

Author's Toolbox

Technical books are complex. A good technical book will be well structured and use a variety of layout devices to present the information in a way which aids comprehension. This chapter will demonstrate these devices. You'll be able to drop them into your own work in order to produce great content which readers will love.

This guide is designed explicitly to act as a sample chapter, and will conform to the rules it sets.

We're going to cover the following topics:

- Writing an introductory paragraph
- General text style
- Headings
- Paragraphs
- Lists
- Code samples
- Box outs
- Images

Writing an introductory paragraph

The overall point of an introduction is to make the reader understand why they should read on.

A good introduction:

- Is brief
- Relates the new topic to things the reader already knows
- Introduces the new topic/idea as a solution to a problem that the reader can understand
- Makes the reader understand why the topic is important and decide to read on
- Should avoid trying to define or explain the topic here. It is very hard to explain anything that the reader hasn't seen.
- Rarely needs more than three short paragraphs in addition to the bulleted list of the topics covered.

General text style

These are the general rules of style that will be applied everywhere in a Packt book:

- Use American English: color instead of colour, among instead of amongst, while instead of whilst.
- Footnotes are not used
- Latin abbreviations such as “etc.”, “e.g.”, “i.e.”, and “viz.” should not be used. Expressions such as “and so on”, “among others”, “for example”, “such as”, “that is”, and “namely” are preferred alternatives.

Structuring information with headings

Headings are maybe *the* most important device in your author's toolbox. They help you, and they help your reader. Generally you want to follow this scheme:

- Topics
- Tasks.
- Sub-tasks.

You never want to jump from a 1 to a 3. You can go from 1, to 2, to 3, and back to 1. But never jump from a topic to a sub-task.

Distinguishing your core topics

Firstly, you need to outline your core topics. A chapter should have 3-5 core topics which the reader can take away feeling confident they have a strong grasp of. These are the meta-level objectives of the chapter. For example, with this chapter each top level heading teaches a fundamental technique the reader needs to learn.

A book on beginner-level programming might have a chapter which covers basic variable information for example. Your top-tier topics could include:

- Understanding variables
- Assigning types
- Performing basic operations

By the end of a core task, or topic, the reader should have hit a key milestone and have created something they did not have before. By the end of the chapter, they should have created 3-5 things which add up to a roughly complete article.

The best chapters use headings to tell the story of the chapter at-a-glance. Your reader should be able to use your level 1 and 2 headings to understand the topic coverage, and maybe even learn some useful information from the topics alone. They should also know what they will be *doing* with that information. There are a number of devices you can use for this.

Organizing hierarchically

Now you've got your main topics, you think about the core steps needed to undertake that activity. This is where your level 2 headings come in. Level 2 headings are where you outline the *specific tasks* the reader will need to perform. They should be things the reader *does*.

Level 3 headings should be used when you've got lots of small tasks which will only take a paragraph or two to explain. Additionally, level 3 headings can be great when you've got loads of small items you want to list, but feel you need greater nesting.

Showing the reader what they will be doing

Each heading should guide the reader through what they will be doing with the chapter. Ideally, each chapter won't just be the topic name. You'll be telling the reader exactly what they will *do* with that section or sub-section. Let's say for the sake of argument, we're going to do a chapter on creating a complete website with HTML, CSS, and JavaScript.

- Creating your website structure with HTML
 - Structuring your content semantically with elements
 - Applying metadata
- Creating style with CSS
 - Customizing text elements
 - Coloring your website
 - Selecting specific sections with selectors
 - Manipulating specific elements with IDs and classes
- Creating interactivity with JavaScript
 - Programming fundamentals
 - Creating interactive functions.

It's telling a story, and showing the reader the progression. When using technical terms, no matter how basic, it's telling the reader in simple language what they *do* with those terms.

Here are a few basic principles:

- Tell the reader what they will have at the end of the section
- Avoid generic terms like “advanced”, or “basic”, or “more X with Y”.
- Use gerund form (“-ing” words)
- Use technical terms only if saying what you *do* with them

Sometimes it's not always easy to apply all these (especially using gerund: editors furrow many brows trying to get it to fit. Sometime it just isn't possible). Try to wherever possible. Imagine yourself telling the story of the chapter with your headings. You might have seen some fiction books that sum up what happens on a page with a sentence at the top. Use your headings like that.

Delivering information with paragraphs

Paragraphs are the key attribute of any book, technical or otherwise. If headings are your skeleton, paragraphs are the meat on the bone. A strong paragraph will convey one point of interest to the reader, explain it, and point the reader to the next point of interest. You don't want your paragraphs to become too fatty either though - you want them to be lean and tasty.

Structuring paragraphs

You should follow a very simple structure with each paragraph. You start by introducing a point that you want the reader to learn. If it's relevant, explain how they do something. You can explain it in as much detail as you need, and if possible give an example. Explain why it is worth knowing. Then point the reader to the next key point.

In summary, you should generally use this format:

- **Point** - What are they learning?
- **Instruct** - Tell the reader how to *do* something.
- **Explain** - Give details of how it functions.
- **Example** - When will this apply in their own work? Particularly stress why it's useful.
- **Signpost** - Where will they go next?
- You don't need to use all of these in a paragraph. But these are the kinds of information types you will be giving the reader.

Avoid complex sentences with multiple clauses. Nobody wants to read a paragraph – or what looks like a paragraph – which is really just one long sentence. Use clear, concise English. It's easier to write and easier to read.

Visually presenting paragraphs

Surely, if you're presenting written information, the words themselves should be enough? Why worry about what it looks like on the page? Information density in a paragraph is important. A dense paragraph, many sentences and lines long will lose your reader. A page of single sentences on a single topic is similarly disruptive, because it makes it difficult to distinguish the hierarchy of information. Many readers generally don't read things line by line, or word by word, but quickly take in the general meaning of a paragraph. By ensuring each paragraph covers one key point they're more likely to understand it.

A note about URLs: URLs are ugly and visually disturbing. They regularly disrupt how the reader processes a paragraph with a good deal of computer-output that the reader does not need to comprehend - just type in or click on. Put them following a paragraph, or in a box-out (see below).

Writing concise paragraphs

When writing a paragraph, start by writing a sentence summing up the point of the paragraph. Think to yourself, do you need any additional information, or does the sentence stand by itself? Think critically about how much you need to expand on a topic. Readers of technical books want clear, concise information that quickly conveys a point and moves onto the next subject.

When you've written a paragraph, critically dissect it again. Are any sentences tangential to the central point? Remove them. Do you repeat yourself in any? Remove them.

If you've got lots of little single sentence paragraphs, chances are you should be using bullet points. Which brings us to the next section...

Breaking information down with lists

Lists are powerful tools. They allow you to quickly and succinctly convey key points you want the reader to remember, or break down a set of instructions into steps.

Applying bullet points

Bullet points (also known as unordered lists) are one of the most powerful means of structuring information in your book. However, they should be used carefully, so as to present the information succinctly, and in a visual manner which aids comprehension and ease of reading.

Good bullet points are:

- **Succinct:** They should deliver a point you want the reader to remember, but not have any extended discussion.

- **Short:** Linked to succinctness, they should be no longer than 2, maybe 3 lines.
- **Consistent:** Follow a clear pattern of **item:** description.
- **Supporting:** Bullet points should aid comprehension by explaining points cleanly that might be lost in body text.
- **Supported:** Never have a section comprised purely of bullet points. Try to book end it with body text before and after.
- **Strategic:** Got a list of items substantially more than about 10? Break it down into sub-sections, and separate them with body text. Similarly, the bulk of your content should not be in bullet points.

Bullet points should never comprise the key area in which you will discuss concepts. Good bullet points are there to give the reader a clear and easy-to-read reference that stands out visually. You don't want to overload them with information. Body text is the key place for more detailed information.

Nesting bullet points

Never use more than 1 additional level of nesting – this disrupts the visual delivery, and confuse readers. You would be better off describing Item 1, then having list with one level of nesting only. Then you would describe Item 2.

Numbered lists

Numbered lists (ordered lists, or numbered bullets) are a superb tool for illustrating a sequence of steps. However, they should be used carefully. Consider the following:

- **Concise:** Anything more than roughly 10-15 steps should be broken down further using other structural elements further. The reader shouldn't be flicking through for pages wondering when the instructions will end.
- **Distinct:** Try not to combine numbered bullets with other layout elements, as it makes it harder for the reader to follow those steps.
- **Images:** These are fine, but each step should have only one image.
- **Paragraphs:** With numbered bullet points, you can use longer paragraphs, though try to avoid multiple paragraphs.

Determining what a “step” is

The key with numbered lists is to be careful about determining a new step against what is just a sub-unit of another step. Consider this example:

1. Click **Save**.
2. You have now saved your document.
3. Click **File**.
4. Click **Open**.
5. Select your file.
6. Click **Open**.

We don't really need individual clicks and what they do to be separate. In fact, if you're just describing a few simple mouse-clicks, do it all in one step. Make sure each step in a sequence of events is clearly distinct from the other steps in your numbered bullet points list. The above example could be changed to:

1. Click **Save**. This saves your document.
2. To open it again, simply click **Open**, select your file, and click **Open**.

Nesting numbered lists

If you've got a list of steps comprised of larger milestones and small instructions you can use nested numbered lists to organize the instructions. For example:

1. First, create a new customer profile, by clicking on **Create New Profile**
2. Fill out their **First Name** and **Surname** fields.
3. Select their location in the **Country** field.
4. Now, you need to assign them to a department. Click on **Assign Department**.
5. Select which department to assign them to from the dropdown field.
6. Next, assign them to an employee from the dropdown field.
7. Make sure **Notify Employee via Email** is checked.
8. Finally, click on **Save** to finalize the new customer profile.

The problem here is that there's no differentiation between the wider steps (creating a profile, assigning it to the right person, and saving it) and the smaller instructions to achieve those steps. Let's try organizing it differently:

1. First, create a new customer profile, by clicking on **Create New Profile**
 1. Fill out their **First Name** and **Surname** fields.
 2. Select their location in the **Country** field.
2. Now, you need to assign them to a department. Click on **Assign Department**.
 1. Select which department to assign them to from the dropdown field.
 2. Next, assign them to an employee from the dropdown field.
 3. Make sure **Notify Employee via Email** is checked.
3. Finally, click on **Save** to finalize the new customer profile.

As you can see, this clearly breaks things down between “larger”, meta tasks, and specific things (like “click on X”).

Describing code samples

Code samples are possibly the most important tool in your author's arsenal. However, the art of explaining code in books is a tricky one. Firstly, a few golden rules.

- Aim for a general max of about 20 lines.
- If you absolutely have to, go to 30 lines.
- If you write a page or more of code, your editor will ask for revisions.
- Break it down – the rule of thumb is to separate where you would normally comment.
- Avoid use in-code comments – they're a sign you need to write something outside the code sample about it!
- When you intend to describe a specific section in the sample highlight it.

Learning series exception: the exception to the 20:30 rule are Learning series books; if you're writing one, the first half of the book should really have 10 line samples maximum. You're dealing with much newer adopters to the technology, largely. You need to break it down further.

The sample

Let's look at a number of ways we could describe this fairly simple Python code sample. It's a very simple program that scans the input and detects the largest number and smallest number in that input.


```
numArray = [1, 5, 3, 12, 14, 75, 37, 83]
largestNumber = None
smallestNumber = None

print ("\nOur numbers are: ", numArray)

for num in numArray:
    if smallestNumber is None and largestNumber is None:
        smallestNumber = num
        largestNumber = num
    elif num > largestNumber:
        largestNumber = num
    elif num < smallestNumber:
        smallestNumber = num
    print ("Smallest Number:", smallestNumber, "Largest Number:",
largestNumber, "Current Number:", num)

print ("The largest number is:", largestNumber)
print ("The smallest number is:", smallestNumber)
```

Breaking it down

Just pasting this one in there, and explaining it in a paragraph underneath isn't going to work. A more advanced audience will probably get it, but the more beginner your audience, and the earlier in the book you are, the more you should break code samples down. You can always refer the reader to the sample available for download from Packt, or more preferably these days on a GitHub repo. The example's a little contrived, as it's designed to explain the maximum amount of information about the lines, to a basic reader.

We're going to try breaking this one down significantly now. Everything from this point is a mock example of explaining code.

The example

To start with, open up `sample.py`.

```
numArray = [1, 5, 3, 12, 14, 75, 37, 83]
largestNumber = None
smallestNumber = None
```

First, we declare our variables. `numArray` is our input. It houses each of the comma-separated integers (whole numbers) which our program will search for. Both `largestNumber` and `smallestNumber` hold the Python special type `None`. This special type allows you to set a null value for the variable. You'll see why we use this soon.

Let's start by displaying to the user what numbers Python will be processing.

```
print ("\nOur numbers are: ", numArray)
```

`print()` sends the item in brackets to the user display inside Python. It is a standard Python Built-In Function. Items inside quotation marks are strings.

The `\n` is an escape sequence. You'll notice that `\n` is not printed when executing the programme. What you instead notice is that it in fact creates a line break in the output. See Chapter X, Python Escape Sequences for more information.

Now, we begin the main task of the program: processing the input. For this we will be using a `for` loop. A `for` loop repeatedly executes either until the condition at the beginning of the statement is complete.

```
for num in numArray:

    if smallestNumber is None and largestNumber is None:
        smallestNumber = num
        largestNumber = num
    elif num > largestNumber:
        largestNumber = num
    elif num < smallestNumber:
        smallestNumber = num
    print ("Smallest Number:", smallestNumber, "Largest Number:", \
largestNumber, "Current Number:", num)
```

The first line states `for num in numArray`. This quite simply means that Python will run through each of the integers in the array called `numArray`, and for each of them will perform the commands after the colon.

We then indent to perform the first step. This is an `if` statement. This asks Python to check

whether or not the two variables `smallestNumber` and `largestNumber` contain the value `None`. If they do, then Python executes the steps indented following the colon. In this case, the variables are set to the value of the value of `num`.

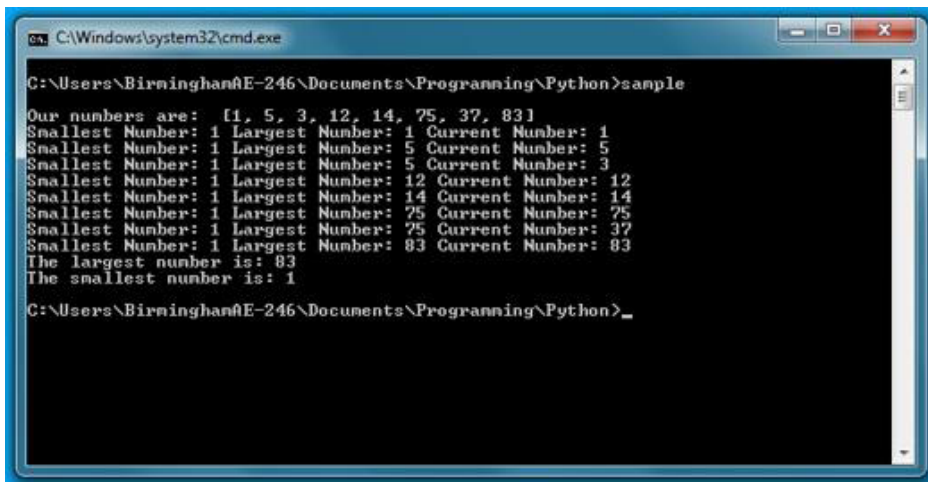
This only runs the first time. After that, we use `elif` which allows us to check if the current integer in this iteration is larger or smaller than the one held in `largestNumber` or `smallestNumber` respectively. If this is the case, Python then tells the program to change the values of `largestNumber` or `smallestNumber` to the new largest or smallest integers.

The final section of our `for` loop tells Python to tell the user what the current iteration has found. Having completed this iteration, the `for` loop resets, and checks again until the original conditions have been met.

With the logic complete, and the answer found, Python displays the final output:

```
print ("The largest number is:", largestNumber)
print ("The smallest number is:", smallestNumber)
The complete user output is:
```

Let's take a quick look at how this will appear on the screen:



```
C:\Windows\system32\cmd.exe
C:\Users\BirminghamE-246\Documents\Progranning\Python>sample
Our numbers are: [1, 5, 3, 12, 14, 75, 37, 831]
Smallest Number: 1 Largest Number: 1 Current Number: 1
Smallest Number: 1 Largest Number: 5 Current Number: 5
Smallest Number: 1 Largest Number: 5 Current Number: 3
Smallest Number: 1 Largest Number: 12 Current Number: 12
Smallest Number: 1 Largest Number: 14 Current Number: 14
Smallest Number: 1 Largest Number: 75 Current Number: 75
Smallest Number: 1 Largest Number: 75 Current Number: 37
Smallest Number: 1 Largest Number: 83 Current Number: 83
The largest number is: 83
The smallest number is: 1
C:\Users\BirminghamE-246\Documents\Progranning\Python>_
```

[0000OS_01_01.png]

And that's the completed explanation.

Explaining this sample

You'll notice this example uses a number of devices. Firstly, it heavily breaks down the code sample into its main logic blocks. It anticipates that the reader cannot look at the whole code sample and instantly understand it.

Key to explaining the code is linking up sections highlighted in the code which are of particular interest (using `CS - HighlightedCode`) with those explained in the body text. Using `CS - InlineCode` helps further link the samples and body text.

The most important thing is that you constantly evaluate each sample, and try to work out if you've explained the core concepts of that code sample before – or whether or not the target audience will understand it.

Box-outs

Sometimes you want to give the reader some extra information, outside the main body of the text. Packt has a couple of options for this.

Information box

These are great for giving the reader additional information. It marks it as useful to the topic at hand, but that they can possibly skip it. It's a reference.

Tip

Got a best-practice, useful link, or general tip or trick? Use one of these. You're telling the reader the information is optional, but they may find it useful.

Illustrating with images

Images should adhere to the following standards:

300 DPI

5.5 inches tall

5.5 inches wide

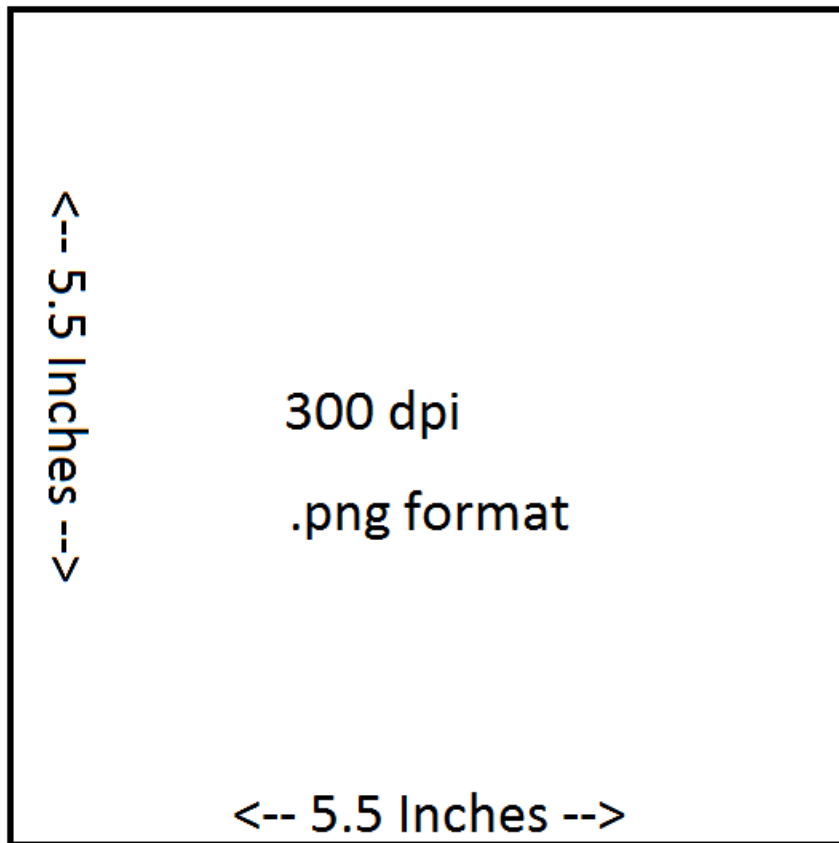
.png file format

Please use the following image file name format: `bookRef_bookNumber_imageNumber.png`

e.g.

`B12043_01_01.png`

Furthermore, we would strongly request that you write the filename information underneath the image, preferably laid out with **C - Comment**



B12043_01_01.png

Using images effectively

Good images generally:

- **Consider printing requirements:** Although most sales are eBooks, you need to consider that our printed copies are black and white. Try to use images that are lighter overall, with strong, dark and clear text.
- **However:** We do also provide the images as downloadable items, if there's no alternative.
- **Command Line Output:** This should be done using **CO - Console** rather than images.

We can redraw diagrams in house, if you would prefer to roughly sketch them out or use another placeholder indicating what you need us to draw. Please let us know when you submit a chapter which images you would like us to redraw.

