QBASIC		(Quick Beginners All-purpose Symbolic Instruction Code = "Schneller Anfänger-Allzweck-symbolischer-Anweisungs-Code" ?!)
	Programm ausführen	UMSCHALT + F5

Nr.	Befehl	Erklärung
1	' (Apostroph, UMSCHALT + #)	Kommentare, die von QBasic nicht beachtet werden

Grundbefehle (PRINT, INPUT, IF...THEN...ELSE, ELSEIF, GOTO)		
2	PRINT "Was auch immer"	Zeigt Was auch immer am Bildschirm an.
3	PRINT "ABC" PRINT "DEF"	Zeigt ABC und in der nächsten Zeile darunter DEF an. Ein neues PRINT bedeutet eine neue Zeile.
4	PRINT "ABC": PRINT "DEF"	Ein Doppelpunkt erlaubt es, zwei Anweisungen in die gleiche Zeile zu schreiben. Ausgabe wie vorher (Den Doppelpunkt kann man mit fast allen Anweisungen verwenden, nicht nur PRINT).
5	PRINT "ABC"; "DEF"; PRINT "GHI"	Durch die Strichpunkte werden die Buchstaben direkt hintereinander Angezeigt, also ABCDEFGHI
6	PRINT "ABC", "DEF"	Zeigt zuerst ABC an, und dann in der nächsten Spalte (Tabulator) DEF.
7	Name$ = "Hugo" PRINT Name$	Speichert Hugo in der Variable Name$. Das $-Zeichen bedeutet, das es sich um Text und nicht um eine Zahl handelt. Danach wird der Inhalt der Variablen (Hugo) angezeigt. Anstatt Name$ kann man alles mögliche schreiben, aber keine Sonderzeichen und das $-Zeichen nicht vergessen. Beim PRINT: Name$ ohne Anführungszeichen!!!!
8	PRINT "Ich heiße "; Name$; "."	Kombinationen sind jederzeit möglich. Ausgabe: Ich heiße Hugo.
9	INPUT "Ihr Name: ", Name1$	INPUT zeigt wie PRINT den Text an, wartet danach aber auf eine Eingabe vom Benutzer. Diese Eingabe wird dann in Name1$ gespeichert.
10	INPUT "Anzahl:" , anzahl%	Wie vorher, nur dass dieses Mal nur Zahlen akzeptiert werden. (Für Berechnungen!) Das %-Zeichen nach den Variablennamen steht für Zahlen.

11	IF Name$ = "Hugo" THEN
	PRINT "Hallo Hugo!"
	END IF
		Untersucht, ob der Inhalt von Name$ Hugo ist. Wenn Ja, wird PRINT "Hallo Hugo!" ausgeführt. Es können belibig viele Anweisungen zwischen IF..THEN und END IF stehen. Wenn die Bedingung nicht erfüllt ist, wird in der Zeile nach END IF weitergemacht, der Teil dazwischen also einfach übersprungen.

12	IF Name$ = "Hugo" THEN
	PRINT "Hallo Hugo!"
	ELSE
	PRINT "Guten Tag!"
	END IF
		Die Anweisungen nach ELSE werden dann ausgeführt, wenn der Vergleich nicht stimmt. (ELSE = sonst).

13	IF Name$ = "Hugo" THEN
	PRINT "Hallo Hugo!"
	ELSEIF Name$ = "Hans" THEN
	PRINT "Tag, Hans!"
	ELSE
	PRINT "Guten Tag!"
	END IF
		Mit einem (oder mehreren) ELSEIF kann man noch mehrere Bedingungen einfügen.
		Die ELSEIF-Überprüfung findet nur dann statt, wenn der vorhergehende IF-Vergleicht nicht stimmte. Stimmt nichts von allen, wird der ELSE-Teil ganz am Schluss ausgeführt.

14	IF a$ = "ABC" THEN PRINT "xy"	Das Ganze geht auch in nur einer Zeile. (Dann ohne END IF!). Mehrere Anweisungen müssten mit Doppelpunkten getrennt werden, Die mehrzeilige Variante ist dann aber besser. Ein ELSE-Teil kann in der selben Zeile noch angehängt werden.
15	IF Zahl% = 13 THEN ...	Man kann auch Zahlen und Zahlenvariablen miteinander vergleichen.
16	GOTO Start ... Start:	GOTO springt zu einer im Programm definierten Marke. Eine Marke besteht aus einem Namen und einem Doppelpunkt.
17	10 .... GOTO 10	Man kann auch Zahlen als Marken verwenden. Dann braucht man keinen Doppelpunkt.

Häufig verwendete Befehle (CLS, LOCATE, COLOR, GOSUB, RANDOMIZE, RND, SELECT CASE, INKEY$, CHR$, ASC, CONST, SYSTEM)		
18	CLS	(=CLearScreen, Anzeige löschen) Alles auf dem Bildschirm löschen (sollte am Anfang von jedem Textmodus-Programm stehen)

19	LOCATE 2, 35 PRINT "ABC"	Setzt den (unsichtbaren) Cursor für PRINT an eine bestimmte Koordinate am Bildschirm. 1.Zahl: Zeile, 2.Zahl: Spalte. (Also ist (2, 1) das Zeichen unter (1,1)) Der Bereich geht von (1,1) bis (24, 80) im normalen Textmodus.
20	COLOR 1	COLOR setzt die Farbe für PRINT-Ausgaben (und auch Grafik), hier blau. (Siehe Farbtabelle)
21	COLOR 1, 2	Die zweite Zahl ist die Hintergrundfarbe der einzelnen Zeichen. (hier grün)
22	COLOR 1 + 16	Wenn man den Farbwert um 16 vergrößert, blinkt die der Text in dieser Farbe. Nur Vordergrund, nur Text!
23	GOSUB 10 ... 10 ... RETURN	GOSUB springt zu einer Marke bzw. Zahl (wie GOTO), kehrt allerdings bei RETURN wieder zur Zeile nach dem GOSUB zurück. Gut für kleine SUBs (Unterprogramme) ohne besondere Parameter.
24	RANDOMIZE TIMER Zzahl% = INT(RND * 6) + 1	RANDOMIZE initialisiert den Zufallszahlengenerator. Verwendet man anstatt TIMER eine Zahl, so ergeben sich bei jeden Programmstart dieselben Zufallszahlen. Mit INT(RND * x) + 1 kann man dann eine Zufallszahl zwischen 1 und x erzeugen. (RND selbst liefert eine Zahl zwischen 0 und 1)
25	INPUT "Zahl (1 bis 6) : ", Zahl% SELECT CASE Zahl% CASE 1: PRINT "1" CASE 2, 3: PRINT "2 oder 3" CASE 4 TO 6: PRINT "4 bis 6" CASE ELSE PRINT "Ungültige Zahl!" END SELECT	SELECT CASE überprüft eine Variable auf bestimmte Werte. In den folgenden Zeilen steht "CASE" + ein oder mehrere Werte, danach jeweils die Anweisungen, die ausgeführt werden sollen. (Hier mit Doppelpunkt in der gleichen Zeile. Bei mehreren Anweisungen neue Zeilen verwenden!). Durch Kommas lassen sich einzelne Werte zusammenfassen, mit x TO y Reihen (Auch Kombinationen möglich). Die Anweisungen von CASE ELSE werden ausgeführt, wenn keiner der Werte zutrifft. END SELCET dient als Abschluss des SELECT CASE-Blocks.
26	Taste$ = INKEY$	Überprüft, ob im Moment der Abarbeitung eine Taste gedrückt wird. Wenn ja, wird die Taste als Ergebnis zurückgegeben. z.B: Taste A: "a", Taste 7: "7" usw. Verwendung immer in Schleifen. Auswertung meist mit SELECT CASE.
27	PRINT CHR$(65)	CHR$ gibt ein Zeichen mit einem Bestimmten ASCII-Wert zurück. (von 65: ein großes A). ASCII-Werte siehe QBasic-Hilfe/Inhalt/ASCII-Zeichen-Codes. Wird zusammen mit INKEY$ verwendet, um Sondertasten auszuwerten, oder um Sonderzeichen anzuzeigen.
28	PRINT ASC("A")	ASC ist die Umkehrfunktion von CHR$ und gibt den ASCII-Code eines Zeichens zurück. Ausgabe: 65
29	CONST MaxWert = 1000	Legt eine Konstante fest. Konstanten sind read-only "Variablen", die immer dann verwendet werden, wenn eine bestimmte (meist größere) Zahl öfter im Programm vorkommt und man diese an allen Stellen im Programm auf einmal ändern will und nicht an jeder Stelle einzeln.
30	SYSTEM	Beendet das Programm und kehrt zu Qbasic bzw. dem Betriebssystem zurück. (Sinnvoll, wenn das Programm durch eine Stapelverarbeitungsdatei oder eine Verknüpfung aufgerufen wird)

Operatoren (+, -, *, /, ^, MOD, \, SQR, INT, ABS)		
	Diese Operatoren funktionieren nur mit Zahlenvariablen:	
31	PRINT 12 + 3	Zeigt das Ergebnis (15) an.
32	a% = 13 - 2	Weist a% das Ergebnis aus 13 - 2 zu. (Entspricht a% = 11)
33	a% = a% * 2	Multipliziert zuerst den Wert von a% mit 2 und weist a% das Ergebnis zu. Beispiel: a% vorher: 10, nachher: 20.
34	a# = 4 / 3	Teilt 4 durch 3 und zeigt die Zahl mit Nachkommastellen an. (1.3333333) Das # nach dem Variablennamen bedeutet, dass die Zahl Nachkommastellen haben kann. (Siehe Variablentypen)
35	PRINT 11 ^ 2	Zeigt 11 hoch 2 = 11 * 11 = 121 an.
36	Rest% = 17 MOD 3	MOD liefert den Rest aus einer Ganzzahldivision. 17 MOD 3 = 2 (17 durch 3 = 5 Rest 2)
37	PRINT = 17 \ 3	Führt eine Ganzzahldivision durch. Ergebnis: 5 (nicht 5.6666667), den Backslash macht man mit Alt Gr + ß.
38	a# = SQR (2)	Berechnet die Quadratwurzel aus 2 (1,4142...). SQR = SQuareRoot
39	a# = 1.75: PRINT FIX(a#)	FIX schneidet die Nachkommastellen einer Zahl ab. Anzeige: 1.
40	a% = -43: PRINT ABS(a%)	ABS liefert den positiven Wert (Absolutwert) einer Zahl. Anzeige: 43.
41	a# = 1.75: PRINT CINT(a#)	CINT rundet eine Zahl. Anzeige: 2.

Variablentypen und Datenorganisation ($, %, &, !, #, DEFxxx, DIM, TYPE)

	Variablen können je nach Endung verschiedene Werte aufnehmen: (Vgl. Qbasic-Hilfe/Inhalt/Grenzen der Qbasic-Umgebung/1.)	
42	a% = 3	Ganzzahl (INTEGER): Zahlen zwischen -32768 und 32767
43	a& = 1000000000 '(1 Mrd.)	Lange Ganzzahl (LONG): Zahlen zwischen -2.147.483.648 und 2.147.483.647
44	a! = 1.5	Einfache Ganauigkeit (SINGLE): Zahl mit bis zu 7 Nachkommastellen
45	a# = 1 / 3	Doppelte Genauigkeit:(DOUBLE): Zahl mit bis zu 16 Nachkommastellen
46	a$ = "Hallo!"	Zeichenketten(STRING): Bis Max. 255 Zeichen, in Anführungszeichen
	Variablen ohne Endung werden automatisch Typ SINGLE, sofern keine DEFxxx-Anweisung vorhanden ist:	
47	Zahl = 1 / 3 PRINT Zahl	Ergebnis : .3333333 (Die 0 wird weggelassen, ami-Schreibweise)
	Wenn dieser Standard-Variablentyp verändert werden soll, so schreibt man als 1.Zeile des Pragramms ein DEFxxx. Die Endungen %, &, !, #, $ oder Deklarationen mit DIM haben natürlich immer Vorrang.	
48	DEFINT A-Z	Alle Variablen ohne Endung sind vom Typ INTEGER.
49	DEFLNG A-Z	Für LONGs. (Das A-Z bedeutet, dass alle Variablen gemeint sind, Überbleibsel von alten BASIC-Versionen)
50	DEFSNG A-Z	SINGLE.
51	DEFDBL A-Z	DOUBLE.
52	DEFSTR A-Z	Für STRINGs. Achtung: Alle Zahlenvariablen müssen dann eine Endung haben !
	Man kann Variablen auch einzeln einen Typ zuweisen:	
53	DIM Zahl AS LONG	Die Variable Zahl ist jetzt ein LONG, auch bei anderem DEFxxx. Ohne Endung!
	Mit DIM kann man STRINGs mit konstanter Länge erzeugen:	
54	DIM Wort AS STRING * 3 Wort = "ABCDE"	Der String Wort hat jetzt eine Länge von 3 Zeichen. Alle Zeichen danach werden abgeschnitten: In Wort wird nur "ABC" gespeichert.
55	DIM Wort AS STRING * 6 Wort = "DEF" PRINT "ABC"; Wort; "GHI"	Ist der inhalt eines STRINGs mit konstanter Länge kürzer wie die Größe, so wird der STRING mit Leerzeichen aufgefüllt. Anzeige: "ABCDEF GHI"
	Wenn man viele Daten hat, kann man diese in einem "Datenfeld" speichern. Ein Datenfeld hat einen normalen Namen wie eine Variable, und für jeden Eintrag eine Zahl (Index). Datenfelder werden mit DIM deklariert:	
56	DIM Feld(1 TO 30) AS INTEGER	Deklariert das Datenfeld "Feld" mit 30 Einträgen.
57	Feld(1) = 25: Feld(5) = -13 IF Feld(30) = Feld(29) THEN ...	Man kann durch die Angabe eines Index innerhalb des deklarierten Bereichs die einzelnen Einträge ansprechen.
	Es gibt auch "mehrdimensionale" Datenfelder:	
58	DIM Quadrat(1 TO 10, 1 TO 10)	Deklariert ein Feld von 10*10 Einträgen.
59	Quadrat(2, 5) = 2	So werden die einzelnen Einträge benutzt.
	Oft ist es sinnvoll, ein Datenfeld zu haben, das aus verschiedenen Datentypen besteht, z.B. eine Tabelle mit 100 Einträgen, und in jedem Eintrag ein String(z.B. Produktname) und eine Zahl(z.B. den Preis). Dazu gibt es benutzerdefinierte Datentypen mit TYPE:	
60	TYPE TabelleEintrag Produkt AS STRING * 30 Preis AS LONG END TYPE	Zuerst TYPE und der Name des neuen Datentyps (Hier TabelleEintrag). In den Zeilen danach die einzelnen Untereinträge. Bei Strings muss (leider) die Länge fest vorgegeben werden, also vorher überlegen, wie lang der längste Eintrag höchstens wird. Das Ganze wird mit END TYPE abgschlossen.
	Bei der Deklaration des Datenfeldes wird der benutzerdefinierte Datentyp wie ein "normaler" behandelt:	
61	DIM Tabelle (1 TO 100) AS TabelleEintrag	
62	Tabelle(1).Produkt = "Typ 1" Tabelle(1).Preis = 200	Angesprochen werden die einzelnen Einträge mit dem Namen des Datenfeldes (Tabelle), dem Index (in Klammern), einem Punkt und dem Namen eines Untereintrags (Produkt oder Preis).

Schleifen (GOTO, DO...LOOP UNTIL, WHILE...WEND, FOR...NEXT, EXIT DO/FOR)

	Schleifen dienen dazu, bestimmte Programmteile zu wiederholen.	
63	1 PRINT "******************" GOTO 1	Eine GOTO-Schleife. Das Programm zeigt unendlich-Mal die Sternchen an. Achtung Endlosschleife: Hier fehlt eine Anweisung, um das Programm zu Beenden! Das Programm hängt sich auf !!!! (Mit Alt+Strg+Entf rausschmeißen, und hoffen dass man vorher gespeichert hat...)
!!!	Schleifen mit GOTO sind schlecht, da man eine IF-Anweisung braucht, um wieder herauszukommen. Folgende Möglichkeiten sind besser und immer zu bevorzugen:	
64	DO INPUT "Zauberwort: ", ZWort$ LOOP UNTIL ZWort$ = "bitte"	Das Programm läuft so lange, bis der Benutzer das "Zauberwort" erraten hat. Bei dieser Schleifenform werden die Anweisungen zwischen DO und LOOP auf jeden Fall mindestens ein Mal ausgeführt. (DO LOOP UNTIL = "Tu Schleife bis"), also: die Schleife wird so lange ausgeführt, bis die Bedingung erfüllt ist.

65	EXIT DO	Nur innerhalb einer DO-LOOP-Schleife: Bewirkt ein sofortiges Verlassen der Schleife. (Es geht in der Zeile nach dem LOOP weiter). Funktioniert auch mit FOR-Schleifen, dann aber EXIT FOR.
66	WHILE x <= 10 INPUT "Zahl > 10: ", x WEND	Die WHILE...WEND-Schleife ist genau das Gegenteil: Die Anweisung innerhalb der Schleife kann ggf. ganz übersprungen werden (wenn x schon 10 oder größer ist) und die Schleife läuft so lange, wie die Bedingung erfüllt ist. (Also x kleiner/gleich 10 ist)
67	FOR x = 1 TO 10 STEP 1 PRINT "**********" NEXT x	FOR weist der Variablen (x) zuerst 1 zu, führt die Anweisungen bis zum NEXT aus, zählt x um 1 hoch, führt wieder die Anweisungen aus ... Bis x 10 ist. Nach dem NEXT sollte der Name der Zählvariablen stehen (hier x). Ein STEP 2 würde bedeuten, dass x immer in Zweierschritten hochgezählt wird. FOR-Schleifen werden oft mit Datenfeldern verwendet, um die einzelnen Einträge der Reihe nach anzusprechen, oder um bestimmte Anweisungen x-Mal zu wiederholen.

Vergleiche (=, <, >, <=, >=, <>, AND, OR, XOR, NOT)		
	Diese Vergleichstypen kann man außer in IF-Anweisungen auch für z.B. für Schleifen-Bedingungen verwenden:	
68	IF a = b THEN ...	Wenn a gleich b ist...
69	IF a < b THEN ...	Wenn a kleiner wie b ist...
70	IF a > b THEN ...	Wenn a größer wie b ist ...
71	IF a <= b THEN ...	Wenn a kleiner oder gleich b ist...
72	IF a >= b THEN ...	Wenn a größer oder gleich b ist...
73	IF a <> b THEN ...	Wenn a ungleich wie b ist...
74	IF a = b% AND c = d ...	Nur wahr, wenn a = b und c = d
75	IF a = b% OR c = d	Nur wahr, wenn a = b oder c = d (oder beides)
76	IF a = b% XOR c = d	Nur wahr, wenn a = b oder c = d (aber nicht beides)
77	IF NOT (a = b) THEN ...	Nur wahr, wenn a = b nicht stimmt.
78	IF ((a = b) AND (c = d)) OR (a < d)	
	Alles kann beliebig kombiniert werden. Wichtig: Dann Klammern verwenden ! (Besser zuviel als zuwenig)	

STRINGs (=, +, LEFT$, RIGHT$, MID$, LEN, VAL, STR$, LTRIM$, RTRIM$, UCASE$, LCASE$)		
79	Str1$ = "ABC"	"ABC" wird in der Variable Str1$ gespeichert. "ABC" in Anführungszeichen.
80	Str2$ = Str1$ + "DEFG"	Bei Strings bedeutet das +-Zeichen "anhängen". Ergebnis: Str2$ = "ABCDEFG"
81	Str3$ = LEFT$(Str2$, 6)	LEFT$ gibt hier die ersten 6 Zeichen von Str2$ zurück. Ergebnis: Str3$ = "ABCDEF"
82	Str4$ = RIGHT$(Str3$, 5)	RIGHT$ gibt hier die letzten 5 Zeichen von Str3$ zurück. Ergebnis: Str4$ = "BCDEF"
83	Str5$ = MID$(Str4$, 2, 3)	MID$ holt Zeichen aus der Mitte: die erste Zahl ist die Startposition, die zweite die Anzahl von Zeichen. Ergebnis: Str5$ = "CDE"
84	MID$(Str5$, 2, 1) = "H"	Mit MID$ lässt sich auch ein Teil eines Strings durch einen Anderen ersetzen: Ergebnis: Str5$ = "CHE". 1.Zahl (2): Startposition, 2.Zahl (1): Anzahl Zeichen.
85	Laenge = LEN(Str5$)	LEN gibt die Anzahl Zeichen in einem String zurück (incl. Leerzeichen), hier 3
86	Str6$ = "12345" Zahl% = VAL(Str6$) Zahl2% = VAL("43r5") ' = 43 Zahl3% = VAL("13e3") ' = 13000	VAL wandelt eine Zeichenkette aus Zahlen in eine Ganzzahl, mit der man rechnen kann, um. Befindet sich ein Buchstabe im String, so werden nur die Ziffern davor in eine Zahl umgewandelt. Beispiel: VAL("123A45r6") gibt nur 123. Achtung: ein "e" wird als 10er-Exponent gedeutet: VAL("12e2") gibt 1200.
87	Str7$ = STR$(3485) Str8$ = STR$(-1234)	STR$ wandelt eine Zahl in einen String um. Achtung: Der fertige String enthält als erstes Zeichen ein Leerzeichen (Als Platzhalter für ein Minus-Zeichen bei negativen Zahlen). LTRIM$ verwenden!
88	Str9$ = " ABC " Str10$ = "*" + LTRIM$(Str9$) Str11$ = RTRIM$(Str10$) + "*"	LTRIM$ entfernt alle Leerzeichen am Anfang eines Strings, RTRIM$ alle am Ende (Das Ergebnis wird zurückgegeben, am Original-String ändert sich nichts). Ergebnisse: Str10$ = "*ABC ", Str11$ = "*ABC*"
89	Str12$ = "AbcDEfG" Str13$ = UCASE$(Str12$) Str14$ = LCASE$(Str12$)	UCASE$ (=UpperCase) verwandelt alle Buchstaben in einem String in Großbuchstaben, LCASE$ (=LowerCase) in Kleinbuchstaben. Ergebnisse: Str13$ = "ABCDEFG", Str14$ = "abcdefg"

Datei Ein/Ausgabe (OPEN, PRINT, LINE INPUT, EOF, PUT, GET, LOF, CLOSE, KILL)	
90	OPEN "C:\TEST.TXT" FOR XXXXXX AS #1
	Öffnet eine Datei, um in sie zu schreiben oder aus ihr zu lesen. Nach dem Wort OPEN kommt als Erstes der Dateiname (In Anführungszeichen, ggf. mit Pfad). Als nächstes der Modus:

91	... FOR OUTPUT ...	Erstellt die Datei oder löscht ihren Inhalt. Der Dateizeiger wird an den Dateianfang gesetzt; Schreibzugriff; Text.
92	... FOR APPEND ...	Dateizeiger am Ende; Schreibzugriff; Text.
93	... FOR INPUT ...	Dateizeiger am Anfang; Lesezugriff; Text. Nur bestehende Dateien!
94	... FOR BINARY ...	Lese-/Schreibzugriff; Binär.
95	... FOR RANDOM ...	Lese-/Schreibzugriff; Daten.
	Zum Schluss steht noch AS und die Dateinummer: #1. Wenn mehrere Dateien geöffnet werden sollen: #2, #3...	
	In Textdateien (also als output oder append geöffnet) kann man schreiben wie an den Bildschirm, es muss nur noch die Dateinummer angegeben werden:	
96	PRINT #1, "Hallo!"	Schreibt Hallo! In die Datei. (Diese muss natürlich fürs Schreiben geöffnet sein)
97	DO LINE INPUT #1, Zeile$ ... LOOP UNTIL (EOF(1))	LINE INPUT liest zeilenweise aus einer fürs Lesen geöffneten Datei (Text) in eine Zeichenkettenvariable. Das LOOP UNTIL (EOF(1)) wird gebraucht, um das Dateiende zu Erkennen. Die Zahl in der Klammer von EOF ist die Dateinummer der Datei. (EOF = End of file)
	Der Binärzugriff ist dann sinnvoll, wenn man Dateien öffnen will, die nicht (nur) aus Text bestehen, wie z.B. Programme oder Grafiken. Ein Vorteil ist auch, dass man an jede beliebige Stelle in der Datei schreiben bzw. jede lesen kann.Das Ganze geht nicht Zeilen- sondern Zeichenweise mit PUT und GET.	
98	Str1$ = "Hallo!" PUT #1, 1, Str1$	Schreibt Hallo! in die Datei (BINARY-Modus) an Byte-Position 1, also den Anfang. Eine Variable ist erforderlich, keine Konstante als 3. Parameter!
99	DIM Str2 AS STRING * 3 GET #1, 2, Str2 PRINT Str2	Liest aus der Datei ab Position 2 in die Stringvariable Str2. Die Stringvariable muss vorher eine feste Länge bekommen (wie man das macht ist egal), damit GET weiß, wieviele Zeichen es einlesen soll. Anzeige: all
100	BytesInFile = LOF(1)	LOF (Lenght of File) gibt die Länge der Datei in Bytes zurück. Gut für FOR-Schleifen, die jedes Zeichen der Datei der Reihe nach verarbeiten. Die Zahl in Klammern ist die Dateinummer einer geöffneten Datei (wie bei LEN).
	Als 3. Möglichkeit gibt es noch den RANDOM-Access-Modus (Wahlfreier Zugriff). Diesen sollte man dann verwenden, wenn das Programm Daten speichern soll, die dann auch nur vom Programm selbst wieder benötigt werden. Hier wird mit "Datensätzen" gearbeitet, die im Normalfall benutzerdefinierte Datentypen sind. Ich gehe hier vom Beispiel mit der Produkt/Preis-Tabelle (Befehl TYPE, siehe weiter oben) aus: (mit Daten in Tabelle!)	
101	FOR x = 1 TO 100 PUT #1, x, Tabelle(x) NEXT x	Schreibt alle 100 Datensätze des Datenfeldes mit allem drum und dran in die Datei. Der 2. Parameter (hier: x) ist die Datensatznummer, über die einzelne Einträge in der Datei angesprochen werden können.
102	GET #1, 100, Tabelle(1)	Liest den 100. Datensatz aus der Datei und speichert den Inhalt in den 1. Datensatz von Tabelle() im Speicher.
!!!	Wenn man in der Datei nur einen einzigen benutzerdefienierten Datentyp speichern will (wie hier), dann kann man bei OPEN die Dateigröße optimieren, indem man LEN = und die "Länge" des Datentyps hinten anhängt. Die Länge ermittelt man mit LEN + in Klammern eine beliebige Instanz des Datentyps, hier z.B. Tabelle(1):	
103	OPEN "C:\TEST.TXT" FOR RANDOM AS #1 LEN = LEN(Tabelle(1))	
	Nach der letzten Lese-/Schreiboperation mit einer Datei sollte man diese immer auch wieder schließen:	
104	CLOSE #1	Schließt eine offene Datei.
105	CLOSE	CLOSE ohne Zahl schließt alle offenen Dateien.
106	KILL "C:\TEST.TXT"	Löscht die Datei aus dem Dateisystem. Achtung, nicht wiederherstellbar!!!!

Grafik (SCREEN, PSET, LINE, DRAW, PAINT, CIRCLE, POINT, GET, PUT, WIDTH)		
	Für die Grafikdarstellung braucht man einen bestimmten Bildschirmmodus. Dieser lässt sich mit SCREEN ändern:	
107	SCREEN 12	Schaltet in den 640*480*16-Modus um. (horizontal*vertikal*Farben)
108	SCREEN 13	Schaltet in den 320*200*256-Modus um.
109	SCREEN 0	Zurück in den normalen Textmodus.
	Leider lassen sich mit QBasic keine SVGA-Grafiken darstellen. Die oben genannten Modi sind die Besten.	
!!!	Koordinatensystem im Grafikmodus: (0, 0) liegt in der Links-oberen Ecke, (639, 0) Rechts-oben, (0, 479) Links-unten und (639, 479) Rechts-unten. 1.Zahl: Spalte, 2.Zahl: Zeile (Also genau andersrum wie bei Text)	
110	PSET (100, 100), 2	Setzt ein Pixel mit der Farbe 2 (grün) auf die Koordinate (100, 100)
111	LINE (0, 0)-(50, 10), 4	Zieht eine Linie zwischen Punkt 1 (0, 0) und Punkt 2 (50, 10) mit Farbe 4 (rot).
112	LINE (0, 0)-(50, 10), 4, B	Zeichnet ein Rechteck anstatt einer Linie. Ein BF anstatt B würde das Rechteck ausfüllen. (B = Bar, F = Filled)
113	DRAW "...."	DRAW zeichnet Bilder anhand des Strings. Groß/Kleinschreibung ist egal.
114	DRAW "c2"	(c = color): setzt die Zeichenfarbe zu 2 (grün). (Entspricht COLOR 2)
115	DRAW "bm100,100"	(BM = blind move): Setzt den Cursor zu (100, 100).

116	DRAW "r3" DRAW "r"	(r = right): bewegt den Cursor um 3 Pixel nach Rechts, zeichnet dabei in der aktuellen Farbe. Achtung: Das aktuelle Cursor-Pixel wird mit angemalt, also insgesamt 4 Pixel. Ein r ohne Zahl entspricht r1.
117	DRAW "l5"	(l = left): zeichnet 5 Pixel nach links.
118	DRAW "u"	(u = up): nach oben.
119	DRAW "d"	(d = down): nach unten
120	DRAW "e"	nach rechts-oben
121	DRAW "f"	nach rechts-unten
122	DRAW "g"	nach links-unten
123	DRAW "h"	nach links-oben
124	DRAW "e20f20l40"	Zeichnet ein rechtwinkliges Dreieck mit Grundseitenlänge 40.
125	DRAW "nu10 nd10 nl10 nr10"	Zeichnet ein Fadenkreuz. Das n bedeutet, dass der Cursor nach dem Zeichnen wieder an die Startposition zurückgesetzt wird.
126	DRAW "r20 br10 r20"	Zeichnet zwei Striche nebeneinander. Das b bedeutet, dass die 10 Pixel nicht gezeichnet werden sollen.
127	DRAW "m120,110" DRAW "m+20,+10" DRAW "m-20,+10"	Mit m kann man eine Linie zwischen der Cursorposition und einer bestimmten Koordiante ziehen. Steht nach dem m ein + (oder ggf. -), so wird die Koordinate relativ zum Cursor gesehen: hier 20 nach rechts, 10 nach unten (für oben -10)
128	DRAW "bm+20,10"	Das Ganze geht natürlich auch ohne zu zeichnen.
129	DRAW "c1 d100r100u100l100" DRAW "bm+1,+1 p2,1"	Zeichnet zuerst ein Quadrat, der Cursor ist zuletzt in der links-oberen Ecke. Dann unsichtbar den Cursor in das Quadrat hineinsetzen. Das p2,1 entspricht einer PAINT-Anweisung an der Cursorposition. (siehe PAINT weiter unten)
130	DRAW "TA15 u50 TA0 u50"	TA dreht einen Winkel (hier 15°) gegen den Uhrzeigersinn. Alle Zeichnungen danach sind um diesen Winkel verdreht. Für Drehungen im Uhrzeigersinn negative Zahlen verwenden, z.B. TA-15, TA0, um alles wieder normal einzustellen.
131	DRAW "C1 BM100,100" DRAW "D200R200U200L200" PAINT (101, 101), 2, 1	PAINT füllt einen Bereich mit einer Farbe aus. Die Koordinate muss innerhalb dieses Bereiches liegen. Der Bereich muss lückenlos von einer einheitlichen Farbe umgeben sein (Randfarbe). Die erste Zahl ist die Frabe zum Ausfüllen, die zweite die Randfarbe. Hier: Farbe: 2 (grün), Randfarbe: 1 (blau)
132	CIRCLE (300, 200), 150, 14 PAINT (300, 200), 14, 14	Zeichnet einen Kreis mit dem Mittelpunkt (300, 200). 150 ist der Radius (In Pixeln), 14 die Farbe (gelb). Der Kreis kann auch ausgefüllt werden (wie hier)
133	Farbe% = POINT (100, 100)	POINT gibt den Farbwert des Pixels an der Koordinate zurück.
134	col% = POINT(0)	POINT(0) gibt die x-Koordiante des Grafikcursors zurück.
135	row% = POINT(1)	POINT(1) gibt die y-Koordinate zurück
136	DIM Feld (1 TO 400) AS LONG GET (0, 0)-(50, 50), Feld(1)	GET speichert einen Bildschirmauschnitt in einem Datenfeld. Der Auschnitt ist ein Rechteck, von dem die links-obere und die rechts-untere Ecke angegeben werden. Das Feld muss groß genug sein, um die Daten aufzunehmen (ausprobieren!)
137	PUT (10, 10), Feld(1), PSET	PUT zeichnet ein von GET gespeichertes Rechteck wieder auf den Bildschirm. Die Koordinate ist die Links-obere Ecke des Rechtecks. Anstatt PSET können auch andere Möglichkeiten probiert werden. (Siehe QBasic-Hilfe, PUT (Grafik))

SUBs und Funktionen (DECLARE SUB/FUNCTION, SUB, FUNCTION, EXIT SUB, SHARED)

	SUBs sind Unterprogramme, die zum einen Teil das Programm übersichtlicher machen und zum anderen Teil Wiederholungen von bestimmten Operationen ermöglichen, die spezielle Parameter benötigen.	
138	DEFINT A-Z	Wenn man SUBs oder Funktionen verwendet, muss unbedingt eine DEFxxx-Anweisung am Dateianfang stehen.
	Wir wollen jetzt ein kleines Unterprogramm schreiben, mit dem man LOCATE und PRINT mit einem Befehl ausführen kann. Dazu braucht man die Parameter y für die Zeile, x für die Spalte und Txt$ für den Ausgabetext.	
139	DECLARE SUB Text (y, x, Txt$)	Dient als Erklärung für das Programm, welche Parameter verwendet werden. Muss am Dateianfang nach der DEFxxx-Anweisung stehen.
140	Text 1, 35, "Überschrift" Text 3, 1, "abcdefghijklmnop"	So wird das Unterprogramm aufgerufen: Name und Parameter, mit Kommas getrennt. Das Unterprogramm kann beliebig oft ausgeführt werden.
	Jetzt muss noch eingegeben werden, wass das Unterprogramm überhaupt machen soll: Dazu geht man im Menü Bearbeten auf den Menüpunkt "Neue SUB..." und gibt den Namen der SUB ein (hier: Text). Danach landet man in einem neuen Fenster, in dem bereits eine DEFxxx-Anweisung und "SUB Text" steht. Dahinter sollte man nun noch die Parameterliste von DECLARE SUB schreiben:	

141	DEFINT A-Z SUB Text (y, x, Txt$) LOCATE y, x PRINT Txt$ END SUB	Danach kann man sein Unterprogramm schreiben. Die "Zeile END SUB" muss als letztes stehen. Das Unterprogramm wird die Werte vom Aufruf (Also im Beispiel oben y = 1, x = 35, Txt$ = "Überschrift) übernehmen. Allerdings sind x, y und Txt$ read-only, es sollte ihnen also nichts zugewiesen werden, auch wenn es theoretisch evtl. möglich ist.
!!!	Um zwischen Haupt-und Unterprogrammen umzuschalten, drückt man im Editor F2 und wählt den Programmteil aus. Der 1. Eintrag (Der Dateiname) ist das Hauptprogramm.	
142	EXIT SUB	Bewirkt ein sofortiges Verlassen der SUB, es geht in dem Programmteil weiter, der die SUB aufgerufen hat.
	Funktionen arbeiten wie SUBs, nur dass sie einen Wert an das Hauptprogramm zurücksenden. (z.B. das Ergebnis einer Berechnung etc.)	
143	DECLARE FUNCTION Durchschnitt#(zahl1, zahl2, zahl3)	
	Diese Funktion soll den Durchschnitt aus drei Zahlen berechnen und das Ergebnis an das Hauptprogramm zurücksenden. Da der Rückgabewert ein DOUBLE sein soll, endet der Funktionsname mit #.	
	Als nächstes wieder Bearbeiten - "Neue FUNCTION...", Name eingeben (Durchschnitt#) und im Unterprogramm die Parameterliste vervollständigen.	
144	FUNCTION Durchschnitt# (zahl1, zahl2, zahl3)	
	Danach kommt der Inhalt der Funktion. Damit das Programm weiß, was es zurück an das Hauptprogramm senden soll, weist man dem Namen der Funktion das Ergebnis zu:	
145	Durchschnitt# = (zahl1 + zahl2 + zahl3) / 3	
146	END FUNCTION	Damit ist die Funktion fertig. Im Hauptprogramm oder in anderen Unterprogrammen wird die Funktion dann aufgerufen:
147	a# = Durchschnitt (11, 13, 15)	Das Programm errechnet 13 als Ergebnis und weist es a# zu. (11 + 13 + 15) / 3 = 13
	In SUBs oder Funktionen können keine Variablen des Hauptprogramms verwendet werden, es sei denn, diese werden vorher im Hautprogramm mit COMMON SHARED als global deklariert:	
148	COMMON SHARED a%, Str1$	Diese Variablen können jetzt in Unterprogrammen verwendet werden.
149	DIM SHARED Zahl AS LONG	Ein SHARED in einer DIM-Anweisung bewirkt dasselbe, auch für Felder.

Sound (BEEP, SOUND, PLAY)

	Leider können mit Qbasic keine Töne über die Soundkarte wiedergegeben werden. Nur PC-Lautsprecher.	
150	BEEP	Erzeugt einen Piepston.
151	SOUND 440, 18	Erzeugt einen Ton mit der Frequenz 440Hz und einer Länge von 18 Zeiteinheiten. (440Hz = a', 18 ZE = 1sec)
	Mit PLAY kann man Melodien abspielen lassen. Bedienung ähnlich wie bei DRAW. Achtung: h = b!	
152	PLAY "CDEFGAB"	Spielt die Töne C bis H.
153	PLAY "L1 C L2 D L4 E L8 F"	L bestimmt die Tonlänge für alle nachfolgeneden Noten. L1 = ganze Noten, L2 = halbe Noten... Bis L64 = Vierundsechzigstel.
154	PLAY "AB>CDEFGAB>CD"	Das Größer-Zeichen (>) macht alle Noten danach eine Oktave höher.
155	PLAY "GFEDC<BAGFEDC<BA"	Das Kleiner-Zeichen (<) macht alle Noten danach eine Oktave tiefer.
156	PLAY "L4 P4 C P8 D	P macht eine Pause. P4 = Viertelpause, P8 = Achtelpause...
157	PLAY "F+ C+ G+"	Spielt Fis, Cis, Gis. Das Plus-Zeichen muss nach der Note stehen.
158	PLAY "B- E- A-"	Spielt Be, Es, As
159	PLAY "L4 C. L8 C.	Spielt erst eine punktierte Viertel und dann eine punktierte Achtel.

Datum / Zeit (DATE$, TIME$, TIMER, SLEEP)

160	PRINT DATE$	DATE$ gibt das aktuelle Datum als String im Format mm-tt-jjjj zurück.
161	PRINT TIME$	TIME$ gibt die Systemzeit als String im Format hh:mm:ss zurück.
162	PRINT TIMER	TIMER gibt die seit Mitternacht vergangenen Sekunden zurück. Sinnvoll, um Zeitdauern zu ermitteln oder festzulegen. Auf 1/100 Sekunden genau.
163	SLEEP x	Unterbricht das Programm für x Sekunden. Wenn während den x Sekunden etwas getan werden soll, stattdessen eine Konstruktion mit TIMER verwenden.

Fehlerbehandlung (ON ERROR, RESUME, ERR, ERROR)

	Wenn ein Laufzeitfehler wie Überlauf oder "Datei nicht gefunden" auftritt, unterbricht Qbasic normalerweise die Ausführung des Programms und zeigt ein Dialogfeld mit der Fehlermeldung an. Da dies bei fertigen Programmen unerwünscht ist, kann man alle Laufzeitfehler im Programm abfangen und ggf. den Fehler beheben.	
164	ON ERROR GOTO Fehler	Wenn ein Fehler auftritt, wird zur Marke Fehler gesprungen. (Diese Zeile sollte in Qbasic am Anfang des Programms stehen)

165	Fehler: SELECT CASE ERR CASE 6 PRINT "Überlauf!" SYSTEM	Irgendwo im Programm steht dann die Marke Fehler, in der mit der Funktion ERR der Fehlercode des Laufzeitfehlers abgefragt werden kann. Die Fehlercodes findet man in der Qbasic-Hilfe, Inhalt, Laufzeitfehler-Codes.
166	CASE 58 PRINT "Datei existiert bereits!" RESUME NEXT END SELECT	RESUME NEXT setzt die Ausführung des Programms nach der fehlerhaften Anweisung fort. Ein RESUME ohne NEXT würde die Programmausführung bei der fehlerhaften Anweisung fortsetzen, also diese noch einmal ausführen. (Davor sollte der Fehler allerdings behoben werden)
167	ERROR 6	Löst Laufzeitfehler 6 aus (macht natürlich nur bei einer passenden Fehlerbehandlung einen Sinn, wenn man einen Laufzeitfehler simulieren möchte)

Farbcodes

168	0	ABC schwarz
169	1	ABC blau
170	2	ABC grün
171	3	ABC cyanblau
172	4	ABC rot
173	5	ABC magenta
174	6	ABC braun
175	7	ABC hellgrau
176	8	ABC grau
177	9	ABC hellblau
178	10	ABC hellgrün
179	11	ABC hellcyan
180	12	ABC hellrot
181	13	ABC pink
182	14	ABC gelb
183	15	weiß
	Zählt man 16 zu den Zahlen dazu, so ergeben sich blinkende Farben. (Nur für Vordergrund und Text)	