Algorithmen und Programme

Wissen: Man unterscheidet zwischen Deklarativen und imperativen Wissen. Deklaratives Wissen: Deklaratives Wissen ist das Wissen von Fakten, wohingegen Imperatives Wissen, das Wissen ist, wie man zu diesen Fakten kommt, also quasi der Algorithmus.

Algorithmus: Eine genau definiert Handlungsvorschrift(Weg, wie man ein bestimmtes Ziel erreichen kann) zur Lösung eines Problems oder einer bestimmten Art oder Klasse von Problemen. In der Informatik spricht man von einem Algorithmus, wenn eine zu dieser Berechnungsvorschrift äquivalente Turingmaschine existiert, die für jede Eingabe, die eine Lösung besitzt, stoppt.
Ein Algorithmus besteht dabei aus einer Menge an einfachen Schritten, Regeln bzw. Flusskontrolle, wann welcher Schritt auszuführen ist und Regeln um zu bestimmen wann gestoppt wird.

Eigenschaften eines Algorithmus:

- Ausführbarkeit: Die Anweisungen müssen verständlich formuliert und ausführbar sein. Dies bedeutet es muss eine vom Ausführenden Teil verstandene Sprache genutzt worden sein.
- Allgemeingültigkeit: Der Algorithmus löst eine Menge ähnlicher Probleme - eine ganze Problemklasse.
- Determiniertheit: Der Algorithmus liefert bei gleichen Ausgangsbedingungen *immer* das gleiche Ergebnis.
- Determinismus: Der Algorithmus liefert nicht nur immer das gleiche Ergebnis sondern durchläuft auch genau den gleichen Prozess um zu dem Ergebnis zu kommen. Heißt auch alle Zwischenergebnisse sind gleich. Ist ein Algorithmus deterministisch so ist er auch determiniert. Nicht aber umgekehrt!
- Statische Finitheit: Die Beschreibung des Algorithmus ist endlich. Wichtig! Hier geht es wirklich um die Beschreibung. Heißt z.B.: der Quellcode darf nicht unendlich groß sein.
- Dynamische Finitheit: Der Algorithmus braucht nur endlich viel Speicher!
- Terminiertheit: Der Algorithmus bricht nach endlicher Zeit kontrolliert ab. Heißt er kommt zu einem Ergebnis.

Big O-Notation: Ein Wert in Big O-Notation gibt die Laufzeit und Speicherkomplexität der Algorithmen an. Es wird dabei immer ein Maß für die Anzahl der Elementarschritte oder der Speichereinheiten in Abhängigkeit der Größe des gegebenen Problems angegeben. Hat man z.B.:

$$f = O(n)$$

so bedeutet dies,dass der Speicher bzw. die Laufzeit linear zur Größe von n wächst.

Berechenbarkeit: Eine Funktion heißt berechenbar wenn es einen Algorithmus gibt, der für jeden Eingabewert aus dem Definitionsbereich terminiert und ein Ergebnis liefert.
Church'sche These: Jede berechenbare Funktion ist dabei Turing-berechenbar.

Programm: Programme sind konkrete Implementierungen von Algorithmen. Ein Programm kann dabei entweder auf eine bestimmte Maschine zugeschnitten sein oder in einer

maschinenunabhängigen Programmiersprache formuliert sein, werden aber in jedem Fall in einer Programmiersprache formuliert.

Programmiersprachen: Programmiersprachen sind formale Sprachen und müssen vom Programmierer und vom Computer verstanden werden.

Paradigmen: Programmiersprachen können nach Paradigma klassifiziert werden. Das Paradigma bestimmt dabei den fundamentalen Programmierstiel. Hauptsächlich unterscheidet man in imperative und deklarative Programmiersprachen.

Generationen
Die sogenannte Generation ist eine weitere Eigenschaft die der Klassifiziert von Programmiersprachen dient.

1. Generation ist Maschinencode selbst. Dieser besteht lediglich aus einer Binärfolge und ist pro CPU unterschiedlich! Binärcode ist aber vom Processor direkt ausführbar.
2. Generation: Assemblersprachen. Dabei werden gewissen Binärcodes Buchstaben/Namen zugewiesen (Mnemonics). Ein Programm, der sogenannte Assembler wandelt diese Namen dann wieder in eine Folge von 1 und 0 um. Programme programmiert in Assembler können sehr effizient sein, sind jedoch immer nur auf genu einem Rechnertyp ausführbar. Außerdem sind sie schwer verständlich.
3. Generation: Diese sogenannten Höheren Programmiersprachen sind Hardware unabhängig und für den Menschen leichter verständlich. Sogenannte Compiler oder Interpreter übersetzen diese Sprachen dann in Binärcode.

Eigenschaften: Programmiersprachen haben eindeutige

- Lexikalität: Heißt jedes Programm darf nur aus einer bestimmten Menge an Zeichen/Wörtern (Symbole/Tokens) bestehen.
- eindeutige Syntax: Heißt jeder Befehl hat eine eindeutige korrekte Zusammensetzung/Reihenfolge an Tokens.
- eindeutige Semantik: Die Bedeutung der Befehle ist eindeutig festgelegt. Wir unterscheiden zwischen
    - statische Semantik: Die Verwendung von Datentypen/Namen ist eindeutig. Solche Fehler können durch den Compiler gefunden werden.
    - dynamische Semantik: Das Programm funktioniert zwar, macht aber nicht das was es soll.

Compiler vs. Interpreter: Compiler übersetzt das gesamte Programm auf einmal in Binärcode wohingegen der Interpreter immer nur Befehlt für Befehl ein Programm übersetzt. Somit bietet ein compiliertes Programm meist bessere Performance wohingegen ein interpretiertes Programm oft leichter zu entwickeln ist (schnelles Ändern, geänderte Programme sofort ausführbar, leichter zu Debuggen, usw.)

Übersetzungsphasen: Zuerst wird der Quellcode in eine Folge von Tokens zerlegt. (Lexikalische Analyse)
Dann wird die Syntax überprüft und die Tokens werden zu Befehlen zusammengeführt. (Syntaktische Analyse)

Dann werden noch Typenregeln überprüft (Semantische Analyse)
Beim Compiler können dann noch Optimierungsversuche gestartet werden bevor dann final der Binärcode erzeugt wird.

Linken: Dies ist ein Schritt beim übersetzen von Code. In diesem Schritt werden bereits übersetze Programmteile (meist aus Bibliotheken) in das Programm eingebunden. Diese bereits übersetzten Programmteile liegen meist in sogenannten Objektcode vor.
Beim Dynamischen im Vergleich zum Statischen Linken wird das Einbinden dieser Programmteile erst während der Ausführung erledigt.

Virtuelle Maschinen: Virtuelle Maschinen dienen als Layer zwischen Hardware und Programmiersprache. Eine Programmiersprache wird so oft in sogenannten Bytecode compiliert (vorteile eines Compilers). Dieser Bytecode wird dann aber nicht vom CPU direkt sondern von der Virtuellen Maschinen ausgeführt (Vorteile eines Interpreters - vor allem maschinenunabhängig).
Bytecode wird dann von den Virtuellen Maschinen entweder Interpretiert oder JIT-Compiliert (vor allem bei Zeitkritischen Aktionen)
Deshalb gibt es heute kaum noch rein interpretierte/compilierte Sprachen.

Skriptsprachen: Bei Skriptsprachen handelt es sich um Sprachen die gedacht sind um vor allem kleinere Programme umzusetzen. Diese Sprachen kommen meist ohne Deklarationszwang von Variablen aus. So kann man schneller Programmieren hat dafür aber keine/kaum Überprüfung von Typenfehlern.
Python ist eine solche Skriptsprache.