Distributed Proofreaders
LaTeX Formatting Manual
August 2012

Contents

1 Overview **1**
- 1.1 The DP Process . . . 1
- 1.2 For New LaTeXers . . . 3

2 Guidelines **7**
- 2.1 Formatting Text . . . 7
- 2.2 Formatting Mathematics . . . 13
- 2.3 Aligned Material . . . 19

1 Overview

This manual contains guidelines for volunteers at Distributed Proofreaders who format projects using the typesetting language LaTeX. It's geared toward a variety of backgrounds: those who have proofed and formatted thousands of pages at DP but have always avoided LaTeX, long-time LaTeX users who are fairly new to DP and its ways, and (it is hoped) everyone in between.

In the interest of brevity, this manual discusses only well-standardized, project-independent formatting, and cautions mostly against common errors. Frequently, where formatting decisions are concerned, you're directed to external resources (especially, writing to `dp-feedback` or posting in the forums).

Section 1 is a very brief introduction to DP, LaTeX, and their distinctive interrelationship. Section 2 constitutes a user's guide and reference manual, meant both to be read for tutorial information and to be thumbed through as you format your first pages.

Both parts are written in mostly-independent sections, culled from existing DP wiki pages, forum posts, and private discussions. Do feel free to skip around as you read.[1] Important items (such as reminders to ask for human help) are repeated as necessary so they won't be overlooked by non-serial readers, and potential pitfalls are clearly marked.

1.1 The DP Process

A book's contents go through three major stages at Distributed Proofreaders: proofreading ("proofing"), formatting ("F1 and F2"), and post-processing ("PP"). Proofers make typographical corrections, so the raw text matches the page scan. Formatters add "tags" to specify how parts of the book will be presented visually. Proofing and formatting are *distributed* processes—performed one page at a time by multiple volunteers per book, possibly over a period of weeks or months.

[1] That is, to read sections non-consecutively.

The post-processor assembles the formatted content, ensuring consistency and coherency. After PPing, a book is ready for publication at Project Gutenberg (PG). By nature, PPing is not distributed; a single person must make stylistic decisions for the project as a whole.

Semantic and Visual Markup

A printed book contains meaningful typographical structures: page numbers, chapter titles, running heads, figure captions, and so forth. In an ebook, these structures can be encoded or *marked up* either "semantically" or "visually".

Semantic markup, or WYSIWYM ("what you see is what you mean"), separates *structural meaning* from *typographical appearance*. Instead of specifying the font weight and alignment of chapter titles (say), you mark each chapter with a dedicated "tag". The visual representation of chapters is controlled by modifying a single piece of information, the definition of the chapter tag. Other structures are marked up analogously.

By contrast, *visual markup*, or WYSIWYG ("what you see is what you get"), dictates a document's appearance element by element, via explicit, low-level formatting instructions ("this text is italicized", "this heading is centered small caps"). Word processors are WYSIWYG by default, and visual markup is many people's first inclination for document encoding.

Unfortunately, visual markup suffers from two Great Flaws: It scatters formatting instructions throughout the document instead of centralizing them (making consistency all but impossible to achieve), and discards structural information (disparate structures are tagged with the same markup simply because their printed appearance is similar). Each deficiency is fatal for easily-maintainable, high-quality ebooks of archival longevity.

Project-Specific Guidelines

Experience has shown that no single set of simple guidelines will adequately handle even a majority of LaTeX projects. Instead, every LaTeX project goes into the formatting rounds with a "working preamble", a set of project-specific macros designed to capture the semantic structure of the printed book.

The working preamble can be found on the project page. Before you start formatting, please peruse the working preamble and project comments, and take care to format using the macros supplied.

Macros in the working preamble are designed to minimize changes to the proofed text. Their names are short and mnemonic to reduce typing, and when possible they are simply wrapped around the proofer text to avoid errors caused by removing characters.

☞ **Caution**: Though preamble macros reproduce the typographical appearance of the original book reasonably closely, doing so is not their primary purpose. Indeed, many typographical decisions *cannot* be made until post-processing, when the font, text block size, and other global document parameters are known. As a formatter, you should not worry if the working preamble falls short of duplicating a book's typography exactly. (If the original typography is inconsistent, a brief `[** Note]` to the PPer is helpful.)

For example, if the book contains run-in "paragraph" headings in larger, boldface type, each followed by a quad space, look for a dedicated macro in the project preamble,

and use this macro consistently to code the headings. Do not explicitly code the large, bold font or the quad space. Such fine-tuning is elegantly and trivially handled in post-processing by adjusting the macro definitions.

☞ **Caution**: LaTeX's standard sectioning commands tend to be non-trivially integrated into the document class. Consequently, sectioning macros in the working preamble are usually *not* the standard `\chapter` or `\section` commands. For flexibility, DP projects instead use an "interface" layer to code a book's structures. When possible, macro names are capitalized variants of standard LaTeX commands, such as `\Chapter`, `\Section`, `\Paragraph`, and `\Tag`. Please take care to use these macros rather than the standard LaTeX equivalents.

1.2 For New LaTeXers

If you speak LaTeX, parts of this section will be superfluous. However, please skim for information specific the DP process.

What is LaTeX?

A small fraction of DP's output is formatted in "LaTeX", conceptually a markup language built on top of the TeX typesetting system. At DP, LaTeX is used almost exclusively for books containing a lot of mathematics.

DP's projects are marked up semantically. In HTML (the majority of DP's output), the PPer handles most semantic aspects of document preparation, while in LaTeX much of this burden falls to the formatters. As a LaTeX formatter, you need to intuit the nature of semantic markup and eschew most visual markup.

LaTeX projects are distributed by PG as a plain text "source file" and as a *compiled* PDF (portable document format) file. This very manual is written in LaTeX, and chances are you're viewing it as PDF.[2] The source file is analogous to the HTML code of an ordinary project, while the PDF corresponds approximately to the formatted text your web browser presents. Unlike HTML, a PDF file has fixed visual appearance (fonts, text block width, pagination, etc.), independent of how the file is previewed.

Instead of cascading style sheets code, a LaTeX file contains a *preamble*, where semantic tags and other style-determining information is defined or loaded from external packages.

Resources

If you don't speak LaTeX, printed manuals and tutorials are an absolute necessity. The DP LaTeX Resources Wiki page contains links to a number of well-known and widely-used references aimed at newcomers and more advanced users. There are over a dozen DP Wiki pages devoted to various aspects of LaTeX at DP. Please re-visit these pages periodically as you study and learn.

DP-specific manuals and training resources are also available. These include a large set of on-line worksheets that allow you to try your hand at formatting pages from real projects, and to check your work against that of an experienced formatter.

[2] Unless you're skipping around the room, holding a sheaf of papers.

There's no substitute for human feedback as you start to format LaTeX. For general questions, use the "Formatting LaTeX: 'All questions welcome' thread", in the DP forums under "Common Formatting Q&A". For project-specific issues, post in the project discussion. The LaTeX community is always pleased to help you advance your skills.

Don't hesitate to ask for feedback repeatedly, even if you speak LaTeX. LaTeX formatting entails numerous tasks, and it takes time and practice to habitualize them. Further, DP's requirements for simple, archival files entail special, non-obvious coding practices that differ significantly from "normal" LaTeX authoring. It's easier for you to receive several short messages, and easier for the pool of potential respondents to write them. It's also, frankly, better not to format dozens of pages only to find you're routinely omitting or mis-handling important code. The community considers mentoring you to be a worthwhile investment in the future!

Formatting Duties

Formatting in a DP non-LaTeX project amounts mostly to indicating chapters and sections with blank lines, marking changes of font with visual tags, and handling miscellaneous structures such as poetry, block quotes, and footnotes. Formatting tasks are similar in a LaTeX project, but more involved.

As a formatter, you'll work with text that's been carefully proofread, but from which the mathematical content is mostly absent. Greek letters and basic algebra should have been added in the proofing rounds, but other symbols and more complicated constructs will have been represented as `$$`. This is *not* a plain TeX display math delimiter, but a DP-specific convention for omitted text. You're expected to resolve every instance explicitly, either replacing it with LaTeX code, flagging it with a `[** F1: note]` for the post-processor, or both.

Chapter and sectional divisions are tagged (using macros rather than blank lines), font changes are marked (semantically if possible), footnotes are marked using the standard LaTeX `\footnote` command, and so on. In addition, please watch for and handle visually subtle details, such as non-breaking spaces, periods that do not end a sentence, and paragraphs starting immediately after a displayed equation. And, of course, there's mathematical formatting, ranging from individual symbols peppered throughout paragraphs of text, to complex aligned equations, and everything in between.

It's very helpful if you communicate in the project forums with other formatters to help ensure overall consistency. Please ask if you aren't sure how to handle a construct, or if you encounter a situation that will require the post-processor's attention but cannot be easily noted in the page text.

The plain truth: LaTeX formatting is complex, multifaceted, time-consuming, painstaking work, and requires a general understanding of what happens to projects in post-processing. An experienced LaTeX formatter *typically* requires 5–15 minutes to format one page in F1, depending on the amount of mathematics and complexity of the typography. When you start out, expect to spend 30 minutes or more per page, especially if you're new to LaTeX.

Despite the large time per page, LaTeX formatting is not impossibly difficult, but it does involve a substantial and extended effort to become fluent, even if you already speak LaTeX. If you enjoy challenging, detail-oriented typographical work, LaTeX will

make you very happy.

Your goal as a LaTeX formatter is not to create page code exactly matching the scanned image, but to express the semantic structure of the page simply, robustly, and flexibly, using preamble constructs. Strive for formatting that is

- *Complete*: Uniform handling of font changes, ties, numerals, periods on abbreviations, and mathematical markup.
- *Consistent*: Coordination of effort when multiple formatters work on a project, so that sectioning and other semantic constructs are handled the same way by everyone.
- *Correct*: Proper, semantic use of preamble commands, eschewal of non-LaTeX formatting, and machine-compilability of the page text.

As noted in the third point, LaTeX projects impose a requirement on formatters having no analog elsewhere at DP. Because formatted page code must be machine-compilable, you're expected to test-compile *every page you format*, to catch both syntax errors and output problems.

When you come across a page that's beyond your level of confidence, don't hesitate to "work around" it; keep the page out, format subsequent pages, and at the end of your session return the page to the round. It's polite in this event to post in the project forum, linking to the page in question. A more advanced formatter will handle the page. There is no shame in doing this; even the most innocent-looking project may contain pages to challenge a TeXnician.

Formatting work flow LaTeX formatting involves many disparate tasks. Developing careful, systematic work habits will both streamline your efforts and help you avoid unnecessary errors in the pages you submit.

When you download a page of a project for formatting, you are presented with proofed text in an editor window, just as with a non-LaTeX project. Your tasks are to (i) add the necessary formatting, (ii) test-compile the page, and (iii) submit your formatted text as "done".

If you use a stand-alone editor on your own computer to format LaTeX, take care to keep one particular version of the page text as the "master" copy. Add formatting and make corrections only in that copy. Cut and paste from the master copy into the test-compiled file, and if necessary, paste the master copy back into the formatting interface before submitting the code.

Source files Structurally, a DP LaTeX source file looks like this:

```
\documentclass[12pt]{book}
% Semantic definitions, a.k.a. preamble

\begin{document}
% Document text, a.k.a. body
\end{document}
```

The comment lines, starting with `%`, stand for the semantic formatting instructions (or *preamble*) and document contents (or *body*), respectively. For test-compiling (see below), you'll probably want to create a "wrapper" file with the project's working preamble pasted into the appropriate location, and the line `\input{project.tex}` after `\begin{document}`. (The file name `project.tex` is up to you, but will be assumed below.)

Test-compiling The code you submit as "done" is not a complete, compilable LaTeX source file, but merely the body. Your work flow must handle the incorporation of preamble code, but you must stringently avoid uploading preamble code when you check in the page. A simple method is to create a project-specific wrapper file as directed above. Each time you format a page, append the current page's text to the file `project.tex`, then compile the *wrapper file* and preview the output. The LaTeX formatting work flow page in the DP wiki contains suggestions on managing files.

If LaTeX encounters errors when you compile your page code, it will print a warning message and continue (if possible) or ask for your assistance. As needed, correct the master copy of your code and recompile. Repeat until there are no compilation errors, then preview the page to check that the output sufficiently resembles the original page scan.

Visual checklist Some visual differences between the page scan and your compiled code are certain to occur, because you're compiling a single page out of context. LaTeX may insert a paragraph indentation before the first word, whether or not a new paragraph begins there. If a numbered footnote appears, the number is almost certain to be wrong. Such discrepancies may be safely ignored.

In mathematics, don't spend a lot of effort duplicating alignment, especially if the need to align clearly depends on the width of the book's text block, or on the book's font. If capital-letter variables are set in an upright font, don't try to duplicate this by marking each symbol with an explicit font-changing command. (In such an event, however, *do* ask in the project forum, in case the PM or PP has special instructions.)

Do check carefully for common, basic, but easily-missed issues. Are there words in italics, boldface, or small capitals, and have you marked them (and any associated punctuation) appropriately? Have you faithfully reproduced paragraph breaks, particularly after displayed equations? Are there abbreviations, and have you properly handled their periods? Are authors' initials tied correctly? Is all the mathematics—including lone numerals—safely ensconced in math mode? Are footnote markers placed correctly, and the footnote text visible at the bottom of the page?

In mathematics, have you enclosed subscripts and superscripts in braces? Are the fractions properly displayed? Have you used commands for named functions? If there are integrals, have you added a thin space before the differential? Have you used `\left` and `\right` delimiters around complicated mathematical expressions?

Remember: You're aiming for a simple, flexible, *semantic* representation of the page, not a visual facsimile. The final presentation will be fine-tuned during post-processing. The more you rely on *meaningful* commands—and the more consistent your work is with that of others on the project—the easier a time the F2er and PPer will have.

With experience you'll develop a feel for acceptable and unacceptable discrepancies between your compiled output and the page scan, but do ask if you're unsure. In time, these checks will become second-nature, as will the entry of all that LaTeX code.

When, at long last, you're satisfied with your test-compiled output, copy the LaTeX code for the current page (being sure to omit any preamble or `\begin{document}` and `\end{document}` lines) back into the proofing interface and save it. Congratulations, and welcome to the addictive world of LaTeX typesetting!

2 Guidelines

If, while formatting, you encounter an issue not covered in these guidelines, or are otherwise uncertain how to handle something, please post your question in the project discussion thread. It's helpful if you provide a link to the scanned page image. In the unlikely event you're unable to resolve the issue, put a note in the proofread text explaining the problem. Your note will explain to the next formatter or post-processor what the problem or question is.

The semantic generalities of the ordinary DP formatting guidelines apply to LaTeX projects, and are not repeated here. Instead, this section highlights the differences between "ordinary" and LaTeX formatting at DP.

There is no formal requirement, but your LaTeX formatting experience will be more pleasant if you're familiar with the ordinary formatting guidelines, and are aware of what to look for when formatting.

On the flip side, LaTeX formatting differs from ordinary DP formatting in almost every detail. Be sure not to apply ingrained habits from non-LaTeX projects.

Command conventions Most LaTeX tags are *control sequences*, strings of characters following a backslash. In this manual, commands are printed in a fixed-width font: `\LaTeX`.

Many commands perform some action on a piece of formatter-provided text, called a (*command*) *argument* and enclosed in curly braces. For example, **boldface** is formatted `\textbf{boldface}`. The string `\textbf{}` signifies a command named `textbf` and accepting a single argument, which is enclosed in the braces.

A few common math commands accept two arguments, such as `\frac{}{}`, which typesets a fraction. The working preamble will contain project-specific commands, possibly accepting two or even three arguments; these are common for chapters, sections, and special structures such as theorems. The order of arguments is *always* significant.

2.1 Formatting Text

Special Characters

Latin-1 encoding LaTeX was written when ASCII input was the norm. Now, thanks to the inputenc package, characters from extended character sets can be digested as-is by LaTeX. Since DP uses the Latin-1 character set, proofread text should come with accented letters and some symbols intact.

Do not convert Latin-1 characters to LaTeX commands. The inputenc package will automatically handle most of them correctly.[3] Retaining the proofers' Latin-1 text makes the source file easier to read and reduces the likelihood of introducing errors.

On a Macintosh, you must set the document encoding to Latin-1 in your text editor while working on DP files. In TeXShop, go to "Preferences" and set "Encoding" to "Western (ISO Latin 1)". Neglecting to do so may have no visible effects as you edit, but could cause you to submit garbage instead of working code when you check the page back in.

Special characters The punctuation characters `#`, `%`, `$`, and `&` have special meanings in LaTeX. As a formatter you will never need the special meaning of `#`, but the other three are common:

`%` The comment character; causes LaTeX to ignore everything from the character to the end of the line.

`$` The "math mode" delimiter; used to surround small snippets of mathematics, such as $y = x^2$ or $\sqrt{\alpha}$ (coded `$y = x^{2}$` and `$\sqrt{\alpha}$`, respectively).

`&` The alignment stop; used in aligned constructs to delimit columns.

An omitted `$` or `&` usually has dramatic visual consequences when the page is compiled. If the compiled output suddenly goes haywire in the middle of a line or displayed equation, inspect the output to find the approximate location in the file where the trouble starts, and look for a missing character. TeXnicCenter, TeXShop, and `emacs` provide syntax highlighting, which helps avoid this type of error.

In text, special characters must by preceded by a backslash. "The 30% Solution, printed by Gutenberg & sons, $0.75" is formatted

```
The 30\% Solution, printed by Gutenberg \& sons, \$0.75.
```

A few other common characters have meanings distinctive to LaTeX. Curly braces "group" code, telling LaTeX to treat several characters as a single unit, as in x_{10} (coded `$x_{10}$`). Use `\{` and `\}` to get printed braces. The underscore and caret signify subscripts and superscripts, just as in non-LaTeX projects. The tilde character represents a non-breaking space, and is discussed at length below.

Comments Just as in a non-LaTeX project, you may need to leave in-line notes for the PPer. The working preamble provides the `\DPnote{}` command. Place the text of your note (or of any proofer notes) in the macro argument. Longer or traditionally-formatted notes can be left by commenging out the text of the `%[** Note]`.

Unless the project comments specify otherwise, minor typographical errors should be corrected in-line. Use, for example, `\DPtypo{txet}{text}` to indicate a misspelling. The first argument is the verbatim text from the page scan. The second argument is the corrected text. Any accompanying notes must be `\DPnote`d separately.

As a general matter, please ensure that every note you leave for the PPer contains the string "`**`", or is in a `\DPnote{}` command, or both.

[3] If the mid-dot (·) is typeset as Δ, post in the project forum. The project's formatting coordinator neglected to handle the character in the working preamble.

Dashes Four types of "dash-like" punctuation occur in LaTeX projects: hyphens, en-dashes, em-dashes, and minus signs. All are made with one or more dash characters, and the method differs from ordinary projects.

As in non-LaTeX projects, hyphens are made with a single dash, and the proofers should have rejoined words broken across lines in the scan.

Ranges of numbers, as in "pp. 24–27", are denoted with a clothed en-dash, made with *two* consecutive dash characters. Em-dashes—used to set off clauses—are created with *three* consecutive dashes. If a project contains a dash longer than this, add an em-dash and leave a note for the PPer. A consecutive string of four or more dashes is *never* acceptable.

Minus signs are made with a single dash character in "math mode": `$y = -x^{2}$` gives $y = -x^2$. Watch for mis-scanned and mis-proofed dashes, particularly in projects with dirty scans. An en-dash (–) and a minus sign (−) are about the same length, but neither semantically nor syntactically equivalent.

Non-breaking spaces LaTeX ordinarily breaks paragraphs into lines with great skill, and normally recognizes ends of sentences and adjusts the inter-word space accordingly. In a handful of situations, however, LaTeX needs human assistance. These low-key adjustments are easily forgotten, overlooked, and/or mis-handled, but unfortunately also fairly common.

Ideally, an author's initials and title, such as "P. D. Q. Bach" or "M. Butterfly", should not become separated from the last name by a line break (which may, in unlucky cases, be a page break). The tilde character ~, or *tie*, acts as a non-breaking space, forbidding LaTeX from ending a line there. As a rule, tie a single initial to the last name, and tie multiple initials to each other but *not* to the last name: `C.~F. Gauss`, `P.~D.~Q. Bach`, `M.~Butterfly`.

☞ **Caution**: The tilde character *is* a space character. Do not leave any additional space characters adjacent to a non-breaking space, and do not place a tilde at the end of a line. Doing so would both add unwanted space and allow LaTeX to break the line *next to* the tie.

Inter-word spaces LaTeX puts more space at the end of a sentence than it does between words. When a space or newline character follows a period, LaTeX uses a simple rule to decide whether a sentence has just ended: If the last character before the period was not a capital letter, the period ends a sentence. Sometimes this assumption is incorrect, in which case you must explicitly inform LaTeX.

Most commonly, LaTeX's guess fails because a period is part of an abbreviation. You, the formatter, must do one of two things. If a line break would be undesirable after the period, place a tie immediately after. This happens surprisingly often, see Vol. XLII of the *DP LaTeX formatting guidelines*, particularly Fig. 4 on p. 789:[4]

```
Vol.~XLII of ... guidelines, particularly Fig.~4 on p.~789
```

[4] As you may have noticed, facetiously inaccurate, yet wryly self-referential, remarks are not unusual in LaTeX documentation. While these *Guidelines* do not actually have 42 volumes, there is a serious lesson: Watch for abbreviations followed by an associated number, and tie them.

On the other hand, if a line break after the period would be fine, simply "escape" the following space with a backslash. This is also not unusual, see e.g. the 10 lb. 12 oz. appendix to the *Guidelines*:

```
see e.g.\ the 10~lb.\ 12~oz.\ appendix
```

Finally, LaTeX assumes a period following a capital letter marks an abbreviation, not the end of a sentence. When the sentence really *does* end there, the period must be preceded by `\@`. This looks awful, but is thankfully rare. Only in the U.S. Q.E.D. End of story.

```
Only in the U.S\@.  Q.E.D\@.  End of story.
```

Font Changes

Typesetters use different fonts and font sizes to signify information to the reader. A few of the most common semantic structures are discussed in the next item. This subsection handles the mechanics of changing fonts.

Just as in non-LaTeX projects, individual words, phrases, and whole blocks of text may be printed in a special font: *italicized*, **boldface**, SMALL CAPS, g e s p e r r t, and so forth. To the extent possible, you should strive to format such text according to its *meaning* rather than its *appearance*, using features of the project preamble.

Achieving consistent semantic markup may require communication between you and your fellow formatters, the project manager, and the post-processor (if any). Please read the project comments and project forum before starting to format, and watch for updates in the forum.

When the *font size* changes, the project preamble will normally contain a semantic command that effects the size change, such as a `Theorem` or `Remark` environment. Otherwise, ask in the forum or leave a note, such as `%[** smaller]`, for the PPer. Generally, it's enough to leave one note per page, but use your judgement as to what information would be most helpful. Similarly, gesperrt words should be handled with a project-specific macro. If none is provided, ask in the project forum.

Special fonts inside semantic units (such as italicized theorem text and boldface sectional headings) should be handled with a dedicated environment. Do not hard-code such font changes.

For book titles, single words representing defined terms, and other short snippets of italics, boldface, or small caps, hard-coding the font is normally acceptable. The commands `\textit{}`, `\emph{}`, `\textbf{}`, and `\textsc{}` accept one argument—placed inside the braces, and possibly spanning multiple lines—and render that argument italic, emphasized ("relatively" italic), bold, or small caps.

The "site convention" is to use `\emph` for terms being defined, and `\textit` for other italicized text, including foreign phrases and book titles.

☞ **Caution**: Projects containing "*i.e.*", "*e. g.*", and the like always provide macros: `\ie`, `\eg`, etc. Never hard-code the font or the spacing between characters.

☞ **Caution**: *Never* use the plain TeX font commands `\it`, `\em`, `\bf`, and `\sc`. Their syntax differs from that of their LaTeX analogs, and their behavior is not equivalent.

Illustrations, Lists, Footnotes, etc.

Front and back matter Unless the project comments say otherwise, much of books' front and back matter needn't be handled by formatters; the post-processor will format these according to the requirements of the project. In a typical project, ignorable elements include the title and copyright pages, and the table of contents. *Retain the proofed text*, but comment it out:

```
\iffalse
  [Text of page]
\fi
```

If both the text and the image are blank, leave a note `% [Blank Page]`

The project comments should say how to handle the index (if the project has one). Generally, mark any changes of font, comment out the text as above, and use indentations by two or four spaces to signify subitems just as in DP's non-LaTeX guidelines. Check carefully for I/l/1 scannos. Do not add `\item` commands.

Illustrations and thoughtbreaks Format these as in non-LaTeX projects, but comment your note: `% [Illustration: <text of caption>]` or `% <tb>`. If captions are printed in a special font you needn't note this, but please *do* format individual caption words if they're in a special font. See *Font changes* above for instructions on setting text in italics, boldface, or small caps.

Internal references If you encounter a page number that refers to the book itself, not to a citation to another work, ("see p. 42"), add a `%[** page]` note at the end of the line. Unless the project comments say otherwise, wrap references to equation numbers in `\Eq{}`: "... by equation `\Eq{(42)}`."

The need to mark other cross references depends on how much hyperlinking the PPer will do, and on how typographically regular the book's cross references are. References associated with a § symbol can be found easily in PP, as can phrases such as "Theorem V", "Fig. 6", and "Ch. 7" *when the original book uses such conventions consistently.* By contrast, it's helpful to note references such as "Theorems 6 and 7", or "Ch. IV" when most references are spelled out as "Chapter II". For such cases, leave a `%[** xref]` note at the end of a line. If several instances occur on a page, one `%[** xrefs]` note is fine. There's no harm in over-noting references, but it's also not important to note every single instance when references have the same form.

Block quotes Unles the project comments say otherwise, use LaTeX's built-in `quote` environment: Surround the quoted text with `\begin{quote}` and `\end{quote}`, leaving a blank line before and/or after as needed to give paragraph breaks. *Do not* match changes of font size or inter-line spacing (which the PPer will handle), but *do* watch for isolated words requiring special formatting.

Numbered lists LaTeX's `itemize` environment is used at DP to handle all list-like structures. The item number or tag is *hard-coded* as an optional argument (i.e. enclosed in *square* braces), as in `\item[42.]`. *Do not* use LaTeX's auto-numbering.

Not all numbered sequences are lists; ordinary paragraphs may be perfectly appropriate. Lists are indented from the left margin, and there is usually extra vertical space between successive items. In case of doubt, ask.

Footnotes Footnotes in LaTeX are placed *in line*, at the location of the footnote marker in the text. LaTeX's `\footnote{}` command creates a footnote, automatically generating the number or marker and setting the footnote text (the command argument) in the proper location. When you test-compile your page, the footnote marker is likely to be wrong. All will be fixed in PP.

Because of DP's proofing practices, formatting footnotes in LaTeX requires a modicum of care: You must cut the footnote text from the page bottom, discard the existing footnote marker, and paste the text inside a `\footnote{}` command elsewhere on the page. Be sure not to lose any of the footnote text, and take care to paste the footnote text at the correct location.

The proper technique is to put a newline after the `\footnote` command, and to indent the footnote text two spaces, namely,

```
to let the author ramble on.\footnote
  {This is the [several lines omitted] footnote text.}
The next sentence resumes here.
```

This convention introduces line breaks not present in the proofed text (an exception to the "preserve line breaks" rule), but results in easier-to-read source code.

Footnote markers sometimes appear within section titles, mathematical displays, and other typographical locations frowned upon nowadays. To handle such situations, you need to use separate commands for the footnote marker and footnote text. First, place a `\footnotemark` command at the location of the footnote marker. Then, *outside* the section title or displayed equation, put the footnote text inside the argument of a `\footnotetext{}` command.

```
y = x^{2}.\footnotemark
\] % This ends a displayed equation
\footnotetext{Heh. The mark looks just like an exponent.}%
The next sentence resumes here.
```

☞ **Caution**: Note carefully the comment character after the closing brace. If this is omitted, an inter-word space will be placed after the footnotetext. In the `\footnote` example that was no problem, because the footnote came just before an inter-word space. Here, however, a space will cause unwanted indentation of the next line. This problem is easy to miss in PP, so please try to avoid it.

Sections and Theorems

Sectional divisions Chapters, sections, and other sectional units should be marked using commands from the working preamble, *not* standard LaTeX commands. Typical usage is something like this:

```
In the Beginning  ->  \Section{In the Beginning}

Chapter 2: After  ->  \Chapter{2: After}
```

For regularity, place two blank lines before the title of a sectional unit.

It's fine if the test-compiled output of `\Chapter`, `\Section`, and other sectioning commands does not exactly match the original. *Do not* attempt to match the visual formatting by adding font commands or spacing.

Theorems and other semantic structures Mathematical texts frequently contain paragraphs typeset in a special font (usually italics), with a numbered heading (often boldface or small caps). When a particular type of structure, such as a theorem, example, problem, remark, or definition occurs repeatedly in a project, the working preamble will contain corresponding macros or environments. For example,

> § 123. THEOREM XIV: *A* red ball *is a red ball if and only if it is a ball and it is red.*

might be marked up like this:

```
\Paragraph{§ 123.} \begin{Theorem}[XIV]
A \emph{red ball} is...red.
\end{Theorem}
```

The project comments should explain how to code similar structures, but (as always) please ask in the project's discussion thread if you're unsure how to handle something.

2.2 Formatting Mathematics

LaTeX's distinctive strength is mathematical typesetting, and most (if not all) DP projects formatted in LaTeX contain a lot of mathematics. While LaTeX text formatting has quirks all its own, mathematical typesetting is where the Real Fun begins.

Like musical notation, mathematics is not exactly line-centric in the same way as ordinary text. The relative placement of symbols carries meaning, allowing enormous compression of information. Aesthetics and the desire for visual consistency impose subtle, arcane conventions on typeset mathematics. Compare:

```
 /b
 | f'(x)dx = f(b)-f(a)
/a
```

$S_a^b f'(x)dx = f(b)\text{-}f(a)$

$$\int_a^b f'(x)\,dx = f(b) - f(a)$$

Each may be serviceable, but only the third is beautiful. Over the course of a book, merely serviceable typesetting becomes tiresome to read, even an impediment to the reader's concentration and understanding.

The third equation works so much better than the other two for a variety of reasons. The fonts and symbols have similar, flowing shapes. The equality relation and binary operator have the same size, and the surrounding space is uniform. Conceptually distinct entities—the integral sign, the integrand $f'(x)$, and the differential dx—are subtly grouped, reinforcing their distinct but intertwined roles. And yet, LaTeX typeset this equation from simple, human-readable code:

```
\int_{a}^{b} f'(x)\, dx = f(b) - f(a)
```

The only explicit "hint" is the thin space `\,` before the dx.

You need not appreciate the fine points of mathematical typography in order to format, but will likely develop an awareness of such subtleties as the depth of your experience increases.

In-Line Math

In-line math is placed within *math mode*, enclosed by single dollar signs `$`. LaTeX ignores whitespace within math mode, so longer expressions may be formatted, even broken across several lines, to improve readability of the code.

☞ **Caution**: Though italicized letters and mathematical variables look superficially similar, compare "x" and "x", they should be carefully distinguished in markup: Neither the semantics nor visual appearance of text italics (*2a+3b=y, different*) and math italics ($2a + 3b = y$, $different$) are interchangeable.

Roman and Greek letters Roman variables are entered using the appropriate characters. Lowercase Greek letters are obtained with commands: a backslash, and the name of the letter spelled out. Capitalize the command name (`\Alpha` to `\Omega`) for a capital letter, see Table 1. (Greek capitals having identical appearance to a Roman capital do not in fact have LaTeX commands. If you need such a letter, ask in the project forum; a plain Roman letter may be fine.) Use a space if necessary to separate the name

A α `\alpha`	B β `\beta`	Γ γ `\gamma`	Δ δ `\delta`	E ϵ (ε) `\epsilon`	Z ζ `\zeta`
H η `\eta`	Θ θ (ϑ) `\theta`	I ι `\iota`	K κ ($\varkappa$) `\kappa`	Λ λ `\lambda`	M μ `\mu`
N ν `\nu`	Ξ ξ `\xi`	O o `\omicron`	Π π (ϖ) `\pi`	P ρ (ϱ) `\rho`	Σ σ (ς) `\sigma`
T τ `\tau`	Υ υ `\upsilon`	Φ ϕ (φ) `\phi`	X χ `\chi`	Ψ ψ `\psi`	Ω ω `\omega`

Table 1: Greek letters.

of a Greek letter from the surrounding expression; πr^2 is formatted `$\pi r^{2}$`

☞ **Caution**: Seven Greek letters have lowercase variants (in parentheses). The "ordinary" form will be seen when you test-compile, and should normally be used even if the scan looks more like the variant. In very rare cases a book may use both forms of a letter *with different meanings*. In this case only, precede the letter's name with "var" (e.g. `\vartheta` or `\varphi`) to obtain the variant letterform.

☞ **Caution**: As with italics, letters and symbols work differently in text and math mode. Compare 2+2=4 (text mode) and $2 + 2 = 4$ (math mode). Greek letters, Roman variables and arithmetic signs *must* be placed in math mode.[5] Unless the project

[5]However, LaTeX will *complain* only if Greek letters appear outside math mode.

comments or the PPer say otherwise, *do not* worry about matching upright letters used as variables.

☞ **Caution**: Numerals in a project may have mathematical meanings (constants and exponents) or non-mathematical meanings (chapter, section, and page numbers, for example). In some fonts, numerals in math mode behave differently than in text mode. Take care to place numbers in math mode or not, as appropriate.

Subscripts and superscripts A subscript is obtained with an underscore character followed by the text of the subscript enclosed in curly braces. Superscripts are entirely analogous, but use a caret instead of an underscore. Nowadays, subscripts and superscripts should be correctly and fully handled by the proofing rounds.

Occasionally, a subscript or superscript will itself have sub- or superscripts. In this event, treat the entire subscript or superscript as an expression:

z^{24}	x_{n+1}	$x^n + 1$	x_1^2	x_{n_k}
`z^{24}`	`x_{n+1}`	`x^{n} + 1`	`x_{1}^{2}`	`x_{n_{k}}`

Resolving proofer code Much of the raw text described above will have been entered and checked during the proofing rounds, but it doesn't hurt to verify existing code one more time. Further, you must place mathematics in math mode, and may have to disambiguate multi-character subscripts and superscripts with curly braces.

Your remaining responsibilities for mathematics are conceptually simple: Type in or otherwise format anything left incomplete by the proofers. A pair of dollar signs (`$$`) is a DP convention signifying a LaTeX-specific symbol or expression the proofers didn't handle. Every instance of `$$` must be handled explicitly by you, either replaced with LaTeX code (almost every case) or noted for the PPer.

Watch for fractions. In books of DP's era, fractions were often set upright ($\frac{1}{2}$) instead of slanted (1/2). However, the proofers are instructed to render fractions on a single line, with a slash denoting division. All instances of fractions should be resolved; detailed instructions appear below.

A Brief Command Miscellany

Most in-line math involves LaTeX commands, mnemonic control sequences for which you'll need a separate printed reference and plenty of practice. Dozens of these commands appear regularly, but you'll learn them surprisingly quickly.

One-character symbols One-character arithmetic signs have the expected meaning when you use the project preamble. These include plus ($+$), minus ($-$), times ($\times$ or $\cdot$), division ($\div$), equals ($=$), less-than ($<$), greater-than ($>$), and plus-or-minus ($\pm$).

The "times", "division", and "plus-or-minus" characters, and the degree sign °, may be found in the Latin-1 drop-down (or "pop-up") menu of the formatting interface. They're not ASCII characters, but can and should be left in place if present. It is common for proofers to use `\dd` to denote the partial derivative symbol ∂.

Frequently-encountered LaTeX symbol macros include "less than or equal" ($\leq$, `\leq`), "greater than or equal" ($\geq$, `\geq`), "square root" ($\sqrt{\ }$, `\sqrt{}`, whose argument is placed in the braces), "nth root" ($\sqrt[n]{\ }$, `\sqrt[n]{}`), and "infinity" (∞, `\infty`).

A few commands, such as `\sqrt{}`, accept an "optional argument", such as the n in `\sqrt[n]{}`. An optional argument is placed in *square* brackets, and generally causes a command to perform a variant of its normal function.

Fractions Fractions are typeset with the `\frac{}{}` command, which takes as arguments the numerator and denominator (top and bottom), respectively, as in $\frac{x}{y}$, `\frac{x}{y}`. LaTeX adjusts the typeset size of fractions automatically according to context, either "text style" for in-line occurrences or "display style" (see below).

☞ **Caution**: When you format a page containing fractions, check to be sure each instance is properly sized when the page is compiled. If LaTeX's default size does not match the scan, you may use the commands `\tfrac{}{}` (text fraction) and `\dfrac{}{}` (display fraction) to size a fraction explicitly. If LaTeX's judgement is correct, use `\frac{}{}`.

Named operators Named functions, appearing as Roman type in math mode, are obtained with identically-named commands. The trigonometric identity

$$\cos x + y = \cos x \cos y - \sin x \sin y$$

should be formatted `$\cos x + y = \cos x \cos y - \sin x \sin y$`, for example. Over a dozen such commands are provided, including the trig and hyperbolic functions, logarithms, and determinant.

Some books use older or variant functions not predefined in LaTeX, such as "cosec", "arccot", and "arccosec". The working preamble should supply any necessary operators. If a command is missing, please post in the project discussion.

☞ **Caution**: *Always* use semantic commands for named functions. They ensure the operators are typeset in the proper font and are surrounded by appropriate spacing.

Limit-like operators A few operators accept optional "subscripts": `\max` and `\min` (maximum and minimum), `\sup` and `\inf` (supremum and infimum), and `\limsup` and `\liminf` (limes superior and inferior). In displayed math, subscripts on these operators are underset: $\max_{0\le x\le 1} x(1-x) = \inf_{0<x}(x+\frac{1}{2})^2$.

Sums and integrals The commands for sums and integrals are `\sum` and `\int`. Lower and upper limits are specified as sub- and superscripts. Both are "large" operators, and like fractions and limits, their appearance is adjusted by LaTeX according to context. By way of illustration, the code

```
\lim_{b\to 1} \int_{0}^{b} e^{x^{2}}\, dx
    = \sum_{k=0}^\infty \frac{1}{(2k+1) k!}
```

gives dramatically different output in text and display style, respectively:

$\lim_{b\to 1} \int_0^b e^{x^2}\,dx = \sum_{k=0}^{\infty} \frac{1}{(2k+1)k!}$ or $$\lim_{b\to 1} \int_0^b e^{x^2}\,dx = \sum_{k=0}^{\infty} \frac{1}{(2k+1)k!}$$

Rarely, a limit or sum appears in-line, but the subscript is placed underneath, as if in display math. (See "Limit-like operators", above.) This effect is achieved with

the command `\limits`, which is placed between the `\lim` or `\sum` command and the subscript, as in `\lim\limits_{0\leq x\leq 1}...`

☞ **Caution**: The Greek letter Sigma (Σ) must *never* be used to denote a sum ($\sum$), despite superficial similarity of appearance. The summation sign behaves correctly with limits of summation, in both text and display, while Sigma doesn't.

Displayed Math

"Displayed" math refers to any typographical construct—such as an expression, equation, or group of equations—set off from the main body of the text. Ordinarily the typesetter displays material too complicated to print clearly in text, so on the average displayed math is more challenging to format than in-line math.

$$\begin{aligned}\int_0^{1/2} e^{-t^2}\,dt &= \sum_{k=0}^{\infty} \frac{(-1)^k}{k!} \int_0^{1/2} t^{2k}\,dt \\ &= \sum_{k=0}^{\infty} \frac{(-1)^k}{k!\,(2k+1)2^{2k+1}} = \frac{1}{2} - \frac{1}{3\cdot 2^3} + \frac{1}{2!\,5\cdot 2^5} - \cdots\end{aligned} \tag{42}$$

Everything said above about in-line math holds for displayed math as well. This sections covers a few salient additional details.

While in-line math is surrounded by dollar signs, displayed math is delimited by a pair of commands: `\[` and `\]`. It's good practice to put these commands on their own lines in the source file; visually, this "sets off" the equation code from the text:

```
...power series
\[
\exp(x) = \sum_{k=0}^\infty \frac{x^k}{x!}.
\]
```

☞ **Caution**: Take care never to leave a blank line before a math display (since this could result in a page break just before a displayed equation), and to leave a blank line immediately after a display if, and only if, the next sentence begins a new paragraph. Extra and missing blank lines around displayed equations are easily overlooked, and a common mistake in formatted code.

Large delimiters Large parentheses, square brackets, and curly braces arise regularly in displayed math. Preceding a delimiter character with `\left` or `\right`, as appropriate, causes LaTeX to make the symbol tall enough to enclose the material between the delimiters. The sizing is imperfect, but fine for the formatting rounds.

☞ **Caution**: Large delimiters in a displayed equation *must* occur in left-right pairs. Sometimes, however, an equation (including a single line of an aligned group) contains only one delimiter. This happens for two common reasons: (i) a large curly brace groups two or more equations, or (ii) a single delimited expression is broken across multiple lines. For such situations, LaTeX provides left- and right *null* delimiters, obtained with a period: `\left\{ [equation code] \right.`

Equation numbers Displayed equations may have an "equation number" (see equation (42) above), usually at the left or right margin, enclosed in parentheses, and used to refer to the equation elsewhere.

Although LaTeX can generate equation numbers automatically, this feature is unsuitable for use at DP.[6] Instead, hard-code equation numbers by wrapping a `\Tag{}` command around the proofed equation number, *including the parentheses*. This DP-specific command minimizes changes to the proofer text, and (unlike the standard AMS `\tag` command) handles math-mode tags and can be trivially used in post-processing to create link anchors.

Unless the project comments say otherwise, wrap in-line references to equation numbers in an `\Eq{}` command. The `\Eq{}` command can be trivially turned into a hyperlink in post-processing.

Don't worry about capturing the typography of the equation numbers (such as indentation and font).[7]

Miscellaneous Fine Points

Text in math mode The `\text` macro is used for ordinary text while in math mode, usually in display mode:

$$a_{ij} = 0 \quad \text{for} \quad i \neq j$$

would be coded `a_{ij} = 0 \quad\text{for}\quad i \neq j` (as a displayed equation). Sometimes text occurs in-line, but it would be semantically awkward to exit math mode. In such cases, it may be necessary to nest bits of math *inside* a `\text` command: {even integers x} would be coded `$\{\text{even integers~$x$}\}$`. Here, "even integers x" forms a logical unit, so the code above is preferable to the visually similar alternative "`$\{\text{even integers } x\}$`".

Spaces in math mode To zeroth order, "spaces do not matter in math mode". However, spaces deserve consideration, even in math. For example, a space or other non-letter must be used to delimit a macro name; πx is coded "`\pi x`", not "`\pix`".

More subtly, packages such as `icomma` turn punctuation into "active characters", effectively macros. If the `icomma` package is in use, "`$10,000$`" and "`$10, 000$`" are typographically distinct. For uniformity, it's therefore desirable to handle spaces consistently in all projects. The first-order rule is "do not leave a space after a decimal-separating comma": "`$10,000$`" is correct for DP work.

For spaces in algebraic expressions, there are principles of mathematical legibility but no hard-and-fast rules. Generally, prefer spaces around binary operators and relations (equality, less-than, etc.), after named operators, and between "large" multiplied expressions, particularly if there are no parentheses delimiting the factors. Omit spaces between single-character factors in a product except to delimit macro names. Thus:

```
ab + cd    a(x + y)    2 \frac{x}{y}    (a + b)^{2} \frac{c}{d}
```

In borderline cases, imagine yourself reading the source file for mathematical meaning.

[6] Automatic numbering is most useful when the document structure changes regularly, as when writing a book. At DP, by contrast, the document numbering is set in stone.

[7] You guessed it: These issues will be fixed in post-processing.

Consecutive math snippets Sometimes an author will concatenate mathematical phrases, as in "if $x > 2$, $x^2 > 4$" or "a_i, $i = 1, \ldots, n$". (Omitting the linking words "then" or "for" is arguably not good expository style, but at DP we don't edit the author's words.)

When you encounter this sort of construct, put each "phrase" in its own math mode:

```
if $x > 2$, $x^{2} > 4$          $a_{i}$, $1 = 1, \dots, n$
```

There are three reasons: space characters in text give ordinary inter-word space, while in math mode they're ignored; LaTeX can break a line at an inter-word space; and "separate" markup better captures the grammar. In extreme cases, you may need to understand the mathematics in order to code it properly; don't hesitate to ask in the project discussion.

2.3 Aligned Material

This section briefly introduces aligned constructs, lying toward the more challenging end of the LaTeX formatting spectrum: tabulars and arrays, and the AMS environments for displayed math.

Arrays LaTeX provides environments for rectangularly-arranged data: `tabular` for textual data and `array` for numerical data. These environments are generically termed "arrays" below. A complete explanation of LaTeX's alignment capabilities is beyond the scope of this short manual. Don't hesitate to ask for help and feedback on your first efforts, or even to skip pages containing rectangular alignments.

Generally, less is more in array formatting. As needed, the project preamble will contain macros tailored to the project. These are designed to be wrapped around pieces of the proofer text where possible. Concentrate on marking data and column headings correctly. Normally each array "cell" should contain one semantic unit, such as a decimal number, mathematical expression, or textual heading. Leave detailed alignment, spacing, and other visual tweaks to the post-processor.

If the printed original contains entries set on multiple lines, code the text in one cell, not in multiple rows. (A simple array is shown below, side-by-side with its formatted text. Note the markup of the second column heading, which uses a project-specific `\ColHead` macro.) Because OCR software and the proofing rounds leave text in its printed position, multi-line headings require careful rearrangement of the proofer text.

Environment	Data type
tabular	text
array	math

```
\begin{tabular}{|l|c|}
\hline
Environment & \ColHead{Data type} \\
\hline\hline
tabular & text \\
\hline
array & math \\
\hline
\end{tabular}
```

A `tabular` or `array` environment starts with an *alignment preamble*, a special argument (here, {`|l|c|`}) specifying the number of columns and how to align their entries: `l` for left, `r` for right, `c` for centered, and a few more advanced possibilities. The "pipe" character `|` signifies a *column separator*, a vertical line running the height of the structure. Separators do not count as columns; the example above has two columns, one left-aligned, the other centered. Separators may be omitted from the alignment preamble (as in {`|lc|`} if they are not desired. Consecutive pipes (`||`) may be used to get a separator consisting of closely-spaced vertical lines.

Data rows consist of multiple *entries*, one for each column, separated by `&`, the *alignment stop*. (Recall that the ampersand is a special character in LaTeX.) Each data row ends with the special command \\. (This command takes an optional argument that explicitly specifies vertical space; leave this sort of adjustment to the post-processor.)

Alignment stops appear only between columns, not at the beginning or end of the row; since the example has two columns, each row contains only one alignment stop. One or more entries of a data row may be omitted, with the obvious effect.

The special command `\hline` draws a horizontal line the width of the structure. An `\hline` command is *not* followed by a \\. Consecutive `\hline` commands may be used with the indicated effect. The command (e.g.) `\cline[2-7]` draws a horizontal line spanning columns 2 through 7 (assuming the array has at least seven columns).

☞ **Caution**: In the formatting rounds, don't explicitly adjust column or row spacing, even if the test-compiled result looks ugly. It's easiest if such changes are made globally in PP.

☞ **Caution**: If a project contains decimal-aligned numerical data, the project preamble will contain a special column type and instructions for handling it. Do not use double `rl`-aligned columns for the integer and decimal parts of data.

Multi-column cells The command `\multicolumn{2}{c}{Spud}` centers the text "Spud" over two columns. The first argument specifies the number of columns to span. The second argument may be `c`, `l`, or `r` (and may also include one or more pipes) for centered, left-aligned, or right-aligned material. If a book requires many such entries, the project preamble will contain one or more special macros that can be wrapped around the proofer text, e.g., `\TwoCol{Spud}`.

The command (e.g.) `\hdotsfor{5}` creates a row of dots spanning five columns of an array. It's fine if the dot spacing doesn't match the original.

The AMS Math Environments

Aligned displays Collections of displayed equations are usually centered as a group, or else aligned on their equals signs to highlight parallel structure or to show the progress of a calculation. Of all mathematical structures, complex displays require the most care and experience to format well, and when formatted correctly they're relatively likely *not* to mimic the page scan. This section is caution-heavy, to help ensure you're not wasting your time attacking difficult problems with plausible but poor techniques.

At DP, aligned displays are formatted using AMS environments, mostly `gather*`, `align*`, `alignat*`, and `multline*`. For detailed information, consult the *AMS*

Short Math Guide, which is linked from the DP wiki LaTeX resources page. Aa always, ask in the project discussion for feedback.

"Typical" aligned groups might look like this:

$$\begin{aligned} y_1(t) &= c_1 \cos \omega t + c_2 \sin \omega t, \\ y_2(t) &= -c_1 \sin \omega t + c_2 \cos \omega t. \end{aligned} \qquad \begin{aligned} v &= k_1 x_1 \ + \ldots + k_n x_n, \\ v &= l_1 x_1 \ + \ldots + l_n x_n, \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ w &= m_1 x_1 + \ldots + m_n x_n, \end{aligned}$$

```
\begin{align*}%[** Attn alignment]
  y_{1}(t) &=  c_{1}\cos \omega t + c_{2}\sin \omega t, \\
  y_{2}(t) &= -c_{1}\sin \omega t + c_{2}\cos \omega t.
\end{align*}
```

The ampersands just to the left of the equals signs delimit columns, alternately aligned flush right and flush left, signifying points in each equation to line up.[8] The code shown above does not perfectly duplicate the alignment shown in the typeset snippet, but final visual polishing is best noted (as indicated) and left to the PPer.

The second may be coded using `align*`, but can also be fine-tuned with `alignat*`:

```
\begin{alignat*}{3}
  v &= k_{1} x_{1} &&+ \dots &&+ k_{n} x_{n}, \\
  v &= l_{1} x_{1} &&+ \dots &&+ l_{n} x_{n}, \\
  \multispan{8}{\dotfill} \\
  w &= m_{1} x_{1} &&+ \dots &&+ m_{n} x_{n},
\end{alignat*}
```

Nested environments The nestable `aligned` and `gathered` environments are used to align or center parts of a complex mathematical display:

$$\begin{aligned} &(a+b)^2 + (a-b)^2 \\ &\quad = (a^2 + 2ab + b^2) + (a^2 - 2ab + b^2) \\ &\quad = 2(a^2 + b^2) \end{aligned}$$

```
\begin{multline*}
(a + b)^{2} + (a - b)^{2} \\
  \begin{aligned}
    &= (a^{2} + 2ab + b^{2})
     + (a^{2} - 2ab + b^{2}) \\
    &= 2(a^{2} + b^{2})
  \end{aligned}
\end{multline*}
```

(42)
$$\left\{ \begin{gathered} x = \sin\theta, \\ y = \sin 2\theta. \end{gathered} \right.$$

```
\[
\Tag{(42)}
\left\{
\begin{gathered}
x = \sin \theta, \\
y = \sin 2\theta.
\end{gathered}
\right.
\]
```

Using an `aligned` environment inside a `multline*` environment captures the semantics, and as a fringe benefit adjusts flexibly if the text block width changes. (Manually picking an alignment point in the first line does neither.)

[8] A row with two equation groups needs *three* alignment stops. Generally, a row containing n equation groups requires $(2n-1)$ stops.

Aligned displays get *considerably* more complex than this. Don't hesitate to ask an expert for help! Ironically, "simple" math (such as long division) is notoriously thorny to format.

☞ **Caution**: Never use LaTeX's `array` or `tabular` environments to format groups of displayed equations. Marking an aligned equation as if it were a table row does not work semantically *or* visually. Never use LaTeX's obsolete `eqnarray` environment.

☞ **Caution**: Flexible formatting adjusts itself automatically to the width of the text block. By contrast, plain visual formatting leads to brittle code, in which small changes of style parameters badly break the appearance. If you're spending a lot of time adding delicately-chosen explicit spaces, you're writing brittle code. Please stop and ask for help immediately. ☺

Condensed Intertext

Many projects of DP's era contain "condensed intertext", where text and aligned equations are placed on the same line, usually to save vertical space on the page.

The PPer has two general strategies: to "discard" or "retain". If the PPer is discarding condensed intertext (the more common option), format the text and equations as if they were set on different lines in the original. If the PPer is retaining (i.e., matching the original), use `DPalign*` and `DPgather*`, custom environments created by User: dcwilson. Do not use the standard AMS `flalign` environment.

Discarding condensed intertext Use ordinary equations or AMS environments:

Define
$$x = \frac{2t}{t^2+1}$$
and
$$y = \frac{t^2-1}{t^2+1}.$$

```
Define
\[
x = \frac{2t}{t^{2} + 1}
\]
and
\[
y = \frac{t^{2} - 1}{t^{2} + 1}.
\]
```

Retaining condensed intertext Use `DPalign*` and `DPgather*`. Note the explicit `\indent` required to mark the start of a paragraph.

Define $x = \frac{2t}{t^2+1}$
and $y = \frac{t^2-1}{t^2+1}.$

```
\begin{DPgather*}
\lintertext{\indent Define}
x = \frac{2t}{t^{2} + 1} \\
\lintertext{and}
y = \frac{t^{2} - 1}{t^{2} + 1}.
\end{DPgather*}
```

In all your LaTeX formatting at DP, remember to focus on semantics rather than raw appearance, keep your code simple and easy-to-read (future document maintainers are unlikely to speak LaTeX), and ask for advice or assistance if you're uncertain. Welcome to the community, and Happy LaTeXing!