C, genannt Cython, mit dem man eigene Funktionen in SageMath beschleunigen kann. Siehe https://openware.org/wiki/Open_writing_projects/Sage_and_cython_a_brief_introduction. SageMath läuft unter den Betriebssystemen Linux, macOS und Windows. Seit SageMath-Version 8 gibt es einen eigenen Windows-Installer; seit SageMath 9 (released am 1.1.2020) nutzt SageMath Python 3 (statt Python 2).

³ <https://doc.sagemath.org/html/en/reference/calculus/sage/calculus/calculus.html>

⁴ - *The SDSU Sage Tutorial*,
<https://mosullivan.sdsu.edu/sagetutorial/>
<https://mosullivan.sdsu.edu/sagetutorial/sagecalc.html>
 - *Sage Quick Reference Cards*, <https://wiki.sagemath.org/quickref>
⁵ - „Bibliothek“: <https://www.sagemath.org/library.html>,
 - *Dokumentationsprojekt*: <https://wiki.sagemath.org/DocumentationProject>,
 - *Lehrmaterial*: https://wiki.sagemath.org/Teaching_with_SAGE,
 - „Sage als Taschenrechner im Gymnasium“: https://doc.sagemath.org/html/de/thematische_anleitungen/sage_gymnasium.html.
 Dies ist ein sehr guter Einstieg.

⁶ - Überblick, welche Kryptografie momentan in SageMath enthalten ist:
<https://doc.sagemath.org/html/en/reference/cryptography/index.html>
 - Lernaspekte beim Entwickeln weiterer Krypto-Module in SageMath:
<https://groups.google.com/g/sage-devel/c/xVcsTY1C0IE>

⁷ - Ein ganzer Kryptografie-Kurs von David Kohel aus 2015: <https://www.sagemath.org/files/kohel-book-2008.pdf>
 - *Introduction to Cryptography with Open-Source Software*, ein hervorragendes Buch von Alasdair McAndrew, CRC, 2011

und SageMath zusammen wirken können. Diese Gesamtschau fehlte in vielen anderen Einführungen. Ergänzt wird es mit vielen kommentierten Verweisen, die sich als nützlich für Lehre und Lernen erwiesen.

Alle SageMath-Beispiele dieser Einführung sind auf der CrypTool-Website zu finden:

<https://www.cryptool.org/de/ctbook/sagemath>

1.1 Die drei üblichen SageMath-Benutzerschnittstellen

Tabelle 1 führt drei Arten auf, wie SageMath meist genutzt wird: für das Terminal und das Jupyter-Notebook jeweils zwei (lokale) Arten und zwei Beispiele per Webseite.⁸

1) Terminal-basiert	a) Sage-Konsole	\$ sage sage: factor(35) sage: ... sage: attach datei.sage	Text-Ausgabe in die Zeilen nach dem Kommando. Grafik-Ausgabe in eine Datei.
	b) Sage-Programm	\$ sage skript.sage	
2) Jupyter-Notebook	a) ausgeführt im Browser	Starten des Kernels: \$ sage -n jupyter <datei.ipynb>	Text- und Grafik-Ausgabe in die Zellen-Ausgabe. Grafik-Ausgabe auch in eine Datei.
	b) per VS Code		
3) Externe Webseite	Per Browser Remote-Zugriff auf eine externe Webseite	https://sagecell.sagemath.org/ oder https://cocalc.com/	Ausgabe im Browser

Tab. 1: Überblick über die SageMath-Aufrufmöglichkeiten (Benutzerschnittstellen)

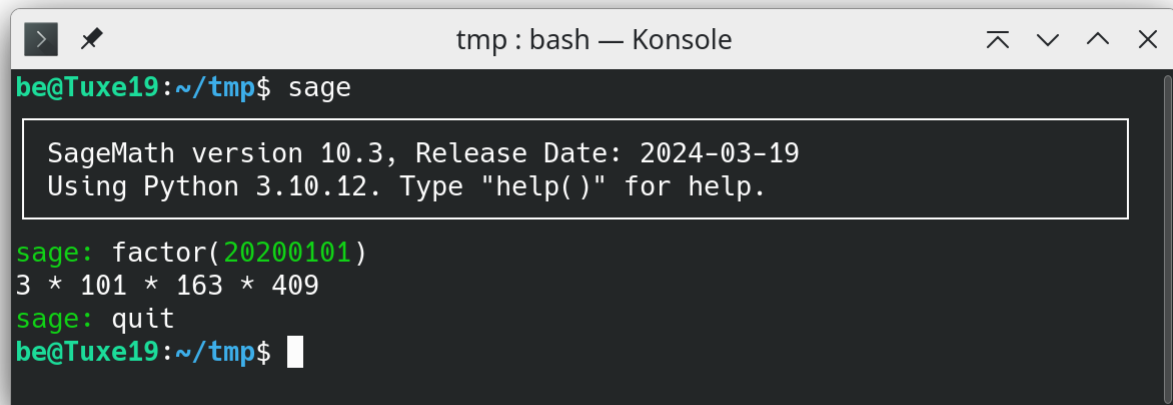
1. Die erste Benutzer-Schnittstelle ist **Kommandozeilen-** oder **Terminal-**basiert (siehe Abb. 1 auf der nächsten Seite). Dies wird normalerweise verwendet, wenn man SageMath lokal installiert hat.

Von der Kommandozeile (Terminal, Shell) des Betriebssystems aus hat man zwei Möglichkeiten:⁹

- a) Entweder wird die interaktive Sage-Konsole (`sage:`)¹⁰ aufgerufen. Textausgaben erscheinen unterhalb des eingegebenen Befehls. Wenn Sie hier eine grafische Ausgabe generieren, erzeugt SageMath eine Datei und ruft asynchron ein Bildverarbeitungs-Programm auf, um das Bild anzuzeigen.
 - b) Als zweite Möglichkeit im Terminal kann man SageMath-Programme (Skripte) schreiben und diese wie eine Batchdatei ausführen (`$ sage skript.sage`). Dies wird genutzt für längere Berechnungen und typische Aufgaben in Programmen.
2. Zweitens gibt es für SageMath auch eine grafische Benutzeroberfläche (genannt Notebook). Am beliebtesten ist heutzutage das **Jupyter-Notebook** (siehe Abb. 2). Diese Möglichkeit ist gut geeignet für interaktive oder didaktische Aufgaben, und wird häufig auch von den Datenanalysten benutzt. Jupyter-Notebooks sind ähnlich zum REPL-Modell (Read-Evaluate-Print-Loop). Mehr zu Jupyter findet man im Abschnitt 1.7.2 auf Seite 26 und in 1.4 auf Seite 17.

Die zwei häufigsten Varianten, ein Jupyter-Notebook zu nutzen sind:

- a) im Browser
- b) in Visual Studio Code (abgekürzt VS Code oder VSC)



```
tmp : bash — Konsole
be@Tuxe19:~/tmp$ sage
SageMath version 10.3, Release Date: 2024-03-19
Using Python 3.10.12. Type "help()" for help.
sage: factor(20200101)
3 * 101 * 163 * 409
sage: quit
be@Tuxe19:~/tmp$
```

Abb. 1: SageMath-Konsole von der Kommandozeile (Terminal)

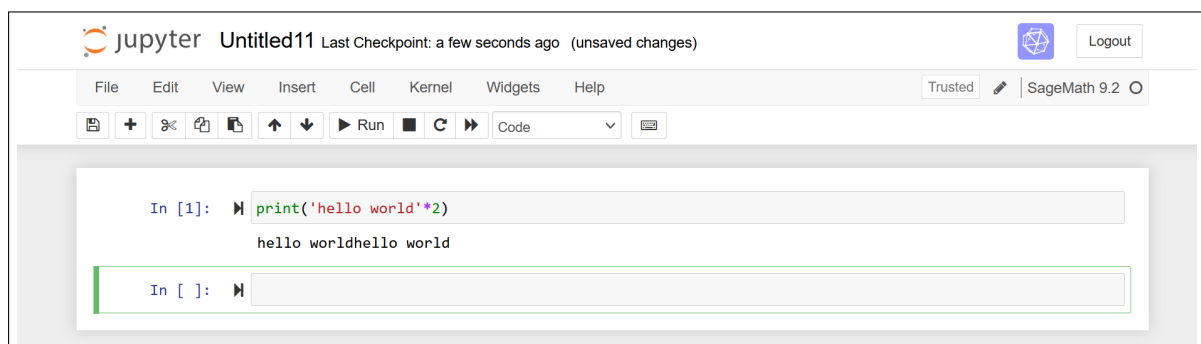


Abb. 2: Erstellen einer SageMath-Datei mit Jupyter im Browser

Anmerkung 1: VS Code mit selbstgebauter SageMath-Installation

Baut man sich seine SageMath-Installation selbst, bspw. weil die vorhandenen Binaries in den Distributionen nicht neu genug sind¹¹ und hat noch eine alte SageMath-Version installiert, kann es vorkommen, dass VS Code als möglichen Kernel nur die alte Sage-Version anzeigt.

Wird der neue Kernel in VS Code nicht aufgeführt, kann man folgendermaßen vorgehen. Voraussetzung, um das zu beheben: Der Kernel (Jupyter-Server) ist gestartet per `$ sage -n jupyter`. Man klickt auf die Kernel-Ikone (oder klickt Strg+Shift+P und wählt „Select Kernel“). Dann wählt man nacheinander:

Select Another Kernel --> Existing Jupyter Server und gibt die URL des laufenden Jupyter-Servers ein, bspw. `http://127.0.0.1:8888/tree?token=...`. Danach steht auch der aktuelle SageMath-Kernel (bspw. 10.3) zur Verfügung.

Anmerkung 2: SageMath-Funktionen in einer Python-Datei nutzen

Wenn SageMath lokal installiert ist, kann man in einer Python-Datei auch SageMath-Funktionen nutzen. Dazu fügt man `from sage.all import *` an den Anfang der Python-Datei ein. Die Operatoren bleiben aber die von Python, d. h. `**` bewirkt Exponentiation und `^` bedeutet XOR; während bei SageMath `^` die Exponentiation bewirkt. Ein Minimalbeispiel (`test_sage-in-py-file.py`) dazu finden Sie auf der CrypTool-Webseite: <https://www.cryptool.org/de/ctbook/sagemath/>.

Anmerkung 3: Evtl. Zukunft: SageMath rein lokal im Browser

Es gibt erste Versuche, auch eine WebAssembly-basierte Version von SageMath zu erstellen, so dass SageMath auch im Browser lokal laufen kann (also nicht nur das Frontend, sondern auch der Kernel). Von in SageMath genutzten Bibliotheken gibt es schon WebAssembly-basierte Versionen, die man im Internet ausprobieren kann, z. B. von Pari-GP: <https://github.com/sagemathinc/wasm-pari>.

1.2 Beispiele für in SageMath eingebaute mathematische Funktionen

Nachdem die grundsätzlichen Möglichkeiten der Installation und Nutzung von SageMath gezeigt wurden, soll kurz beschrieben werden, was SageMath bietet. SageMath-Beispiele 1.1 und 1.2 auf dieser Seite und auf Seite 10 zeigen exemplarisch, was man auf der Kommandozeile mit SageMath machen kann.¹²

SageMath-Beispiel 1.1 enthält Aufrufe zum Umgang mit Zahlen (z. B. Länge bestimmen), aus der Zahlentheorie (z. B. Primzahlen, zu einer gegebenen Zahl teilerfremde Zahlen oder auch Primitivwurzeln finden), der Trigonometrie und dem Umgang mit Polynomen (Analysis, symbolische Ausdrücke).

Interessant ist, wie SageMath mit symbolischen Ausdrücken umgeht. Wenn Sie eine mathematische Funktion als symbolischen Ausdruck mit symbolischen Variablen definieren, kann SageMath unterscheiden zwischen „Argumenten“ als unabhängigen Variablen einerseits und Parametern, also Variablen, von denen erwartet wird, dass sie feste Werte annehmen, andererseits.

SageMath-Beispiel 1.1: Kleine Beispiele aus verschiedenen Gebieten der Mathematik (1)

```
print("\n# SageAppendix--SAMPLE 010: =====")

# Allgemein: Bestimmen der Länge der dezimalen und der binären Darstellung von 26!
# General: Determine the length of the decimal and the binary representation of 26!

print("14.digits():", 14.digits()) # Show the digits of an integer number
```

⁸ See <https://doc.sagemath.org/html/en/installation/launching.html>.

⁹ Die Unterschiede sind zusammengefasst in Abschnitt 5.12.5.4 des CrypTool-Buchs.

¹⁰ https://doc.sagemath.org/html/de/tutorial/interactive_shell.html

¹¹ Im Mai 2024 gab es auf macOS per brew die SageMath-Release-Version 10.3, aber unter Ubuntu hatte das neueste Pre-Built Binary nur die Version 9.5. Auch im Windows Subsystem for Linux WSL bekam man 9.5 (die früheren Pre-Built-Binaries für Windows auf Basis von Cygwin sind veraltet). Siehe auch <https://doc.sagemath.org/html/en/installation/>.

¹² Diese Beispiele stammen größtenteils aus einem nicht mehr existierenden Blog von Alasdair McAndrew.


```

b=factorial(26)
print("Factorial b:", b)
print("Factorial b.n():", b.n())
print("Factorial b.n(prec=16):", b.n(prec=16))
print("Factorial b.ndigits():", b.ndigits())
print("Factorial b.ndigits(base=2):", b.ndigits(base=2))
print("Factorial b.nbits():", b.nbits())

# Zahlentheorie / Number theory:
print("\n14.coprime_integers(max=16):", 14.coprime_integers(16))
i=randint(2^49,2^50); p=next_prime(i)
print("Next prime of %d: p=%d" % (i,p))
# p=1022095718672689 # for testing
r=primitive_root(p)
print("Primitive_root(p):", r)
pl=log(mod(10^15,p),r)
print("pl:", pl)
print("mod(r,p)^pl", mod(r,p)^pl)

# Trigonometrie (SageMaths Trigonometrie-Funktionen nutzen Bogenmaß statt Grad; pi=180°)
# Trigonometry (Sage's trigonometry functions use radians instead of degrees; pi=180°)
print("\nsin(pi/6):", sin(pi/6)) # 30° (Grad / degrees)

# Analysis (Infinitesimalrechnung) / Calculus // Symbolic expression:
# x=var('x')
var('a, x') # symbolic variables (can be parameters or independent variables)
p1=a*exp(x^2)
print("\nExpression p1: ", p1)
p2=diff(p1,x,10) # compute tenth derivative
print("Expression p2: ", p2)
p(x)=diff(p1,x,10)*exp(-x^2) # via "(x)" only x is considered independent variable
print("Expression p: ", p)
print("Polynom p simplified:", p.simplify_full())# now a polynomial since exp canceled out
print("Variables in p: {0}; Arguments in p: {1}".format(
    p.variables(), p.arguments())) # an "argument" is an independent variable
print("Type of p: {0}".format(type(p)))

#-----
# SageAppendix--SAMPLE 010: =====
# 14.digits(): [4, 1]
# Factorial b: 403291461126605635584000000
# Factorial b.n(): 4.03291461126606e26
# Factorial b.n(prec=16): 4.033e26
# Factorial b.ndigits(): 27
# Factorial b.ndigits(base=2): 89
# Factorial b.nbits(): 89
#
# 14.coprime_integers(max=16): [1, 3, 5, 9, 11, 13, 15]
# Next prime of 816414536954577: p=816414536954609
# Primitive_root(p): 3
# pl: 171454160901578
# mod(r,p)^pl 183585463045391
#
# sin(pi/6): 1/2
#
# Expression p1: a*e^(x^2)
# Expression p2: 1024*a*x^10*e^(x^2) + 23040*a*x^8*e^(x^2) + 161280*a*x^6*e^(x^2) + 403200*a*x^4*e^(x^2) +
# 302400*a*x^2*e^(x^2) + 30240*a*e^(x^2)
# Expression p: x |--> 32*(32*a*x^10*e^(x^2) + 720*a*x^8*e^(x^2) + 5040*a*x^6*e^(x^2) + 12600*a*x^4*e^(x^2)
# 9450*a*x^2*e^(x^2) + 945*a*e^(x^2))*e^(-x^2)
# Polynom p simplified: 1024*a*x^10 + 23040*a*x^8 + 161280*a*x^6 + 403200*a*x^4 + 302400*a*x^2 + 30240*a
# Variables in p: (a, x); Arguments in p: (x,)
# Type of p: <class 'sage.symbolic.expression.Expression'>

```

SageMath-Beispiel 1.2 enthält Aufrufe aus der Linearen Algebra (LA) und Funktionen zu endlichen Körpern.

Vektoren werden in der linearen Algebra manchmal als Zeilenvektoren und manchmal als Spaltenvektoren dargestellt. Beide Darstellungen sind äquivalent und treten in der mathematischen Literatur auf. SageMath kann mit beiden Darstellungen umgehen, da es mit sog. „generischen“ Vektoren arbeitet. Definiert man etwa einen Vektor $v = \text{vector}([1, 2])$ und eine Matrix $A = ([3, 4], [5, 6])$, so kann man v sowohl von links als auch von rechts an A heranmultiplizieren. Der Ausdruck $A * v$ verwendet v als Spaltenvektor und $v * A$ verwendet v als Zeilenvektor. Der Vektor muss nur die richtige Länge haben. Um einen expliziten Zeilenvektor aus einem „generischen“ Vektor zu erstellen, verwenden Sie einfach die Methode `row()`. Das Ergebnis hat dann aber den Typ `Matrix`. Und nur auf den Typ `Matrix` kann man die `transpose()`-Methode anwenden. Gibt man `v.transpose()` ein, erhält man eine Fehlermeldung – bei `v.row().transpose()` dagegen nicht.

SageMath-Beispiel 1.2: Kleine Beispiele aus verschiedenen Gebieten der Mathematik (2)

```
print("\n# SageAppendix--SAMPLE 020: =====")

# Lineare Algebra / Linear algebra:

u = zero_vector(SR, 10) # create a zeros vector (Nullvektor); SR = Symbolic Ring
print("Zero vector u:\n", u, sep="")

O = matrix.ones(SR, 2, 10) # create 2 ones vectors. As there is no function like ones_vector(),
                           # define the vector as the first row of a matrix of ones, or
print("Ones matrix O:\n", O, sep="")

v = vector(SR, [1]*10) # [1]*10 is the Python way to construct a list with 10 repeats of 1.
print("Ones vector v:\n", v, sep="")

M=matrix([[1,2,3],[4,5,6],[7,8,10]])
print("M echo:\n", M.echelon_form(), sep="") # show the echelon basis matrix

c=random_matrix(ZZ,3,1)
print("Random vector c:\n", c, sep="")
print("Random vector c.transpose():\n", c.transpose(), sep="")
b=M*c
print("Matrix * vector: b = M*c:\n", b, sep="")
print("Using inverse matrix M^-1 * b:\n", M^-1*b, sep="")

A = matrix([[1,1,0],[0,2,0],[0,0,3]]) # 3*3
print("A:", type(A))
print("rows:", A.nrows()) # 3 # find out the number of rows of a matrix
print("cols:", A.ncols()) # 3 # find out the number of cols of a matrix
print("A echo:\n", A.echelon_form(), sep="") # show the echelon basis matrix

v = A.column(1) # (1, 2, 0) # get the 2nd column (results in a 3*1 vector)
print("v:", type(v))
print("len(v):", v.length()) # Alternative: len(v)
print(v)

print("A*v:", A * v) # (3, 4, 0) # results in a 3*1 vector
print("v*A:", v * A) # (1, 5, 0) # results in a 1*3 vector

r = v.row() # [1 2 0] # REMARK: Vector.row() and .column() create a matrix type
print("r:", type(r))
print("r=v.row():", r) # Remark: len(r) doesn't work, as r is a matrix (1*3)
c = r.transpose() # [1] [2] [0] # REMARK: r,c are type matrix (not vector).
print("c:", type(c))
print("c=transposed r:\n", c, sep='')

# vector multiplication
print("r*c:", r * c) # [5]
print("c*r:\n", c * r, sep='') # [1 2 0] [2 4 0] [0 0 0]
```

```

# multiplication of matrix and vector (explicit a row or col one)
# What does not work is: A * r and c * A
print("r*A:", r * A) # (1, 5, 0) # results in a 1*3 vector
print("A*c:\n", A * c, sep='') # (3, 4, 0) # results in a 3*1 vector

# Create LaTeX command to typeset a matrix equation
# print('$ A*c=b: %s * %s = %s $(\text{latex}(A), \text{latex}(c), \text{latex}(A*c)))

# Endliche Körper (\url{http://de.wikipedia.org/wiki/Endlicher_K%C3%B6rper})
# Finite Fields (\url{http://en.wikipedia.org/wiki/Finite_field})
F.<x>=GF(2)[]
G.<a>=GF(2^4,name='a',modulus=x^4+x+1)
print("\nIn GF: a^2/(a^2+1): ", a^2/(a^2+1))
print("a^100: ", a^100)
print("log(a^2,a^3+1): ", log(a^2,a^3+1))
print("(a^3+1)^13: ", (a^3+1)^13)

#-----
# SageAppendix--SAMPLE 020: =====
# Zero vector u:
# (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
# Ones matrix 0:
# [1 1 1 1 1 1 1 1 1 1]
# [1 1 1 1 1 1 1 1 1 1]
# Ones vector v:
# (1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
# M echo:
# [1 2 0]
# [0 3 0]
# [0 0 1]
# Random vector c:
# [-1]
# [ 0]
# [-5]
# Random vector c.transpose():
# [-1  0 -5]
# Matrix * vector: b = M*c:
# [-16]
# [-34]
# [-57]
# Using inverse matrix M^-1 * b:
# [-1]
# [ 0]
# [-5]
# A: <class 'sage.matrix.matrix_integer_dense.Matrix_integer_dense'>
# rows: 3
# cols: 3
# A echo:
# [1 1 0]
# [0 2 0]
# [0 0 3]
# v: <class 'sage.modules.vector_integer_dense.Vector_integer_dense'>
# len(v): 3
# (1, 2, 0)
# A*v: (3, 4, 0)
# v*A: (1, 5, 0)
# r: <class 'sage.matrix.matrix_integer_dense.Matrix_integer_dense'>
# r=v.row(): [1 2 0]
# c: <class 'sage.matrix.matrix_integer_dense.Matrix_integer_dense'>
# c=transposed r:
# [1]
# [2]
# [0]

```

Fortsetzung SageMath-Beispiel 1.2

```
# r*c: [5]
# c*r:
# [1 2 0]
# [2 4 0]
# [0 0 0]
# r*A: [1 5 0]
# A*c:
# [3]
# [4]
# [0]
#
# In GF: a^2/(a^2+1): a^3 + a
# a^100: a^2 + a + 1
# log(a^2,a^3+1): 13
# (a^3+1)^13: a^2
```

1.3 Hilfe beim Benutzen von SageMath

Wenn man SageMath auf der Kommandozeile startet und `help()` eingibt, erhält man Ausgaben wie in SageMath-Beispiel 1.3:

SageMath-Beispiel 1.3: SageMath-Hilfe (Welcome)

```
sage: help()
Welcome to Sage 10.3!
```

To view the Sage tutorial in your web browser, type `"tutorial()"`, and to view the (very detailed) Sage reference manual, type `"manual()"`. For help on any Sage function, for example `"matrix_plot"`, type `"matrix_plot?"` to see a help message, type `"help(matrix_plot)"` to see a very similar message, type `"browse_sage_doc(matrix_plot)"` to view a help message in a web browser, and type `"matrix_plot??"` to look at the function's source code.

(When you type something like `"matrix_plot?"`, `"help(matrix_plot)"`, or `"matrix_plot??"`, Sage may start a paging program to display the requested message. Type a space to scroll to the next page, type `"h"` to get help on the paging program, and type `"q"` to quit it and return to the `"sage:"` prompt.)

For license information for Sage and its components, read the file `"COPYING.txt"` in the top-level directory of the Sage installation, or type `"license()"`.

To enter Python's interactive online help utility, type `"python_help()"`. To get help on a Python function, module or package, type `"help(MODULE)"` or `"python_help(MODULE)"`.

1.3.1 Hilfe von Webseiten

Die offizielle SageMath-Dokumentation wird mit jedem Release von SageMath verteilt (siehe Abb. 3). Dazu gehören folgende Dokumente:

- Tutorial – Das Tutorial ist für SageMath-Einsteiger. Es ist dafür gedacht, sich in ein bis drei Stunden mit den wichtigsten Funktionen vertraut zu machen.

- **Constructions** – Dieses Dokument ist im Stil eines „Kochbuchs“ mit einer Sammlung von Antworten auf Fragen zur Konstruktion von SageMath-Objekten.
- **Developers' Guide** – Dieser Führer ist für Entwickler, die selbst SageMath mit weiter entwickeln wollen. Enthalten sind darin z. B. Hinweise zum Stil und zu Konventionen beim Programmieren, zur Modifikation von SageMath-Kern-Bibliotheken oder von SageMath-Standard-Dokumentation, und zum Code-Review und zur Software-Verteilung.
- **Reference Manual** – Dieses Handbuch enthält die komplette Dokumentation aller wichtigen SageMath-Funktionen. Zu jeder Klassen-Beschreibung gibt es normalerweise mehrere Code-Beispiele. Alle Code-Beispiele im Referenz-Handbuch werden erneut getestet, wenn ein neues SageMath-Release veröffentlicht wird.
- **Installation Guide** – Dieser Führer erklärt, wie man SageMath auf verschiedenen Plattformen installiert.
- **A Tour of Sage** – Diese Tour durch SageMath zeigt exemplarisch verschiedene Funktionen, die für Einsteiger sinnvoll sind.
- **Numerical Sage** – Dieses Dokument führt Werkzeuge auf, die in SageMath für numerische Mathematik verfügbar sind.
- **Three Lectures about Explicit Methods in Number Theory Using Sage** – Drei Vorlesungen über Methoden der Zahlentheorie, die explizit SageMath nutzen. Dieses Dokument zeigt wie man mit SageMath Berechnungen in fortgeschrittener Zahlentheorie durchführt.

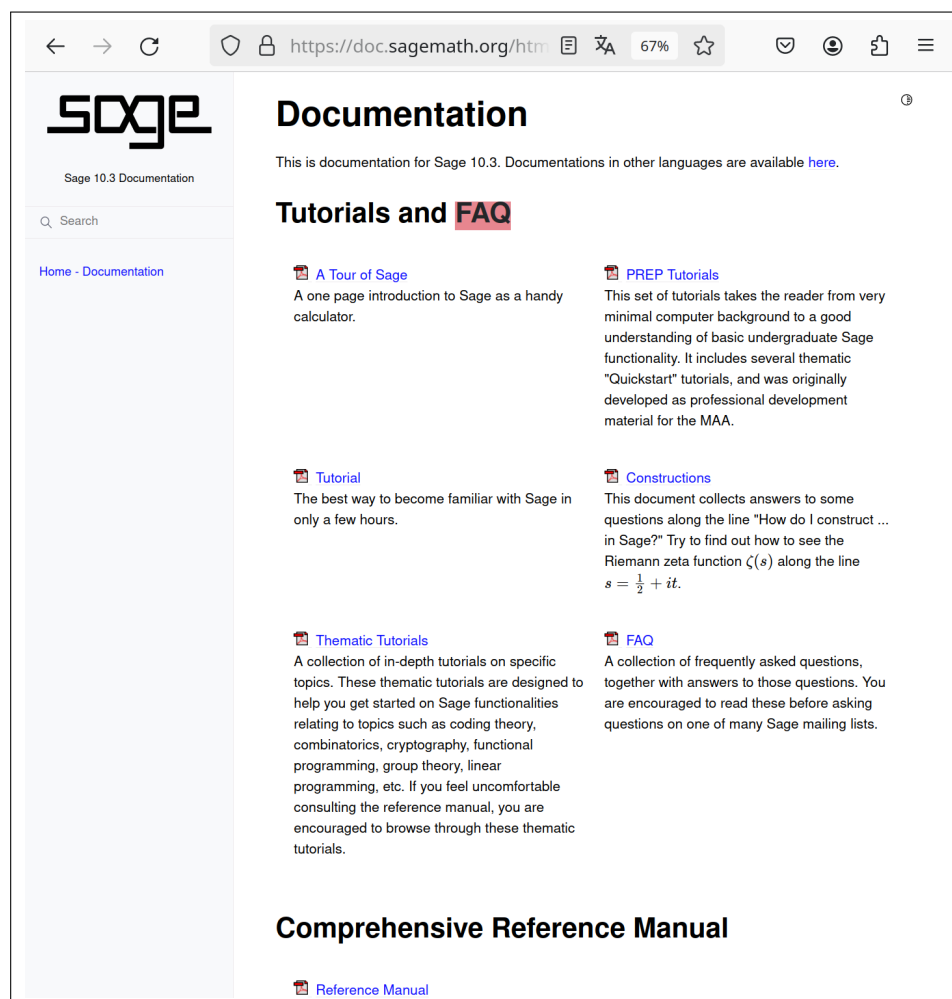


Abb. 3: Webseite zur Standard-Dokumentation von SageMath (<https://doc.sagemath.org/html/en/>)

Weitere Unterstützung für spezifische Probleme gibt es

- in den Archiven der sage-support Mailing-Liste unter <https://groups.google.com/g/sage-support>
- auf <https://ask.sagemath.org>
- auf <https://math.stackexchange.com/questions/tagged/sagemath>
- auf <https://doc.sagemath.org/html/en/reference/genindex.html>

1.3.2 Hilfe per `help()`, `?`, `??` oder `search_src`

Kennt man den genauen Namen einer Funktion, kann man `help` oder das Fragezeichen „?“ nutzen. Zum Beispiel liefern die Kommandos `help(SubstitutionCryptosystem)` bzw. `SubstitutionCryptosystem?` die Dokumentation zu der angegebenen eingebauten Klasse (siehe SageMath-Beispiel 1.4). In beiden Fällen blättert man im Dokumentationsmodus weiter, indem man die Leertaste betätigt und man verlässt den Dokumentationsmodus mit „q“. Bei `help(fname)` wird der Hilfetext nicht auf den Bildschirm der aktuellen Session ausgegeben sondern in einem Extrafenster, das man nach dem Lesen wieder schließt. So wird die aktuelle Session nicht „zugemüllt“.

Man kann das Fragezeichen auch *vor* den Suchbegriff schreiben, also `?fname`. Außerdem kann man statt einem auch zwei Fragezeichen verwenden, also `fname??` oder `??fname`. In diesem Fall erhält man dann zusätzlich noch den Quellcode, der `fname` definiert.

Zuletzt sei noch auf die verschiedenen Funktionen hingewiesen, die man auf <https://doc.sagemath.org/html/en/reference/misc/sage/misc/sagedoc.html> findet, insbesondere `search_src`, `my_getsource` oder `search_def`. Diese können hilfreich sein, wenn die normale Hilfe nicht weiterhilft.

SageMath-Beispiel 1.4: SageMath-Hilfe (zu Einzelfunktion)

```
sage: help(SubstitutionCryptosystem)

SubstitutionCryptosystem(...)
    Create a substitution cryptosystem.

INPUT:
- "S" - a string monoid over some alphabet

OUTPUT:
- A substitution cryptosystem over the alphabet "S".

EXAMPLES::
sage: M = AlphabeticStrings()
sage: E = SubstitutionCryptosystem(M)
sage: E
Substitution cryptosystem on Free alphabetic string monoid on A-Z
sage: K = M([ 25-i for i in range(26) ])
sage: K
ZYXWVUTSRQPONMLKJIHGFEDCBA
sage: e = E(K)
sage: m = M("THECATINTHEHAT")
sage: e(m)
GSVXZGRMGVSZG

TESTS::
sage: M = AlphabeticStrings()
sage: E = SubstitutionCryptosystem(M)
sage: E == loads(dumps(E))
True
```

1.3.3 Nutzen der Tab-Vervollständigung (Tab-Completion)

Wir stellen hier die Vervollständigung mit Hilfe der [Tab]-Taste für die Suche nach zwei unterschiedlichen Suchbegriffen vor: zum Finden der Standard-Kommandos und zum Finden der Methoden der Sage-Objekte.

Tab-Completion zum Finden eines Kommandos Von der SageMath-Kommandozeile erhält man eine Liste aller verfügbaren Kommandos (Standardbefehle, Funktionsnamen etc.), die ein bestimmtes Muster haben, wenn man die ersten Zeichen tippt, und dann die [Tab]-Taste drückt wie in SageMath-Beispiel 1.5:

SageMath-Beispiel 1.5: SageMath-Hilfe (Ausdehnen mit Tab-Taste)

```
sage: Su[TAB]
Subsets          Subwords          SuperPartitions
SubstitutionCryptosystem Sudoku          SupersingularModule
SubwordComplex    SuperPartition    SuzukiGroup

sage: su[TAB]
subfactorial      sum              supersingular_D
subsets           sum_of_k_squares supersingular_j
sudoku            super            surfaces
```

Tab-Completion zum Finden der Methoden eines Sage-Objekts Um alle Methoden (member functions, Attribute) eines Sage-Objekts AA aufzulisten, geben Sie einfach AA. ein und drücken dann die [Tab]-Taste auf der Tastatur. Einen möglichen Output zeigt Abb. 4.

```
sage: AlphabeticStrings.
AlphabeticStrings.Element    AlphabeticStrings.base    AlphabeticStrings.category    Alp...gs.coerce_map_from
AlphabeticStrings.Hom        AlphabeticStrings.base_ring    Alp...gs.characteristic_frequency    Alp...gs.convert_map_from
AlphabeticStrings.alphabet    AlphabeticStrings.cardinality    AlphabeticStrings.coerce    AlphabeticStrings.dump
AlphabeticStrings.an_element    AlphabeticStrings.categories    Alp...gs.coerce_embedding    AlphabeticStrings.dumps
```

Abb. 4: Output nach Punkt bei Tab-Vervollständigung

Auflisten der Methoden eines Sage-Objekts per Programm Anstatt die [Tab]-Taste zu verwenden, kann man auch ein kleines Programm schreiben, das alle Methoden eines Sage-Objekts (hier AA) ausgibt – wie in SageMath-Beispiel 1.6.

SageMath-Beispiel 1.6: Programm, um alle Methoden eines SageMath-Objekts auszugeben

```
sage: AA = AlphabeticStrings()
sage: object_methods = [method_name for method_name in dir(AA)
....:                    if callable(getattr(AA, method_name))]; object_methods
['CartesianProduct',
 'Element',
 'Hom',
 '__bool__',
 ...,
 'unrank',
 'unrank_range',
 'variable_name',
 'variable_names']
```

1.3.4 Die vollständige Befehlsliste: Der SageMath-Index

Es gibt auch eine komplette, alphabetisch geordnete Befehlsübersicht zu SageMath im Internet. Abb. 5 zeigt diese Seite, den sog. Index. Klickt man oben einen der Buchstaben an, erhält man alle Befehle, die mit diesem Buchstaben beginnen und kann darin weiter suchen wie in Abb. 6. Alternativ kann man die umfangreiche Liste aller Befehle anzeigen.

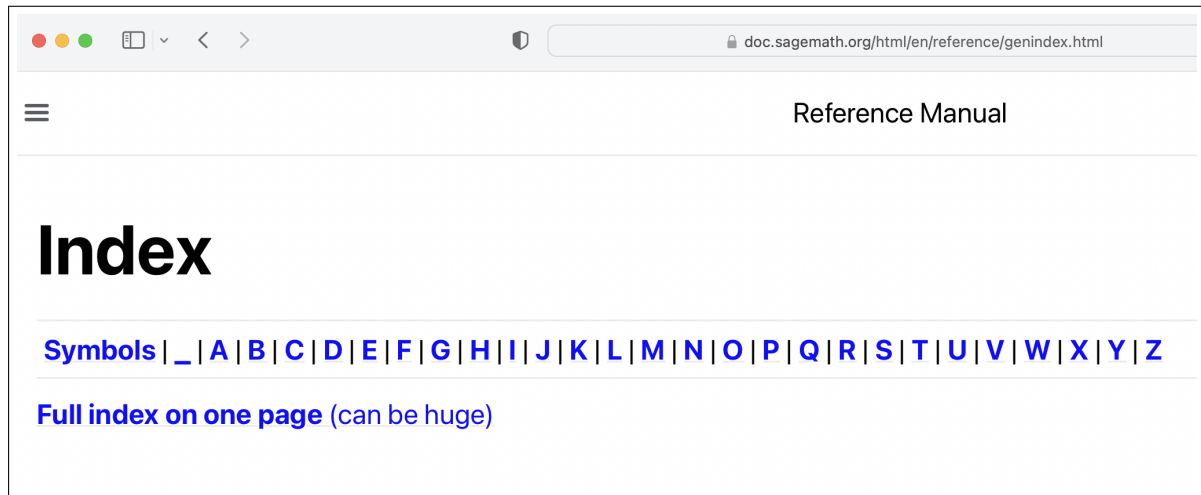


Abb. 5: Index

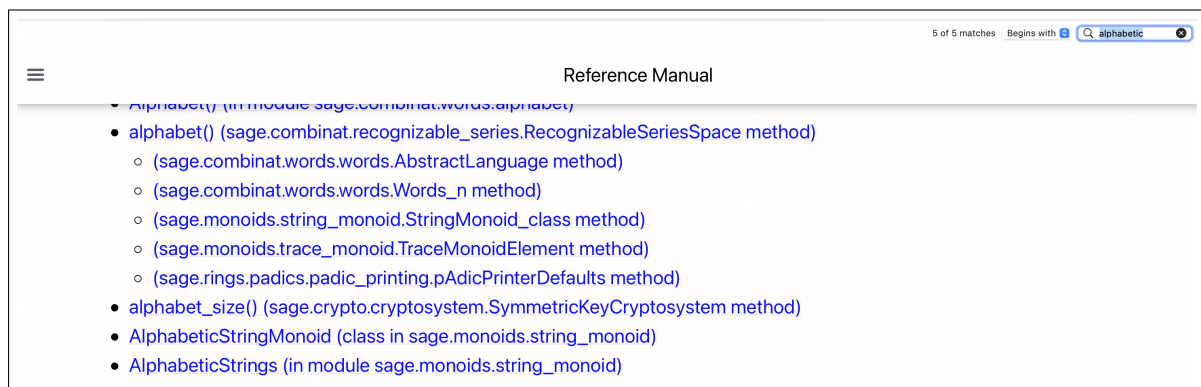


Abb. 6: Suche innerhalb der Befehle, die mit dem Buchstaben *a* beginnen.

1.4 Bedienen des Jupyter-Notebooks

Bisher haben wir die Funktionen von SageMath im Terminal gezeigt. Hier beschreiben wir, was ein Jupyter-Notebook ist und wie man es im Browser und in Visual Studio Code (VSC) verwendet.¹³

Ein Jupyter-Notebook (früher IPython Notebook) ist eine Browser-basierte interaktive Umgebung, mit der Jupyter-Notebook-Dokumente erstellt werden können. Die Dateinamens-Endung ist „ipynb“. IPython wurde ursprünglich als erweiterter Python-Terminal-Interpreter entwickelt.¹⁴

Formal ist ein Jupyter-Notebook eine JSON-Datei und besteht aus einer Reihe von Zellen – siehe Abb. 7.

Es gibt drei Arten von Zellen: Code-Zellen, Markdown-Zellen und Roh-Zellen. Die Standardeinstellung ist zunächst „Code“. Roh-Zellen werden eher selten verwendet. Markdown-Zellen (selten auch Textzellen genannt) werden für die Dokumentation verwendet. Pro Zelle wird der Zelltyp angezeigt und kann dort oder über das Menü geändert werden.

Codezellen bestehen aus einem Eingabeteil und einem Ausgabeteil. Der Ausgabeteil (die Ausgabe der Zelle) zeigt das Ergebnis an, nachdem der Code in der Eingabe der Zelle vom Kernel (hier SageMath) ausgeführt wird.¹⁵ Mehr über den Kernel in Abschnitt 1.5. So wird zum Beispiel eine Codezeile, die mit einem Zahlenzeichen (#) beginnt, in kursiver grüner Schrift formatiert, als Kommentar interpretiert und vom Notebook nicht ausgeführt. Die Ausgabe der Zelle wird direkt unter der Eingabe der Code-Zelle angezeigt und kann sowohl Text und als auch grafische Ausgaben enthalten. Die grafische Ausgabe kann auch interaktive Teile enthalten. Siehe Abb. 8 und 9.

Der Editor des Jupyter-Notebooks hat zwei Modi: den Bearbeitungsmodus (Edit) und den Befehlsmodus (Command). Im Bearbeitungsmodus und im Befehlsmodus können Sie unterschiedliche Tastenkombinationen verwenden. Wenn Sie in eine Zelle klicken, wechseln Sie in den Bearbeitungsmodus.

Beim Ausführen einer Zelle wird ihr Code mit allen zugehörigen Operationen ausgeführt. Eine Zelle kann also immer nur im Ganzen ausgeführt werden. Zum Ausführen klicken Sie in der Werkzeugleiste auf „Run“, oder klicken Sie auf „Cells > Run Cells“. Alternativ können Sie auch **Strg+Eingabetaste** (Ctrl+Enter) drücken, was sowohl im Editier- wie im Kommando-Modus geht: Dann wird die Zelle ausgeführt, in der sich der Mauszeiger befindet. Diese Tastenkombination ist gegenüber dem Mausklick die bevorzugte Methode zum Ausführen einer Zelle:¹⁶

- **Strg+Eingabe** wertet die aktuelle Zelle aus und hält den Fokus in der aktuellen Zelle.
- **Shift+Eingabe** wertet die aktuelle Zelle aus und verschiebt den Fokus auf die nächste Zelle.

Im Befehlsmodus (Command Mode) sind spezielle Tastaturbefehle verfügbar. Um den Befehlsmodus aufzurufen, drücken Sie **Esc** oder klicken Sie mit der Maus außerhalb des Editorbereichs einer Zelle. Der Kommandomodus wird durch einen grauen Zellenrand mit einem blauen linken Rand angezeigt.

Ein paar nützliche Tastatur-Kurzbefehle im Kommandomodus:

- **Eingabe+O** schaltet ein- und aus, ob die folgende Ausgabezelle angezeigt wird oder nicht
- **a** fügt eine Zelle vor der aktiven Zelle ein
- **b** fügt eine Zelle nach der aktiven Zelle ein
- **d d** löscht die ausgewählte(n) Zellen

¹³ Wenn die Jupyter-Extension installiert ist, reicht es, eine Datei mit der Endung „ipynb“ zu laden, damit VS Code in die Notebook-Ansicht umschaltet.

¹⁴ https://de.wikipedia.org/wiki/Project_Jupyter, <https://jupyter.org/> und https://www.tutorialspoint.com/jupyter/ipython_introduction.htm

¹⁵ <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>

¹⁶ Es gibt viele Shortcuts, die die Bedienung deutlich flüssiger gestalten. Siehe <https://jupyter-tutorial.readthedocs.io/de/latest/worksapce/jupyter/notebook/shortcuts.html>

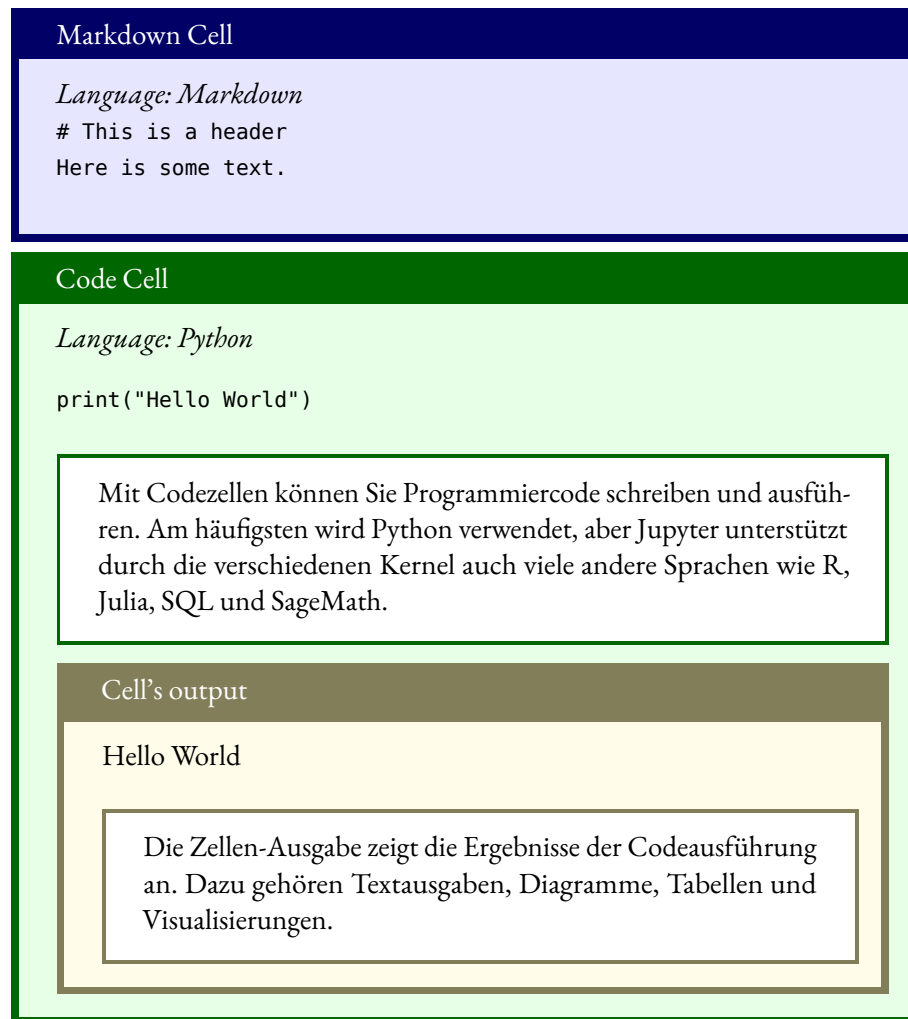


Abb. 7: Typische generelle Abfolge von Zellen in einem Jupyter-Notebook

Wenn man ein Jupyter-Notebook (wieder) öffnet, empfiehlt es sich, zuerst alle Zellen auszuführen, indem man in die erste Zelle geht und auf den Befehl „Execute cell and all below“ klickt.

Anmerkung: Für das Zurücksetzen der Variablen in einer Zelle gibt es drei Kommandos:

reset()

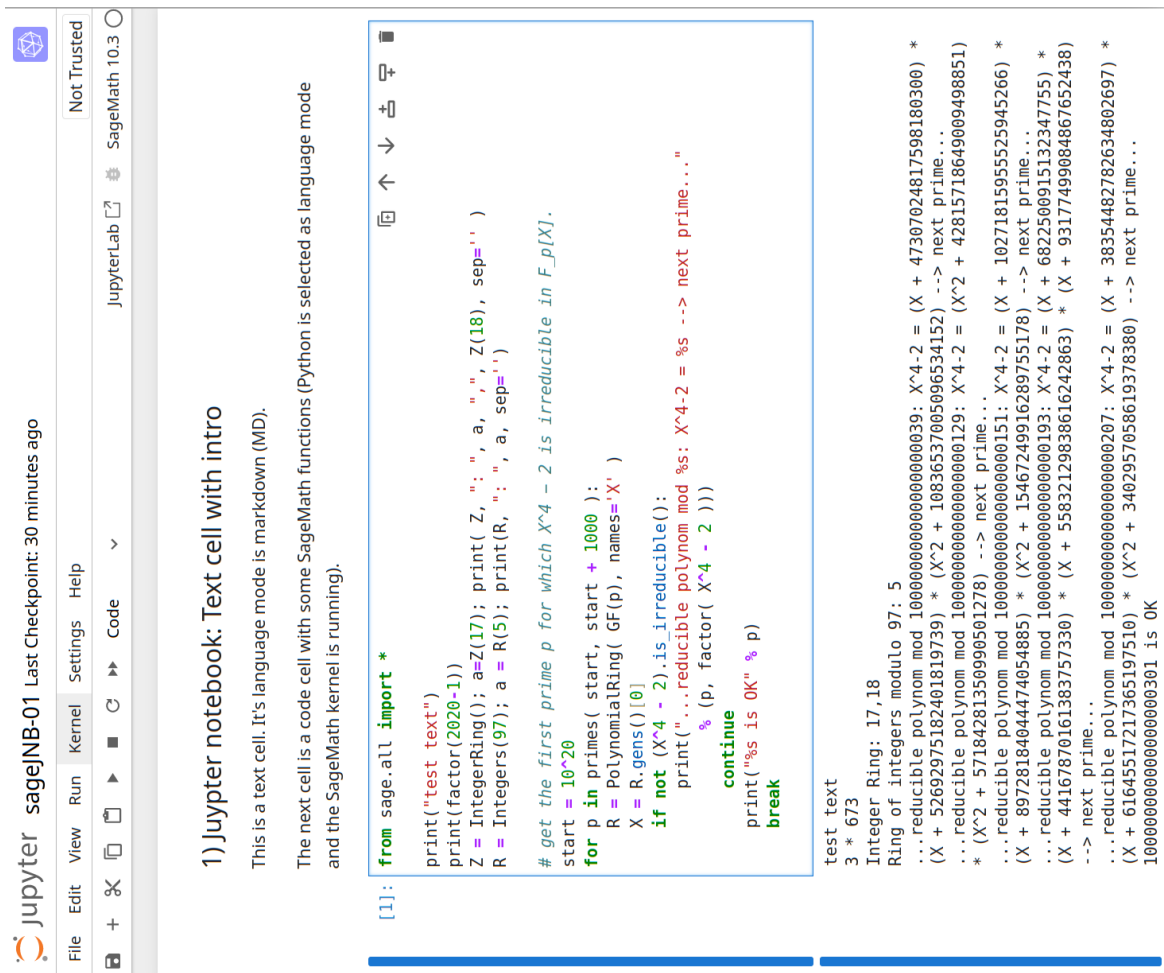
Löscht selbst definierte Variablen und setzt alle globalen Variablen zurück auf ihre Standardwerte.

restore()

Setzt alle vordefinierten globalen Variablen wie QQ zurück auf ihre Standardwerte.

clear_vars()

Löscht alle symbolischen 1-Zeichen-Variablen, die beim Start von SageMath schon definiert sind.



The screenshot shows a Jupyter Notebook titled 'sagejNB-01' with a last checkpoint of 30 minutes ago. The interface includes a top bar with 'JupyterLab' and 'SageMath 10.3.0' tabs, and a left sidebar with 'File', 'Edit', 'View', 'Kernel', 'Run', 'Settings', and 'Help' menus. The main area displays a code cell with the following SageMath code:

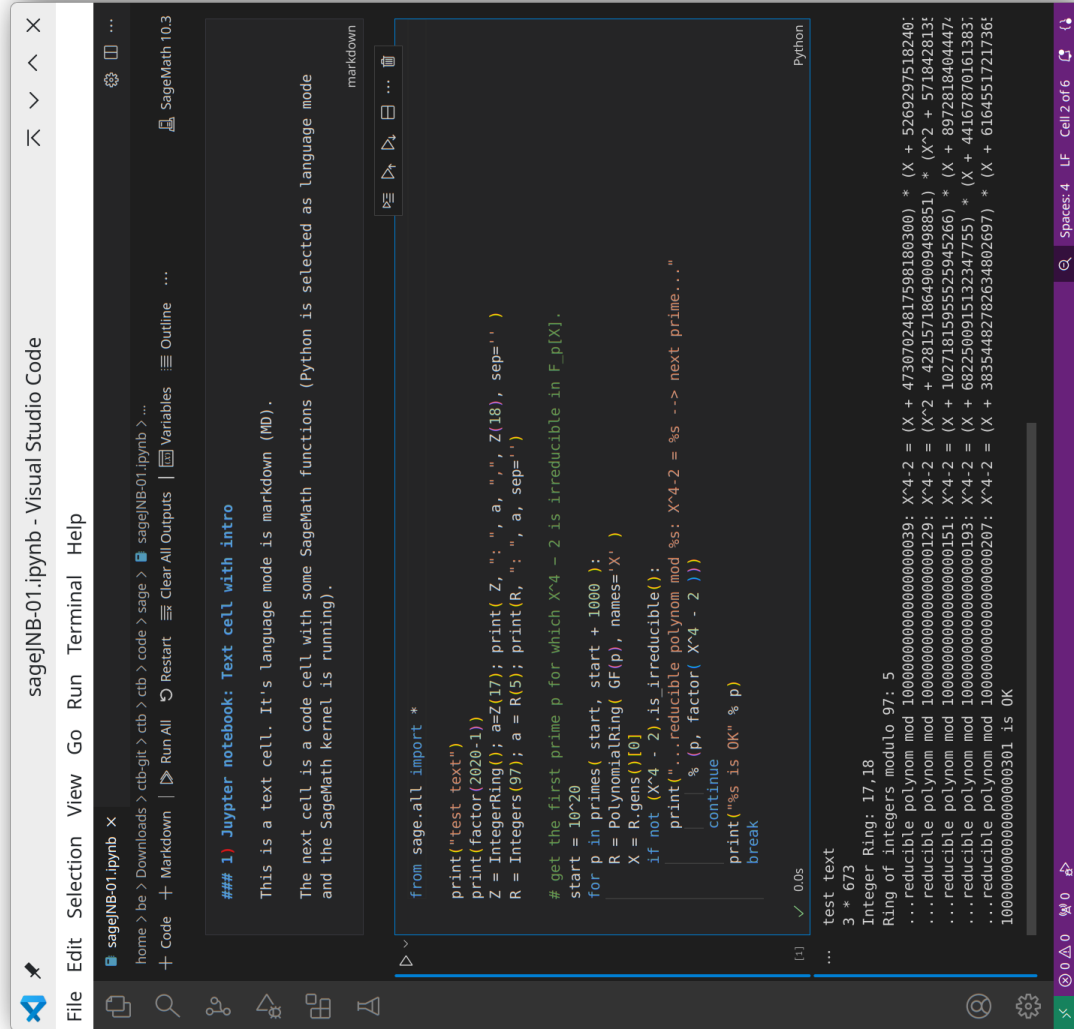
```
[1]: from sage.all import *
print("test text")
print(factor(2020-1))
Z = IntegerRing(); a=Z(17); print( Z, ":", a, ":", Z(18), sep=' ' )
R = Integers(97); a = R(5); print(R, ":", a, sep=' ')

# get the first prime p for which X^4 - 2 is irreducible in F_p[X].
start = 10^20
for p in primes( start, start + 1000 ):
    R = PolynomialRing( GF(p), names='X' )
    X = R.gens()[0]
    if not (X^4 - 2).is_irreducible():
        print("...reducible polynomial mod %s: X^4-2 = %s --> next prime..."
              % (p, factor( X^4 - 2 )))
        continue
    print("%s is OK" % p)
    break
```

The output of the code cell is as follows:

```
test text
3 * 673
Integer Ring: 17, 18
Ring of integers modulo 97: 5
...reducible polynomial mod 1000000000000000000039: X^4-2 = (X + 47307024817598180300) *
(X + 52692975182401819739) * (X^2 + 10836537005096534152) --> next prime...
...reducible polynomial mod 1000000000000000000129: X^4-2 = (X^2 + 42815718649009498851)
* (X^2 + 57184281350990501278) --> next prime...
...reducible polynomial mod 1000000000000000000151: X^4-2 = (X + 10271815955525945266) *
(X + 89728184044474054885) * (X^2 + 15467249916289755178) --> next prime...
...reducible polynomial mod 1000000000000000000193: X^4-2 = (X + 6822500915132347755) *
(X + 44167870161383757330) * (X + 55832129838616242863) * (X + 93177499084867652438)
--> next prime...
...reducible polynomial mod 1000000000000000000207: X^4-2 = (X + 38354482782634802697) *
(X + 61645517217365197510) * (X^2 + 3402957058619378380) --> next prime...
10000000000000000000301 is OK
```

Abb. 8: Typische Abfolge von Zellen in einem Jupyter-Notebook für Sage-Code mit Text-Output im Browser und in VSC



The screenshot shows a Visual Studio Code interface with a file named 'sagejNB-01.ipynb' open. The interface includes a top bar with 'sagejNB-01.ipynb - Visual Studio Code' and a left sidebar with 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help' menus. The main area displays a code cell with the following SageMath code:

```
from sage.all import *
print("test text")
print(factor(2020-1))
Z = IntegerRing(); a=Z(17); print( Z, ":", a, ":", Z(18), sep=' ' )
R = Integers(97); a = R(5); print(R, ":", a, sep=' ')

# get the first prime p for which X^4 - 2 is irreducible in F_p[X].
start = 10^20
for p in primes( start, start + 1000 ):
    R = PolynomialRing( GF(p), names='X' )
    X = R.gens()[0]
    if not (X^4 - 2).is_irreducible():
        print("...reducible polynomial mod %s: X^4-2 = %s --> next prime..."
              % (p, factor( X^4 - 2 )))
        continue
    print("%s is OK" % p)
    break
```

The output of the code cell is as follows:

```
test text
3 * 673
Integer Ring: 17, 18
Ring of integers modulo 97: 5
...reducible polynomial mod 1000000000000000000039: X^4-2 = (X + 47307024817598180300) * (X + 52692975182401819739) * (X^2 + 10836537005096534152) --> next prime...
...reducible polynomial mod 1000000000000000000129: X^4-2 = (X^2 + 42815718649009498851) * (X + 57184281350990501278) --> next prime...
...reducible polynomial mod 1000000000000000000151: X^4-2 = (X + 10271815955525945266) * (X + 89728184044474054885) * (X^2 + 15467249916289755178) --> next prime...
...reducible polynomial mod 1000000000000000000193: X^4-2 = (X + 6822500915132347755) * (X + 44167870161383757330) * (X + 55832129838616242863) * (X + 93177499084867652438)
--> next prime...
...reducible polynomial mod 1000000000000000000207: X^4-2 = (X + 38354482782634802697) * (X + 61645517217365197510) * (X^2 + 3402957058619378380) --> next prime...
10000000000000000000301 is OK
```

jupyter

sagejNB-01 Last Checkpoint: 38 minutes ago

File

Edit

View

Run

Kernel

Settings

Help

Not Trusted

JupyterLab

SageMath 10.3

2) Text cell before 2nd code cell

In MD cells, the math mode can be used in a reduced Latex-like way. Here are 2 examples, what can be rendered nicely WITHIN a markdown cell:

Einstein told us that $E = mc^2$. The dot product of two n -vectors is $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i b_i$.

The following code cell then plots a Matplotlib graphics and outputs it.

```
[2]: import matplotlib.pyplot as plt
# %matplotlib inline is no longer necessary to make Matplotlib plots appear inline
from IPython.core.interactiveshell import InteractiveShell
# InteractiveShell.ast_node_interactivity = "last_expr" # default
InteractiveShell.ast_node_interactivity = "none" # no evaluation of the last line
plt.plot([0,2,1.4,9]) # In an ipynb, this writes the graph into the next output cell.
plt.xlabel('$x$')
plt.ylabel('Just for testing $(e)^{\sin x}$')
```

```
[2]: Text(0, 0.5, 'Just for testing $(e)^{\sin x}$')
```

Abb. 9: Typische Abfolge von Zellen in einem Jupyter-Notebook für Sage-Code mit grafischem Output im Browser und in VSC

sagejNB-01.ipynb - Visual Studio Code

File

Edit

Selection

View

Go

Run

Terminal

Help

sagejNB-01.ipynb

x

home > be > Downloads > ctb-git > ctb > code > sage > sagejNB-01.ipynb > ...

+ Code + Markdown | ▶ Run All | Clear All Outputs | Outline ...

2) Text cell before 2nd code cell

In MD cells, the math mode can be used in a reduced Latex-like way. Here are 2 examples, what can be rendered nicely WITHIN a markdown cell:

Einstein told us that $E = mc^2$. The dot product of two n -vectors is $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i b_i$.

The following code cell then plots a Matplotlib graphics and outputs it.

```
[2]: import matplotlib.pyplot as plt
# %matplotlib inline is no longer necessary to make Matplotlib plots appear inline
from IPython.core.interactiveshell import InteractiveShell
# InteractiveShell.ast_node_interactivity = "last_expr" # default
InteractiveShell.ast_node_interactivity = "none" # no evaluation of the last line
plt.plot([0,2,1.4,9]) # In an ipynb, this writes the graph into the next output cell.
plt.xlabel('$x$')
plt.ylabel('Just for testing $(e)^{\sin x}$')
```

```
[2]: Text(0, 0.5, 'Just for testing $(e)^{\sin x}$')
```

1.5 Der Kernel eines Jupyter-Notebooks

Das Jupyter-Notebook, egal ob es im Browser oder in VS Code läuft, zeigt nur die Ergebnisse des sogenannten Kernels an, also hier des SageMath-Servers. Der Kernel kann lokal oder auch remote auf einem Server ausgeführt werden. Genau genommen ist zwischen Jupyter-Notebook und Kernel noch ein Jupyter-Server, der entweder lokal oder remote laufen kann. In beiden Fällen muss SageMath dem Notebook-Server als „Kernel“ zugänglich sein.¹⁷ Dies wird in Abb. 10 dargestellt.

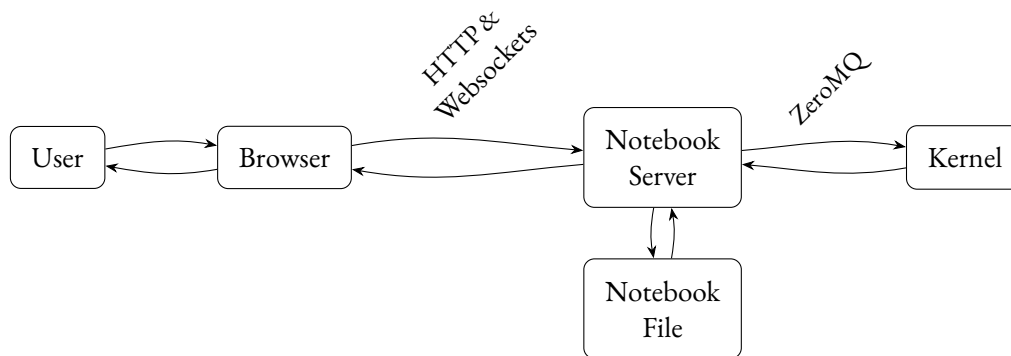


Abb. 10: Umgebung eines Jupyter-Notebook¹⁸

Wie in Tabelle 1 aufgeführt, kann man SageMath auch remote nutzen, ohne SageMath lokal zu installieren:

<https://sagecell.sagemath.org/> oder <https://cocalc.com/>

Zwei übliche Methoden, den Jupyter-Server lokal zu betreiben, sind:

- Mit auf Ihrem Computer installiertem Anaconda aktivieren Sie zuerst die SageMath-Umgebung und starten dann im gleichen Terminal den Jupyter-Server, der dann das angegebene Notebook `Untitled.ipynb` im Browser startet:

```
$ conda activate sage
$ jupyter notebook Untitled.ipynb19
```

- Ohne Anaconda, aber mit installiertem lokalem SageMath, kann man unter Linux das Notebook normalerweise so vom Terminal aus starten:

```
$ sage -n jupyter
oder
$ sage -n jupyterlab
```

Führt man `$ sage -n jupyter` auf der Kommandozeile aus, öffnet sich ein Browserfenster, das alle Dateien des aktuellen Verzeichnisses zeigt. Per Doppelklick auf eine `ipynb`-Datei erscheint diese dann in einem weiteren Browser-Tab als Jupyter-Notebook.

Manchmal kommt bei ersten Klick auf den „Run“-Button die Meldung „Error displaying widget: model not found“. Den Button nochmal drücken löst das Problem bei einer korrekten Installation meist.

¹⁷ Es kann vorkommen, dass das Notebook eine Funktion wie `interact` nicht kennt: Dann bitte erst prüfen, dass es eine Code-Zelle ist und dass SageMath als Kernel gesetzt ist, und dann auf die Ikone „restart the kernel“ klicken.

¹⁸ Leicht verändert übernommen von <https://mlsummit.ai/blog/jupyter-notebooks-fuer-lehre-und-entwicklung-alles-im-blick-notizbuch-fuer-entwickler/>

¹⁹ Im Browser steht dann als Link etwas wie `http://127.0.0.1:8888/notebooks/Untitled1.ipynb?kernel_name=sagemath`

1.6 Programmieren mit SageMath in der Sage-Konsole oder per Start eines Sage-Skripts im Terminal

Wenn man anfängt, ein CAS (Computer-Algebra-System) zu bedienen, wird man fertige Befehle aufrufen – so wie für die Kommandozeile im SageMath-Beispiel 1.1 auf Seite 8.²⁰

Wenn man eigene Funktionen entwickelt, sie ändert und erneut aufruft, dann ist es viel einfacher, die Entwicklung der Funktionen in einem Editor vorzunehmen (einfacher im Vergleich zur puren Nutzung der Sage-Konsole). Hierzu sind drei Vorgehensweisen üblich:

- Nutzen des Jupyter-Notebooks. Dies wird ausführlich beschrieben in 1.7.2 und in 1.8.
- Ausführen des editierten Skripts innerhalb der Sage-Konsole
- Ausführen des editierten Skripts als Sage-Programm durch das Terminal (z. B. in der Bash-Shell)

Die letzten beiden Fälle werden in diesem Unterkapitel näher erläutert.

Alle drei Arten, Code zu entwickeln, wurden in den CrypTool-Buch-Kapiteln 1.12 (Anhang: Didaktische Beispiele für symmetrische Chiffren mit SageMath), 2.8 (Anhang: Beispiele mit SageMath), 4.15 (Anhang: Beispiele mit SageMath), 5.20 (Anhang: Beispiele mit SageMath) und 9.4 (Anhang: Boolesche Abbildungen in SageMath) angewandt.

Um den in Programmdateien gespeicherten SageMath-Code zu testen und zu laden, gibt es zwei nützliche Befehle: `load()` und `attach()`.²¹

Angenommen Sie haben die folgende Funktions-Definition, die in der Datei `primroots.sage` gespeichert wurde:

SageMath-Beispiel 1.7: Funktionsdefinition in einer Datei mit Endung `.sage`

```
def somefunction(var1): # inside file with name e.g. primroots.sage
    """
    DocText.
    """
    ...
    return (L)
```

Load Der Befehl `load()` kann in allen drei in Tabelle 1 auf Seite 6 aufgeführten Fällen benutzt werden: auf der Sage-Kommandozeile, in einem im Terminal aufgerufenen Sage-Skript und in einem Jupyter-Notebook.

Im SageMath-Beispiel 1.8 wird die obige Funktion aus SageMath-Beispiel 1.7 mit dem Befehl `load()` geladen.

SageMath-Beispiel 1.8: `load`

```
sage: load("primroots.sage")
```

²⁰Code für die Sage-Konsole wird so präsentiert, dass die Zeilen mit „sage:“ und „...“ beginnen:

```
sage: m = 11
sage: for a in range(1, m):
....:     print( [power_mod(a, i, m) for i in range(1, m)] )
....:
```

Code in SageMath-Skriptdateien oder Jupyter-Zellen wird genau so wiedergegeben, wie er dort vorkommt.

²¹Vergleiche das SageMath-Tutorial über Programmierung, Kapitel *Loading and Attaching Sage files*, <https://doc.sagemath.org/html/en/tutorial/programming.html>

Danach kann man auf der Kommandozeile alle Variablen und Funktionen nutzen, die im SageMath-Skript `primroots.sage` definiert wurden.²²

Der Befehl `load` führt erst mal alle Befehle aus, unabhängig davon, ob er auf der Sage-Kommandozeile oder in einem Sage-Skript aufgerufen wird. Danach stehen die Funktionen aus der mit `load` geladenen Datei weiterhin zur Verfügung. Leider kann man nicht wie in Python nur die Funktionen aus einer SageMath-Datei bekannt machen. Diese Funktionalität wurde mit der eigenen Funktion `my_import` verfügbar gemacht (siehe Punkt (c) in Abschnitt 5.20.4 des CrypTool-Buchs).

Attach Der Befehl `attach()` kann nur auf der Sage-Kommandozeile (direkt oder in einem darin geladenen Sage-Skript) benutzt werden, aber nicht in einem im Terminal aufgerufenen Sage-Skript und in einem Jupyter-Notebook (dort geht nur `load()`).

Normalerweise möchte man ein eigenes SageMath-Skript editieren und den Inhalt des geänderten Skripts wieder in SageMath laden. Um eine Datei automatisch nach jeder Änderung neu zu laden, wird einmalig der Befehl `attach()` benutzt:²³

SageMath-Beispiel 1.9: attach

```
sage: attach("primroots.sage")
```

Nun kann man das SageMath-Skript ändern und die geänderte Funktionsdefinition wird – solange man die SageMath-Session nicht beendet – beim nächsten Enter in SageMath geladen (und syntaktisch gleich geprüft). Der Befehl `attach()` lässt SageMath also permanent die genannte Datei auf Änderungen überwachen.

Abb. 11 auf der nächsten Seite zeigt SageMath-Code im Editor VS Code – mit aktiviertem Syntax-Highlighting für Python. Genaugut können Sie andere Editoren wie GVim oder Notepad++ verwenden.

Falls man die Ausgabe einer attachten Datei so angezeigt haben möchte, wie wenn man die Einzelbefehle direkt auf der Kommandozeile eingibt (also nicht nur das, was per `print()` ausgegeben wird), kann man den Befehl `iload()` verwenden: Jede Zeile wird dann einzeln geladen. Um die nächste Zeile zu laden, muss man die Enter-Taste drücken. Das muss man so lange wiederholen, bis alle Zeilen des SageMath-Skripts in die SageMath-Session geladen sind.

SageMath-Beispiel 1.10: iload

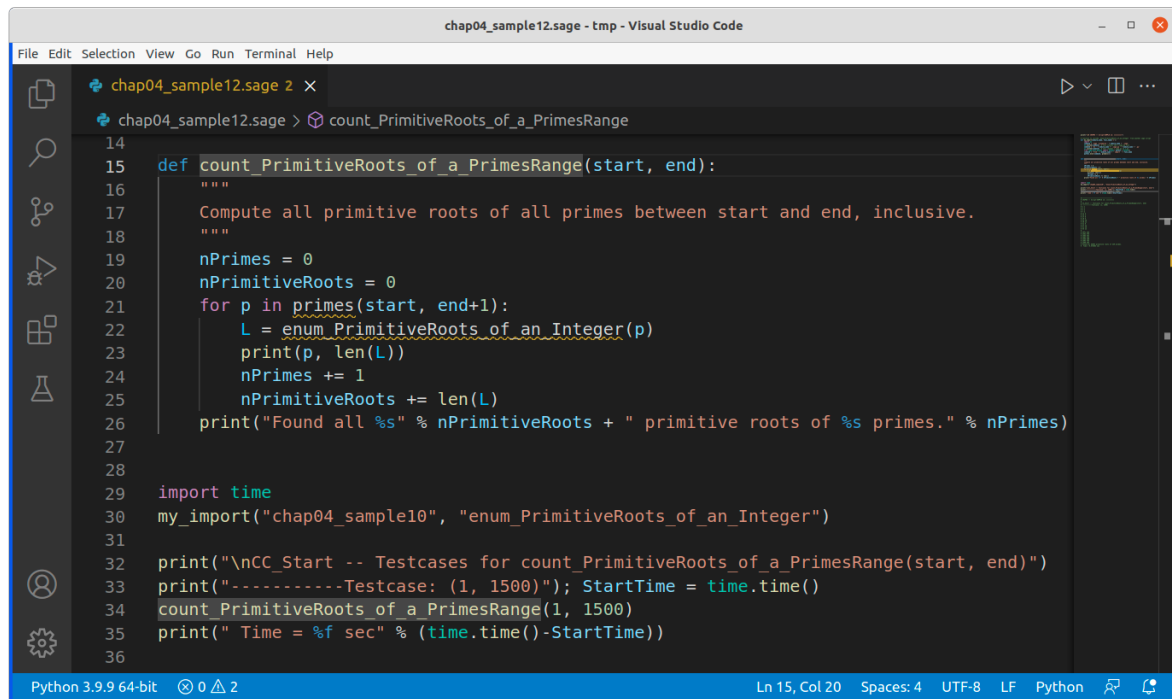
```
sage: iload("primroots.sage")
```

²² Anmerkungen/Hinweise für SageMath-Skripte:

- Bitte keine Leerzeichen oder White Spaces im Dateinamen.
- Es empfiehlt sich, der SageMath-Skriptdatei die Datei-Extension `.sage` statt `.py` zu geben. Hat ein SageMath-Skript die Dateinamens-Endung `.sage`, dann wird beim Laden der Datei in SageMath auch gleich die normale SageMath-Umgebung mit geladen, um die Syntax zu prüfen.
- Statt ein SageMath-Skript vom SageMath-Prompt zu laden, kann man es auch gleich beim Start von SageMath mitaufrufen, bspw. von der Bash-Shell mit `$ sage primroots.sage`. Auch dann wird sofort ein Syntaxcheck durchgeführt und, wenn der erfolgreich ist, das Skript ausgeführt.
- Beim Laden des obigen SageMath-Skripts wird es von SageMath zuerst geparkt, und dann in eine andere Datei mit der Endung „py“ kopiert. SageMath ergänzt dann alle notwendigen Variablen in `primroots.py` und alle Import-Statements. Somit wird das SageMath-Skript genauso ausgeführt, als hätte man die Befehle einzeln auf der Kommandozeile eingetippt.
- Ein bedeutender Unterschied zwischen Befehlen in einer Skript-Datei und der händischen Eingabe nach dem SageMath-Prompt ist, dass alle Ausgaben ein `print()` benötigen. Beispielsweise muss man statt „`m=11; m`“ also „`m=11; print(m)`“ schreiben.

²³ `attach()` kann man auch direkt nach dem `load()` aufrufen, und nicht erst, wenn man das Skript ändert; man kann `load()` sogar weglassen und gleich `attach()` aufrufen, da `load()` in `attach()` enthalten ist.

²⁴ Den in diesem Editor gezeigten Quellcode finden Sie in dem SageMath-Beispiel 5.20.11 des CrypTool-Buchs. und in der Datei „chap05_sample120.sage“.

Abb. 11: SageMath-Beispiel in einem Editor mit aktiviertem Code-Highlighting²⁴

1.7 SageMath und L^AT_EX

„Sage und der TeX-Dialekt L^AT_EX haben eine intensive synergetische Beziehung.“²⁵ Die für uns wesentlichen Beziehungen werden wir anhand von Tabelle 2 erläutern.

Mit SageMath	Sage-Konsole Sage-Programm	latex()
	Jupyter-Notebook	Markdown-Zelle: $\dots$ Code-Zelle: <code>latex(<<Latex code>>)</code> <code>Latex(r"<<Latex code>>")</code> <code>view</code> oder <code>show(<<Latex code>>)</code>
Innerhalb eines L ^A T _E X-Dokuments	Das SageMath-Ergebnis von latex() nutzen	z. B. $\frac{1}{5}$, z^5
	Im Mathematik-Modus SageTeX-Makros nutzen	Aufruf-Reihenfolge: pdflatex → sage → pdflatex (siehe Abb. 18)

Tab. 2: Überblick über das Zusammenwirken SageMath und L^AT_EX

Anmerkung: L^AT_EX trennt die Aufgaben, Mathematik oder normalen Text zu setzen. Dies wird durch die Verwendung von zwei Betriebsmodi erreicht, dem Text- und dem Mathematik-Modus. Der Textmodus ist der Standardmodus für die Dokumentumgebung und muss nicht explizit aufgerufen werden. In L^AT_EX muss sich die gesamte Mathematik in einer Konstruktion des Mathematik-Modus befinden, und viele Symbole, darunter `\circ` oder `\sum_{n=1}^{\infty}`, sind so definiert, dass sie nur im Mathematik-Modus funktionieren. Die gebräuchlichste Art, den Mathematik-Modus aufzurufen, ist $\dots$, wobei sich der Text innerhalb der

²⁵Zitat aus dem Kapitel „Sage, L^AT_EX und ihre Freunde“ zu SageMath 9.4, <https://doc.sagemath.org/html/de/tutorial/latex.html>, einer Übersetzung aus dem englischen Artikel von Rob Beezer (2010-05-23).

Dollarzeichen im Mathematik-Modus befindet. Zwei weitere Möglichkeiten zum Aufrufen des Mathematik-Modus sind `\begin{equation} ... \end{equation}` und `\[ ... \]` (wird auch Doppel-Dollar- oder Display-Stil genannt und für abgesetzte Formeln benutzt). Der Mathematik-Modus ignoriert Leerzeichen. Wenn wir Text in die Mathematik einfügen wollen, müssen wir LaTeX mitteilen, dass wir Text schreiben, da es sonst annimmt, dass das Wort in Wirklichkeit eine Folge von Symbolen ist, die kursiv dargestellt werden. Es gibt zwei Möglichkeiten, einen solchen Text korrekt zu formatieren, zum einen mit dem Befehl `\mbox` und zum anderen mit dem Befehl `\text{}`, der das Paket `amsmath` erfordert.

Um den LaTeX-Code eines Sage-Ausdrucks innerhalb von SageMath-Code zu erhalten, wird normalerweise die Funktion `latex()` verwendet. Diese Funktion erwartet, dass die Eingabe im Mathematik-Modus erfolgt, wie innerhalb von `$ ... $`. Bitte beachten Sie, dass dann die normalen LaTeX-Textmakros wie `\texttt{}` nicht funktionieren.

1.7.1 LaTeX und SageMath auf der Konsole

Von innerhalb SageMath ist der einfachste Weg, SageMath's LaTeX-Unterstützung zu nutzen, die `latex()`-Funktion. Die erzeugten Zeichenketten können dann direkt in LaTeX-Dokumenten genutzt werden. Das funktioniert sowohl in einer Code-Zelle im Notebook als auch mit der SageMath-Kommandozeile. Mit `latex()` können SageMath-Objekte auf zwei Arten in der üblichen mathematischen Form ausgegeben werden: Wir können die Objekte an den Befehl `latex()` übergeben; alternativ liefern manche SageMath-Objekte die SageMath-Beschreibung selbst, indem man ihre eigene `latex()`-Methode aufruft. Die erzeugte Textform (LaTeX-Code) kann dann über eine $\text{\LaTeX}$ -Anzeigesoftware in die übliche mathematische Form umgewandelt werden.

Mit anderen Worten: Man übergibt einen SageMath-Ausdruck an die Funktion `latex()`, um einen LaTeX-Ausdruck zu bekommen. So wandelt einem SageMath die LaTeX-relevanten Bestandteile direkt in LaTeX-Syntax um und man kann diese 1:1 in ein LaTeX-Dokument einbetten.

Beispiel 1: Aufruf des Kommandos `latex(y)` (manchmal auch `y.latex()`):²⁶

```
sage: z=var('z'); latex(z^12)
z^{12}
sage: latex(integrate(z^4, z))
\frac{1}{5} \, z^5
sage: latex('This is a string')
\text{\texttt{This} }is{ }a{ }string}
sage: latex(QQ)
\Bold{Q}
```

Beispiel 2: Jedes Objekt hat eine $\text{\LaTeX}$ -Repräsentation. Übergibt man `$y = (a*x^2)/5$` an `latex()`, so erhält man eine Fehlermeldung, denn eine Zuweisung (dem `y` wird der Term rechts vom Gleichheitszeichen zugewiesen) ist kein Objekt in SageMath. Die rechte Seite alleine ist ein Objekt und dann kann man `latex()` darauf ausführen:

```
sage: var('a,x,y')
(a, x, y)

sage: latex(y=(a*x^2)/5)
TypeError                                Traceback (most recent call last)
<ipython-input-4-23375ccb8aef> in <module>
----> 1 latex(y=(a*x**Integer(2))/Integer(5))
```

²⁶Ganz fehlerfrei ist das Kommando `latex()` nicht. Zum Beispiel ist `\Bold` nicht in normalem LaTeX verfügbar, sondern wird über `\newcommand{\Bold}` in `sage.misc.latex_macros` definiert. Verwenden Sie also `\Bold` nur interaktiv oder verwenden Sie es in einer TeX-Datei nur nach Einfügen der Definition von `\newcommand`. Einfacher wäre `$\mathbb{Q}$` für $\mathbb{Q}$, wobei `\mathbb` das Paket `amsmath` erfordert.

```
TypeError: LatexCall.__call__() got an unexpected keyword argument 'y'
```

```
sage: latex((a*x^2)/5)
\frac{1}{5} \, a \, x^{2}
```

```
sage: y=(a*x^2)/5
```

```
sage: latex(y)
\frac{1}{5} \, a \, x^{2}
```

Beispiel 3: Der letzte Befehl in dem Teil zur linearen Algebra von SageMath-Beispiel 1.2 auf Seite 10 enthält auch `latex()`-Aufrufe, um **automatisch** den Code für eine Matrixgleichung zu generieren.

Siehe die dort auskommentierte Zeile:

```
# print('$A*c=b: %s * %s = %s' % (latex(A), latex(c), latex(A*c)))
```

Das ist das Ergebnis:

$$A * c = b : \quad \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} * \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ 0 \end{pmatrix}$$

1.7.2 LaTeX und SageMath im Jupyter-Notebook

Die Abb. 8 und 9 auf Seite 19 und auf Seite 20 zeigen die ersten beiden Teile des Notebooks `sageJNB-01.ipynb`. In jedem dieser zwei Teile ist eine Markdown- und eine Code-Zelle (incl. ihrer Ausgabe) enthalten. Diese beiden Code-Zellen mit Sage-Code enthalten noch keine „LaTeX-Kommandos“.

Der dritte Teil dieses Notebooks enthält in der Code-Zelle mehrere Varianten verschiedener „LaTeX-Kommandos“ – siehe Abb. 13 auf Seite 28.

Die mit `latex()` gerenderten Sage-Ausdrücke werden per `print()`, `show()` und `view()` ausgegeben. Die Unterschiede sieht man im Zellen-Output. `show()` kann man auch ohne vorheriges `latex()` aufrufen und erhält in der Zellen-Output eine Ausgabe wie in einem LaTeX-Dokument.

Die in Abb. 12 gezeigten drei Einzelfenster werden mit dem `view`-Kommando erzeugt.

$$\frac{1}{5} ax^2 + 91$$
$$\mathbf{F}_{5^2}$$
$$\text{PDF: } (a * x^3)/9$$

Abb. 12: Plotausgabe von Code in Abb. 13 in drei extra Fenstern



Jupyter sageNB-01 Last Checkpoint: 42 minutes ago

File Edit View Run Kernel Settings Help

Not Trusted

JupyterLab SageMath 10.3

3) Text cell before 3rd code cell

The following cell uses LaTeX-like expressions for the output and shows what syntax works in a Python code cell.

```
[3]: from IPython.display import display, Math
# display(Math()) writes result of math mode to output cell (like a string between $...$).
display(Math(r'\m \times m = x \m^2 \setminus \ Formula: A = \pi \times r^2',
format(13.49, round(2.2576,2))))

r, theta = var('r', 'theta'); f = r * sin(theta)
expr1=latex(f+1); expr2=latexExpr(f+2)
# display(Math(latex(f))) [=expr1] looks ok; display(Math(LatexExpr(f+2))) looks wrong.
# for simple math expressions show() is enough [display(Math(expr1)) not needed.]
show("E1-s: ", expr1)

print("E1-p: print latex(): ", expr1); print("E2-p: print LatexExpr(): ", expr2)

# --> display(), print(), and show() just do text output in the next cell,
# the output of show() and display() is rendered.
# --> view() stores rendered term as graph in a PDF and opens this PDF in extra windows.
expr3a=latex('(x^2 + 1)/5'); print("E3a-p: ", expr3a); show("E3a-s: ", expr3a)
x=var('x'); expr3b=latex((x^2+22)/5); print("E3b-p-var: ", expr3b); show("E3b-s-var: ", expr3b)

a,x=var('a,x'); expr4=(a*x^2) / 5; show("E4-s-var-no-latex(): ", expr4+81); view(expr4+91)
print("E4-p-var-no-latex(): ", expr4+71)

expr5=FiniteField(25, 'a'); print("E5-p: ", latex(expr5)); view(latex(expr5))

print(LatexExpr(r'\text{TXT: } (a*x^3) / 7')) # shows LaTeX code as text in output cell
view(LatexExpr(r'\text{PDF: } (a*x^3) / 9 ')) # shows rendered LaTeX code in extra window

via display : 13.4900000000000 m \times 2.26 m = x m^2 - Formula: A = \pi \times r^2

E1-s:r\sin(\theta) + 1
E1-p: print latex(): r \sin\left(\theta\right) + 1
E2-p: print LatexExpr(): r*\sin(\theta) + 2
E3a-p: \text{\texttt{(x\char`^2}{ }+{ }1)/5}}
E3a-s:(x^2 + 1)/5
E3b-p-var: \frac{1}{5}x^2 + \frac{22}{5}
E3b-s-var: \frac{1}{5}x^2 + \frac{22}{5} + 81
E4-s-var-no-latex(): \frac{1}{5}ax^2 + 81
E4-p-var-no-latex(): 1/5*a*x^2 + 71
E5-p: \Bold{F}_5^{\{2\}}
\text{TXT: } (a*x^3) / 7
```

Abb. 13: Typische Abfolge von Zellen in einem Jupyter-Notebook für Sage-Code mit LaTeX-Ausdrücken im Browser und in VSC

Im Groben hat man drei Möglichkeiten, um in einem Jupyter-Notebook mit SageMath-Code LaTeX zu nutzen.

a) `latex()` Der Befehl `latex()` bietet hier dieselben Möglichkeiten wie auf der Sage-Konsole – siehe Unterkapitel 1.7.1 und Abb. 14.

b) `view()` oder `show()` Wir nutzen `show()`, um aus dem Sage-Ausdruck einen LaTeX-Ausdruck zu erzeugen und diesen in einer schön formatierten Ausgabe anzuzeigen.^{27,28} Siehe Abb. 14.

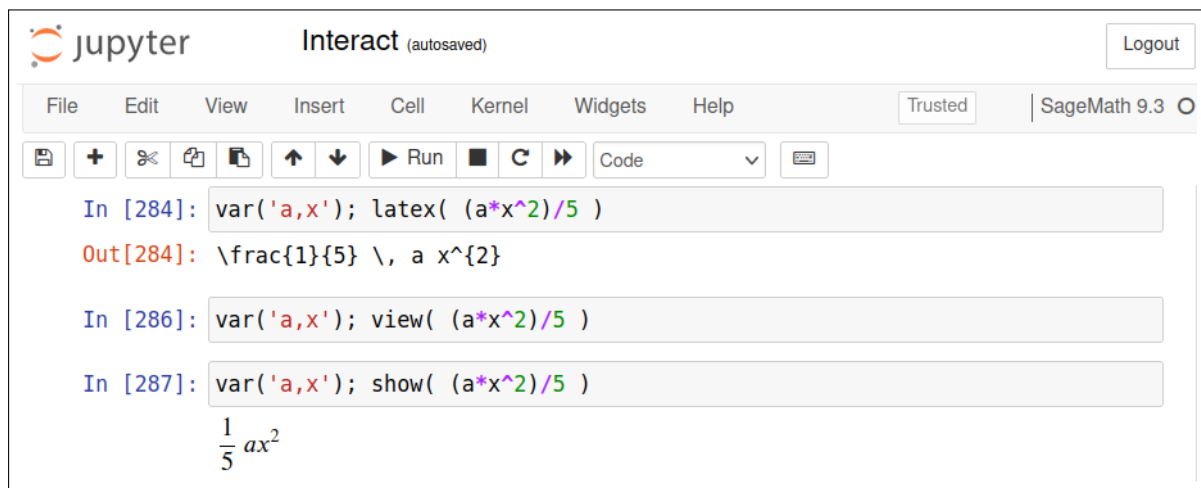


Abb. 14: `latex()` und `show()` in SageMath-Befehle im Jupyter-Notebook

c) `Latex()` Wenn man schon schön aufbereiteten LaTeX-Code aus einem Dokument hat, kann man diesen auch in der Zelle für Sage-Code verwenden. Am stabilsten ist wahrscheinlich die Nutzung der `Latex`-Befehl.²⁹

Per `Latex()` wird der LaTeX-Code zwischen den Begrenzern `"""` gleich vom LaTeX-Display im Mathemode gerendert. Im Unterschied dazu verfährt `latex()` so, dass der LaTeX-Code zwischen den `"""` nur in LaTeX-Code im Mathemode umgewandelt wird.

```
from IPython.display import Latex
Latex(r"""
\begin{align}
c = \sqrt{a^2 + b^2}
\end{align}
""")
```

Built-in magic commands für LaTeX in einem Jupyter-Notebook Diese weitere Variante ist eher unüblich geworden. Hier wird (statt mit `display()` zu arbeiten) das sogenannte „magische“³⁰ Kommando `%latex` benutzt. Dieses Kommando verwandelt die gesamte Zelle in eine LaTeX-Zelle. Siehe Abb. 15 auf der nächsten Seite.

²⁷Weitere, nicht immer ganz aktuelle Details finden Sie unter <https://doc.sagemath.org/html/de/tutorial/latex.html>.

²⁸LaTeX muss installiert sein, um das `show()`-Kommando zu nutzen.

²⁹Weniger häufig genutzt wird die weitere Display-Möglichkeit, die für das Mathe-Objekt einen Raw-String übergeben bekommt.

³⁰Jupyter-Notebooks haben eingebaute magische Kommandos („Magics“), die mit `%` beginnen. So erweitert der Kernel IPython bspw. die Python-Syntax. Dabei wird zwischen zwei verschiedenen Arten von Magics unterschieden:

- Zeilen-Magics, die durch einen einzelnen `%`-Präfix gekennzeichnet sind und auf einer einzelnen Eingabezeile ausgeführt werden
- Zellen-Magics, denen ein doppeltes Symbol `%%` vorangestellt wird. Damit kann man eine einzelne Notebook-Zelle in einem Unterprozess eines anderen Interpreters (wie `bash`, `R` oder `latex`) ausführen (statt im eingestellten Kernel).

Ob und welche Erweiterungen verfügbar sind, hängt vom eingestellten Kernel ab (SageMath bietet weniger Magics als IPython).

```
%%latex
\begin{align}
c = \sqrt{a^2 + b^2}
\end{align}
```

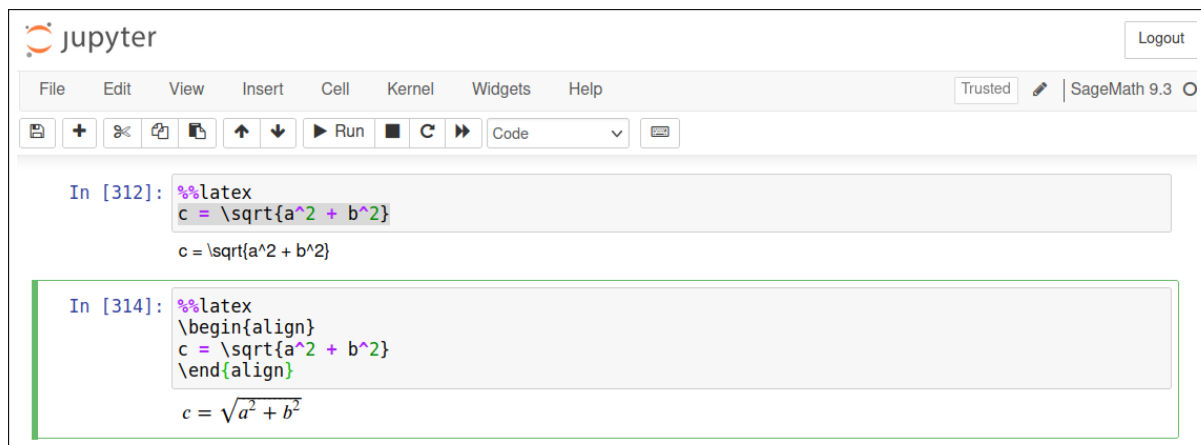


Abb. 15: Verwendung von `%%latex()` im Jupyter-Notebook

Zwei weitere Beispiele:³¹ Hier werden Text und Sage-Code auf unterschiedliche Weise miteinander verbunden. Siehe Abb. 16 auf der nächsten Seite.

1) Verbinden mit `LatexExpr()`

```
var('A, x, y, alpha, beta')
U = A*x^(alpha) * y^(beta)    # typical Sage code
show(U)                       # output Sage code in rendered form
LatexExpr("U(x)=" + latex(U)) # build another Sage code expression
```

2) Erzeugen von html-Code mit eingebetteten Sage-Ausdrücken und eingebetteten LaTeX-Befehlen

SageMath-Beispiel 1.11: Erzeugen von html-Code mit eingebetteten Sage-Ausdrücken und eingebetteten LaTeX-Befehlen

```
# SageJupyter_sample010.sage: SageMath within a Jupyter notebook
var('A, x, y, alpha, beta')
U = A*x^(alpha)*y^(beta)
text = fr"""
<h3>This is a title</h3>
<p>This is some text explaining several interesting
things. <strong>HTML</strong> can be used to
format these lines.</p>
<p>Now we write an inline mathematical expression
$U(x,y)={\rm latex(U)}$, as well as a displayed one:
${\rm latex(\frac{\partial^2 U}{\partial x \partial y})(x,y)}
= {\rm latex(diff(U,x,y))}$</p>
"""
show(html(text))
```

³¹Aus <https://ask.sagemath.org/question/47978/add-a-text-in-latex-in-front-of-a-result/>



Abb. 16: Verwendung von Text und Code in einer Code-Zelle des Jupyter-Notebooks

Die Ausgaben der Jupyter-Zellen werden mit MathJax³² gerendert. MathJax ist eine Browser-übergreifende JavaScript-Bibliothek, die mathematische Notationen in Browsern anzeigt. Die Ausgabe kann in verschiedenen Formaten erzeugt werden, darunter HTML mit CSS-Styling oder skalierbare Vektorgrafiken (SVG). MathJax kann nur eine Teilmenge von TeX und LaTeX abbilden.

Abschnitt 1.8 auf Seite 35 (mit Jupyter und interact) beschreibt einige weitere Anwendungsfälle, in denen automatisierter L^AT_EX-Schriftsatz per $\$ \dots \$$ verwendet wird, z. B. für Labels. Dadurch sehen diese Labels in der Ausgabe im **Jupyter-Notebook** schöner aus.

1.7.3 Im LaTeX-Dokument den von latex() erzeugten Sage-Befehl nutzen

Wenn man in SageMath Sage-Objekte an den Befehl latex() übergibt, erhält man LaTeX-Code. Diesen kann man einfach in ein LaTeX-Dokument kopieren (Einschränkung siehe 1.7.1 auf Seite 25).

Im SageMath-Beispiel 1.12 wird eine Matrix-Gleichung³³ gelöst und anschließend per show() die Gleichung ausgegeben. Am Ende werden die LaTeX-Befehle erzeugt, mit denen man die gelöste Gleichung ausgeben kann (siehe unten). Das zugehörige Jupyter-Notebook ist in Abb. 17 auf der nächsten Seite zu sehen.

SageMath-Beispiel 1.12: Code zur Ausgabe des LaTeX-Befehls einer Gleichung

```
# SageJupyter_sample070.sage: SageMath within a Jupyter notebook
# Solve a matrix equation and generate LaTeX code to show the solved equation e.g. in a pdf.
```

³² <https://www.mathjax.org/>

³³ Die Gleichung stammt aus dem Buch von Craig Finch, S. 118. Der Code wurde stark angepasst.

Fortsetzung SageMath-Beispiel 1.12

```

M4= MatrixSpace(ZZ, 4)
A = M4.matrix([[0, -1, -1, 1], [1, 1, 1, 1], [2, 4, 1, -2], [3, 1, -2, 2]])
b = vector(ZZ, [0, 6, -1, 3])
bT= b.column()

var('x1 x2 x3 x4')
x = vector( [x1, x2, x3, x4] )
xT= x.column()

print('$$ A*x=b:~~~ %s * %s = %s $$'%(latex(A), latex(xT), latex(bT)))

solution = A.solve_right(b)
solutionT= solution.column()

show('s:  $$A*x=b:~~~ %s * %s = %s $$'%(latex(A), latex(solutionT), latex(bT))) # 'show' is an alias for ▶
▶pretty_print
# which should choose the "best" output supported by the user interface; but showed only escaped ▶
▶text on console
print('p:  $$A*x=b:~~~ %s * %s = %s $$'%(latex(A), latex(solutionT), latex(bT))) # 'print' shows text for ▶
▶LateX on console; 'view' had no effect.

import matplotlib.pyplot as plt
plt.plot([0,2,1,4,9]) # In a sage file (e.g. in VSC with sage extension), this has no effect.

```

In [38]:

```

M4 = MatrixSpace(ZZ, 4)
A = M4.matrix([[0, -1, -1, 1], [1, 1, 1, 1], [2, 4, 1, -2], [3, 1, -2, 2]])
b = vector(ZZ, [0, 6, -1, 3])
bT=b.column()

var('x1 x2 x3 x4')
x = vector( [x1, x2, x3, x4] )
xT=xx.column()

show('$$ A*x=b:~~~ %s * %s = %s $$'%(latex(A), latex(xT), latex(bT)))

solution = A.solve_right(b)

show('$$ A*x=b:~~~ %s * %s = %s $$'%(latex(A), latex(solutionT), latex(bT)))
print('$$ A*x=b:~~~ %s * %s = %s $$'%(latex(A), latex(solutionT), latex(bT)))

```

$$A * x = b : \begin{pmatrix} 0 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 \\ 2 & 4 & 1 & -2 \\ 3 & 1 & -2 & 2 \end{pmatrix} * \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 6 \\ -1 \\ 3 \end{pmatrix}$$

$$A * x = b : \begin{pmatrix} 0 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 \\ 2 & 4 & 1 & -2 \\ 3 & 1 & -2 & 2 \end{pmatrix} * \begin{pmatrix} 2 \\ -1 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 6 \\ -1 \\ 3 \end{pmatrix}$$

```

$$ A*x=b:~~~ \left(\begin{array}{rrrr}
0 & -1 & -1 & 1 \\
1 & 1 & 1 & 1 \\
2 & 4 & 1 & -2 \\
3 & 1 & -2 & 2
\end{array}\right) * \left(\begin{array}{r}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{array}\right) = \left(\begin{array}{r}
0 \\
6 \\
-1 \\
3
\end{array}\right)

```

Abb. 17: Löse Matrixgleichung und erzeuge den LaTeX-Code der Gleichung

Hier folgt das Ergebnis aus dem von `latex()` erzeugten LaTeX-Befehlen:

$$A * x = b : \begin{pmatrix} 0 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 \\ 2 & 4 & 1 & -2 \\ 3 & 1 & -2 & 2 \end{pmatrix} * \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 6 \\ -1 \\ 3 \end{pmatrix}$$

$$A * x = b : \begin{pmatrix} 0 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 \\ 2 & 4 & 1 & -2 \\ 3 & 1 & -2 & 2 \end{pmatrix} * \begin{pmatrix} 2 \\ -1 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 6 \\ -1 \\ 3 \end{pmatrix}$$

1.7.4 Im LaTeX-Dokument von SageTeX() bearbeiteten Sage-Code nutzen

Dies ist – im Vergleich zu Unterkapitel 1.7.3 – der spannendere Teil.

Hat man eine TeX-Distribution auf seinem Rechner, kann man direkt in LaTeX-Dokumenten Sage-Code hinein schreiben, diesen ausführen lassen und das Ergebnis automatisch in das erzeugte (normalerweise: PDF) Dokument einbinden.

Dazu muss braucht man das Paket **SageTeX**, welches per `\usepackage{sagetex}` im Header der TeX-Datei eingebunden wird. MikTeX bringt `sagetex` bereits mit; wer TeXLive oder MacTeX hat, kann sich das Paket und die Dokumentation hier <https://ctan.org/pkg/sagetex?lang=de> holen.

Die TeX-Datei, in der man SageTeX verwenden will, sollte im selben Ordner liegen, wie die SageTeX-Dateien, die man als .zip heruntergeladen und entpackt hat. Die SageTeX-Installation bringt das Paket auch mit, hier muss man aber noch etwas Zusatzarbeit leisten, um es verwenden zu können (siehe hier <https://doc.sagemath.org/html/en/reference/misc/sagetex.html>).

Verwendet ein LaTeX-Dokument dieses Paket, vollzieht sich die PDF-Dokumentenerstellung in drei Schritten:

- `pdflatex datei.tex` → es entsteht `datei.sagetex.sage` sowie `datei.aux`, `datei.log`, `datei.pdf`.
- `sage datei.sagetex.sage` → `sage` wird angewendet, um den SageMath-Code, der beim vorherigen Schritt in die Datei `datei.sagetex.sage` geschrieben wurde, auszuführen. Dabei entstehen die drei Dateien `datei.sagetex.[sage.py, scmd, sout]`.
- Erneuter Lauf `pdflatex datei.tex` → fertige pdf-Datei `datei.pdf`

Den Workflow kurz zusammengefasst zeigt Abb. 18.

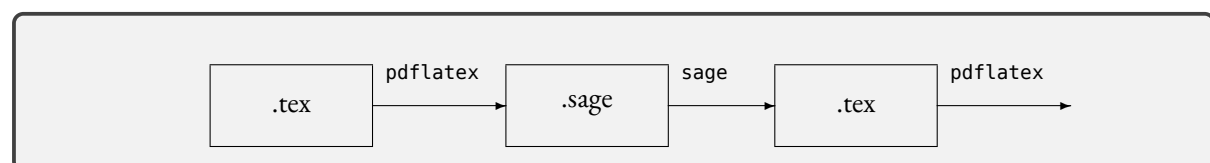


Abb. 18: Workflow mit SageTeX

Anmerkung: Für erfahrene LaTeX-Benutzer und größere LaTeX-Projekte empfehlen wir die Verwendung von **arara**³⁴, um solche Workflows wie in Abb. 18 abzubilden. Vorstellen wollen wir arara anhand eines Minimalbeispiels (minimal working example MWE) – siehe SageMath-Beispiel 1.13. Rufen Sie hierzu im Terminal `arara`

³⁴ arara von Paulo Cereda ist ein TeX-Automatisierungs-Tool, das auf Regeln basiert, die am Anfang der Main-TeX-Datei eines LaTeX-Projekts stehen. Diese Regeln, was arara machen soll, muss man selbst in sein LaTeX-Dokument schreiben. arara bestimmt dann seine Aktionen anhand dieser gegebenen Regeln – anstatt sich die Regeln automatisch aus indirekten Ressourcen wie der Analyse

-w mwe.tex auf, nachdem Sie das Minimalbeispiel als Datei mit dem Namen mwe.tex abgespeichert haben. Das Ergebnis dieses Aufrufs ist, dass die Zahl 32 im PDF gedruckt wird.

Voraussetzung ist, dass „pdflatex“ Zugriff auf die Datei „sagetex.sty“ aus der installierten SageMath-Version hat. Falls das nicht gegeben ist, finden Sie eine Anleitung unter <https://doc.sagemath.org/html/en/tutorial/sagetex.html#sec-sagetex-install>.

SageMath-Beispiel 1.13: Arara und sagetex

```
\documentclass{scrartcl}
% arara: pdflatex
% arara: sage: { files: [mwe.sagetex.sage]}
% arara: pdflatex
\usepackage{sagetex}
\begin{document}
$$\sage{2^5}$$
\end{document}
```

Mit SageTeX können alle Rechen- und LaTeX-Formatierungsfunktionen von SageMath automatisch gehandhabt werden. Das SageTeX-Paket ist eine Sammlung von TeX-Makros³⁵, die es einem LaTeX-Dokument ermöglichen, Anweisungen einzuschließen, um SageMath verschiedene Objekte berechnen und/oder formatieren zu lassen.

Im folgenden Beispiel wird der LaTeX-Code für eine Matrix aus Sage-Code erzeugt – die LaTeX-Datei enthält nur die folgenden vier Zeilen:³⁶

```
\begin{sagesilent}
var('x,y')
M = matrix([[i*x+j*y for i in range(3)] for j in range(3)])
\end{sagesilent}
```

Die erzeugte Matrix sieht dann so aus:

$$M := \begin{pmatrix} 0 & x & 2x \\ y & x+y & 2x+y \\ 2y & x+2y & 2x+2y \end{pmatrix}$$

SageMath-Befehle kann man auch in interact-Beispielen einbinden (vergleiche Anhang 1.8.3 auf Seite 37 und Anhang 1.8.4 auf Seite 38).

von Protokolldateien herzuleiten. arara benötigt eine Java-Runtime. Siehe <https://gitlab.com/islandoftex/arara> und <https://islandoftex.gitlab.io/arara/>. Seit 2024 ist arara nicht mehr Teil der MacTeX-Auslieferung, kann dort aber nachinstalliert werden.

³⁵Weitere Details zu SageTeX-Makros finden Sie unter: <https://ctan.org/pkg/sagetex>, <https://doc.sagemath.org/html/de/tutorial/latex.html> und <https://doc.sagemath.org/html/de/tutorial/sagetex.html>.

Neben **SageTeX** gibt es noch andere Python-Pakete, mit denen man Dinge in einem LaTeX-Dokument automatisieren kann. Das bekannteste ist **PythonTeX** (<https://ctan.org/pkg/pythontex>).

In dem knapp 60-seitigen englischen Dokument <https://cryptool.org/download/ctb/PythonTex-by-Examples.pdf> sind viele Beispiele und Testfälle enthalten. PythonTeX wird oft zusammen mit SymPy benutzt, einem LaTeX-Paket, mit dem man die Ergebnisse eines symbolischen Mathe-Paketes in ein LaTeX-Dokument einbetten kann.

Alternativ-Pakete wie pyLaTeX von Jelte Fennema, v1.3.4, 2020 und hybrid-latex von Leo Brewin, v0.1, 2018 werden nicht mehr gepflegt.

³⁶aus <https://phubert.github.io/sagetex-tutorial.pdf>, Seite 8

1.8 SageMath mit Jupyter und interact

Mit `interact` werden SageMath-Programme interaktiv – der Einfluss von Parametern auf die Berechnungen kann dynamisch visualisiert werden. Dies geschieht auf sehr allgemeine Weise unter Verwendung von Python-Funktionalität. „Daher könnte fast jede mögliche Abhängigkeit gezeigt werden.“³⁷

Man kann also nicht nur das Ergebnis als Text, Grafik oder animiertes GIF zeigen: Mit `interact` kann der Benutzer die vorgegebenen Parameter manipulieren und die veränderten Ergebnisse sofort sehen.

1.8.1 Ein typisches `interact()`-Beispiel

Abb. 19 zeigt ein leicht verändertes Beispiel aus den SageMath-Tutorials³⁸: In den zwei Dropdown-Controls kann man die Längen der beiden Rechteck-Seiten n und m ändern. Dann wird das geänderte Rechteck angezeigt, seine Fläche dynamisch berechnet und die Faktoren der Flächenmaßzahl (Flächengröße) werden in Textform ausgegeben.

SageMath-Beispiel 1.14: Startbeispiel zur Nutzung von `interact` im Jupyter-Notebook

```
# SageJupyter_sample020.sage: SageMath within a Jupyter notebook
@interact
def f( n=(1..10), m=selector([1..15],default=6) ):
    print("Area: n * m = {} = {}".format(n * m, factor(n * m)))
    P = polygon([(0, 0), (0, n), (m, n), (m, 0)])
    P.show(aspect_ratio=1, gridlines='minor', figsize=[3, 3], xmax=15, ymax=10)
```

```
In [230]: #1' https://more-sagemath-tutorials.readthedocs.io/en/latest/tutorial-start-here.html
@interact
def f( n=(1..10), m=selector([1..15],default=6) ):
    print("Area: n * m = {} = {}".format(n * m, factor(n * m)))
    P = polygon([(0, 0), (0, n), (m, n), (m, 0)])
    P.show(aspect_ratio=1, gridlines='minor', figsize=[3, 3], xmax=15, ymax=10)
```

n 1 ▼
m 6 ▼

Area: $n * m = 6 = 2 * 3$

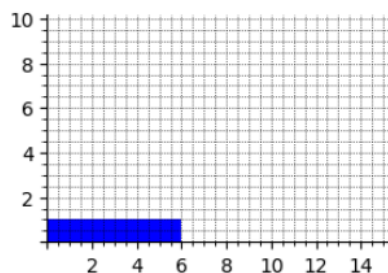


Abb. 19: Ein Startbeispiel zur Nutzung von `interact` im Jupyter-Notebook

Für die Interaktion mit dem Benutzer gibt es derzeit acht Steuerelemente (Sage-Widget-Controls):

- Rechtecke (Boxes)
- Slider
- Bereichs-Slider
- Kästchen (Checkboxes)
- Selektoren (Drop-down-Listen oder Buttons)

³⁷<https://www.sagemath.org/tour-graphics.html>

³⁸<https://more-sagemath-tutorials.readthedocs.io/en/latest/tutorial-start-here.html>

- Grid mit Boxen
- Farb-Selektoren
- Einfacher Text

In Abb. 19 auf der vorherigen Seite werden zwei Drop-down-Selektoren verwendet. Der erste Selektor für n wird implizit durch das Tupel $(1..10)$ erzeugt; der zweite für m wird explizit per `selector()` erzeugt, um per Option einen Initialwert (6) zu setzen (macht man das nicht, wird der erste Wert in der Liste als Default verwendet).

Viele weitere Details finden sich auf den folgenden Seiten und bei den Links in der Fußnote.³⁹

1.8.2 Technisches – was sind Decorators

Decorators sind ein Konstrukt in Python 3, mit dem man eine existierende Funktion (oder Methode) um eine bestimmte Funktionalität erweitern kann. Ein Decorator nimmt eine Funktion auf, fügt Funktionalität hinzu und gibt sie zurück. Decorators fungieren also als Wrapper.⁴⁰

Die beiden folgenden Kurzbeispiele sind – trotz unterschiedlicher Schreibweisen – identisch. Aufgrund der besseren Lesbarkeit wird jedoch nur die `@`-Schreibweise genutzt („syntactic sugar“), um die Decorator-Funktion zu implementieren.

```
def myplot(f=x^2):
    show(plot(f, (x, -3, 3)))
myplot=interact(myplot)

@interact
def myplot(f=x^2):
    show(plot(f, (x, -3, 3)))
```

Diese Kurz-Beispiele stammen aus der SageMath-Dokumentation.⁴¹ Der Graph einer Funktion f wird gezeichnet und der Benutzer kann die Funktion f ändern. Wir definierten eine neue Funktion `myplot` und machten f zu einer Variablen. Damit ein „Steuerelement“ es dem Benutzer ermöglicht, die Funktion f interaktiv einzugeben, stellen wir der Funktion `myplot` einfach `@interact` voran. Beachten Sie, dass wir immer noch die `myplot`-Funktion aufrufen können, selbst wenn wir `@interact` verwendet haben. Dies ist oft beim Debuggen nützlich: `myplot(x^4)`.

Es ist eine Konvention, den Underscore „_“ für Wegwerfnamen zu verwenden, die uns nicht interessieren, wie bspw. `myplot`. Siehe das nächste Beispiel und Abb. 20 auf der nächsten Seite. In der Abbildung wird die Option `aspect_ratio` verwendet: Werte kleiner als 1 stauchen den Graphen vertikal, was hilft, den benötigten vertikalen Platz zu reduzieren. Dann wird einmal `_(x^2)` und einmal `_(x^3)` geplottet.

```
@interact
def _(f=x^2):
    show(plot(f, (x, -3, 3)))
_(x^3)
```

`@interact` bewirkt „wie ein Zauber“, dass in der nächsten Zelle für jedes Argument der unbenannten Funktion ein Control (Steuerelement) verfügbar ist, mit dem der Benutzer das zugehörige Argument ändern kann. Das Ergebnis (Berechnung oder Plot) wird angepasst, ohne dass erneut auf „Ausführen“ geklickt werden muss.

³⁹<https://more-sagemath-tutorials.readthedocs.io/en/latest/tutorial-start-here.html>,
<https://doc.sagemath.org/html/en/prep/Quickstarts/Interact.html>,
<https://wiki.sagemath.org/interact/>,
<https://wiki.sagemath.org/interact/graphics>

⁴⁰Die Parameter der Decorator-Funktion sind dieselben wie die Parameter der Funktion, die sie dekoriert. Es können also beliebig viele Parameter verwendet werden. In Python wird diese „Magie/Zauber“ über `function(*args, **kwargs)` ausgeführt, wobei `args` ein Tupel von Positionsargumenten ist und `kwargs` ist ein Dictionary von Schlüsselwort-Argumenten. Siehe <https://docs.python.org/3/tutorial/controlflow.html#defining-functions>.

⁴¹<https://doc.sagemath.org/html/en/prep/Quickstarts/Interact.html>

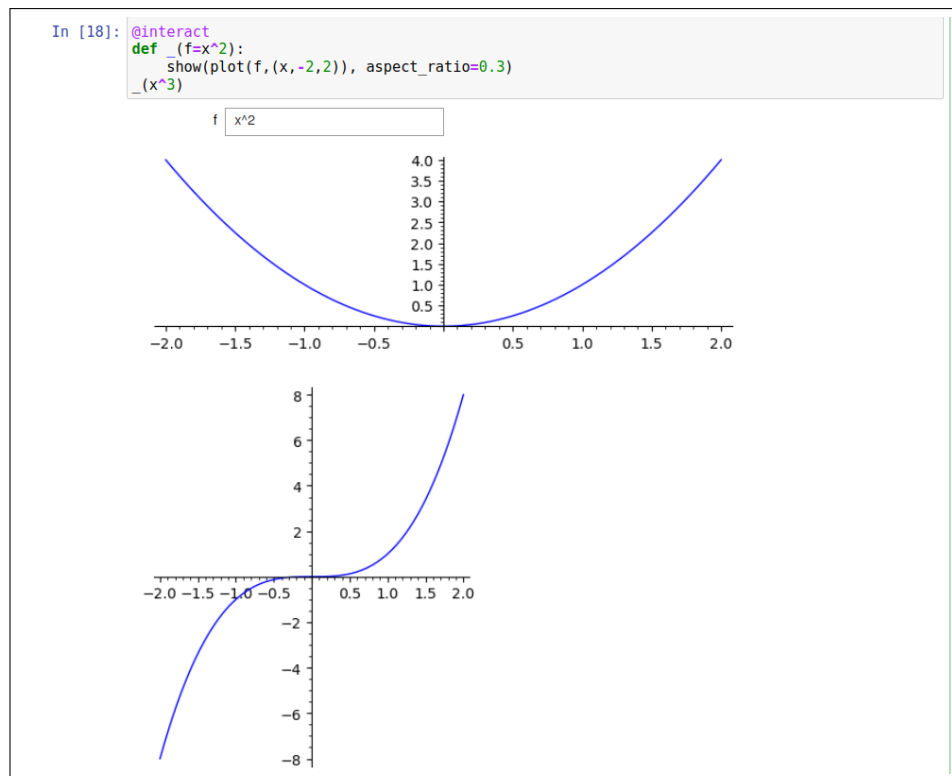


Abb. 20: Aufruf einer unbenannten Funktion per interact automatisch und anschließend nochmal direkt

Mit Hilfe der `interact`-Methode können Basis-Widgets (sind in SageMath schon enthaltene grafische Steuerelemente) automatisch angezeigt und mit den Parametern von benutzerdefinierten Funktionen verknüpft werden. Widgets aus Matplotlib oder aus dem Jupyter-Notebook sind noch flexibler – ihr Einsatz ist in SageMath-Beispiel 1.17 auf Seite 41 zu sehen.

Widgets können entweder direkt oder über die `interact`-Funktion erstellt werden. Da es bequem ist für eine schnelle Verwendung, wird meistens die `interact`-Methode verwendet. `Interact` nimmt eine Funktion als erstes Argument, gefolgt von den Argumenten dieser Funktion mit ihren möglichen Werten. Dadurch werden Widgets erstellt, mit denen diese Werte ausgewählt werden können, wobei für jede Auswahl ein Callback mit dem aktuellen Wert durchgeführt wird.

1.8.3 Interact-Beispiele ohne Grafik

`@interact` wird meist benutzt, um interaktive Grafiken anzuzeigen, aber man kann es auch nutzen, um parametrisierte Textausgaben anzuzeigen.

Das folgende Beispiel⁴² faktorisiert (multipliziert aus) einen symbolischen Ausdruck, der vom Exponenten n abhängt.

```
@interact
def _(n=(2,100,1), auto_update=False):
    show(factor(x^n - 1))
```

Bewegt man den Slider (Schieberegler) in Abb. 21 auf der nächsten Seite, ändert das den Exponenten. Hier werden die Neuberechnung und das Neuzeichnen nur durchgeführt, nachdem der Benutzer den Button „Run Interact“ gedrückt hat (dieses Verhalten entsteht durch Setzen der Option `auto_update` auf `False`). Das Deaktivieren dieser Voreinstellung ist sinnvoll für Funktionen, deren Auswertung eine Weile dauert, oder um ein Flackern zu verhindern, da normalerweise die Neuberechnung an jedem Punkt des Schiebereglers gestartet wird, über den Sie den Slider bewegen.

⁴²aus <http://fe.math.kobe-u.ac.jp/icms2010-dvd/SAGE/www.sagemath.org/doc/reference/sagenb/notebook/interact.html>

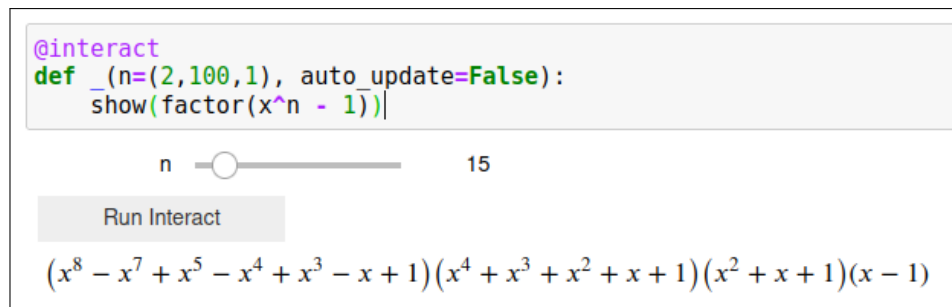


Abb. 21: Aufruf einer Funktion ohne Graph per interact (mit auto_update)

Das Beispiel in Abb. 22⁴³ erstellt ein interaktives Steuerelemente-Set mit drei Eingaben: zwei Slider-Eingaben für a und y (die den Bereich von Ganzzahlen von -5 bis 15 und von 0 bis 20 durchlaufen), und eine Texteingabe für den Ausdruck c . Beachten Sie, dass die `print()`-Ausgabe der Variablen c ihren symbolischen Ausdruck $y+2$ zeigt, während $c(y)$ ihren Wert ausgibt.

```
var('y')
@interact
def _(a=5, y=(0..20), c=y+2):
    print("a+y:", a + y); print("c: ", c, "=", c(y))
```

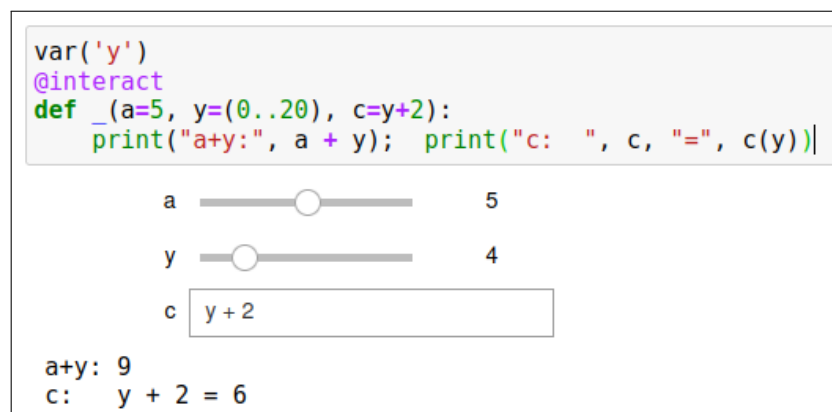


Abb. 22: Funktion mit 2 Eingaben und Evaluierung eines symbolischen Ausdrucks per interact

1.8.4 Interact-Beispiele mit Grafik

In den folgenden Beispielen werden Grafiken mit `interact` manipuliert.⁴⁴

Das Beispiel in Abb. 23 auf der nächsten Seite und in SageMath-Beispiel 1.15 auf der nächsten Seite ist angepasst und erweitert – nach dem sehr guten Buch [Fin11]. Die Kurve $1/((x - a)^e) = (x - a)^{-e}$ hat zwei `interact`-Parameter a, e . Der Graph zeigt die Pole bei $x = a$ für ungerade Werte von e und einen flexiblen Bereich von y -Werten, abhängig vom Wert von e . Die Bezeichnungen der Achsen werden manuell in LaTeX in der Code-Zelle gesetzt.

⁴³modifiziert aus <http://fe.math.kobe-u.ac.jp/icms2010-dvd/SAGE/www.sagemath.org/doc/reference/sagenb/notebook/interact.html>

⁴⁴Viele weitere Beispiele finden sich bei:

<https://wiki.sagemath.org/interact/misc>
<https://wiki.sagemath.org/interact/games>
https://wiki.sagemath.org/interact/graph_theory
<https://www.sagemath.org/tour-graphics.html>
<https://wiki.sagemath.org/art>

SageMath-Beispiel 1.15: Graph mit Asymptote und Parametern (siehe Abb. 23)

```
# SageJupyter_sample030.sage: SageMath within a Jupyter notebook
@interact
def _(a=slider([-5..5], default=2, label='Param a: '),
     e=slider([-5..5], default=1, label='Param e: ')):
    xs=0.15 # space in x direction shown left an right of the pole
    pole_plot = plot( 1 / ((x - a)^e), (x, a-(xs), a+(xs)), detect_poles='show' )
    pole_plot.ymax((100.0)^e)
    pole_plot.ymin(-(100.0)^e)
    print("min y = {0:.3f} // max y = {1:.3f}".format( pole_plot.ymin(), pole_plot.ymax() ))
    print("min x = {0:.3f} // max x = {1:.3f}".format( pole_plot.xmin(), pole_plot.xmax() ))
    pole_plot.axes_labels([r'$x$', r'$1/(x-a)^e$'])
    pole_plot.show()
```

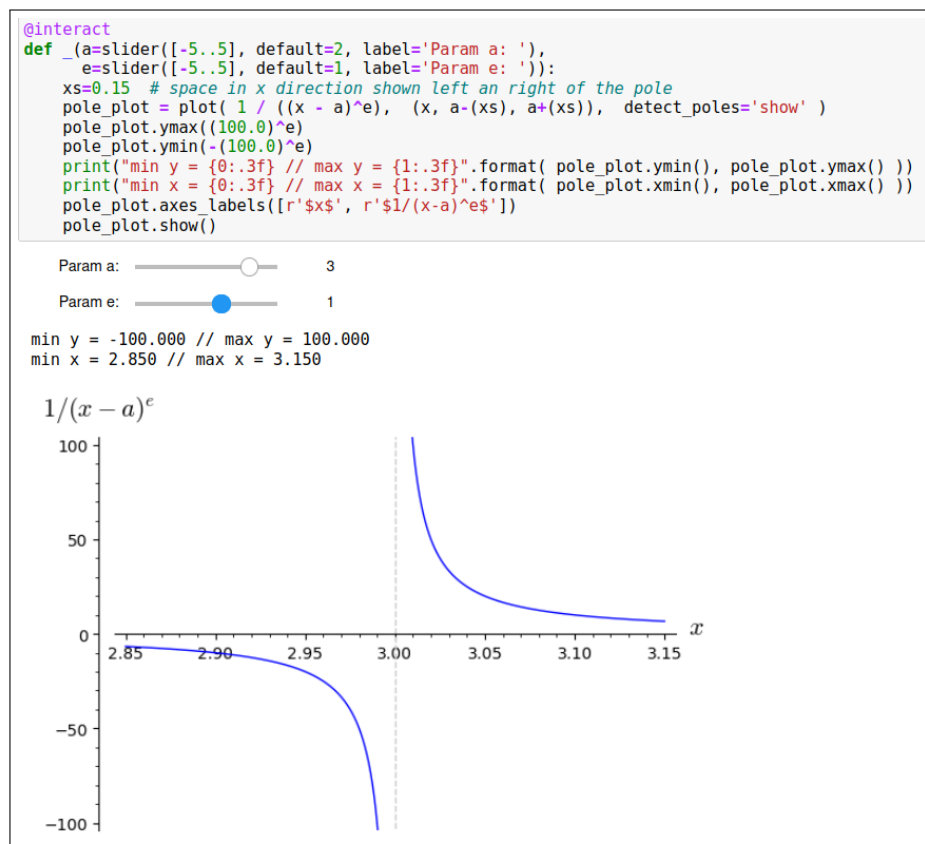


Abb. 23: Graph mit Asymptote und Parametern (vgl. Abb. 24)

Noch etwas flexibler als SageMath-Beispiel 1.15 ist das SageMath-Beispiel 1.16 auf der nächsten Seite (ohne Abbildung): Es hat eine ähnliche Funktionalität, aber die Funktion $g()$ wird **vor** der unbenannten Funktion definiert und daher kann der Funktionsname g in der unbenannten Funktion verwendet werden anstatt mehrfach seinen Ausdruck hinzuschreiben. Innerhalb von `plot()` funktioniert das problemlos – und auch innerhalb von `axes_labels` wird die kursive LaTeX-Darstellung für die ausgewertete g -Funktion angezeigt: Der Trick (gefunden in ask.sagemath.org) hier ist, dass man den String für die Beschriftung der y-Achse verkettet erzeugt: `'$'+latex(g(a,e,x))+'$'`.⁴⁵

⁴⁵Einige Parameter im SageMath-Plot sind noch etwas unflexibel. Die direkte Verwendung der zugrunde liegenden Mathplotlib kann für mehr Robustheit sorgen.

SageMath-Beispiel 1.16: Graph mit Asymptote und Parametern und generischer Funktion (ohne Abb.)

```
# SageJupyter_sample040.sage: SageMath within a Jupyter notebook
var('a, x')
g(a,x) = a*x^3
@interact
def _(a=slider([-5..5], default=2, label='Param a: ')):
    p = plot(g(a,x),(x,-3,3), color='purple') # make the plot line purple
    # p.axes_labels([ '$x$', '$a*x^3$' ])
    # p.axes_labels([ '$x$', eval('g()') ])
    p.axes_labels([ '$x$', '$'+latex(g())+'$' ])
    show(p)
    print("g: %s, g()=%s, g(a)=%s, g(a,x)=%s, g(a,1)=%d, g(a,2)=%d" % (g, g(), g(a), g(a,x), g(a,1),
    # g(a,2) ))
#
# g: (a, x) |--> a*x^3, g()=a*x^3, g(a)=2*x^3, g(a,x)=2*x^3, g(a,1)=2, g(a,2)=16
```

1.9 SageMath mit Jupyter und Matplotlib interactive_output

Die Grafiken (Plots) in SageMath werden erstellt mit dem Python-Paket Matplotlib.⁴⁶ Matplotlib ist die verbreitetste Grafikbibliothek für Python generell.

Die am häufigsten verwendeten Funktionen von matplotlib sind über Sage-Funktionen direkt zugänglich. Bisher beschränkten wir uns in den Beispielen auf die in SageMath eingebauten Grafikfunktionen (siehe die Sage-Widgets in 1.8.1 auf Seite 35).

Alternativ kann man entweder das ganze Mathplotlib-Paket in SageMath importieren (und dann seine komplette Funktionalität nutzen) oder die Grafikfunktionen (Widgets oder Controls) aus dem Jupyter-Notebook nutzen, die ebenfalls auf Mathplotlib beruhen. Im folgenden Beispiel benutzen wir die Jupyter-Widgets und erzielen damit ein schöneres Layout als mit den Sage-Widgets.

Das bisherige Beispiel in Abb. 23 und in SageMath-Beispiel 1.15 auf der vorherigen Seite zeigte die Controls einfach untereinander an. Das ist praktisch für ein schnelles Ergebnis, aber nicht für eine ansprechende GUI. Nun bauen wir dieses Beispiel m. H. von Jupyter-Widgets (ipywidgets)⁴⁷ um.

Als Eingabe fungieren die beiden ipywidgets.IntSlider; als Ausgabe (statt print) wird ipywidgets.Label genutzt. Alle vier Widgets werden erstmal unabhängig erzeugt. Dann werden die Widgets in ein Grid eingefügt, das aus $2 \cdot 3 = 6$ Zellen besteht. Die Slider nehmen jeweils die ersten beiden Spalten ein, die Labels nehmen die letzte Spalte; Slider haben also die doppelte Breite wie die Labels. Labels gibt es also nicht nur bei Koordinatenachsen (so wie in den bisherigen Beispielen), sondern frei auch platzierbar. Label-Widgets zeigen den Value-Text an, sind aber nicht editierbar, was hier gewollt ist.

Nicht nur `g()`, sondern auch `myplot()` ist extra definiert, so dass die interaktive Funktion relativ klein ist: Sie ruft `myplot()` und ordnet die Slider-Eingabecontrols `aa` und `ee` den Eingabevariablen `a` und `e` der Funktion `myplot()` zu.

Mit Hilfe von `interactive_output` wird die Grafik nicht sofort ausgegeben, sondern in einem Output-Widget erzeugt. Dadurch könnte die Grafik später per `display` auch mehrfach ausgegeben werden.

Insgesamt erlauben die ipywidgets ein viel flexibleres Layout als die Basic-Widgets, die in SageMath integriert sind. Dafür ist der Code länger 😊.

⁴⁶<https://matplotlib.org/>

⁴⁷Siehe <https://ipywidgets.readthedocs.io/en/latest/examples/Using%20Interact.html>. Zwei sehr schöne Einführungen in Jupyter-Widgets findet sich in <https://www.elab2go.de/demo-pyl/jupyter-notebook-widgets.php> und <https://kapernikov.com/ipywidgets-with-matplotlib/> (2020).

SageMath-Beispiel 1.17: Code für Graph mit Controls von ipywidgets (siehe Abb. 24)

```
# SageJupyter_sample060.sage: SageMath within a Jupyter notebook
# Widgets from ipywidgets and called via interactive_output

from ipywidgets import GridspecLayout, Layout, IntSlider, Label, interactive_output

var('a, e, x')
g(a,e,x) = 1 / ((x - a)^e)

# Create the controls
aa = IntSlider(min=-5, max=5, value=2, description='Param a: ', layout=Layout(width='auto', height='auto'))
ee = IntSlider(min=-5, max=5, value=1, description='Param e: ', layout=Layout(width='auto', height='auto'))

label1 = Label(description="for_x", layout=Layout(display='flex', justify_content="center", border='1px solid green'))
label2 = Label(description="for_y", layout=Layout(display='flex', justify_content="center", border='1px solid green'))

# Distribute the controls into a grid for better layout
grid = GridspecLayout(int(2), int(3))
grid[0,0:2] = aa
grid[1,0:2] = ee
grid[0,-1] = label1
grid[1,-1] = label2

def myplot(a, e):
    xs=0.15 # space in x direction shown to the left and right of the pole
    pole_plot = plot( g(a,e,x), (x, a-(xs), a+(xs)), detect_poles='show' )
    pole_plot.ymax((100.0)^e)
    pole_plot.ymin(-(100.0)^e)
    # Add spaces via the unicode character \xa0 ("nonbreaking space")
    skip = ':\xa0\xa0\xa0\xa0'
    s1 = 'x' + skip + "min = {0:.2f} \xa0 // \xa0 max = {1:.2f}".format( pole_plot.xmin(), pole_plot.xmax() )
    s2 = 'y' + skip + \
        ( "min = {0:.0f} \xa0 // \xa0 max = {1:.0f}".format(pole_plot.ymin(), pole_plot.ymax()) if (
            pole_plot.ymin()>100 or pole_plot.ymin()<-100
        ) else "min = {0:.2f} \xa0 // \xa0 max = {1:.2f}".format(pole_plot.ymin(), pole_plot.ymax()) )
    label1.value = s1
    label2.value = s2
    pole_plot.axes_labels([ '$x$', '$'+latex(g())+'$' ])
    pole_plot.show()

out = interactive_output( myplot, {'a': aa, 'e': ee} )
display(grid, out)
```

1.10 Weitere interact SageMath-Beispiele zur Kryptografie

Neben den Beispielen in diesem Buch finden Sie unter dem Link in Fußnote 48 viele einfache Beispiele, die die Befehle aus dem Gebiet der Kryptografie in SageMath mit interact darstellen:⁴⁸

1. Verschiebe-Chiffre (Ver- und Entschlüsselung)
2. Affine Chiffre (Ver- und Entschlüsselung)
3. Substitutions-Chiffre
4. Playfair-Chiffre (Ver- und Entschlüsselung)
5. Häufigkeitsanalyse
 - a) Häufigkeitsanalyse

⁴⁸ <https://wiki.sagemath.org/interact/cryptography> (erstellt zuerst auf den „Sage Days 103“, August 2019)

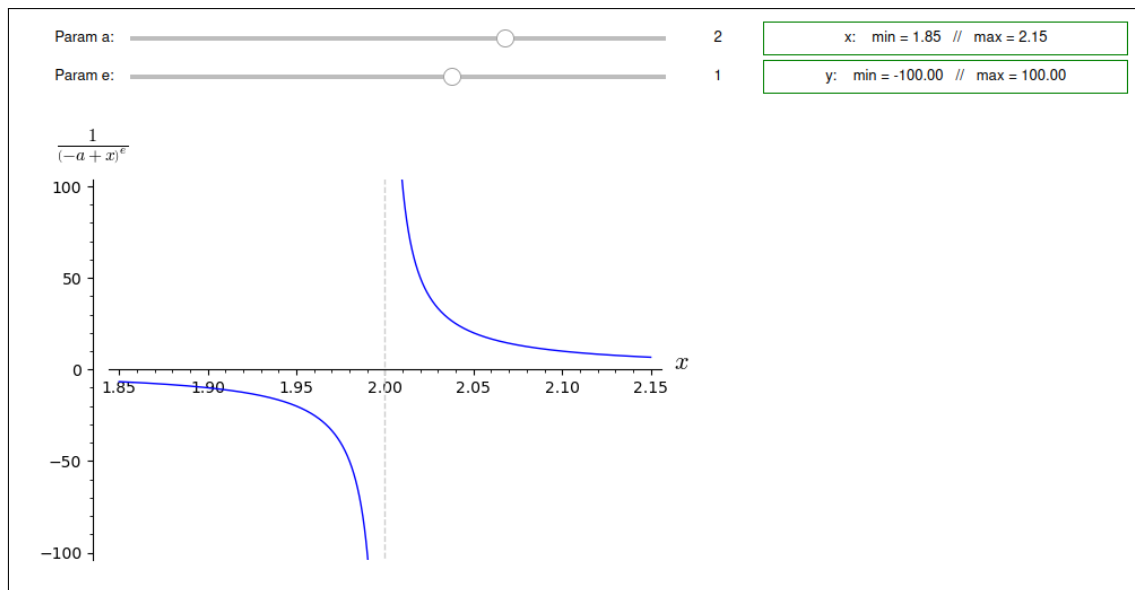


Abb. 24: Graph mit Controls von ipywidgets (vgl. Abb. 23)

- b) Raten mit Hilfe der Frequenzanalyse
- 6. Vigenère-Chiffre (Ver- und Entschlüsselung)
- 7. One-Time-Pad (OTP)
- 8. Hill-Chiffre (Ver- und Entschlüsselung)
- 9. Modulare Arithmetik (Voraussetzungen für RSA, Diffie-Hellman, ElGamal)
 - a) Modulare Arithmetik Multiplikationstabelle
 - b) Modulare Exponentiation
 - c) Diskretes Logarithmusproblem (Lösen nach x und nach b)
- 10. RSA
 - a) RSA, aus der Perspektive von Alice
 - b) RSA, aus der Perspektive von Babette
 - c) Digitale Signaturen mit RSA

1.11 Der Verlauf der MTW-Punkteverteilungs-Kurven mit SageMath

Im Krypto-Rätsel-Wettbewerb MysteryTwister⁴⁹ (MTW, früher MTC3) gibt es drei Level, in denen Punkte für eine gelöste Aufgabe automatisch anhand einer Formel berechnet werden. Durch das Level ist eine Basis-Punktzahl vorgegeben, die man mindestens erreicht und der sich die erhaltene Punktzahl immer mehr annähert, je länger die Aufgabe schon publiziert ist. Wie schnell diese Annäherung erfolgt hängt vom Level ab: Je niedriger das Level, desto schneller (siehe <https://mysterytwister.org/challenges/points-calculation/>).

Der zugehörige Graph wird in Abb. 25 auf der nächsten Seite für die ersten 500 Tage verdeutlicht. Erzeugt wird er mit dem SageMath-Beispiel 1.18 auf der nächsten Seite.

In blau sind die 3 Kurven der 3 Levels zu sehen – normalisiert auf einen Basiswert von 100. Mit dem Slider kann man das Gewicht c in kleinen Schritten auswählen, um in einer roten Kurve zu sehen, wie schnell sich die „Fallgeschwindigkeit“ gegen den Basiswert anpasst. Die rote Kurve bewegt sich immer über den anderen Kurven; in Abb. 25 verdeckt sie die blaue Kurve von Level 1, denn c hat in Level 1 den Wert 1,00.

Der Basiswert ist 50 % des Maximalwerts (hier in der Grafik immer 200; in MTW sind es 200, 2000 und 20000). Jedes der drei Level L1, L2 und L3 startet mit seinem Maximalwert. Nach 10 Tagen sind (L1/L2/L3) jeweils

⁴⁹ <https://mysterytwister.org/>

bei (55/74/86) % des Maximalwerts. Level 1 (L1) erreicht nach 101 Tagen schon 50 %. Nach 500 Tagen sind (L2/L3) jeweils bei (57/70) % des Maximalwerts.

SageMath-Beispiel 1.18: Code zum Graph der MTW-Punkte im Zeitverlauf

```
# SageJupyter_sample050.sage: SageMath within a Jupyter notebook
# MTW formular: points for one challenge after d days

c,d = var('c, d')

def f(c, d):
    r = 1 - ( 1 / (( 2^(1-c)) * ((d+1)^c)) )
    return ( r )

def g(c,d):
    100 / f(c,d)

@interact
def _( c = slider( 0, 2, step_size=0.05, default=1 ) ):
    g1 = 100 / f(1, d)
    p1 = plot(g1, (1, 500), thickness=1, color='blue' )

    g025 = 100 / f(0.25, d)
    p025 = plot(g025, (1, 500), thickness=1, color='blue' )

    g01 = 100 / f(0.1, d)
    p01 = plot(g01, (1, 500), thickness=1, color='blue' )

    g = 100 / f(c, d) # 100 = 10*10^i, where i=1 (so all curves use level 1 base value)
    # the x-axis variable is d and runs from day 1 to 500
    p = plot(g, (1, 500), thickness=2, color='red' )

    (p1+p025+p01+p).show(title='MTW curve courses for level L1, L2, and L3')
```

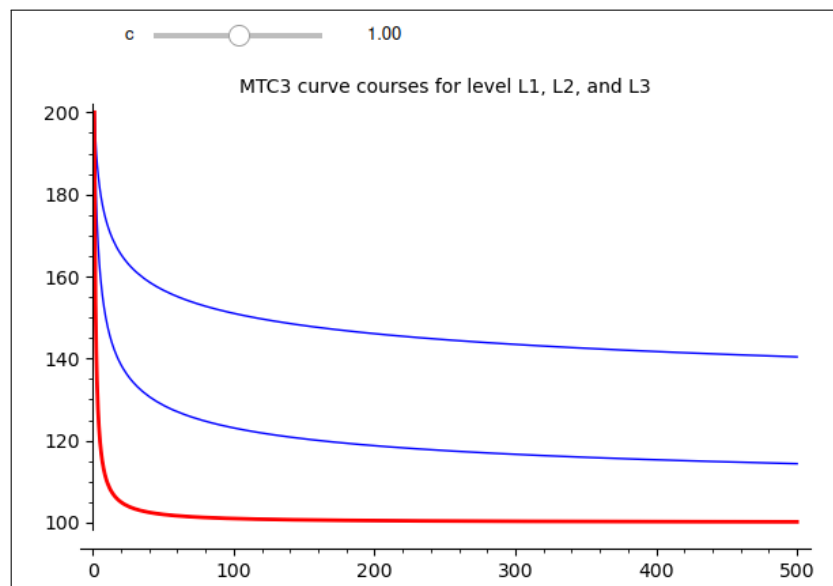


Abb. 25: Graph der MTW-Punkte im Zeitverlauf

1.12 Professionellere Sage-Programme

In der Literatur und auch hier findet man zu SageMath oft kurze Code-Beispiele. Man kann mit Sage aber auch ganz professionelle Programme schreiben. Das liegt vor allem daran, dass man in Sage alle Möglichkeiten von

Python 3 nutzen kann.

In dem SageMath-Beispiel 2.8.19 des CrypTool-Buchs kann man das ansatzweise erkennen. Das Analyse-Programm

- hat den typischen Aufbau eines Python-Programms mit einer übersichtlichen main-Funktion am Ende.
- verteilt alle Tasks in handhabbare Funktionen.
- parst die Kommandozeile mit `argparse`.
- enthält Testdaten (hier in einem `dictionary`).
- unterstützt einen Verbose-Modus, mit dem man sich innere Abläufe und Log-Informationen ad-hoc ausgeben kann.
- enthält Kommentare, usage-Infos und Versionierung.

1.13 Weitere Hinweise zu SageMath in diesem Buch

Weitere Hinweise:

- Abfrage der Version Ihrer SageMath-Umgebung mit: `version()`
- Um schnell die SageMath-Programmbeispiele in diesem Buch zu finden, können Sie
 - im Index nach SageMath -> Programmbeispiele schauen, oder
 - sich diese in Kapitel 2 auf Seite 46 ansehen.
- Die SageMath-Beispiele aus diesem Buch finden sich zum Download auf der CrypTool-Webseite. Weitere Details dazu am Ende der Übersicht in Kapitel 2 auf Seite 46.

Alle Links dieses Anhangs wurden am 5.6.2024 überprüft.

Dank

Für das gründliche und konstruktive Korrekturlesen dieses Kapitels: Doris Behrendt.

2 Verzeichnisse der Abbildungen, Tabellen, Code-Beispiele, etc.

2.1 Abbildungsverzeichnis

1	SageMath-Konsole von der Kommandozeile (Terminal)	7
2	Erstellen einer SageMath-Datei mit Jupyter im Browser	7
3	Webseite zur Standard-Dokumentation von SageMath (https://doc.sagemath.org/html/en/)	13
4	Output nach Punkt bei Tab-Vervollständigung	15
5	Index	16
6	Suche innerhalb der Befehle, die mit dem Buchstaben <i>a</i> beginnen.	16
7	Typische generelle Abfolge von Zellen in einem Jupyter-Notebook	18
8	Typische Abfolge von Zellen in einem Jupyter-Notebook für Sage-Code mit Text-Output im Browser und in VSC	19
9	Typische Abfolge von Zellen in einem Jupyter-Notebook für Sage-Code mit grafischem Output im Browser und in VSC	20
10	Umgebung eines Jupyter-Notebook ¹	21
11	SageMath-Beispiel in einem Editor mit aktiviertem Code-Highlighting	24
12	Plotausgabe von Code in Abb. 13 in drei extra Fenstern	27
13	Typische Abfolge von Zellen in einem Jupyter-Notebook für Sage-Code mit LaTeX-Ausdrücken im Browser und in VSC	28
14	latex() und show() in SageMath-Befehle im Jupyter-Notebook	29
15	Verwendung von %latex() im Jupyter-Notebook	30
16	Verwendung von Text und Code in einer Code-Zelle des Jupyter-Notebooks	31
17	Löse Matrixgleichung und erzeuge den LaTeX-Code der Gleichung	32
18	Workflow mit SageTeX	33
19	Ein Startbeispiel zur Nutzung von interact im Jupyter-Notebook	35
20	Aufruf einer unbenannten Funktion per interact automatisch und anschließend nochmal direkt	37
21	Aufruf einer Funktion ohne Graph per interact (mit auto_update)	38
22	Funktion mit 2 Eingaben und Evaluierung eines symbolischen Ausdrucks per interact	38
23	Graph mit Asymptote und Parametern (vgl. Abb. 24)	39
24	Graph mit Controls von ipywidgets (vgl. Abb. 23)	42
25	Graph der MTW-Punkte im Zeitverlauf	43

2.2 Tabellenverzeichnis

1	Überblick über die SageMath-Aufrufmöglichkeiten (Benutzerschnittstellen)	6
2	Überblick über das Zusammenwirken SageMath und LaTeX	24

¹Leicht verändert übernommen von <https://mlsummit.ai/blog/jupyter-notebooks-fuer-lehre-und-entwicklung-alles-im-blick-notizbuch-fuer-entwickler/>

2.3 Verzeichnis der SageMath-Programmbeispiele

1.1	Kleine Beispiele aus verschiedenen Gebieten der Mathematik (1)	8
1.2	Kleine Beispiele aus verschiedenen Gebieten der Mathematik (2)	10
1.3	SageMath-Hilfe (Welcome)	12
1.4	SageMath-Hilfe (zu Einzelfunktion)	14
1.5	SageMath-Hilfe (Ausdehnen mit Tab-Taste)	15
1.6	Programm, um alle Methoden eines SageMath-Objekts auszugeben	15
1.7	Funktionsdefinition in einer Datei mit Endung <code>.sage</code>	22
1.8	<code>load</code>	22
1.9	<code>attach</code>	23
1.10	<code>iload</code>	23
1.11	Erzeugen von html-Code mit eingebetteten Sage-Ausdrücken und eingebetteten LaTeX-Befehlen	30
1.12	Code zur Ausgabe des LaTeX-Befehls einer Gleichung	31
1.13	<code>Arara</code> und <code>sagetex</code>	34
1.14	Startbeispiel zur Nutzung von <code>interact</code> im Jupyter-Notebook	35
1.15	Graph mit Asymptote und Parametern (siehe Abb. 23)	39
1.16	Graph mit Asymptote und Parametern und generischer Funktion (ohne Abb.)	40
1.17	Code für Graph mit Controls von <code>ipywidgets</code> (siehe Abb. 24)	41
1.18	Code zum Graph der MTW-Punkte im Zeitverlauf	43

Alle SageMath-Beispiele aus diesem Buch finden sich auf der CrypTool-Webseite:

<https://www.cryptool.org/de/ctbook/sagemath>

Die Namen der SageMath-Skript-Dateien enthalten zuerst die Kapitel-Nummer und dann die laufende Nummer des Skripts innerhalb dieses Kapitel, z. B. `chap08_sample11.sage`.

Alle Beispiele wurden mit den SageMath-Versionen 9.3 (Release Date 2021-05-09) unter Windows, 9.5 (Release Date 2022-01-30) unter Linux und 10.0 (Release Date 2023-05-23) unter macOS getestet.

3 Gesamtliteraturverzeichnis

[Fin11] Craig Finch. *Sage Beginner's Guide*. PACKT Publishing, 2011 (siehe S. [38](#)).

4 Index

A

Analyse

 Häufigkeits-, 41

arara, 33

C

Chiffre

 affine, 41

D

Decorator, 36

H

Hill, 42

K

kwargs, 36

L

Logarithmusproblem

 diskret, 42

M

modulare Arithmetik, 42

MTW, 42

 Formel, 42

my_import, 23

O

OTP, 42

P

Playfair, 41

Python, 5

R

RSA, 42

S

SageMath, 5

 Programmbeispiele, 5, 46

SageTeX, 33

Substitution, 41

syntactic sugar, 36

V

Verschiebe-Chiffre, 41

Vigenère, 42