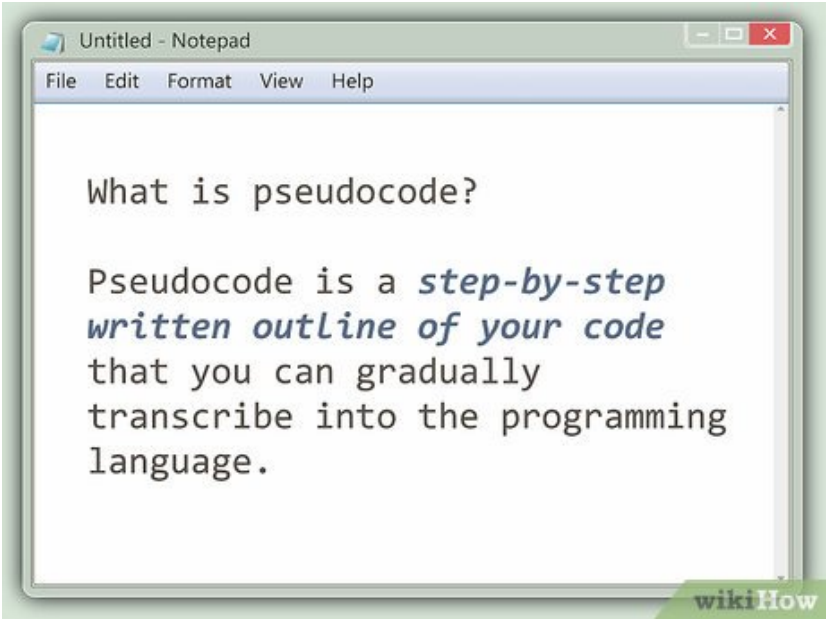
How to Write Pseudocode

This wikiHow teaches you how to create a pseudocode document for your computer program. Pseudocode essentially entails creating a non-programming language outline of your code's intent.https://whatis.techtarget.com/definition/pseudocode...

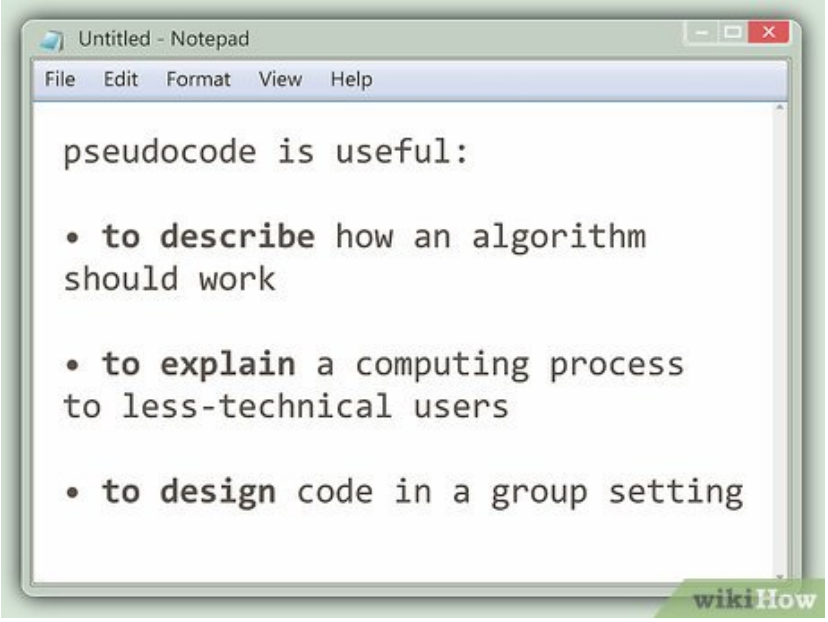
Part 1 of 3:

Understanding Pseudocode Basics

1. 

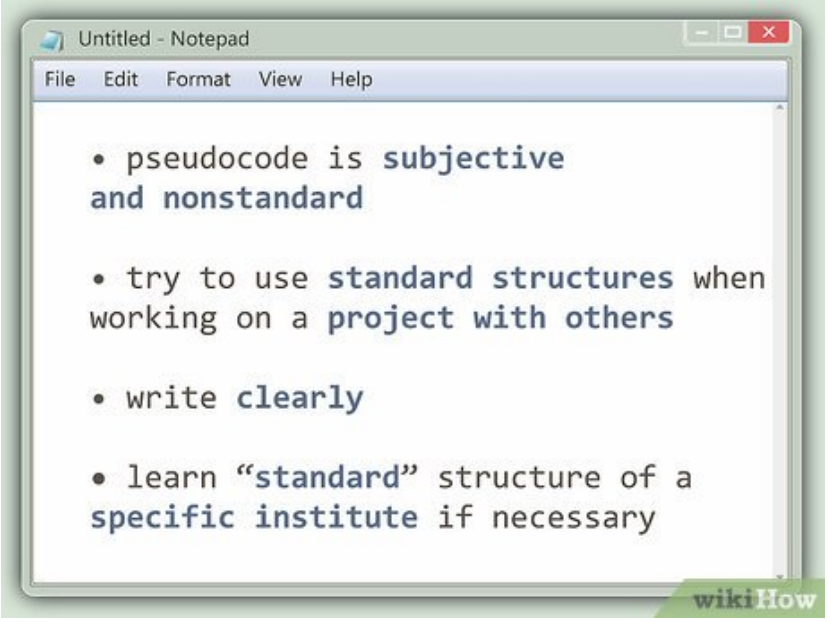
Know what pseudocode is. Pseudocode is a step-by-step written outline of your code that you can gradually transcribe into the programming language. Many programmers use it to plan out the function of an algorithm before setting themselves to the more technical task of coding.

Pseudocode serves as an informal guide, a tool for thinking through program problems, and a communication option that can help you explain your ideas to other people.

2. 

Understand why pseudocode is useful. Pseudocode is used to show how a computing algorithm should work. Coders often use pseudocode as an intermediate step in programming in between the initial planning stage and the stage of writing actual executable code. Some other uses of pseudocode include the following:

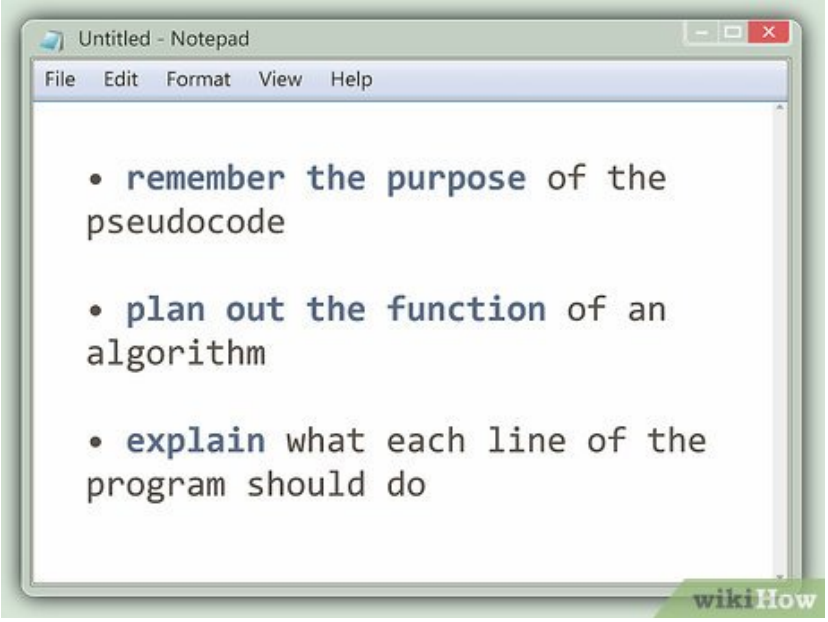
1. Describing how an algorithm should work. Pseudocode can illustrate where a particular construct, mechanism, or technique could or must appear in a program.
2. Explaining a computing process to less-technical users. Computers need a very strict input syntax to run a program, but humans (especially non-programmers) may find it easier to understand a more fluid, subjective language that clearly states the purpose of each line of code.
3. Designing code in a group setting. High-level software architects will often include pseudocode into their designs to help solve a complex problem they see their programmers running into. If you are developing a program along with other coders, you may find that pseudocode helps make your intentions clear.

3. 

Remember that pseudocode is subjective and nonstandard. There is no set syntax that you absolutely must use for pseudocode, but it is a common professional courtesy to use standard pseudocode structures that other programmers can easily understand.[2] If you are coding a project by yourself, then the most important thing is that the pseudocode helps you structure your thoughts and enact your plan.

1. If you are working with others on a project—whether they are your peers, junior programmers, or non-technical collaborators—it is important to use at least some standard structures so that everyone else can easily understand your intent.
2. If you are enrolled in a programming course at a university, a coding camp, or a company, you will likely be tested against a taught pseudocode "standard". This standard often varies between institutions and teachers.

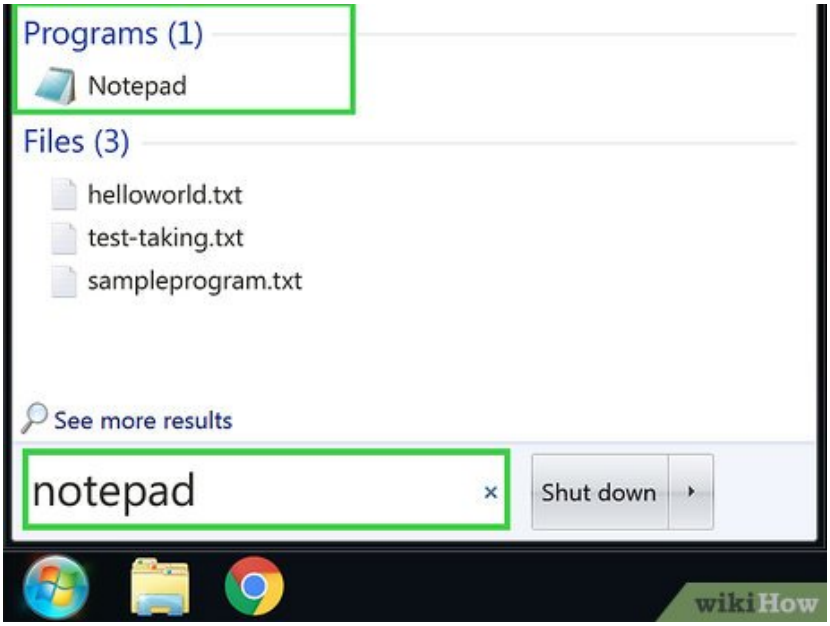
Clarity is a primary goal of pseudocode, and it may help if you work within accepted programming conventions. As you develop your pseudocode into actual code, you will need to transcribe it into a programming language – so it can help to structure your outline with this in mind.

4. 

Focus on the main purpose of pseudocode. It can be easy to revert to writing in code once you hit your stride. Remembering the purpose of your pseudocode—explaining what each line of the program should do—will keep you grounded while creating the pseudocode document.

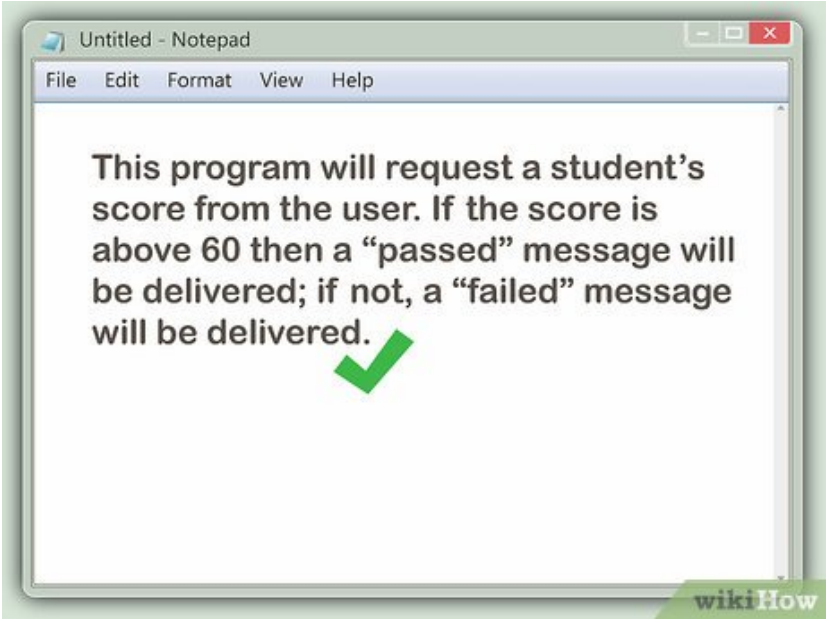
Part 2 of 3:

Writing Good Pseudocode

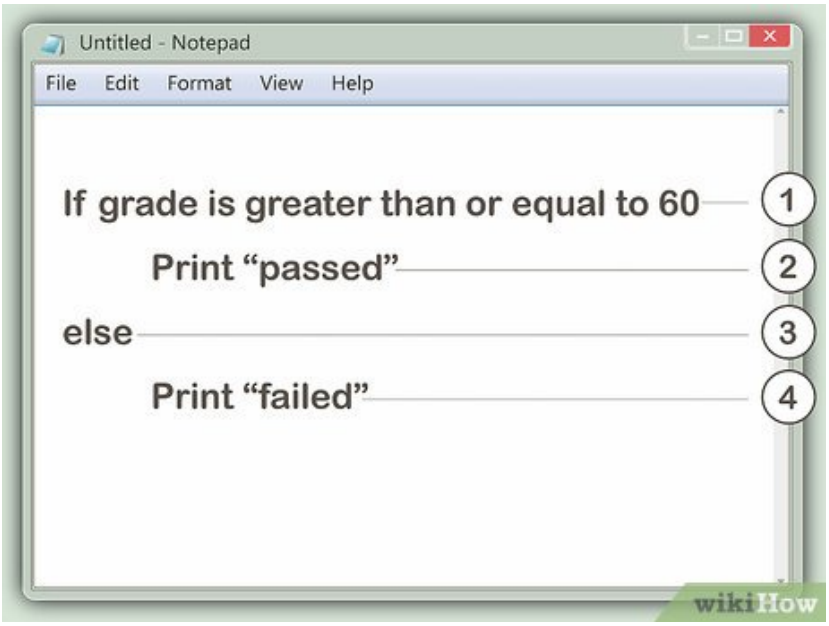
1. 

Use a plain-text editor. It can be tempting to use a word processor (e.g., Microsoft Word) or a similar program to create a rich-text document, but pseudocode needs as little formatting as possible to keep it simple.

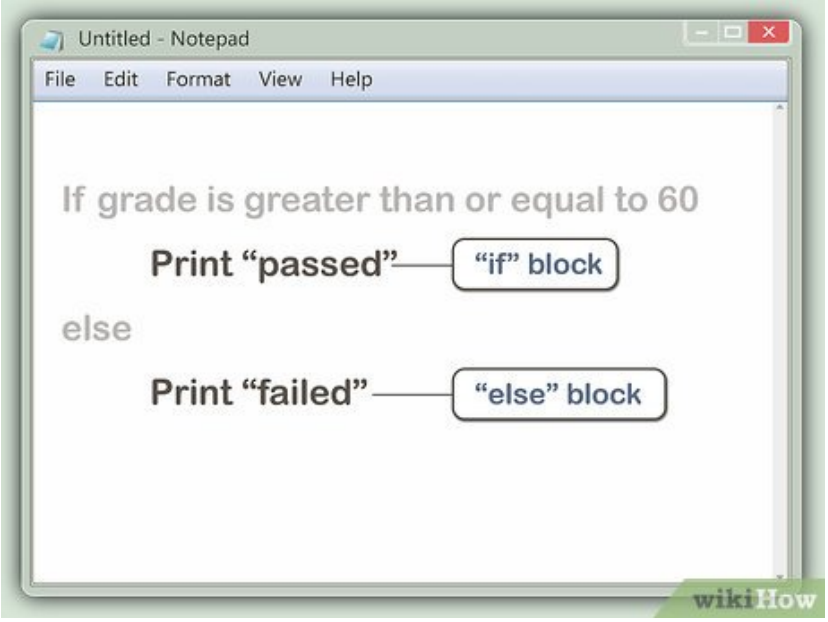
Plain-text editors include Notepad (Windows) and TextEdit (Mac).

2. 

Start by writing down the purpose of the process. Dedicating a line or two to explaining the purpose of your code will help set up the rest of the document, and it will also save you the task of explaining the program's function to each person to whom you show the pseudocode.

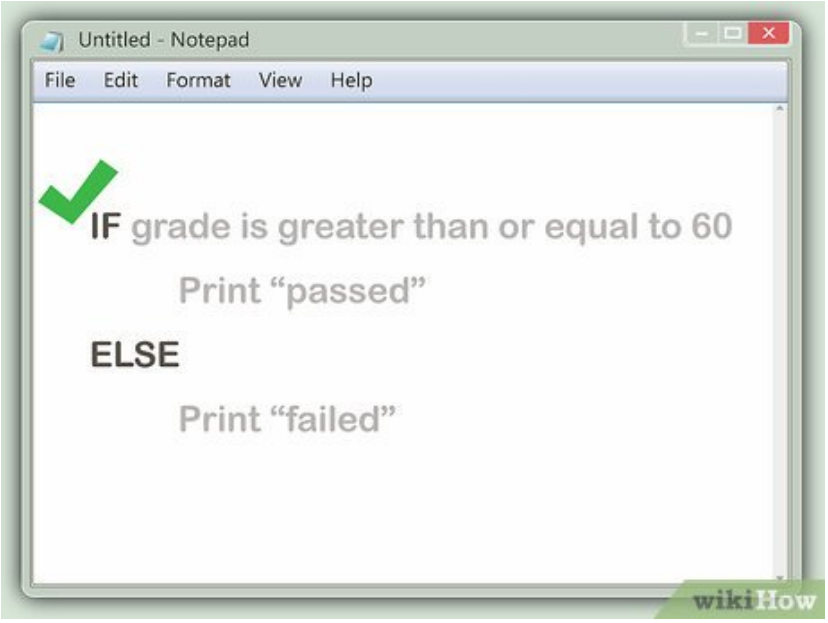
3. 

Write only one statement per line. Each statement in your pseudocode should express just one action for the computer. In most cases, if the task list is properly drawn, then each task will correspond to one line of pseudocode. Consider writing out your task list, then translating that list into pseudocode, then gradually developing that pseudocode into actual, computer-readable code.[3]

4. 

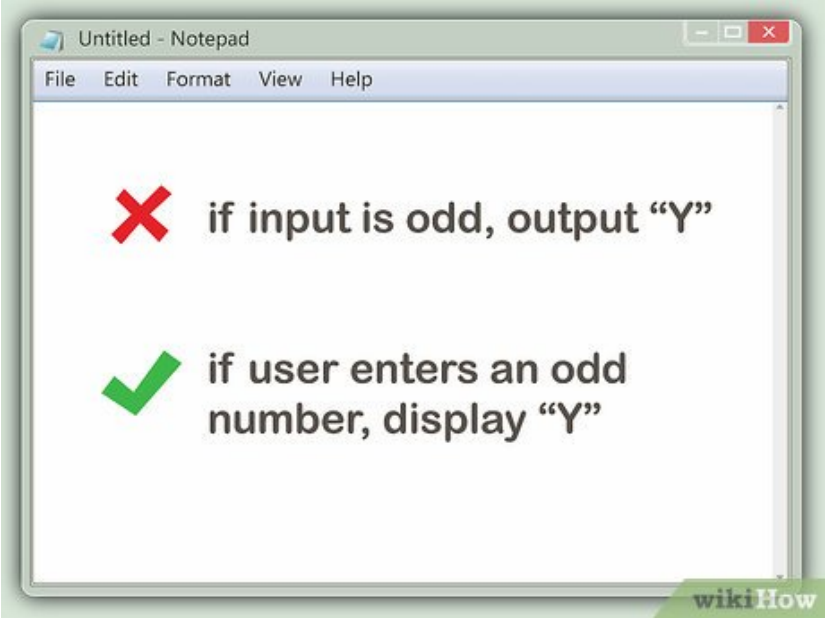
Use white space and indentation effectively. Using white spaces between "blocks" of text will help keep different components of your pseudocode isolated, and indenting different pieces of each block will indicate that those pieces of pseudocode go under a less-indented section.

1. For example, a section of pseudocode that discusses entering a number should all be in the same "block", while the next section (e.g., the section that discusses the output) should be in a different block.

5. 

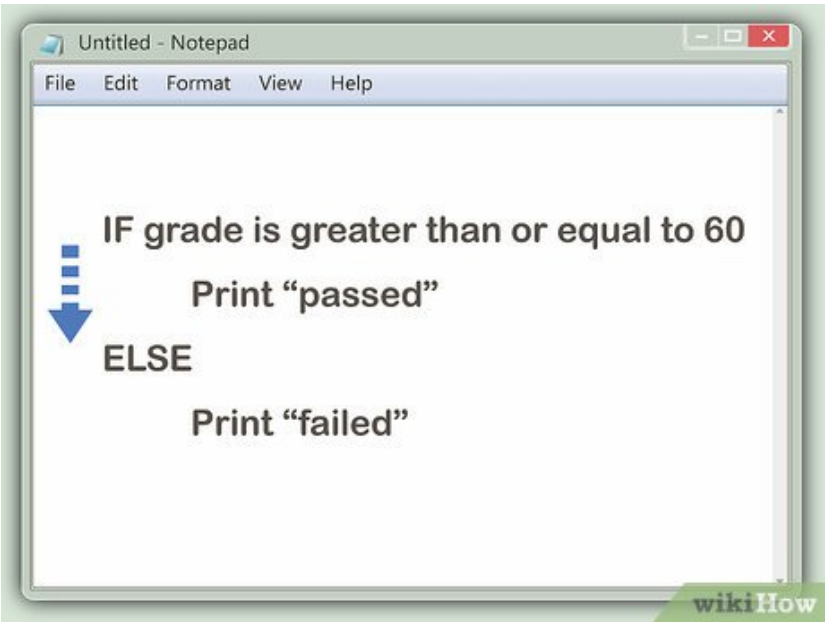
Capitalize key commands if necessary. Depending on your pseudocode requirements or the environment in which you're publishing the pseudocode, you may need to capitalize commands that will remain in the actual code.

1. For example, if you use "if" and "then" commands in your pseudocode, you might want to change them to read "IF" and "THEN" (e.g., "IF input number THEN output result").

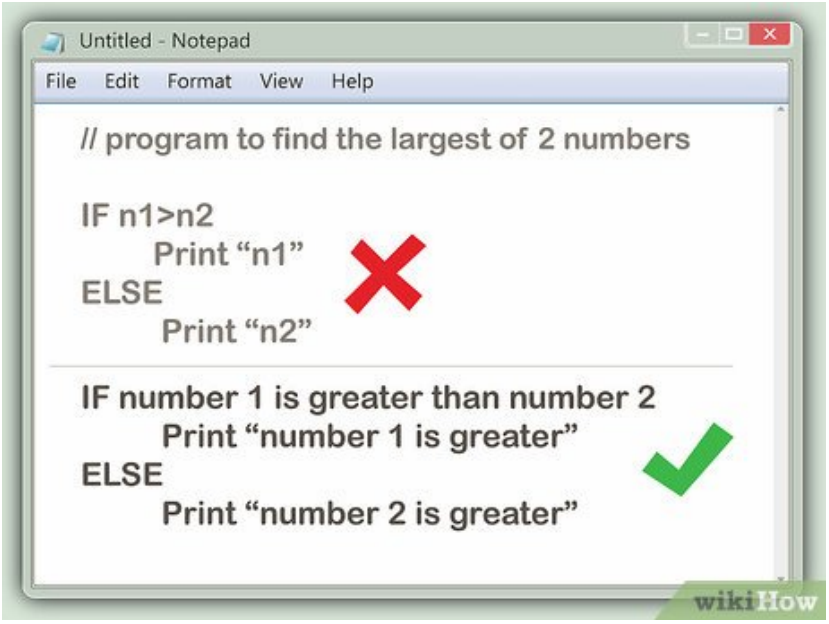
6. 

Write using simple terminology. Remember, you're writing about what the project will *do*, not summarizing the code itself. This is especially important if you're writing pseudocode to serve as a demonstration for a customer who doesn't know coding, or as a project for a beginner programmer.[4]

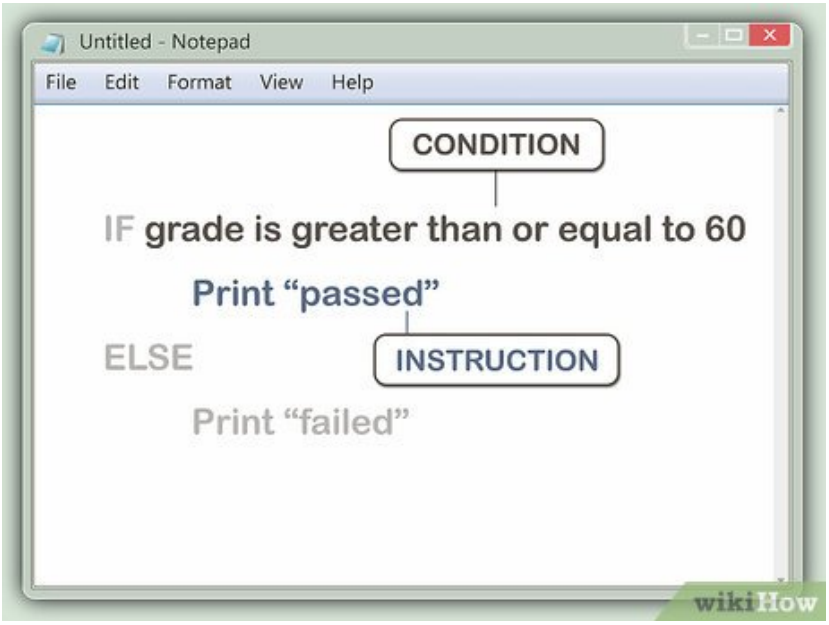
You might even want to get rid of any coding commands altogether and just define each line's process in plain language. For example, "if input is odd, output 'Y'" might become "if user enters an odd number, display 'Y'" instead.

7. 

Keep your pseudocode in the proper order. While the language you use to modify your pseudocode should be simple, you still need to keep each piece of your pseudocode in the order in which it needs to be executed.

8. 

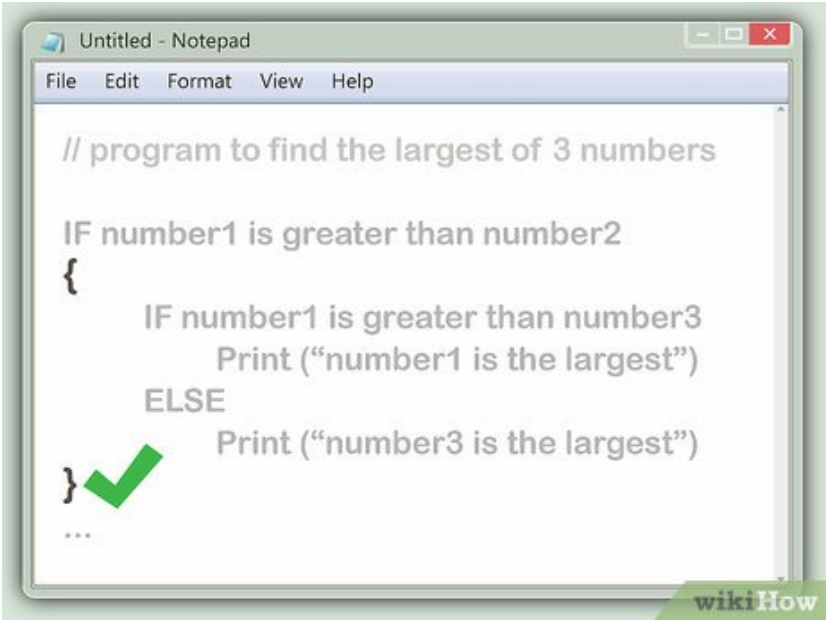
Leave nothing to the imagination. Everything that is happening in the process must be described completely. Pseudocode statements are close to simple English statements. Pseudocode does not typically use variables, but instead describes what the program should do with close-to-real-world objects such as account numbers, names, or transaction amounts.[5]

9. 

Use standard programming structures. Even if there is no standard for pseudocode, it will be easier for other programmers to understand your steps if you use structures from existing (sequential) programming languages.[6] Use terms like "if", "then", "while", "else", and "loop" the same way that you would in your preferred programming language. Consider the following structures:

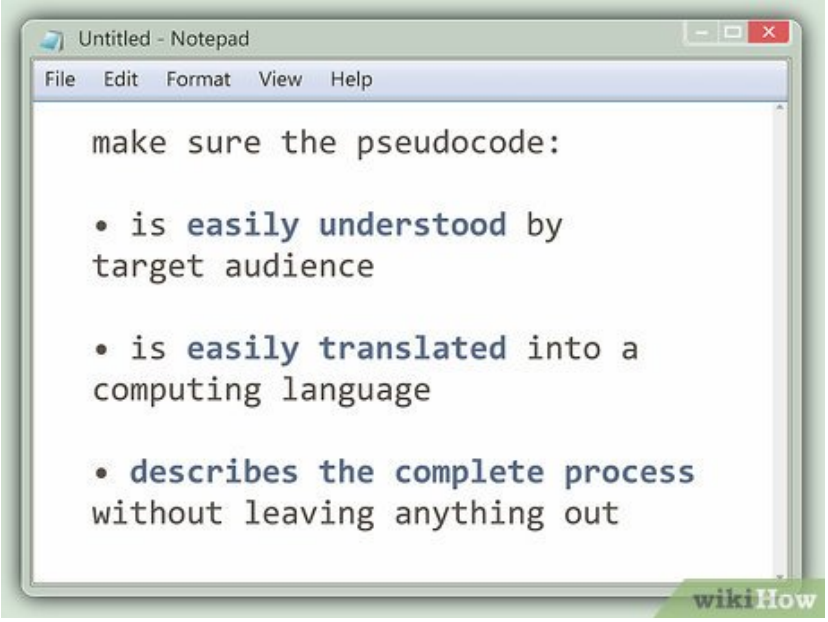
1. *if CONDITION then INSTRUCTION* — This means that a given instruction will only be conducted if a given condition is true. "Instruction", in this case, means a step that the program will perform, while "condition" means that the data must meet a certain set of criteria before the program takes action.[7]

2. *while CONDITION do INSTRUCTION* — This means that the instruction should be repeated again and again until the condition is no longer true.[8]
3. *do INSTRUCTION while CONDITION* — This is very similar to "while CONDITION do INSTRUCTION". In the first case, the condition is checked before the instruction is conducted, but in the second case the instruction will be conducted first; thus, in the second case, INSTRUCTION will be conducted at least one time.
4. *function NAME (ARGUMENTS): INSTRUCTION* — This means that every time a certain name is used in the code, it is an abbreviation for a certain instruction. "Arguments" are lists of variables that you can use to clarify the instruction.

10. 

Organize your pseudocode sections. If you have large sections of pseudocode that define other pieces of pseudocode within the same block, you may want to use brackets or other identifiers to keep everything contained.

1. Brackets—both standard (e.g., [code]) and curved (e.g., {code})—can help contain long segments of pseudocode.
2. When coding, you can add comments by typing "//" on the left side of the comment (e.g., `//This is a temporary step.`). You can use this same method when writing pseudocode to leave notes that don't fit into the coding text.

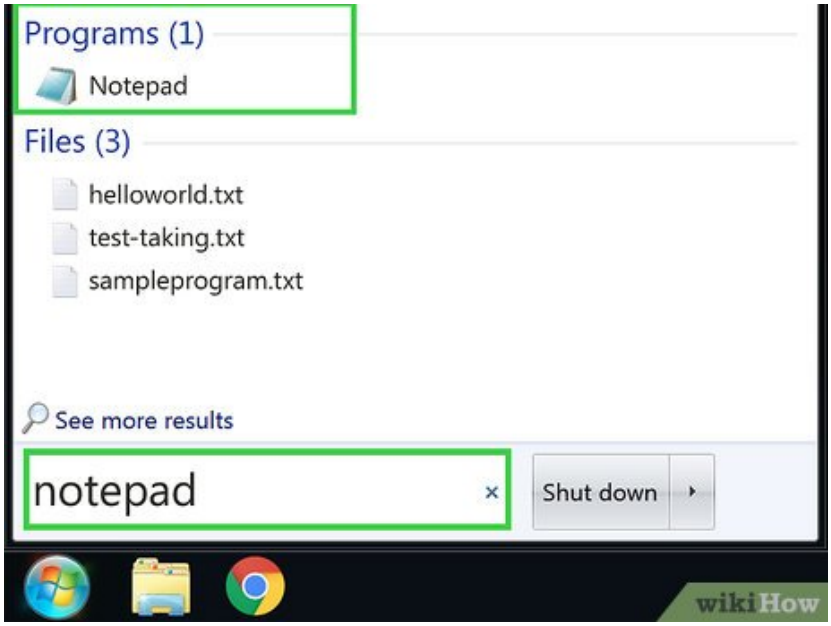
11. 

Double-check your pseudocode for readability and clarity. You should be able to answer the following questions by the end of the document:

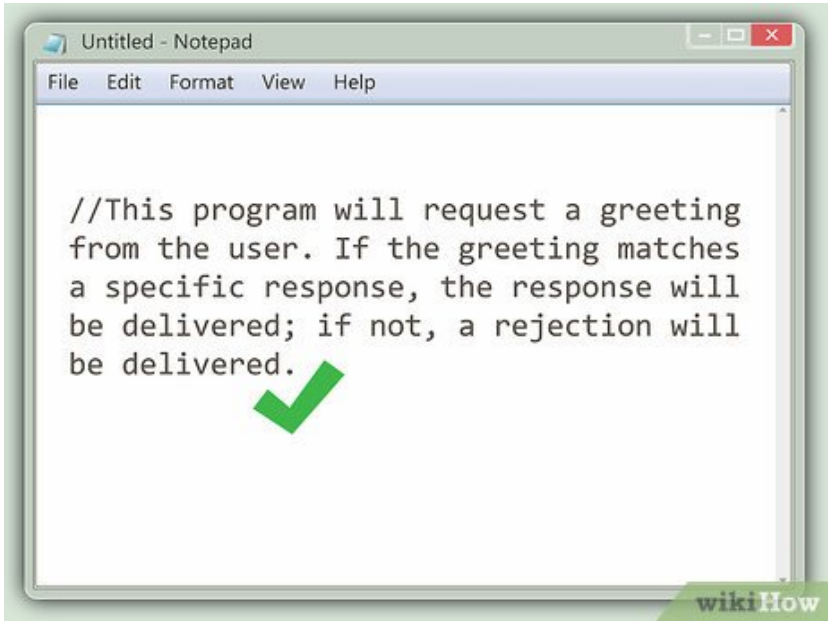
1. Would this pseudocode be understood by someone who isn't familiar with the process?
2. Is the pseudocode written in such a way that it will be easy to translated it into a computing language?
3. Does the pseudocode describe the complete process without leaving anything out?
4. Is every object name used in the pseudocode clearly understood by the target audience?
5. If you find that a section of pseudocode needs elaboration or it doesn't explicitly outline a step that someone else might forget, go back and add the necessary information.

Part 3 of 3:

Creating an Example Pseudocode Document

1. 

Open a plain-text editor. You can use Notepad (Windows) or TextEdit (Mac) by default if you don't want to install a new program.

2. 

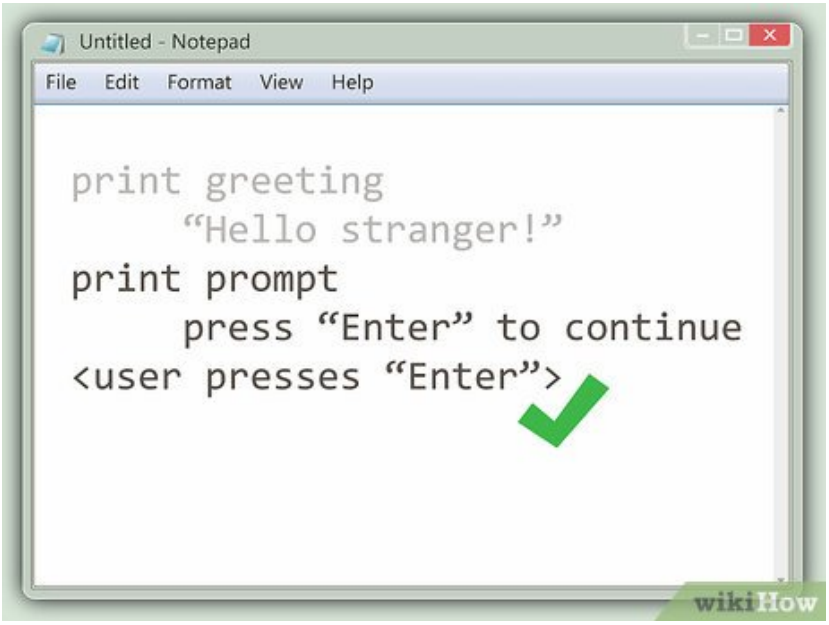
Define your program. While not strictly necessary, writing a one- or two-sentence line at the top of the document will make clear from the beginning the intent of the program:

```
This program will request a greeting from the user. If the greeting matches
; if not, a rejection will be delivered.
```

3. 

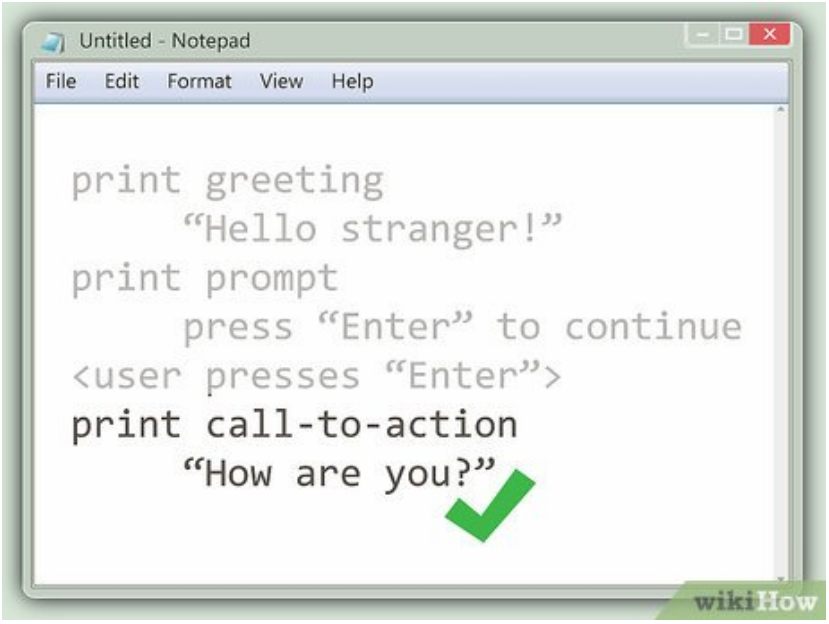
Write the opening sequence. Your first command—that is, the first thing your program should do upon running—should be the first line:

```
print greeting "Hello stranger!"
```

4. 

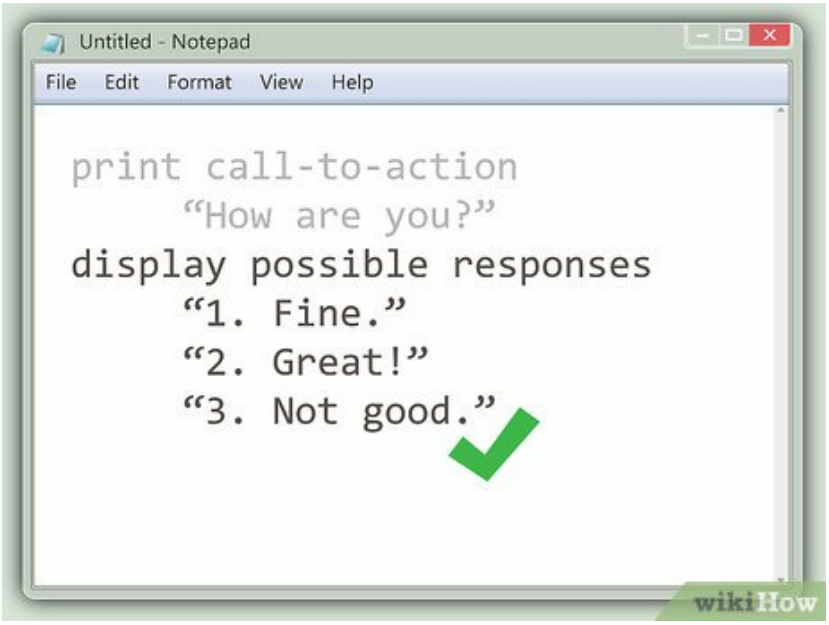
Add the next line. Place a space between the last line and the next one by pressing ? Enter, then create the next line of code. In this example, the user should prompt the next line of dialogue:

```
print prompt press "Enter" to continue "Enter">
```

5. 

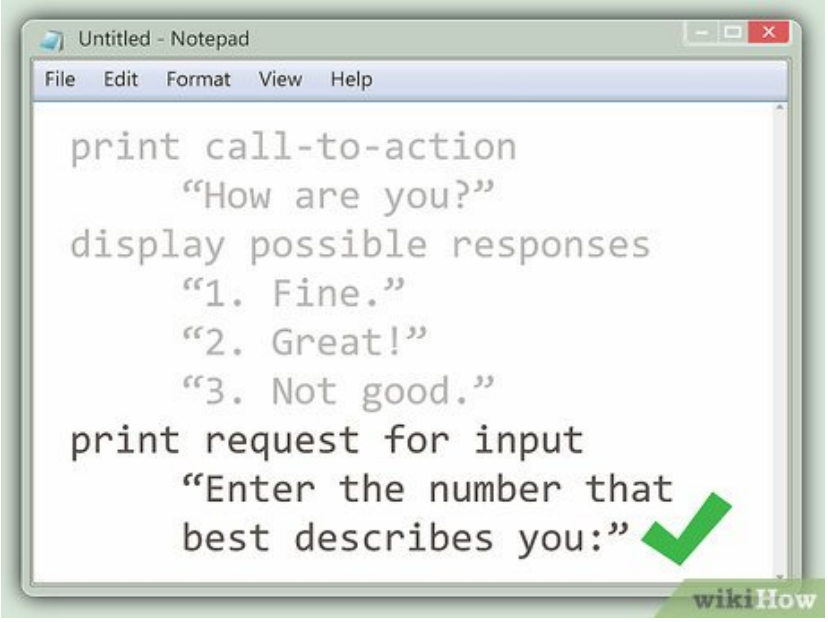
Add the call to action. In this example, the user will be prompted for a greeting:

```
print call-to-action "How are you?"
```

6. 

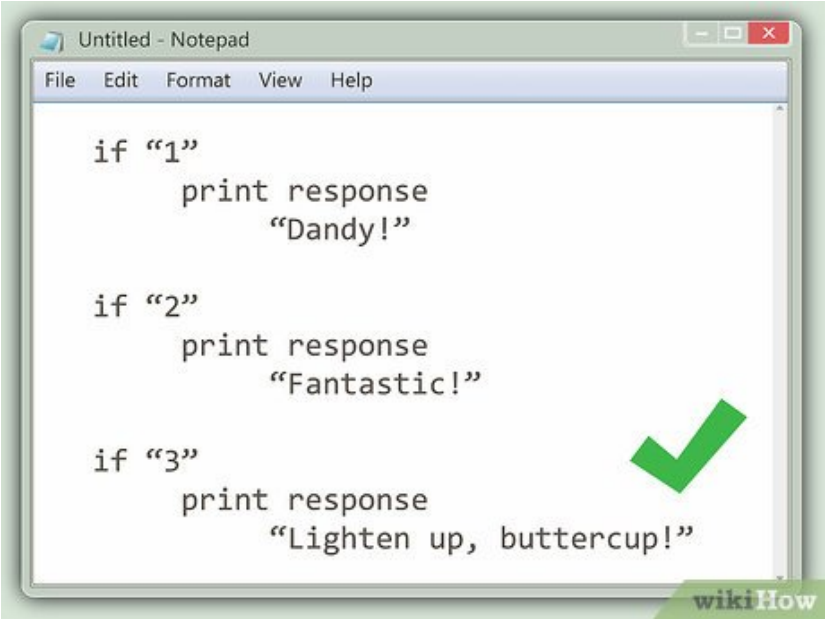
Show the user a list of responses. Again, after pressing ? Enter in this example, the user should see a list of possible responses:

```
display possible responses "1. Fine." "2. Great!" "3. Not good."
```

7. 

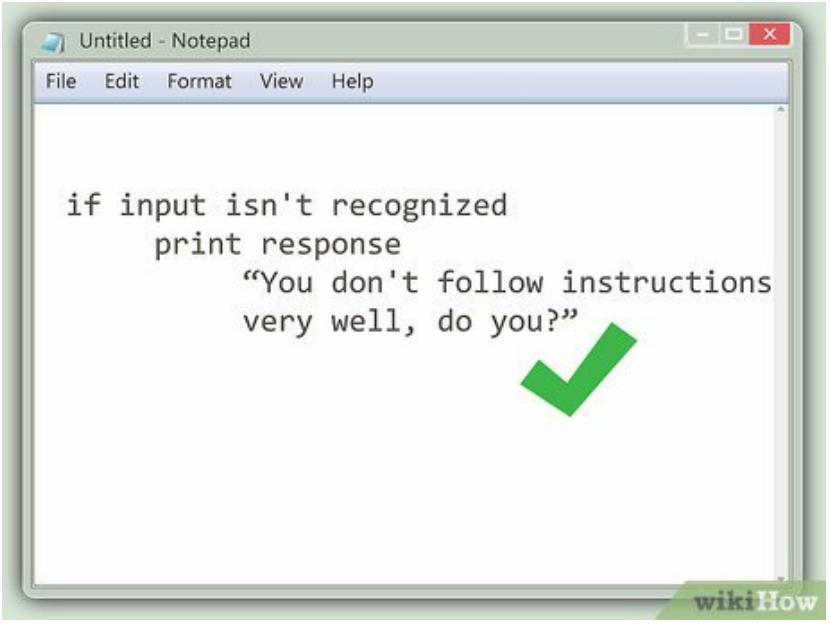
Request input from the user. This is where the program will ask the user to enter a response:

```
print request for input "Enter the number that best describes you:"
```

8. 

Create "if" commands for the user's input. Since there are multiple responses the user can select, you'll want to add multiple results based on their selected response:

```
if "1" print response "Dandy!" if "2" print response "Fantastic!" if "3" print response "Lighten up, buttercup!"
```

9. 

Add an error message. In the event that the user incorrectly chooses a response, you can have an error message ready:

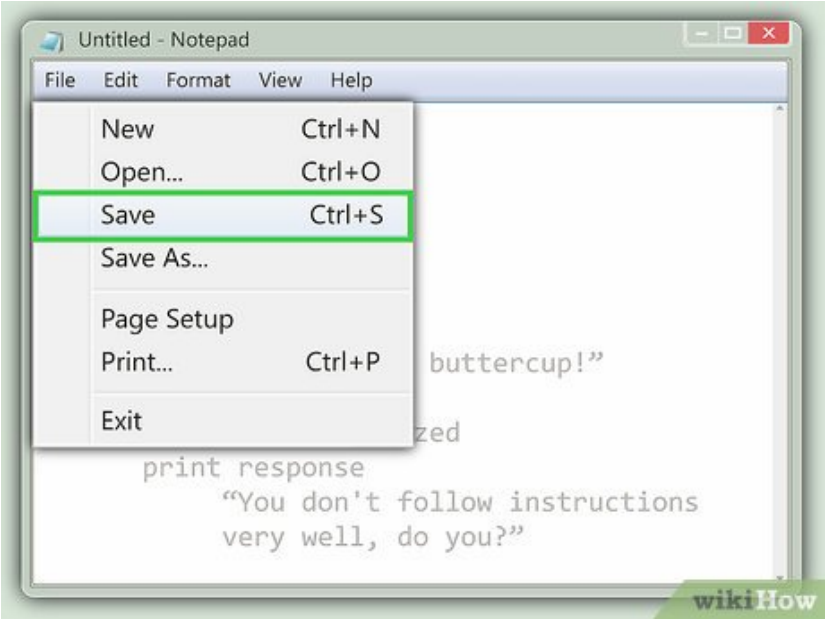
```
if input isn't recognized  print response "You don'
t follow instructions very well, do you?"
```

10. 

Add any other components of the program. Go through your document and add or flesh out any details to ensure that both you and anyone reading the document will understand its meaning. As per this method's example, your final pseudocode document should look something like this:

```
This program will request a greeting from the user. If the greeting matches
; if not, a rejection will be delivered. print greeting
```

```
"Hello stranger!" print prompt press "Enter" to continue
 "Enter"> print call-to-action "How are you today?"
 display possible responses "1. Fine." "2. Great!" "3. Not good."
 print request for input "Enter the number that best describes you:" if
 "1" print response "Dandy!" if "2" print response "Fantastic!" if "3"
 print response "Lighten up, buttercup!" if input isn't recognized
 print response "You don't follow instructions very well, do you?"
```

11. 

Save your document. Press Ctrl + S (Windows) or ? Command + S (Mac), enter a name, and click **Save** to do so.