Pre-Class material
Introduction (2 min)
Self-editing (10 min)
Organizing large docs (20 min)
Illustrating (10 min)
Creating sample code (10 min)
Summary (1 min)

Creating sample code

On this page
Correct
Running sample code
Concise
Understandable
Exercise
...

Estimated Time: 10 minutes

Good sample code is often the best documentation. Even if your paragraphs and lists are as clear as blue water, programmers still prefer good sample code. After all, text and code are different languages, and it is code that the reader ultimately cares about. Trying to describe code with text is like trying to explain an Italian poem in English.

Good samples are **correct** and **concise** code that your readers can **quickly understand** and **easily reuse** with **minimal side effects**.

Correct

Sample code should meet the following criteria:

- Build without errors.
- Perform the task it claims to perform.
- Be as production-ready as possible. For example, the code shouldn't contain any security vulnerabilities.
- Follow language-specific conventions.

Sample code is an opportunity to directly influence how your users write code. Therefore, sample code should set the best way to use your product. If there is more than one way to code the task, code it in the manner that your team has decided is best. If your team hasn't considered the pros and cons of each approach, take time to do so.

Always test your sample code. Over time, systems change and your sample code may break. Be prepared to test and maintain sample code as you would any other code.

Many teams reuse their unit tests as sample programs, which is sometimes a bad idea. The primary goal of a unit test is to test; the only goal of a sample program is to educate.

A **snippet** is a piece of a sample program, possibly only one or a few lines long. Snippet-heavy documentation often degrades over time because teams tend not to test snippets as rigorously as full sample programs.

Running sample code

Good documents explain how to run sample code. For example, your document might need to tell users to perform activities such as the following prior to running the samples:

- Install a certain library.
- Adjust the values assigned to certain environment variables.
- Adjust something in the integrated development environment (IDE).

Users don't always perform the preceding activities properly. In some situations, users prefer to run or (experiment with) sample code directly in the documentation. ("Click here to run this code.")

Writers should consider describing the expected output or result of sample code, especially for sample code that is difficult to run.

Concise

Sample code should be short, including only essential components. When a novice C programmer wants to learn how to call the `malloc` function, give that programmer a brief snippet, not the entire Linux source tree. Irrelevant code can distract and confuse your audience. That said, never use bad practices to shorten your code; always prefer correctness over conciseness.

Understandable

Follow these recommendations to create clear sample code:

- Pick descriptive class, method, and variable names.
- Avoid confusing your readers with hard-to-decipher programming tricks.
- Avoid deeply nested code.
- Optional: Use bold or colored font to draw the reader's attention to a specific section of your sample code. However, use highlighting judiciously—too much highlighting means the reader won't focus on anything in particular.

Exercise

Which of the following would be a more helpful line of code in a sample program? Assume that the target audience consists of software engineers new to the `go.so` API.

1. `MyLevel = go.so.Level(5, 28, 48)`
2. `MyLevel = go.so.Level(rank=5, 28, 48)`
3. `MyLevel = go.so.Level(rank=5, dimension=28, opacity=48)`

Click the icon to see the answer.

Answer **3** is the best choice here. Although it is tempting to keep sample code as short as possible, omitting parameter names makes it harder for novices to learn.

Commented

Consider the following recommendations about comments in sample code:

- Keep comments short, but always prefer clarity over brevity.
- Avoid writing comments about *obvious* code, but remember that what is obvious to you (the expert) might not be obvious to newcomers.
- Focus your commenting energy on anything non-intuitive in the code.
- When your readers are very experienced with a technology, don't explain *what* the code is doing, explain *why* the code is doing it.

Should you place descriptions of code inside code comments or in text (paragraphs or lists) outside of the sample code? Note that readers who copy-and-paste a snippet gather not only the code but also any embedded comments. So, put any descriptions that belong in the pasted code into the code comments. By contrast, when you must explain a lengthy or tricky concept, you should typically place the text before the sample program.

Note: If you must sacrifice production readiness in order to make the code shorter and easier to understand, explain your decisions in the comments.

Exercise

What problems do you see in the comments within the following snippet? Assume that the code is aimed at programmers who are new to the `br` API but who have some experience with the concept of streams:

```
/* Create a stream from the text file at pathname /tmp/myfile. */
mystream = br.openstream(pathname="/tmp/myfile", mode="z")
```

Click the icon to see the answer.

The comments contain the following flaws:

- The comment elaborates on a fairly obvious part of the code.
- The snippet doesn't explain the non-obvious portion of the code. Namely, what is the `mode` parameter and what does a value of z mean?

Reusable

For your reader to easily reuse your sample code, provide the following:

- All information necessary to run the sample code, including any dependencies and setup.
- Code that can be extended or customized in useful ways.

Having easy-to-understand sample code that's concise and compiles is a great start. If it blows up your reader's app, though, they won't be happy. Therefore, when writing sample code, consider any potential side effects caused by your code being integrated into another program. Nobody wants insecure or grossly inefficient code.

The example and the anti-example

In addition to showing readers *what to do*, it is sometimes wise to show readers *what not to do*. For example, many programming languages permit programmers to place white space on either side of the equals sign. Now suppose that you were writing a tutorial on a language (such as bash) that does not permit white space on either side of the equals sign. In this case, showing both a good example and an anti-example will benefit the reader. For example:

```
# A valid string assignment.
s="The rain in Maine."
```

```
# An invalid string assignment because of the white space on either side of the
# equals sign.
s = "The rain in Maine."
```

Sequenced

A good sample code set demonstrates **a range of complexity**.

Readers completely unfamiliar with a certain technology typically crave simple examples to get started. The first and most basic example in a sample code set is usually termed a Hello World program. After mastering the basics, engineers want more complex programs. A good set of sample code provides a healthy range of simple, moderate, and complex sample programs.

Exercise

Which of the following would be a good set of sample functions to support a tutorial introducing newcomers to the concept of functions?

1. The following set of functions:
   a. A function that takes no parameters and doesn't return anything.
   b. A function that takes one parameter but doesn't return anything.
   c. A function that takes one parameter and returns one value.
   d. A function that takes three parameters and returns one value.
2. The following set of functions:
   a. A function that takes three parameters and returns one value.
3. The following set of functions:
   a. A function that takes one parameter and returns one value.
   b. A function that takes three parameters and returns one value.

Click the icon to see the answer.

The best answer is **1**. Providing samples that cover a range of complexity is usually the wisest choice—particularly for newcomers. Resist the temptation to *rush* towards very complex sample programs, bypassing the beginner and intermediate sample programs that newcomers crave.

What's next?

Congratulations: you've completed the pre-class work for Technical Writing Two.

If the in-class portion of Technical Writing Two is available in your organization, please take it. If you'd like to facilitate the in-class portion of Technical Writing Two, see the facilitator's guide.

A quick compilation of the topics covered in Technical Writing Two is available on the Summary page.

Was this helpful?

Send feedback

Based on this session

Words

Educational resources about technical writing.

Updated Nov 8, 2021

Paragraphs

Educational resources about technical writing.

Updated Feb 20, 2021

Free technical writing courses

Educational resources about technical writing.

Updated Nov 19, 2021

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 4.0 License, and code samples are licensed under the Apache 2.0 License. For details, see the Google Developers Site Policies. Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2021-11-17 UTC.