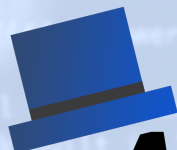


An Invitation to



MadHat







and Mathematical  
Typesetting

Dan Romik


## About the author

Dan Romik is a professor of mathematics at the University of California, Davis. His book *The Surprising Mathematics of Longest Increasing Subsequences* was published by Cambridge University Press in 2015. You can learn more about him at <https://www.math.ucdavis.edu/~romik/>


# Table of Contents



<b>Preface</b> . . . . .	4
<b>Part I: Mathematical typesetting and  MadHat</b> . . . . .	5
<b>Chapter 1: Introduction</b> . . . . .	6
<b>Chapter 2: The nine design principles of  MadHat</b> . . . . .	18
<b>Chapter 3: Future plans</b> . . . . .	31
<b>Chapter 4: Summary and a call for help</b> . . . . .	34
<b>Part II:  MadHat Version 1.1.1 User Manual</b> . . . . .	36
<b>About  MadHat</b> . . . . .	37
<b>About this help guide</b> . . . . .	38
<b>User Manual table of contents</b> . . . . .	39
<b> MadHat version history and changelog</b> . . . . .	210
<b>Index of  MadHat commands and keywords</b> . . . . .	212

# Preface

In late 2019 I embarked on what could be described without excessive hyperbole as a fool’s errand: the development of a new, feature-complete software system for mathematical typesetting. This eventually turned into the  **MadHat** desktop application and content specification language, currently at **Version 1.1.1**.

This book presents the current state of the project, as well as my vision for why it is needed and what it should become in the future. The book is part manifesto, part status report, part user manual, and (I expect) part semi-coherent ramblings that will distract you from more important work.

The book is aimed at people with an interest in mathematical typesetting and where it is headed. If you are a user of LaTeX or another mathematical typesetting system and are curious about software solutions that may make your life easier, my goal is to tell you about  **MadHat**, why I created it, and what work still remains to make it a more complete solution for mathematical writers.

The book is divided into two parts. The **first part** is written as a kind of “manifesto” that gives a detailed accounting of my rationale and thought process in creating  **MadHat**. The **second part** is the  **MadHat** user manual, which gives a full description of the software’s current features and how to use them.

If you have any comments or suggestions on this book or on  **MadHat**, I would love to hear from you!

Dan Romik  
Berkeley, September 2022

[hatter@madhat.design](mailto:hatter@madhat.design)  
<https://madhat.design>



# *Part I*

# Mathematical typesetting and **MadHat**

# Chapter 1: Introduction

## 1.1 The importance of mathematical typesetting

Millions of people around the world regularly use mathematics in their work, and communicate mathematical ideas to others. The visual language and notation of mathematics — equations, formulas, mathematical special symbols, etc — form the basic mechanism for such communications. However, mathematical notation is notoriously complex, being a **two-dimensional writing system**, and standard software tools such as word processors, presentation applications, email software, etc, which facilitate written communication and are used by essentially all members of our technology-driven society, in many cases lack the features that make it practical to include mathematical content in the documents one is composing.

For this reason, starting in the late 1970s, specialized software tools and formats have been developed for the specification and typesetting of mathematical content. I will survey these tools and the advantages they offer in the next section. The use of such tools, while fairly widespread among professional mathematicians and other technically-minded communicators of mathematics, remains rather limited from the perspective of the general population, or even the mathematically-literate portion of it.

I see this as a problem on several levels. First, it is a **first-order economic problem**. By this I mean that the scientific publishing industry, which relies in a critical way on the ability to efficiently publish content containing large amounts of mathematics, is a **\$10 billion a year industry**. While this industry is doing fine using existing tools, those tools are heavily geared towards the “traditional” publishing of

bound volumes of printed paper, or their electronic equivalents. Seeing as how a large amount of content is these days presented using newer, non-traditional modalities of communication — e.g., **videos**, **interactive apps**, and more — it is not a stretch of the imagination to suppose that improvements to the software tools that facilitate the composition of documents with mathematical content could well lead to a dramatic growth in the amount (and quality) of mathematically-oriented content created and presented using those modalities. Naturally, some economic benefits could result from such a process.

Second, it is a **second-order economic problem**. By this I mean, there are economic benefits of a more elusive or indirect nature that could result from an improvement in people’s ability to create and communicate mathematical content efficiently. Whenever people communicate better, they work better together, and can achieve more: more technologies and products developed and brought to market, more new inventions dreamed up, etc. These effects are impossible to quantify, but they are real.

Third, it is a **problem for STEM (Science, Technology, Engineering and Mathematics) education**. Communicating mathematical ideas is at the heart of a STEM education, and this communication takes place between professors and students, students and professors, and students and other students. And while professors are usually proficient enough with the existing software tools to prepare well-formatted electronic documents — lecture notes, homework assignments, etc — containing mathematical content, many students lack those technical skills and are constrained to writing their course notes and assignments by hand. This is an unnecessary barrier to efficient studying and communication.

(Again, I should stress that this supposed “problem” isn’t causing the sky to fall by any means; the current system is working fine, but one can still imagine things being better — particularly when comparing the situation in STEM with other disciplines like the humanities, where handwritten

class assignments are essentially unheard of these days, and where it is very practical for students to take real-time notes on their laptops or tablets while sitting in class.)

Fourth, it is a **problem for STEM research**. Again, as with the discussion of the economics of technical innovations above, the point is that improvements to communication are likely to yield at least indirect dividends in the research output of STEM researchers and scholars. As one example, the online mathematics research community

**MathOverflow** relies on the MathJax typesetting engine for inserting mathematical content into a post. The availability of such a means of communication for research mathematicians has been a real **boon to research**, and one can easily imagine that a further improvement to this system could allow researchers to communicate even more easily with each other, leading to new collaborations and research discoveries that might not have happened otherwise.

Last, and perhaps least importantly (but still somewhat importantly, at least for me personally and I think for many other people), it is a **cultural problem**. The issue is that in today's technology landscape, mathematical content is relegated to the status of a second-class citizen. While non-mathematical textual content is completely standardized and supported by the most widely used software systems (web browsers, word processors, email software, instant messaging applications, etc), anyone wishing to communicate mathematical content typically has to resort to less standard tools, and/or to make assumptions about which tools the recipient of the content has access to, which typesetting languages they are familiar with, etc. As a small example, when I communicate by email with other mathematicians, we often discuss mathematics using LaTeX code embedded in the email, which we “compile” in our heads while reading and writing.

For the many people who use mathematics and discuss it regularly, this is a frustrating state of affairs. Some might argue that this is exactly how


things should be, given that mathematics is “niche” and not used by everyone. Well, let them argue that; I argue the opposite: *everyone* should be able to communicate mathematics easily, using highly polished software that is on par with anything else we have come to expect these days from our apps and operating systems. Think about it: doesn’t mathematical content deserve to get at least as much love and respect from software developers as, say, **emoji**, **animoji**, **memoji** (or any other *moji*)?

Indeed, the language of mathematics is more universal than any particular spoken language, and is appreciated and used by millions of people around the world. Popular YouTube channels such as **Numberphile** and **3Blue1Brown** covering mathematical content have racked up hundreds of millions of views. The **Khan Academy** has produced content used by over 70 million people (despite its videos using handwritten mathematical formulas in a relatively traditional lecture style). Are such forms of mathematical content “niche”? To me, the argument is clear that this type of content deserves to be well-supported by software tools.

(Lastly, I should note that I am far from the only person to consider the importance of this issue: standards-developing bodies such as the **Unicode Consortium** and the **World Wide Web Foundation** have been working for decades towards the goal of promoting the incorporation of mathematical content into digital communication standards, for exactly the same reasons.)

The problems I outlined above motivated me to develop a software system, called **MadHat**, that tries to improve on the current state of the art in mathematical writing systems. In this part of the book I will offer more details on what **MadHat** does, what it is meant to ultimately do, and why it could be an improvement on existing tools.

I’ll start by describing the mathematical typesetting tools that are currently available. After that I’ll discuss what are some of the issues

with these tools and why we need new ones. Finally, I'll explain how  **MadHat** aims to meet the need for new tools.

## 1.2 The current state of mathematical typesetting

The current landscape of tools for mathematical typesetting is heavily dominated by TeX, LaTeX and their descendant systems. Below I've listed the main available tools and technologies, with relevant links. (This list is probably incomplete; if you are aware of any significant items I've missed, please let me know.)

### Typesetting engines

- [TeX \(Wikipedia page\)](#)
- [LaTeX \(Wikipedia page\)](#)
- [ConTeXt \(Wikipedia page\)](#)
- [XeTeX \(Wikipedia page\)](#)
- [LuaTeX \(Wikipedia page\)](#)

### Mathematics on the Web

- [MathJax \(Wikipedia page\)](#)
- [KaTeX \(Wikipedia page\)](#)
- [MathML \(Wikipedia page\)](#)
- [Overleaf \(Wikipedia page\)](#)
- [Mathematics markup on Wikipedia](#)
- [Unicode support for mathematics](#)

### TeX/LaTeX front ends

- [Comparison of TeX editors \(Wikipedia\)](#)
- [LaTeX Editors/IDEs \(TeX - LaTeX Stack Exchange\)](#)

## Mathematics typesetting without TeX

- [Alternatives to LaTeX \(Tex - LaTeX Stack Exchange\)](#)
- [Mathematical Typesetting \(Wolfram Language Reference\)](#)
- [List of formula editors \(Wikipedia\)](#)

## 1.3 Why we need new tools

*“Why is the whole document recompiled every time? Why can’t some of the work for each paragraph (or section or figure) be kept if the context (font, spacing, block width, etc.) in which the paragraph is recompiled does not change? LaTeX should memoize the result of compilation for each section based on the context in which it is compiled. This would eliminate the need for breaking your document into various tex files and recompiling them yourself (doing the job of the compiler) [...]”*

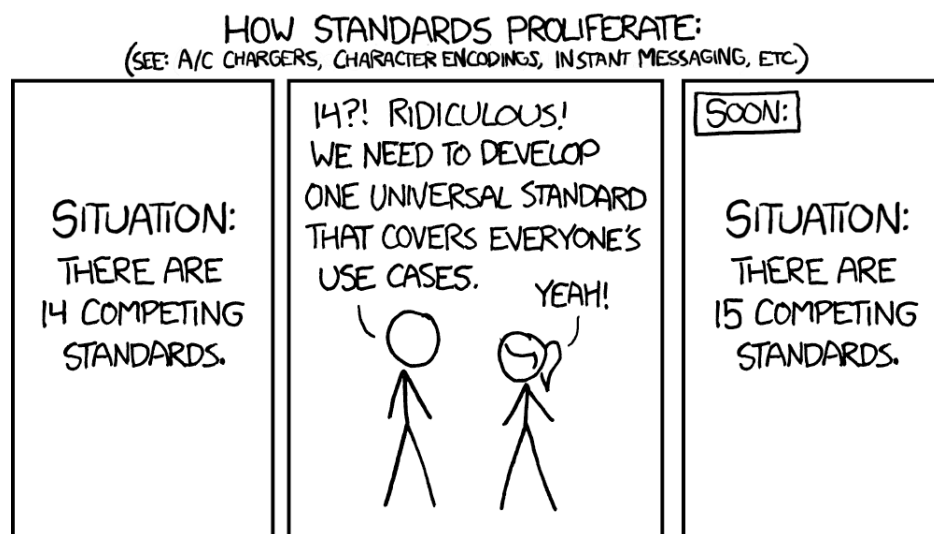
—Neil G, in [“Are there any open research problems in the world of TeX?”](#) (TeX - LaTeX Stack Exchange, 2013)

*“**Automatic breaking of display equations.** Currently the `breqn` package implements the ideas of Michael J Downes, but AFAIK, the algorithmic aspects are not as well understood as that of line-breaking of text. Is it possible to case line-breaking of display equations as an optimization problem and determine a solution based on penalties and badness?”*

—Aditya, in [“Are there any open research problems in the world of TeX?”](#) (TeX - LaTeX Stack Exchange, 2013)

From the list of existing software tools for writing mathematics given above, it might appear as if there is no need for new tools, and indeed, it seems incontrovertible that there is already a healthy ecosystem of tools that meet the needs of most mathematical writers at a basic level; even at a high level, in many ways.

On the other hand, the existing tools can be said to be satisfactory in the same way that horse-drawn carriages were considered a satisfactory means of transportation in the early 1900s by most people. And yet, Henry Ford famously came up with a better solution, not **faster horses**.



(Credit: [xkcd](#))

The simplest answer to the question of why we need new tools is that after writing thousands of pages of mathematical content over more than twenty years, I became frustrated with the existing tools, and felt that they were not meeting my needs, nor those of other people I know. And I started to imagine what a better tool might look like, and how it might be implemented.

A more detailed answer has to do with the technical reasons for *why* the existing tools were not meeting my needs, and why they *could not even conceivably be improved* to meet my needs. The underlying issue is that the existing tools are all based on the TeX/LaTeX languages, code, and



algorithms, much of which date from the early 1980s. Computing and the way people interact with computers have evolved considerably since that time, but the programming languages and user interface paradigms that TeX/LaTeX were designed to work with have not kept up. Indeed, there are reasons why they *could not* keep up, having to do with the very ways in which these systems were architected and designed.

Let me elaborate on this. I will preface this critique by stating that I consider the TeX/LaTeX software stack to be among the best software systems ever developed. It is a true work of genius, and the fact that it is still widely used today, around 38 years after its introduction, is a testament to its many strengths. [*Insert mental image of this author lying prostrate in front of a home-made shrine and singing a hymn of praise to Donald Knuth, Leslie Lamport and their collaborators.*]

Soul-cleansing religious rituals aside, I must beg forgiveness for the blasphemous rant that follows, in which I describe the *shortcomings* of TeX/LaTeX from the vantage point of 2022.

- **An inefficient compilation model.** The TeX engine is built specifically as a *compiler* for the TeX language (which is classified as a **Turing-complete programming language**). That means that in order to generate its final product (these days, a PDF document; in the past, a **DVI file**) it scans your entire document code in a linear fashion, from beginning to end. This is done *each time you want to view your compiled code*.

When you are writing a short document of a few pages, this is usually not a big problem. But when writing a book, thesis, or other long document, the time spent waiting for your document to compile, and the effort of hitting the re-compile keyboard shortcut, becomes a big drain on the writer's time and cognitive bandwidth. Even with workarounds such as splitting your project into multiple smaller files and using conditional compilation commands to focus on the sections you are currently editing, this results in considerable

frustration and fatigue.

- **An outdated notion of what a “document” is.** TeX and LaTeX were developed at a time when the notion of a “document” was synonymous with a static pile of papers, or its electronic analogue. A document was meant to be viewed, but not interacted with in any way.

Today, most people will routinely refer to “documents” that are meant to be viewed on an electronic device and contain videos, audio files, clickable elements, expandable and collapsible sections, a linear (or nonlinear) slide presentation, and other interactive elements, up to and including even a complete video game.

Even in the limited context of mathematical documents, the **live notebook interface** popularized by software such as **Mathematica** and **Jupyter** has shown us that there are enormous benefits to having documents that one can interact with: expandable and collapsible sections make it much easier to navigate a long document with a complicated content hierarchy; interactive mathematical plots with manipulable elements such as sliders that control a parameter are an incredible tool for data visualization and intuition-building; animations and embeddable video clips can spice up a dry presentation, etc. All of those types of content are impossible to include in any document generated by the TeX engine.

- **An un-adaptable code base.** The TeX code base, descended from a version of TeX called TeX82, is written in the WEB programming language. It is compiled into C code using a utility called web2c. The code is not designed according to object-oriented programming methodology. These architectural issues make TeX very difficult to adapt for new uses and to integrate with other modern software tools. As far as I understand, a modern software


developer wishing to employ the TeX engine has to essentially use the engine as an impenetrable black box by writing TeX code into a file, running the TeX engine on that file, and then reading the output from the file generated by the engine. This is not a recipe for fostering innovation.



- **An outdated user interface paradigm.** TeX/LaTeX were designed as command line tools, the prevailing user interface paradigm for the period in which they were designed. Modern front-ends such as TeXShop, TeXWorks, WinEdt and **numerous others** attempt to hide this interface behind a modern user experience with an editor window, graphical controls and other design elements we expect in 2022. But they can do so with only limited success, as for example any errors produced by the typesetting engine have to be reported to the user, and this is done through a “console” window that produces the errors in the original, 1980s-style, headache-inducing format inherited from the original TeX. From the point of view of 2022 user experience standards, the TeX approach to error reporting and code debugging seems hardly acceptable.
- **No Unicode support.** TeX and LaTeX do not support the Unicode text encoding standard. (This is one of the main issues the **XeTeX** typesetting engine was developed to address.)
- **No accessibility support.** The area of **accessibility** has developed tremendously since the 1980s, but TeX and LaTeX offer essentially zero support for accessibility.


TeX/LaTeX have a variety of other, more minor, deficiencies that one learns about over time, and that one imagines may be addressed by new tools. (The same can be said of any other software, of course.) Some of them were already addressed by packages developed from within the LaTeX developer community, but others would be difficult to address due


to the structural issues I outlined above. Ultimately, it seems desirable to have mathematical writing tools that are built independently of the TeX toolchain, using modern approaches, with the needs of today’s users in mind, and without the historical baggage that reliance on TeX and its descendant systems entails.

## 1.4 How MadHat aims to satisfy the need


 **MadHat**, which I started developing in 2019, represents my initial (though by no means final) attempt at realizing my vision of a “dream” mathematical writing system. Its design was informed by my experiences with TeX/LaTeX, and with various other mathematical software systems (Mathematica, SageMath, and others), word processors, and text editors and formats such as Markdown. I also embarked along the way on a deep study of TeX’s typesetting algorithms, and of the weird and wonderful world of mathematical (and non-mathematical) typography. Many of the interesting ideas I learned about — as well as some new ones I thought up myself — influenced the design.



The journey of  **MadHat** is far from complete, but as of the current release (Version 1.1.1) I am now at a point where I feel comfortable saying that I have developed an actual tool that *works*, for at least some people to do at least some of the things they need. Moreover, it is a tool built with an eye towards the future, and that already addresses at a fairly good level several of the important structural issues that I saw with existing mathematical writing tools. Most importantly,  **MadHat** documents need no “compilation” — your typesetting code is compiled instantaneously as you type. This gives the software more of the feel of using a word processor than a computer language compiler.

 **MadHat** at this point in time consists of two main elements:


1. A content specification language, called  **MadHat**, for writing your document in; this is analogous to (and draws heavy inspiration

from) the LaTeX language.


2. A desktop app, called  **MadHat**, for editing and viewing your documents. This is analogous to the LaTeX graphical front-end systems (e.g., TeXShop).

My hope is that in releasing this book, and publicly releasing the source code for the current version (which I plan to do soon), I will kickstart a process that will get people using  **MadHat**, and enable me to get help (in various forms — see **Chapter 4**) in continued development of the project.  **MadHat** is not yet a complete publishing solution in the same way that a mature system like LaTeX is, but what I have created can be regarded as a foundation for creating a system that can aspire to evolve to someday be as good as, or better than, LaTeX.

## Chapter 2: The nine design principles of MadHat



When creating a complex piece of software like  MadHat, you find yourself having to make a large number of technical and design decisions. These decisions are connected by a complicated web of interconnections, with each decision affecting many others. One can quickly get overwhelmed by the sheer complexity of the structure one is building.

A helpful way to deal with the complexity is to have a set of principles that point the general way to the destination you are trying to reach. This makes decision-making somewhat easier.


Here are the principles I set to myself as I embarked on the journey of developing  MadHat:


1. no compilation
2. no error messages
3. no invalid code
4. Unicode support
5. eliminate binary files
6. expansive notion of a “document”
7. book-quality typesetting
8. beautify and simplify code
9. “Real artists ship”

Below I lay out the principles in detail. I also describe some of the consequences they led to, and the tradeoffs and choices I had to make

when principles clashed with practical realities, or with other principles. I hope this will go some way to explain my thought process and why  **MadHat** works the way it does. Where appropriate, I have added links to the relevant sections in the  **MadHat** user manual which comprises the later part of this book.


## Design Principle 1: no compilation






The “no compilation” rule was my North Star in developing  **MadHat**. The need to compile documents in LaTeX is both my biggest personal source of frustration when writing, and, in my opinion, the biggest impediment to mass adoption of LaTeX by “ordinary” (that is, non-technical) users. For this reason, right from the outset, every design decision I made was informed by the need to facilitate writing code that gets mapped into a rendered document, instantaneously — or as close to instantaneously as can be achieved.

This goal of instantaneous typesetting is difficult to achieve, and forces some serious constraints on how a  **MadHat** document needs to be structured. The issue is that the way a given piece of text gets typeset is heavily context-dependent. Typesetting algorithms have a state they keep track of, which includes contextual information about the current font, font size, weight, text color, etc. In theory, typing even a single character in the code can have downstream effects that affect the way the entire part of the formatted document below the current editing position looks. It can even have *upstream* effects that affect the *earlier* part of the document. (This issue comes up in LaTeX with equation and section cross-referencing, the table of contents, page reference numbers, and more; this is why LaTeX code usually needs to be compiled **multiple times** to produce a correctly formatted document.)

I developed a few strategies to make the goal of instantaneous typesetting achievable:




- **Separate document content and configuration.** The principle of separating content from styling/configuration in digital documents is well-known. In the world of web design, this manifests as the organization of web page files into CSS files (styling) and HTML files (content). In LaTeX, documents are logically divided into the preamble (configuration) and body (content).

In  **MadHat**, **notebook configuration** is handled in a separate editing interface from the main notebook code editor. The configuration code can be edited freely, again with instant responsiveness, but any changes to this code, *when applied to the notebook*, result in the notebook pages being retypeset — an operation that may take a few seconds. This is one necessary deviation from instant responsiveness, but the idea however is that configuration code is not edited frequently, so this momentary pause does not interfere with the normal workflow of editing the actual content of the notebook.


- **Compartmentalize: the submarine document model.** Ships and submarines are built with a compartmentalized design, to contain damage in the event of **flooding** or **fire**. Borrowing from this naval engineering concept, the need for an efficient parsing model for the  **MadHat** language that will allow instantaneous typesetting led me to a similar compartmentalized design for  **MadHat** notebooks that will ensure that only a minimal amount of content needs to be retypeset with each user edit. The main types of “compartments” (in addition to the notebook configuration code section described above, which is itself a kind of compartment) are:
  - **Paragraphs.** A **paragraph** of  **MadHat** code represents the basic unit of typesettable  **MadHat** content, in the sense that with each user keystroke or other editing operation,  **MadHat** will retypeset the code paragraph (or paragraphs, in some cases) affected by the edit.



For each paragraph, the typesetting context at the beginning and end of the paragraph is memoized. In the event that your edit operation affects the typesetting context, for example if you enter a command to change the font size or turn on italics, the change in the context will percolate down to the following paragraphs, which are also retypeset as appropriate until the typesetting context stabilizes. In this way, for those infrequent user edits that cause a substantial amount of the page content to be retypeset, that will be handled seamlessly (with a momentary delay, which again is not a problem because such operations are rare).



- **Pages.**  **MadHat** notebooks are separated into **pages**. This has many advantages, and makes a lot of sense regardless of the technicalities of implementing instantaneous retypesetting, since it makes it easier to organize the content in large projects. However, one of the less obvious advantages of this organizational structure is that it creates a natural compartment boundary for where retypesetting can stop in the event of an edit that percolates a long distance down the stream of notebook content.
- **Avoid pagination.**  **MadHat** notebook pages are scrollable surfaces of unlimited height, similar to web pages. Thus, unlike LaTeX and traditional word processors,  **MadHat** does not put a strong emphasis on paginating your document into discrete pages of equal size; the idea is that pagination is something writers really don't care much about or need to pay attention to until they are in a fairly advanced stage of the writing process, so it's a waste of CPU cycles and the user's attention to be constantly calculating page boundaries the way most word processors do, or the way LaTeX does each time you compile a document.


Deferring or avoiding pagination reduces the complexity of the

retypesetting algorithm, and again helps makes typesetting happen instantaneously or nearly so. However, rest assured:  **MadHat** still knows how to do pagination, and will perform it at the time when you are printing your notebook or exporting it as a PDF.

I haven't yet solved all the technical issues associated with enforcing the no compilation principle. Equation and section cross-referencing, and automated table of contents generation, listed in the section on missing features in **Chapter 3**, are some of the areas where I foresee having to come up with new ideas for how to minimize the disruption caused by having to retypeset several different places in the document in response to a user edit that affects cross-referencing. This is a pending issue and poses somewhat of a challenge to deal with in a satisfactory way, but I am confident that it can be addressed successfully.


## Design Principle 2: no error messages

Another source of frustration that motivated me were the abstruse error messages produced by the TeX compiler. While error messages are a fact of life for anyone who writes code — and  **MadHat** code *is* code, after all — the TeX-style error reporting format gave me such an allergy that I decided to try making a firm rule that in  **MadHat** there will be no error messages. I wanted to follow that decision to its logical conclusion and see if a usable system can still emerge.



The need to stand by this principled decision led to several corollary decisions that were interesting in their own right. First, the  **MadHat** language had to have less power than a full programming language: in the case of TeX, the awesome power of **Turing-completeness** is one of the things that, although beloved by TeX's more hard-core fans, adds complexity and ends up making code very difficult to debug (this issue is a known one that is discussed in various places, e.g., [here](#) and [here](#)). By making the language less powerful and expressive, a lot of things become


simpler.

Second, I found that it is not terribly difficult to develop creative ways of guiding the user to write code that does what they want without explicitly displaying text-based error messages. In other words, I came to the view that text-based error messages are a lazy solution we inherited from an earlier era of computing history, but that is actually largely unnecessary today. For example, **syntax highlighting** can be used to indicate to the user that a keyword they typed is not a valid symbol in the language. **Code completion** helps the user find the correct symbol name they are looking for; unmatched brackets can also be indicated using syntax coloring cues, etc. Such mechanisms can perhaps be *augmented* with textual error messages, but the default system behavior should be that of a kind of “butler” who quietly assists the user but stays out of the way as much as possible.

Third, the need to eliminate error messages also led me to the realization that the whole concept of syntax errors in computer languages is itself somewhat outdated, and is unnecessary in the context of what  **MadHat** is trying to do. This led me to the next design principle.

## Design Principle 3: no invalid code

This principle states that any string of (Unicode-encoded) text is considered valid code in  **MadHat**. That is,  **MadHat** is maximally forgiving of “errors”, to such an extent that the whole notion of an “error” loses its meaning, and there is no longer a reason to display an “error message”. No matter what code you punch in, *something* will be displayed as your rendered document content, and your workflow will not be interrupted by a compiler complaining that you made an “error.”


Of course, what can and does occasionally happen when you are writing content in  **MadHat** is that you write code *that does not produce the output you wanted*. This might be considered by some people as the same thing

as an “error”; but I find it helpful to make a distinction between *code that the software refuses to process* (the dreaded “syntax error”) and *code that does not give the results you want*. With the former, the user has their workflow interrupted, and may over time get the sense that the software is shaming them for doing something “wrong.” With the latter, the user merely sees that their code has some issues, and can work on resolving those issues, using the various mechanisms the software has for pointing them towards recognizing and fixing the issues.

It is a subtle distinction and perhaps mainly a philosophical one, but nonetheless, it has been a useful guiding principle for me.




## Design Principle 4: Unicode support




The lack of support for Unicode is a major shortcoming of LaTeX (although it is a problem that people in English-speaking countries find it easy to not care about). The XeTeX system, a variant of LaTeX, was developed to address that among other issues. As someone who was designing a new system from the ground up and had modern text processing APIs available to work with, including support for Unicode as a basic design principle seemed to me like an obvious decision that would offer many benefits and maximize the usefulness of the software for the largest number of people.



Moreover, Unicode support can be seen as a step towards the broader goal of improved accessibility of the software to people from all cultures. Eventually, my goal is for the  MadHat application interface and documentation should be **internationalized and localized** (or, if you are a speaker of British English, internationalised and localised). There are many other accessibility features that could be added over time, such as the long sought after goal of a mathematics-aware screen reader (see these online discussions: [\[1\]](#) [\[2\]](#) ).

## Design Principle 5: eliminate binary files


As a further nod towards the idea of making software that is more human-friendly and accessible, I wanted to move away from the basic compiler model of a software system that reads an “input file” of “source code” (typically, a plain text file) and generates a “output file” that is usually in a binary file format. This separation of “source content” from “compiled content” in my opinion reduces content accessibility and future-proofed-ness, since with these types of compiled formats it is often the binary file that gets distributed, with the human-readable and -editable file that generated that binary content being either lost through carelessness or indifference, or intentionally withheld.




I decided that in  **MadHat** there will only be one kind of file (namely, a “.mhat” file, representing a  **MadHat** notebook, or document), which will consist only of human-readable content, to the extent possible. Thus, there is no file format for a “compiled  **MadHat** document”, nor a need for such a format, since the content is simply compiled in real time when you open the document in order to view it.

In practical terms,  **MadHat** notebooks do allow for binary content in the form of image and video files, so this principle had to be compromised in a small way. Also, a  **MadHat** notebook is technically a folder that presents to the user as a single file (using the macOS concept of a **package**) and contains the  **MadHat** code text files of the notebook’s **pages**, as well as any binary **media files** that are included in the notebook.


As another acknowledgement of the fact that  **MadHat** operates in the real world rather than the utopian world of my dreams, if your goal is to produce a PDF document from your notebook,  **MadHat** will happily **export your notebook to PDF**, again compromising the “no binary files” principle. Similarly to **Groucho Marx**, I have a firm set of principles, and if you don’t like them... I have another set!



## Design Principle 6: expansive notion of a “document”

I mentioned in **Chapter 1** that I found the TeX/LaTeX premise that a “document” is a static collection of pages limiting and outdated. I decided that  **MadHat** will adopt a more modern view of what a document is, and will attempt to provide users with the tools to create dynamic documents containing interactive elements, animations, videos, sequenced slide presentations, and more.

The document structure I ended up with in  **MadHat** is what I came to call  **MadHat notebooks** (I have no claims to originality on this name, it goes without saying). This concept realizes many elements of my vision of dynamic and content-rich documents. I hope that future  **MadHat** versions will enable even more dynamism and interactivity.

## Design Principle 7: book-quality typesetting

With all the attention paid to exciting dynamic features, an area in which it is easy to outdo an old horse like TeX that only knows old tricks, we should not forget that  **MadHat** is at its heart a tool for *writing*, and I wanted the typeset content it produces to be uncompromisingly elegant and beautiful. In other words, the typesetting has to be of a quality suitable for professional-level book publication.

Of course, the old horse that is TeX is still the unchallenged champion and is considered the gold standard in terms of the quality of its typesetting algorithms. The current version of  **MadHat** does not quite live up to the same standard, but I think it’s a good start, and future work will continue to be guided by this principle. (Meanwhile, I am **eating my own dog food** by writing and publishing this book using  **MadHat**. The result is not too shabby, I think.)

## Design Principle 8: beautify and simplify code

This is probably the least important of the design principles, but it's still an idea that informed my thinking. One of my gripes with LaTeX coding is that the code is, for lack of a better word, ugly. Again, I mean no disrespect to Donald Knuth and his pioneering work, it's just that things change over the course of 40-odd years. We are now older, wiser, have better keyboards, computer screens, a much larger character set to work with, etc. If you are designing a new computer language in the 2020's, why not move away from some of the annoying legacies of the early days of computing?

What I mean by “simplify code” is this: first, LaTeX has some inefficient and annoying (and hard to debug) syntax rules. Multi-argument commands are an example, e.g., the code to type out the fraction “ $\frac{12}{34}$ ” is `\frac{12}{34}`. This is a wasteful use of braces. In **MadHat**, this became `⌘frac⟨12; 34⟩`.

As another example, LaTeX interprets `alpha` typed in math mode as “the symbol  $a$  times the symbol  $l$  times the symbol  $p$  times [etc]”. Have you ever read a mathematical text where someone wanted to multiply those precise symbols in that precise order? I haven't. It is more efficient to interpret `alpha` as the Greek letter alpha and render it as  $\alpha$ , which is what **MadHat** does. So the general idea is that mathematical keywords do not have to be preceded by a backslash or special control character — in this way we save keystrokes and make the code nicer-looking and easier to read. And for those times when you do want to spell out the multiplication of several symbols (say, “ $m$  times  $u$ ”, which in LaTeX can be written `\mu`, in **MadHat** you can do that just by inserting a space between the symbols, typing `m u`. In this way you avoid writing `mu`, the keyword for the Greek letter mu ( $\mu$ ).

What I mean by “beautify code” is this: when TeX was invented, Donald Knuth had only the ASCII character set to work with. This led him to



assign various ASCII characters as special characters, while still reserving ways to use those symbols for use according to their “ordinary” meaning — that is, characters are (heavily) overloaded. When designing a new computer language these days, we have the entire Unicode world of characters available to work with, and can easily avoid this overloading, which makes code more verbose and ugly. So, I decided that in **MadHat**, there will be a set of **special characters** that will be used only as special characters and for no other reason. (On the off-chance that you want to actually “print” the special characters themselves, there are ways to do that, but the idea is that you will almost never want to do that, unless perhaps you are writing a manual on how to use **MadHat**). So for example, the LaTeX backslash `\` got replaced by the command symbol `⌘`; the curly braces `{}` were replaced in **MadHat** by more specialized braces `⟨⟩`; and the illogical dollar sign `$`, which after all signifies a particular unit of currency and has no logical connection to mathematics, was replaced with the **em-hat symbol** `̂`.

There are obvious tradeoffs involved in following this line of thinking. The main one is that no one knows how to *type* the command symbol and other esoteric characters I chose to use. See [this page](#) for how I chose to address this issue.

There are other minor decisions I made that were informed by the “beautify and simplify code” principle. I won’t describe them all here — you will find them scattered in various places across the user manual (which I know you will read from cover to cover... right?).

Some people who knew I was developing a typesetting system have advised me to try to make it backwards-compatible with LaTeX, i.e., the syntax should be as similar as possible to LaTeX syntax. That idea would clash with the “beautify and simplify code” principle and the decisions it led me to make, as I was describing here. On the other hand, I recognize that LaTeX-compatibility could make a lot of people who are already comfortable with LaTeX more easily inclined to try **MadHat** and perhaps



switch to using it. So, it's a dilemma that I struggle with, even today.

Perhaps it's my quirky personality, individualism, rebellious nature, or I don't know what exactly, but in the end I couldn't fathom the idea of creating Yet Another TeX Clone. I wanted to create something genuinely new that satisfied my particular aesthetic sensibilities. Perhaps it will turn out to be a self-defeating decision that dooms the entire enterprise to failure. But this is one place where in the clash between a philosophical principle and a practical calculation, the philosophical principle won out (for now at least).

As another post-hoc rationalization for my decision, consider this: there are many variants of TeX/LaTeX out there, and they all use the familiar syntax invented by Donald Knuth in the late 1970s. Wouldn't it be interesting for there to be at least *one* system that tries to go in a different direction, and give the TeX crowd a run for its money? Who knows, it might teach us something new, and find at least a niche audience of people who like to try new things.

As a final remark about this principle: I did conclude in the end after all my travails that code can probably never be beautiful, and in some situations it's also unrealistic to expect it to be simple. So perhaps much of my thinking on this issue has been for naught, and all I really managed to achieve was to create a language syntax that is *different* from what people are currently used to. Whether that's a good or a bad thing remains to be seen.

## Design Principle 9: “Real artists ship”

*“But to do that, to make a difference in the world and a dent in the universe, you had to ship. You had to ship. You had to ship.*

*Real artists ship.”*


— Steven Levy, **Insanely Great**

“Real artists ship,” a slogan coined by Steve Jobs as he cat-herded a band of brilliant Apple engineers into **shipping the Macintosh computer** in early 1984, is the principle that overrides all other principles. It symbolizes the ever-present tension between idealism and the need to deliver *something* that at least does, well, *something*.


If you catch me in an inconsistency or in a failure to abide by any of the above-listed principles, or even in doing something that is the exact opposite of what I said I wanted to do, I will point to this principle and say in my defense: at least I shipped *something*.


## Chapter 3: Future plans


### 3.1 Overview of plans

 **MadHat** is at an early stage of its life cycle, and much work remains to be done, probably by many people, to get the project to where it needs to be in order to be usable by a large number of people in the way that I imagine it can be. I will be the first to admit that the project is far from finished. (But I offer no apology for this; see the “Real artists ship” design principle in **Chapter 2**.)

At the same time, the set of things to be done is fairly concrete, and these goals are all achievable given time, persistence, and support from people who believe in the concept.

My plan for the immediate future is to continue to develop the software and add more features, while recruiting support from other people who are interested in  **MadHat**. This support is essential, as the scope of the project is clearly too large for one person to handle successfully. (See **the next chapter** for various ways in which you can help.)

A medium-term plan is to set up an organizational structure in which a group of people can work to continue to improve  **MadHat**. I have not decided what this structure would be — it may be as a non-profit, a for-profit enterprise, or a loose collection of individuals working towards a shared goal. (Or something else entirely! I am open to suggestions.)

At a more concrete level, here is a list of the main shortcomings that exist in the current version, which will need to be addressed to take  **MadHat** to the next level in terms of its usefulness.

### 3.2 List of missing features

- Section numbering and cross-referencing
- Equation numbering and cross-referencing
- User-definable environments — theorems, definitions etc.
- User-definable symbols
- Line-breaking algorithm for double-justified paragraphs
- Hyphenation algorithm
- Floating figures
- Footnotes
- Bibliography support
- Table of contents
- Customizable user-definable sections for PDF export
- Importing content from LaTeX





### 3.3 Existing features that can be improved

- User interface improvements
- Customizable formats for **lists**
- Customizable formats for **page numbers**
- More options for **tables**
- More options for **slide animations**
- Improved **plotting of mathematical functions**
- Improve **LaTeX exporting**



- Improve Unicode support

## 3.2 Longer term goals


Assuming I am successful in finding the time, help and other resources I need to keep the project going, there are some longer term goals that I would like to see accomplished:

- Adapt  MadHat to run on Windows and/or Linux.
- Adapt  MadHat to run on iPad, iPhone, and Android phones and tablets
- Adapt  MadHat to run on a web browser
- Release a public API that enables  MadHat content to be processed and used by third-party software
- Add a mouse-based interface for editing graphical diagrams
- Add a mouse-based interface for editing animations
- Add a **mathematics-aware screen reader** and other accessibility features

## Chapter 4: Summary and a call for help







The preceding chapters have been a kind of brain-dump in which I shared my vision for how  **MadHat** can advance mathematical typesetting (and why that would be a good thing) and the various dilemmas I have been dealing with since I started working on it. It is a fairly high-level description, and there are many details I did not discuss. If you are interested in learning more, read on to the second part of the book, which is the  **MadHat** user manual that describes in detail everything the software can do.

One of my main reasons for writing this exposé is that if the project is to become something more than a hobby project of a lone programmer, I need your help and the help of other people, so I felt a need to explain where I am and where I'd like to be going.

An obvious way for you to help is to use  **MadHat** yourself, send me feedback that can help me improve it, and tell other people about it. There are many other ways in which you can help; here are some of them:

- **If you have money:** please donate some money to the project. I have set up a donation button [here](#).
- **If you have useful skills:** contributing your time and skills could be even more valuable than money, so feel free to [contact me](#). Here are things I need help with:

- Coding
- Writing documentation and marketing materials
- Graphic design

- Making video tutorials
  - Building and maintaining a website
  - Helping to set up a legal structure for  MadHat
  - Fundraising, setting up a Kickstarter campaign
  - Creating  MadHat-themed merchandise and an online store
  - Other legal, business advice, etc
- **If you are an academic.** You can help by inviting me to speak at your institution to help spread the word about  MadHat. You can also tell your colleagues about the project.
  - **If you are a student.** Please use  MadHat for your own work, and tell other students and your professors about it. If you are looking for internship opportunities, [contact me](#).
  - **If you are active on social media.** Spread the word about  MadHat!
  - **If you know other people** who fall into any of the above categories, tell them about  MadHat, and refer them to this page.

# *Part II*






Version 1.1.1

# User Manual





## About MadHat



 **MadHat** is a system for writing documents. It consists of a language for coding up the text and other content of your document, and an app to edit and view  **MadHat** documents.

**System requirements.**  **MadHat** is a macOS app. It will run on Mac computers running version 10.13 or higher of macOS.




## About this help guide

This guide documents the main features of  MadHat. An interactive version of the guide is included as the Help feature within the  MadHat app. To access it, click Help → MadHat Help in the app main menu.

**A note about code snippets.** Throughout the guide, you will find many examples of code in the MadHat language. These code snippets are displayed in the default **syntax highlighting theme** of the MadHat code editor. For example, the code snippet

```
"In italic«that» direction," the Cat said,
waving its right paw round, "lives a Hatter: and
in italic«that» direction," waving the other paw,
"lives a March Hare. Visit either you like:
they're both mad."
```

displays a **paragraph** of code parsed in the default mode for entering code, known as **text mode**, as it would appear when entered in the code editor. Similarly, the snippet

```
: a = m^2-n^2 newline.
b = 2mn newline.
c = m^2 + n^2
```

shows a paragraph parsed in **math mode**.

Clicking on a code snippet in the interactive help window copies the code shown in the snippet into the clipboard. You can then paste it into your own notebook and experiment with it.



# MadHat Help

Version 1.1.1 (April 2022)

Use the search bar to search for help on a specific topic, or browse the help topics suggested in the links below

For sample code and additional resources, go to <https://madhat.design>


- Essential help topics:
  - [About this help guide](#)
  - [Introduction to MadHat and MadHat notebooks](#)
  - [Typing text with MadHat](#)
  - [Typing mathematical formulas with MadHat](#)
  - [Special symbols in MadHat](#)
  - [Key substitutions in the editor window](#)
- Help topics on the MadHat language:
  - [Grouping content in blocks](#)
  - [MadHat commands](#)
  - [Command attributes](#)
  - [Delimited lists](#)
  - [Notebook configuration](#)
  - [Paragraphs](#)
  - [Spaces and newlines](#)


- **Styling text**
- Text styling options: **bold**, **italic**, **underlining**, **strikethrough**, **highlighting**, **substitutions**
- Setting **fonts** and the **font size**
- **Colors**
- **Headers and subheaders**
- **Lists and their use for outlining**
- **Formatting tables**
- **Formatting boxes**
- **Hyperlinks and intralinks**
- **Creating slide presentations**
- **Mathematical formatting topics:**
  - **Subscripts and superscripts**
  - **Fractions**
  - **Square roots**
  - **Greek letters**
  - **Mathematical font variants**
  - **Differentials**
  - **Standard mathematical operators**
  - **Brackets**
  - **Horizontal brackets**


- **Extensible symbols**
- **Special mathematical symbols**
- **Binary relations**
- **Binary operators**
- **Matrices**
- **Mathematical symbol decorations**
- **Adding images to a page**
- **Adding videos to a page**
- **Drawing graphics figures**
- **Plotting mathematical functions**
- User interface topics:
  - **The media library of a notebook**
  - **Printing and PDF exporting notebooks**
  - **Exporting notebooks to LaTeX**
  - **Syntax highlighting themes and the themes editor**
- Additional information:
  - **About MadHat**
  - **MadHat version history and changelog**
  - **Index of MadHat commands and keywords**

# Introduction to MadHat

## Overview

 **MadHat** is a system for writing and reading documents. It consists of a language for coding up the text and other content of your document — called the MadHat language — and an app to edit and view MadHat documents.


 **MadHat** is suitable for use by anyone who wants to write anything, either for personal use or for sharing with others. It is especially optimized for writing content that contains mathematical formulas.

 **MadHat** documents are called **notebooks**. A notebook consists of one or more **pages**. Each page is a scrollable body of content of potentially unlimited length.

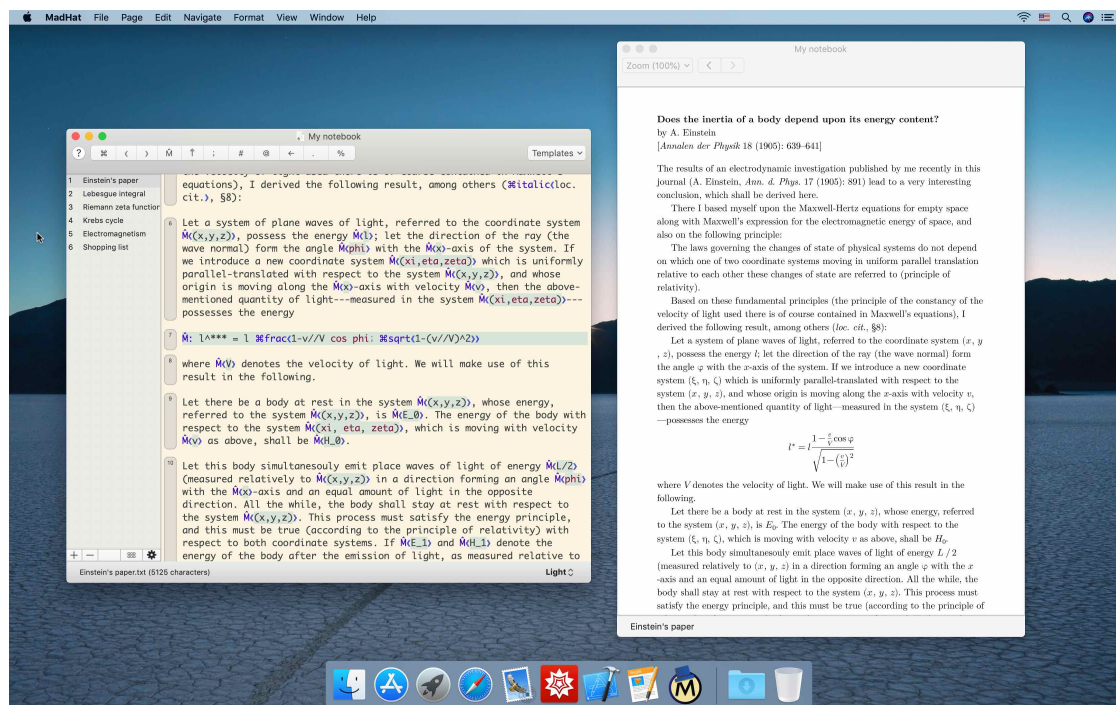
The types of content that a  **MadHat** notebook can contain are:

- Text
- Mathematical formulas
- Slide presentations
- Images and graphical diagrams
- Videos

## The notebook windows

 **MadHat** notebooks are viewed and edited in two main windows, the **editor window** and the **viewer window**, shown in the screenshot

below. The editor window (shown on the left) is where you enter the code that specifies the content of the notebook. The viewer window, shown on the right, is where the formatted content appears.






## Important things to know

Here are some of the main things to know about **MadHat** notebooks:


- A page in a notebook can (but does not have to) consist of multiple **slides**, which are intermediate states in which only some of the page content is shown.
- **MadHat** notebooks can be **exported to PDF format**. However, viewing a notebook in the native viewer inside the app rather than as an exported PDF document makes it possible to interact with the notebook in ways that are not possible to do with a PDF document.
- The content in a page can be further partitioned into sections,

















subsections, etc. Pages can also contain hierarchical content organized in itemized (numbered, or unnumbered) lists. Sections as well as list items can be easily closed (“collapsed”) and opened (“expanded”) by the reader.

-  **MadHat** offers features for easy navigation between pages and slides; internal linking across pages (intralinks); external linking (hyperlinks) to web pages; and many more features that make it useful as a means for sharing information and ideas.
-  **MadHat** adopts the **Unicode** text encoding standard as its technical foundation for representing textual content. This makes the full range of international character sets available for use. Currently only left-to-right writing is supported.
-  **MadHat** notebooks can be **exported to LaTeX**.



# List of special symbols


The  **MadHat** language reserves certain Unicode characters as having a special meaning in entering code. Here is the list of special characters and their meanings, with links to the help pages explaining how they are used.

- Characters that are used in  **MadHat** code
  - **Command symbol**  (Unicode: **U+2318**)  
**Keyboard shortcut:**  (backslash key)
  - **Open block symbol**  (Unicode: **U+27EA**)  
**Keyboard shortcut:**  (left square brace key)
  - **Close block symbol**  (Unicode: **U+27EB**)  
**Keyboard shortcut:**  (right square brace key)
  - **Close command symbol**  (Unicode: **U+FF0E**)  
**Keyboard shortcut:**  (period key — this substitution applies only at the end of a command)
  - **Math shift**  (Unicode: the letter “M” followed by **U+0302**)  
**Keyboard shortcut:**  key (dollar sign key)
  - **Text shift**  (Unicode: the letter “T” followed by **U+0302**)
  - **Attributes symbol**  @ (Unicode: **U+FF20**)  
**Keyboard shortcut:**  key
  - **Attribute declaration operator**  (Unicode: **U+2190**)
  - **Primary list delimiter**  ; (Unicode: **U+FF1B**)

**Keyboard shortcut:**  key (semicolon key)

- **Secondary list delimiter**  (Unicode: **U+FF03**)

**Key shortcut:**  (sharp sign key)

- **Comment symbol**  (Unicode: **U+FF05**)

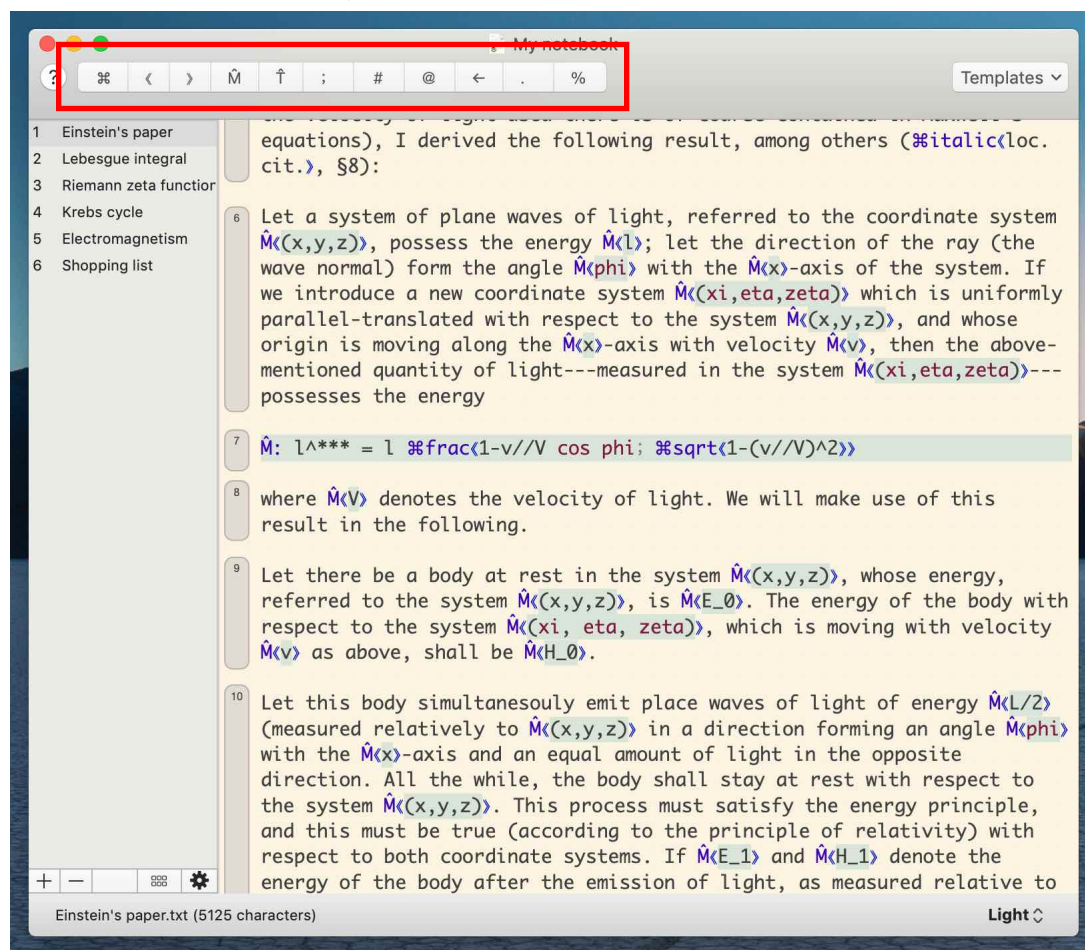
**Key shortcut:**  (percent key)

- Forbidden characters
  - The Unicode characters **U+3010**, **U+3011**, **U+3016**, **U+3017**, **U+2053** are reserved and should not be used.

## Typing special symbols

You can type special symbols by clicking the appropriate symbol in the special symbols bar in the editor window toolbar:


special symbols bar



You can also use the various keyboard shortcuts described in the help page on **key substitutions**.

# Typing text

## The basics



Typing text in  MadHat is as easy as... typing it. For example, in the editor window, you can type:

```
"In that direction," the Cat said, waving its
right paw round, "lives a Hatter: and in that
direction," waving the other paw, "lives a March
Hare. Visit either you like: they're both mad."
```

This typesets in the page viewer window as:




“In that direction,” the Cat said, waving its right paw round, “lives a Hatter: and in that direction,” waving the other paw, “lives a March Hare. Visit either you like: they’re both mad.”

We can now start adding some formatting to our text by using **commands**. Typing the code

```
"In italic<that> direction," the Cat said,
waving its right paw round, "lives a Hatter: and
in italic<that> direction," waving the other paw,
"lives a March Hare. Visit either you like:
they're both mad."
```

will typeset as:

“In *that* direction,” the Cat said, waving its right paw round, “lives a Hatter: and in *that* direction,” waving the other paw, “lives a March Hare. Visit either you like: they’re both mad.”

Note that the above code uses the symbols , , and , which are some of

the **MadHat special symbols**. These symbols are indispensable for formatting any content with richer formatting than basic, unstyled text.

Two additional help topics that are relevant to typing basic text content are **spaces and newlines** and **code paragraphs**.

If you mastered the above topics, you are now ready to start typing text. You can learn about more options for formatting and styling your text in the following help pages:


- **Styling text**
- **Bold text formatting**
- **Italic text formatting**
- **Underlining**
- **Strikethrough**
- **Highlighting**
- **Text substitutions**
- **Setting fonts**
- **Setting the font size**
- **Colors**


For even more advanced text formatting options, refer to these help pages:

- **Headers and subheaders**
- **Lists and their use for outlining**
- **Formatting tables**
- **Formatting boxes**

- **Hyperlinks and intralinks**

## Automatic substitutions

 MadHat implements the following automatic substitutions that make it easier to type certain characters not supported by most standard keyboard layouts.

- A unidirectional apostrophe `'` gets transformed by the  MadHat parser into a right quotation mark character: `'`
- Unidirectional double quote characters `"` are matched with each other, transforming them into the appropriate left or right double quotation marks. For example: `"Hello," she said` will typeset as

“Hello,” she said.


- An ordinary hyphen character `-` will typeset as an ordinary hyphen `-`. Two repeated hyphens `--` are typeset as an en dash `–`. Three repeated hyphens `---` are typeset as an em dash `—`.

The substitution feature is designed as a convenience, but you also have the option of inserting the “fancy” substituted characters (directional single and double quotes, en dashes, em dashes) directly into your code, by typing them if your keyboard layout supports it, or by using the standard macOS interface for special symbols, accessible by selecting Edit → Emoji & Symbols in the app main menu.

## See also

- **Typing mathematical formulas with MadHat**

# Spaces and newlines

 **MadHat** interprets space and newline characters in your code in a particular way that tries to balance functionality, convenience, and code readability. In addition to the use of those characters, the commands `⌘space⟨...⟩` and `⌘newline⟨...⟩` are available for inserting horizontal and vertical spaces of arbitrary dimensions in your document. Those commands also have the advantage of being clearly visible to anyone reading the code.

## Spaces and newlines in text mode

### Horizontal spaces

Space characters inserted in your code map to a horizontal space in your formatted notebook page, according to the following rule:

- One space 


This typesets as an ordinary space, whose width is specified in the font you are using.

- Two consecutive spaces 

This typesets as a wide space that is slightly wider than an ordinary space.

- Three consecutive spaces 


This typesets as a double space, that is, a space of exactly twice the width of an ordinary space.

- Four or more consecutive spaces 


This typesets as a quadruple space, that is, a space of exactly four times the width of an ordinary space.

- `⌘space`*«space width in points»* command

Inserts a horizontal space of the specified width.

Some  MadHat users may find the wide space a convenient type of space to insert at the end of a sentence. Modern English typographical conventions seem to favor the use of ordinary spaces at the end of a sentence. See [this Wikipedia article](#) for further discussion.

## Newlines and vertical spaces

Within a paragraph of  MadHat code, a newline character gets interpreted as a line break. This effect can be suppressed by inserting a **comment symbol** just before the newline character.


You can insert a line break without a newline character by writing the command `⌘newline.` and continuing the code in the same line.

To insert a vertical space of arbitrary height, use the `⌘newline`*«...»* command, with the syntax:

`⌘newline`*«space height in points»*


## Spaces and newlines in math mode

Space and newline characters get processed differently in math mode than in text mode. The main rules to keep in mind are:

- Spaces in math mode have no effect on the typesetting of your formula, other than to serve as logical separators between different symbols. For example,  MadHat will interpret “**xy**” as “the two-letter symbol ‘xy’”, but will interpret **x y** (and also **x y**, **x y**, etc, regardless of the number of spaces you insert) as



“the one-letter symbol ‘ $x$ ’ followed by the one-letter symbol ‘ $y$ ’.

Based on this interpretation, the  **MadHat** typesetting engine uses a custom algorithm to insert an appropriate spacing between symbols, in keeping with the conventions and accepted aesthetics rules for mathematical typesetting.

- Newline characters in math mode are similarly interpreted as logical separators and otherwise do not affect the amount of space inserted by the typesetting engine.
- In a **math display**, you can use the `\newline.` command to insert a logical line break, specifying the beginning of a new equation or chain of equations/inequalities. This is useful when typesetting multi-equation displays, such as

```

 $\hat{M}$ :
tau = beta (t-vx/c^2), \newline.
xi = beta (x-vt), \newline.
eta = y, \newline.
zeta = z.

```

which will typeset as

$$\tau = \beta(t - vx / c^2),$$


$$\xi = \beta(x - vt),$$

$$\eta = y,$$

$$\zeta = z.$$

# Typing mathematics

## Math mode

 **MadHat** makes it easy to typeset mathematical expressions by entering **math mode**. This is done using the em-hat, or **math shift**, symbol  $\hat{M}$ . If your expression is part of an ordinary text paragraph (in which case we refer to it as **inline math**), the syntax for inserting such an expression in your text is

*some text content...  $\hat{M}$ ⟨your mathematical content⟩ ...more text content*

Alternatively, the mathematical content can be shown in its own space separated vertically from the surrounding text, known as a **math display**. The syntax for this is

$\hat{M}$ : *your mathematical content*

See also the help page on **paragraphs**.


A keyboard shortcut to enter math mode is to type the dollar sign ‘\$’. See the help page on **key substitutions**.

## Syntax for mathematical content

To understand how to typeset the main elements of a mathematical expression, refer to these help pages:

- **Fractions**

- **Subscripts and superscripts**
- **Greek letters**
- **Commands and math keywords**

 **MadHat** supports many additional constructs in the vocabulary of mathematical formulas. Here is a complete list of the supported mathematical typesetting features:

- **Fractions**
- **Subscripts and superscripts**
- **Greek letters**
- **Commands and math keywords**
- **Square roots**
- **Mathematical font variants**
- **Differentials**
- **Standard mathematical operators**
- **Brackets**
- **Horizontal brackets**
- **Extensible symbols**
- **Special mathematical symbols**
- **Binary relations**
- **Binary operators**
- **Matrices**
- **Mathematical symbol decorations**

## Automatic substitutions

Just like in **text mode**, the math mode parser performs some automatic substitutions to simplify entering common symbols. Here is the list of implemented substitutions:

- `-` (a hyphen) is replaced by the symbol “—”, the proper Unicode symbol for a minus sign
- `*` (a single asterisk) is replaced by the “times” symbol “ $\times$ ”
- `**` (two successive asterisks) are replaced by “ $\cdot$ ”, the dot symbol
- `***` (three successive asterisks) are replaced by “ $\ast$ ”, the star or convolution operator
- `<=` is replaced with the “less than or equal” symbol “ $\leq$ ”
- `>=` is replaced with the “greater than or equal” symbol “ $\geq$ ”
- `/=` is replaced with the “not equal” symbol “ $\neq$ ”
- `+ -` is replaced with the “plus or minus” symbol “ $\pm$ ”
- `- +` is replaced with the “minus or plus” symbol “ $\mp$ ”
- `...` (three successive dots) is replaced by the ellipsis symbol “ $\dots$ ”
- `||` (two successive vertical bars) are replaced by the double vertical bar symbol “ $\parallel$ ” (which, like the single vertical bar symbol, is interpreted by **MadHat** as a type of **bracket**)
- `'` (an apostrophe) is replaced with the prime symbol “ $'$ ”, the correct Unicode symbol for prime or derivative notation

See also: **binary relations**; **binary operators**; **brackets**

## Including text inside a mathematical expression

To include ordinary text from inside math mode, you need to leave math mode temporarily and re-enter text mode. With an inline math expression, you can simply close out the math mode block, which returns you to text mode typing. Within a math display, this is done using the **text shift** symbol  $\hat{\text{T}}$ . The syntax for this is

$\hat{\text{M}}$ : *some math content...  $\hat{\text{T}}$ «your text» ...more math content*

## Examples

The code paragraph

```
 $\hat{\text{M}}$ :
tau = beta (t-vx/c^2), \text{\texttt{\textbackslash new}}line.
xi = beta (x-vt), \text{\texttt{\textbackslash new}}line.
eta = y, \text{\texttt{\textbackslash new}}line.
zeta = z
```

typesets as

$$\tau = \beta(t - vx / c^2),$$

$$\xi = \beta(x - vt),$$

$$\eta = y,$$

$$\zeta = z$$

See also

- **Typing text with MadHat**

# Blocks

A body of MadHat code of the form `«some code»` is called a **block**. The symbols enclosing the block are the **open block** symbol `«` and the **close block** symbol `»`. See the **special symbols** help page.

A block cannot span across different **paragraphs** of code, that is, the matching open block and close block characters must lie in the same code paragraph.

You can use the keyboard shortcuts '[' and ']' (left square brace and right square brace) to type the open and close block symbols. See the help page on **key substitutions**.

Blocks allow you to change the typing style locally without affecting all the content that follows. Any change in the style only applies to the block in which it is applied, or globally if it is applied outside of any block.

For example, the code

```
A quick brown fox «#color«blue»jumps #bold on.
over the lazy» dog
```

will typeset as

A quick brown fox jumps over the lazy dog

Blocks can be nested inside each other. Thus, the code

```
A quick brown fox «#color«blue»jumps «#bold on.
over» the lazy» dog
```

will typeset as

A quick brown fox jumps over the lazy dog


The syntax for blocks is also used for the argument of a command. For example, the code

```
⌘bold«Very few castaways can claim to have  
survived so long at sea as Mr. Patel»
```

will produce the result:

**Very few castaways can claim to have survived so long at sea as  
Mr. Patel**

# Comments

You can insert comments in your code by using the comment symbol `%`. Any text that appears in a line of code following the comment symbol is not processed by  **MadHat**.

Inserting a comment also suppresses the effect of the newline character at the end of the line containing the comment. So, the paragraph of code


```
Hello, % a comment  
world
```

will typeset as:


Hello, world




# Commands and math keywords


Two main mechanisms for specifying content in  MadHat are **commands** and **math keywords**.

## Commands

Commands are text strings that start with the command symbol . They can be used in either text mode or math mode (although, depending on their purpose, some commands will be used primarily in text mode and others primarily in math mode), and come in two flavors:

- Commands that require no argument follow the syntax:

 **command** .


that is, the command symbol, followed by the name of the command, followed by the “**close command**” symbol .

- Commands that take one or more arguments follow the syntax:

 **command**  *argument block*

that is, the command symbol, followed by the name of the command, followed by a **block** with the argument or arguments. For commands that take more than argument, the arguments are still provided in one block, but are delimited using **list delimiters**.

## Examples

-  **bold on.** turns on bold typing. This command does not take an argument.

- `\bold⟨text to be typed in bold⟩` will typeset the text provided as an argument in boldface.
- `\fraction⟨numerator; denominator⟩` will typeset a fraction.

The behavior of a command can in some cases be modified by the inclusion of additional optional arguments called **attributes**.

## Math keywords

Math keywords are used in **math mode** to typeset mathematical symbols such as Greek letters, the infinity symbol, an integral sign, etc. They are meant to be easy to type, so they consist of just the keyword itself without a preceding command symbol or a trailing command closing symbol. In contrast to command names, keywords cannot contain a space character.

As an example, entering `sum_n a_n cos(2 pi n/3) = beta` produces the output

$$\sum_n a_n \cos(2\pi n / 3) = \beta$$

Here, the math keywords `sum`, `cos`, `pi` and `beta` are recognized.

## Aliases

Both commands and math keywords can have an **alias**, which is an alternative name for the same command. Some examples of commands with aliases are:

- The command `\color⟨...⟩` has the alias `\colour⟨...⟩`
- The math keyword `integral` has the alias `int`

- The math keyword `product` has the alias `prod`
- The command `\binomial⟨...⟩` has the alias `\binom⟨...⟩`
- The command `\bezier⟨...⟩` has the alias `\b\'ezier⟨...⟩`

# Attributes

Attributes modify the behavior of a command. They are included by inserting an attributes block of the form

```
@⟨attribute name←value; attribute name←value; ...; attribute
name←value⟩
```

somewhere (it does not matter where) inside the argument block of the command.

## Boolean attributes

Boolean attributes take either of the values “yes” or “no”. You can specify them in the straightforward way as `@⟨boolean att. name←yes⟩` or `@⟨boolean att. name←no⟩`, or using a shorthand notation that omits the `←` assignment operator:

`@⟨boolean att. name⟩` is equivalent to `@⟨boolean att. name←yes⟩`

`@⟨no boolean att. name⟩` is equivalent to `@⟨boolean att. name←no⟩`

For example, a `@⟨no crop⟩` attribute in a `⌘graphics canvas⟨...⟩` specifies that cropping should be turned off.

Some attributes accept as argument a *list* of Boolean values. For example, in a `⌘table⟨...⟩` command, the `hlines` attribute specifies which horizontal lines should be drawn. This list of values is provided in the form

```
att. name←xxxx...x
```

where each of the `x` symbols is either `y` (for “yes”) or `n` (for “n”).

## Examples

See the documentation for **paragraphs**, the **⌘graphics canvas⟨...⟩** **command** and the **⌘table⟨...⟩** **command** for examples of the uses of attributes.

# Delimited lists and delimited tables

A delimited list is a **block** that takes the form

*«1st delimited expression; 2nd delimited expression; ... ; last delimited expression»*


This uses the “primary list delimiter” symbol `;`. When such a block is passed as an argument to a command, the command can interpret each of the delimited expressions as a separate argument.


In addition to primary list delimiters, an additional delimiter symbol is the “secondary list delimiter” `#` which can be thought of as a “next line” character for specifying two-dimensional arrays. In this way you can succinctly specify tables and matrices for commands that are suitably designed as delimiter-aware. For example, entering

`\matrix«1; 2; 3#4; 5; 6»` in math mode will produce the output:

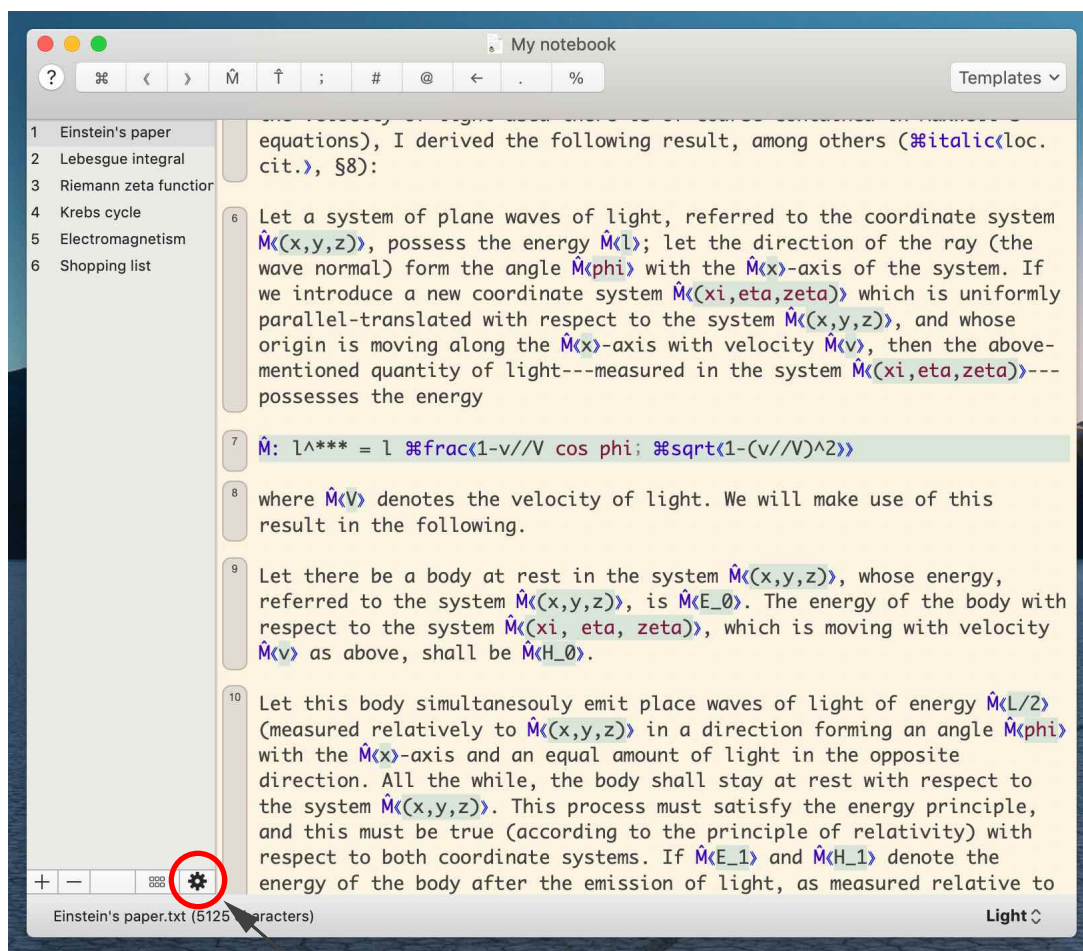
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

# Notebook configuration

The readability of a document depends not only on its content but also on how the content is presented.  **MadHat** makes available a collection of customization options that allow you to modify the appearance of an entire notebook and make it as engaging and aesthetically appealing as possible. This collection is referred to as the **notebook configuration**.

The notebook configuration is specified using code in the  **MadHat** language, called the **notebook configuration code**. This code is entered not in the usual code editing area, but through a separate editing interface, the **notebook configuration panel**.

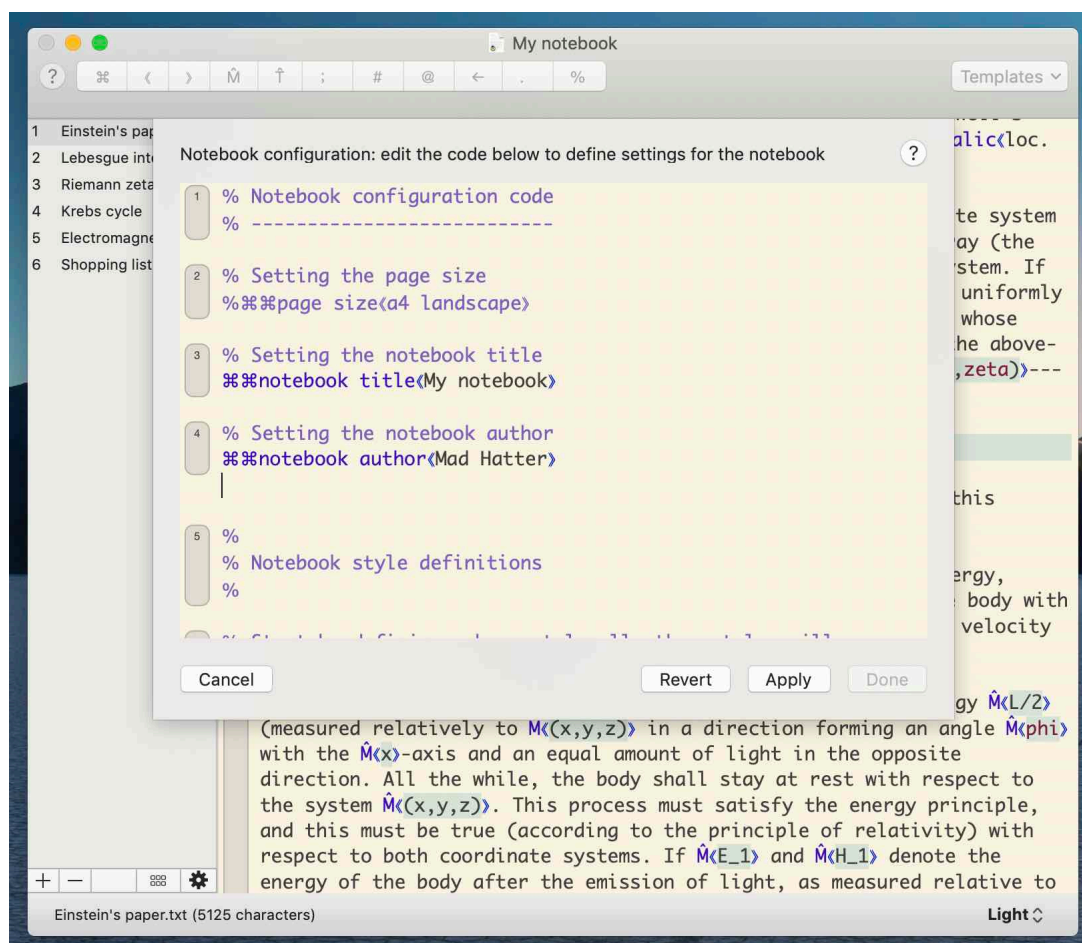
To access the notebook configuration panel, click the configuration icon in the bottom-left corner of the notebook editor window:



notebook configuration icon

The notebook configuration panel will appear and allow you to edit the configuration code, as shown in this screenshot:






In the configuration code, you specify the global behavior of the notebook using **notebook configuration commands**; these are commands that start with a double command symbol ( $\&\&$ ). They can only be used as part of the notebook configuration code.

The help pages linked below list the different aspects of the notebook behavior you can customize using configuration commands, with explanations about the relevant configuration commands and how to use them:

- **Customizing the notebook metadata**
- **Customizing the page geometry**
- **Customizing the notebook styles**
- **Customizing the notebook line and paragraph spacing**

- **Customizing the PDF export settings**

You can include text, code comments, or any other type of  MadHat content in your configuration code in addition to configuration commands. These have no effect on the notebook configuration.

When you create a new notebook, it will come equipped with a default template for the configuration code. This makes it easy to start editing the configuration settings.

# Customizing the notebook metadata

The notebook metadata is information about the notebook that is not a part of the notebook contents. Currently this refers to the notebook title and author. You can set these fields using the following **notebook configuration** commands:

- `⌘⌘notebook title⟨notebook title⟩`

Set the notebook title

- `⌘⌘notebook author⟨notebook author⟩`

Set the notebook author

When you export a notebook as a PDF, the title and author fields are used to populate the corresponding fields in the PDF document metadata. The notebook title is also displayed at the top of the viewer window.

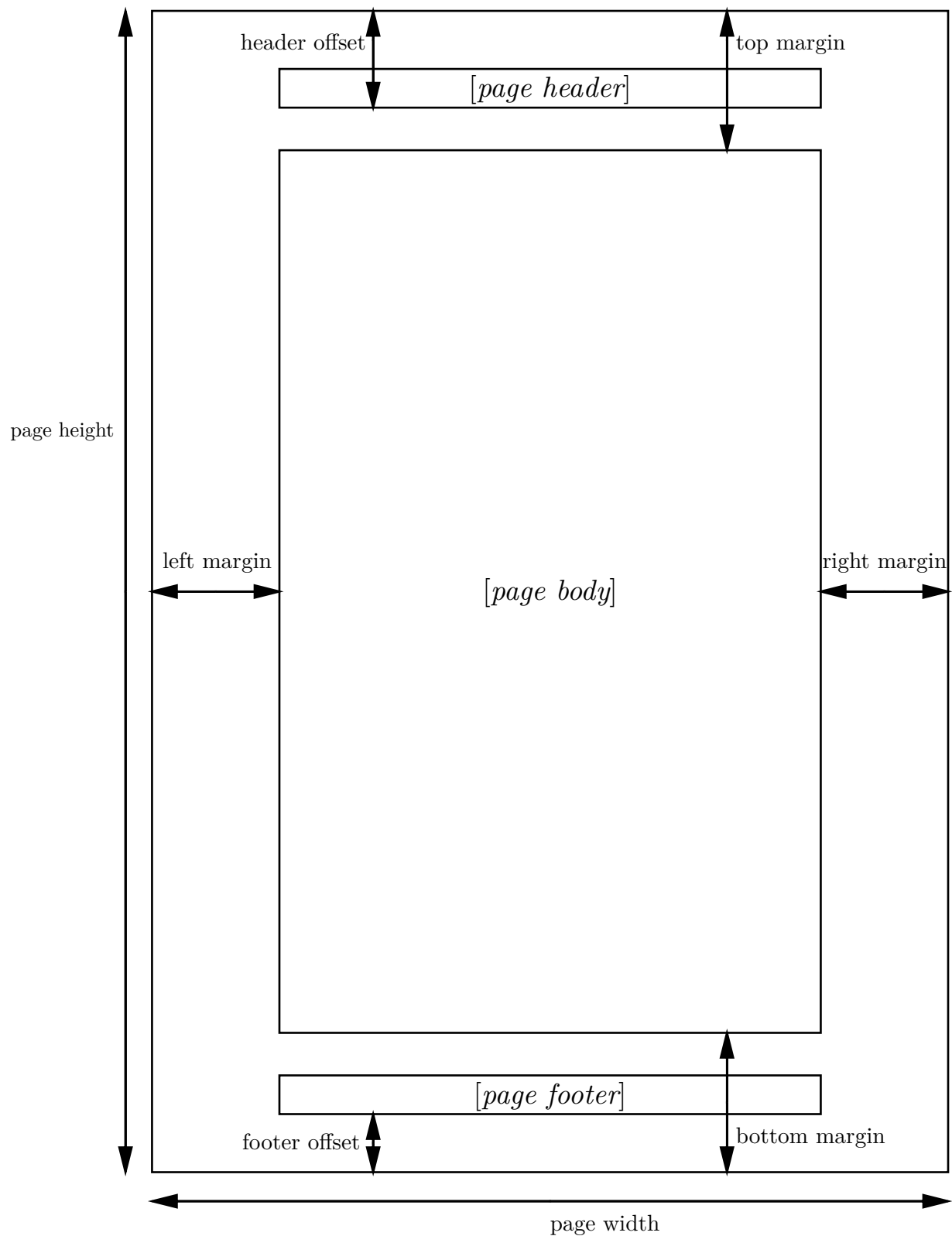
## See also:

- [Notebook configuration](#)
- [Customizing the page geometry](#)
- [Customizing the notebook styles](#)
- [Customizing the notebook line and paragraph spacing](#)
- [Customizing the PDF export settings](#)

# Customizing the notebook page geometry

**Page geometry** refers to the page dimensions (height and width) as well as the parameters affecting the positioning of the page content, and, in the version of the notebook exported to PDF, the page header and footer. All of these parameters can be customized through the use of appropriate **notebook configuration** commands, discussed below.

The basic page geometry parameters are shown in the diagram below:



A point to note is that page headers and footers are only added when the notebook is exported to PDF. Thus, the header offset and footer offset

parameter only apply to the PDF-exported version of the notebook. Furthermore, when specifying the vertical (top and bottom) margins, you can specify separate values for the vertical margins as viewed in the viewer window, and for the margins to be used for PDF exporting. This duplication facilitates making adjustments to account for the presence of page headers and footers in the PDF-exported version.

By contrast, for left and right margin widths, the same value is used for page layout in the viewer window and for the PDF-exported version of the notebook.

## Setting the page size

The page dimensions — a width and height measured in points — are the default dimensions for the page viewer window (which can be resized to an arbitrary size). They are also used to specify the page dimensions when the notebook is exported as a PDF. You can set them using the following commands:

- `%%page size`*⟨page width; page height⟩*

Set the page size (width and height), measured in points

- `%%page size`*⟨page size keyword⟩*

Set the page size to one of the standard sizes given by the following key words:

- `default`
- `letter portrait` (alias: `letter`)
- `letter landscape`
- `a4 portrait` (alias: `a4`)

- `a4 landscape`
- `large`

## Setting additional page geometry parameters

The page geometry parameters other than the page dimensions can be customized using the following commands:

- `%%left margin⟨left margin width⟩`

Set the page left margin width, measured in points.

- `%%right margin⟨right margin width⟩`

Set the page right margin width, measured in points.

- `%%top margin⟨top margin height⟩`

Set the top margin height, measured in points.

- `%%bottom margin⟨bottom margin height⟩`

Set the bottom margin height, measured in points.

- `%%exported top margin⟨top margin height⟩`

Set the top margin height in the version of the notebook exported to PDF, measured in points.

- `%%exported bottom margin⟨bottom margin height⟩`

Set the bottom margin height in the version of the notebook exported to PDF, measured in points.

- `%%exported header offset⟨offset⟩`

Set the header offset in the version of the notebook exported to PDF, measured in points.

- `%%exported footer offset⟨offset⟩`

Set the header offset in the version of the notebook exported to PDF, measured in points.

## See also:

- **Notebook configuration**
- **Customizing the notebook metadata**
- **Customizing the notebook styles**
- **Customizing the notebook line and paragraph spacing**
- **Customizing the PDF export settings**



# Customizing the notebook styles

A **style** is a set of parameters specifying the way text is formatted, i.e., which font and font weight it is set in, its color, etc. In the **notebook configuration** code you can define a set of styles to use in your notebook. Each style has a name. Certain style names are reserved to specify standard, predefined types of text (for example, **default** refers to the default style used for normal text; **header** is the style name used for headers). Other names can be used to define custom styles to suit your own particular needs.

The syntax for defining styles is as follows:

- `⌘⌘define style⟨style name; list of style commands⟩`

Defines a named style. This can be either one of the standard names for predefined styles, or a custom style name to be used in a `⌘⌘styled⟨...⟩ text styling wrapper`. The standard predefined style names are:

- **default**
- **url hyperlink**
- **text hyperlink**
- **intralink**
- **header**
- **subheader**
- **subsubheader**
- **paraheader**

- **superheader**

**Style inheritance.** When you define a style, including the **attribute inherits from** `←style name` in the command argument block specifies that the style you are defining should inherit its properties from another style you already defined. This allows endowing the set of styles with a hierarchical structure.

For example, the default notebook configuration code defines a set of styles with the following commands:

```
% Start by defining a base style all other styles
will inherit from
%%define style⟨base; %
font size⟨14⟩%
font cluster⟨Latin Modern⟩%
⟩
```

```
% Now define the default style for normal text
%%define style⟨default; %
@⟨inherits from←base⟩%
% add styling commands here if you want the
default style to differ from the base style
⟩
```

```
% Style definitions for links
%%define style⟨%
@⟨inherits from←base⟩%
link; %%bold on. %
⟩
```

```
%%define style⟨%
@⟨inherits from←link⟩%
hyperlink; %%color⟨0.1; 0; 0.65⟩%
⟩
```

```
%%define style⟨%
@⟨inherits from←hyperlink⟩%
text hyperlink; %
⟩
```

```
%%define style⟨%
@⟨inherits from←hyperlink⟩%
url hyperlink; %%font⟨Latin Modern Sans⟩%
⟩

%%define style⟨%
@⟨inherits from←link⟩%
intralink; %%color⟨0.6; 0.05; 0⟩%
⟩

% Style definitions for headers
%%define style⟨%
@⟨inherits from←base⟩%
headers base; %%bold on. %
⟩

%%define style⟨%
@⟨inherits from←headers base⟩%
superheader; %%font size⟨32⟩%%color⟨0.7; 0; 0⟩%
⟩

%%define style⟨%
@⟨inherits from←headers base⟩%
header; %%font size⟨24⟩%
⟩

%%define style⟨%
@⟨inherits from←headers base⟩%
subheader; %%font size⟨18⟩%
⟩

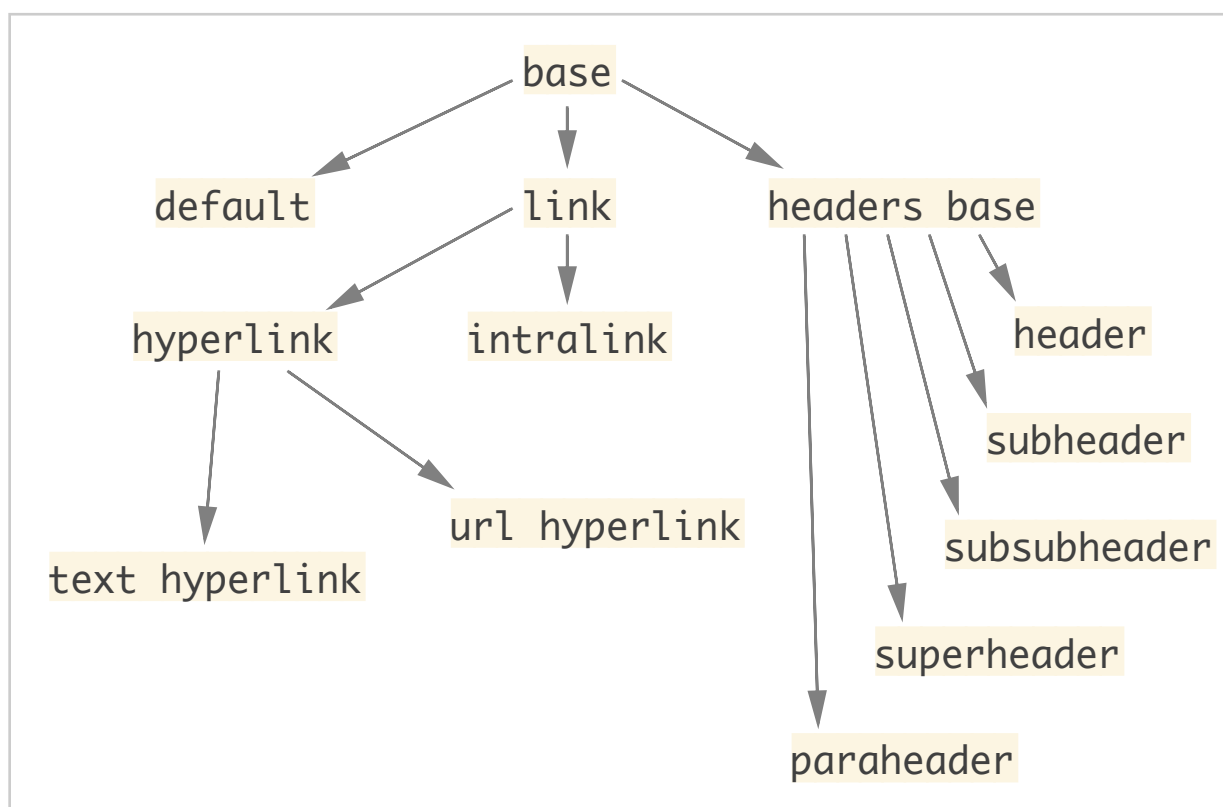
%%define style⟨%
@⟨inherits from←headers base⟩%
subsubheader; %%font size⟨14⟩%
⟩
```

```

%%define style«%
@«inherits from←headers base»%
paraheader; %%font size«14»%
»

```

This creates hierarchical inheritance relationships between the named styles illustrated in the diagram below:



Given such an inheritance structure, making a change to one style affects that style and potentially all others that inherit from it. This makes it easier to achieve a consistent look for your notebook and to quickly make changes when you want to, in such a way that the change will affect precisely the set of styles it should logically apply to.

See also:

- [Styling text](#)
- [Headers and subheaders](#)
- [Hyperlinks and intralinks](#)
- [Notebook configuration](#)
- [Customizing the notebook metadata](#)
- [Customizing the page geometry](#)
- [Customizing the notebook line and paragraph spacing](#)
- [Customizing the PDF export settings](#)

# Line and paragraph spacing


In the **notebook configuration** code you can specify the settings for line spacing and paragraph spacing.

## Line spacing

- `%%line spacing`*relative spacing*

Set the line spacing, in units of the predefined height of a line in the current text font. The default spacing value is 1.0.

## Paragraph spacing

The vertical space inserted between paragraphs is known as the **paragraph spacing**. In our terminology, “paragraph” can refer to document elements such as headers, subheaders, list items, etc. This means that customization of the paragraph spacing involves specifying not just a single number, but rather a collection of numbers that govern the spacings between different types of paragraphs according to the logical role they play in the document.  **MadHat** classifies each paragraph as being one of the following 9 logical types:

1. normal text paragraph
2. **header**
3. **subheader**
4. **subsubheader**
5. **paragraph header**

6. **superheader**
7. **list** item
8. beginning of a **box**
9. end of a **box**

To specify the paragraph spacing data, first you need to specify the **base paragraph spacing** — a reference value that all other paragraph spacing parameters use as a unit. The syntax for customizing this value is:

- `⌘⌘base paragraph spacing`*«spacing in font units»*

The default value for this parameter is 1.0, corresponding to a spacing equal to the current font size, in points.

Having defined the base paragraph spacing, you can now specify how much vertical space is inserted before and after a paragraph of each logical type. These spacing values are provided in a two-line array (entered using **delimited list notation**) according to the following syntax:

- 

```
⌘⌘paragraph before and after spacings«%
...list of pre-paragraph spacing values... # %
...list of post-paragraph spacing values...%
»
```

with each of the two lists consisting of 9 spacing values, in units of the base paragraph spacing.

**Example.** The command

```
⌘⌘paragraph before and after spacings«%
0; 0; 1.5; 0; 0; 0; 0; 0; 0 # %
0; 0; 0; 0; 0; 0; 0; 0; 2%
»
```

specifies that 1.5 spacing units are inserted before a subheader, 2 spacing units are inserted after the end of a box, and no other paragraph spacings are inserted.

## Paragraph spacing at the top of a page


It is sometimes desirable to insert a vertical space at the top of a page before certain logical types of paragraphs (for example a superheader, if you use such headers for chapter headings). You can specify these spacing values using the following commands:

- `⌘⌘top of page preparagraph spacings⟨list of spacing values⟩`

Here, *list of spacing values* is a list of 9 values, entered using **delimited list notation**, corresponding to the 9 logical paragraph types.

These spacing values are set to 0 by default.

## Setting the paragraph spacings matrix

In addition to the commands mentioned above,  **MadHat** offers a more advanced method for specifying the paragraph spacings that allows customizing the vertical space inserted between *each pair* of logical paragraph types. This is done by entering a matrix of spacing values, which we refer to the as the **paragraph spacings matrix**.

`⌘⌘paragraph spacings matrix⟨spacings matrix⟩`

Here, *kerning matrix* is a  $9 \times 9$  table of, entered using **delimited list notation**, of the form



$$\begin{pmatrix} k_{11} & k_{12} & k_{13} & k_{14} & k_{15} & k_{16} & k_{17} & k_{18} & k_{19} \\ k_{21} & k_{22} & k_{23} & k_{24} & k_{25} & k_{26} & k_{27} & k_{28} & k_{29} \\ k_{31} & k_{32} & k_{33} & k_{34} & k_{35} & k_{36} & k_{37} & k_{38} & k_{39} \\ k_{41} & k_{42} & k_{43} & k_{44} & k_{45} & k_{46} & k_{47} & k_{48} & k_{49} \\ k_{51} & k_{52} & k_{53} & k_{54} & k_{55} & k_{56} & k_{57} & k_{58} & k_{59} \\ k_{61} & k_{62} & k_{63} & k_{64} & k_{65} & k_{66} & k_{67} & k_{68} & k_{69} \\ k_{71} & k_{72} & k_{73} & k_{74} & k_{75} & k_{76} & k_{77} & k_{78} & k_{79} \\ k_{81} & k_{82} & k_{83} & k_{84} & k_{85} & k_{86} & k_{87} & k_{88} & k_{89} \\ k_{91} & k_{92} & k_{93} & k_{94} & k_{95} & k_{96} & k_{97} & k_{98} & k_{99} \end{pmatrix}$$

Each number  $k_{ij}$  in row  $i$  and column  $j$  of the table, for  $i, j = 1, 2, \dots, 9$ , specifies the vertical spacing that will be inserted after a paragraph of logical type  $i$  that is followed by a paragraph of logical type  $j$  in the list of paragraph types above.

Note that it is only paragraphs with visible content that have an effect on paragraph spacings. Code paragraphs that do not produce any visible content, for example a paragraph containing only a code comment

```
%a comment
```

or a paragraph containing only a

```
⌘begin list.
```

command, will not have any effect on the paragraph spacings.

## Setting the paragraph indent

- `⌘⌘paragraph indent`⟨*paragraph indent width in font units*⟩

Set the paragraph indent width, measured in units of the current

font size.

## See also:

- **Notebook configuration**
- **Customizing the notebook metadata**
- **Customizing the page geometry**
- **Customizing the notebook styles**
- **Customizing the PDF export settings**

# Customizing the notebook PDF export settings

Using the commands listed below, you can customize the way in which the notebook is formatted when it is exported to PDF.

- `%%exported page header⟨page header⟩`

Set the page header for the version of the notebook exported as a PDF

- `%%exported page footer⟨page footer⟩`

Set the page footer for the version of the notebook exported as a PDF

- `%%exported page number.`

When inserted inside the argument block for either the `%%exported page footer⟨...⟩` command or the `%%exported page header⟨...⟩` command, this will display as the page number on each page in the PDF-exported notebook.

- The headers and footers specified in the `%%exported page header⟨...⟩` and `%%exported page footer⟨...⟩` commands are by default added to all the pages in the exported PDF document. To add the header and footer only for a specified range of notebook pages, use the command


```
%%exported header and footer range⟨start index;  
end index⟩
```

where *start index* and *end index* are the indices of the first and last notebook pages, respectively, to which the header and footer should be added. If *end index* is left unspecified (that is, the argument block for it is an empty string), it is taken as the index of the last page in the notebook.

## See also:


- [Notebook configuration](#)
- [Customizing the notebook metadata](#)
- [Customizing the page geometry](#)
- [Customizing the notebook styles](#)
- [Customizing the notebook line and paragraph spacing](#)

# Paragraphs

The word “paragraph” traditionally refers to a logically connected block of content in a document. Here, we wish to make a distinction between **code paragraphs**, which are contiguous blocks of  **MadHat** code in the editor window that are processed by the app in a particular way; and **semantic paragraphs**, which are logical blocks of content in the formatted notebook page, and correspond more closely to the traditional notion of a paragraph.



For notebook content consisting only of text, code paragraphs are in a one-to-one correspondence with semantic paragraphs, so the distinction between the two is not very important. However, for reasons that will be explained below, when your notebook content contains math display, the association between code paragraphs and semantic paragraphs will in general be many-to-one, so it is important to discuss both notions of a paragraph and how one gets mapped into the other.


## Code paragraphs

When you enter your code in the editor window,  **MadHat** divides it as you type into **code paragraphs**, with each code paragraph being marked with a small marker to the left of the code. A code paragraph is terminated with two or more newline characters, and the next paragraph starts at the first non-newline character following those newline characters. The successive newline characters separating code paragraphs are not considered as belonging to any paragraphs.

(By contrast, a single newline character inside a code paragraph will not terminate it but will be considered as part of the content of that paragraph.)


## The logic behind code paragraphs

The division of code into paragraphs forms a crucial part of  MadHat's parsing and typesetting algorithms, in the following way: each paragraph of code is sent to the  MadHat parser for processing that results in a block of content being typeset in the notebook page viewer. This is done for all paragraphs in a page when you open an existing notebook, or incrementally as you type or edit code: for each code editing operation, only the paragraph in which the edit is made gets re-processed, saving the computational work of re-processing the entire page's code.


(To be even more precise: some edits will affect more than one code paragraph, for example causing two existing paragraphs to coalesce, or splitting a single paragraph into two, or changing the text typing style in a way that propagates to the next paragraphs on your page;  MadHat's algorithm handles that appropriately by processing the minimal number of paragraphs to ensure the correct formatting of your page content.)

## Text paragraphs and math paragraphs

Code paragraphs are classified as belonging to one of two types: **text paragraphs**, and **math paragraphs**. Each of the types is parsed according to different rules and typeset using a different algorithm.

Code paragraphs are parsed as text paragraphs by default. Paragraphs beginning with the paragraph math mode shift prefix  (see also: **special symbols**) are parsed as math paragraphs, and are formatted as **math displays** (see also: **typing mathematics**).

## Attributes of code paragraphs

A code paragraph is considered by the  MadHat parser as a type of **block**, even though it is not surrounded by an open block/close block symbol pair. In particular, you can include in a code paragraph an **attributes**

**block** to modify its behavior in the same way that an attributes block can modify the behavior of a command argument block.

Currently there is only one supported attribute, for specifying the paragraph text alignment type:

- `align` ← *alignment type*

Specifies the paragraph alignment type: `left` (the default for text paragraphs), `right`, or `center` (the default for math paragraphs).

## Semantic paragraphs

What we call a semantic paragraph corresponds to the traditional notion of a paragraph in a written document: that is, a block of content that occupies its own vertical space on the page and is visually separated from the surrounding content (using paragraph indentation and/or paragraph spacing).

For a page containing only text, each code paragraph will get formatted as a single semantic paragraph. However, when mathematical content is included, the association between code paragraphs and semantic paragraphs becomes more complicated. The reason is that according to the rules of mathematical writing, mathematical displays — which are the formatted result of a math (code) paragraph — are considered a part of the semantic paragraph of the text that precedes them, and sometimes (but not always) of the text that follows them. For example, the following content constitutes a single semantic paragraph:

A power series is a function of a complex variable  $z$  that is defined by

$$f(z) = \sum_{n=0}^{\infty} a_n z^n,$$

where  $(a_n)_{n=0}^{\infty}$  is a sequence of complex numbers, or more generally by

$$g(z) = f(z - z_0) = \sum_{n=0}^{\infty} a_n (z - z_0)^n,$$

where  $(a_n)_{n=0}^{\infty}$  is again a sequence and  $z_0$  is some fixed complex number. These functions are defined whenever the respective series converges.

The code that produces this semantic paragraph is made up of five code paragraphs:

A power series is a function of a complex variable  $\hat{M}\langle z \rangle$  that is defined by

$\hat{M}$ :  $\hat{M}$ :  $f(z) = \text{sum\_}\langle n=0 \rangle^{\langle \text{inf} \text{ty} \rangle} a_n z^n,$

where  $\hat{M}\langle (a_n)_{n=0}^{\langle \text{inf} \text{ty} \rangle} \rangle$  is a sequence of complex numbers, or more generally by

$\hat{M}$ :  $\hat{M}$ :  $g(z) = f(z - z_0) = \text{sum\_}\langle n=0 \rangle^{\langle \text{inf} \text{ty} \rangle} a_n (z - z_0)^n,$


where  $\hat{M}\langle (a_n)_{n=0}^{\langle \text{inf} \text{ty} \rangle} \rangle$  is again a sequence and  $\hat{M}\langle z_0 \rangle$  is some fixed complex number. These functions are defined whenever the respective series converges.

This example raises the question of how to control where semantic paragraphs begin and end — an issue that will affect where vertical spacing between paragraphs is inserted, and other subtle factors that




affect the look and readability of your content. We address this question next.

## Controlling the boundaries of a semantic paragraph

 **MadHat** tries to guess where you wish for a semantic paragraph to end, by following common sense rules that apply in the majority of cases, so that most of the time you do not need to do anything special other than writing out your content. The rules are as follows:

- any math paragraph is considered by default to be a part of the same semantic paragraph as the code paragraph preceding it (if there is one);
- any text paragraph is by default marked as starting a new semantic paragraph if it follows a text paragraph (or if it is the first paragraph on the page), but marked as belonging to the same semantic paragraph as the preceding code paragraph if that preceding code paragraph is a math paragraph.

One common situation in which the above default assumptions do not hold is when a math display comes at the *end* of a semantic paragraph. In that case, the text paragraph that follows it should start a new semantic paragraph. To indicate to  **MadHat** that that is what you are intending, add the command

```
⌘new paragraph.
```

at the beginning of the text paragraph that follows the math display.

## Paragraph indentation

 **MadHat** has the ability to add an automatic indentation at the beginning

of a new semantic paragraph. The indentation width is set to 0 by default, but can be customized in the **notebook configuration code**; see the help page on **customizing line and paragraph spacings and paragraph indents**

When automatic paragraph indentation is enabled, you can suppress the paragraph indentation in any individual text paragraph by including the command ,

```
⌘suppress paragraph indent.
```

or its alias

```
⌘no indent.
```

at the beginning of the paragraph.

## Multi-justified lines

It is sometimes useful to have a paragraph consisting of a single line split into two parts, the first of which is justified to the left, and the second of which is justified to the right; or to have a paragraph with a single line with *three* parts, justified to the left, center and right, respectively. We call such paragraph-like constructs **multi-justified lines**. They can be entered using the following commands:

- `⌘left right line`«*left side text*; *right side text*»

A line of text with two parts, the first being left-justified and the second being right-justified.

- `⌘left center right line`«*left side text*; *center text*; *right side text*»

A line of text with three parts, the first being left-justified, the second being centered, and the third being right-justified.

The commands `\left right line⟨...⟩` and `\left center right line⟨...⟩` both accept the boolean attributes `overline` and `underline`.

# Styling text

 MadHat offers two main mechanisms for styling your text.

- Using styling commands that directly affect the styling attributes of your text. These commands are described in the following help pages:
  - **Bold text formatting**
  - **Italic text formatting**
  - **Underlining**
  - **Strikethrough**
  - **Highlighting**
  - **Text substitutions**
  - **Setting fonts**
  - **Setting the font size**
  - **Colors**
- You can also define custom named styles in the **notebook configuration code** using the `⌘⌘define style⟨...; ...⟩` configuration command. You can then apply the style to arbitrary blocks of text using the command

```
⌘styled⟨style name; content to style⟩
```

## Styling text in boldface

The following commands are used to style text in boldface:

- `⌘bold on.`

Turns on bold styling.

- `⌘bold off.`

Turns off bold styling.

- `⌘bold text⟨...text...⟩`  
`⌘bold⟨...text...⟩`

A wrapper for text that should be styled in boldface.

These commands do not affect the styling of mathematical symbols. See the help page on **mathematical font variants**.

## Examples

The code

```
@⟨align←center⟩⌘bold⟨This text is styled in bold.⟩
```

will typeset as

**This text is styled in boldface.**

## See also

- **Mathematical font variants**

## Styling text in italic

The following commands are used to style text in italic:

- `⌘italic on.`

Turns on italic styling.

- `⌘italic off.`

Turns off italic styling.

- `⌘italic text⟨...text...⟩`  
`⌘italic⟨...text...⟩`

A wrapper for text that should be styled in italic.

These commands do not affect the styling of mathematical symbols. See the help page on **mathematical font variants**.

## Examples

The code

```
@⟨align←center⟩⌘italic⟨This text is styled in  
italic.⟩
```

will typeset as

*This text is styled in italic.*

See also

- 
- **Mathematical font variants**

# Underlining text

The following commands are used to underline text:

- `⌘underline on.`

Turns on text underlining.

- `⌘underline off.`

Turns off text underlining.

- `⌘underline⟨...text...⟩`

A wrapper for text that should be underlined.

## Examples

The code

```
@⟨align←center⟩⌘underline⟨This text is  
underlined.⟩
```

will typeset as

This text is underlined.

## See also

- **Mathematical symbol decorations**



# Striking through text

The following commands are used to strike through text:

- `⌘strikethrough on.`

Turns on text strikethrough.

- `⌘strikethrough off.`

Turns off text strikethrough.

- `⌘strikethrough⟨...text...⟩`

A wrapper for text that should be struck through.

## Examples

The code

```
@⟨align←center⟩⌘strikethrough⟨This text is struck through.⟩
```

will typeset as

~~This text is struck through.~~

# Highlighting text

The following commands are used to highlight text:

- `⌘highlight on.`

Turns on highlighted text styling.

- `⌘highlight off.`

Turns off highlighted text styling.

- `⌘highlight⟨...text...⟩`

A wrapper for text that should be highlighted.

You can highlight text in different colors using the `⌘highlight color⟨...⟩` command. See the help page on **colors**.

## Examples

The code

```
@⟨align←center⟩⌘highlight⟨This text is
highlighted.⟩
```

will typeset as

This text is highlighted.

The code

```
@⟨align←center⟩⟨⌘highlight color⟨pink⟩
⌘highlight⟨This text is highlighted in a different
color.⟩⟩
```

will typeset as

This text is highlighted in a different color.

## See also

- **Colors**

## Text substitutions

Text substitutions are operations that transform the text you enter by examining each character of the specified text and potentially replacing it with a different character. The following substitutions are available:

- `⌘lowercase⟨text⟩`

A wrapper for text that should be converted to lowercase

- `⌘uppercase⟨text⟩`

A wrapper for text that should be converted to uppercase

- `⌘redact⟨text⟩`

A wrapper for text that should be presented in redacted form

- `⌘obfuscate⟨text⟩`

A wrapper for text that should be presented in (mildly) obfuscated form

Text substitutions are only applied to content entered in text mode. Mathematical symbols are unchanged.

## Examples

- Original text:

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, “and what is the use of a book,” thought Alice

“without pictures or conversations?”

- Applying a `⌘lowercase⟨...⟩` substitution:

alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, “and what is the use of a book,” thought alice “without pictures or conversations?”

- Applying a `⌘uppercase⟨...⟩` substitution:

ALICE WAS BEGINNING TO GET VERY TIRED OF SITTING BY HER SISTER ON THE BANK, AND OF HAVING NOTHING TO DO: ONCE OR TWICE SHE HAD PEEPED INTO THE BOOK HER SISTER WAS READING, BUT IT HAD NO PICTURES OR CONVERSATIONS IN IT, “AND WHAT IS THE USE OF A BOOK,” THOUGHT ALICE “WITHOUT PICTURES OR CONVERSATIONS?”


- Applying a `⌘redact⟨...⟩` substitution:

```
***** *** ***** ** *** ***** ***** ** ***** ** *** *****
** *** ***** *** ** ***** ***** ** *** ***** ** ***** *** ***
***** ***** *** ***** *** ***** ***** ***** ***** ** ** ** **
***** ** ***** ***** ** *** ***** ***** ** *** ** ** *
***** ***** ***** ***** ***** ***** ** **********
```

- Applying a `⌘obfuscate⟨...⟩` substitution:

Bmjdf xbt cfhjoojoh up hfu wfsz ujsfe pg tjuujoh cz ifs tjtufts po uif cbol, boe pg ibwjoh opuijoh up ep: podf ps uxjdf tif ibe qffqfe joup uif cppl ifs tjtufts xbt sfbejoh, cvu ju ibe op qjduvsft ps dpowfstbujpot jo ju, “boe xibu jt uif vtf pg b cppl,” uipvhiu Bmjdf “xjuipvu qjduvsft ps dpowfstbujpot?”


# Setting fonts

 **MadHat** typesets text and mathematical expressions using two different fonts: the **text font** and **math font**. You can configure each of those fonts separately, or, for certain pre-configured pairs of matching fonts, set both of them together in a single command. The syntax for font-setting commands is as follows.

- **⌘font**⟨font name⟩

Set the text font. The default is Latin Modern Roman.

- **⌘math font**⟨font name⟩

Set the math font. This should be a font that supports the **Unicode mathematical symbols** that you wish to use in your mathematical expressions.  **MadHat** comes prepackaged with five mathematical fonts:

- Latin Modern Math (the default)
- TeX Gyre Termes Math
- TeX Gyre Bonum Math
- TeX Gyre Schola Math
- TeX Gyre Pagella Math.

These are free and open source fonts developed by the **GUST e-foundry**. Each of the five math fonts is also accompanied by a family of matching ordinary text fonts (for example, Latin Modern Math is accompanied by Latin Modern Roman, Latin Modern Sans, Latin Modern Sans, and several other fonts in the Latin Modern

family).

- `⌘font cluster⟨font family name⟩`.

Set the text and math fonts simultaneously to matching fonts belonging to one of the five font families bundled with the app.

- `⌘font cluster⟨latin modern⟩`

Use the Latin Modern Roman and Latin Modern Math fonts

- `⌘font cluster⟨termes⟩`

Use the TeX Gyre Termes and TeX Gyre Termes Math fonts

- `⌘font cluster⟨bonum⟩`

Use the TeX Gyre Bonum and TeX Gyre Bonum Math fonts

- `⌘font cluster⟨schola⟩`

Use the TeX Gyre Schola and TeX Gyre Schola Math fonts

- `⌘font cluster⟨pagella⟩`

Use the TeX Gyre Pagella and TeX Gyre Pagella Math fonts

## Examples

- `⌘font⟨Helvetica⟩`Some text in Helvetica

Some text in Helvetica.

- `⌘font⟨Courier⟩`Some text in Courier

Some text in Courier

- `⌘font⟨Georgia⟩`Some text in Georgia

Some text in Georgia

- `⌘font cluster⟨latinmodern⟩`Some text and some math:  
 $\hat{M}\langle\text{sum\_}\langle n=1\rangle^{\langle\text{infty}\rangle}\frac{\langle 1; n^2\rangle}{\langle 6\rangle} = \frac{\langle \pi^2; 6\rangle}{\langle 6\rangle}$ ,  
 set in the Latin Modern font cluster

Some text and some math:  $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$ , set in the Latin Modern font cluster

- `⌘font cluster⟨termes⟩`Some text and some math:  
 $\hat{M}\langle\text{sum\_}\langle n=1\rangle^{\langle\text{infty}\rangle}\frac{\langle 1; n^2\rangle}{\langle 6\rangle} = \frac{\langle \pi^2; 6\rangle}{\langle 6\rangle}$ ,  
 set in the Termes font cluster

Some text and some math:  $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$ , set in the Termes font cluster

- `⌘font cluster⟨bonum⟩`Some text and some math:  
 $\hat{M}\langle\text{sum\_}\langle n=1\rangle^{\langle\text{infty}\rangle}\frac{\langle 1; n^2\rangle}{\langle 6\rangle} = \frac{\langle \pi^2; 6\rangle}{\langle 6\rangle}$ ,  
 set in the Bonum font cluster

Some text and some math:  $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$ , set in the Bonum font cluster

- `⌘font cluster⟨schola⟩`Some text and some math:  
 $\hat{M}\langle\text{sum\_}\langle n=1\rangle^{\langle\text{infty}\rangle}\frac{\langle 1; n^2\rangle}{\langle 6\rangle} = \frac{\langle \pi^2; 6\rangle}{\langle 6\rangle}$ ,  
 set in the Schola font cluster

Some text and some math:  $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$ , set in the Schola font cluster

- `⌘font cluster⟨pagella⟩`Some text and some math:  
 $\hat{M}\langle\text{sum\_}\langle n=1\rangle^{\langle\text{infty}\rangle}\frac{\langle 1; n^2\rangle}{\langle 6\rangle} = \frac{\langle \pi^2; 6\rangle}{\langle 6\rangle}$ ,  
 set in the Pagella font cluster

Some text and some math:  $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$ , set in the Pagella font cluster



## Setting the font size

- `⌘font size⟨font size in points⟩`

Set the base font size for both text and mathematical expressions

### Examples

```
⌘font size⟨12⟩The quick brown fox jumps over a  
lazy dog
```

The quick brown fox jumps over a lazy dog

```
⌘font size⟨20⟩The quick brown fox jumps over a  
lazy dog
```

The quick brown fox jumps over a lazy dog

# Colors

## Setting colors

- `⌘color⟨color argument⟩`  
`⌘colour⟨color argument⟩`

Set the primary (text) color

- `⌘highlight color⟨color argument⟩`  
`⌘highlight colour⟨color argument⟩`

Set the text highlight color

- `⌘box frame color⟨color argument⟩`  
`⌘box frame colour⟨color argument⟩`

Set the frame color for boxes

- `⌘box background color⟨color argument⟩`  
`⌘box background colour⟨color argument⟩`

Set the background color for boxes

- `⌘page background color⟨color argument⟩`  
`⌘page background colour⟨color argument⟩`

Set the background color for the page

- `⌘fill color⟨color argument⟩`  
`⌘fill colour⟨color argument⟩`

Set the fill color for filled graphics shape

- `⌘stroke color⟨color argument⟩`  
`⌘stroke colour⟨color argument⟩`

Set the stroke color for stroked graphics shapes

## Format for a color

Each of the commands above takes a color argument, whose format is specified as either:

- the name of a named color such as “red”, “gray”, “orange”, “teal”, etc.;
- a block of the form  $\langle\text{red}; \text{green}; \text{blue}\rangle$  where each of the three delimited subblocks is a floating point value between 0 and 1, representing the red, green and blue components, respectively;
- a block of the form  $\langle\text{red}; \text{green}; \text{blue}; \text{alpha}\rangle$  where each of the four delimited subblocks is a floating point value between 0 and 1, representing the red, green, blue and alpha components, respectively.

## Available named colors

blue		white	
red		gray	
orange		dark grey	
purple		black	
green		dark gray	
magenta		yellow	
cyan		pink	
brown		light gray	
light grey		teal	
grey			

## Examples

- Some text in blue: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
- Some text in orange: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
- Some text highlighted in pink: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

# Headers and subheaders

Headers can be used to give your document a hierarchical structure with several levels. The default level is called “header”; below it are “subheader”, “subsubheader”, and “paragraph header”. Above the default level is one level called “superheader”, which can be used in very long documents (e.g., as chapter headings in a book) but is otherwise unnecessary.

Headers at the header, subheader and subsubheader levels can be clicked to collapse and expand the contents of the section they demarcate.

The commands for headers are:

- `⌘header⟨...header text...⟩`

A header

- `⌘subheader⟨...subheader text...⟩`

A subheader

- `⌘subsubheader⟨...subsubheader text...⟩`

A subsubheader

- `⌘paragraph header⟨...paragraph header text...⟩`  
`⌘paraheader⟨...paragraph header text...⟩`

A paragraph header

- `⌘superheader⟨...superheader text...⟩`

A superheader

The commands for all header types except for paragraph headers should appear on their own paragraph with no additional content.

## Customizing header styles

The help page on **notebook configuration** explains how to customize the appearance of headers.

### Examples

**A superheader**


**A header**

**A subheader**

**A subsubheader**

**A paragraph header.** Some text in a paragraph with a header.

# Lists and outlining

 MadHat supports ordered and unordered lists, including nested lists. Lists also function as **outliner** structures that can be dynamically collapsed and expanded.

The syntax for lists and list items is as follows:

- `⌘begin list.`

Start a new list or sublist. This command should appear on its own paragraph with no additional content.

- `⌘end list.`

Close the current list or sublist. This command should appear on its own paragraph with no additional content.

- `*`

Start a new unordered list item. The code `*` (an asterisk followed by a space) must be at the beginning of a code paragraph in order to be interpreted as a list item command.

- `*.`

Start a new ordered list item. The code `*.` (an asterisk followed by a period followed by a space) must be at the beginning of a code paragraph in order to be interpreted as a list item command.

## Lists as outliners

When you click on the marker of a list item (e.g., a bullet, or the number marking an item in an ordered list), the content of the item will toggle

between the collapsed and expanded state. The default behavior for collapsing is to hide all the content below the opening paragraph of the item, up to the beginning of the next list item or to the closing of the current list or sublist. This behavior can be modified by adding a `⌘collapsehere.` command somewhere in the opening paragraph of the item.

- `⌘collapse here.`

Marks the point in the content of a list item beyond which the item's content will collapse when the item is clicked.

## Example

Here is an example of a hierarchical list. Try clicking on the item markers to experiment with the outliner feature.

There are many types of birds in Australia, including:

1. nocturnal birds
  - 1.1. frogmouths

Frogmouths are named for their large flattened hooked bill and huge frog-like gape, which they use to capture insects. Their flight is weak. They rest horizontally on branches during the day, camouflaged by their cryptic plumage. Up to three white eggs are laid in the fork of a branch, and are incubated by the female at night and the male in the day. The three *Podargus* species are large frogmouths restricted to Australia and New Guinea, that have massive flat broad bills.

- 1.2. nightjars

Nightjars are medium-sized nocturnal or crepuscular birds in



the family Caprimulgidae / and order Caprimulgiformes, characterised by long wings, short legs, and very short bills. They are sometimes called goatsuckers, due to the ancient folk tale that they sucked the milk from goats (the Latin for goatsucker is caprimulgus), or bug eaters,[1] their primary source of food being insects. Some New World species are called nighthawks. The English word “nightjar” originally referred to the European nightjar.

Nightjars are found all around the world, with the exception of Antarctica and certain island groups such as Madagascar and the Seychelles. They can be found in a variety of habitats, most commonly the open country with some vegetation. They usually nest on the ground, with a habit of resting and roosting on roads.

### 1.3. owls

Owls are birds from the order Strigiformes (/ which includes over 200 species of mostly solitary and nocturnal birds of prey typified by an upright stance, a large, broad head, binocular vision, binaural hearing, sharp talons, and feathers adapted for silent flight. Exceptions include the diurnal northern hawk-owl and the gregarious burrowing owl.

## 2. marsh birds

### 2.1. crakes

### 2.2. grebes

### 2.3. snipes

# Tables

To format a table, use one of the commands below. Tabular content is entered using primary and secondary **list delimiter notation** to indicate column and row boundaries. An optional **attributes** block can be provided to modify the appearance of the table.

- `⌘table`«*table cells*»

A table of left-justified (by default; this is modifiable) text cells

- `⌘math table`«*table cells*»

A table of centered (by default; this is modifiable) cells that is vertically centered with respect to the math axis.

See also: **Matrices**

## Attributes to modify the appearance of a table

- `hlines`«*lines specification*»

A specification of horizontal cell border lines.

In a table with  $k$  rows, the specification should be a list of  $k + 1$  **Boolean attribute specifiers** (`y` or `n`) to specify which horizontal lines are drawn, from top to bottom.

For example, in a table with 4 rows, providing the specification `hlines`«*yynny*» would format as follows:

First name	Last name	Student ID	GPA
Mickey	Mouse	1	3.5
Donald	Duck	2	3
Minnie	Mouse	3	3.8

- `vlines` ← *lines specification*

A specification of vertical cell border lines.

The specification follows the same format as the specification for horizontal lines, except that there should be  $n + 1$  **Boolean attribute specifiers**, where  $n$  is the number of columns in the table

- `alignments` ← *cell alignments specification*

A specification of the horizontal cell alignments.

The specification should be a list `xxx...x` of  $n$  characters, each specifying the alignment for one of the table columns. The allowed values are ‘l’ (for “left”), ‘r’ (for “right”), and ‘c’ (for “center”).

- `header rows` ← *header rows specification*

A specification of the header rows: a list of  $k$  **Boolean attribute specifiers**, where  $k$  is the number of rows in the table, specifying which of the rows are header rows

- `header columns` ← *header columns specification*

A specification of the header columns: a list of  $n$  **Boolean attribute specifiers**, where  $n$  is the number of columns in the table, specifying which of the columns are header columns

- `frame, no frame`

A Boolean specifying whether a frame is drawn around the table. Defaults to **no**.

- **Specifying color fills for table cells**

- **fill, no fill**

A boolean value specifying whether uniform fill (a colored background for all table cells) is enabled. Defaults to **no**.

- **alternating fill, no alternating fill**

A boolean value specifying whether alternating row fill (a colored background for all table cells in alternating rows) is enabled. Defaults to **no**

- **fill color** ← *color argument*

The fill color for uniform fills

- **alt fill color** ← *color argument*

The alternating fill color, to be used for the alternating row fill style

- **header fill color** ← *color argument*

The fill color to use for header rows and columns

- **header alt fill color** ← *color argument*

The alternating fill color to use for header rows and columns when alternating row fills are specified

See the **colors** help page for an explanation of how to specify colors.

## Examples

- The code

```
@«align←center»%
⌘table«%
@«hlines←yynny; alignments←llcr»%
First name; Last name; Student ID; GPA # %
Mickey; Mouse; 1; 3.5 # %
Donald; Duck; 2; 3 # %
Minnie; Mouse; 3; 3.8%
»
```

typesets as

First name	Last name	Student ID	GPA
Mickey	Mouse	1	3.5
Donald	Duck	2	3
Minnie	Mouse	3	3.8

- A more complicated table is described by the code

```
@«align←center»%
⌘table«%
@«hlines←nynnnny; vlines←nyny;
alignments←rcccl; frame»%
⌘bold«Child»; ⌘bold«Birthday month»;
⌘bold«Favorite fruit»; ⌘color«0.4; 0; 0»
⌘bold«Nut allergy» # %
Kim; April; Banana; yes # %
Bobby; September; Orange; yes # %
Michael; January; Apple; no # %
Samantha; January; ⌘bold«Kiwi»; yes # %
Ray; March; Banana; ⌘highlight«not sure» # %
Dina; December; Banana; no # %
Andy; February; Grapefruit; no # %
Lily; March; Melon; no # %
William; July; Banana; yes%
»
```

This typesets as

Child	Birthday month	Favorite fruit	Nut allergy
Kim	April	Banana	yes
Bobby	September	Orange	yes
Michael	January	Apple	no
Samantha	January	<b>Kiwi</b>	yes
Ray	March	Banana	not sure
Dina	December	Banana	no
Andy	February	Grapefruit	no
Lily	March	Melon	no
William	July	Banana	yes

- Here is code for the same table, with some colors added:

```

@⟨align←center⟩%
⌘table⟨%
@⟨hlines←nynnnny; vlines←nyny;
alignments←rcccl; frame; alternating fill; alt
fill color←⟨.5; 0.1; 0.3; 0.15⟩; fill
color←⟨0.85; 0.8; 0.9; 0.3⟩; %
header alt fill color←⟨0.85; 0.85; 0.85; 1⟩; %
header fill color←⟨0.92; 0.92; 0.92; 1⟩; %
header columns←ynny; header rows←ynnnnnn;
header color←⟨0.8; 0.8; 0.8⟩%
⌘bold⟨Child⟩; ⌘bold⟨Birthday month⟩;
⌘bold⟨Favorite fruit⟩; ⌘color⟨0.4; 0; 0⟩
⌘bold⟨Nut allergy⟩# %
Kim; April; Banana; yes# %
Bobby; September; Orange; yes# %
Michael; January; Apple; no# %
Samantha; January; ⌘bold⟨Kiwi⟩; yes# %
Ray; March; Banana; ⌘highlight⟨not sure⟩# %
Dina; December; Banana; no# %
Andy; February; Grapefruit; no# %
Lily; March; Melon; no# %
William; July; Banana; yes# %

```

This typesets as:

Child	Birthday month	Favorite fruit	Nut allergy
Kim	April	Banana	yes
Bobby	September	Orange	yes
Michael	January	Apple	no
Samantha	January	<b>Kiwi</b>	yes
Ray	March	Banana	not sure
Dina	December	Banana	no
Andy	February	Grapefruit	no
Lily	March	Melon	no
William	July	Banana	yes

# Boxes

The content of one or more paragraphs can be framed inside a **box**, optionally filled with a background of a specified color.

Boxes can also have divider lines separating different paragraphs. The background color can be changed partway through the box paragraphs.

- `⌘begin box.`

Start a new box. This command should appear on its own paragraph with no additional content.

- `⌘end box.`

Close the current box. This command should appear on its own paragraph with no additional content.

- `⌘box divider.`

Insert a box divider line. This command should appear on its own paragraph with no additional content.

- `⌘box frame thickness`*⟨thickness⟩*

Set the frame thickness for boxes

- `⌘box frame color`*⟨color argument⟩*  
`⌘box frame colour`*⟨color argument⟩*

Set the frame color for boxes (see the **colors** help page)

- `⌘box background color`*⟨color argument⟩*  
`⌘box background colour`*⟨color argument⟩*

Set the background color boxes (see the **colors** help page)



## Examples

The code

```
⌘box frame color⟨black⟩%  
⌘begin box.
```

A simple box

There was a table set out under a tree in front of the house, and the March Hare and the Hatter were having tea at it: a Dormouse was sitting between them, fast asleep, and the other two were using it as a cushion, resting their elbows on it, and talking over its head. “Very uncomfortable for the Dormouse,” thought Alice; “only, as it’s asleep, I suppose it doesn’t mind.”

The table was a large one, but the three were all crowded together at one corner of it: “No room! No room!” they cried out when they saw Alice coming. “There’s plenty of room!” said Alice indignantly, and she sat down in a large arm-chair at one end of the table.

```
⌘end box.
```

produces

A simple box
--------------

There was a table set out under a tree in front of the house, and the March Hare and the Hatter were having tea at it: a Dormouse was sitting between them, fast asleep, and the other two were using it as a cushion, resting their elbows on it, and talking over its head. “Very uncomfortable for the Dormouse,” thought Alice; “only, as it’s asleep, I suppose it doesn’t mind.”

The table was a large one, but the three were all crowded together at one corner of it: “No room! No room!” they cried out when they saw Alice coming. “There’s plenty of room!” said Alice indignantly, and she sat down in a large arm-chair at one end of the table.

The code

```
⌘box frame color⟨black⟩⌘box background  
color⟨0.89; 0.95; 0.8⟩
```

```
⌘begin box.
```

A box with several sections

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, “and what is the use of a book,” thought Alice “without pictures or conversations?”

```
⌘box divider.
```

So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

⌘box divider.

⌘box background color⟨light gray⟩%

There was nothing so very remarkable in that; nor did Alice think it so very much out of the way to hear the Rabbit say to itself, “Oh dear! Oh dear! I shall be late!” (when she thought it over afterwards, it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural); but when the Rabbit actually took a watch out of its waistcoat-pocket, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.

⌘end box.

typesets as

A box with several sections

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, “and what is the use of a book,” thought Alice “without pictures or conversations?”

So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so very remarkable in that; nor did Alice think it so very much out of the way to hear the Rabbit say to itself, “Oh dear! Oh dear! I shall be late!” (when she thought it over afterwards, it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural); but when the Rabbit actually took a watch out of its waistcoat-pocket, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.

The code

```
⌘begin box.
```

```
⌘box background color⟨0.95; 0.9; 0.9⟩%
```

```
⌘paraheader⟨Theorem (Euclid).⟩ There are  
infinitely many prime numbers.
```

```
⌘box background color⟨white⟩%
```

```
⌘paraheader⟨Proof.⟩ Let  $\hat{M}_{\langle n \rangle \geq 1}$ , and let  $\hat{M}_{\langle p_1,$   
 $p_2, \dots, p_k \rangle}$  be the prime numbers in  $\hat{M}_{\langle [1, n] \rangle}$ .  
Let  $\hat{M}_{\langle N = n! + 1 \rangle}$ , and note that  $\hat{M}_{\langle N \rangle}$  is relatively  
prime to  $\hat{M}_{\langle 2, \dots, n \rangle}$ . Thus if  $\hat{M}_{\langle p \rangle}$  is a prime factor  
of  $\hat{M}_{\langle N \rangle}$  then, since  $\hat{M}_{\langle N \rangle}$  is relatively prime to  
 $\hat{M}_{\langle 2 \rangle}$ ,  $\hat{M}_{\langle p \rangle}$  cannot be  $\hat{M}_{\langle 2 \rangle}$ ; since  $\hat{M}_{\langle p \rangle}$  is relatively  
prime to  $\hat{M}_{\langle 3 \rangle}$ ,  $\hat{M}_{\langle p \rangle}$  cannot be  $\hat{M}_{\langle 3 \rangle}$ , etc. In general,  
 $\hat{M}_{\langle p \rangle}$  cannot be any of the numbers  $\hat{M}_{\langle p_1, \dots, p_k \rangle}$ .  
Therefore we have  $\hat{M}_{\langle n \rangle} < n$ . We have shown that for
```

⌘end box.

typesets as

**Theorem (Euclid).** There are infinitely many prime numbers.

**Proof.** Let  $n \geq 1$ , and let  $p_1, p_2, \dots, p_k$  be the prime numbers in  $[1, n]$ . Let  $N = n! + 1$ , and note that  $N$  is relatively prime to  $2, \dots, n$ . Thus if  $p$  is a prime factor of  $N$  then, since  $N$  is relatively prime to  $2$ ,  $p$  cannot be  $2$ ; since  $p$  is relatively prime to  $3$ ,  $p$  cannot be  $3$ , etc. In general,  $p$  cannot be any of the numbers  $p_1, \dots, p_k$ .

Therefore we have  $p > n$ . We have shown that for any  $n \geq 1$  there is a prime  $p$  bigger than  $n$ . Thus, there are infinitely many prime numbers.

See also

- **Colors**

# Links

With link commands, you can include hyperlinks to an external URL (for example a web page), and intralinks, which link to other pages in the notebook.

- `⌘hyperlink⟨URL⟩`

A hyperlink with the link URL string serving also as the link text

- `⌘hyperlink⟨URL; link text⟩`

A hyperlink with a standard link text that links to a URL

- `⌘intraLink⟨page name⟩`

An intralink to another notebook page, with the page name serving as the link text

- `⌘intraLink⟨page name; link text⟩`

An intralink to another notebook page, with arbitrary link text

Including the **attribute** `slide←slide number` in an `⌘intraLink⟨...⟩` command will direct the link to the specified **slide** number on the destination page.

## Customizing link styles

The help page on **notebook configuration** explains how to customize the appearance of links.

## Examples

- The code

```
Some common household pets are
⌘hyperlink⟨https://en.wikipedia.org/wiki/Cat;
cats⟩, ⌘hyperlink⟨https://en.wikipedia.org/
wiki/Dog; dogs⟩, and ⌘hyperlink⟨https://
en.wikipedia.org/wiki/Green_iguana; green
iguanas⟩.
```

typesets as:

Some common household pets are **cats**, **dogs**, and **green iguanas**.

- The code

```
You can read about green iguanas at
⌘hyperlink⟨https://en.wikipedia.org/wiki/
Green_iguana⟩
```

You can read about green iguanas at

**[https://en.wikipedia.org/wiki/Green\\_iguana](https://en.wikipedia.org/wiki/Green_iguana)**

typesets as


- Entering the code

```
Go to the ⌘intraLink⟨MadHat Help⟩ page, or
⌘intraLink⟨Matrices; click here for a
different page⟩ about matrices.
```

will typeset as

Go to the **MadHat Help** page, or **click here for a different page** about matrices.


# Slides

 **MadHat** supports a way to format a notebook (or parts of a notebook) as a slide presentation, with pages that reveal their content gradually and contain dynamic effects such as animations, content appearing and disappearing at different stages, etc.

The basic unit of a slide presentation is a **slide**, which is the look of a notebook page at a particular stage of the presentation. A page can have many slides. A typical slide presentation is a succession of pages, each partitioned into slides. It is easy to navigate from one slide to the next (or the preceding one), or to different slides or pages, using the Navigate submenu of the app main menu; navigation actions are also accessible via keyboard shortcuts.

There are two main mechanisms to specify the behavior of your slides, each described on a separate help page:

- **Slide transitions**
- **Slide fragments**

The [documentation page](#) of the  **MadHat** website has code samples you can download with many examples of slides and how they can be used to present content with interesting, dynamic effects.



## Slide transitions

Adding slide transitions to a page is as simple as including one of the slide transition commands below at the appropriate places in your content, with optional attributes to customize the transition.

- `⌘pause`.

Marks a slide transition point in a page. User action is needed to proceed to the next slide.

The page will automatically calculate how many slides it has based on the number of slide transitions you included (unless you also included **slide fragments**, which could affect the calculation). For example, if there are 4 `⌘pause` commands, the page will have 5 slides (or more, with slide fragments).

## Attributes to modify the behavior of a slide transition

- `animate in←animation type`

A specification of the type of animation for the new content being revealed. Allowed values are: `slide from left`, `slide from right`, `slide from top`, `slide from bottom`, `fade in`, and `none`. The default is `slide from right`.

- `duration←duration`

A specification of the animation duration, in seconds. The default value is 0.6

- `profile←animation profile`

Specify the profile for the animation. Allowed values are `ease`, `linear`, `bounce`, and `elastic`. The default is `ease`.

# Slide fragments

**Slide transition commands** offer a good level of control in partitioning your page content into slides, but still have a limited flexibility in that they only allow content to be *revealed* in a linear fashion.

An alternative mechanism that can be used in addition to, or in place of, slide transition commands, is that of **slide fragments**.

A slide fragment is a piece of content that you would like to be visible only at a specified range of slides on the page. For example, if the page has 10 slides, you may want some content to appear at slide number 3, and disappear at slide number 7. This is done by wrapping the content in a `⌘slide fragment⟨...⟩` command.

Animations can be specified with an optional attributes block to control how the content appears, how it disappears, and how it moves and, in the case of graphics objects, changes shape, during the range of slides when it is visible.

The syntax is described below.

- `⌘slide fragment⟨from index; to index; fragment contents⟩`

A fragment that will be visible only within a specified range of slides in the page, and which can be optionally animated in prescribed ways.

## Attributes to modify the behavior of a slide fragment

- `animate in←animation type`

A specification of the type of animation when the slide fragment appears. Allowed values are: “slide from left”, “slide from right”, “slide from top”, “slide from bottom”, “fade in”, and “none”. The default is “fade in”.

- `animate out`  $\leftarrow$  *animation type*

A specification of the type of animation when the slide fragment disappears. Allowed values are: “slide to left”, “slide to right”, “slide to top”, “slide to bottom”, “fade out”, and “none”. The default is “fade out”.

- `duration in`  $\leftarrow$  *duration*, `duration out`  $\leftarrow$  *duration*

A specification of the duration of the respective animations, in seconds. The default value is 0.6.

- `profile in`  $\leftarrow$  *profile*, `profile out`  $\leftarrow$  *profile*

Specify the profile for the respective animation. Allowed values are `ease`, `linear`, `bounce`, and `elastic`. The default is `ease`.

- `move on transition`

```
move on transition  $\leftarrow$   $\langle\langle$  %
transition index 1 # x movement 1; y movement 1 # %
transition index 2 # x movement 2; y movement 2 %
...
transition index k # x movement k; y movement k %
 $\rangle\rangle$ 
```

Animates the fragment by moving it on the specified list of transition indices, each time by the specified movement vector.

**Note.** The scale of the coordinates of the movement vector is normally interpreted in ordinary (“document”) coordinates;

however, if the slide fragment is inside a **graphics canvas**, then the coordinates are interpreted in the scale of the  $x$ - and  $y$ -coordinates of the graphics canvas. (A similar conditional interpretation of coordinates is made in the `⌘annotate⟨...⟩` command — see the page on **graphics drawing**.)

- **change on transition**

```
change on transition←⟨%
  transition index 1; property name 1 # list of values # %
  transition index 2; property name 2 # list of values # %
  ...
  transition index k; property name k # list of values # %
  ⟩
```

Animates the fragment by changing its specified properties on the specified list of transition indices, each time to the specified value.

# Subscripts and superscripts

In math mode, mathematical expressions can be annotated with subscripts and superscripts, either using the appropriate commands, or by using the underscore and caret shorthand symbols. The syntax for both of these options is described below.

- **Command notation for subscripts and superscripts**

**Î:** `\subscript`«*main expression; subscript expression*»

Annotate the main expression with another expression as a subscript

**Î:** `\superscript`«*main expression; superscript expression*»

Annotate the main expression with another expression as a superscript

**Î:** `\subsuperscript`«*main expression; subscript expression; superscript expression*»

Annotate the main expression with a subscript and a superscript

- **Underscore and caret shorthand notation**

Subscripts and superscripts can be entered using the underscore `_` and caret `^` shorthand notation used in many computer algebra and typesetting packages.

- `main expression_subscript expression` is equivalent to

**⌘subscript**«*main expression*; *subscript expression*»

- *main expression*<sup>*superscript expression*</sup> is equivalent to

**⌘superscript**«*main expression*; *superscript expression*»

- 

*main expression*<sub>*subscript expression*</sub><sup>*superscript expression*</sup>

and

*main expression*<sup>*superscript expression*</sup><sub>*subscript expression*</sub>

are both equivalent to each other, and to

**⌘:** **⌘subsuperscript**«*main expression*; *subscript expression*; *superscript expression*»

## Example

Entering the code

The equation for the Bernoulli lemniscate in the **⌘**«*X*<sub>1</sub>,*X*<sub>2</sub>» coordinate system is **⌘**«*X*<sub>1</sub><sup>2</sup> + *X*<sub>2</sub><sup>2</sup> = 2*c*<sup>2</sup> (*X*<sub>1</sub><sup>2</sup> - *X*<sub>2</sub><sup>2</sup>)».

produces

The equation for the Bernoulli lemniscate in the  $(X_1, X_2)$  coordinate system is  $(X_1^2 + X_2^2)^2 = 2c^2(X_1^2 - X_2^2)$ .

## Pre-subscripts and pre-superscripts

**Pre-subscripts** and **pre-superscripts** are annotations on a symbol that are attached to the symbol on its left side. (Together, they are sometimes referred to as **prescripts**.) MadHat supports adding any combination of

a pre-subscript, pre-superscript, subscript and superscript annotation on an expression. This is done using the `\multiscript⟨...⟩` command.

- `\multiscript⟨...⟩`

**M:** `\multiscript⟨main expression; subscript; superscript; presubscript; presuperscript`

## Examples

- Entering the code

Uranium-232 is an isotope of uranium with a half-life of 68.9 years. Its chemical symbol is `\hat{\multiscript⟨\roman math⟨U⟩ ; ; ; ; 232⟩}`.

typesets as

Uranium-232 is an isotope of uranium with a half-life of 68.9 years. Its chemical symbol is  $^{232}\text{U}$ .

- The code

The binomial coefficient `\hat{\binom{n}{k}}` is sometimes denoted `\hat{\multiscript⟨C; k; ; n; ⟩}`.

typesets as

The binomial coefficient  $\binom{n}{k}$  is sometimes denoted  ${}_nC_k$ .

- The code

The Gauss hypergeometric function `\hat{\multiscript⟨F; 1; ; 2; ⟩}(a,b;c;z)` can be evaluated using Euler's integral formula



```

M̂: \multiscript{F; 1; ; 2; }(a,b;c;z) =
\frac{\Gamma(c); \Gamma(b)\Gamma(c-b)}{\Gamma(b)\Gamma(c-b)} \int_0^1
x^{b-1} (1-x)^{c-b-1} (1-zx)^{-a} dx

```

will appear as

The Gauss hypergeometric function  ${}_2F_1(a, b; c; z)$  can be evaluated using Euler's integral formula

$${}_2F_1(a, b; c; z) = \frac{\Gamma(c)}{\Gamma(b)\Gamma(c-b)} \int_0^1 x^{b-1} (1-x)^{c-b-1} (1-zx)^{-a} dx$$

See also

- **Typing mathematical formulas in Madhat**

# Mathematical font variants

- `⌘bold math⟨...text...⟩`  
`⌘bmath⟨...text...⟩`

Styles math content in the bold math font variant

- `⌘italic math⟨...text...⟩`  
`⌘itmath⟨...text...⟩`

Styles math content in the italic math font variant

- `⌘blackboard math⟨...text...⟩`  
`⌘bbmath⟨...text...⟩`

Styles math content in the blackboard math font variant

- `⌘calligraphy math⟨...text...⟩`  
`⌘calmath⟨...text...⟩`

Styles math content in the calligraphy math font variant

- `⌘fraktur math⟨...text...⟩`  
`⌘frakmath⟨...text...⟩`

Styles math content in the fraktur math font variant

- `⌘roman math⟨...text...⟩`

Styles math content in the roman math font variant

- `⌘mono math⟨...text...⟩`

Styles math content in the monospace math font variant

- `⌘sans math⟨...text...⟩`

Styles math content in the sans serif font variant

## Examples

$$\mathbf{Ax} = \mathbf{b}$$

$$function(x) = Q(x)$$

$$\text{function}(x) = Q(x)$$

$\mathbb{Q}$  is the field of rationals

$\mathfrak{S}_n$  denotes the symmetric group of order  $n$

$(\Omega, \mathcal{F}, P)$  is a common notation for a probability space

Monospaced math font

**abcdefghijklmnopqrstuvwxy**

**ABCDEFGHIJKLMNPOQRSTUVWXYZ**

*abcdefghijklmnopqrstuvwxy*

*ABCDEFGHIJKLMNPOQRSTUVWXYZ*

abcdefghijklmnopqrstuvwxy

ABCDEFGHIJKLMNPOQRSTUVWXYZ

*ABCDEF GHIJK LMNOP QRSTUVWXYZ*

abcdefghijklmnopqrstuwrh3

ABCD EFGH IJKL MNOP QRSTUVWXYZ

abcdefghijklmnopqrstuvwxy

ABCDEFGHIJKLMNPOQRSTUVWXYZ

abcdefghijklmnopqrstuvwxy

ABCDEFGHIJKLMNPOQRSTUVWXYZ

abcdefghijklmnopqrstuvwxy

ABCDEFGHIJKLMNPOQRSTUVWXYZ

## See also

- **Typing mathematical formulas in Madhat**

# Fractions

- `\fraction⟨numerator; denominator⟩`  
`\frac⟨numerator; denominator⟩`

A fraction

- A shorthand notation for entering fractions:  
`numerator//denominator`

## Example

`\tan x = \frac{\sin x}{\cos x}`

$$\tan x = \frac{\sin x}{\cos x}$$

## Related commands

- `\quasifraction⟨numerator; denominator⟩`

Typeset exactly like a fraction, but the horizontal fraction line is not drawn.

- `\binomial⟨n; k⟩`  
`\binom⟨n; k⟩`

A binomial coefficient  $\binom{n}{k}$

- `\continued fraction⟨n_1; n_2; ...; n_k⟩`  
`\contfrac⟨n_1; n_2; ...; n_k⟩`

A continued fraction  $\frac{1}{n_1 + \frac{1}{n_2 + \frac{1}{\dots + \frac{1}{n_k}}}}$

See also

- **Typing mathematical formulas in Madhat**

## Square roots

- `\square root⟨expression⟩`  
`\sqrt⟨expression⟩`

A square root.

### Example

```
\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2}}}}}}}
```

$$\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2}}}}}}}$$

See also

- [Typing mathematical formulas in Madhat](#)

# Operators

Large operators that adjust their size according to the display mode:

- `sum`

Summation symbol  $\sum$

- `product`, `prod`

Product symbol  $\prod$

- `integral`, `int`

Integral sign  $\int$

- `contourintegral`, `contourint`

Contour integral sign  $\oint$

- `doubleintegral`, `doubleint`

Double integral sign  $\iint$

- `tripleintegral`, `tripleint`

Triple integral sign  $\iiint$

Operators of a fixed size:

- `nabla`, `grad`

The gradient (nabla) operator  $\nabla$



- `smallsum`, `bigsum`

Small and big summation symbols  $\sum$ ,  $\Sigma$

- `smallproduct`, `smallprod`; `bigproduct`, `bigprod`

Small and big product symbols  $\Pi$ ,  $\prod$

- `smallintegral`, `smallint`; `bigintegral`, `bigint`

Small and big integral signs  $\int$ ,  $\int$

- `smallcontourintegral`, `smallcontourint`;  
`bigcontourintegral`, `bigcontourint`

Small and big contour integral signs  $\oint$ ,  $\oint$

**See also**

- **Typing mathematical formulas in Madhat**

## Greek letters

- `alpha`, `beta`, `gamma`, `delta`, ..., `omega`.

Lower case Greek letters

- `Alpha`, `Beta`, `Gamma`, `Delta`, ..., `Omega`.

Upper case Greek letters

You can also type Greek letters by inserting them in the text editor as standard Unicode Greek characters.

## Examples

$$\alpha^2 + \beta^2 = \gamma^2$$

$$\Gamma(\xi)\Gamma(1-\xi) = \frac{\pi}{\sin(\pi\xi)}$$

## See also

- **Typing mathematical formulas in Madhat**

# Differentials

- `da`, `db`, `dc`, ..., `dz`

Differentials of lower case roman letters

- `dA`, `dB`, `dC`, ..., `dZ`

Differentials of upper case roman letters

- `dalpha`, `dbeta`, `dgamma`, ..., `domega`.

Differentials of lower case Greek letters

- `dAlpha`, `dBeta`, `dGamma`, ..., `dOmega`.

Differentials of upper case Greek letters

- `partial`

Partial derivative sign

## Examples

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}$$

$$i\hbar \frac{\partial \Psi}{\partial t} = -a \frac{\hbar^2}{2m} \frac{\partial^2 \Psi}{\partial x^2} + V(x, t) \Psi$$

See also

- **Typing mathematical formulas in Madhat**

# Brackets

Brackets are the parentheses and other forms of bracketing delimiters used to surround expressions in a mathematical formula. MadHat supports smart brackets that automatically adjust their size to fit the height of the content they surround, or whose size can be specified manually.

The simplest way to enter brackets is by using the appropriate bracket symbol in your formula. For example, `(\frac{x+y}{2})^2 \geq x y` will typeset as

$$\left(\frac{x+y}{2}\right)^2 \geq xy$$

The bracket symbols that are available for use in this way are parentheses (`(` and `)` symbols); square brackets (`[` and `]` symbols); curly braces (`{` and `}` symbols); the vertical bar symbol `|`; and two successive vertical bar symbols `||`, which are typeset as the double vertical bar mathematical symbol  $\|$ .

For other bracket types, or for a more precise specification of the bracket behavior, use the bracket commands listed below.

## Bracket commands

- `\left bracket<bracket specification>`  
`\left<bracket specification>`

A left (opening) bracket

- `\right bracket<bracket specification>`  
`\right<bracket specification>`

A right (closing) bracket

- `⌘middle bracket⟨bracket specification⟩`  
`⌘middle⟨bracket specification⟩`

A middle bracket

- `⌘close bracket.`

A closing bracket of matching type and size to the opening bracket it is associated with

## Format for bracket specification

A bracket specification consists of a bracket symbol or keyword, followed by an optional size parameter. The available symbols and keywords are:

- `(` and `)` (parentheses)
- `[` and `]` (square brackets)
- `{` and `}` (curly braces)
- `<` and `>` (angle brackets)
- `|` (vertical bar)
- `||` (double vertical bar)
- `floor` (floor notation)
- `ceiling` or `ceil` (ceiling notation)
- `same` (a closing bracket type that will match the opening bracket it is associated with)

The optional size parameter is a number between 1 and 8, where 1 is the smallest available bracket size, and 8 is the largest available (fixed) bracket size. For example, `⌘left⟨(3⟩` and `⌘left⟨(7⟩` typeset as “(”

and “ $\left($ ”, respectively.

## Additional notes

- If the size is not specified for a matching pair of opening and closing brackets, the brackets will size themselves automatically to fit the vertical dimensions of the bracketed expression. Matching opening and closing brackets need to be in the same top-level code **block** in order for MadHat to match them with each other and determine their size correctly.
- If the size is left unspecified for just the closing bracket, it will size itself to match the opening bracket it is associated with.
- The `⌘middle bracket⟨...⟩` command only works with the vertical bar (`|`) or double vertical bar (`||`) bracket types.
- Closing brackets can be omitted. In that case, if the size of the opening bracket is unspecified, it will size itself to match the height of the entire expression that follows it within the same code **block**.
- A closing bracket without a matching opening bracket will be sized at the specified size, or at the smallest available size if the size is not specified.
- Closing and opening brackets do not need to be of the same type in order to be associated with each other. For example,  $\left(\frac{1}{x}, \frac{1}{y}\right]$  will typeset as  $\left(\frac{1}{x}, \frac{1}{y}\right]$ .
- Closing and opening brackets do not need to be on the same line of a multiline math display in order to be associated with each other.

## Examples

This code illustrates the different bracket types and sizes:

```

M̂:
⌘left⟨(1⟩ x ⌘right⟨)⟩
⌘left⟨(2⟩ x ⌘right⟨)⟩
⌘left⟨(3⟩ x ⌘right⟨)⟩
⌘left⟨(4⟩ x ⌘right⟨)⟩
⌘left⟨(5⟩ x ⌘right⟨)⟩
⌘left⟨(6⟩ x ⌘right⟨)⟩
⌘left⟨(7⟩ x ⌘right⟨)⟩
⌘left⟨(8⟩ x ⌘right⟨)⟩
%
⌘newline.
% This illustrates the use of the "same" bracket
type to match the left bracket
⌘left⟨[1⟩ x ⌘right⟨]⟩
⌘left⟨[2⟩ x ⌘right⟨]⟩
⌘left⟨[3⟩ x ⌘right⟨same⟩
⌘left⟨[4⟩ x ⌘right⟨same⟩
⌘left⟨[5⟩ x ⌘right⟨same⟩
⌘left⟨[6⟩ x ⌘right⟨same⟩
⌘left⟨[7⟩ x ⌘right⟨same⟩
⌘left⟨[8⟩ x ⌘right⟨same⟩
%
⌘newline.
%
⌘left⟨{1⟩ x ⌘right⟨}⟩
⌘left⟨{2⟩ x ⌘right⟨}⟩
⌘left⟨{3⟩ x ⌘right⟨}⟩
⌘left⟨{4⟩ x ⌘right⟨}⟩
⌘left⟨{5⟩ x ⌘right⟨}⟩
⌘left⟨{6⟩ x ⌘right⟨}⟩
⌘left⟨{7⟩ x ⌘right⟨}⟩
⌘left⟨{8⟩ x ⌘right⟨}⟩
⌘newline.
⌘left⟨<1⟩ x ⌘right⟨>⟩
⌘left⟨<2⟩ x ⌘right⟨>⟩
⌘left⟨<3⟩ x ⌘right⟨>⟩
⌘left⟨<4⟩ x ⌘right⟨>⟩
⌘left⟨<5⟩ x ⌘right⟨>⟩
⌘left⟨<6⟩ x ⌘right⟨>⟩

```

This typesets as:

$$\begin{array}{c}
 (x) (x) (x) (x) (x) (x) (x) (x) \\
 [x] [x] [x] [x] [x] [x] [x] [x] \\
 \{x\} \{x\} \{x\} \{x\} \{x\} \{x\} \{x\} \{x\} \\
 \langle x \rangle \langle x \rangle \langle x \rangle \langle x \rangle \langle x \rangle \langle x \rangle \langle x \rangle \langle x \rangle \\
 |x| |x| |x| |x| |x| |x| |x| |x| \\
 \lceil x \rceil \lceil x \rceil \lceil x \rceil \lceil x \rceil \lceil x \rceil \lceil x \rceil \lceil x \rceil \lceil x \rceil
 \end{array}$$

The code

```

 $\hat{M}$ : SU(3)={ A= $\mathbb{matrix}\langle a_{\langle 1\ 1\rangle}; a_{\langle 1\ 2\rangle}; a_{\langle 1\ 3\rangle} \# a_{\langle 2\ 1\rangle}; a_{\langle 2\ 2\rangle}; a_{\langle 2\ 3\rangle} \# a_{\langle 3\ 1\rangle}; a_{\langle 3\ 2\rangle}; a_{\langle 3\ 3\rangle} \rangle$ 
 $\mathbb{space}\langle 10\rangle \mathbb{middle}\langle | \rangle \mathbb{space}\langle 10\rangle$   $a_{\langle j\ k\rangle}$  in
complexnumbers,  $\mathbb{space}\langle 10\rangle$ 
A  $A^{\langle ***\rangle} = I$ ,  $\mathbb{space}\langle 10\rangle$   $\det A = 1$ }
 $\mathbb{newline}$ .
 $\|f\| = ( \int_0^{\infty} |f(x)|^2 dx \mathbb{right}\langle \text{same}\rangle$ 
 $\wedge \langle 1/2\rangle$ 

```

typesets as



$$\mathrm{SU}(3) = \left\{ A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \mid a_{jk} \in \mathbb{C}, \quad AA^* = I, \quad \det A = 1 \right\}$$

$$\|f\| = \left( \int_0^\infty |f(x)|^2 dx \right)^{1/2}$$

See also

- **Typing mathematical formulas in Madhat**

# Horizontal brackets

- `⌘overbrace⟨expression; annotation⟩`

A top-side annotation with a horizontal curly brace

- `⌘underbrace⟨expression; annotation⟩`

A bottom-side annotation with a horizontal curly brace

- `⌘overbracket⟨expression; annotation⟩`

A top-side annotation with a horizontal square bracket

- `⌘underbracket⟨expression; annotation⟩`

A bottom-side annotation with a square bracket

- `⌘overparenthesis⟨expression; annotation⟩`  
`⌘overparen⟨expression; annotation⟩`

A top-side annotation with a horizontal parenthesis

- `⌘underparenthesis⟨expression; annotation⟩`  
`⌘underparen⟨expression; annotation⟩`

A bottom-side annotation with a horizontal parenthesis

- `⌘overtortoise⟨expression; annotation⟩`

A top-side annotation with a horizontal tortoise shell bracket

- `⌘undertortoise⟨expression; annotation⟩`

A bottom-side annotation with a horizontal tortoise shell bracket

## Examples

- `k ** x = ⌘overbrace⟨x+...+x; k ⧻ times⟩⟩`

$$k \cdot x = \overbrace{x + \dots + x}^{k \text{ times}}$$

- `k ** x = ⌘underbrace⟨x+...+x; k ⧻ times⟩⟩`

$$k \cdot x = \underbrace{x + \dots + x}_{k \text{ times}}$$

- `k ** x = ⌘overbracket⟨x+...+x; k ⧻ times⟩⟩`

$$k \cdot x = \overbracket{x + \dots + x}^{k \text{ times}}$$

- `k ** x = ⌘underbracket⟨x+...+x; k ⧻ times⟩⟩`

$$k \cdot x = \underbracket{x + \dots + x}_{k \text{ times}}$$

- `k ** x = ⌘overparen⟨x+...+x; k ⧻ times⟩⟩`

$$k \cdot x = \overparen{x + \dots + x}^{k \text{ times}}$$

- `k ** x = ⌘underparen⟨x+...+x; k ⧻ times⟩⟩`

$$k \cdot x = \underparen{x + \dots + x}_{k \text{ times}}$$

- `k ** x = ⌘overtortoise⟨x+...+x; k ⧻ times⟩⟩`

$$k \cdot x = \overbrace{x + \dots + x}^{k \text{ times}}$$

- `k ** x = ⌘undertortoise⟨x+...+x; k ⧻ times⟩⟩`

$$k \cdot x = \underbrace{x + \dots + x}_{k \text{ times}}$$

See also

- **Typing mathematical formulas in Madhat**

# Extensible arrows and relations

- 

`\xrightarrow[bottom annotation]{top annotation}`

An extensible right arrow with optional annotations above and below it

- 

`\xrightarrow[bottom annotation]{top annotation}`

An extensible double right arrow with optional annotations above and below it

- 

`\xleftarrow[bottom annotation]{top annotation}`

An extensible left arrow with optional annotations above and below it

- 

`\xleftrightarrow[bottom annotation]{top annotation}`

An extensible double left arrow with optional annotations above and below it

- 

`\xleftrightarrow[bottom annotation]{top annotation}`

An extensible bidirectional (left-right) arrow with optional annotations above and below it

- 

`\xleftrightarrow[bottom annotation]{top annotation}`

An extensible double bidirectional (left-right) arrow with optional annotations above and below it

- `↔annotated equal⟨top annotation; bottom annotation⟩`

An extensible equal sign with optional annotations above and below it

## Examples

$$\sum_{k=1}^n \frac{(-1)^k}{k} \xrightarrow[n \rightarrow \infty]{} \log 2$$

$$\mathbb{P} \left( \frac{X_1 + \dots + X_n}{\sqrt{n}\mu} \leq t \right) \xrightarrow[n \rightarrow \infty]{} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^t e^{-x^2} dx$$

$$\frac{X_1 + \dots + X_n}{\sqrt{n}\mu} \xrightarrow[n \rightarrow \infty]{D} N(0, 1)$$

$$\frac{X_1 + \dots + X_n}{\sqrt{n}\mu} \xRightarrow[n \rightarrow \infty]{} N(0, 1)$$

$$N(0, 1) \xleftarrow[n \rightarrow \infty]{} \frac{X_1 + \dots + X_n}{\sqrt{n}\mu}$$

$$\frac{X+Y}{\sqrt{2}} \stackrel{dist}{=} X$$

## See also

- Keywords for ordinary arrows are described in the **binary relations** page
- **Typing mathematical formulas in Madhat**

## Special mathematical symbols

- `infinity`, `infty`

Infinity symbol  $\infty$

- `naturalnumbers`

The set of natural numbers  $\mathbb{N}$

- `integers`

The set of integers  $\mathbb{Z}$

- `rationals`

The set of rationals  $\mathbb{Q}$

- `reals`

The set of reals  $\mathbb{R}$

- `complexnumbers`

The set of complex numbers  $\mathbb{C}$

- `hbar`

Planck's constant  $\hbar$

### See also

- **Typing mathematical formulas in Madhat**

# Binary relations

- Arrows:

- `leftarrow`, `from`  $\leftarrow$
- `rightarrow`, `to`  $\rightarrow$
- `uparrow`  $\uparrow$
- `downarrow`  $\downarrow$
- `leftrightharrow`, `fromto`  $\leftrightarrow$
- `doublerightarrow`  $\Rightarrow$
- `doubleleftarrow`  $\Leftarrow$
- `doubleleftrightharrow`, `iff`  $\Leftrightarrow$
- `mapsto`  $\mapsto$

- Ordinary relations:

- `approx`  $\approx$
- `cong`  $\equiv$
- `sim`  $\sim$
- `/=`  $\neq$
- `<=`  $\leq$
- `>=`  $\geq$

- Set-theoretic relations:

- `elementof`, `in`  $\in$



- `subset`  $\subset$
- `subseteq`  $\subseteq$
- `superset`  $\supset$
- `supseteq`  $\supseteq$

## See also

- [Extensible arrows and relations](#)
- [Typing mathematical formulas in Madhat](#)

# Binary operators

- **Arithmetic operators:**

- `+`, `plus`  $+$
- `-`, `minus`  $-$
- `+-`, `plusminus`  $\pm$
- `-+`, `minusplus`  $\mp$
- `*`, `times`  $\times$
- `**`, `dot`  $\cdot$
- `***`, `convolve`  $*$
- `/`, `dividedby`  $/$

- **Miscellaneous algebra operators:**

- `fatdot`  $\bullet$

- **Set-theoretic operators:**

- `intersection`  $\cap$
- `bigintersection`  $\bigcap$
- `union`  $\cup$
- `bigunion`  $\bigcup$

- **Logic operators:**

- `conjunction`, `and`  $\wedge$
- `bigconjunction`, `bigand`  $\bigwedge$

- `disjunction`, `or`     $\vee$
- `bigdisjunction`, `bigor`     $\bigvee$
- **Ellipses:**
  - `centerdots`     $\cdots$
  - `verticaldots`     $\vdots$
  - `sedots`     $\ddots$
  - `nedots`     $\cdot\cdot\cdot$

See also

- **Typing mathematical formulas in Madhat**

# Matrices

- `\matrix⟨matrix cells⟩`

A matrix. The syntax for specifying the matrix cells is the same as for **tables**.

## Examples

- The code `\matrix⟨1; 2; 3 # 4; 5; 6⟩` typesets as  $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ .
- The code

```

 $\hat{M}$ : p_A(x) =det(x I-A) =\matrix⟨
x-a_⟨11⟩ ; -a_⟨12⟩ ; -a_⟨13⟩ ; centerdots ; -a_⟨1n⟩
#
-a_⟨21⟩ ; x-a_⟨22⟩ ; -a_⟨23⟩ ; centerdots ; -a_⟨2n⟩
#
-a_⟨31⟩ ; -a_⟨32⟩ ; x-a_⟨32⟩ ; centerdots ; -a_⟨3n⟩
#
verticaldots ; verticaldots ; verticaldots ;
sedots ; verticaldots
#
-a_⟨n1⟩ ; -a_⟨n2⟩ ; -a_⟨n3⟩ ; centerdots ; x-a_⟨n n⟩
⟩

```

typesets as

$$p_A(x) = \det(xI - A) = \begin{vmatrix} x - a_{11} & -a_{12} & -a_{13} & \cdots & -a_{1n} \\ -a_{21} & x - a_{22} & -a_{23} & \cdots & -a_{2n} \\ -a_{31} & -a_{32} & x - a_{32} & \cdots & -a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_{n1} & -a_{n2} & -a_{n3} & \cdots & x - a_{nn} \end{vmatrix}$$

See also

- [Tables](#)
- [Typing mathematical formulas in Madhat](#)

# Mathematical symbol decorations

**Decorations** are annotations on a mathematical symbol that denote an operation done to the object denoted by the symbol, or denote an entirely new symbol that relates to the original symbol in some way. Examples of decorations are the **hat operator** and the **overdot** notation.

The following decoration commands are available in  **MadHat**:

- `\overbar⟨expression⟩`  
`\bar⟨expression⟩`

Overbar decoration

- `\underbar⟨expression⟩`

Underbar decoration

- `\hat⟨expression⟩`

Hat decoration

- `\tilde⟨expression⟩`

Tilde decoration

- `\vector⟨expression⟩`  
`\vec⟨expression⟩`

Vector decoration

- `\overdot⟨expression⟩`

Dot (time derivative) decoration

- `\double dot⟨expression⟩`

`\ddot{expression}`

Double dot (second time derivative) decoration

- `\triple dot{expression}`  
`\dddots{expression}`

Triple dot (third time derivative) decoration

## Examples

- The code

```
\hat{M}: \tilde{F}(u) = \int_{-\infty}^{\infty} F(x) e^{-i u x} dx
```

typesets as

$$\tilde{F}(u) = \int_{-\infty}^{\infty} F(x) e^{-i u x} dx$$

- The code

```
\hat{M}: \ddot{x} = \vec{M} \tilde{\tau} - \overline{\dot{x}} - \overline{(x + iy)} - \underbar{\lambda}
```

typesets as

$$\ddot{x} = \vec{M} \tilde{\tau} - \overline{\dot{x}} - \underline{\lambda}$$

$$\overline{x + iy} = x - iy$$

$$\overline{\overline{(x + iy)}} = x + iy$$

See also

- **Typing mathematical formulas in Madhat**



# Images

- `⌘image⟨image file name⟩`
- `⌘image⟨image file name; scaling factor⟩`

Insert an image from the notebook's **media library**.

If the scaling factor is provided, the image is scaled proportionally by that factor (this is an abbreviated syntax for providing a value for the `scale` attribute; see the list of attributes below allowing more detailed control over the sizing and positioning of the image.

## Attributes

- `scale width←scaling factor`
- `scale height←scaling factor`
- `scale←scaling factor`
- `width←width`
- `height←height`
- `raise←points`
- `lower←points`
- `mathcenter`, `no mathcenter` – a Boolean attribute that controls whether the image is vertically aligned to be flush with the baseline, or centered around the math axis

## Examples



- Some text, and a scaled picture inserted: , then more text after the picture.



- The same picture, stretched horizontally:



- The same picture, stretched vertically:



- The same picture, width 100 points:



- The same picture, width 100 points and raised:  
Then more text.

- The same picture, width 100 points and lowered:



Then more text.

- Math-centered pictures inside a math equation:

$$\int_0^1 F\left(\img[alt="A small tabby cat lying down." data-bbox="322 175 417 205"]\right) dx = \img[alt="A larger tabby cat lying down." data-bbox="502 163 753 205"]$$

**See also**

- **Videos**

## Adding videos to a page

- `⌘video⟨video file name⟩`
- `⌘video⟨video file name; scaling factor⟩`

Insert a video from the notebook's **media library**.

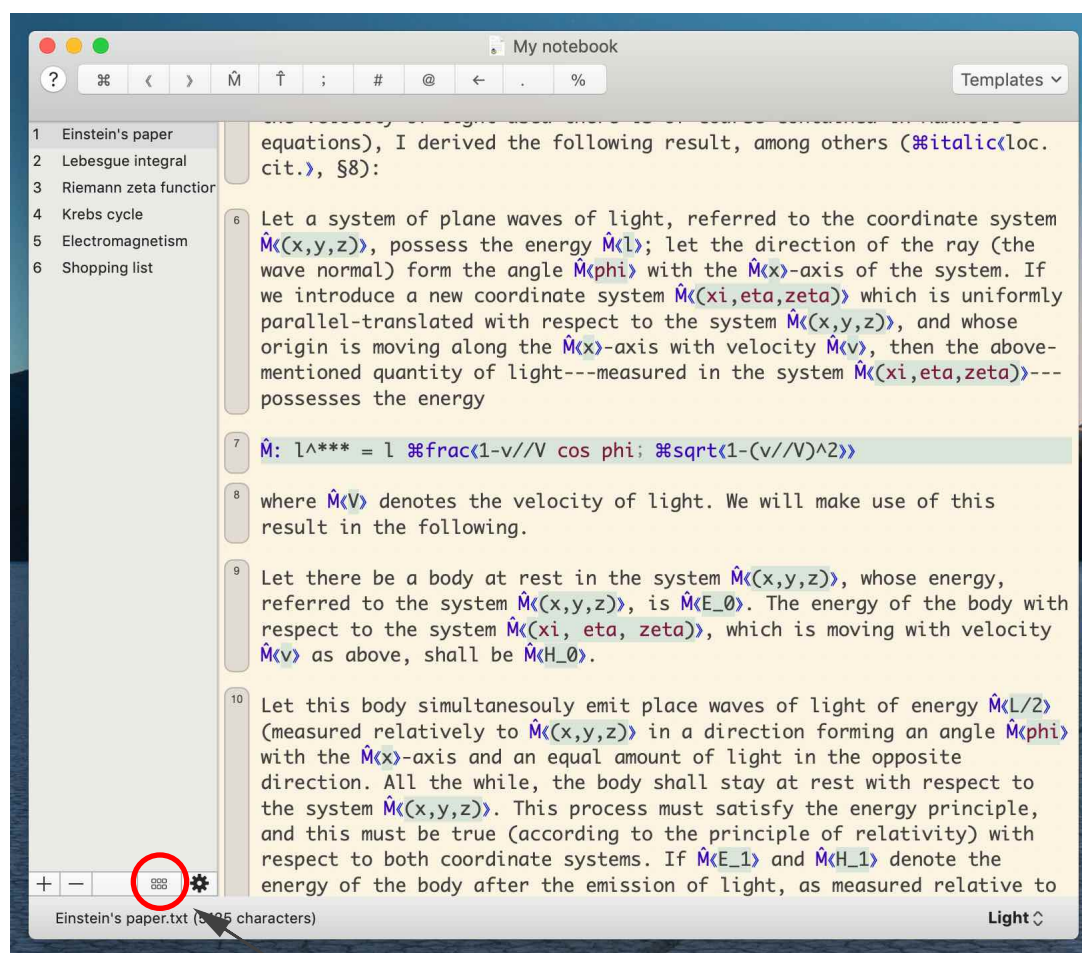
If the scaling factor is provided, the video is scaled proportionally by that factor.

### See also

- **Images**

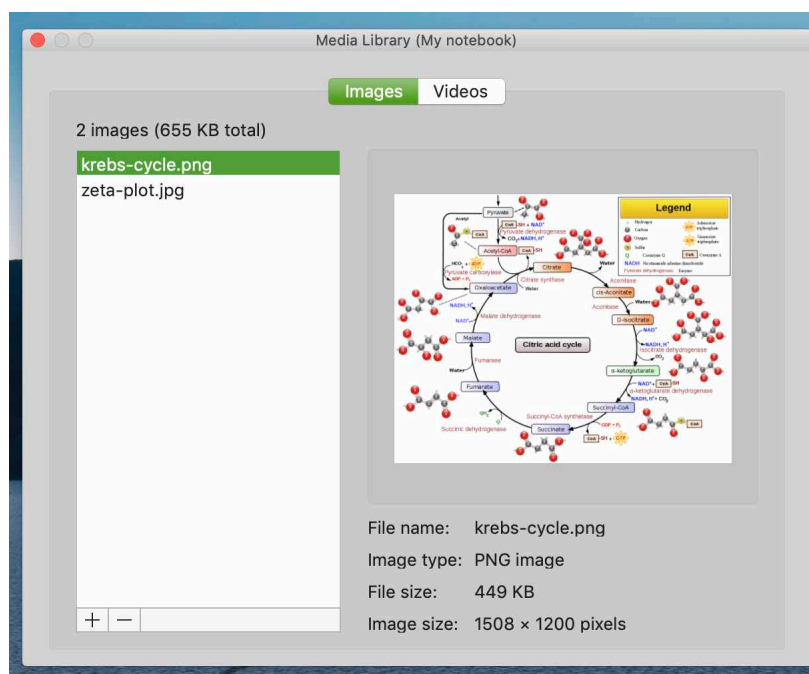
# The media library of a notebook

Every **MadHat** notebook contains a media library into which you can import image and video files. You access the media library by clicking the media library icon at the bottom left corner of the editor window, as shown in the screenshot:



media library icon

The media library window for the notebook will appear, and allow you to import media files, view your existing files, reorder and rename them, etc. Here is a screenshot of the window:



Once you have imported your files, you can add them as part of the content of your notebook pages by using the `⌘image⟨...⟩` and `⌘video⟨...⟩` commands, described in the help pages listed below.

## See also

- **Images**
- **Videos**

# Graphics canvasses

The graphics canvas is the basic high-level object for graphics drawing. A graphics canvas will render as a rectangle of specified dimensions inside your document. Inside the canvas (or even outside it, if you disable cropping) you can draw lines, curves, shapes, and other graphical objects by including the appropriate commands. The syntax for creating a graphics canvas is:

```
⌘graphics canvas«list of graphics commands»
```

An optional attributes block can be included; see below for a list of recognized attributes.

Graphics commands are divided into several types, each described in a separate help page below:

- **Drawing commands**
- **Styling commands**
- **Pen drawing commands**
- **Pen control commands**
- **Mathematical plotting commands**

The [documentation page](#) of the  MadHat website has code samples you can download with examples of how these commands are used in practice.

## Attributes for a graphics canvas

The parameters to customize the graphics canvas are provided as attributes:

- `width` ← `width`

Specify the width, in points. The default value is 200.

- `height` ← `height`

Specify the height, in points. The default value is 200.

- `min x` ← `min x`, `max x` ← `max x`, `min y` ← `min y`, `max y` ← `max y`

Specify the ranges for the  $x$  and  $y$  coordinates that are mapped to the canvas rectangle. The default values are 0 for `min x` and `min y`, 10 for `max x` and `max y`.

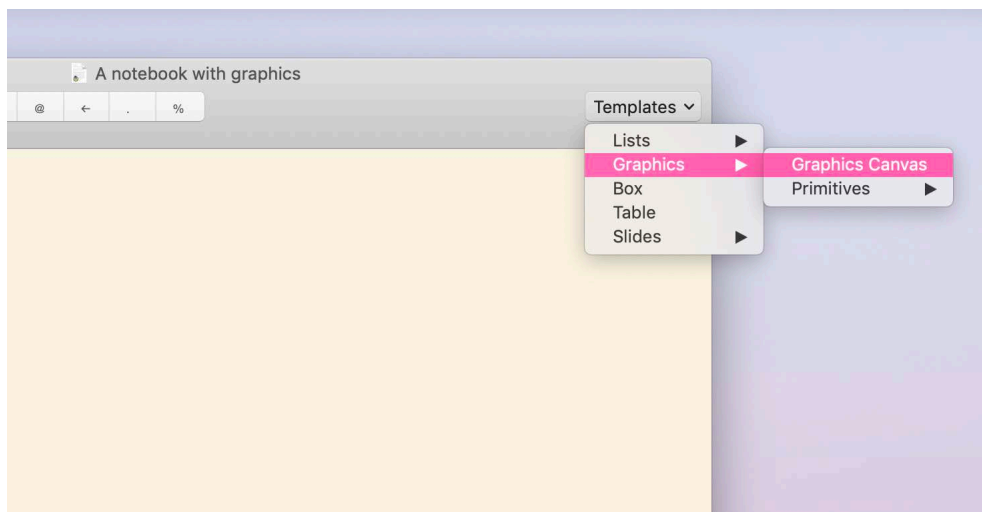
- `crop`, `no crop`

Specify whether the contents should be cropped to the image bounding rectangle. Cropping is enabled by default.

## Quickly creating a graphics canvas

A convenient way to create a graphics canvas with a code block including some of the standard attributes is by selecting “Graphics Canvas” from the code templates menu in the top right corner of the editor window, as shown in this screenshot:





# Graphics drawing commands

To draw graphics, you include a list of **graphics primitive** commands inside a `⌘graphics canvas⟨...⟩` command block. You can also configure the behavior of a graphics primitive by including graphics styling commands before it to modify the graphics style used for drawing the primitive.

Here is a list of the recognized graphics primitives.

- 

```
⌘frame.
```

Draw a (stroked) frame around the canvas rectangle

- 

```
⌘filled frame.
```

Draw a filled frame

- 

```
⌘filled stroked frame.
```

Draw a filled and stroked frame

- 

```
⌘grid⟨x spacing; y spacing⟩
```

Draw a grid with the specified  $x$  and  $y$  spacing values

•

**⌘line**⟨*point 1 x; point 1 y # point 2 x; point 2 y # ... # point n x; point n y*⟩

Draw a line (aka polyline) connecting the specified sequence of points

•

**⌘polygon**⟨*point 1 x; point 1 y # point 2 x; point 2 y # ... # point n x; point n y*⟩

Draw a closed polygon connecting the specified sequence of points

•

**⌘filled polygon**⟨*point 1 x; point 1 y # point 2 x; point 2 y # ... # point n x; point n y*⟩

Draw a filled polygon connecting the specified sequence of points

•

**⌘filled stroked polygon**⟨*point 1 x; point 1 y # point 2 x; point 2 y # ... # point n x; point n y*⟩

Draw a filled and stroked polygon connecting the specified sequence of points

•

**⌘marker**⟨*x coordinate; y coordinate*⟩

Draw a marker at the specified point. The marker type can be set using the `⌘marker type⟨...⟩` command.

- 

```
⌘circle⟨x coordinate; y coordinate # radius⟩
```

Draw a circle centered at the specified point, with the given radius

- 

```
⌘disk⟨x coordinate; y coordinate # radius⟩
```

Draw a disk (filled circle) centered at the specified point, with the given radius

- 

```
⌘circle disk⟨x coordinate; y coordinate # radius⟩
```

Draw a filled, stroked circle centered at the specified point, with the given radius

- 

```
⌘arc⟨x coordinate; y coordinate # radius # start angle; end angle⟩
```

Draw a circular arc centered at the specified point, with the given radius, start angle and end angle

- 

```
⌘ellipse⟨x coordinate; y coordinate # x radius # y radius⟩
```

Draw an ellipse centered at the specified point, with the given  $x$  and  $y$  radii

- 

```
⌘filled ellipse⟨...⟩
```

Draw a filled ellipse centered at the specified point, with the given  $x$  and  $y$  radii

- 

```
⌘filled stroked ellipse⟨...⟩
```

Draw a filled, stroked ellipse centered at the specified point, with the given  $x$  and  $y$  radii

- 

```
⌘rectangle⟨corner 1 x; corner 1 y # corner 2 x; corner 2 y⟩
```

Draw a rectangle whose opposing corners are the two specified points

- 

```
⌘filled rectangle⟨...⟩
```

Draw a filled rectangle whose opposing corners are the two specified points

- 

```
⌘filled stroked rectangle⟨...⟩
```

Draw a filled, stroked rectangle whose opposing corners are the two specified points

•

```
⌘arrow«point 1 x; point 1 y # point 2 x; point 2 y»
```

Draw an arrow from the specified *point 1* to *point 2*

•

```
⌘bezier«point 1 x; point 1 y # point 2 x; point 2 y # point 3  
x; point 3 y # point 4 x; point 4 y»
```

```
⌘bézier«point 1 x; point 1 y # point 2 x; point 2 y # point 3  
x; point 3 y # point 4 x; point 4 y»
```

Draw a cubic Bézier curve from *point 1* to *point 4*, with *point 2* and *point 3* serving as control points

•

```
⌘annotation«x coordinate; y coordinate # annotation»
```

Place an annotation with arbitrary content (such as text or a mathematical expression) with its typesetting origin at the specified point

•

```
⌘centered annotation«x coordinate; y coordinate #  
annotation»
```

Place an annotation with arbitrary content (such as text or a

mathematical expression) with its center the specified point

**Note.** The `⌘annotation⟨...⟩` and `⌘centered annotation⟨...⟩` commands can also be used outside of a graphics canvas. In that case the annotation point is interpreted in document coordinates (relative to the current typesetting point) instead of canvas coordinates, but the effect is otherwise the same.

- 

`⌘curved text layout⟨curve specification; text⟩`

Lay out the provided text along the specified curve (entered using standard graphics drawing commands such as `⌘line⟨...⟩`, `⌘bezier⟨...⟩`, `⌘circle⟨...⟩`).

## Animatable properties

Graphics primitives can animate their shape during a slide transition by wrapping them in a `⌘slide fragment⟨...⟩` command and including the optional `change on transition` attribute inside an attributes block. Here is a list of the animatable properties for the different graphics primitives.

- Animatable properties for `⌘line⟨...⟩`, `⌘polygon⟨...⟩`, `⌘filled polygon⟨...⟩`, and `⌘filled stroked polygon⟨...⟩` commands:  
`points`: this property should be specified as a flat list of the  $x$ - and  $y$ -coordinates of the points of the line or polygon, in the format

*pt 1 x; pt 1 y; pt 2 x; pt 2 y; ...; pt k x; pt k y*

- Animatable properties for `⌘arrow⟨...⟩` command:  
`start`  
`end`
- Animatable properties for `⌘rectangle⟨...⟩` command:  
`corner 1`  
`corner 2`
- Animatable properties for `⌘circle⟨...⟩`, `⌘disk⟨...⟩` → `⌘circle disk⟨...⟩` commands:  
`center`  
`radius`
- Animatable properties for `⌘ellipse⟨...⟩`, `⌘filled ellipse⟨...⟩` → `⌘filled stroked ellipse⟨...⟩` commands:  
`center`  
`x radius`  
`y radius`
- Animatable properties for `⌘arc⟨...⟩` command:  
`center`  
`radius`  
`start angle`  
`end angle`
- Animatable properties for `⌘bezier⟨...⟩` command:  
`point 1`  
`point 2`  
`point 3`  
`point 4`

## See also

- **Graphics canvasses**
- **Graphics styling commands**



- **Graphics pen drawing commands**
- **Graphics pen control commands**
- **Creating a slide presentation**

# Graphics styling commands

Graphics styling commands control the style attributes of graphics primitives that you include in your diagram. Here are the different styling commands.

- `⌘line thickness`⟨*thickness*⟩

Set the line thickness to the specified thickness, in points

- `⌘marker scale`⟨*scale*⟩

Scale markers up or down from their default size, by the specified factor

- `⌘marker type`⟨*type*⟩

Set the marker type. Allowed values are `disk`, `square`, `diamond`, `star`, and `triangle`. The default is `disk`.

- `⌘stroke color`⟨*color argument*⟩  
`⌘stroke colour`⟨*color argument*⟩

Set the stroke color for stroked graphics shapes

- `⌘fill color`⟨*color argument*⟩  
`⌘fill colour`⟨*color argument*⟩


Set the fill color for filled graphics shapes

## See also

- **Graphics canvasses**

- **Graphics drawing commands**
- **Graphics pen drawing commands**
- **Graphics pen control commands**

# Graphics pen drawing commands

In addition to graphics primitives, the  **MadHat** graphics environment supports a “pen drawing” model, also known as **turtle graphics**. In this model, you give drawing commands to a virtual pen that has a position and a direction. You can tell the pen to move forward back, to rotate to the left or right, etc.

Pen graphics commands are classified into pen *drawing* commands, described on this page, and a separate class of **pen control commands** described on a separate help page.

- **⌘move to**⟨ $x$ ;  $y$ ⟩

Move the pen to an absolute position  $(x, y)$  on the canvas. The initial position is the center of the canvas rectangle. When the pen is moved, the pen direction is set to the vector pointing from the current position to the new position.

- **⌘line to**⟨ $x$ ;  $y$ ⟩

Stroke a line from the current pen position to an absolute position  $(x, y)$ . The pen direction is updated to the vector pointing from the current position to the new position.

- **⌘curve to**⟨ $x_1$ ;  $y_1$  #  $x_2$ ;  $y_2$  #  $x_3$ ;  $y_3$ ⟩

Stroke a cubic Bézier curve from the current pen position to the point  $(x_3, y_3)$  using the specified control points  $(x_1, y_1), (x_2, y_2)$ . The pen direction is updated to the tangent vector of the Bézier curve at its endpoint.

- **⌘pen forward**⟨*distance*⟩

Move the pen forward by a specified distance


- `⌘pen backward`*«distance»*

Move the pen backward by a specified distance

## See also

- **Graphics canvasses**
- **Graphics drawing commands**
- **Graphics styling commands**
- **Graphics pen control commands**

# Graphics pen control commands

The  MadHat graphics environment supports a “pen drawing” model, also known as **turtle graphics**. In this model, you give drawing commands to a virtual pen that has a position and a direction. You can tell the pen to move forward back, to rotate to the left or right, etc.

Pen graphics commands are classified into pen *drawing* commands, described on this page, and a separate class of **pen control commands** described on a separate help page.

- **⌘pen up.**

Lift the pen up (i.e., turn off drawing when the pen moves)

- **⌘pen down.**

Lower the pen (i.e., turn on drawing when the pen moves)

- **⌘pen direction⟨*x*; *y*⟩**

Set the pen direction to a specified vector

- **⌘pen angle⟨*angle*⟩**

Set the pen direction to a specified angle relative to the positive  $x$  axis, in degrees. The initial angle is 90 degrees (that is, the pen is turned north).

- **⌘turn pen left⟨*angle*⟩**

Rotate the pen direction to the left by a specified angle, in degrees

- **⌘turn pen right⟨*angle*⟩**

Rotate the pen direction to the right by a specified angle, in degrees

- `⌘turn pen north.`  
`⌘turn pen south.`  
`⌘turn pen east.`  
`⌘turn pen west.`  
`⌘turn pen northeast.`  
`⌘turn pen northwest.`  
`⌘turn pen southeast.`  
`⌘turn pen southwest.`


Turn the pen to the given compass direction

## See also

- **Graphics canvasses**
- **Graphics drawing commands**
- **Graphics styling commands**
- **Graphics pen drawing commands**

# Plotting mathematical functions

Within a graphics canvas, you can plot the graph of a mathematical function, specified as an ordinary formula written in standard math mode syntax, using plotting commands.

The sample code found on the [documentation page](#) of the  MadHat website shows various examples of how to use these features.

Below is the syntax for the available plotting commands.

- Plotting a function

- `⌘plot⟨⟨function in math mode⟩⟩`

Plots the graph of a function. The function is assumed to be a function of the variable name  $x$ , and  $x$  ranges from 0 to 1.

- `⌘plot⟨⟨function in math mode⟩⟩ ; var name⟩`

- `⌘plot⟨⟨function in math mode⟩⟩ ; a < var name < b⟩`

Plots the graph of a function (in the specified variable name, for example “ $x$ ”, or “ $t$ ”). If a range is provided through the use of the notation `a < var name < b⟩`, the plotting variable ranges between the specified values. Otherwise if only the variable name is specified, the variable ranges from 0 to 1.

- Polar plots:

```
⌘polar plot⟨⟨function in math mode⟩⟩ ; a < angle var
name < b⟩
```

Inserts a polar plot of a function (in the specified angular variable



name) as the angle ranges between the two specified values.

- Parametric plots:

```
⌘parametric plot⟨M⟨x func. in math mode⟩ ; M⟨y func. in
math mode⟩ ; a < var name < b⟩
```

## Plots with parameters

When you plot a function, in the simplest example the function will depend only on a single variable, say  $x$ , which will be the label of the plotting variable. You can also allow your function to depend on several variables, where one of the variables will be used as the plotting variable, and the other variables have the meaning of **parameters**. The value of the parameters needs to be specified by including an attributes block inside the plot command. (If the value of a parameter is not specified, it is assumed to be 0.)

As an example, you can plot the function  $5\sin(x) + t$  with respect to the variable  $x$ , giving the parameter  $t$  the value 3, by writing the code

```
⌘plot⟨M⟨5sin(x)+t⟩%
@⟨t←3⟩ ; %
-10<x<10%
@⟨t←3⟩%
⟩
```

One main use case for including parameters in your functions is when you want to create animations in which the value of a parameter is changed continuously; this is described below.

## Animating the plot range

The plot range can be animated during a slide transition, by wrapping the plot command inside a `⌘slide fragment⟨...⟩` command and adding an attribute animation instruction modifying the `plot range` attribute. See the [slide fragments](#) help page for details on animating attributes, and see the sample code available online for examples.

## Animating plots by changing parameter values

When you include a parameter in the function you are plotting, say  $t$ , the label of that parameter becomes an attribute whose value can be animated during a transition between slides on the page. See the [slide fragments](#) help page for details on animating attributes, and see the sample code available online for examples showing the use of these animation options.

## Notes on the formula parser and evaluator

Below are some technical notes on how the formula parser works and what are some of its abilities as well as limitations. Understanding these details may be useful to use the plotting features effectively.


- Calculations are done in the 64 bit “double” C data type. Functions are evaluated using the standard C library functions. This implies limitations on the precision and fidelity of plots.
- The formula parser recognizes the `⌘frac⟨...; ...⟩` and `⌘sqrt⟨...⟩` commands; superscripts (which are interpreted as exponents); the mathematical constants  $e$  and  $\pi$  (where  $\pi$  is typed in the usual way as `pi` in math mode, or can be entered directly as the symbol  $\pi$  in the source editor); the brackets `(, )`, `[, ]`, `{, }`, and absolute value signs `|`. Brackets can also be entered using the `⌘left bracket⟨...⟩` and `⌘right bracket⟨...⟩`, as described

on the **brackets** help page.

- The following mathematical functions are recognized:  $\exp$ ,  $\log$ ,  $\ln$ ,  $\sin$ ,  $\cos$ ,  $\tan$ ,  $\arcsin$ ,  $\arccos$ ,  $\arctan$ .
- Plotting variable and parameter names can be single-letter symbols or multi-letter symbols using Latin letters. They can also be the names of Greek letters. The symbols  $e$  and  $\pi$  cannot be used for variable and parameter names since they represent the mathematical constants.
- The formula parser supports implicit (or “implied”) multiplication, for example interpreting “ $3x$ ” as “three times  $x$ ”, and treats the binary operators  $+$  and  $-$  in the usual way as unary operators when they appear at the beginning of a formula or subformula (such as a bracketed expression, or inside a square root).
- The parser respects the standard conventions on order of operations (also known as operator precedence). Implicit multiplication is assigned the same precedence as explicit multiplication and division, so that the expression  $1 / 2x$  is parsed from left to right and evaluates to “one half times  $x$ ”.
- The vertical bar symbol “ $|$ ”, interpreted as an absolute value bracket, is used both as an opening (left) and closing (right) bracket. To resolve possible ambiguities that may arise as a result in parsing an expression involving such symbols, vertical bars are processed by the formula parser in the following way: if a vertical bar follows a binary operator or a left bracket of any type, or appears at the beginning of the formula (or subformula), it is interpreted as an opening bracket; otherwise, if the vertical bar can be interpreted as a closing bracket in a way that satisfies the matching rules for brackets (i.e., it can be paired in a consistent way with an already scanned vertical bar that was interpreted as an opening bracket), it is interpreted as a closing bracket; and finally,

if such a matching does not apply, the vertical bar is interpreted as an opening bracket.

- When entering vertical bars to denote absolute value operations, you can override the behavior described above by explicitly marking a vertical bar as a left bracket or a right bracket, using the `⌘left bracket⟨|⟩` and `⌘right bracket⟨|⟩` commands.
- The rules described above make it possible to successfully parse some mathematical expressions which nonetheless may be ambiguous in how they are interpreted by humans. This ambiguity is related to two aspects of the structure of mathematical formulas on which there is no universal agreement regarding how they are to be interpreted:
  - Expressions such as  $1/2 \times x$  and  $1/2x$  are understood as “ $1/(2x)$ ” by some people and  $(1/2)x$  by other people. Which interpretation is more “correct” has been the source of some **debate**. Technically, the issue is whether a multiplication operator (and even more so the implicit multiplication operator) should be considered as having a higher precedence than the division operator, or the same precedence.
  - An expression such as  $|x|y|z|$  lends itself to two possible interpretations, either as “[absolute value of  $x$ ] times  $y$  times [absolute value of  $x$ ]” or as “absolute value of [ $x$  times (the absolute value of  $y$ ) times  $z$ ]”. In ordinary mathematical writing, the ambiguity is sometimes resolved by using vertical bars of different heights when an expression involves nested absolute value operations, but the formula parser does not consider this type of information, instead applying the rules mentioned above.

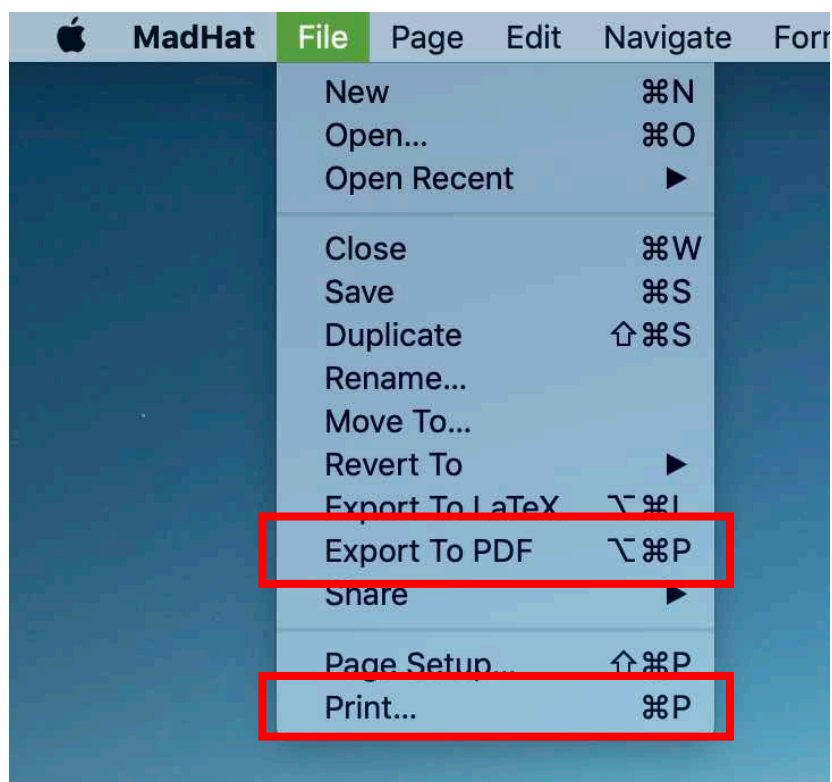
The recommendation for formulas within  **MadHat** is to be aware of these sources of ambiguities and to avoid plotting expressions

containing them (which can easily be done by adding parentheses) even if they define expressions that can be successfully plotted, to avoid confusion.

# Printing and PDF exporting

To print a notebook, select the Print option from the File menu of the main menu.

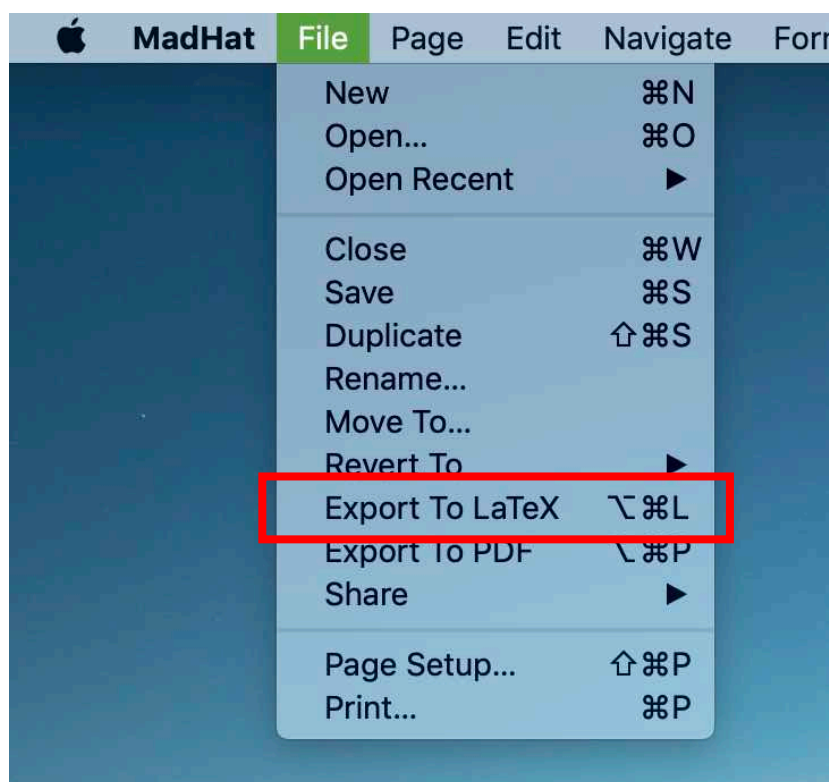
To export a notebook as a PDF, select the Export to PDF option from the File menu.




# Exporting to LaTeX




 **MadHat** enables exporting the content of a notebook to the **LaTeX** file format.

To export a notebook to a LaTeX file, select the Export to LaTeX option from the File menu.





## Caveats

The design of the  **MadHat** language differs sufficiently from that of LaTeX that perfect conversion from one language to another seems impractical. When you export a notebook into LaTeX, you should keep in mind the following caveats:

- The conversion from  **MadHat** to LaTeX code is not completely faithful. Certain formatting instructions are not converted, as well as graphics commands, slide commands, attributes blocks, and other  **MadHat** language constructs that do not map in a simple way to LaTeX commands. Text, mathematical symbols and other “normal” content will be converted correctly, but it is a good idea to inspect your exported LaTeX code after exporting and make appropriate manual adjustments as needed to get the LaTeX document looking the way you intended.
- Math displays in the  **MadHat** notebook code are exported to a LaTeX.

```
\begin{align*} ... \end{align*}
```



block. However,  **MadHat**’s typesetting algorithm differs from the LaTeX `align*` environment in that the content is automatically broken up into multiple lines and each line is aligned based on the position of equal signs and other binary relations. The LaTeX conversion algorithm does not attempt to reproduce these features, so you may need to insert LaTeX alignment and linebreaking symbols manually in the exported file.


- LaTeX does not support the Unicode standard.  **MadHat**’s conversion algorithm will replace Greek letters and standard mathematical symbols in your notebook code with the correct LaTeX commands (e.g.,  $\alpha$  will be converted to `\alpha`), but any other unsupported Unicode symbols in your code will be output verbatim into the exported LaTeX file and then lead to a compilation error when you try to compile the file with the LaTeX typesetting engine. Manual adjustment will be necessary to fix the problem.



# Themes and the themes editor

## Syntax highlighting

The  **MadHat** editor window applies syntax highlighting to your code so as to provide easy visual cues for the syntactical roles played by different parts of your content. This facilitates easy reading and editing of  **MadHat** language content.

The collection of colors constituting the settings for syntax highlighting, together with the choice of a font and font size in which to view your code, is referred to as a **theme**.  **MadHat** provides pre-installed themes you can select for your content editing, and a **themes editor** that enables you to define new themes to suit your personal preferences. Examples of syntax highlighting in the pre-installed themes are shown below:

```

9  %%subheader<Definition>

10 The Riemann zeta function  $\hat{M}\langle zeta(s) \rangle$  is a function of a complex variable
 $\hat{M}\langle s = \sigma + i t \rangle$ . (The notation  $\hat{M}\langle s \rangle$ ,  $\hat{M}\langle \sigma \rangle$  and  $\hat{M}\langle t \rangle$  is used
traditionally in the study of the zeta function, following Riemann.)

11 For the special case where  $\hat{M}\langle \text{Re}(s) \text{cong } \sigma > 1 \rangle$ , the zeta function can
be expressed by the following integral:

12  $\hat{M}$ :  $zeta(s) = \frac{1}{\Gamma(s)} \int_0^\infty \frac{x^{s-1}}{e^x - 1} dx$ 
 $\text{\hspace{20pt}} \text{for } \text{Re}(s) \text{cong } \sigma > 1,$ 

```

(a) A code snippet highlighted using the “Light” theme

```

9  %%subheader<Definition>

10 The Riemann zeta function  $\hat{M}\langle zeta(s) \rangle$  is a function of a complex variable
 $\hat{M}\langle s = \sigma + i t \rangle$ . (The notation  $\hat{M}\langle s \rangle$ ,  $\hat{M}\langle \sigma \rangle$  and  $\hat{M}\langle t \rangle$  is used
traditionally in the study of the zeta function, following Riemann.)


11 For the special case where  $\hat{M}\langle \text{Re}(s) \text{cong } \sigma > 1 \rangle$ , the zeta function can
be expressed by the following integral:

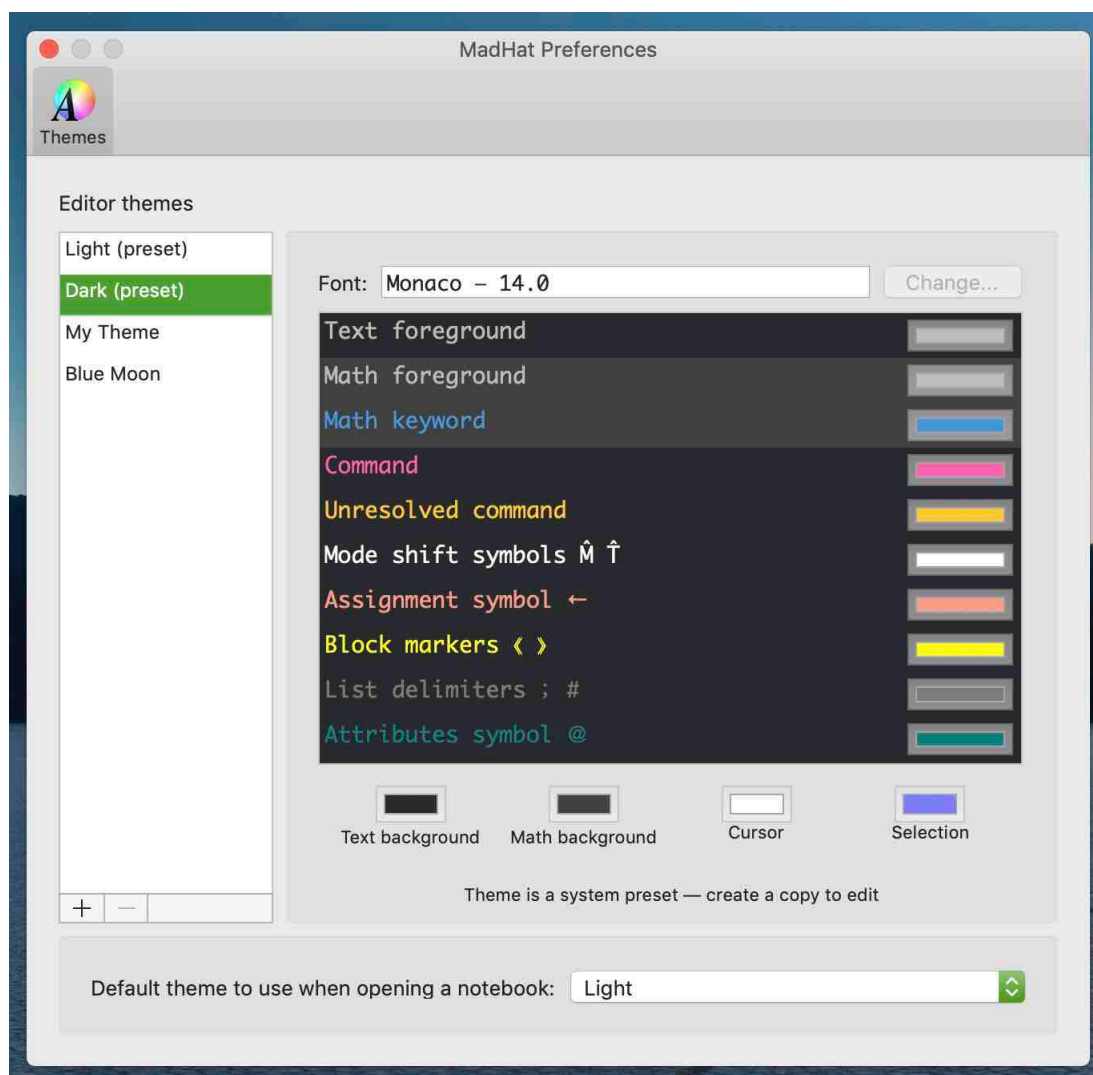
12  $\hat{M}$ :  $zeta(s) = \frac{1}{\Gamma(s)} \int_0^\infty \frac{x^{s-1}}{e^x - 1} dx$ 
 $\text{\hspace{20pt}} \text{for } \text{Re}(s) \text{cong } \sigma > 1,$ 

```


(b) A code snippet highlighted using the “Dark” theme









## The themes editor

The themes editor offers an intuitive interface to create and edit new themes. It is accessed through the  MadHat Preferences window, accessible using the standard Mac keyboard shortcut (“Command-,”), or from the File menu. Here is a screenshot of the themes editor:



## List of key substitutions

Here is a list of key substitutions implemented in the editor window to facilitate entering  MadHat's **special symbols**.

- The keystroke “\” is substituted with a command symbol .
- The keystroke Option-“\” produces an ordinary “\” character.
- The keystroke “\$” is substituted with a math mode shift command: , or  if the cursor is positioned at the beginning of a paragraph.
- The keystroke Option-“\$” produces an ordinary “\$” character.
- The keystroke “.” (an ordinary period) is substituted with a close command character  if it is entered when entering a command name is in progress.
- The keystroke “;” (a semicolon) is substituted with a primary list delimiter .
- The keystroke Option-“;” produces an ordinary “;” character.
- The keystroke “#” is substituted with a secondary list delimiter .
- The keystroke Option-“#” produces an ordinary ‘#’ character.
- The keystroke “[” is substituted with an “open block”  symbol.
- The keystroke Option-“[” produces an ordinary “[” character.
- The keystroke “]” is substituted with an “close block”  symbol.
- The keystroke Option-“]” produces an ordinary “]” character.

- The keystroke “%” is substituted with a comment symbol %.
- The keystroke Option-“%” produces an ordinary “%” character.
- The keystroke “@” is substituted with an attributes symbol @.
- The keystroke Option-“@” produces an ordinary “@” character.
- The keystroke Option-“/” produces an assignment symbol ←.


# MadHat version history and changelog

- Version 1.1.1 (April 20, 2022) additions:
  - Miscellaneous bug fixes
  - Added notebook configuration commands to customize the page geometry
- Version 1.1 (January 17, 2022) additions:
  - Miscellaneous bug fixes
  - Added a Preferences window with a new editor for syntax highlighting themes
  - Added an Export To LaTeX feature
  - Added notebook configuration options to allow customization of paragraph spacings, line spacing, and paragraph indent.
  - Improved formatting of boxes (content enclosed between `⌘begin box.` and `⌘end box.` commands)
  - Improved pagination algorithm for exporting notebooks to PDF; pagination can now occur between successive lines of a paragraph
  - Expanded and improved in-app help pages
- Version 1.0.2 (Dec. 4, 2021) additions:
  - Miscellaneous bug fixes
  - Improvements to in-app help pages
  - Added customization options for exported PDFs: page

headers, page footers, page numbers and more

- Added customization options for header and link styles and an option to define additional custom styles
- Improved support for macOS's Dark Mode
- Version 1.0.1 (Nov. 13, 2021) additions:
  - Bug fixes and improvements to app performance
  - Improvements to in-app help pages: help pages improved; clicking on code snippets copies code to the clipboard
  - Improvements to code templates
  - Improvements to PDF export feature: notebook intralinks are now exported correctly; exported PDF now includes a table of contents
- Version 1.0 (Nov. 3, 2021): initial release

# Index of commands and keywords

This page shows an automatically generated index of all  **MadHat** commands and math keywords. Click on a command name to go to its help page.

- Package **standard** (6 modules, 24 clusters, 381 commands):
  - Module **graphics** (5 clusters, 56 commands):
    - Cluster **graphics-graphics** (4 commands):
      - [⌘graphics canvas](#)
      - [⌘image](#)
      - [⌘madhat logo](#)
      - [⌘video](#)
    - Cluster **graphics-primitives** (28 commands):
      - [⌘frame](#)
      - [⌘filled frame](#)
      - [⌘filled stroked frame](#)
      - [⌘axes](#)
      - [⌘grid](#)
      - [⌘line](#)
      - [⌘polygon](#)
      - [⌘filled polygon](#)



- `⌘filled stroked polygon`
- `⌘marker`
- `⌘circle`
- `⌘disk`
- `⌘circle disk`
- `⌘arc`
- `⌘ellipse`
- `⌘filled ellipse`
- `⌘filled stroked ellipse`
- `⌘rectangle`
- `⌘filled rectangle`
- `⌘filled stroked rectangle`
- `⌘arrow`
- `⌘bezier`
- `⌘plot`
- `⌘parametric plot`
- `⌘polar plot`
- `⌘annotation`
- `⌘centered annotation`
- `⌘curved text layout`

- Cluster `graphics-penprimitives` (5 commands):

- `⌘move to`
- `⌘line to`
- `⌘curve to`
- `⌘pen forward`
- `⌘pen backward`
- Cluster **graphics-penformatting** (14 commands):
  - `⌘pen direction`
  - `⌘pen angle`
  - `⌘turn pen left`
  - `⌘turn pen right`
  - `⌘pen up`
  - `⌘pen down`
  - `⌘turn pen north`
  - `⌘turn pen south`
  - `⌘turn pen east`
  - `⌘turn pen west`
  - `⌘turn pen northeast`
  - `⌘turn pen northwest`
  - `⌘turn pen southeast`
  - `⌘turn pen southwest`
- Cluster **graphics-formatting** (5 commands):

- `\line thickness`
- `\marker type`
- `\marker scale`
- `\stroke color`
- `\fill color`
- Module **notebook** (1 cluster, 17 commands):
  - Cluster **notebook-notebook** (17 commands):
    - `\begin box`
    - `\end box`
    - `\box divider`
    - `\box frame thickness`
    - `\@notebook title`
    - `\@notebook author`
    - `\@exported page header`
    - `\@exported page footer`
    - `\@exported header and footer range`
    - `\@exported page number`
    - `\@define style`
    - `\@paragraph spacings matrix`
    - `\@`  
paragraph before and after spacings

- `⌘⌘top of page preparagraph spacings`
- `⌘⌘base paragraph spacing`
- `⌘⌘line spacing`
- `⌘⌘paragraph indent`
- Module `misc` (2 clusters, 20 commands):
  - Cluster `misc-misc` (11 commands):
    - `⌘⌘subscript`
    - `⌘⌘superscript`
    - `⌘⌘subsuperscript`
    - `⌘⌘multiscript`
    - `⌘⌘table`
    - `⌘⌘math table`
    - `⌘⌘matrix`
    - `⌘⌘continued fraction`
    - `⌘⌘left right line`
    - `⌘⌘left center right line`
    - `⌘⌘checkbox`
  - Cluster `misc-developer` (9 commands):
    - `⌘⌘debug`
    - `⌘⌘test expression`
    - `⌘⌘paragraph`

- `⌘fillertext`
- `⌘mhatsymbol`
- `⌘thatsymbol`
- `⌘commandsymbol`
- `⌘openblock`
- `⌘closeblock`
- Module **basic** (2 clusters, 13 commands):
  - Cluster **basic-basic** (5 commands):
    - `⌘space`
    - `⌘newline`
    - `⌘vertical skip`
    - `⌘glyph`
    - `⌘h`
  - Cluster **basic-info** (8 commands):
    - `⌘package info`
    - `⌘package author`
    - `⌘package modules`
    - `⌘package clusters`
    - `⌘package commands`
    - `⌘module commands`
    - `⌘cluster commands`

→ `⌘module clusters`

○ Module **math** (10 clusters, 205 commands):

▪ Cluster **math-functions** (11 commands):

→ `cos`

→ `sin`

→ `tan`

→ `sec`

→ `cosec`

→ `arccos`

→ `arcsin`

→ `arctan`

→ `exp`

→ `log`

→ `ln`

▪ Cluster **math-differentials** (49 commands):

→ `partial`

→ `dalpha`

→ `dbeta`

→ `dgamma`

→ `ddelta`

→ `depsilon`

- `dzeta`
- `deta`
- `dtheta`
- `diota`
- `dkappa`
- `dlambda`
- `dmu`
- `dnu`
- `dxi`
- `domicron`
- `dpi`
- `drho`
- `dsigma`
- `dtau`
- `dupsilon`
- `dphi`
- `dchi`
- `dpsi`
- `domEGA`
- `dAlpha`
- `dBeta`

- `dGamma`
- `dDelta`
- `dEpsilon`
- `dZeta`
- `dEta`
- `dTheta`
- `dIota`
- `dKappa`
- `dLambda`
- `dMu`
- `dNu`
- `dXi`
- `dOmicron`
- `dPi`
- `dRho`
- `dSigma`
- `dTau`
- `dUpsilon`
- `dPhi`
- `dChi`
- `dPsi`



→ `dOmega`

▪ Cluster **math-extensible symbols** (15 commands):

→ `\overbrace`

→ `\overbracket`

→ `\underbrace`

→ `\underbracket`

→ `\overparenthesis`

→ `\underparenthesis`

→ `\overtortoise`

→ `\undertortoise`

→ `\annotated equal`

→ `\annotated right arrow`

→ `\annotated double right arrow`

→ `\annotated left arrow`

→ `\annotated double left arrow`

→ `\annotated left right arrow`

→ `\annotated double left right arrow`

▪ Cluster **math-decorations** (8 commands):

→ `\hat`

→ `\tilde`

→ `\overdot`

→ `\double dot`

→ `\triple dot`

→ `\vector`

→ `\overbar`

→ `\underbar`

■ Cluster **math-greek** (48 commands):

→ `\alpha`

→ `\beta`

→ `\gamma`

→ `\delta`

→ `\epsilon`

→ `\zeta`

→ `\eta`

→ `\theta`

→ `\iota`

→ `\kappa`

→ `\lambda`

→ `\mu`

→ `\nu`

→ `\xi`

→ `\omicron`

- `pi`
- `rho`
- `sigma`
- `tau`
- `upsilon`
- `phi`
- `chi`
- `psi`
- `omega`
- `Alpha`
- `Beta`
- `Gamma`
- `Delta`
- `Epsilon`
- `Zeta`
- `Eta`
- `Theta`
- `Iota`
- `Kappa`
- `Lambda`
- `Mu`

- `Nu`
- `Xi`
- `Omicron`
- `Pi`
- `Rho`
- `Sigma`
- `Tau`
- `Upsilon`
- `Phi`
- `Chi`
- `Psi`
- `Omega`

■ Cluster `math-binary relations` (17 commands):

- `\leftarrow`
- `\rightarrow`
- `\uparrow`
- `\downarrow`
- `\leftrightarrow`
- `\doublerightarrow`
- `\doubleleftarrow`
- `\doubleleftrightharrow`

- `mapsto`
- `approx`
- `cong`
- `sim`
- `\elementof`
- `subset`
- `subseteq`
- `superset`
- `supseteq`
- Cluster `math-special` symbols (7 commands):
  - `\infinity`
  - `naturalnumbers`
  - `integers`
  - `rationals`
  - `reals`
  - `complexnumbers`
  - `hbar`
- Cluster `math-limits` operators (21 commands):
  - `\smallintegral`
  - `\bigintegral`
  - `\smallcontourintegral`

- `\bigcontourintegral`
- `smallsum`
- `bigsum`
- `\smallproduct`
- `\bigproduct`
- `\nabla`
- `\limit`
- `\minimum`
- `\maximum`
- `\infimum`
- `\supremum`
- `\determinant`
- `\integral`
- `\contourintegral`
- `\doubleintegral`
- `\tripleintegral`
- `sum`
- `\product`

▪ Cluster `math-binary` operators (21 commands):

- `intersection`
- `bigintersection`

- `union`
- `bigunion`
- `\conjunction`
- `\bigconjunction`
- `\disjunction`
- `\bigdisjunction`
- `times`
- `dividedby`
- `plus`
- `minus`
- `plusminus`
- `minusplus`
- `convolve`
- `dot`
- `fatdot`
- `centerdots`
- `verticaldots`
- `sedots`
- `nedots`
- Cluster `math-basic` `math` (8 commands):
  - `\fraction`

- `⌘quasifraction`
- `⌘binomial`
- `⌘squareroot`
- `⌘left bracket`
- `⌘right bracket`
- `⌘middle bracket`
- `⌘close bracket`
- Module **formatting** (4 clusters, 70 commands):
  - Cluster **formatting-color** (6 commands):
    - `⌘named colors`
    - `⌘color`
    - `⌘highlight color`
    - `⌘page background color`
    - `⌘box background color`
    - `⌘box frame color`
  - Cluster **formatting-style** (55 commands):
    - `⌘font size`
    - `⌘font`
    - `⌘edit font`
    - `⌘math font`
    - `⌘font cluster`



- `⌘bold on`
- `⌘bold off`
- `⌘bold text`
- `⌘italic on`
- `⌘italic off`
- `⌘italic text`
- `⌘underline on`
- `⌘underline off`
- `⌘underline`
- `⌘strikethrough on`
- `⌘strikethrough off`
- `⌘strikethrough`
- `⌘highlight on`
- `⌘highlight off`
- `⌘highlight`
- `⌘styled`
- `⌘lowercase`
- `⌘uppercase`
- `⌘redact`
- `⌘obfuscate`
- `⌘bold math`

- `\italic math`
- `\blackboard math`
- `\calligraphy math`
- `\fraktur math`
- `\mono math`
- `\sans math`
- `\roman math`
- `\suppress paragraph indent`
- `\new paragraph`
- `\page width`
- `\page height`
- `\page size`
- `\left margin`
- `\right margin`
- `\top margin`
- `\exported top margin`
- `\bottom margin`
- `\exported bottom margin`
- `\exported header offset`
- `\exported footer offset`
- `\header`

- `⌘subheader`
- `⌘subsubheader`
- `⌘paragraph header`
- `⌘superheader`
- `⌘hyperlink`
- `⌘intralink`
- `⌘help page link`
- `⌘command help link`
- Cluster `formatting-lists` (6 commands):
  - `⌘begin list`
  - `⌘end list`
  - `⌘list item`
  - `⌘num item`
  - `⌘checkbox item`
  - `⌘collapse here`
- Cluster `formatting-transitions` (3 commands):
  - `⌘pause`
  - `⌘next page in`
  - `⌘slide fragment`