Algorithmus[1]

Definition

*Ein **Algorithmus** ist eine **eindeutige**, **ausführbare** Folge von Anweisungen **endlicher** Länge zur **Lösung** eines **Problems**.*
Ein Algorithmus besteht

- aus einem Deklarationsteil (Was wird benötigt?), z.B. bei einem Rezept die Zutaten
- und einem Anweisungsteil (Wie wird das Problem gelöst?), z.B. beim Rezept die Verarbeitungsanweisungen.

Beispiele für Algorithmen

- Vorschriften zum Addieren, Subtrahieren oder Multiplizieren von Zahlen
- euklidischer Algorithmus zur Berechnung des ggT zweier Zahlen
- (gute) Bastelanleitungen
- Spielregeln und alltägliche Vorschriften (sind aber leider nur selten exakt ausformuliert und enthalten Teile, die unterschiedlich interpretiert werden können)
- Spielen einer Melodie
- Bedienung eines Handys

Genauigkeit der Definition

Die obige Definition für den Begriff des Algorithmus ist zwar verständlich, aber mathematisch ungenau. In der Mathematik ab Beginn des 20. Jahrhunderts wurden eine Reihe von Ansätzen entwickelt, die zu einer genauen Definition führen sollten. Alan Turing zeigte, dass alle gefundenen Methoden letztlich alle gleich leistungsfähig (gleich mächtig). Das gelang ihm mit Hilfe eines mathematischen Modells: der Turing-Maschine. Mit Hilfe des Begriffs der **Turingmaschine** konnte folgende formale Definition des Begriffs formuliert werden:
Eine Berechnungsvorschrift zur Lösung eines Problems heißt genau dann Algorithmus, wenn eine zu dieser Berechnungsvorschrift äquivalente Turingmaschine existiert, die für jede Eingabe, die eine Lösung besitzt, stoppt.
Daraus ließen sich die folgenden Eigenschaften ableiten, die einen Algorithmus (für eine Maschine) genauer beschreiben.

Eigenschaften eines Algorithmus für eine Maschine

1. Endlichkeit (Finitheit)
   Die Beschreibung eines Algorithmus besitzt eine **endliche** Länge, d. h. er besteht aus einer begrenzten Anzahl von Anweisungen mit begrenzter Länge. Zudem darf ein Algorithmus zu jedem Zeitpunkt für seine Daten nur endlich viel Platz belegen -> Das Verfahren muss in einem endlichen Text eindeutig beschreibbar sein.
2. Ausführbarkeit
   Jede einzelne Anweisung eines Algorithmus muss vom Computer (vom Menschen) ausführbar sein -> Jeder Schritt des Verfahrens muss tatsächlich ausführbar sein.
3. Terminierung
   Der Algorithmus ist nach endlich vielen Schritten **beendet**, d. h. er liefert ein Ergebnis oder hält an.

Für die Praxis werden Algorithmen häufig auf die folgenden Eigenschaften eingeschränkt:

Allgemeinheit

Ein Algorithmus ist allgemeingültig, d. h. er löst eine **Vielzahl von Problemen** (der gleichen Art). Die Auswahl eines einzelnen konkreten Problems erfolgt über **Eingabedaten** oder **Parameter**.
Beispiel: Um das Problem „Dreieck zeichnen“ in TigerJython mit der Turtle zu lösen, kann man eine Funktion `zeichneDreieck (hoehe)` schreiben, die für unterschiedliche Werte des Parameters hoehe verschieden große Dreiecke zeichnet.

Eindeutigkeit

An jeder Stelle des Algorithmus muss **eindeutig** festgelegt sein, was zu tun ist und welcher Schritt der Nächste ist. Dafür muss jede Anweisung unmissverständlich formuliert sein. Für die Umsetzung eines Algorithmus in eine Programmiersprache bedeutet das eine korrekte **syntaktische und semantische** Formulierung.

Determiniertheit

Wird ein Algorithmus mit den **gleichen Eingabe**werten und **Start**bedingungen wiederholt, so liefert er stets das **gleiche Ergebnis**.

[1] Mit Material von: http://www.info-wsf.de/index.php/Grundlagen_der_Programmierung

Darstellungsformen für Algorithmen

Es gibt verschiedene **Darstellungsformen**, um algorithmische Grundbausteine und Algorithmen darzustellen. An dieser Stelle soll auf die drei wichtigsten eingegangen werden:

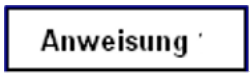
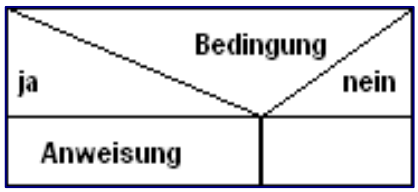
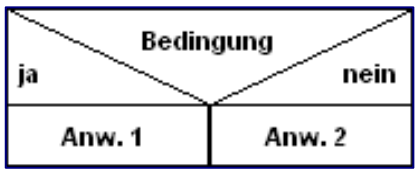
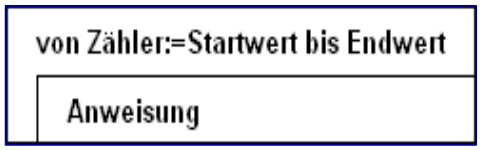
Pseudocode

Der Pseudocode stellt einen Algorithmus in einer Schreibweise in der Art einer Programmiersprache dar, die aber in der Regel näher an der natürlichen Sprache ist als am formalen Programmcode. Es gibt keine Standardisierung bzw. Vorschrift, wie ein Pseudocode auszusehen hat; die Ablaufstrukturen werden grob durch Texte und fest vorgeschriebene Schlüsselwörter beschrieben. Guter Pseudocode zeichnet sich durch eine kurze präzise Beschreibung des Algorithmus aus.

Struktogramm

Struktogramme (auch nach ihren Entwicklern **Nassi-Shneidermann-Diagramme** genannt) ist eine genormte (DIN 66261) Entwurfsmethode für Algorithmen. Die Methode zerlegt ein Gesamtproblem, das man mit dem gewünschten Algorithmus lösen will, in immer kleinere Teilprobleme bis schließlich nur noch elementare Grundstrukturen wie Anweisungsfolgen (Sequenzen) und Kontrollstrukturen zur Lösung des Problems übrigbleiben. In der Softwareentwicklung werden Struktogramme sehr selten eingesetzt, da normaler Programmcode einfacher zu schreiben und zu verändern ist: Korrigiert man einen Fehler oder macht eine Ergänzung, muss man ein Struktogramm in der Regel komplett neu zeichnen. Im **Informatik-Unterricht** werden Struktogramme dagegen verwendet, damit Schüler den Aufbau logischer Abläufe, die für die Programmierung nötig sind, trainieren können. Die Umsetzung von Struktogrammen in Programmcode ist immer noch Bestandteil der schulischen Ausbildung.

Grundelemente des Pseudocode und von Struktogrammen mit Python-Beispielen (Auswahl)

Strukturelement	**Pseudocode**	**Struktogramm**	**Python-Beispiel**
Verarbeitung (Elementarblock)	Anweisung		anzahl = 17
Reihenfolge (Sequenz)	Anweisung1 Anweisung2 Anweisung3		zahl = 11 ergebnis = zahl * zahl print ergebnis
Verzweigung (ohne Alternative)	wenn Bedingung dann Anweisung		**if** anzahl>167**:** anzahl = anzahl / 2
	wenn Bedingung dann Anweisung1 sonst Anweisung2		**if** bmi < 16: print “untergewichtig“ else: print “Normalgewicht“
Zählschleife	von Zähler:=Startwert bis Endwert Anweisung		**for** zaehler **in** range (1,11): print zaehler
anfangsgeprüfte Schleife	solange Bedingung Anweisung		zaehler = 0 **while** zaehler < 11: print zaehler zaehler = zaehler + 1

Ausblick: Das Konzept für das Nassi-Shneiderman-Diagramm (Struktogramm) stammt aus den 1960er-Jahren und bildet einen linearen Programmfluss ab. Dies ist insbesondere für **imperative** Programmiersprachen gut geeignet. Für die Abbildung **objektorientierter** Programmkonzepte hat sich dieses Konzept als ungeeignet erwiesen. Als Konsequenz wurde die Unified Modelling Language (UML) für objektorientierte Programmkonzepte entwickelt.