UNIVERSITÄT DUISBURG ESSEN

*Offen* im Denken

Fakultät für Mathematik

Dr. Marcel Klinger

# Blockübersicht benötigter Blöcke in Scratch zu Algorithmische Mathematik

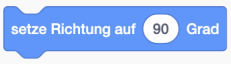
*Im Folgenden finden Sie eine Übersicht der wichtigsten Blöcke in Scratch. Diese sollten Sie gut kennen und u.a. in den jeweils beschriebenen Situationen verwenden können. Die Kenntnis weiter Blöcke kann hilfreich sein, ist im Hinblick auf die Klausur aber nicht zwingend erforderlich. Sollten andere als die hier aufgeführten Blöcke in der Klausur verwendet werden, werden diese dort explizit eingeführt. Bei der folgenden Auflistung wird jeweils davon ausgegangen, dass die Blöcke für eine Figur und nicht das Bühnenbild (falls überhaupt möglich) hinterlegt sind.*

## Bereich Bewegung


Der Block führt eine Bewegung der Figur in die aktuell gespeicherte Richtung und mit einer Länge der angegebenen Bühnenpixel durch.


Der Block ändert die Richtung der Figur im bzw. gegen den Uhrzeigersinn um einen Winkel der angegebenen Gradzahl

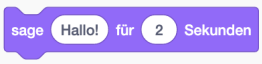

Der Block führt die Figur zu den angegebenen Bühnen-Koordinaten, wobei genau die Mitte der Figur zu diesem Punkt springt. Die Koordinaten messen in der gleichen Einheit wie die Schritte des obigen Gehe-Blocks. Im Hintergrund der Bühne ist ein unsichtbares Koordinatensystem hinterlegt. Dessen Ursprung befindet sich in der Mitte der Bühne. Die x-Achse reicht von -240 bis 240 Bühnenpixeln, die y-Achse von -180 bis 180 Bühnenpixeln. Mit Hilfe des Bühnenbild „Xy-grid“ kann dieses Koordinatensystem sichtbar gemacht werden.



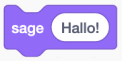

Setzt die aktuelle Richtung, auf die sich auch der Gehe-Block bezieht, auf die angegebene Gradzahl. Standardmäßig stehen Figuren $90°$. Das Setzen der Richtung auf $0°$ führt somit zu einer Drehung der Figur um $90°$ nach links.


Mit diesen Blöcken kann die aktuelle x- bzw. y-Koordinate der Figur abgefragt werden. Die beiden Blöcke stehen somit für eine natürliche Zahl und können nicht alleine, sondern nur eingebaut in einem anderen Block verwendet werden. Überall wo eine Zahl stehen kann, können die entsprechenden Blöcke eingefügt werden. Durch die Maße der Bühne sind die Werte der beiden Blöcke entsprechend beschränkt.

## Bereich Aussehen




Lässt für die Figur für die angegebene Anzahl von Sekunden eine Sprechblase mit dem entsprechenden Text erscheinen. Während der angegebenen Zeit wird der weitere Code pausiert. Der Block eignet sich, um Zwischenschritte sichtbar zu machen, wenn in den Text Variablen (bzw. Texte verbunden mit Variablen) eingefügt werden.



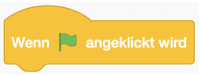

Dieser Block ist gleichbedeutend zum vorherigen Block, nur dass der darunterstehende Code unmittelbar ausgeführt wird und die Sprechblase solange erhalten bleibt, bis eine neuer Sage- oder Denke-Block aufgerufen wird.

Die Denke-Blöcke unterscheiden sich nicht von den Sage-Blöcken, außer dass eine Denkblase statt einer Sprechblase dargestellt wird.

## Bereich Klang

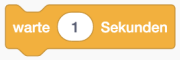
Hier gibt es mit Blick auf die Klausur verständlicherweise keine relevanten Blöcke.

## Bereich Ereignisse

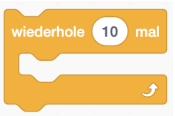



Dieser Block fungiert als Anfangsblock jedes Scratch-Programm. Was darunter steht, wird ausgeführt, unmittelbar nachdem die grüne Flagge geklickt wird.

## Bereich Steuerung

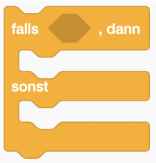



Der Block lässt das Programm für die angegebene Anzahl von Sekunden pausieren. Er ist vor allem zum Debuggen nützlich, um den Programmverlauf zu verlangsamen und mithilfe eingeblendeter Variablen nachzuvollziehen.




Der Block bildet die Zählschleife in Scratch. Leider ist es nicht ohne Tricks möglich innerhalb der Schleife zu überprüfen, in der wievielten Wiederholung man sich gerade befindet.


Hierbei handelt es sich um die einfache If-Anweisung, bei der der Code innerhalb der Klammer nur ausgeführt wird, falls die gesetzte Bedingung wahr ist. Ansonsten wird der gesamte umklammerte Block übersprungen.



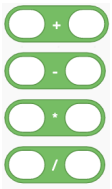

Hierbei handelt es sich um die einfache If-Else-Anweisung, bei der der Code innerhalb der ersten Klammer nur ausgeführt wird, falls die gesetzte Bedingung wahr ist. Ansonsten wird der zweite umklammerte Block ausgeführt. Erst nachdem die erste oder zweite Klammer ausgeführt wurde, wird das Programm fortgesetzt. Unabhängig davon, ob die gesetzte Bedingung wahr oder falsch ist, wird eine der beiden Klammern auf jeden Fall ausgeführt, sobald der gesamte Block im Programm erreicht wird.


Hierbei handelt es sich um die kopfgesteuerte Schleife in Scratch. Wenn die gesetzte Bedingung falsch ist, wird der Inhalt der Klammer ausgeführt. Ist die Bedingung danach weiterhin falsch, wird der Inhalt der Klammer erneut ausgeführt. Erst wenn die gesetzte Bedingung zum ersten Mal wahr ist, wird der gesamte Inhalt der Klammer übersprungen und das Programm am Ende des Blocks fortgesetzt. Bei einer „Solange“-Formulierung, die wir oft in Pseudocode verwendet haben, wäre es genau umgekehrt.

## Bereich Fühlen

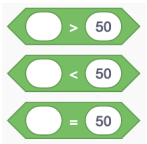

Die beiden Blöcke müssen immer in Kombination vorkommen. Der Frage-Block erzeugt dabei eine Sprechblase mit der entsprechenden Frage. Das Programm wird solange angehalten, bis der Benutzer eine daraufhin eingegebene Antwort in ein gleichzeitig erscheinendes Textfeld getippt hat. Im Anschluss daran wird das Programm unterhalb des Frage-Blocks fortgesetzt und die eingegebene Antwort in der Spezial-Variable Antwort gespeichert. Nutzt man mehrere Frage-Blöcke, ist es unbedingt notwendig die Antworten zwischenzeitlich in andere Variablen zu kopieren, damit diese bei der nächsten Frage-Anweisung nicht überschrieben werden und somit verloren gehen.

## Bereich Operatoren




Operatoren mit halbkreisförmigen Enden stehen immer für Zahlen oder Texte. In diesem Fall wird das Ergebnis der entsprechenden Rechenoperationen zurückgegeben. Wie alle Operatoren können auch diese beliebig tief ineinander verschachtelt werden. Hierdurch ergibt sich aus mathematischer Sicht eine implizite Klammerung, die auch Regelungen wie Punkt-vor-Strichrechnung außer Kraft setzt.


Der Operator erzeugt eine ganze Zufallszahl zwischen den angegebenen Grenzen, wobei diese einschließlich zu verstehen sind. Kommt der Block mehrmals in einem Programm vor, können jedes Mal unterschiedliche Zahlen gezogen werden. Benötigt man dieselbe gezogene Zufallszahl im Programmverlauf noch einmal, ist es sinnvoll, diese in einer Variablen abzuspeichern.




Operatoren mit dreieckigen Enden stehen immer für Wahrheitswerte und somit für wahr oder falsch. Je nachdem welche mathematischen Werte in die runden Einfassungen eingesetzt werden, ergibt sich der entsprechende Wahrheitswert. Man benötigt sie vor allem als Bedingung innerhalb der oben dargestellten Kontrollstrukturen.


Auch diese Operatoren erzeugen einen Wahrheitswert. Diese gehen aber nicht von Zahlen, sondern ihrerseits von Wahrheitswerten aus. Nach den Gesetzmäßigkeiten der booleschen Logik (s. Exkurs in Video 210) werden entsprechend die Werte wahr und falsch erzeugt. Der Und-Operator ist etwa genau dann wahr, wenn beide eingesetzten Wahrheitswerte gleichzeitig wahr sind.


Der Operator dient dem Verbinden von Texten und tritt häufig eingesetzt in einen Sage- oder Denke-Block auf. Er ist notwendig, um Werte von Variablen mit Textpassagen zu verbinden und den Benutzer so z.B. über das Ergebnis einer Rechnung zu informieren. Es ist häufig notwendig mehrere Verbinde-Operatoren ineinander zu verschachteln. Eine Klammerung spielt hierbei keine Rolle.


Der Operator gibt den Wert der entsprechenden Modulo-Operation zurück. Hierbei handelt es sich um den Rest, den die erste Zahl beim Teilen durch die zweite Zahl lässt.


Der Operator liefert wichtige mathematische Spezialfunktionen und gibt das entsprechende Ergebnis bezogen auf die eingesetzte Zahl zurück. Hierzu gehört das Auf- und Abrunden, die Quadratwurzel, trigonometrische Funktionen, die $e$-Funktion, der natürliche und dekadische Logarithmus und die Zehnerpotenz $10^x$.

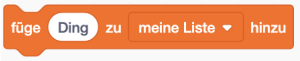
## Bereich Variablen


Einen solcher Block wie der dargestellte existiert für jede Variable, die man erstellt hat. Er kann nicht alleine stehen, sondern existiert nur als eingesetzter Block innerhalb eines anderen Blocks. Hier steht er für den momentanen Wert, der in ihm gespeichert ist.

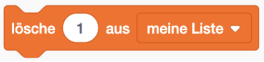

Der Setze-Block ist der grundlegende Block, der benötigt wird, um Variablen im Laufe des Programmablaufs Werte zuzuweisen. Hierbei kann es sich entweder um Texte, Zahlen oder Wahrheitswerte handeln. Trotz der halbkreisförmigen Enden innerhalb seines Arguments kann der Block also auch Operatoren mit dreieckigen Enden aufnehmen.


Der Block addiert die angegebene Zahl zur ausgewählten Variable hinzu. Ist die Zahl negativ, wird entsprechend eine Subtraktion ausgeführt. Der Block ist äquivalent zur folgenden Komposition:


Neben diesen Blöcken, die sich auf einfache Variablen beziehen, gibt es auch die Möglichkeit Listen zu erstellen. Sobald die erste Liste über den Button „Neue Liste“erstellt wurde, stehen die folgenden weiteren Blöcke zur Verfügung:

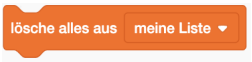



Der Block fügt „Ding“ zur ausgewählten Liste hinzu. Hierbei kann „Ding“ für einen Texte, Zahlen oder Wahrheitswerte stehen. Das entsprechende Objekt wird an das Ende der bestehenden Liste angefügt. Ist die Liste noch leer, bildet es das erste Objekt innerhalb der Liste.

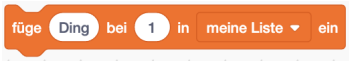



Der Block löscht das Objekt aus der ausgewählten Liste, das an der angegebenen Position steht.

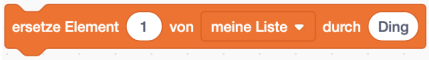
Hierbei wird anders als bei Arrays üblich bei 1 und nicht bei 0 angefangen zu zählen.




Der Block leert eine Liste vollständig. Dieser Block sollte immer am Anfang eines Programms stehen, sobald mit Listen gearbeitet wird, da ansonsten die Listeninhalte eines vorherigen Durchlaufs bestehen bleiben.




Der Block fügt ein Objekt an der genannten Position in die Liste ein. Alle anderen Objekte verschieben sich entsprechend. Auch hier wird bei 1 angefangen zu zählen.




Der Block ersetzt das Element an der genannten Position in der Liste durch das angegebene „Ding“. Auch hier kann Ding für Zahlen, Texte oder Wahrheitswerte stehen. Befindet sich an der genannten Position noch kein Objekt, bleibt der Block tatenlos.


Der Block gibt den Wert des an angegebener Position stehenden Objekts in der Liste zurück. Er kann nicht alleine stehen, sondern kommt nur eingesetzt innerhalb eines anderen Blocks vor.

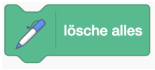

Der Block gibt die Anzahl der Element innerhalb einer Liste als Zahl wieder.

## Bereich Meine Blöcke

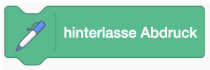
Hier lassen sich eigene Blöcke erstellen, welche wiederum beliebige andere Blöcke aufnehmen können. Außerdem kann eine vom Programmierer festgelegte Anzahl an Argumenten übergeben werden, auf welche innerhalb der Block-Ausführung zugegriffen werden kann. Hierzu zieht man in der Block-Definition das entsprechende Argument wie eine Variable in den Code. Die übergebenen Argumente können innerhalb des Blocks nicht verändert werden. Die Funktion eigene Blöcke zu erstellen, ist vor allem nützlich, wenn eine spezielle Code-Sequenz mehrmals oder an verschiedenen Stellen im Stammprogramm ausgeführt werden soll. Er hilft außerdem dabei, die Übersicht zu behalten.

## Bereich Malstift

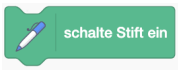
Der Malstift-Bereich muss zuvor als Erweiterung aktiviert werden. Dies geschieht über den Plus-Button ganz unten links.



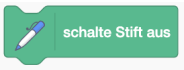

Dieser Block entfernt alle gesetzten Linien, die eventuell bei einer vorherigen Programmausführung gemalt wurden. Er sollte unbedingt zu Beginn des Programms gesetzt werden, um entsprechend aufzuräumen.




Dieser Block setzt einen Stempel in Form der Figur, für die er hinterlegt ist. Wird diese danach wegbewegt, bleibt ein entsprechender Abdruck vorhanden. Gerade mit Figuren, die nur aus einem Punkt bestehen, lassen sich hiermit nahezu pixelweise Gemälde zeichnen.




Dieser Block schaltet den Stift ein. Die entsprechende Figur hinterlässt nun bei jeder Bewegung eine entsprechende Linie. Wird die Figur direkt zu anderen Koordinaten gesetzt, entsteht eine gerade Linie zwischen ihrer vorherigen und ihrer neuen Position.




Dieser Block deaktiviert einen Stift wieder. Bewegt sich die entsprechende Figur danach, hinterlässt sie keine Linie mehr. Wird dieser Block ausgeführt, obwohl der Stift zuvor gar nicht aktiviert worden ist, bleibt er tatenlos.

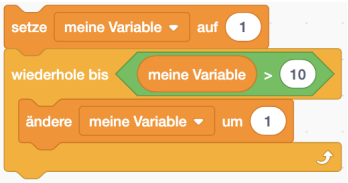
## Häufige Kompositionen

Die folgenden Kompositionen verschiedener Blöcke werden häufig benötigt.


Diese Komposition fragt einen Wert beim Benutzer ab und speichert diesen Direkt in eine gewählte Variable. So bleibt der abgefragt Wert selbst dann erhalten, wenn eine weitere ähnliche Abfrage vorgenommen wird. Außerdem kann mit ihm gerechnet werden.


Die Komposition gibt den Wert einer Variablen gemeinsam mit einem erläuternden Text aus. Es können auch mehrere Verbinde-Blöcke geschachtelt werden, um mit einer Mitteilung mehrere Variablen oder Text nach der Variablen auszugeben.


Mit dieser Komposition ist es möglich, eine Unzulänglichkeit vom Scratch per Trick zu umgehen: Fügt man den eigentlichen Inhalt der Zählschleife vor dem Ändere-Block ein, lässt sich über die Variable zwischenzeitlich die Nummer des jeweiligen Durchlaufs der Schleife abrufen und weiterverarbeiten.



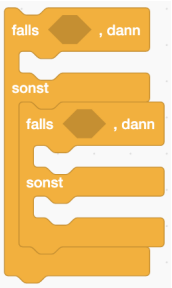

Diese Komposition ist äquivalent zum Vorstehenden, realisiert dies jedoch technisch durch eine kopfgesteuerte Schleife und eine entsprechende Bedingung. Statt dieser Bedingung könnte im konkreten Fall auch auf $= 11$ geprüft werden.


Diese Kompositionen gibt Dezimalzahlen zwischen 0 und 1 aus. Ersetzt man die 10000 an beiden Stellen durch eine noch größere Zahl, erhöht man entsprechend die maximale Anzahl an Nachkommastellen der zufälligen Dezimalzahlen. Wichtig ist hier, dass die Blöcke wie dargestellt verschachtelt sind.


Die Komposition prüft, ob eine Variable gerade ist. Sie kann z.B. als Bedingung in einer Schleife oder If-Anweisung verwendet werden.


Die Komposition prüft, ob eine Variable ungerade ist. Sie kann z.B. als Bedingung in einer Schleife oder If-Anweisung verwendet werden.


Durch eine derartige Komposition ist es möglich eine kopfgesteuerte Schleife statt mit einer „Wiederhole bis“ mit einer „Solange“-Bedingung auszustatten. Dies ist in einigen Situationen näher am Pseudocode und somit ggfs. einfacher zu implementieren.




Durch diese Komposition lässt sich eine If-Elseif-Else-Anweisung realisieren. Hierbei sind die freien Felder zwischen den Klammern genau in der entsprechenden Reihenfolge zu verstehen. Durch weiteres Schachteln lassen sich beliebig viele Fälle unterscheiden, bevor der Else-Teil ausgeführt wird.