Eckart Modrow

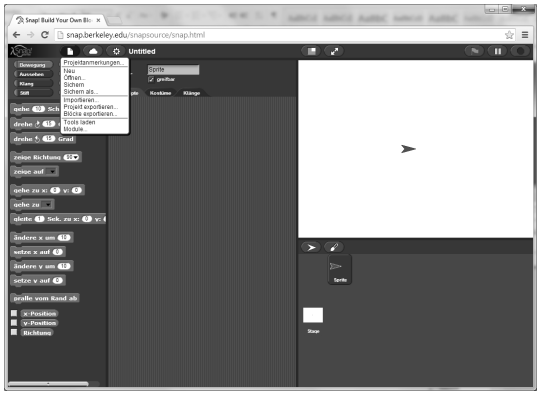
Neues von BYOB / Snap!

Abstract:
In diesem Artikel werden Änderungen, Neuerungen und einige Erweiterungen von BYOB4 / Snap! anhand von Beispielen vorgestellt, die insbesondere die Möglichkeiten des neuen http-Blocks illustrieren.

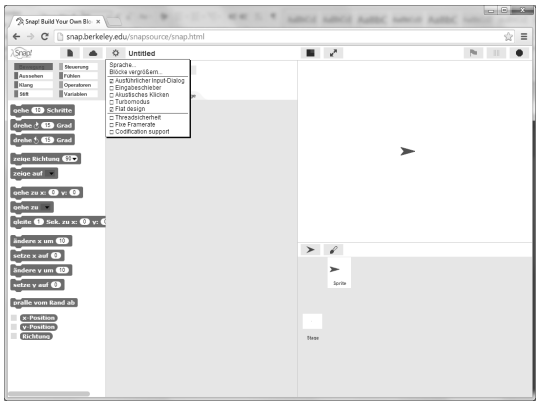
1. Was ist neu?

BYOB (BYOB3) in seiner vierten Version heißt jetzt Snap! (mit Ausrufungszeichen) und ist derzeit unter (Snap!, 2013) zu finden. Auch wenn es sich noch um eine Betaversion handelt, lassen sich die wesentlichen Änderungen erkennen. Was ist neu? Alles ist neu. Im Gegensatz zu den älteren Versionen, die auf Scratch und so letztlich auf Smalltalk aufsetzten, handelt es sich um eine völlige Neuimplementation auf der Basis von HTML5 und JavaScript. Snap! läuft als Browseranwendung auf allen gängigen Systemen, insbesondere auch mobilen wie den Tablets, und benötigt keine Installation. Besteht keine schnelle Internetverbindung, dann kann es auch als gespeicherte Webseite gestartet werden.

Bedingt durch die geänderte technische Basis laufen Anwendungen etwa um den Faktor 15 schneller als bisher – je nachdem, welche Blöcke verwendet werden. Ältere BYOB-Projekte können durch einfaches „Ziehen und Fallenlassen“ der Projektdatei auf die Skriptebene von Snap! (meist) importiert werden, ebenso Scratch-Projekte. Mit Bildern und Klängen verfährt man ebenso, muss das aber auch, weil der aktuelle Kostümeditor nicht so recht an den alten heranreicht.

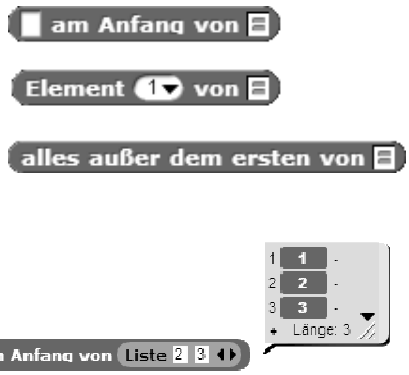


Von der Oberfläche her wirkt Snap! etwas „seriöser“ als BYOB – besonders, wenn man das „flat design“ aus der Werkzeugkiste wählt. Der verspielte Alonzo wurde durch einen Richtungszeiger ersetzt. Mag sein, dass das dazu beiträgt, das Werkzeug ernster zu nehmen. In der Werkzeugkiste findet sich auch die Möglichkeit, die Blöcke größer darzustellen. Auf Tablets ist das hilfreich. Benötigt man Skriptbilder etwa für Arbeitsblätter, dann sollte man das flat Design verwenden, weil sich der Hintergrund leichter entfernen lässt.



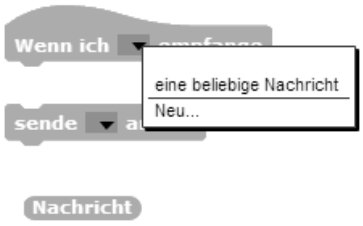
Geändert haben sich die Möglichkeiten, Elemente zu laden und zu speichern. Als Speicherorte stehen der benutzerspezifische Speicherbereich des Browsers sowie eine Cloud zur Verfügung, in der Projekte auch veröffentlicht werden können. Als dritte Möglichkeit lassen sich Projekte als XML-Dateien exportieren und an beliebiger Stelle speichern. Solche Dateien mit Projekten, Klängen, Kostümen oder Blöcken lassen sich natürlich wieder importieren oder auf den entsprechenden Snap!-Bereich ziehen. Besonders interessant ist es, selbst geschriebene Blöcke oder Blockgruppen einzeln zu speichern und wieder zu laden. Auf diese Art können Bibliotheken für bestimmte Einsatzbereiche zusammengestellt werden. Beispiele dafür finden sich unter dem Menüpunkt Module … sowie in der stark erweiterten Tools-Bibliothek.

Unter den neuen Blöcken von Snap! sind besonders die erweiterten Listenoperationen für die Schule interessant. Zusätzlich zu den alten Möglichkeiten von BYOB, die eher den Zugriff auf dynamische Reihungen beschreiben, gibt es jetzt drei neue Funktionen (Reporter), die mit anders implementierten Listen arbeiten und in etwa den traditionellen LISP-Operationen cons (nicht ganz), car und cdr entsprechen. Die erste erzeugt eine neue Liste, ohne die alte zu verändern, die beiden anderen sind selbsterklärend. Die beiden Darstellungsformen von Listen werden bei Bedarf automatisch ineinander umgerechnet. Da das Zeit kostet, sollte man sie nicht unnötig mischen. Notwendig erscheint das auch nicht, weil die erste Repräsentation eher bei imperativen, die zweite bei funktionalen, meist rekursiven Lösungen benötigt wird.



In der Steuerung-Rubrik finden wir eine Gruppe von Blöcken mit grauen Ringen. Diese zeigen, dass hier Skripte als Parameter erwartet werden und ersetzen die the script- und the block-Kacheln von BYOB. Folglich finden sich die Ringe auch unter den Operatoren und in den entsprechenden Kontextmenüs. Beispiele für ihre Verwendung kann man z. B. im meinem Skript „Informatik mit BYOB“ (ImB, 2013) finden.

Lässt man Lernenden mit Scratch / BYOB genügend Freiheiten, dann wird ein großer Teil Lösungen mit Botschaften realisieren, während andere lieber die algorithmischen Grundbausteine verwenden. Da sich mit beiden Versionen die gleichen Operationen durchführen lassen, ist die Erweiterung des Botschaftensystems für den Unterricht bedeutsam, weil sie Freiräume für die Lernenden schafft. In Scratch / BYOB konnten zwar beliebige Botschaften versandt werden, indem man den join-Zeichenkettenoperator verwendet, aber es konnte nur auf bestimmte vorzugebende Botschaften reagiert werden. In Snap! ist das anders: da auf eine beliebige Nachricht reagiert werden kann, lassen sich Nachrichten z. B. aus den drei Blöcken (Sender, Empfänger, Nachricht) zusammensetzen und von den adressierten Sprites auswerten. In den folgenden Beispielen nutzen wir diesen Weg. Da Nachrichten von allen Beteiligten lesbar sind, ergibt sich direkt der Bedarf z. B. von Verschlüsselungsverfahren, um andere als den Adressaten von Nachrichtenwegen auszuschließen. Auch komplexere Algorithmen wie etwa Routingverfahren lassen sich auf diese Weise direkt implementieren.



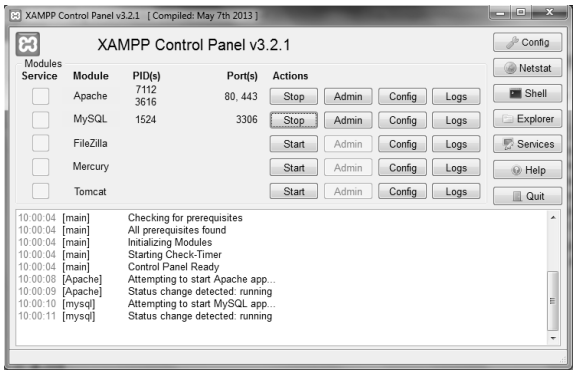
Die völlig neu hinzugekommenen continuations erfordern einen eigenen Artikel und es fehlt bisher Vererbung durch Delegation. Diese soll aber in der nächsten Version wieder bereitgestellt werden. Als Browseranwendung kann Snap! leider nicht mehr direkt auf technische Geräte wie LEGO-WeDo oder das Picoboard zugreifen. Hier zeichnen sich allerdings weitergehende Lösungen ab (s.u.). Und es fehlt leider immer noch ein Block zum Lesen des aktuellen Farbwerts. In den folgenden Abschnitten wird anhand von Beispielen etwas genauer auf einige Neuerungen eingegangen.

Zu erwähnen sind noch die Möglichkeiten, entweder alle Standardblöcke einer Rubrik oder einzelne (s. Kontextmenüs) auszublenden. Da einzelne Blöcke gespeichert und in anderen Projekten geladen werden können, lassen sich Snap!-Konfigurationen für spezielle Zwecke zusammenstellen. Als Beispiel mag die unten angegebene Konfiguration ohne Variablen-

Blöcke, mit Rekursion und nur der Alternative als Kontrollstruktur dienen, in der sich funktional andere Sprachen mit eigenen Kontrollstrukturen gestalten lassen.

2. Zugriff auf Datenbanken

Der neue http-Block ermöglicht den Zugriff auf externe Ressourcen unter Kontrolle der Benutzer, und er kann auch genutzt werden, um die Funktionalität von Snap! zu erweitern. Auf der Snap!-Website findet sich bereits eine Liste mit unterschiedlichen Robotermodellen und anderen Erweiterungen, die diesen Weg gehen. Am Anschluss von Picobord, LEGO-Geräten und des Arduino wird gearbeitet. Wir wählen deshalb ein anderes Beispiel: den Zugriff auf Datenbanken. Dazu benötigen wir z. B. mySQL und einen Webserver wie z. B. Apache. Da es beides zusammen im auch an Schulen verbreiteten Paket XAMPP (XAMPP) gibt, benutzen wir dieses. Wir starten das XAMPP-Controlpanel und mit diesem mySQL und Apache. Packen wir jetzt geeignete PHP-Skripte in das Verzeichnis htdocs von XAMPP, dann können wir diese von Snap! aus über den http-Block starten und das Ergebnis als Zeichenkette auswerten.

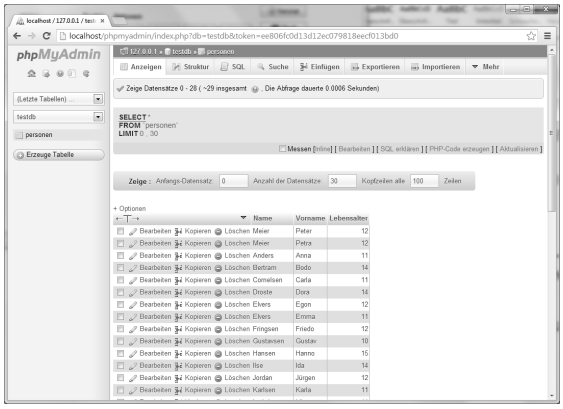


Zuerst schreiben wir ein PHP-Skript, um zu probieren, ob wir Zugriff auf die gewünschte Datenbank haben. Dabei freuen wir uns über die grauenhafte PHP-Syntax und dass wir endlich wieder textbasiert programmieren können.

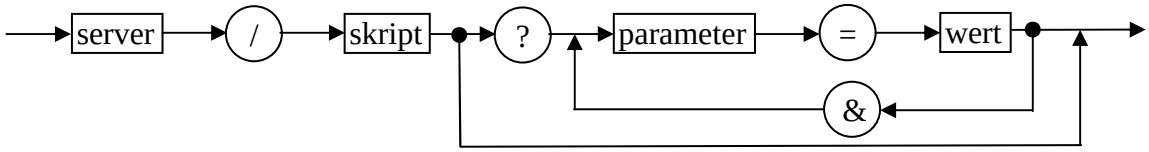
```
<?php
  // Zugriff auf die Daten erlauben
  header("Access-Control-Allow-Origin:*");
  // Verbindung zum Datenbankserver herstellen
  $verbindung = mysql_connect($_GET['server'],$_GET['user'],$_GET['password']);
  if(!$verbindung)
    echo "Fehler: keine Verbindung zum Datenbankserver.";
  else{
    // Datenbank auswählen
    $db = mysql_select_db($_GET['datenbank'],$verbindung);
    if(!$db) echo "Fehler: Datenbank nicht gefunden";
    else echo "ok";
  }
?>
```

Das Skript erwartet vier Parameter server, user, password, datenbank und liefert im Erfolgsfall die Zeichenkette „ok“, sonst eine entsprechende Fehlermeldung – und Dank an Jonatan Mosner für sein Beispielskript.

Jetzt benötigen wir eine mySQL-Datenbank, wenn wir noch keine haben. Am einfachsten erledigen wir dieses mit dem phpMyAdmin, den wir erreichen, wenn wir bei gestartetem Apache-Server in einem Browser die Adresse localhost eingeben. Links können wir das gewünschte Programm wählen. In diesem richten wir eine Datenbank testDB mit einer Tabelle personen ein, in die wir einige Daten eintragen.



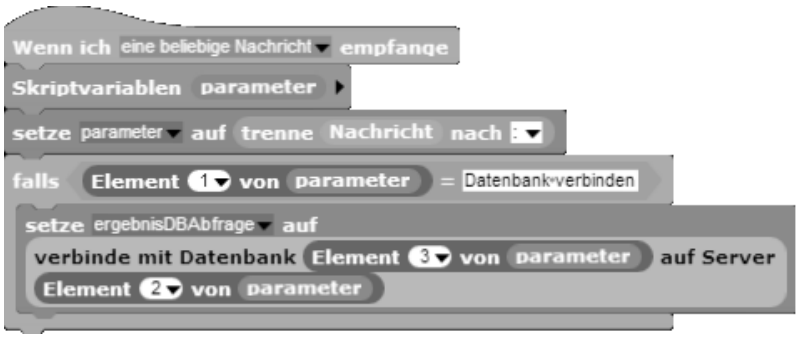
Skripte können mit einer http-Botschaft an den Server gestartet werden, die die folgende Struktur hat:



Diese Botschaft basteln wir uns in einem Snap!-Funktionsblock zusammen, der den Servernamen und den Namen der Datenbank als Parameter erhält. Den Namen unserer neuen Datenbank wählen wir als Default-Wert. Das Ergebnis des http-Blocks geben wir als Funktionswert zurück. (Der verbinde-Block verkettet jetzt eine beliebige Zahl von Zeichenketten.)

Binden wir diesen Block an ein Datenbank-Sprite DB-Sprite, das die Fähigkeiten zur Kommunikation mit Datenbanken haben soll, dann können andere Sprites dieses DB-Sprite bitten, eine Datenbankabfrage zu starten. Die erforderlichen Parameter sollen als Zeichenkette in der Botschaft übergeben werden. In unserem Fall setzen wir die Parameter mit etwas Text zu einer einzigen Zeichenkette zusammen, deren Teile durch Doppelpunkte getrennt sind.

Das DB-Sprite empfängt eine Nachricht und zerlegt diese in ihre Teile – immer zwischen Doppelpunkten. Anhand des ersten Teils kann es entscheiden, ob es die Botschaft „versteht“. Die erforderlichen Parameter müssen sich dann in den nächsten Elementen der „Zerlegeliste“ finden, die wir mit dem neuen trenne… nach-Block erzeugt haben.

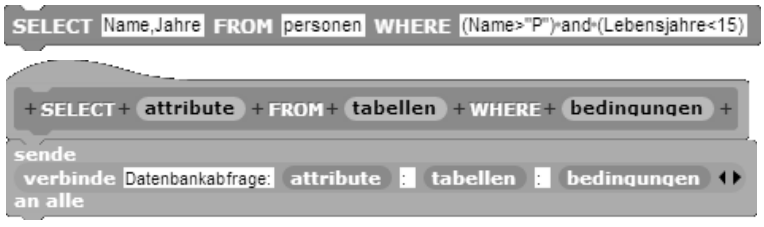


Wir spendieren uns jetzt ein weiteres PHP-Skript, mit dem wir beliebige Datenbankabfragen vornehmen können.

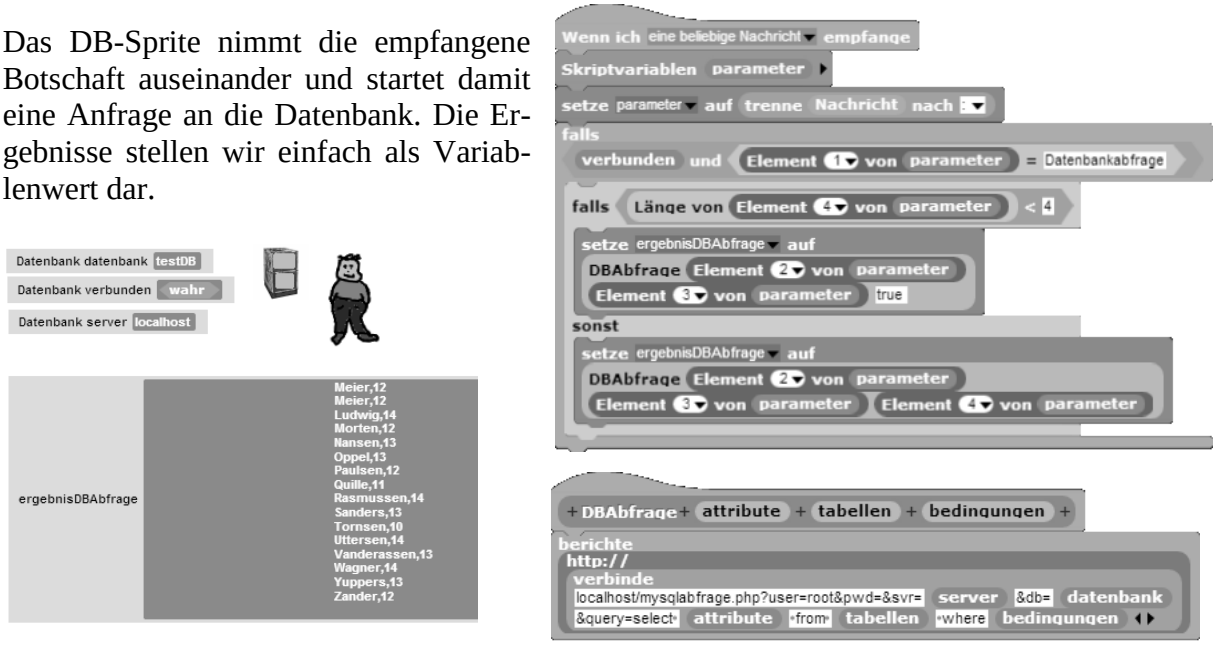
```
<?php
  // Zugriff auf die Daten erlauben
  header("Access-Control-Allow-Origin:*");
  // Verbindung zum Datenbankserver herstellen
  $verbindung = mysql_connect($_GET['svr'],$_GET['user'],$_GET['pwd']);
  if(!$verbindung) echo "Fehler: keine Verbindung zum DB-Server.";
  else{ // Datenbank auswählen
    $db = mysql_select_db($_GET['db'],$verbindung);
    if(!$db) echo "Fehler: DB nicht gefunden";
    else{ // Anfrage übermitteln
      $query = $_GET['query'];
      if($query == "") echo "leere Anfrage";
      else{
        $ergebnis = mysql_query($query);
        //Daten zusammenbasteln
        if(!$ergebnis) echo "fehlerhafte Abfrage";
        else{
          $daten = "";
          $maxzeilen = mysql_num_rows($ergebnis);
          $maxspalten = mysql_num_fields($ergebnis);
          if($maxzeilen==0 || $maxspalten==0) echo "keine Daten vorhanden";
          else{
            for($zeile = 0; $zeile < $maxzeilen; $zeile++){
              for($spalte = 0; $spalte < $maxspalten; $spalte++)
                $daten = $daten.mysql_result($ergebnis,$zeile,$spalte).",";
              $daten = substr($daten,0,strlen($daten)-1)."\n";
            }
            print_r(substr($daten,0,strlen($daten)-1));
  }}}}}
```

Auch hier freuen wir uns über die Schönheit und Lesbarkeit der textbasierten Programmierung. Das Skript erwartet zusätzlich zu den oben genannten Parametern eine SQL-Anweisung query und liefert entweder Fehlermeldungen oder eine Zeichenkette, in der die Datensätze durch Zeilenvorschübe getrennt sind und die Daten innerhalb dieser Sätze durch Kommas. Die Nutzung entspricht der des ersten Skripts:

Der Benutzer setzt eine Botschaft an das DB-Sprite ab, die die erforderlichen Informationen enthält. Hier: durch Doppelpunkte getrennt.



Das DB-Sprite nimmt die empfangene Botschaft auseinander und startet damit eine Anfrage an die Datenbank. Die Ergebnisse stellen wir einfach als Variablenwert dar.



Was können wir damit anfangen? Im einfachsten Fall kopieren wir die PHP-Skripte in das richtige Serververzeichnis und stellen das Datenbank-Sprite als Bibliothek zum Laden bereit. Anwendersprites können dann über Botschaften die Datenbankanfragen stellen und auswerten.

Da serverseitig nur die beiden angegebenen Skripte benötigt werden, liegt die Erfassung, Kontrolle und Aufbereitung der erforderlichen Daten vollständig bei Snap!. Hier bietet sich ein weites Feld z. B. bei Sicherheitsproblemen, Fehlerbehandlung usw. Auch die Auswertung der übermittelten Daten liefert reichlich Teilprobleme bei Datenstrukturen und z. B. Diagrammerstellung, die weitgehend disjunkt und damit sehr gut arbeitsteilig zu lösen sind.

Und natürlich haben wir jetzt die Möglichkeit, Informationen dauerhaft zu speichern und ggf. zwischen verschiedenen Snap!-Instanzen auszutauschen. Damit eröffnen sich alle Möglichkeiten, die bei BYOB durch die Mesh-Funktionalität gegeben waren.

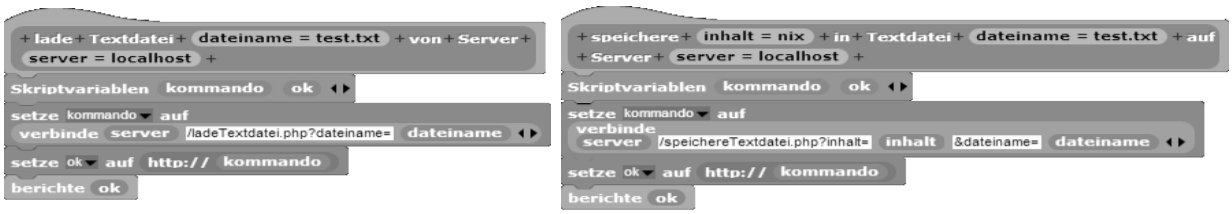
3. Zugriff auf Textdateien

Wenn schon mal ein Webserver läuft, dann können wir den auch weitergehend nutzen. Wir schreiben zwei PHP-Skripte, die eine Zeichenkette in eine Textdatei schreiben bzw. eine Textdatei in eine Zeichenkette lesen und übergeben – ohne jeden weiteren Komfort.

```
<?php
  // Textdatei lesen
  //zuerst den Parameter auslesen
  $dateiname = $_GET['dateiname'];
  if(!file_exists($dateiname))
    echo "Datei nicht vorhanden.";
  else{ //Datei öffnen
    $datei = fopen($dateiname,"r");
    //Inhalt lesen
    $inhalt = fgets($datei);
    //Datei schließen, Ergebnis zurückgeben
    $ok = fclose($datei);
    if($ok) echo $inhalt;
    else echo "Fehler beim Lesen der Datei";
  }
?>
```

```
<?php
  // Textdatei ohne Nachfrage überschreiben
  //zuerst die Parameter auslesen
  $dateiname = $_GET['dateiname'];
  $inhalt = $_GET['inhalt'];
  //Datei öffnen
  $datei = fopen($dateiname,"w");
  //Inhalt schreiben
  fputs($datei,$inhalt);
  //Datei schließen und Ergebnis zurückgeben
  $ok = fclose($datei);
  if($ok) echo "ok";
  else echo "Fehler beim Schreiben der Datei";
?>
```

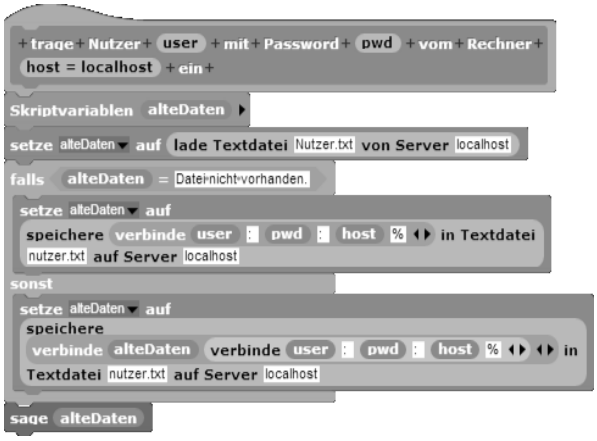
Diese Skripte legen wir im Verzeichnis htdocs von XAMPP ab und verlagern den Rest der Arbeit nach Snap!. Im einfachsten Fall werden diese Skripte einfach mit den entsprechenden Parametern aufgerufen:



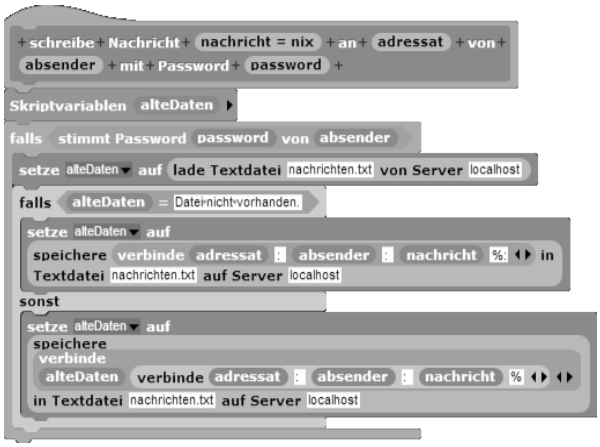
Mit diesen einfachen Blöcken können wir dann Daten dauerhaft auf unterschiedlichen Servern speichern und/oder wiederverarbeiten. Als Beispiele mögen verschlüsselte / entschlüsselte Texte oder die Kommunikation zwischen verschiedenen Snap!-Instanzen wie im nächsten Beispiel dienen.

4. Ein einfaches Client-Server-System mit Textdateien

Wir wollen eine weitere Möglichkeit schaffen, Botschaften zwischen verschiedenen Snap!-Instanzen im Netz auszutauschen. Beispiele dafür sind Chat- oder Mail-Systeme, die Modellierung von Netzwerken, Kommunikationsstrecken usw. Wir benutzen dafür zwei fest vorgegebene Dateien: eine namens nutzer.txt, in der Name, Password und Rechnername der verschiedenen Benutzer des Systems eingetragen werden – die Daten durch Doppelpunkte getrennt und die Datensätze durch Prozentzeichen. In einer zweiten Datei nachrichten.txt stehen die Informationen in der Reihenfolge adressat, absender, nachricht mit denselben Trennzeichen.

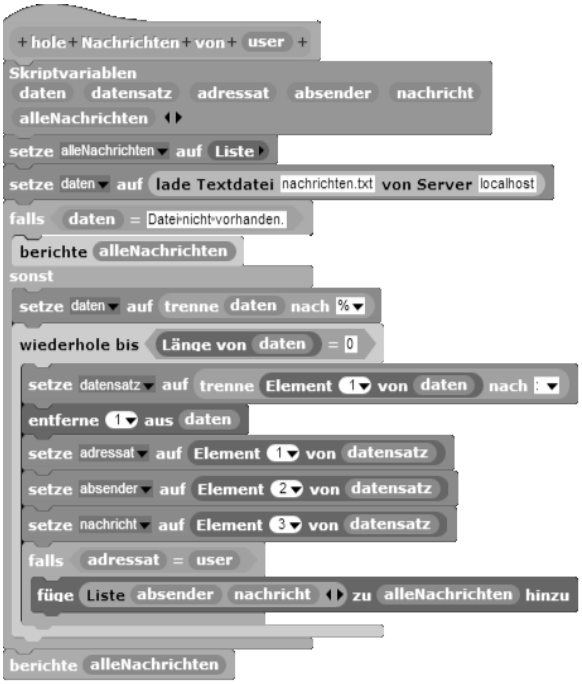


Das Eintragen der Nutzerdaten ist (ohne Kontrolle auf Doppelungen) einfach. Ist die Datei noch nicht vorhanden, wird ein Datensatz direkt als Textdatei gespeichert, sonst wird er an die vorhandenen Daten angehängt.



Etwas komplizierter ist das Schreiben einer Nachricht, denn hier muss natürlich das Password kontrolliert werden. Fällt das Ergebnis positiv aus, dann erfolgt das Speichern ähnlich wie bei den Nutzerdaten. Ein Password stimmt, wenn der angegebene Parameter mit dem Eintrag in den Nutzerdaten übereinstimmt.

Und wie holt man die Nutzerdaten? Man liest die Datei, spaltet sie (s.u.) nach Datensätzen (%) auf und diese wiederum nach den Daten (:). Dann vergleicht man den Nutzernamen. Im Erfolgsfall liefert der Reporter einen Datensatz, sonst eine leere Liste.



Zuletzt wollen wir noch die Nachrichten holen, die für einen Nutzer vorliegen – natürlich mit Passwordabfrage. Dafür erstellen wir eine Liste der Nachrichten, jeweils mit Absender, indem wir die Nachrichtendatei wie oben in Datensätze aufspalten (%) und diese wieder nach Daten (:). Stimmt der Adressat mit dem übergebenen Namen überein, wird ein Datensatz aus Absender und Nachricht an die Ergebnisliste alleNachrichten angehängt.

Die vollständigen Skripte und Dateien findet man unter (SnapExtensions, 2013) zum Download.

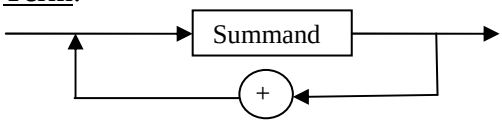
Auch hier ergeben sich zahlreiche Erweiterungen von selbst: Nachrichten und Benutzer müssen auch wieder gelöscht werden können. Was passiert eigentlich bei gleichzeitigen Zugriffen

von verschiedenen Nutzern auf die Daten? Welche Protokolle müssen für den Datenaustausch vereinbart werden? All dieses aber nicht als fertiges Ergebnis zum Kennenlernen, sondern als Schülerprodukt z. B. beim Umgang mit Transaktionen, auf das diese mit Recht stolz sein können.

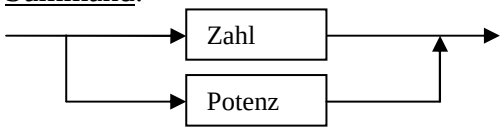
5. Etwas Computeralgebra: Blöcke, Rekursionen und funktionale Programmierung

Wir wollen die Möglichkeiten von Blöcken, insbesondere Prädikaten, anhand eines kleinen „*Computeralgebrasystems*“ zeigen, das weitgehend funktional arbeitet und das top-down mithilfe anfangs leerer Prädikatsblöcke entwickelt wird. Dazu müssen wir definieren, was wir unter Funktionstermen verstehen – der Einfachheit halber nur Summen ohne viel Vorzeichen:

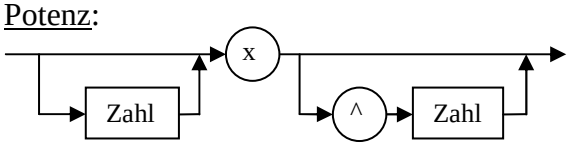
<u>Term</u>:



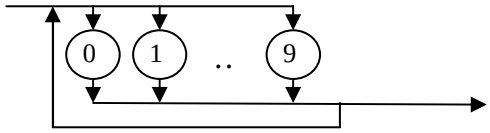
<u>Summand</u>:



<u>Potenz</u>:

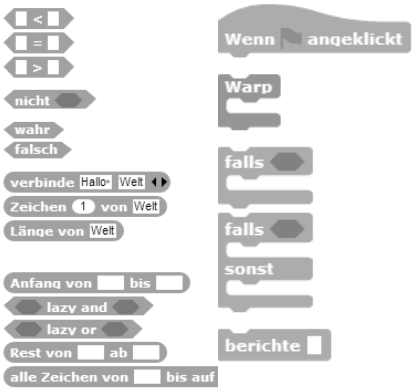


<u>Zahl</u>:

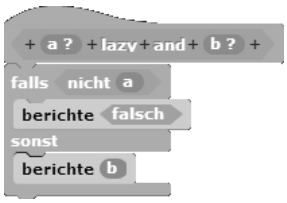


Funktionsterme sind also z. B.: 4x 2x+1 x^2+2 1+2x^4

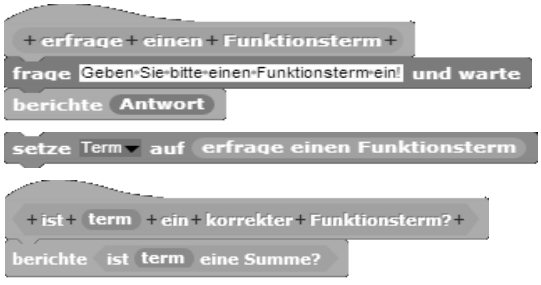
Um uns auf funktionale Skripte zu beschränken, blenden wir aus den entsprechenden Paletten die nicht erforderlichen Blöcke aus. Das ergibt z. B. in der Steuerung-Palette ein recht reduziertes Bild, und auch die Variablen-Palette enthält kaum noch etwas. Bei den Operatoren beschränken wir uns auf die Prädikate und die Zeichenkettenfunktionen, spendieren uns dafür aber die Möglichkeit, Teilzeichenketten bis/ab einem bestimmten Zeichen zu bestimmen sowie zwei neue Prädikate, die logische Operationen „lazy“ auswerten, d. h. den zweiten Ausdruck nur dann bestimmen, wenn sich das Ergebnis nicht schon aus dem ersten ergibt.



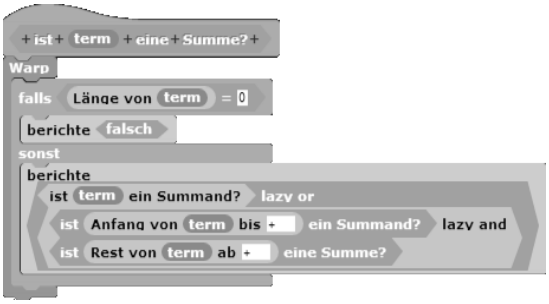
Zuerst einmal müssen wir Funktionsterme einlesen können. Dafür bittet er den Benutzer um eine entsprechende Eingabe mithilfe des Blocks frage <question> und warte aus der *Sensing*-Rubrik. Die Antwort des Benutzers kann anschließend mithilfe des Antwort-Blocks ermittelt werden. Im Skript weisen wir das Ergebnis einer Variablen namens Term zu.



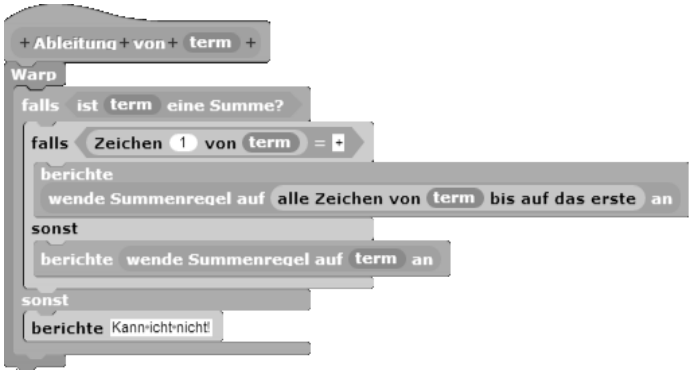
Als nächstes überprüfen wir, ob die Eingabe korrekt ist – wir schreiben also ein Prädikat. Dafür nutzen wir die Fähigkeiten von Snap! für ein Top-down-Vorgehen aus und definieren (hier nur ein) erstmal noch leeres Prädikat ist <term> eine Summe?. Dieses verwenden wir entsprechend den Syntaxdiagrammen in der Definition des Prädikats ist <term> ein korrekter Funktionsterm?



Mit diesen Hilfen definieren wir das erste Prädikat zur Prüfung korrekter Funktionsterme. Um sie mit Inhalten zu füllen, setzen wir das Verfahren für alle Elemente der Sprachdefinition korrekter Funktionsterme fort. Zuerst nehmen wir uns die Summe vor. Diese besteht entweder aus einem einzelnen Summanden oder einem Summanden, gefolgt vom Operator (+) und einer Summe. Das können wir direkt hinschreiben, wenn wir über ein vorerst noch leeres Prädikat ist <term> ein Summand? verfügen. Wie brauchen aber noch etwas mehr. Der eingegebene Term wird ja nicht mehr insgesamt untersucht, sondern wir müssen ihn ggf. in zwei Teile aufspalten: den Anfang von <term> bis <zeichen> und den Rest von <term> ab <zeichen>. Beide Funktionen arbeiten mit Zeichenketten und sind leicht funktional zu schreiben. Damit erhalten wir die Prädikate ist <term> ein Summand? und ist <term> eine Summe? – jeweils mit einer zusätzlichen Sicherheitsabfrage.

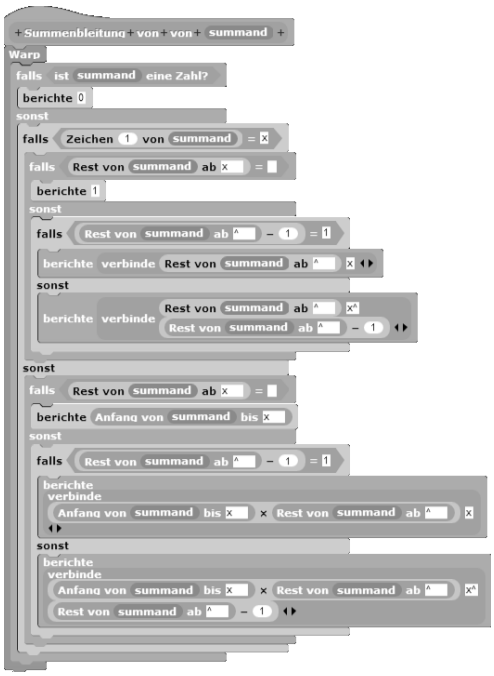


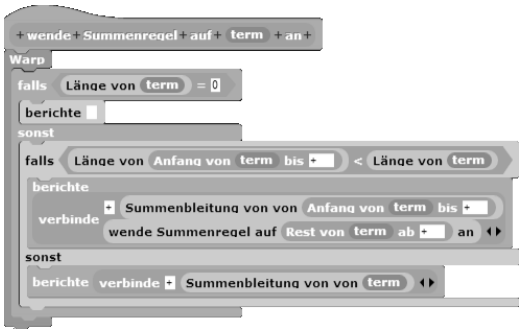
Wir nähern uns dem Ende. ist <term> eine Zahl? ist sehr einfach zu schreiben, wenn man ist <zeichen> eine Ziffer? kennt. Und wie überprüft man eine Potenz? Das steht ja auch im Syntaxdiagramm – wir müssen nur alle Möglichkeiten abschreiben. Wir können damit überprüfen (parsen), ob ein eingegebener Term der gewählten Syntax entspricht.



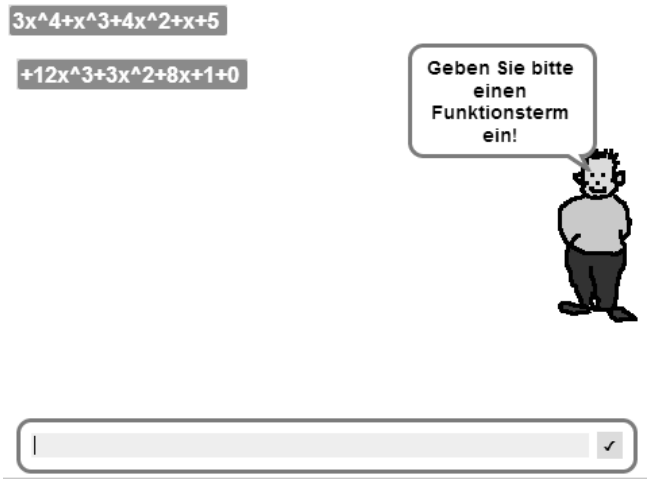
Wir wollen jetzt die erste Ableitung korrekter Funktionsterme bestimmen. Da nur die Summenregel infrage kommt, entfernen wir ggf. führende Vorzeichen und prüfen, ob die Regel überhaupt anwendbar ist.

Bei Anwendung müssen wir die Summanden bestimmen und diese ableiten. Anschließend werden die verschiedenen Möglichkeiten entsprechend den Regeln der Mathematik behandelt.





Beim der Ausgabe haben wir auf alle Vereinfachungen zugunsten der Kürze verzichtet, trotzdem kann man es halbwegs lesen. Paul, unser kleiner Mathematiker, kann Funktionsterme erfragen und erste Ableitungen anzeigen lassen. Mehr kann er noch nicht – aber es kann ja noch werden.



6. Fazit

Snap! in der vorliegenden Betaversion verfügt über fast alle Fähigkeiten, die ein leistungsfähiges Werkzeug für den schulischen Algorithmikunterricht braucht. Insbesondere die starken Visualisierungsmöglichkeiten erleichtern bottom-up-Entwicklung und eigenständige Schülerarbeit. Bleibt das Problem mit der textbasierten Programmierung. Nach anfänglichen erheblichen Vorbehalten gegen ein grafisches Werkzeug wie BYOB verlagert sich die Diskussion derzeit in die Richtung „Über Scratch und BYOB zu Java“ (oder einer anderen textuelle Sprache). Damit wird nahe gelegt, dass hier eine Hierarchie beschrieben wird: das textbasierte Werkzeug stellt den Gipfel dar. Doch nach welchen Kriterien ist die Hierarchie gegliedert?

Konzeptionell sind Sprachen wie Delphi oder Java Snap! klar unterlegen. Durch die Herkunft von Scheme verfügt Snap! über weitergehende OOP-Modelle und in Zukunft mit der Delegation bei Sprites auch über ein für schulische Zwecke weit überlegenes. Sowohl imperative wie funktionale Programmierung werden gleich gut unterstützt, bei den Datenstrukturen besticht Snap! durch extreme Anschaulichkeit und Beschränkung auf das Wesentliche. Und versuchen Sie mal, z. B. in Java das laufende System durch neue Kontrollstrukturen zu erweitern! Vor allem aber: das Verhältnis zwischen inhaltlicher Arbeit und Werkzeugschulung der Lernenden wird mindestens umgekehrt. Die Schülerinnen und Schüler haben viel mehr Zeit, eigene Projekte zu realisieren. Es bleibt also die Frage nach der Effizienz. Natürlich sind professionelle Systeme sehr viel schneller als ein interpretierendes grafisches Werkzeug. Aber spielt das eine Rolle? Wenn die Abituraufgaben der Bundesländer den Gipfel der Schulinformatik markieren, dann offensichtlich nicht. In Projekten kann das sicherlich mal anders sein, aber seit wann richtet sich der gesamte Unterricht nach irgendwelchen Spezialanwendungen?

Ich halte also den beschriebenen Weg über grafische Systeme hin zur textuellen Programmierung für einen Irrweg, für den keine inhaltlichen Argumente erkennbar sind. Im Gegenteil: er verhindert die fällige Entscheidung, ob im Unterricht mit textbasierten Sprachen zwar sehr effiziente, aber nur für wenige Lernende angemessen einsetzbare Werkzeuge benutzt werden, oder ob mit den neuen grafischen Systemen die Chance genutzt wird, das Schulfach Informatik auf eine viel breitere Basis zu stellen. Wollen wir also wenigen Lernenden (und Lehrenden) einen engen Bereich für ihre Hobbies erhalten, oder möglichst vielen Schülerinnen und Schülern Kenntnisse und Erfahrungen im algorithmischen Lernen und Handeln vermitteln. Darum geht es.

Wie sieht es mit Spezialbereichen wie der PHP-Programmierung in den vorgestellten Beispielen aus? Wenn keine „schulangemessenen“ Werkzeuge in einem bestimmten Kontext zur Verfügung stehen, dann muss man eben nehmen, was vorhanden ist, um die gewünschte Funktionalität überhaupt nutzen zu können. Unsere PHP-Skripte bestehen im Wesentlichen aus Bibliotheksaufrufen, über deren Bedeutung man sich irgendwo informieren muss. Die algorithmische Komplexität dagegen hält sich sehr in Grenzen. Wenn also z. B. im Projektunterricht ein Roboter angeschlossen oder eine andere Ressource genutzt werden soll, dann kann es durchaus richtig sein, die dafür gebauten Werkzeuge einzusetzen. In diesen Fällen befindet sich das textbasierte System aber hierarchisch „neben“ den grafischen. Es befindet sich durchaus auf gleicher Höhe, aber nicht darüber, und kann eingesetzt werden, etwa um zu zeigen, dass es noch mehr als grafische Oberflächen gibt. Als Standardwerkzeug im normalen Unterricht spielt es aber keine Rolle.

Literatur und Internetquellen

BYOB3 – Moenig, J., Harvey, B.: BYOB 3.0 – Build Your Own Blocks, 2010, http://byob.berkeley.edu/

Snap! – Moenig, J., Harvey, B.: BYOB 4.0 – Build Your Own Blocks, 2013, http://snap.berkeley.edu/snapsource/snap.html

ImB – Modrow, E. – Informatik mit BYOB, 2013, http://www.uni-goettingen.de/de/423680.html

XAMPP - http://www.apachefriends.org/de/xampp-windows.html

Snapextensions – Modrow, E., 2013 - http://www.uni-goettingen.de/de/??????.html

Alle Internetquellen wurden zuletzt am 22.9.2013 geprüft.

Prof. Dr. Eckart Modrow
Universität Göttingen
Institut für Informatik – Didaktik der Informatik
Goldschmidtstraße 7
37077 Göttingen
E-Mail: emodrow@informatik.uni-goettingen.de