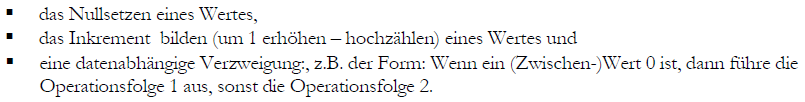
PRG1 – Zusammenfassung

Computer Def: (digitalelektronische) Maschine zur Speicherung + automatische Verarbeitung, durch flexible (programmierbare) Vorschriften.

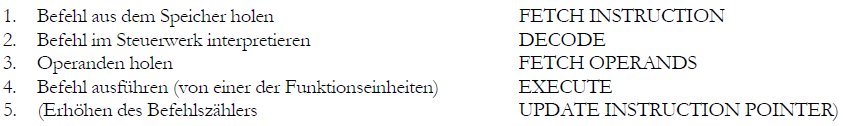
Kerneigenschaften:  
1.digitalelektronisch  
2. Speicherung von Daten  
3. Automatische Verarbeitung von Daten  
4. Programmierbare (flexible) Rechenvorschriften, d.h. Software

Bedingungen für Turing-vollständige Maschine:  


(minimal notwendig um alle berechenbaren Funktionen zu errechnen (Church-Turing-These)

Neumann-Architektur:  
5 Funktionseinheiten:  
1. Speicher (M), Steuerwerk/Leitwerk (CC), Rechenwerk, Eingabe, Ausgabe  
(Steuerwerk + Rechenwerk = Prozessor)

2. Im Speicher sind Daten und Programm abgelegt (kann nicht gleichzeitig zugegriffen werden)

3. Befehls-Ausführungszyklus:  


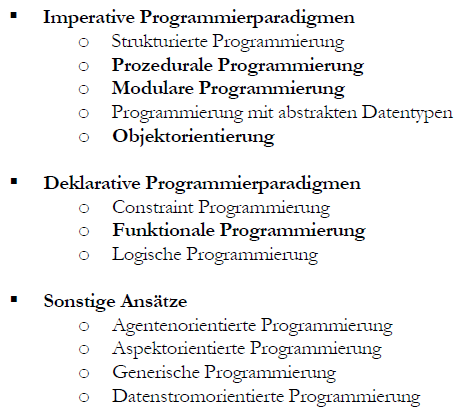
Eigenschaften Algorithmus:

Determiniertheit (bei gleichen Startwerten muss gleiches Ergebnis kommen), Deterministisch (nach jedem Schritt darf es nur einen möglichen nächsten Schritt geben), Statische Finitheit (Beschreibung des Algorithmus ist endlich), Dynamische Finitheit (Menge an Daten und Zwischenspeicherungen sind endlich), Terminiertheit (bricht nach endlicher Zeit kontrolliert ab)

Programmierspachen müssen Algorithmen (Bearbeitungsvorschriften) in einer für Computer und Menschen verständliche und lesbare Form ausdrücken.

Programmiersprachen:

1. Generation: Maschinensprache (kann Prozesse direkt ausführen)
2. Generation: Assembler (ersetzen Zahlencode durch symbolische Bezeichner = Mnemonics)
3. Generation: Höhere Programmiersprachen (Die meisten modernen Programmiersprachen + objektorierntierte Sprachen OO-Generation)
4. Generation: anwendungsbezogene (aplikative) Sprachen für Zugriff auf Datenbanken (meist SQL) oder Gestaltung von graphical user interfaces (GUIs)

Programmiersprachenparadigmen: 

Imperative Programmierung:  
Computerprogramm als lineare abfolge von Befehlern

Deklarative Programmierung:  
basierend auf mathematischen, rechnerunabhängigen Theorien (es wird nicht der Weg zum Ergebnis programmiert, sondern nur das gewünschte Ergebnis)

Funktionale Programmierung:  
Aufgabenstellung wird als funktionaler Ausdruck formuliert. Selbstständiges anwenden von Funktionsersetzungen von Interpreter oder Compiler lösen Aufgabenstellung

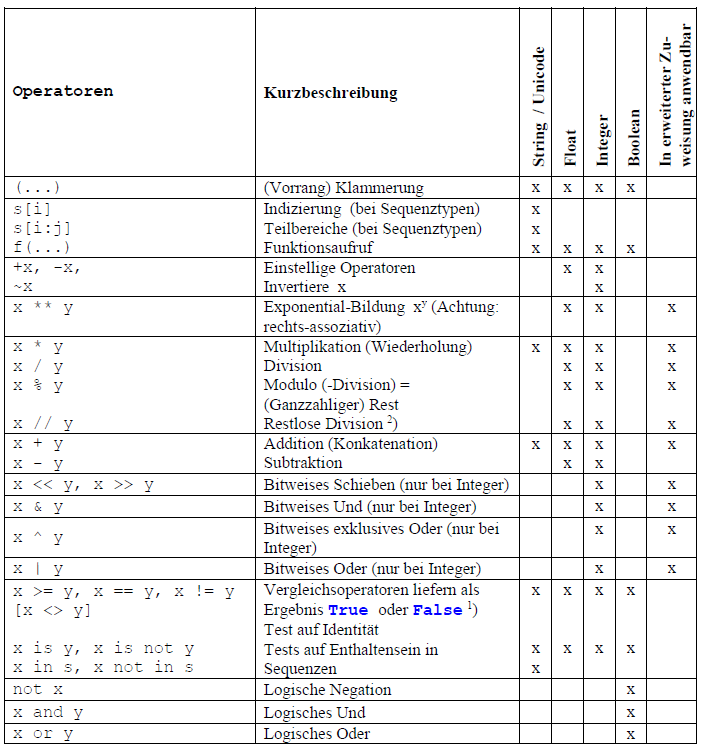
Compiler: Übersetzt (kompiliert) Quellprogramm in Assemblersprache, Bytecode oder Maschinensprache  
(während Kompilierung: Lexikalische, Syntaktische und Semantische Analyse, Zwischencodeerzeugung, Programmoptimierung, Codegenerierung) [Übersetzen – Binden – Laufen]

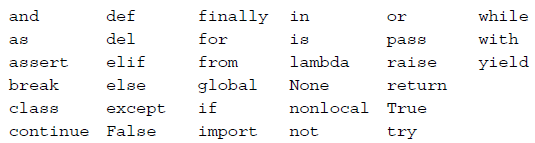
Interpreter: Übersetzt nicht, sondern liest Programm Zeile für Zeile und führt direkt aus  
(Liest erstes Statement, prüft lexikalisch, syntaktisch, semantisch, generiert Maschinencode und führt dann Maschinencode aus. Beginnt dann bei nächstem Statement)

Bytecode (Programm, das aus Befehlen für virtuelle Maschine besteht). Prüfung von Programm mittels Compiler, aber dann keine Übersetzung in Maschinensprache, sondern in einfach strukturierten Zwischencode, der dann von Interpreter in virtueller Maschine ausgeführt wird

(Vorteil: Plattformunabhängigkeit, dynamische Optimierung möglich; Nachteil: langsam)

Einer Variablen zugewiesener Wert: Literal

Prioritäten der Operatoren: 

In Python reservierte Variablennamen: 

Einer-Komplement:  
Erste Stelle der Binärzahl gibt Vorzeichen an (0 = +, 1 = -)  
Darstellung von positiver Zahl x in binär (00112)  
Darstellung von negativer Zahl -x mit allen Zahlen von x invertiert (11002)

Nachteile: Doppelkodierung (+0 / -0)  
Ändert Addition von +1 die Vorzeichenstelle, so muss Ergebnis durch Addition von +1 korrigiert werden

Zweier-Komplement:

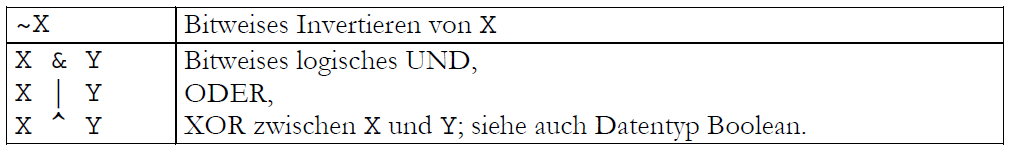
Gleiches Spiel wie Einer-Komplement für positive Zahlen und maximale negative Zahl. Alle anderen

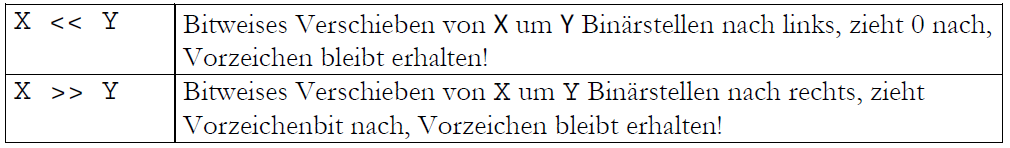
negativen Zahlen werden wie Einer-Komplement invertiert und dann +1 auf die Binärzahl gerechnet.

Von Dezimal in Zweier-Komplement:  
positiv: Wandle in binär um --> fertig

Negativ: Wandle in positive Binärzahl um, gehe von rechts nach links, nimm erste 1, lass sie stehen und invertiere restliche Zahlen

Von Zweier-Komplement in Dezimal:  
Wenn hinterste Zahl 0, dann rechne in Dezimal um  
Wenn hinterste Zahl 1, dann rechne -1 in Binär und invertiere Zahl, dann rechne zu Dezimal um

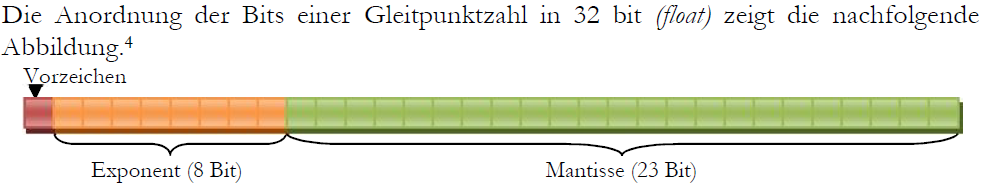
Operationen mit Bitzahlen:  


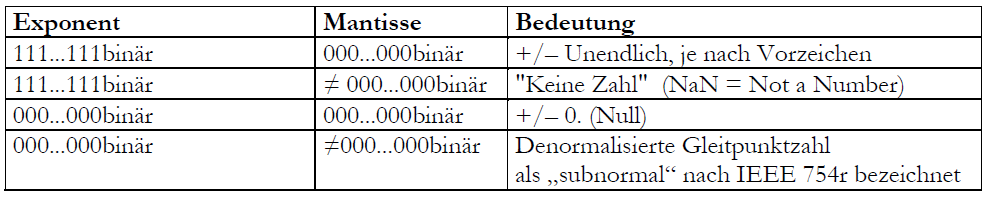


Gleichtkommazahlen (IEEE 754, bzw. IEEE 754-2008)

Standartdarstellung für binäre Gleitkommazahlen  
Format der Darstellung: x = (-1)s \* m \* 2e  
s = Vorzeichenbit, m = Mantisse, e = Exponent

Darstellung von Gleitkommazahlen in 16 (10 M, 5 E), 32 (single; 23 M, 8 E), 64 (double; 52 M, 11 E) und 128 (112 M, 15 E) Bit



Besondere Codewörter in IEEE 754:  


Unendlich ist synonym für nicht darstellbare Zahl

Keine Zahl stellt ungültige (oder nicht definierte) Ergebnisse dar, z.B. Wurzel aus negativer Zahl

Denormalisierte Zahl ist Zahl die zu klein ist um Exponent nicht als von Null verschieden zu speichern --> Veränderte Interpretation:  


Wird zu

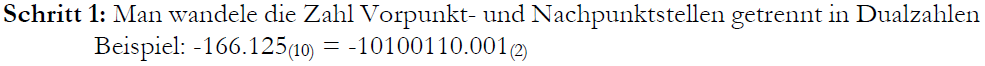


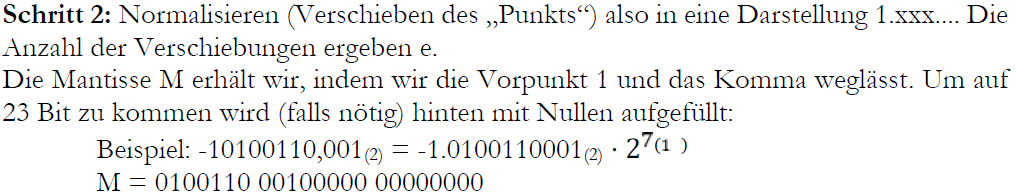
Denormalisierte Zahlen haben geringere Genauigkeit (Präzision) als normalisierte Zahlen

em ist kleinster möglicher Exponent (negativer bias, abhängig von Anzahl der Bits)

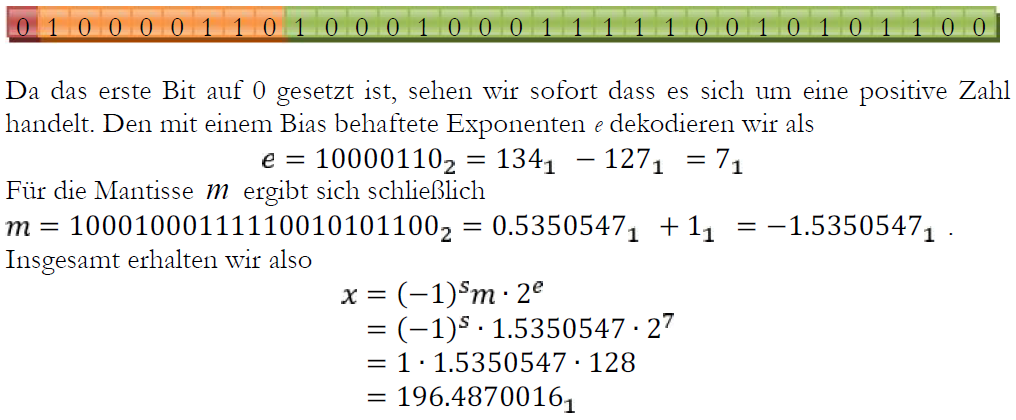
Bsp: Umwandlungen: Dezimalzahl ⬄ Gleitpunktzahl

(-166.125(10) = -10100110.001(2)

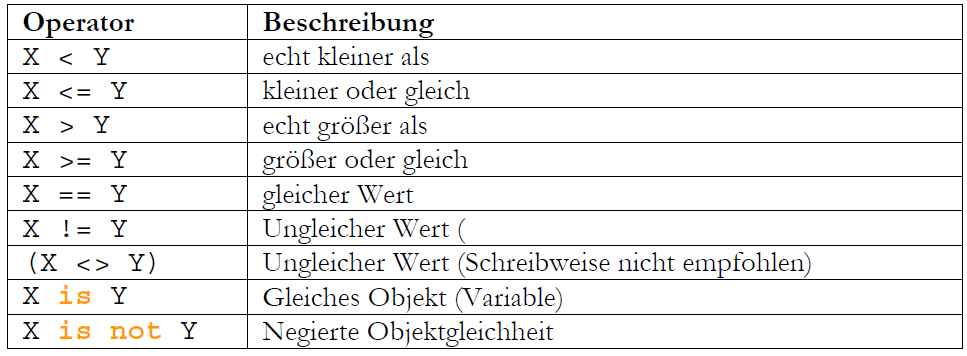






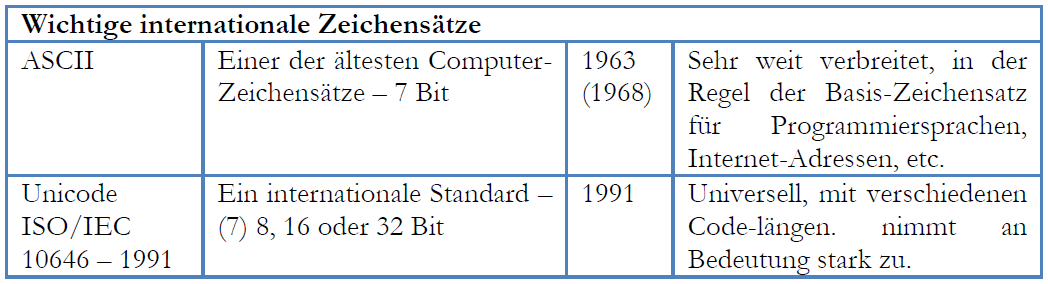
Umgekehrt:  


Gleitpunktzahlen haben unterschiedliche Genauigkeit (Präzision), abhängig von der Länge der Mantisse

Vergleichsoperatoren:  


Zeichendarstellung:  
ASCII-Code: Darstellung von Zeichen mittels 7 oder 8 Bit (viele Verschiedene Varianten)  
[American Standard Code for Information Interchange]

Unicode: 16 Bit pro Zeichen (international anerkannt) (U+00000 bis U+10FFFF, ersten beiden Ziffern identifizieren eien von 17 „Ebenen“, die letzten vier Ziffern ein Zeichen in der jeweiligen Ebene)



Starke Typisierung:  
Es existiert eine einmalig durchgeführte Bindung zwischen Variable und Datentyp. Verschiedene Datentypen können nicht miteinander „interagieren“ (integers können nicht zu strings addiert werden, etc.). Umwandlungen von Datentypen (wenn möglich) müssen explizit durch einen Befehl durchgeführt werden.

(Vorteil: Schnellere Compilerarbeit, da keine Typprüfungen nötig)

Vs.

Schwache Typisierung:  
Es existiert keine feste Bindung zwischen Variable und Datentyp. Unterschiedliche Datentypen können ggf. miteinander interagieren.

Statische Typisierung:

Zuweisung des Datentyps zu Variablen erfolgt durch Deklaration (Code der festlegt, welcher Datentyp mit einer Variable verbunden ist).

Vs.

Dynamische Typisierung:

Zuweisung von Datentyp zu Variable erfolgt automatisch während Laufzeit des Programms. Keine Typisierung von Hand während Programmierung notwendig

Python ist eine starke, dynamische Sprache

Coercion (implizite Typkonvertierung): Automatische Umwandlung von einem Datentyp in einen anderen, bei logisch sinnvollen Operationen

z.B. ein integer --> Float, wenn das integer mit einem Float addiert wird. Ergebnis ist logischerweise ein Float, ohne das Information verloren geht

Cast(ing) (explizite Typkonvertierung): Explizite Umwandlung von einem Datentyp in einen Anderen. Z.B. ein Integer in einen String

Coercion in Python:  
Bei numerischen Ausdrücken: bool --> Int --> long --> float --> complex

Bei Sting-Ausdrücken nach Schema: str --> unicode

Direkte Rekursion: Rekursion, die sich auf sich selbst bezieht

Indirekte Rekursion: Aktion, die sich auf eine andere Aktion bezieht, die sich auf die erste Aktion bezieht (Menge der Aktionen egal, solange sich der Kreis schließt)

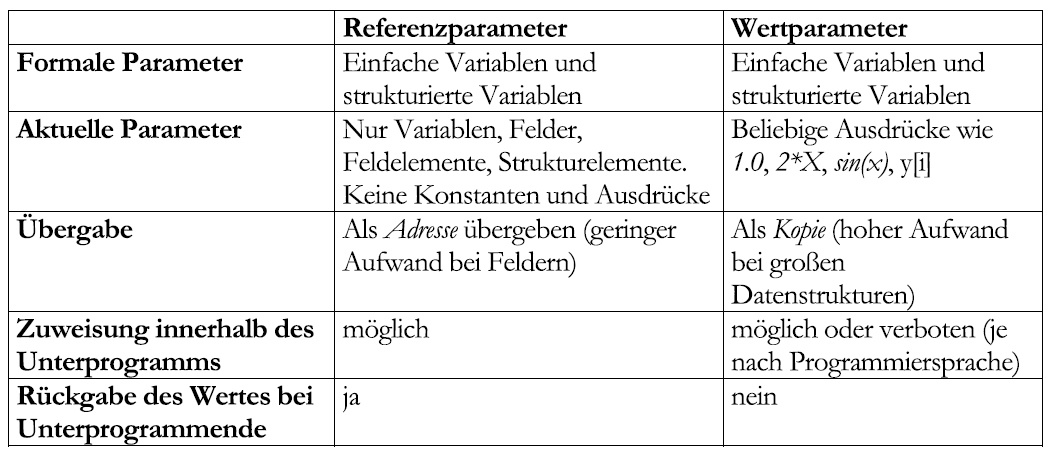
Primitive Rekursion: Rekursion, die durch eine Iteration ersetzt werden kann

Unterprogramme können Argumente haben (=Parameter), in Programmdefinition werden sie formale Parameter genannt. Bei Aufruf der Funktion werden sie durch aktuelle Parameter ersetzt

Mechanismen zur Parameterübergabe:  
Call by reference: Compiler oder Interpreter übergibt Adresse des Speicherbereichs übergibt --> übergibt einen Zeiger (=Referenz). Änderungen am formalen Parameter ändert auch aktuellen Paramerter

Call by value: Compiler oder Interpreter übergeben Wert (bei Variable). Änderungen an diesem Wert wirken sich nicht auf die ursprüngliche Variable aus.

Call by name: Werte die nicht bei Übergabe, sondern erst bei Benutzung berechnet werden (in modernen Sprachen unüblich) [sog. call by need merkt sich bereits ausgewertete Variablen und behebt damit Problem der mehrfachen Berechnung von Werten in call by name)



Funktionen: Errechnen einen Wert, der an das rufende Programm zurückgegeben wird.

Prozeduren: Führen eine bestimmte Aktion aus (Verändern entweder interne Variablen oder über Referenzparameter Änderungen an Variablen im rufenden Programm vorgenommen werden.

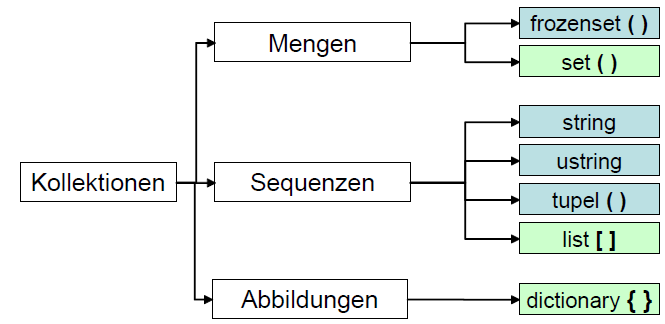
Wirkungen: Alles was in einem Programm an Aktionen stattfindet

Nebenwirkung: Alles was passiv in einem Programm durch Wirkungen verändert wird

In Python werden Parameter, die an eine Funktion weitergegeben werden über call by reference aufgerufen (können also verändert werden)

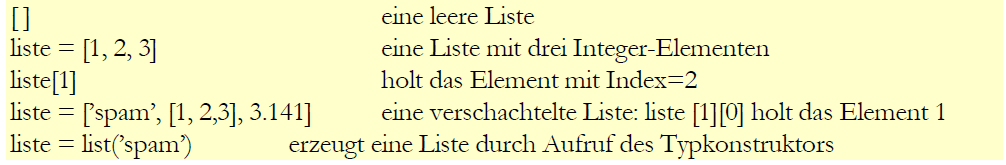
Funktionen rufen neue lokale Namensräume auf

Aggregierte Datentypen (Python)

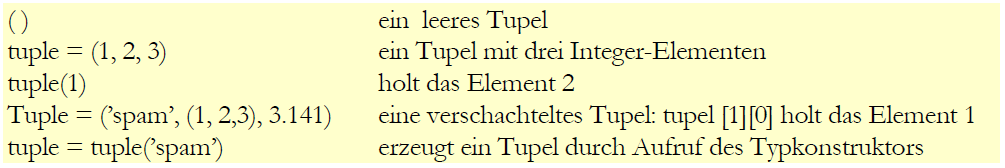


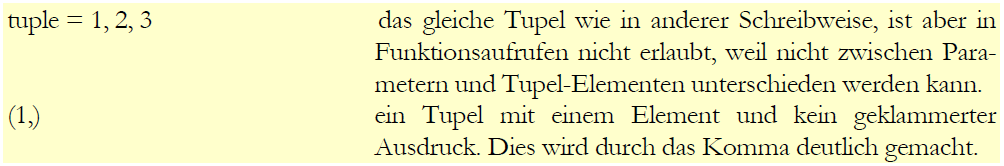
Set, list und dictionary sind mutable (veränderlich)

Frozenset, string, ustring und tupel sind unmutable (unveränderlich)

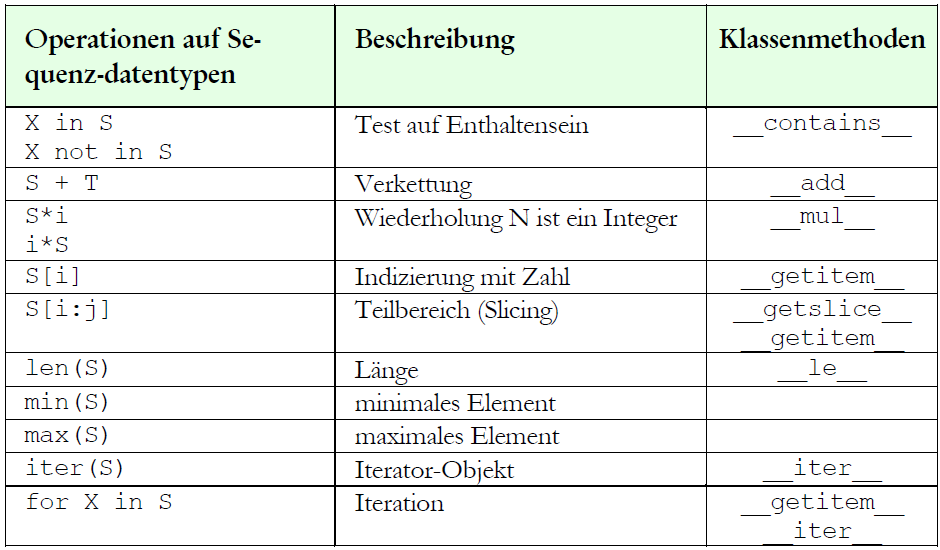
Listen: Eckige Klammern, Inhalt durch Komma getrennt

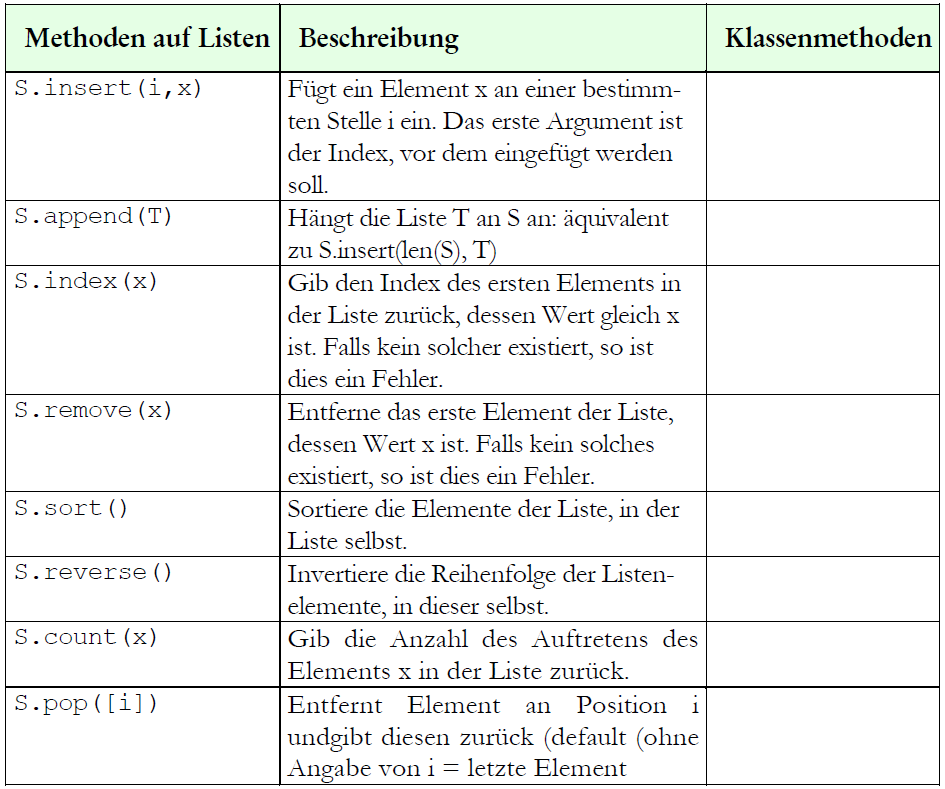
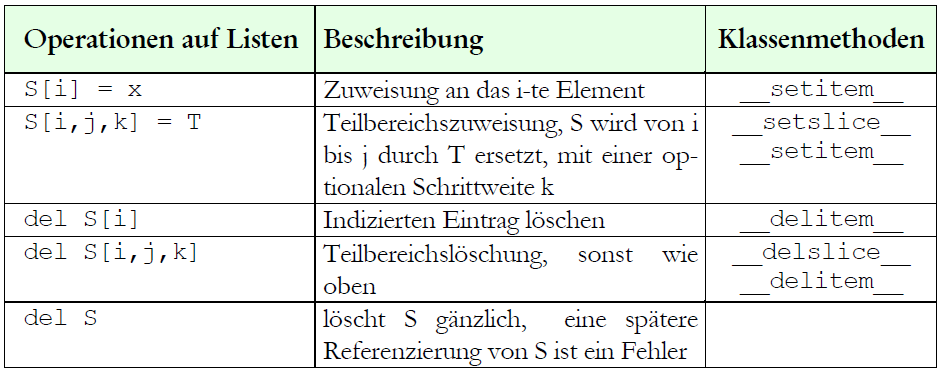
Tupel: Runde Klammern, Inhalt durch Komma getrennt



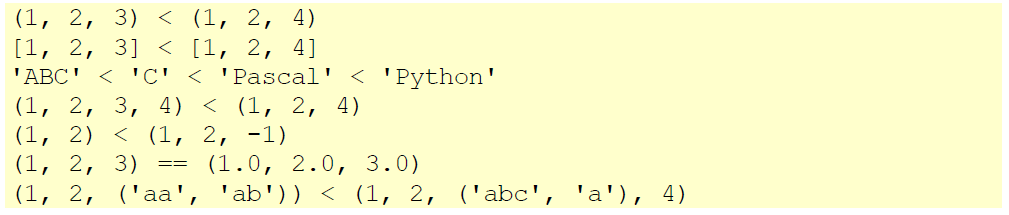


Sequenz - Operationen für Stings, Listen und Tupel:



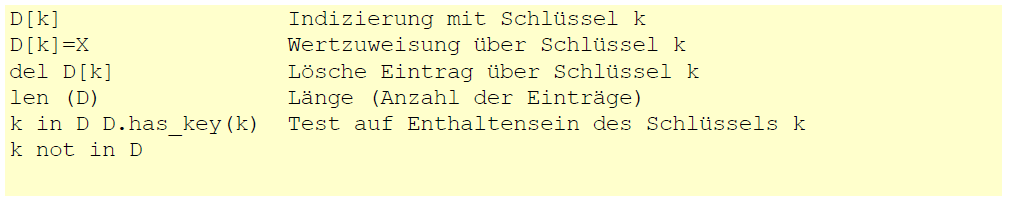
Operationen speziell für Listen (da mutable): 

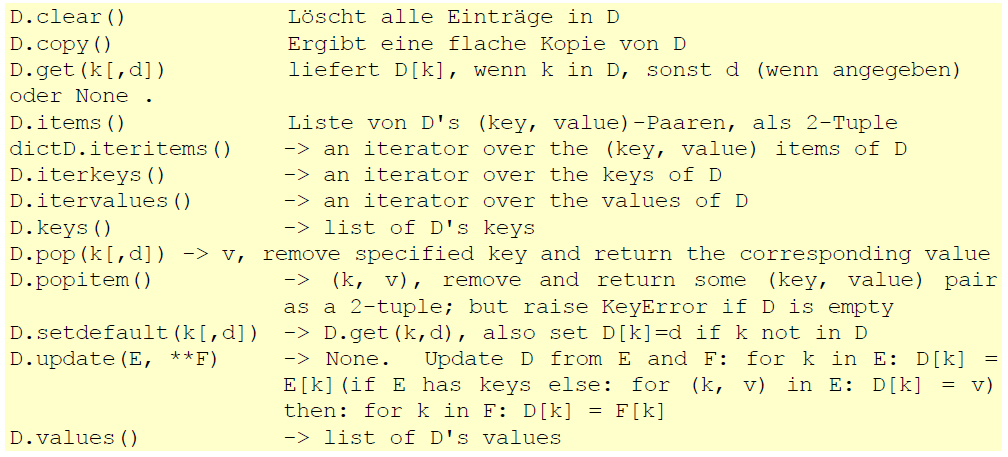
Vergeleich von Objekten desselben typs:



Zuerst wird jeweils erster Eintrag verglichen, wenn gleich, wird zweiter Eintrag verglichen, etc. Wenn beide Sequenzen gleich sind, ist die Kürzere kleiner.

Schlüssel von Dictionaries können Strings, Zahlen und Tupel sein. Listen nicht

Operationen für Dictionaries:  




Funktionen:

Filter(Prüfung, Sequenz); gibt Liste aller Werte von Sequenz aus, die True für Prüfung sind

Map(Funktion, Sequenz); Gibt Liste aller Werte von Sequenz aus, nachdem Funktion darauf angewendet wurde

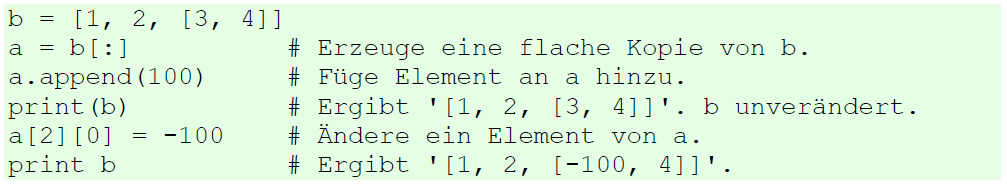
Reduce(Funktion, Sequenz); Gibt einzelnen Wert aus, der entsteht, wenn man Funktion auf nacheinander auf die Elemente von Sequenz anwendet. --> Ergebnis1 = Funktion(Sequenz[0], Sequenz[1]) --> Ergebnis2 = Funktion(Ergebnis1, Sequenz[2]) --> etc.

Objekt, dass auf eine anderes Objekt verweist = Container

Python:

Bei Zuweisung von Variablen a = b, wird kopie von a erstellt, wenn a eine Zahl oder ein String ist. Ist a allerdings eine Liste oder ein Dictionary, so ist b nur eine Referenz zu a --> Veränderungen an a führen auch zu Veränderungen an b und umgekehrt.

Flache Kopie: Kopie eines Objekts, aber Liste innerhalb des Objekts sind nur Referenzen



Tiefe Kopie: Kopie eines Objekts und rekursive Kopie aller darin enthaltenen Objekte

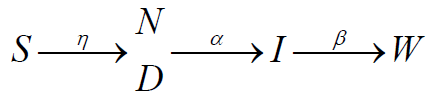
Begriffe zu Wissen:

Signal (analog/digital)

Nachricht/Daten (Speicherzustand von Information in Rechner o.ä.)

Information (Zeichenfolge, die Wissen enthält)

Wissen



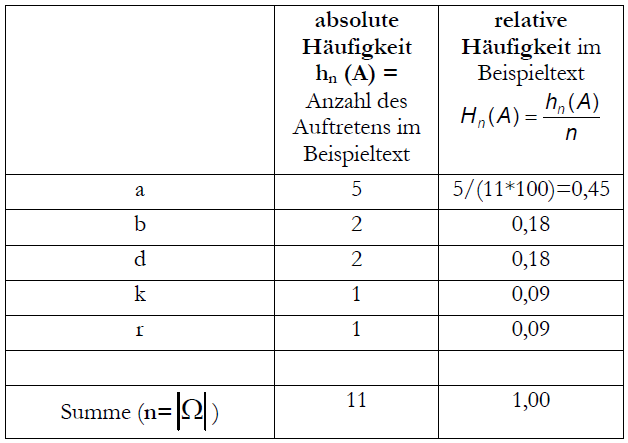
Kodierung: Zuordnung (Abbildung) von Werten eines Zeichenvorrats auf einen anderen Zeichenvorrat.

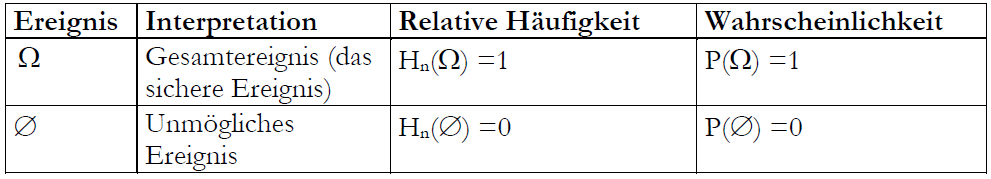
Shannonsche Informationstheorie

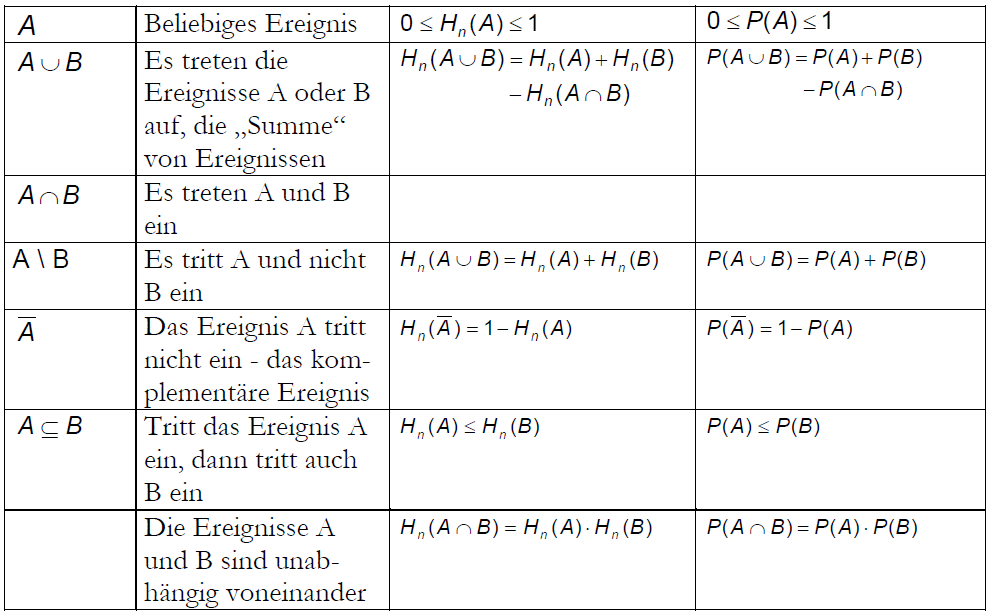
Unterschiedliche Zeichen kommen unterschiedlich häufig in Sprachen vor

--> aus dem Auftreten von selteneren Zeichen, kann man mehr Information über das gesamte Wort ziehen (Enthält ein unbekanntes Wort ein x, dann schränkt das die Anzahl der im deutschen möglichen Wörter deutlich ein)

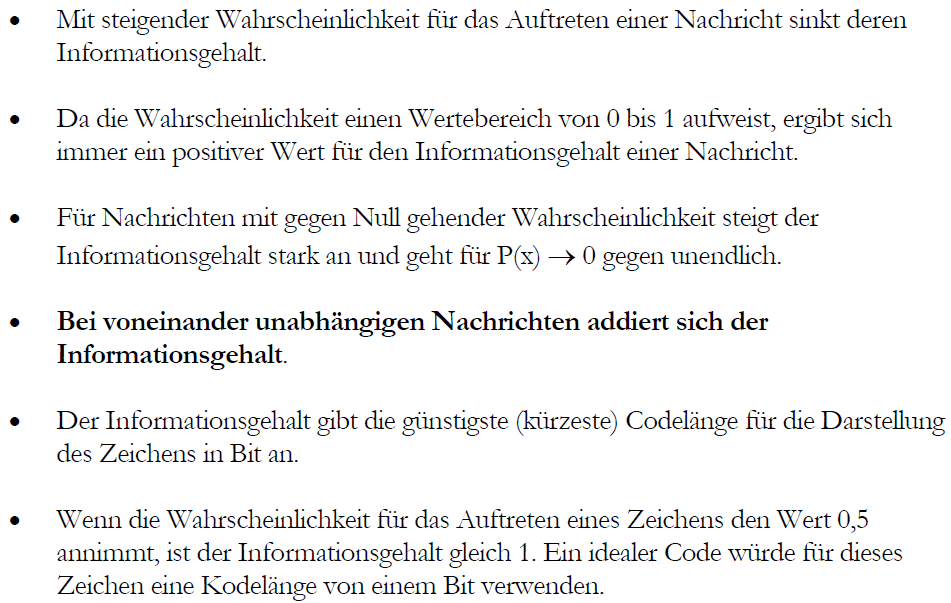
abrakadabra



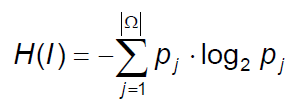




Informationsgehalt von Nachrichten:



Entropie H einer Nachrichtenquelle macht Aussage über „Maß an Zufälligkeit“ ihres Inhalts

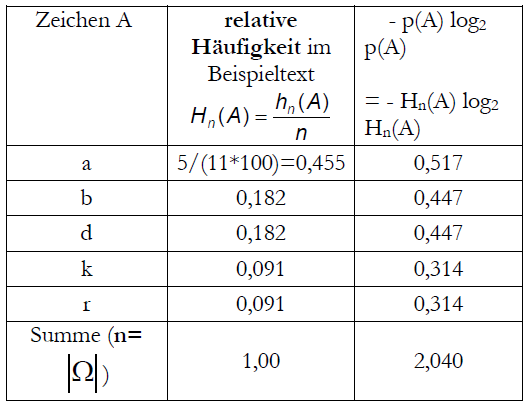


Omega = Zeichenvorrat

I = Informationsquelle (Nachrichtenquelle)

pj = Wahrscheinlichkeit mit der Nachricht auftritt

Bsp für abrakadabra:



Ergebnis ist Summe aller: (–1) \* (relative Häufigkeit(A)) \* log2(relative Häufigkeit(A))

A = Zeichen in Wort (Nachricht)

Ergebnis sind bit (hier 2.040 bit)

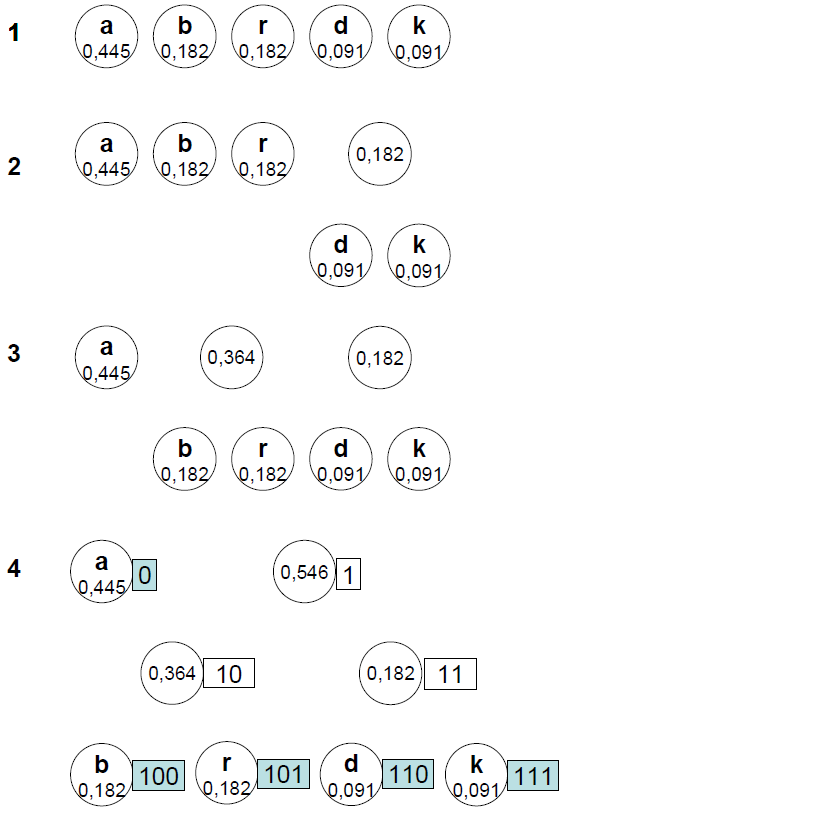
Für Binäre Nachrichtenquelle gilt: P(A) = 1 – P(B)

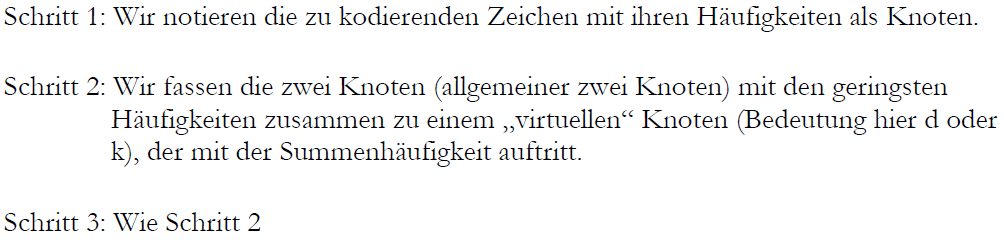
abrakadabra enthält 5 verschiedene Zeichen --> 3 Bit für Codierung notwendig

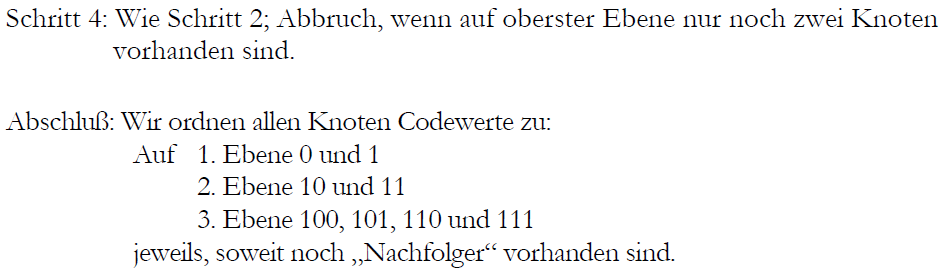
Retundanz = 3bit – 2.040 bit = 0.96 bit

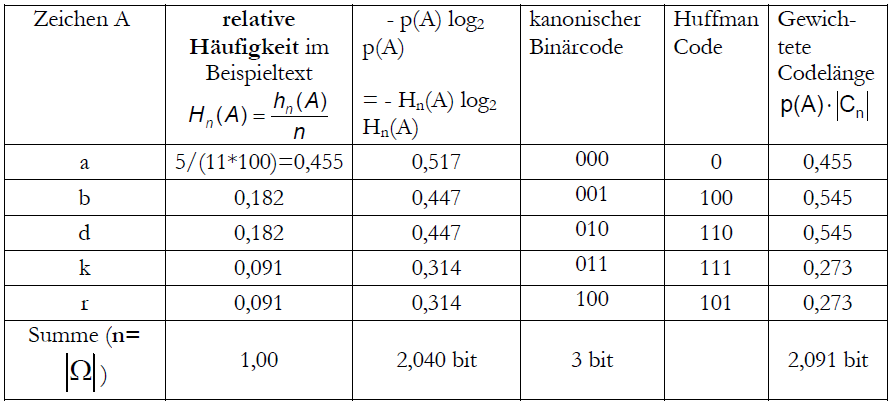
0.96 bit werden nicht genutzt

Variable Codelänge um Retundanz zu minimieren:  
Bsp Huffman Code







Huffman Code Ergebnis: 

Ergebnis bei Huffman Code 2.091

--> Retundanz = 2.091 bit – 2.040 bit = 0.051 bit

--> Deutlich niedrigere Retundanz, da Codes, die mit 0 beginnen immer sofort als a interpretiert werden

Hoffman Kodierung ist Entropiecodierung

Nyquist-Frequenz (Irgendwas mit Signalübertragung und Informationsverlust)