**Example: Filter()**
The main idea is that the word "filter" in programming can mean either picking things or getting rid of them. To make it clearer, it's better to use names like "select()" when you're choosing things and "exclude()" when you're removing them. This helps people understand the code better and prevents mistakes.

**Example: Clip(text, length)**
This passage shows how it's really important to have clear and easy-to-understand names when you're writing code. For example, instead of calling something "Clip," calling it "Truncate" makes it clearer. Also, instead of just saying "length," saying "max_chars" is more specific. Adding units to names, like "chars" in "max_chars," helps make things even clearer, so everyone understands what's going on in the code.

**Prefer min and max for (Inclusive) Limits**
This part shows why it's really important to give things clear names when you're writing code. For instance, if you name something in a confusing way, it can cause mistakes and make things hard to understand. But if you use clear prefixes like "max_" or "min_" when you're naming limits, such as "MAX_ITEMS_IN_CART," it makes your code easier to read and less likely to have errors. So, it's super important to pick good names when you're writing code to make sure it works well and is easy to keep up with.

**Prefer first and last for Inclusive Ranges**
The passage talks about how names for parameters can be unclear, especially when it comes to saying if a range includes its endpoints. It recommends using terms like "first" and "last" or "min" and "max" to make things clearer. Basically, it's saying that it's important to name things clearly, especially when they control what's included in a range.

**Prefer begin and end for Inclusive/Exclusive Ranges**
This passage talks about how it's useful to use ranges that include both the start and end points when programming, like when you want to print events that happened during a certain time. It says it's easier to understand and write code this way. It suggests calling the start and end points "begin" and "end" even though "end" can be a bit unclear. It says even though English doesn't have a perfect word for "just past the last value," using "begin" and "end" is still the best choice because it's what people are used to in languages like C++.

**Naming Booleans**
The main points are about naming things clearly when it comes to boolean variables and functions that give either true or false answers. If names are unclear or misleading, it can make it hard to understand what's really true or false. Using words like "need," "is," "has," "can," or

"should" can make intentions clearer. Also, it's better to avoid using negative words in names because it makes things easier to read and understand. For example, instead of saying "read_password," it's better to say "need_password," and instead of "disable_ssl," it's clearer to say "use_ssl."

**Matching Expectations of Users**
The main idea is that using clear and consistent names in programming is really important. If names don't match what they actually do, it can cause confusion and make the code run slower. But if developers name things accurately, it can help avoid confusion and make the code easier to understand and maintain.

**Example: Evaluating Multiple Name Candidates**
The passage talks about choosing a name for a feature in a config file. This feature lets one experiment reuse stuff from another. It suggests four names: template, reuse, copy, and inherit. It discusses the advantages and disadvantages of each. "copy_experiment" and "inherit_from_experiment_id" are seen as the best names because they explain the function well and are less likely to be misunderstood.