# Bemerkungen für Lehrpersonen

Die *kursiv geschriebenen Ceebot-Befehle* stehen nur in der jeweiligen Übung zur Verfügung.

### Allgemeine Bemerkungen

- Die meisten Übungen sind so gestaltet, dass sie sich mit relativ kurzen Programmen lösen lassen. Die Erstellung längerer Programme erscheint nicht sinnvoll, solange die Möglichkeit fehlt, diese durch Funktionen bzw. Methoden in kleinere Einheiten zu zerlegen. Wenn von Seiten der Lehrperson an einer bestimmten Stelle des Programmierkurses eine Programmieraufgabe über einen Zeitraum von mehreren Unterrichtseinheiten gewünscht wird, so eignet sich dafür am besten ein Wettbewerb. Auf Arbeitsblatt 7.3 wird ein relativ einfacher Sammelwettbewerb besprochen, schwierigere Beispiele bieten das Roboterrennen und der Roboterfußball der Originalversion von Ceebot 4.
- Erfahrungsgemäß bewirkt die graphisch anspruchsvolle Umsetzung der Ceebot-Umgebung und die Vielfältigkeit der Roboter-Aktionen in den ersten Monaten bei den Schülern einen recht hohen Motivationsgrad, der natürlich auch durch das spielerische Element gefördert wird. Deshalb ist auch in der Anfangsphase des Unterrichts unbedingt darauf zu achten, dass die Arbeitsblätter sorgfältig ausgefüllt werden und dass das auf den Arbeitsblättern vermittelte Wissen auch tatsächlich beherrscht wird. Natürlich ist es über einen gewissen Zeitraum möglich, die Übungsaufgaben zu lösen, ohne die Erklärungen und Verständnisfragen auf den Arbeitsblättern durchzuarbeiten. Allerdings wird dann das an Hand der Übungsaufgaben erarbeitete intuitive Verständnis nicht systematisiert und damit auch nicht gefestigt und steht daher längerfristig kaum zur Verfügung.
- Auf den Arbeitsblättern wird Wert auf einen sauberen und übersichtlichen Programmierstil gelegt, der von den Schülerinnen und Schülern von Anfang an unbedingt eingehalten werden sollte. Einen bereits zu Beginn gefestigten schlechten Programmierstil kann man sich nur schwer wieder abgewöhnen, was längerfristig dazu führt, dass komplexe Programmieraufgaben nicht mehr beherrscht werden.
- Auf den Arbeitsblättern werden Struktogramme als didaktisches Hilfsmittel verwendet, um den Schülerinnen und Schülern einen Lösungsansatz zu bieten. Darüber hinaus wird mit Hilfe von Struktogrammen die Idee der „Top-Down“-Programmentwicklung vermittelt, die es in einem ersten Schritt ermöglicht, gewisse Programmschritte verbal grob zu formulieren. Dadurch erhält man einen Überblick über den Gesamtalgorithmus, ohne sich dabei in Detailproblemen zu verlieren. Die schrittweise Verfeinerung und Konkretisierung dieser verbalen Formulierungen führt schlussendlich zum fertigen Programmcode. Wenn bei bestimmten Programmierübungen eine derartige Arbeitsweise der Schülerinnen und Schüler gewünscht wird, empfiehlt sich die Verwendung des Struktogramm-Editors „StruktEd“, dessen Verwendung in Anhang B beschrieben wird.
- Insbesondere bei Programmieraufgaben, die von den Schülerinnen und Schülern weitgehend selbständig erstellt werden, ist es didaktisch sinnvoll, eine Programmdokumentation erstellen zu lassen. Im Rahmen einer derartigen Dokumentation sollte der grundsätzliche Lösungsansatz in Form eines Struktogramms vorgestellt werden. Struktogramme eignen sich auch, wenn im Unterricht erarbeitete Lösungen (beispielsweise in Partner- oder Gruppenarbeiten) der Klasse in Form von Kurzreferaten vorgestellt werden sollen.
- Die Arbeitsblätter sind grundsätzlich so aufgebaut, dass alle Elemente einer Programmiersprache an Hand von Ceebot vermittelt werden. Allerdings zeigt die Erfahrung, dass vor allem leistungsstärkere Schülerinnen und Schüler nach einiger Zeit den Wunsch äußern, mit einer professionellen Entwicklungsumgebung zu arbeiten. Aus diesem Grund sind bereits einige der früheren Programmierübungen so gestaltet, dass sie sich problemlos auch als Konsolenanwendung etwa in Java oder C++ umsetzen lassen: „Arbeitsblatt 5.4 / Oh Tannenbaum“, „Arbeitsblatt 5.5 / Rechenknecht“, „Arbeitsblatt 8.5 / Reaktionstest 2“.

### Arbeitsblatt 1.1: Erste Programme 1
- Befehle: `grab, move, drop, turn`
- Grundlegende Bedienung von Ceebot
- Begriff „Programm“

### Arbeitsblatt 1.2: Erste Programme 2
- Parameter eines Befehls
- Default-Wert eines Parameters
- Kommentare
- Debugging: Mitverfolgen der Arbeitsschritte im Programmcode

### Arbeitsblatt 2.1: Ein Quadrat zeichnen
- Befehle: `pendown, penup, repeat`
- Welche Befehle gehören in die Schleife, welche gehören außerhalb ?

**Bemerkung:** Die `repeat`-Schleife gibt es in Java, C und seinen Dialekten nicht. Die Einführung dieser Schleife in Ceebot stellt einen didaktischen Zwischenschritt dar, der Übungen mit Schleifen ermöglicht, bevor Variablen eingeführt werden. Übungsprogramme mit Schleifen sollten im Rahmen eines Programmierkurses so früh wie möglich behandelt werden, da Schleifen erfahrungsgemäß eines der Hauptprobleme von Programmieranfängern darstellen. In den nachfolgenden Arbeitsblättern wird das grundlegende Konzept einer Schleife zuerst ausführlich an Hand der `repeat`-Schleife verdeutlicht und eingeübt, bevor die `for`-Schleife besprochen wird, durch die man ja ganz problemlos die `repeat`-Schleife ersetzen kann.

### Arbeitsblatt 2.2: Erzeugen von fünf Titanwürfeln
- Eine weitere Übung mit der `repeat`-Schleife
- Berechnung einer Fahrtstrecke mit Lehrsatz von Pythagoras

### Arbeitsblatt 2.3: Kompliziertere Bewegungen mit Schleifen
- Mehrere Möglichkeiten zur Festlegung eines Schleifendurchlaufs
- „Vorspann“ und „Nachspann“ einer Schleife
- Bei den Lösungsvorschlägen ergibt sich am Ende des Programmdurchlaufs eine eigentlich „unnötige“ Bewegung des Roboters, nachdem bereits der letzte Waypoint überfahren wurde. Mit den Schülern sollte diskutiert werden, wie man diese unnötige Bewegung verhindern kann.

### Arbeitsblatt 3.1: Ein strichliertes Quadrat zeichnen
- Zwei ineinander verschachtelte Schleifen

### Arbeitsblatt 3.2: Schatzsuche
- Befehle: `sniff`
- Komplizierte Verschachtelung von Schleifen

### Arbeitsblatt 4.1: Bunte Linien
- Befehle: `int, float, string, message, pencolor`
- Beispiel mit Integer-Variablen
- Variablentypen, Variablendeklaration

### Arbeitsblatt 4.2: Ein buntes Quadrat zeichnen
- Befehle: `sqrt, wait`
- Variablen in einer verschachtelten Schleife

### Arbeitsblatt 4.3: Apportieren
- Befehle: *`distance_to`*
- Beispiel mit Gleitkomma-Variablen
- Verwendung einer Zählvariablen

### Arbeitsblatt 4.4: Aufräumen 1

- Befehle: *direction_to*, `aim`, `fire`
- Ansonsten ähnlich wie Arbeitsblatt 4.3

### Arbeitsblatt 4.5: Lagerplatz 1

- Etwas komplizierter wie Arbeitsblätter 4.3 und 4.4, weil es die Ideen beider Arbeitsblätter zusammenfasst. Der Roboter muss nach dem Ablegen jeder Batterie immer wieder die Ausgangsposition und die Ausgangsorientierung einnehmen, wozu er sich merken muss, wie weit er sich bewegt und gedreht hat. Außerdem wird eine Zählvariable für die Batterien benötigt, aus der die Ablage-Position berechnet wird.

### Arbeitsblatt 4.6: Länge einer Spirale

Auf diesem Arbeitsblatt wird eine häufig benötigten Programmstruktur verwendet: Eine Variable wird vor einer Schleife initialisiert und in der Schleife aufgrund ihres vorhergehenden Wertes neu berechnet. Diese Struktur sollte im Unterricht ausführlich besprochen werden.

Neben den relativ einfachen mathematischen Beispielen, die als weiterführende Übungen angegeben sind, bieten sich hier auch kompliziertere mathematische Iterationen an (da noch keine bedingten Schleifen zur Verfügung stehen, ist die Anzahl der Iterationen fix vorzugeben):

- Berechnung der Quadratwurzel von a mit Hilfe der Heron'schen Iteration $x_{neu} = (x_{alt} + a / x_{alt}) / 2$ mit dem Startwert 1 für $x_{alt}$;
- Berechnung der ersten 20 Fibionacci-Zahlen, bei der jede Zahl die Summe ihrer beiden Vorgänger ist: 1, 1, 2, 3, 5, 8, 13, ...
- Wenn man die `if`-Verzweigung an dieser Stelle und damit etwas früher als vorgesehen einführt (was wegen der einfachen Syntax didaktisch kein Problem darstellt), so lässt sich auch die Nullstellensuche mit Hilfe der Regula falsi behandeln.

Da die ausführliche Behandlung derartiger mathematischer Programmieraufgaben gerade mittlere und schwächere Schüler oft frustriert, werden diese Fragestellungen auf den Arbeitsblättern nicht erwähnt. Bei ausreichendem Zeitrahmen stellen sie jedoch für leistungsstärkere Schüler wichtige Übungsbeispiele dar.

### Arbeitsblatt 5.1: Jedes Mal ein bisschen weiter

- Befehle: `for`
- Natürlich kann man alle bisher behandelten Übungen zur `repeat`-Schleife auch als Übungsbeispiele zur `for`-Schleife verwenden.
- Da die `repeat`-Schleife nur in Ceebot existiert, erscheint es sinnvoll, den Schülern ab der Besprechung dieses Arbeitsblattes die Verwendung der `for`-Schleife anstelle der `repeat`-Schleife vorzuschreiben.
- Zur Frage, an welcher Stelle die Laufvariable deklariert werden soll (drittes Beispiel einer `for`-Schleife) , siehe Arbeitsblatt 6.2.

### Arbeitsblatt 5.2: Gültigkeitsbereich von Variablen

- Beim Programm `Test2` werden wohl manche Schüler Schwierigkeiten mit der Erklärung der erzeugten Ausgabe haben. Nachdem einige Zeit gerätselt worden ist, sollte folgendes besprochen werden:
- Wenn man versucht, eine bereits deklarierte Variable in einem „untergeordneten" Anweisungsblock noch einmal zu deklarieren, so erhält man eine zweite Variable, die zwar denselben Namen hat, aber auf einem anderen Speicherplatz abgelegt wird. Deshalb kommt es auch zu keinem Überschreiben. Dies hat allerdings auch zur Folge, dass innerhalb des „untergeordneten" Anweisungsblocks nicht auf den Speicherplatz zugegriffen werden kann, der bei der Variablendeklaration im übergeordneten Anweisungsblock für diesen Variablennamen reserviert wurde.
- Die Verwendung gleicher Variablennamen in zwei verschachtelten Anweisungsblöcken kann allerdings leicht zu Programmfehlern führen und sollte daher vermieden werden.

- Beim Programm `Test4` sollte besprochen werden, dass sich der Wert von 8 für i **nach** dem letzten Schleifendurchlauf folgendermaßen ergibt: Am Ende eines Schleifendurchlaufs wird die Anweisung `i=i+2` ausgeführt und unmittelbar danach, nämlich zu Beginn des nächsten Schleifendurchlaufs, erfolgt die Prüfung auf `i<=6`. Ist diese Bedingung nicht erfüllt, so wird die Schleife zwar abgebrochen, aber der erhöhte Wert der Laufvariable bleibt bestehen.
- Obwohl bei den bisherigen Beispielen nicht darauf geachtet wurde, sollte für die im weiteren zu erstellenden Programme darauf bestanden werden, dass jede Variablendeklaration stets nur für den kleinstmöglichen Anweisungsblock vorgenommen wird. Ein konsequentes Durchhalten dieses Prinzips führt später zu sauberen Unterprogrammen (Funktionen) und zu einem besseren Verständnis der Kapselung beim objektorientierten Programmieren.

### Arbeitsblatt 5.3: Zeitnehmung

- Befehle: `textout, gotoxy, abstime`

### Arbeitsblatt 5.4: Oh Tannenbaum

- Das Ceebot-Anzeigefenster dient dazu, den Übergang von Ceebot zu Java oder C++ vorzubereiten. Es bietet sich an, zuerst die Übungen dieses Arbeitsblatts in Ceebot zu programmieren und dann dieses Programm auf die jeweilige Java- oder C++-Entwicklungsumgebung zu übertragen. Auf diese Weise lernen die SchülerInnen den Umgang mit einer komplexeren Entwicklungsumgebung an Hand eines Programms, das sie bereits kennen.

### Arbeitsblatt 5.5: Rechenknecht

- Befehle: `printf`
- Für den Umstieg von Ceebot auf eine andere Entwicklungsumgebung gilt dasselbe wie beim Arbeitsblatt 5.4.

### Arbeitsblatt 6.1: Koordinatensystem

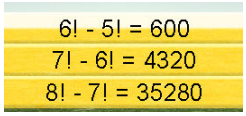
- Befehle: `goto`
- Übung zum Verständnis eines dreidimensionalen Koordinatensystems mit minimalem Programmieraufwand
- Da der Befehl `goto` sehr „intelligent“ ist, erledigt er einige Aufgaben selbständig, die die Schüler eigentlich in den nachfolgenden Übungen programmieren sollen – so stellt der Roboter bei der Verwendung dieses Befehls selbständig den Objekten aus, die ihm im Weg stehen. Deshalb sollte die weitere Verwendung von `goto` nach dieser Übung entweder gänzlich unterlassen werden oder nur in einer ersten Programmversion erfolgen. Eine Begründung dafür kann man den Schülern dadurch liefern, dass die Wegsuche von `goto` nicht wirklich perfekt funktioniert.

### Arbeitsblatt 6.2: Kilometerzähler

- Befehle: *`position_of`*, `sqrt`, `**` (Potenzieren), `message`, Aneinanderhängen von Zeichenketten (Strings), `distance`
- Im allgemeinen empfiehlt sich das Besprechen des Kapitels „Objektorientierte Programmierung (OOP) mit Hilfe von Klassen“ im Referenz-Teil des Schulbuchs (Seite 104) unmittelbar VOR der Bearbeitung dieses Arbeitsblatts.
- Einleitend eine kurze Zusammenfassung der Begriffe „Klasse“, „Attribut“ und „Instanz“ (Methoden werden erst im Kapitel „Objektorientiertes Programmieren“ behandelt), so weit dies auf einem Arbeitsblatt sinnvoll erscheint. Im Referenz-Teil des Schulbuchs werden diese Begriffe ausführlicher erläutert.
- Da gerade diese Begriffe etwas abstrakter sind als der bisherige Stoff, ist deren ausführliche Besprechung durch die Lehrperson sehr zu empfehlen.
- Zum Verständnis dieser Begriffe ist zu empfehlen, dass die Lehrperson mit den Schülern weitere Beispiele von sinnvollen Klassendefinitionen durchspielt, zum Beispiel hat eine Klasse `stadt` die Attribute `float flaeche` und `int einwohner`. Eine Instanz dieser Klasse, die die Stadt Salzburg beschreiben soll, wird dann mit `stadt salzburg` deklariert.

`salzburg.flaeche` und `salzburg.einwohner` bezeichnen dann die entsprechenden Daten dieser Stadt.

- In den unmittelbar folgenden Arbeitsblättern werden die Begriffe „Klasse“, „Attribut“ und „Instanz“ in konkreten Anwendungen nochmals erläutert. Wenn einigen Schülern zum jetzigen Zeitpunkt diese Begriffe auch nach längerer Besprechung noch unklar sind, so erlaubt der Aufbau der folgenden Arbeitsblätter einen „learning by doing“-Zugang, der manchen Schülern mehr entgegenkommt als die theoretische Besprechung der Begriffe.
- Bereits an dieser Stelle kann betont werden, dass der Vorteil des Arbeitens mit Klassen vor allem dann sichtbar werden, wenn wir in einem Programm mit mehrere Instanzen einer Klasse arbeiten. Wie wir unsere eigenen Klassen in Ceebot definieren, wird erst im Kapitel „Objektorientiertes Programmieren“ behandelt.
- Die doppelte Bedeutung des „+“-Zeichens als Addition von Zahlen und zum Aneinanderhängen von Zeichenketten sollte an Hand des „Weiterführenden Beispiels 1“ ausführlicher besprochen werden.
- Als Herausforderung für gute Schüler eignet sich ein Programm zur schrittweisen Berechnung der Differenz zweier aufeinander folgender Fakultäten, also von n!-(n-1)!




### Arbeitsblatt 7.1: Aufräumen 2

- Befehle: `if, radar, this`
- Der erste Teil des Arbeitsblattes - Übung “Aufräumen 2a” - ist recht einfach und führt die Klasse `object`, die `if`-Verzweigung und den `radar`-Befehl ein. Mögliche Verständnisprobleme bei dieser Übung sind wahrscheinlich auf ein mangelhaftes Verständnis der Begriffe „Klasse“, „Attribut“ und „Instanz“ zurückzuführen, auf das möglichst individuell eingegangen werden sollte.
- Der zweite Teil des Arbeitsblattes - Übung “Aufräumen 2b” – ist deutlich komplizierter, weil der Roboter zwei Aufgaben zugleich zu erledigen hat, die in keinem vorherigen Arbeitsblatt genau so vorgekommen sind (Erkennen des „Ungeziefers“ und Nachladen seiner Batterie). Die beiden Aufgaben werden zuerst in zwei getrennten Programmen jede für sich gelöst und sind dann mit Hilfe eines vorgegebenen Struktogramms zu einem Programm „zusammenzufügen“.
- An Hand dieses Struktogramms sollte besprochen werden, wie sich diese Methode des „graphischen Programmentwurfs“ sinnvoll einsetzen lässt: Ein Struktogramm stellt nur dann ein sinnvolles Element der Programmentwicklung und –dokumentation dar, wenn es einen Überblick über den gesamten Programmcode bietet. Das heißt aber, dass es nicht zu detailliert sein soll. Wenn im Extremfall jeder Befehl eins zu eins im Struktogramm vorkommt, dann ist das Struktogramm auch nicht übersichtlicher als der Code selbst. Wichtig ist also, dass zusammengehörige Befehle im Struktogramm zu einem Block zusammengefasst werden und dieser mit einer prägnanten verbalen Beschreibung versehen wird.
- Im Gegensatz zu früheren Aufgaben wird bei Übung “Aufräumen 2b” bewusst auf die Vorgabe wesentlicher Teile des Programmcodes verzichtet, um einen Schritt in Richtung möglichst selbständiger Codeerstellung zu setzen. Dies wird auch bei den folgenden Arbeitsblättern verstärkt der Fall sein.
- Bei komplexeren Programmieraufgaben ist ein Zerlegen in Teilprobleme und deren getrennte Lösung gerade für Programmieranfänger eine sehr empfehlenswerte Herangehensweise. Dadurch wird die für jede Einzellösung geeignete Programmiertechnik separat geübt, und mögliche Probleme treten klarer hervor als bei der sofortigen Behandlung des Gesamtproblems. Erfahrungsgemäß führt die gleichzeitige Behandlung zweier (oder mehrerer) neuer Programmiertechniken in denselben Beispiel gerade bei schwächeren Schülern zur Verwirrung.
- Beim momentanen Kenntnisstand der Schüler ist es zu empfehlen, die Gesamtlösung von Grund auf neu programmieren zu lassen. Ein „Zusammenkopieren“ der beiden Teillösungen führt erfahrungsgemäß zu unnötig langer Fehlersuche. Sehr wohl sollte man als Lehrperson aber betonen, dass diese Wiederverwendung einmal erstellter Problemlösungen in anderen

Programmen unbedingt erstrebenswert ist. Allerdings wird eine saubere modulare Programmierweise erst durch Funktionen (siehe Kapitel 9) bzw. Methoden (siehe Kapitel 11) möglich.

- Als weitere Übungsbeispiele und zur gleichzeitigen Überprüfung des Kenntnisstandes der Inhalte früherer Lektionen kann man beispielsweise die folgenden Aufgaben mit einer `for`-Schleife lösen lassen:
  - Der Roboter soll an den Batterien in Übung „Aufräumen 2a“ entlang fahren und dabei den Mittelwert aller Batteriefüllstände bestimmen (einfacher, weil der Roboter immer nur die nächststehende Batterie orten muss).
  - Der Roboter soll den Mittelwert der Batteriefüllstände in Übung „Aufräumen 2a“ bestimmen, ohne sich dabei selbst zu bewegen (schwieriger, weil der Roboter auch weiter entfernt stehende Batterien orten muss).

### Arbeitsblatt 7.2: Lagerplatz 2

- Befehle: `switch, direction`
- Da es bei diesem Beispiel um eine möglichst geordnete Abfolge von Bewegungen geht, sollte es auf jeden Fall ohne Verwendung des `goto`-Befehls programmiert werden, da dieser Befehl die Wegsuche weitgehend selbständig erledigt.
- Bei diesem Arbeitsblatt vollziehen die Schüler einen wesentlichen Schritt hin zum selbständigen Programmieren: Da weder wesentliche Teile des Programmcodes noch ein Struktogramm vorgegeben sind, müssen sie die Programmstruktur (Schleife, was gehört innerhalb, was gehört außerhalb der Schleife, Initialisierung von Zählvariablen etc. ) selbständig aus der verbalen Beschreibung des Problems und einzelner Lösungsschritte erkennen. Dies wird dadurch erleichtert, dass alle Elemente der Programmstruktur in den vorhergehenden Übungen eingehend diskutiert wurden – wenn auch bei anderen Aufgabenstellungen.
- Die verbalen Hinweise sollten den meisten Schülern bei entsprechendem zeitlichen Rahmen zum Erstellen beider Programme („Gegenstände zählen“ und „Gegenstände sortieren“) genügen. Bei leistungsschwächeren Schüler, die die Inhalte der vorhergehenden Übungen nicht oder nur teilweise verstanden haben, sind allerdings Probleme bei der selbständigen Lösung zu erwarten. Wenn hier entsprechende Hilfestellungen durch die Lehrperson nötig sind (zum Beispiel gemeinsame Erarbeitung eines Struktogramms), so weist dies auf ein mangelhaftes Verständnis vorhergehender Übungen hin. In diesem Fall ist es sinnvoll, die betroffenen Schüler ein zweites Mal mit den jeweiligen Übungen zu konfrontieren (beispielsweise nochmaliges Durcharbeiten im Rahmen einer Hausübung, eventuell mit entsprechenden Verständnisfragen in der darauf folgenden Unterrichtseinheit).
- Als weitere Zusatzübung und/oder Überprüfung des Wissensstandes in der darauf folgenden Unterrichtseinheit bietet sich an, in beiden Programmen die `switch`-Struktur durch mehrere `if`-Verzweigungen ersetzen zu lassen.
- Die Weiterführende Aufgabe Nr. 2 dient als Vorbereitung auf den nachfolgenden Wettbewerb und sollte auf jeden Fall als Hausübung gegeben werden.

### Arbeitsblatt 7.3: Sammelwettbewerb

- Befehle: `abstime`
- Beim Wettbewerb ist die Verwendung des `goto`-Befehls untersagt.
- Bevor die Schüler ihre Verbesserungsvorschläge für das Programm zum Sortieren der Gegenstände (Weiterführende Aufgabe Nr. 2 am Arbeitsblatt 6.2) in ein Programm umsetzen, sollten sie zumindest in Partnerarbeit ihre Vorschläge vergleichen und sich auf ein gemeinsames Struktogramm einigen. Wenn die entsprechende Unterrichtszeit zur Verfügung steht, sollten die Ideen gemeinsam durchdiskutiert werden, damit möglichst alle Schüler mit jedem Verbesserungsvorschlag konfrontiert werden.
- Da der Wettbewerb nur dann sinnvoll ist, wenn die Schüler den größten Teil der Programmentwicklung/-verbesserung selbst durchführen (zum Beispiel in Partnerarbeit), muss man den Schülern entsprechend viel Zeit zum Diskutieren und Programmieren geben. Für den

gesamten Wettbewerb (Programmieren der Verbesserungsvorschläge und Durchführung des Wettbewerbs) sollte zumindest eine Doppelstunde veranschlagt werden.
- Da immer zwei Roboter gegeneinander antreten, eignet sich dieser Wettbewerb besonders dazu, das unterschiedliche Verhalten der Roboter während ihrer Arbeit zu beobachten und zu kommentieren. Das direkte Antreten zweier besonders interessanter Lösungen gegeneinander (zum Beispiel über Beamer) bekommt so beinahe den Charakter eines sportlichen Wettbewerbs.

### Arbeitsblatt 7.4: Feuerleitstelle
- Befehle: `keypushed`, Endlosschleife mit `repeat(true)`

### Arbeitsblatt 8.1: Flugtraining
- Befehle: `jet`, `while`
- Falls im Unterricht Dualzahlen bereits besprochen wurden, kann hier als „klassisches“ Programmierbeispiel die Umwandlung einer ganzen Dezimalzahl x in eine Dualzahl behandelt werden: So lange x>1 ist, soll x immer wieder ganzzahlig durch 2 dividiert werden und die Reste dieser Divisionen zu einer Zeichenkette zusammengehängt werden.
- Als weitere Übungen bieten sich „9. Bedingte Schleifen 2“ / „Radarflug“ und „Auf und Ab“ an.

### Arbeitsblatt 8.2: Aufräumen 3
- Der „Wert“ einer nichtexistenten Instanz: `null`
- Die Nr. 2 der Weiterführenden Übungen ist etwas komplex, deshalb sind vorher die Übungen „7. Bedingte Schleifen 1“ / „Batterie 9“ und „Batterie 10“ zu empfehlen.
- Weiteres Übungsbeispiel: „7. Bedingte Schleifen 1“ / „Im richtigen Moment“

### Arbeitsblatt 8.4: Reaktionstest 1
- Befehle: rand, `do … while`
- Weiteres Übungsbeispiel: In “SB7: Verzweigungen” / “Aufräumen 2b” soll der Roboter genau so lange auf der Plattform der Ladestation warten, bis seine Batterie voll ist.

### Arbeitsblatt 8.5: Reaktionstest 2
- Das hier erstellte Programm zur Bestimmung der Reaktionszeit eignet sich gut zur Übertragung auf eine Entwicklungsumgebung für Java oder C++.

### Arbeitsblatt 9.1: Bergen des Flugschreibers
- Funktionen ohne Rückgabewert (`void`),aber mit Parametern

### Arbeitsblatt 9.2: Flugstaffel
- Befehle: Verwendung der selben Funktion mit einer oder zwei Parametern; `public`
- Die Ceebot-Übung „Flugstaffel“ ist recht anspruchsvoll. Insbesondere bei Zeitmangel empfiehlt es sich, die Lösung dieser Übung gemeinsam mit den Schülern zu erarbeiten, während der erste Teil des Arbeitsblatts (Bergung der Flugschreiber) von den Schülern möglichst selbständig bearbeitet werden sollte.
- Um deutlich zu machen, dass Änderungen im Programmcode durch die konsequente Verwendung von Funktionen stark vereinfacht werden, sollte man die Funktion `collectAll` in der Übung „Flugstaffel“ nach erfolgreicher Programmierung noch etwas modifizieren. Beispielsweise kann der Roboter am Ende eine Nachricht ausgeben, wie viele Gegenstände einer bestimmten Kategorie gesammelt wurden, oder er kann seine gesamte Flugstrecke und die dafür benötigte Gesamtzeit ausgeben. Der Programmcode muss dann eben nicht bei jedem Roboter geändert werden, sondern nur ein Mal in der Funktion `collectAll`.

### Arbeitsblatt 9.4: Komm zu mir
- Funktionen mit Rückgabewert

### Arbeitsblatt 10.1: Patrouillengang
- Deklaration und Verwendung von Feldern

- Je nach Leistungsfähigkeit der Schüler kann man im Anschluss an dieses Arbeitsblatt verschiedene Sortieralgorithmen besprechen. Allerdings sollten diese Algorithmen eher direkt in C++ oder in Java programmiert werden, womit sich eine gute Übung zum Umgang mit dem Debugger der jeweiligen Entwicklungsumgebung bietet.

### Arbeitsblatt 10.2: Langzeitspeicher

- Befehle: `file`, `open`, `writeln`, `readln`, `close`
- Als nicht ganz einfaches Übungsbeispiel bietet sich an, für den Reaktionstest von Arbeitsblatt 8.5 die Möglichkeit zu programmieren, den Highscore zu speichern.

### Arbeitsblatt 10.3: Zeichenmaschine

- Befehle: `eof`, `strfind`, `strleft`, `strright`, `strval`, `strlen`, *`drawto`*
- Die Zeichenmaschine kann auch in Java oder in C++ programmiert werden, womit der Umgang mit Dateien und mit Grafikdarstellung in der jeweiligen Programmiersprache geübt wird.

### Arbeitsblatt 11.1: Stoppuhr

- Definition und Verwendung einer Klasse
- Diese und die nächste Ceebot-Übung sind als erster Einstieg in die Objektorientierte Programmierung gedacht. Sie sollen klar machen, welchen Vorteil die Zusammenfassung zusammengehöriger Daten (Attribute) und Aktionen (Methoden) bieten. Die weitere Vermittlung der Objektorientierten Programmierung erfolgt am besten in C++ oder Java in einer speziell dafür entwickelten Entwicklungsumgebung wie zum Beispiel BlueJ.

### Arbeitsblatt 11.2: Rundenzähler

- Konstruktor
- Grundidee der Kapselung

### Fortsetzung der Programmierausbildung

Je nach Wunsch und Leistungsfähigkeit der Schülerinnen und Schüler ist es sinnvoll, teilweise parallel zu Ceebot bereits mit einer Entwicklungsumgebung zu arbeiten, mit der sich Konsolenanwendungen programmieren lassen. Der Einstieg in eine professionelle Entwicklungsumgebung gelingt den Schülerinnen und Schülern erfahrungsgemäß weit problemloser, wenn sie mit Hilfe von Ceebot bereits mit einigen grundlegenden Strukturen einer Programmiersprache vertraut gemacht wurden. Allerdings ist eine Entwicklung umfangreicher Programme keineswegs zu empfehlen, so lange den nicht zumindest die Möglichkeit bekannt ist, Teilschritte der Gesamtlösung in Form von Funktionen zu formulieren. Noch besser ist es, wenn die Entwicklung längerer Programme erst zu einem Zeitpunkt erfolgt, zu dem die Schüler gewisse Grundkenntnisse in Objektorientierter Programmierung besitzen, wie sie im letzten Kapitel des Schulbuchs vermittelt werden.
Um den Schülerinnen und Schülern die Fähigkeit zur Lösung relativ umfangreicher Programmieraufgaben zu vermitteln, bewährt sich vor allem die Erstellung von Spielen als Konsolenanwendung (beispielsweise unter Dev-C++ oder Borland-C++-Builder). Vom Schwierigkeitsgrad her passend wären hier zum Beispiel einfache Versionen der aus der Urzeit der Computerspiele und von Mobiltelefonen her bekannten Spiele Pacman, Brickout, Snake oder Tetris. Empfehlenswert, jedoch nicht unbedingt notwendig, ist die objektorientierte Umsetzung mit Hilfe von Klassen. Denkbar ist auch die Erstellung einer ersten Version in Rahmen von Ceebot durch Verwendung des Alphanumerischen Anzeigefensters, die dann auf eine C++- oder Java-Entwicklungsumgebung übertragen wird.
Es ist ebenso möglich, nach Vermittlung grundlegender Kenntnisse des Objektorientierten Programmierens in Kapitel 11 mit der Erstellung von Windows-Applikationen zu beginnen. Hier bieten sich Entwicklungsumgebungen wie der Borland-Builder für C++ oder Netbeans für Java an, die eine komfortable Erstellung von graphischen Benutzerinterfaces ermöglichen. Mit dem Verständnis der Begriffe „Klasse“, „Instanz“, „Attribut“ und „Methode“ sind die Schülerinnen und Schüler in der Lage, den von diesen Entwicklungsumgebungen automatisch generierten Programmcode zu verstehen und damit zu arbeiten.