Programmiersprachen – gestern, heute, morgen

Informationen für die Lehrperson

Übersicht

Bei dieser Lerneinheit handelt es sich um „gelenktes entdeckendes Lernen“. Die Studierenden befassen sich über einen längeren Zeitraum (3h) mit einem Aspekt der Geschichte der Programmiersprachen. In der ersten Phase verschaffen sich die Studierenden einen Überblick über das Thema und wählen einen Aspekt aus, den sie interessiert.

In der zweiten und zeitintensivsten Phase setzen sich die Studierenden mit dem gewählten Aspekt auseinander. Es ist wichtig, dass sie genügend Zeit haben, um Entdeckungen zu machen. An der Oberfläche zu kratzen ist nicht der Sinn der Sache.

In der dritten Phase formulieren die Studierenden ihre Entdeckungen in einem 10-Minuten Vortrag. Da dieser Vortrag die Grundlage für die Bewertung darstellt, muss der Inhalt des Vortrages schriftlich formuliert sein. Auch Folien und andere Unterlagen werden abgegeben.

Erforderliches Vorwissen

Diese Unterrichtseinheit richtet sich nicht an Informatik-Neulinge! Das Entdeckungspotenzial ist höher, wenn der Studierende Erfahrungen in der Informatik gesammelt hat. Ein Jahr Informatik-Studium - an einer FH, Universität oder ETH – liefert das Grundverständnis, um die Informationsquellen vollumfänglich zu verstehen.

Programmierkenntnisse

Es ist sinnvoll, dass die Studierenden mit mindestens einer Programmiersprache gut umgehen können. Sie kennen die Grundprinzipien des Programmierens und sind mit den Problemen vertraut, die sich beim Programmieren ergeben. Sie kennen eine der gängigen imperativen Sprachen (C, Java, Pascal,...), um den Code in den Beispielen zu lesen.

Begriffsverständnis

Die Studierenden kennen alle grundlegenden Begriffe der Informatik. Dazu gehören zum Beispiel „Syntax“, „Semantik“, „Compiler“, „Interpreter“, „Assembler“, „HW-Architektur“, etc.

Ablauf

Austeilen der Unterlagen Durchlesen der Aufgabenstellung Organisatorische Hinweise durch die Lehrperson	25’
Auswahl eines Aspektes & Entdeckungsphase	180’ (inkl. Pause)
Erstellen des 10 Minuten Vortrages	45’
Vier Vorträge werden gehalten	45’
Diskussion	25’

Simulation möglicher Ergebnisse

Exemplarisch einige mögliche Vorträge:

Beispiel 1: Der Simple User Syntax Interpreter – SUSI

Einleitung:

Ich habe mich auf interpretierte Sprachen konzentriert. Es gibt viele dieser Sprachen, bekannte Vertreter sind Perl, Python und PHP. In diesen Sprachen können einfache, kleine Programme geschrieben werden. Dank der Tatsache, dass sie interpretiert sind, ist es möglich, Zeile für Zeile interaktiv abzuarbeiten. Dies bedeutet: Der Benutzer gibt eine Zeile Code ein, die dann interpretiert wird. Danach gibt er die nächste Zeile ein u.s.w.

In Python ist dieses Feature gut ersichtlich. Startet man Python, erhält der Benutzer einen Prompt „>>>“. Python wartet auf die Eingabe einer Zeile, die dann ausgeführt wird. z.B „2+2<enter>“. Python antwortet mit „4“ und einem neuen Prompt.

```
Python 2.3.3 (#51, Dec 18 2003, 20:22:39) [MSC v.1200 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> _
```

Beim nächsten Beispiel reicht reine Intuition nicht mehr:

```
>>> sin(pi)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'sin' is not defined
>>> _
```

Hier möchte ich meine erdachte Sprache platzieren. SUSI soll eine möglichst einfache, dafür nicht so mächtige Sprache unterstützen.

Die Sprache ermöglicht folgende Ausdrücke:

Einfache mathematische Operationen (Taschenrechner-ähnlich)

```
> 2+2
> sin(pi)
> sqrt(4)
> 5*2e3
```

Variablenzuweisungen

```
> a := 1
> b := sin(pi/2)
> c := a + b
```

Einfache Kontrollstrukturen

```
> if (a>0) then b else c end
```

Beliebige Kombinationen von Ausdrücken der obigen Art:

```
> x := if (a>0) then b else c end
> x := 2 + (if (a>0) then b else c end)
```

Im letzten Fall wird `x := 2+b` gesetzt wenn `a > 0` ist, sonst wird `x := 2+c` gesetzt.

Schlaufenstrukturen unterstütze ich nicht. Die Sprache kann nur mit reellen Zahlen und Booleans umgehen. Es gibt keine weiteren Datentypen.

Mögliche Einsatzbereiche:
- Taschenrechnerprogramm (Ersatz des Windows-Taschenrechner)
- Erstellen einfacher Formeln in Eingabemasken anstelle von Zahlen (ähnlich wie die Formeln in den Zellen eines EXCEL Sheets).

Zusammenfassend:
Die Sprache kann intuitiv gebraucht werden und alle wichtigen Konstanten sind vordefiniert. Es ist eine Art integrierter Taschenrechner, der in andere Programme eingebaut werden kann. So ist es dem Benutzer möglich, statt festen Werten mathematische Ausdrücke zu schreiben, die zur Laufzeit des Programms vom SUSI ausgewertet werden.

Beurteilung dieses Vortrages

Positiv:
- Aus den gegebenen Informationen werden eigene Ideen und Gedanken entwickelt.
- Realistische und durchaus sinnvolle Sprache wird vorgestellt
- Originelle Arbeit

Negativ:
- Wenig Inhalt aus den Informationsquellen

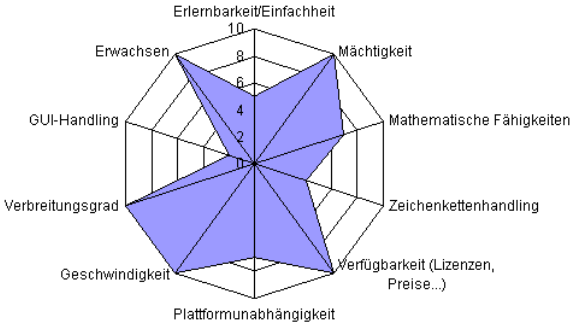
Beispiel 2: Kategorisierung von bekannten Programmiersprachen

In den folgenden 10 Minuten werde ich Ihnen die markantesten Unterschiede der bekanntesten Programmiersprachen aufzeigen. Dies geschieht durch die Einordnung der Sprachen in unterschiedlichen Kategorien.

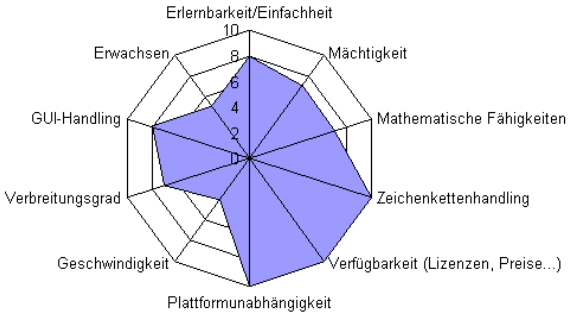
Folgende Kategorien habe ich gewählt:
- Erlernbarkeit/Einfachheit
- Mächtigkeit
- Mathematische Fähigkeiten
- Zeichenkettenhandling
- Verfügbarkeit (Lizenzen, Preise...)
- Plattformunabhängigkeit
- Geschwindigkeit
- Verbreitungsgrad
- GUI-Handling
- Erwachsen (bewährt und gereift)

Wir sehen verschiedene Diagramme. Darin sind die Sprachen in den 10 Kategorien bewertet. 10=“voll zutreffend“ und 0=“überhaupt nicht zutreffend“.

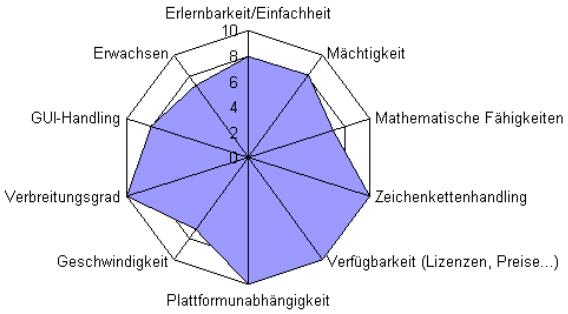
C



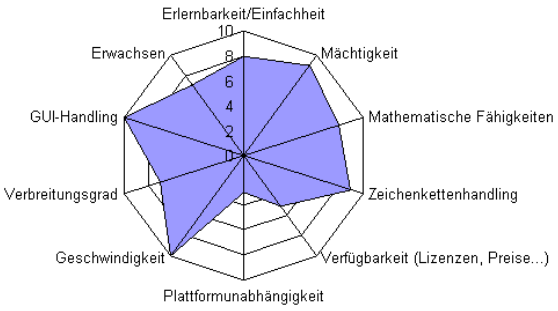
Python



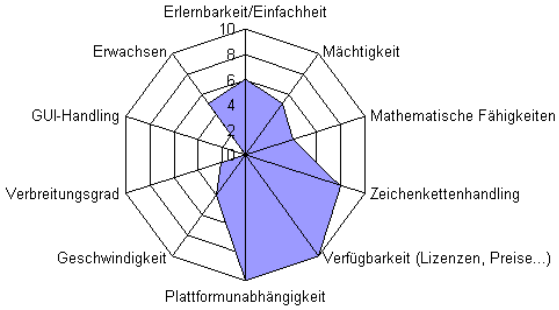
Java



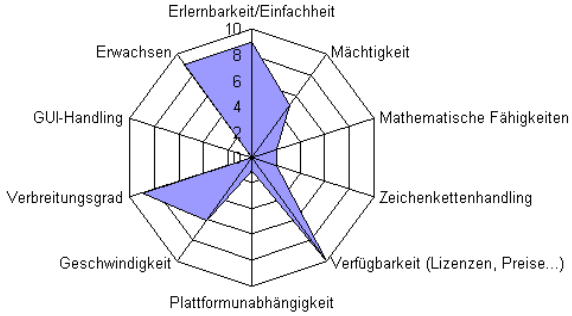
Delphi



Prolog



Unix Shell Skript (sh)



Wenn man die Grafiken betracht, kann man sich fragen, weshalb nicht alles mit Java programmiert wird, denn diese Fläche ist am grössten. Mit anderen Worten: Welche Relevanz hat diese Analyse der gewählten Sprachen?
Es können viele Aussagen gemacht werden. Ein paar Beispiele:

- Java ist eine sehr allgemein-taugliche Sprache. Sie ist meistens eine gute Wahl.
- Wenn sehr schnelle Programme benötigt werden, die auch nach langer Zeit noch funktioniere und viele systemnahe Operationen beinhalten, ist „C“ die beste Wahl. C ist schnell, ausgereift, und sehr mächtig. Mit C kann alles gemacht werden, was die Hardware erlaubt.

- Soll in kurzer Zeit eine GUI Anwendung geschrieben werden, die gut aussieht und schnell läuft, ist Delphi angebracht.

Es ist nicht möglich, alle Aspekte mit diesen Grafiken aufzuzeigen. Die Daseinsberechtigung von Prolog ist aus der Grafik zum Beispiel nicht ersichtlich. Es ist so, dass sich diese Sprache aufgrund ihrer Einzigartigkeit schlecht in die gewählten Kategorien einteilen lässt.

Wir sehen, dass die Auflistung der Kategorien nicht komplett ist. Für die gängigen Sprachen ist sie eine gute Hilfe für die Wahl der richtigen Sprache.

Beurteilung dieses Vortrages

Positiv:

- Anschaulicher, interessanter Vortrag.
- Echter Gewinn: Es wird für die gängigsten Sprachen grafisch aufgezeigt, für welche Zwecke sie geeignet sind.
- Intensive Befassung mit den Informationsquellen

Negativ:

- Das Thema ist zu gross für einen 10 Minuten Vortrag, es müssten mehr Programmiersprachen verglichen werden.

Beispiel 3: Programmier-Paradigmen

Einleitung

Seit der Einführung der ersten höheren Programmiersprache "FORTRAN" ist ein halbes Jahrhundert vergangen und über 1000 verschiedene Sprachen haben sich entwickelt. So unterschiedlich die Programmiersprachen sein mögen, sie sind alle in wenige Programmier-Paradigmen einteilbar. Diese Paradigmen stellen unterschiedliche Methoden dar, wie Programme geschrieben und vom Computer ausgeführt werden.

Imperative Programmiersprachen

Die zwei populärsten Ausprägungen sind die „prozeduralen“ und „objektorientierten“ Sprachen. Beide sind sogenannte „imperative“ Sprachen. Dies soll heissen, die Sprache besteht aus Befehlen (imperativ = befehlsorientiert), welche der Reihe nach abgearbeitet werden. Welcher Befehl als nächster abgearbeitet wird, kann mittels Sprungbefehlen beeinflusst werden.

Bekannte Vertreter prozeduraler Sprachen sind: C, Pascal, Basic, sowie die "Ursprachen" Cobol und Fortran.

Aus den prozeduralen Sprachen sind die objektorientierten Sprachen entstanden. Die grundlegende Neuerung ist die Einführung von Objekten, Objekte sind „Päcklein“ bestehend aus Daten und Programmcode. Objekte werden während das Programm läuft erzeugt und gelöscht.

Bekannte Objektorientierte Sprachen sind: Smalltalk, C++, Java, Python und Eiffel.

Die meisten bekannten Programmiersprachen sind imperative Sprachen.

```
#include<stdio.h>

int                                                          main()
{
   printf("Hello                                          World\n");
   return                                                        0;
}
```

Beispielcode: "Hello World" in C

```
/* Basisklasse "Shape" hier sind die allgemeinen Attribute einer Fläche de-
finiert */
class Shape {
public:
   Move(int x, int y);
   SetColour(ColourType c);
private:
   int xpos,ypos;
   ColourType colour;
};

Shape::SetColour(ColourType c)
{
  colour = c;
}

/* Klasse zum Zeichnen eines Kreises. Erbt die allgemeinen Attribute von
Shape */
class Circle : public Shape {
public:
    Circle();
    Draw();
private:
    int rad;
};

/* Klasse zum Zeichnen eines Vierecks. Erbt die allgemeinen Attribute von
Shape */
class Rectangle : public Shape {
public:
     Rectangle();
     draw();
private:
     int width, length;
};
```

Beispielcode: Eine Klasse „Shape" in C++ und zwei Spezialisierungen „Circle" und „Rectangle"

Logik-Programmiersprachen:
Der bekannteste Vertreter von „Logik-Programmiersprachen" ist Prolog. Prolog ist als KI[1]-Programmiersprache entstanden. Prolog besteht aus Regeln und Fakten in beliebiger Reihenfolge. An das Programm können Anfragen gestellt werden, die der Prolog-Interpreter anhand der zur Verfügung stehenden Fakten und Regeln auswer-

[1] KI = Künstliche Intelligenz

tet. Die Antwort (Ausgabe) ist „Ja“ oder „Nein“. Ist die Antwort ja, gibt der Interpreter noch die Lösungen an.

```
% Fakten
female(amy).
female(johnette).

male(anthony).
male(bruce).
male(ogden).

parentof(amy,johnette).
parentof(amy,anthony).
parentof(amy,bruce).
parentof(ogden,johnette).
parentof(ogden,anthony).
parentof(ogden,bruce).

% Regeln
siblingof(X,Y) :-
            parentof(Z,X),
            parentof(Z,Y),
            X  Y.
```

Beispielcode: ein Programm in Prolog mit Fakten (`male(bruce).`) und Regeln (`siblingof(X,Y) :- […]`)

Funktionale Sprachen:
Eine der rein „funktionalen Sprachen“ ist Haskell. Im wesentlichen bestehen funktionale Sprachen aus Gleichungen, die ausgewertet werden. Der Quellcode besteht aus einer Auflistung von Gleichungen. Wird das Programm ausgeführt, sucht sich der Interpreter die passende Gleichung, und ersetzt die linke Seite durch die rechte Seite. Diese wird dann analog weiter bearbeitet. Es ist daher möglich mit funktionalen Sprachen eine mathematische Aufgabe leicht zu implementieren, da nur die Gleichungen aufgeschrieben werden müssen, der Interpreter kümmert sich um das korrekte Einsetzten und Auswerten.

```
fun fib(0) = 0
|   fib(1) = 1
|   fib(n) = fib(n-1) + fib(n-2)
```

Beispielcode: Berechnung von Fibonacci-Zahlen mittels SML

Innerhalb dieser Paradigmen gibt es diverse Ausprägungen. Es folgen zwei Beispiele solcher Ausprägungen.

Datenstrukturen:
Ein berühmter Vertreter der Ausprägung "Datenstrukturen" ist die Sprache LISP. Ursprünglich entstanden als KI-Programmiersprache wurde LISP in vielen Bereichen eingesetzt und hat sich zu einer beliebten Sprache entwickelt. Dies ist angesichts der umständlichen Notation erstaunlich. Die grundlegende Datenstruktur von LISP sind Listen, selbst der Programmcode ist eine Liste.

```
; Funktion zum vereinigen von zwei Listen SET1 und SET2 ohne Duplikate.
(DEFUN UNION (SET1 SET2)

        ; wenn SET1 leer ist, nimm SET2 als Lösung
        (COND((NULL SET1)SET2)

        ; wenn das erste Element von SET1 in SET2 ist, ignoriere es und
        ; berechne die Vereinigung ohne dieses Element.
        ((MEMBER(CAR SET1)SET2)(UNION (CDR SET1) SET2))

        ; ansonsten berechne Union für SET1 ohne das erste Element
        ; und SET2 und füge das erste Element dem Ergebnis hinzu.
        (T(CONS(CAR SET1)(UNION(CDR SET1)SET2))))))
```

Beispielcode: Union von zwei Sets in LISP

Esoterische Programmiersprachen:
Neben seriösen Sprachen gibt es „esoterischen Programmiersprachen“. Dazu gehören Sprachen, die nicht für einen ernsthaften Zweck entwickelt wurden, sondern aus Freude am Programmieren entstanden sind. Ein bekannter Vertreter dieser „sinnlosen“ Programmiersprachen ist „Brainfuck“.

```
++++++++++[>+++++++>++++++++++>+++>+<<<<]>++.>+.+++++++..+++.>++.<<++++++++
+++++++.>.+++.------.--------.>+.>.
```

Beispielcode: „Hello World“ in Brainfuck. Eine Erklärung des Programms würde den Rahmen dieses Vortrages sprengen.

Beurteilung dieses Vortrages

Positiv:

- Gut strukturierter Vortrag.
- Einbezug mehrerer Quellen und Zusammenführen der Erkenntnisse.
- Beispiele

Negativ:

- Ein guter Schluss würde den Vortrag aufpeppen.

Weitere mögliche Ergebnisse kurzgefasst

- Ein Student befasst sich mit nur einer Programmiersprache. Er untersucht die Hintergründe und Umstände, welche zu der Entstehung der Sprache beigetragen haben. Dazu gehört die Vorstellung der Person, welche die Sprache entwickelt hat. Er setzt die Sprache in einen grösseren Kontext.

- Ein Student sucht im Stammbaum der Programmiersprachen eine Ahnenlinie, und zeigt die Entwicklung von der ersten Sprache dieser Linie bis zur letzten. Weiterführende Informationen wie zum Beispiel die Angabe von Gründen, weshalb sich die Sprache weiterentwickelt hat, machen den Vortrag zu einer sehr guten Arbeit.

- Der Vortrag besteht aus einer Gegenüberstellung von zwei aktuellen Programmiersprachen. Es werden Argumente für die eine Sprache aufgeführt, jedoch werden diese durch Argumente für die zweite Sprache aufgehoben.

Ziel: Aufzeigen, dass alle Sprachen ihre Berechtigung haben und dass es keine „beste“ Programmiersprache gibt.

- Ein Student beschäftigt sich mit dem Begriff „Morgen“ im Titel der Aufgabenstellung und erstellt in seinem Vortrag eine Zukunftsvision. Er beschreibt, wie die Programmierung von Computern in 10, 20 oder 100 Jahren aussehen könnte. Natürlich ist dies weder richtig noch falsch zu werten. Eine einleuchtende Argumentation sowie eine originelle Bearbeitung des Themas sind hier die entscheidenden Kriterien.

- [...]

Die Schlussphase

Nachdem die Studierenden ihre Vorträge erstellt haben, werden vier Vorträge gehalten. Es liegt an der Lehrperson, die Vorträge auszuwählen. Wählen Sie vier Vorträge die einen spannenden und originellen Eindruck machen.

Bewertungskriterien

Generelles

- Anspruch des Aspektes
- Inhaltliche Korrektheit
- Originalität
- Dauer
- Bezug zum Thema

Alle Kriterien sollten einzeln bewertet werden. Die Note für den Vortrag setzt sich aus den einzelnen Bewertungen sowie dem Gesamteindruck zusammen.

Abweichung vom reinen Durchschnitt der Kriterien

Ist zum Beispiel der Anspruch eines gewählten Aspektes gering, werden diese aber originell vorgetragen, kann das Kriterium Originalität besonders gewichtet werden, um trotzdem eine sehr gute Note zu geben. Der Gesamteindruck wird als Bonus gewichtet und sollte begründet sein.

Speziell zu beachten

- Kommunizieren Sie die Einzelnoten
- Kommentieren Sie Ihre Bewertung
- Honorieren Sie ausgefallene Ideen!