

# Pseudocode schreiben

unter Mitarbeit von **17 Artikelschreibern**

Pseudocode ist ein informelles Werkzeug, das du benutzen kannst, um deine Algorithmen zu planen. Wenn du damit anfängst, komplexere Codes zu schreiben, kann es schwer sein, das gesamte Programm im Kopf zu behalten, bevor du es kodierst. Stelle dir Pseudocode als einen verbalen Schritt-für-Schritt-Entwurf deines Codes vor, den du später in eine Programmiersprache umschreiben kannst. Es ist eine Kombination aus menschlicher und Programmiersprache: Es ahmt den Satzbau tatsächlicher Computercodes nach, sorgt sich aber mehr um die Lesbarkeit als um technische Genauigkeit.

Methode

1

Methode 1 von 5:

## Pseudocode verstehen

**1 Du musst wissen, was Pseudocode ist.** Pseudocode ist ein verbaler Schritt-für-Schritt-Entwurf deines Codes, den du nach und nach in Programmiersprache umschreiben kannst. Viele Programmierer benutzen ihn, um die Funktion eines Algorithmus' zu planen, bevor sie sich an die technischere Aufgabe des Kodierens setzen. Pseudocode dient als informelle Anleitung, als Werkzeug, um Programmierungsprobleme zu durchdenken, und als Kommunikationsgerät, das dir dabei helfen kann, anderen Personen deine Ideen zu erklären.

**2 Verstehe, warum Pseudocode nützlich ist.** Pseudocode wird benutzt, um zu zeigen, wie ein EDV-Algorithmus funktionieren sollte und könnte. Programmierer benutzen Pseudocode oftmals als Zwischenschritt der Programmierung, zwischen dem anfänglichen Planungsstadium und dem Stadium des Schreibens des tatsächlichen, ausführbaren Codes. Guter Pseudocode kann im letztendlichen Programm zu Anmerkungen werden und den Programmierer in der Zukunft anleiten, wenn er den Code korrigiert und überarbeitet. Pseudocode kann außerdem nützlich sein, um zu:

- Beschreiben, wie ein Algorithmus funktionieren sollte. Pseudocode kann illustrieren, wo ein bestimmtes Konstrukt, ein Mechanismus oder eine Technik in einem Programm erscheinen könnte oder muss. Führende Programmierer benutzen den Pseudocode oftmals, um schnell die Schritte zu erklären, die ihre untergeordneten Programmierer befolgen müssen, um eine erforderliche Aufgabe zu erfüllen.
- Weniger technisch versierten Personen einen EDV-Prozess zu erklären. Computer benötigen einen sehr strengen Eingabesatzbau, um ein Programm laufen zu lassen. Menschen (insbesondere Nicht-Programmierer) finden es möglicherweise einfacher, eine flüssigere, subjektivere Sprache zu verstehen, die den Zweck jeder Zeile des Codes klar darlegt.
- Codes in einer Entwicklungsgruppe zu entwerfen. Software-Architekten auf hohem Niveau schließen oftmals Pseudocode in ihre Entwürfe ein, um ein komplexes Problem zu lösen, in das sie ihre Programmierer hineinlaufen sehen. Falls du ein Programm zusammen mit anderen entwickelst, könntest du feststellen, dass Pseudocode dabei hilft, deine Absichten zu verdeutlichen.

**3 Vergiss nicht, dass Pseudocode subjektiv und kein Standard ist.** Es gibt keinen festgelegten Satzbau, den du für Pseudocode unbedingt benutzen musst. Es ist aber übliche Berufsethik, Pseudocode-Standardstrukturen zu verwenden, die andere Programmierer leicht verstehen können. Falls du ein Projekt allein codierst, ist das Wichtigste, dass Pseudocode dir dabei hilft, deine Gedanken zu strukturieren und deinen Plan umzusetzen. Falls du mit anderen zusammen an einem Projekt arbeitest – ob sie nun deine Fachkollegen, untergeordneten Programmierer oder nicht-technische Mitarbeiter sind – ist es wichtig, zumindest einige Standardstrukturen zu verwenden. So kann jeder deine Absicht leicht verstehen.

- Falls du in einem Programmierkurs an einer Universität, in einem Programmiercamp oder in einer Firma eingeschrieben bist, wirst du wahrscheinlich in einem gelehrten Pseudocode-„Standard“ geprüft. Dieser Standard variiert oftmals zwischen den Institutionen und Lehrkräften.
- Klarheit ist ein primäres Ziel von Pseudocode, und sie könnte dir helfen, wenn du innerhalb akzeptierter Programmierkonventionen arbeitest. Wenn du deinen Pseudocode zu tatsächlichem Code entwickelst, musst du ihn in eine Programmiersprache umschreiben – so kann sie dir mit diesem im Hinterkopf dabei helfen, deinen Entwurf zu strukturieren.

**4 Verstehe Algorithmen.** Ein Algorithmus ist eine Prozedur, mit der ein Problem gelöst wird, was die Aktionen, die ein Programm unternimmt (und die Reihenfolge, in der es diese Aktionen unternimmt), betrifft. Ein Algorithmus ist bloß die Sequenz der Schritte, die unternommen werden, um ein Problem zu lösen. Die Schritte sind normalerweise „Sequenz“, „Auswahl“, „Wiederholungsschleife“ und eine Angabe zur Groß- und Kleinschreibung.

- In C sind „Sequenzangaben“ Imperative.
- Die „Auswahl“ ist die „if then else“-Angabe.
- Die Wiederholungsschleife wird durch eine Anzahl an Angaben erfüllt, wie etwa das „while“, „do“ und das „for“.
- Die Angabe zur Groß- und Kleinschreibung wird durch die „switch“-Angabe erfüllt.

**5 Denke an die drei Grundkonstrukte, die den Fluss des Algorithmus' kontrollieren.** Wenn du eine „Sequenz“-Funktion, eine „while“(Schleifen)-Funktion und eine „if-then-else“(Auswahl)-Funktion einbauen kannst, dann hast du die Grundwerkzeuge, die du brauchst, um einen „richtigen“ Algorithmus zu schreiben.[1]

- SEQUENZ ist eine lineare Abfolge, bei der eine Aufgabe auf eine andere folgend ausgeführt wird. Zum Beispiel:
  - LIES Höhe des Rechtecks
  - LIES Breite des Rechtecks
  - BERECHNE Fläche als Höhe mal Breite
- WHILE ist eine Schleife (Wiederholung) mit einer einfachen Bedingungsprüfung am Anfang. Der Anfang und das Ende der Schleife werden von den beiden Schlüsselwörtern WHILE und ENDWHILE angezeigt. Die Schleife wird nur betreten, wenn die Bedingung zutrifft. Zum Beispiel:
  - WHILE Population < Limit
    - Berechne Population als Population + Geburten - Todesfälle
  - ENDWHILE
- IF-THEN-ELSE ist eine Entscheidung (Auswahl), bei der zwischen zwei alternativen Handlungsweisen gewählt wird. Eine binäre Wahl wird durch diese vier Schlüsselwörter angezeigt: IF, THEN, ELSE und ENDIF. Zum Beispiel:
  - IF Arbeitsstunden > NormalMaximum THEN
    - Überstundenbenachrichtigung anzeigen
  - ELSE
    - Benachrichtigung über reguläre Zeit anzeigen
  - ENDIF

Methode  
2

Methode 2 von 5:

### Beispiel-Pseudocode

**1 Ziehe ein einfaches Beispielprogramm in Betracht.** Stelle dir vor, dass das Programm jedes Vorkommen des Wortes „XYZ“ in einer Textdatei ersetzen soll. Das Programm liest jede Zeile in einer Datei, sucht in jeder Zeile nach einem bestimmten Wort und ersetzt das Wort dann. Du kannst sehen, dass die zu wiederholenden Schritte im Pseudocode eingerückt sind, genauso, wie sie es idealerweise in einem echten Code wären. Ein erster Entwurf des Pseudocodes könnte so aussehen:

- Öffne die Datei
- In jeder Zeile der Datei:
  - Suche nach dem Wort
  - Entferne die Zeichen des Wortes
  - Setze die Zeichen des neuen Wortes ein
- Schließe die Datei dann.

**2 Benutze Pseudocode als Wiederholungsschleife:** "Schreibe es einmal und überarbeite es später". Eine der Stärken von Pseudocode ist, dass du die Grundlagen umreißen und das schwere Zeug für später übriglassen kannst. Beachte, dass es in dem Wortersetzungs-Beispiel oben keine Einzelheiten dazu gibt, wie nach dem Wort gesucht werden soll. Du, der Programmierer, kannst den Pseudocode umschreiben, um Algorithmen zum Entfernen der Zeichen des Wortes und zum Einsetzen des neuen Wortes einzuschließen. Ein zweiter Entwurf des Pseudocodes könnte so aussehen:

- Öffne die Datei
- In jeder Zeile der Datei:
  - Suche nach dem Wort, indem du dieses tust:
    - Lies das Zeichen in der Zeile
    - Falls das Zeichen passt, dann:
      - Falls alle folgenden Zeichen passen,
      - hast du ein Zutreffen.
      - Entferne die Zeichen des Wortes
      - Setze die Zeichen des neuen Wortes ein
- Schließe die Datei dann.

**3 Benutze Pseudocode, um Funktionen hinzuzufügen.** Pseudocode hilft Programmierern dabei, sich durch ein Problem hindurch zu denken, genau wie Zwischenschritte in einer Mathematikaufgabe. Wenn er richtig genutzt wird, kann Pseudocode eine komplizierte Programmierungsherausforderung einfach erscheinen lassen. Du kannst den Pseudocode nach und nach einen Schritt nach dem anderen verbessern:

- Öffne die Datei
- Frage den Benutzer nach dem zu ersetzenden Wort
- Frage den Benutzer nach dem Wort, durch das es ersetzt werden soll
- In jeder Zeile der Datei:
  - Suche nach dem Wort, indem du dieses tust:
    - Lies das Zeichen in der Zeile
    - Falls das Zeichen passt, dann:
      - Falls alle folgenden Zeichen passen,
      - hast du ein Zutreffen.
  - Zähle das Vorkommen des Wortes.
  - Entferne die Zeichen des Wortes
  - Setze die Zeichen des neuen Wortes ein
  - Zeige die Menge des Vorkommens des Wortes an.
- Schließe die Datei dann.

**1 Schreibe nur eine Angabe pro Zeile.** Jede Angabe in deinem Pseudocode sollte nur eine Handlung für den Computer ausdrücken. In den meisten Fällen entspricht jede Aufgabe einer Zeile des Pseudocodes, wenn die Aufgabenliste richtig skizziert wird. Ziehe in Erwägung, deine Aufgabenliste aufzuschreiben, diese Liste dann in Pseudocode zu übersetzen und den Pseudocode dann allmählich zu einem tatsächlichen computerlesbaren Code zu entwickeln.[2]

- Aufgabenliste:
  - Lies Namen, Stundensatz, Arbeitsstunden, Abzugssatz
  - Führe Berechnungen durch
  - $\text{brutto} = \text{Stundensatz} * \text{Arbeitsstunden}$
  - $\text{Abzug} = \text{Bruttolohn} * \text{Abzugssatz}$
  - $\text{Nettolohn} = \text{Bruttolohn} - \text{Abzug}$
  - Schreibe Namen, Brutto, Abzug, Nettolohn
- Pseudocode:
  - LIES name, stundenSatz, Arbeitsstunden, abzugsSatz
  - $\text{bruttoLohn} = \text{stundenSatz} * \text{Arbeitsstunden}$
  - $\text{abzug} = \text{bruttoLohn} * \text{abzugsSatz}$
  - $\text{nettoLohn} = \text{bruttoLohn} - \text{abzug}$
  - SCHREIBE name, bruttoLohn, abzug, nettoLohn

**2 Schreibe das erste Schlüsselwort jeder Hauptanweisung ganz groß.** Im obigen Beispiel werden LIES und SCHREIBE ganz großgeschrieben, um anzuzeigen, dass es die primären Funktionen des Programms sind. Relevante Schlüsselwörter könnten unter anderem LIES, SCHREIBE, IF, ELSE, ENDIF, WHILE, ENDWHILE, REPEAT und UNTIL sein.

**3 Schreibe, was du meinst, und nicht, wie du es programmieren willst.** Einige Programmierer schreiben Pseudocode wie ein Computerprogramm: Sie schreiben etwas wie "if ein % 2 == 1 then". Die meisten Leser müssen jedoch anhalten und über Zeilen nachdenken, die so abstrakt symbolisch sind. Es ist einfacher, eine verbale Zeile wie „if ungerade Zahl then“ zu verstehen. Je deutlicher du bist, desto leichter können die Leute verstehen, was du meinst.[3]

**4 Überlasse nichts der Phantasie.** Alles, was im Prozess passiert, muss vollständig beschrieben werden. Pseudocode-Angaben sind nahe an englischen/deutschen Aussagen. Pseudocode verwendet normalerweise keine Variablen, sondern beschreibt, was das Programm tun sollte, stattdessen mit lebensnahen Objekten wie Kontonummern, Namen oder Transaktionsbeträgen.

- Einige Beispiele für zulässigen Pseudocode sind:
  - Falls (IF) die Kontonummer und das Passwort gültig sind, dann (THEN) zeige die Basisinformationen des Kontos an.
  - Lege die Gesamtkosten proportional zur Rechnungssumme für jeden Versand um.
- Einige Beispiele für unzulässigen Pseudocode sind:
  - $\text{Lasse } g = 54/r$  "(Warum: Benutze keine Variablen. Stattdessen solltest du die Bedeutung in der realen Welt beschreiben.)"
  - Fahre mit der Hauptverarbeitung fort, bis sie erledigt ist "(Warum: Du solltest spezifisch sagen, was 'Hauptverarbeitung' bedeutet und was sie als 'erledigt' qualifiziert.)"

**5 Benutze Standard-Programmierungsstrukturen.** Auch wenn es für Pseudocode keinen Standard gibt, ist es für andere Programmierer einfacher, deine Schritte zu verstehen, wenn du Strukturen von bestehenden (sequentiellen) Programmiersprachen verwendest. Benutze Begriffe wie „if“, „then“, „else“ und „loop“ genauso, wie du es in deiner bevorzugten Programmiersprache tatest. Ziehe die folgenden Strukturen in Betracht:

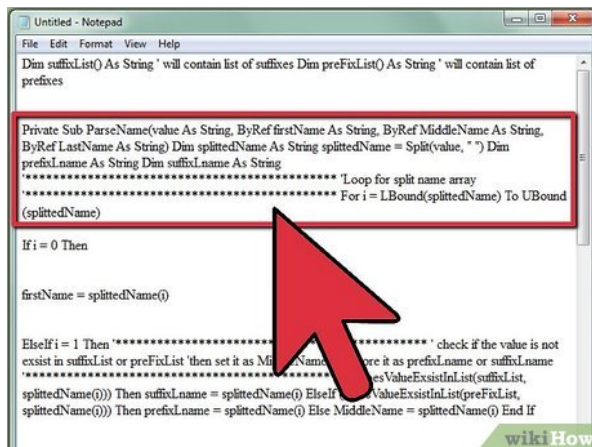
- if BEDINGUNG then ANWEISUNG – Das bedeutet, dass seine gegebene Anweisung nur dann ausgeführt wird, falls eine gegebene Bedingung zutrifft. „Anweisung“ bedeutet in diesem Fall einen Schritt, den das Programm ausführt. „Bedingung“ bedeutet, dass die Daten einen bestimmten Satz Kriterien erfüllen müssen, bevor das Programm in Aktion tritt.
- while BEDINGUNG do ANWEISUNG – Das bedeutet, dass die Anweisung immer wieder wiederholt werden sollte, bis die Bedingung nicht mehr zutrifft.
- Do ANWEISUNG while BEDINGUNG – Dieses ist „while BEDINGUNG do ANWEISUNG“ sehr ähnlich. Im ersten Fall wird die Bedingung überprüft, bevor die Anweisung ausgeführt wird, aber im zweiten Fall wird die Anweisung zuerst durchgeführt. Also wird ANWEISUNG im zweiten Fall mindestens einmal ausgeführt.
- for a = ZAHL1 to ZAHL2 do ANWEISUNG – Dieses bedeutet, dass „a“, eine Variable, automatisch auf ZAHL1 gesetzt wird. „a“ wird in jedem Schritt um eins erhöht, bis der Wert der Variable ZAHL2 erreicht. Du kannst für die Variable jede Bezeichnung verwenden, von der du denkst, dass sie besser als „a“ passt.
- function NAME (ARGUMENTE): ANWEISUNG – Das bedeutet, dass jedes Mal, wenn in dem Code ein bestimmter Name verwendet wird, er eine Abkürzung für eine bestimmte Anweisung ist. „Argumente“ sind Listen mit Variablen, die du benutzen kannst, um die Anweisung zu klären.

**6 Verwende Blöcke, um Schritte zu strukturieren.** Blöcke sind syntaktische Werkzeuge, die mehrere Anweisungen zu einer Anweisung verknüpfen. Du kannst Blöcke benutzen, um Informationen zu sortieren (sagen wir, die Schritte in Block 1 kommen immer vor den Schritten in Block 2) oder einfach um Informationen zu umwickeln (sagen wir, Anweisung1 und Anweisung2 sind thematisch verwandt). Rücke im Allgemeinen alle Angaben ein, die „Abhängigkeit“ von einer anderen Angabe zeigen.<sup>[4]</sup> Dazu gibt es zwei Methoden.

- Mit geschweiften Klammern:
  - {
  - ANWEISUNG1
  - ANWEISUNG2
  - ...}
- Oder Leerstellen. Wenn du Leerstellen benutzt, muss jede Anweisung desselben Blocks auf derselben Position beginnen. Blöcke innerhalb von Blöcken haben mehr Leerstellen als der Stammblock. Eine Anweisung aus einem Stammblock beendet den Nachfolgeblock, selbst wenn es später eine Anweisung mit derselben Menge an Leerstellen davor gibt.
  - BLOCK1
  - BLOCK1
    - BLOCK2
  - BLOCK2
    - BLOCK3
  - BLOCK2
    - BLOCK3
  - BLOCK1

**1 Beginne damit, den Zweck des Prozesses aufzuschreiben.** Das gibt dir die Möglichkeit, zu beurteilen, ob der Pseudocode vollständig ist: Kurz, wenn der Pseudocode den Zweck erfüllt, ist er vollständig. Fahre fort, indem du den Prozess aufschreibst. Wenn der Prozess einfach ist, brauchst du dafür möglicherweise nur einen Durchgang. Sieh dir an, was du aufgeschrieben hast, und stelle dir folgende Fragen:

- Würde dieser Pseudocode von jemandem verstanden, der zumindest einigermaßen mit dem Prozess vertraut ist?
- Ist der Pseudocode so geschrieben, dass er leicht in eine Computersprache übersetzt werden kann?
- Beschreibt der Pseudocode den vollständigen Prozess, ohne irgendetwas auszulassen?
- Wird jede Objektbezeichnung, die in dem Pseudocode verwendet wird, vom Zielpublikum klar verstanden?



```
Dim suffixList() As String ' will contain list of suffixes
Dim preFixList() As String ' will contain list of prefixes

Private Sub ParseName(value As String, ByRef firstName As String, ByRef middleName As String, ByRef lastName As String)
    Dim splittedName As String
    splittedName = Split(value, " ")
    Dim prefixName As String
    Dim suffixName As String

    ' Loop for split name array
    For i = LBound(splittedName) To UBound(splittedName)

        If i = 0 Then

            firstName = splittedName(i)

        ElseIf i = 1 Then ' check if the value is not exist in suffixList or preFixList ' then set it as MiddleName else store it as prefixName or suffixName
            ' If (DoesValueExistInList(suffixList, splittedName(i))) Then suffixName = splittedName(i) Else If (DoesValueExistInList(preFixList, splittedName(i))) Then prefixName = splittedName(i) Else MiddleName = splittedName(i) End If

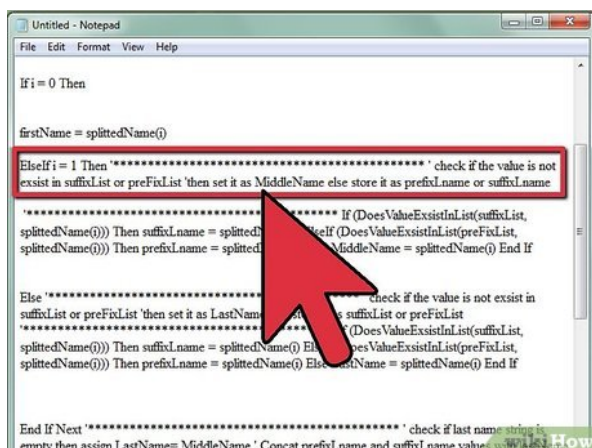
        End If

    Next i

    ' check if last name string is empty then assign LastName= MiddleName ' Concat prefixName and suffixName values with LastName
    lastName = MiddleName & " " & prefixName & " " & suffixName
End Sub
```

**2 Schreibe die anfänglichen Pseudocode-Schritte, welche die Voraussetzungen für Funktionen schaffen.** Die ersten Teile eines Codes definieren typischerweise die Variablen und anderen Elemente, die du benutzen wirst, um den Algorithmus funktional zu machen.

- Schließe Dimensionsvariablen ein. Schreibe einen Code, der zeigt, wie jede Variable oder jedes Datenelement benutzt wird.
- Stelle Einschränkungen auf. Du musst deine Einschränkungen in Pseudocode definieren – von Text und Bildfeldern in objektorientierten Programmiersprachen (OOP) bis zu Grundeinschränkungen in einfacheren Codes – genauso, wie du es in einem regulären Projekt tatest.[5]



```
Dim suffixList() As String ' will contain list of suffixes
Dim preFixList() As String ' will contain list of prefixes

Private Sub ParseName(value As String, ByRef firstName As String, ByRef middleName As String, ByRef lastName As String)
    Dim splittedName As String
    splittedName = Split(value, " ")
    Dim prefixName As String
    Dim suffixName As String

    ' Loop for split name array
    For i = LBound(splittedName) To UBound(splittedName)

        If i = 0 Then

            firstName = splittedName(i)

        ElseIf i = 1 Then ' check if the value is not exist in suffixList or preFixList ' then set it as MiddleName else store it as prefixName or suffixName
            ' If (DoesValueExistInList(suffixList, splittedName(i))) Then suffixName = splittedName(i) Else If (DoesValueExistInList(preFixList, splittedName(i))) Then prefixName = splittedName(i) Else MiddleName = splittedName(i) End If

        End If

    Next i

    ' check if last name string is empty then assign LastName= MiddleName ' Concat prefixName and suffixName values with LastName
    lastName = MiddleName & " " & prefixName & " " & suffixName
End Sub
```

**3 Schreibe funktionalen Pseudocode.** Wende Pseudocode-Prinzipien an, um spezifischen „Ereignis-gesteuerten“ oder „Objekt-gesteuerten“ Code hinzuzufügen, wenn du den „Rahmen“ für dein Projekt fertiggestellt hast. Jede Zeile deines Codes sollte eine Sequenzfunktion, eine Schleifenfunktion, eine Auswahlfunktion oder eine andere spezifische Handlung beschreiben.

```

Untitled - Notepad
File Edit Format View Help

If i = 0 Then

firstName = splittedName(i)

ElseIf i = 1 Then '*****' check if the value is not
exist in suffixList or preFixList' then set it as MiddleName
store it as prefixName or suffixName

'*****' check if the value is not exist in
suffixList or preFixList' then set it as LastName else store it as suffixList or preFixList
(DoesValueExistInList(suffixList,
splittedName(i))) Then suffixName = splittedName(i) ElseIf (DoesValueExistInList(preFixList,
splittedName(i))) Then prefixName = splittedName(i) ElseIf (DoesValueExistInList(MiddleName, splittedName(i))) Then
MiddleName = splittedName(i) Else LastName = splittedName(i) End If

Else '*****' check if the value is not exist in
suffixList or preFixList' then set it as LastName else store it as suffixList or preFixList
(DoesValueExistInList(suffixList,
splittedName(i))) Then suffixName = splittedName(i) ElseIf (DoesValueExistInList(preFixList,
splittedName(i))) Then prefixName = splittedName(i) Else LastName = splittedName(i) End If

End If Next '*****' check if last name string is
empty then assign LastName= MiddleName' Concat prefixName and suffixName values
End If

How

```

**4 Füge bei Bedarf Kommentare hinzu.** Im tatsächlichen Computercode dienen Kommentare der Identifizierung von Aufgaben und Teilen des Codes für den Leser. Pseudocode sollte diese Schritte in beinahe gewöhnlichem Englisch/Deutsch deutlich erklären. Daher musst du vermutlich Kommentare benutzen, bis du deinen Pseudocode in eine Programmiersprache übersetzt.

- Viele Programmierer entscheiden sich dafür, ihren Pseudocode in Form von Kommentaren in den fertigen Code umzuwandeln. Das hilft anderen Programmierern – die möglicherweise am Code mitarbeiten, ihn bearbeiten oder von ihm lernen –, dabei, die Absicht hinter jeder Zeile zu verstehen.
- Beginne Kommentare mit //, damit der Computer sie nicht liest. Achte darauf, dass die Doppelschrägstriche eingerückt sind. Zum Beispiel:
  - // IF Roboter kein Hindernis vor sich hat THEN
  - // Forderung Bewegung Roboter
  - // Füge den Bewegungsbefehl zum Befehlsverlauf hinzu
  - // RETURN zutreffend
  - //ELSE
  - // RETURN falsch ohne den Roboter zu bewegen
  - //END IF



```
Dim suffixList() As String ' will contain list of suffixes
Dim prefixList() As String ' will contain list of prefixes

Private Sub ParseName(value As String, ByRef firstName As String, ByRef middleName As String, ByRef lastName As String)
    Dim splittedName As String
    splittedName = Split(value, " ")
    Dim prefixName As String
    Dim suffixName As String

    ' Loop for split name array
    For i = LBound(splittedName) To UBound(splittedName)

        If i = 0 Then
            firstName = splittedName(i)
        Else If i = 1 Then ' check if the value is not exist in suffixList or prefixList then set it as MiddleName else store it as prefixName or suffixName
            ' If (DoesValueExistInList(suffixList, splittedName(i))) Then suffixName = splittedName(i) Else If (DoesValueExistInList(prefixList, splittedName(i))) Then prefixName = splittedName(i) Else MiddleName = splittedName(i) End If
        Else ' check if the value is not exist in suffixList or prefixList then set it as LastName else store it as suffixList or prefixList
            ' If (DoesValueExistInList(suffixList, splittedName(i))) Then suffixList = splittedName(i) Else If (DoesValueExistInList(prefixList, splittedName(i))) Then prefixList = splittedName(i) Else LastName = splittedName(i) End If
        End If
    Next i
End Sub
```

**5 Lies das fertige Projekt auf logische und Syntaxfehler durch.** Noch einmal, der Satzbau braucht nicht perfekt zu sein, aber dein Pseudocode sollte trotzdem einen Sinn ergeben. Versuche, dich in die Lage einer anderen Person zu versetzen, die diesen Code liest. Überlege, ob deine Anweisungen so deutlich sind, wie sie es sein können.

- Schätze deine Codemodule bezüglich der verschiedenen Elemente ein, aus denen sie bestehen. Zum Beispiel: Kernoperationen für den Computer enthalten, eine Datei zu lesen (oder Informationen aus ihr zu beziehen), in eine Datei zu schreiben (oder sie auf dem Bildschirm darzustellen), mathematische Berechnungen anzustellen, Datenvariablen zu bewerten und mit einem oder mehreren Elementen zu vergleichen. Jedes davon hat in einem Computercodeprojekt seinen eigenen Platz, ebenso wie in dem Pseudocode, den du schreibst, um das Projekt zu unterstützen.
- Arbeite spezifische Aufgaben in den Pseudocode ein. Stelle jede Aufgabe mit einem lesbaren Pseudocode dar, wenn du sie identifiziert hast. Während du die eigentliche Kodierungssprache, die du benutzen willst, nachahmst, brauchst du der Computerprogrammiersprache nicht exakt zu folgen.
- Achte darauf, dass alle zutreffenden Elemente im Pseudocode enthalten sind. Selbst wenn du einige der technischen Begriffe eventuell nicht brauchst, wie Variablenerklärungen, musst du trotzdem jede Aufgabe und jedes Element in dem Text deutlich repräsentieren.

**6 Sieh den Pseudocode durch.** Gehe ihn mit weiteren Akteuren im Projekt durch, wenn der Pseudocode den Prozess ohne üble Fehler einigermaßen beschreibt. Bitte deine Teammitglieder darum, dir Feedback zu geben, welche Teile des Pseudocodes eine Verbesserung benötigen. Beschreibungen von Prozessen sind oftmals unvollständig, daher hilft es dir, die Einzelheiten des Prozesses einzufügen. Lies dir durch, was du geschrieben hast, und ziehe in Erwägung, jemand anderen darum zu bitten, es Korrektur zu lesen, falls du ganz allein an dem Code arbeitest.

- Schreibe ihn zur Klärung neu, falls dein Team den Pseudocode nicht anerkennt. Frage dein Team, was schief gelaufen ist: Waren deine Schritte bloß unklar, oder hast du vergessen, ein wichtiges Teil des Prozesses einzuschließen?

**7 Speichere deinen Pseudocode.** Speichere ihn in einem Archiv, wenn du und dein Team den Pseudocode abgesegnet habt. Schließe den Pseudocode unbedingt als Kommentar in die Computercodedatei ein, wenn du den eigentlichen computerlesbaren Code schreibst. Noch einmal, beginne Kommentare mit //, damit der Computer sie nicht liest.

## Methode 5

Methode 5 von 5:

## Pseudocode in eine Programmiersprache übersetzen

**1 Vollziehe den Pseudocode nach und verstehe, wie er funktioniert.** Der Pseudocode stellt dir einen Algorithmus vor. Zum Beispiel könnte der Code eine Liste alphabetisch ordnen. Der grundsätzliche „genetische“ Code führt dich, wenn du den Algorithmus in der bevorzugten Programmiersprache ausbaust.



**2 Verwende Kodierungselemente, die in deine Programmiersprache hinein passen.** Zu diesen Elementen könnten die Erklärung von Variablen, if-Angaben und loop-Angaben gehören. Jede Prozedurzeile kann auf viele verschiedene Arten eingebaut werden. Deine Optionen hängen vom Level deiner Programmiersprache ab.

- Versuche zum Beispiel, Daten so darzustellen, dass der Benutzer sie sieht. Du kannst ein Benachrichtigungsfenster verwenden, um die Daten anzuzeigen, oder das bestehende graphische Interface benutzen, auf dem du entwirfst.

**3 Baue den Pseudocode ein.** Schreibe den Pseudocode ordentlich, einfach und effizient. Ein gut geschriebener Pseudocode könnte den gesamten Algorithmus effizienter und fehlerfreier machen, wenn du das Programm startest.

**4 Vollziehe den eigentlichen Code erneut nach und vergleiche ihn mit dem Pseudocode.** Achte darauf, dass dein eigentlicher, umgesetzter Code die im Pseudocode dargelegten Anweisungen befolgt. Zum Beispiel: Probiere alle möglichen Eingaben aus und vergleiche die Ergebnisse deines umgesetzten Codes mit den vom Pseudocode berechneten Ergebnissen, falls der Pseudocode Eingaben und Ergebnisse annimmt. Du kannst einen Mitprogrammierer darum bitten, ihn nachzuvollziehen oder dir eine bessere Methode für deinen Code zu empfehlen.

## Tipps

- Verstehe die Kernoperationen eines Computers. Beim Computercode geht es nur darum, den Computer anzuweisen, bestimmte Kernaufgaben zu erledigen. Eine gute Kenntnis dieser Aufgaben hilft dir dabei, Pseudocode zu schreiben, der nachvollzieht, was ein tatsächliches Codeprojekt tut.
- Verwende Whitespace effektiv. Es ist in der Computerprogrammierung üblich, Whitespace zu benutzen, um einen Code aufzubrechen. Er ist sogar noch wichtiger, wenn der Code nur dazu gedacht ist, von Menschen gelesen zu werden. Stelle dir Whitespace als „Blöcke“ vor: Zeilen, die mit derselben Menge an Whitespace beginnen, sind im selben „Block“. Sie sind für den Prozess des Algorithmus‘ von derselben relativen Wichtigkeit.

## Referenzen

1. ↑ [http://users.csc.calpoly.edu/~jdalbey/SWE/pdl\\_std.html](http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html)
2. ↑ <http://faculty.ccri.edu/mkelly/COMI1150/PseudocodeBasics.pdf>
3. ↑ <http://www.bfoit.org/itp/Pseudocode.html>
4. ↑ <http://www.unf.edu/~broggio/cop2221/2221pseu.htm>
5. ↑ <http://searchsoa.techtarget.com/definition/object-oriented-programming>

## Über dieses wikiHow

wikiHow ist ein "wiki", was bedeutet, dass viele unserer Artikel von zahlreichen Mitverfassern geschrieben werden. An diesem Artikel arbeiteten bis jetzt 17 Leute, einige anonym, mit, um ihn immer wieder zu aktualisieren. Dieser Artikel wurde 76.181 Mal aufgerufen.

---

Kategorien: [Computersprachen](#)

<https://de.wikihow.com/Pseudocode-schreiben>

---

Text und Bilder dieses PDFs sind nur für deinen persönlichen, nichtgewerblichen, Gebrauch zugelassen. Jegliche gewerbliche Verwendung des Inhaltes dieses Artikels, ohne die ausdrückliche Genehmigung von wikiHow, ist verboten.