Lernskript

Basic256

2 mal 5 Befehle

Print
Input
Rand
For-Next
If-Then

} Text & Zahl

Color
Rect
Circle
While
Key

} 2D-Grafik

Titelseite: Vision einer Marsmission, NASA

Inhalt

Vorstellung

Basic256 ist eine vollwertige Programmiersprache. Sie wurde 2007 entwickelt. Basic256 läuft auf Linux, Windows und ist auch als App sowie als Cloud-Anwendung erhältlich. Das Programm wird normalerweise kostenlos angeboten.

Basic256 ist vor allem für Anfänger und Gelegenheitsprogrammierer geeignet. Mit wenigen Befehlen kann man schon recht gute Programme schreiben, zum Beispiel 2D-Spiele. Dazu reicht es, 10 Befehle zu kennen:

1. PRINT
2. INPUT
3. RAND
4. FOR-NEXT
5. IF-THEN

} Teil I – Text & Zahl

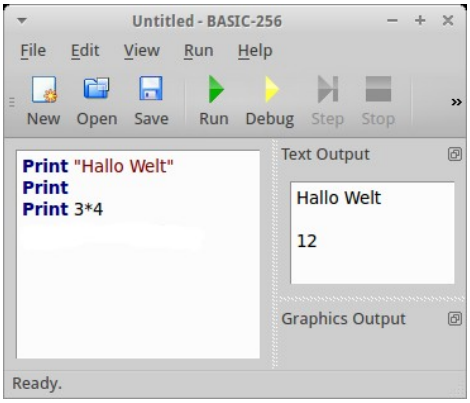
6. COLOR
7. RECT
8. CIRCLE
9. WHILE
10. KEY

} Teil II – 2D-Grafiken

Viele Schüler und auch Studenten aus Aachen und Umgebung haben mit diesem Skript einen Einstieg in Basic256 gefunden. Über Rückmeldungen und Verbesserungsvorschläge freuen wir uns jederzeit:

Internetseite: www.mathe-ac.de
Email-Adresse: heim@mathe-ac.de
Permalink: https://www.rhetos.de/html/lsk/basic256.pdf

Teil I – Text & Zahl



Print – Sachen ausgeben

Rechts siehst du die Basic256 Programmierumgebung. Programme werden in das weiße Feld links geschrieben. Tippe das kleine Programm mit den drei Zeilen links im Bild ab. Drücke dann oben das grüne Dreieck mit “Run”. Run startet Programme. Wenn du keine Tippfehler gemacht hast, erscheint in dem kleinen Feld rechts (Text Output) die Ausgabe. Sehen wir uns die drei Zeilen des Programms einzeln an:

Print “Hallo Welt”

Mit dem Wort Print erhält der Computer den Befehl, dass er etwas in die Textausgabe schreiben soll (weißes Feld oben rechts). Was der Computer dort hin schreiben soll, sagst steht zwischen den Anführungsstrichen (hier war es: Hallo Welt).

Print 3*4

Hier taucht der Printbefehl ohne Anführungszeichen auf. Dann weiß der Computer, dass er rechnen soll. Wenn er rechnet, schreibt er immer das Rechenergebnis in die Textausgabe. Der Stern * bedeutet in vielen Programmiersprachen “mal”. Der Computer rechnet also drei-mal-vier. Daher kommt die Zahl Zwölf in der Textausgabe.

Print

Print alleine sagt dem Computer, dass er einfach eine leere Zeile in die Textausgabe schreiben soll. Man macht das oft, um Ausgaben übersichtlicher zu gestalten.

Print – als Taschenrechner

Statt dem * für die Multiplikation kennt Basic256 noch eine Reihe weiterer Rechenzeichen:

12+4 gibt 16	12-4 gibt 8
12*4 gibt 48	12/4 gibt 3
3^4 gibt 81	(3 hoch 4)
Sqr(9) gibt 3	(Wurzel)
12%5 gibt 2	(Rest beim Teilen 12 durch 5)
Pi gibt 3.141593	(schreibt die Kreiszahl Pi hin)
Sin (pi) gibt 0	Asint (1) gibt 1.570796
Cos (pi) gibt -1	Acos (1) gibt 0
Tan (pi) gibt -0	Atan (1) gibt 0.785398
radians (180) gibt 3.141593	(Wandelt Grad in Rad um)
degrees (pi) gibt 180	(Wandelt Rad in Grad um)
Int (3.5) gib 3	(Rundet immer ab)
Abs (-7.5) gibt 7.5	(Lässt minus-Vorzeichen weg)
Exp(2) gibt 7.389056	Rechnet e hoch 2
Log(9) gibt 0.693147	Gibt den ln von 9
log10(100) gibt 2	Logarithmus zur Basis 10 von 100
Ceil (-5.3) gibt -5	Kleinste Ganzzahl, größer oder gleich
Floor (-5.3) gibt -6	Größte Ganzzahl, kleiner oder gleich
Rand gibt z. B. 0.35897	Gibt Zufallszahl von 0 bis 1 aus

Noch ein paar Dinge rund um's Rechnen:

→ Runde Klammern werden erkannt.
→ Kommazahlen gibt man mit Punkt ein.
→ Basic256 wendet Punkt-vor-Strich an.
→ Wenn eine Rechnung nicht geht, erscheint nan (not a number, wie “error” bei einem normalen Taschenrechner)

Jetzt üben wir die richtige Schreibweise für Rechenausdrücke. (Syntax). Links steht immer eine Rechenaufgabe. Nutze den Print-Befehl und die Rechenzeichen. Lasse Basic für dich rechnen. Rechts hinter dem senkrechten Strich steht zur Kontrolle das Rechenergebnis:

8+4-10	| 2
Die Hälfte von 10 erhöht um 7	| 12
Das Dreifache von 8 vermindert um ein Halb	| 23.5
Der Rest von 100 geteilt durch 80	| 20
25 zum Quadrat (also 25 hoch zwei)	| 625
2 hoch 10	| 1024
Die Wurzel aus 121	| 121
Die Wurzel aus -100	| nan
Der Rest aus 5-hoch-4 geteilt durch 301	| 23
Die Wurzel aus 0,25	| 0.5

Mit Print und den Rechenzeichen kann man jetzt Formeln programmieren.

Input – Gewitterformel

Wenn es blitzt fließt schnell starker Strom zwischen Wolken und Erde. Um den Strom herum erhitzt sich die Luft so schnell, dass sie sozusagen explodiert. Das gibt den Donnerknall. Das Licht ist fast sofort da. Der Donner kommt aber in einer Sekunde nur etwa 340 Meter weit. Daraus lässt sich eine Faustformel ableiten:

(Sekunden von Blitz zu Donner) : 3 = Abstand in Kilometern

Das Programm:

```
Print “Gib die Anzahl der Sekunden vom Blitz bis zum Donner ein.”
Input t
Print
Print “Ungefähre Entfernung in Kilometern:”;
Print t/3
```

Starte das Programm über das grüne Run-Dreieck. Hier die Erklärung: Der erste Printbefehl sagt dem Benutzer, was er eingeben soll. Dann kommt der Inputbefehl. Er lässt den senkrechten Strich im Ausgabefeld blinken. Der Strich heißt Cursor. Wenn er blinkt, soll man an dieser Stelle etwas eingeben. Gehe mit der Maus dorthin, tippe die 9 ein und drücke “Enter”. In der dritten Zeile kommt ein Printbefehl ohne alles. Er druckt einfach eine Leerzeile. In der vierten Zeile schreibt der Computer, wie man das Ergebnis verstehen soll. Das Semikolon (;) am Ende sagt dem Computer, dass der nächste Print-Befehl in derselben Zeile weiterschreiben soll. In der letzten Zeile rechnet er die Formel aus und schreibt das Ergebnis auf. Hier sind noch drei Testdatensätze:

Man zählt 0 Sekunden – Entfernung?	0 km
Man zählt 1 Sekunde – Entfernung?	0.333333 km
Man zählt 15 Sekunden – Entfernung?	5 km

Input – Vollbremsung

Wenn man bei einem Auto mit voller Kraft auf die Bremse tritt, dann führt das Auto eine sogenannte Vollbremsung aus. Es hält dabei so schnell wie möglich an. Die ungefähre Strecke, die es dann bis zum Stillstand braucht, nennt man den Bremsweg. Man kann den ungefähren Bremsweg mit einer Faustformel berechnen:

Nimm die Geschwindigkeit in km/h

Teile diese Zahl durch 10.

Multipliziere das Ergebnis mit sich selbst.

Teile diese Zahl durch 2.

Das Ergebnis ist der ungefähre Bremsweg in Metern.

Schreibe nun ein Programm, bei dem man die Geschwindigkeit des Auto eingibt und der Bremsweg in Metern ausgegeben wird.

Unten sieht du Geschwindigkeiten mit den richtigen Bremswegen. Mit diesen Testdaten kannst du überprüfen, ob dein Programm richtig arbeitet:

0 km/h	| 0 m
10 km/h	| 0.5 m
20 km/h	| 2 m
40 km/h	| 8 m
50 km/h	| 12.5 m
100 km/h	| 50 m
200 km/h	| 200 m

Speichere dein Programm mit “Save” aus der Menüleiste oben. Ein guter Name wäre “deinvorname_vollbremsung”.

Input – Tsunamiformel

Tsunamis - so nennt man sehr große Wellen. Auf dem Ozean sind sie schnell, flach und ungefährlich. An der Küste türmen sie sich dann zu großen Wasserbergen auf. Werden die Küstenbewohner nicht rechtzeitig gewarnt, kommt es zu Katastrophen.

Für eine gute Vorhersage ist es wichtig, die Geschwindigkeit eines Tsunamis zu kennen. Dabei gilt: je flacher das Wasser, desto langsamer (und höher!) wird er. So wird die Geschwindigkeit berechnet:

Nimm die Wassertiefe in Metern.

Multipliziere sie mit 10.

Ziehe daraus die Wurzel (sqrt).

Multipliziere das Ergebnis mit 3,6.

Das Ergebnis ist die Geschwindigkeit in km/h.

Schreibe ein Programm, in das man die Wassertiefe in Metern eingibt. Es soll dann die Tsunamigeschwindigkeit in Kilometern pro Stunden (km/h) abgerundet auf ganze Zahlen (Int) ausgeben. Hier sind wieder ein paar Testdaten:

10000 m Wassertiefe	| 1138 km/h
5000 m Wassertiefe	| 804 km/h
2000 m Wassertiefe	| 509 km/h
100 m Wassertiefe	| 113 km/h
10 m Wassertiefe	| 36 km/h

For-Next-Step – Quadrate

Jede Zahl hat ein sogenanntes “Quadrat”. Das Quadrat einer Zahl r ist das Ergebnis von r mal r. Beispiel: 3 mal 3 ist 9. Also ist die 9 das Quadrat von der 3. Noch ein Beispiel: 11 mal 11 ist 121. Also ist 121 das Quadrat von der 11. Hier ist ein Programm, dass uns die Quadrate für die natürlichen Zahlen von 0 bis 10 aufschreibt:

```
For z = 0 to 10
print z;
print “ Quadrat ist “;
print z*z
next z
```

for z = 1 to 10 sagt dem Computer, dass er für den Buchstaben z der Reihe nach die Zahlen von 0 bis 10 einsetzen soll und bis zum Befehl next z mit dieser Zahl rechnet. Solch einen Platzhalter wie z nennt man beim Programmieren eine Laufvariable.

Print z; sagt dem Computer, dass er den momentanen Wert der Laufvariable anzeigen soll. Das Semikolon am Ende sagt dem Computer, dass er beim nächsten Print-Befehl in der gleichen Zeile weiterschreiben soll (also keine neue Zeile anfängt).

Print “ Quadrat ist “; sagt dem Computer, dass er genau das hinschreiben soll, was zwischen den Anführungszeichen steht. Das Semikolon am Ende sagt wieder, dass die Zeile noch nicht fertig ist (es also noch weiter geht).

Print z*z lässt den Computer das Quadrat der momentanen Zahl von z ausrechnen und hinschreiben. Danach kommt kein Semikolon. Das heißt, die Ausgabezeile ist damit zu Ende.

Next z sagt dem Computer, dass die Berechnung für den ersten Wert von z (das war die 0) fertig ist. Er geht jetzt wieder an den Anfang der Schleife und geht alles für den nächsten Wert von z durch (also die 1). Das macht er so lange, bis er z=10 erreicht hat. Dann hört er auf.

For-Next-Step – Wurzeln

Das Gegenteil von Quadratzahlen sind Wurzeln. Die Wurzel einer Zahl r ist die Zahl, die mit sich selbst malgenommen wieder r ergibt. 4 mal 4 ist 16. Also ist 4 die Wurzel von 16. Wurzeln gibt es auch für Kommazahlen: 1,5 mal 1,5 ist 2,25. Also ist 1,5 die Wurzel von 2,25.

Das nächste Programm schreibt für ganze Zehnerzahlen (0, 10, 20, 30 und so weiter) bis zur 100 die Wurzeln auf. Der For-next-Befehl wird dazu um das Wort “Step” erweitert. Step sagt dem Computer, dass er die Laufvariable nicht immer nur ein größer machen soll. “Step” heißt auf Deutsch “Schritt”. Mit Step kann man ihm sagen, um wieviel die Variable immer erhöht werden soll:

```
For z = 0 to 100 Step 10
print z;
print “ Wurzel ist “;
print sqrt(z)
next z
```

Und so sollte das Ergebnis im Ausgabefeld aussehen:

```
0 Wurzel ist 0
10 Wurzel ist 3.162278
20 Wurzel ist 4.472136
30 Wurzel ist 5.477226
40 Wurzel ist 6.324555
50 Wurzel ist 7.071068
60 Wurzel ist 7.745967
70 Wurzel ist 8.3666
80 Wurzel ist 8.944272
90 Wurzel ist 9.486833
100 Wurzel ist 10
```

For-Next-Step – Rumpfgeschwindigkeiten

Schiffe, die beim Fahren mit ihrem Rumpf im Wasser bleiben heißen “Verdränger”. Die meisten normalen Schiffe wie Fähren, Tanker, Kreuzfahrtschiffe oder auch Tretboote sind Verdränger. Keine Verdränger sind zum Beispiel Tragflächenboote, Schnellboote oder Hovercrafts. Die maximal mögliche Geschwindigkeit v in km/h eines Verdrängers heißt Rumpfgeschwindigkeit. Die Formel dazu ist einfach:

v = 4,5 mal [Wurzel aus der Schiffslänge in Metern]

Beispiel: Wenn ein Schiff 100 Meter lang ist rechnet man: Wurzel aus Hundert ist Zehn. 4,5 mal 10 ist 45. Das Schiff kann also maximal eine Geschwindigkeit von 45 Kilometern pro Stunde erreichen. Mehr ist aus physikalischen Gründen nicht möglich, auch wenn man den Antrieb immer stärker macht.

Erstelle ein Programm, das für Schiffslängen von 0 bis 300 Metern in 20 Metern-Schritten die maximale Geschwindigkeit in km/h berechnet. Nachkommastellen sollen wieder mit INT abgeschnitten sein.

Hier sind Testdaten, mit denen du dein Programm stichprobenartig überprüfen kannst:

0m Schiffslänge	0 km/h
100m Schiffslänge	45 km/h
200m Schiffslänge	63 km/h
300m Schiffslänge	77 km/h

In Wirklichkeit fahren Schiffe oft viel langsamer. Sie tun dies um Treibstoff zu sparen. Die meisten großen Schiffe fahren um die 40 km/h schnell. Bei dieser Geschwindigkeit verbrauchen sie am wenigsten Treibstoff.

Rand - Zufallsgenerator

Zufall nennen wir oft das, was wir nicht voraussagen können. Und genau das erst macht viele Spiele interessant. Viele Programmier-sprachen haben einen sogenannten Zufallsgenerator. Ein Generator ist ein “Erzeuger”. Ein Zufallsgenerator ist so etwas wie ein digitaler Würfel. Bei Basic256 lautet der entsprechende Befehl Rand. Rand kommt vom englischen Wort Random und meint einfach Zufall, Unbestimmtheit. So funktioniert Rand:

```
Print Rand
```

Dieses einzeilige Programm schreibt eine zufällige Zahl zwischen 0,000001 und 0,999999. So benutzen wir RAND, um Zahlen von 1 bis 6 zu erzeugen:

```
Print INT (6*Rand) + 1
```

Erst produziert Rand eine Zufallszahl von 0 bis 1, z. B.	0,346778
Dann wird diese Zahl mal sechs genommen:	2.080668
Dann wird abgerundet auf Einer:	2
Dann wird noch eins dazuaddiert:	3

Die Addition von 1 am Ende ist wichtig. Ohne die Erhöhung um 1 kämen die Zahlen von 0 bis 5 zustande, weil ja immer abgerundet wird. Mit Rand können wir jetzt jederzeit genau passende Zufallszahlen erzeugen.

Ändere das Würfelprogramm so ab, dass es Zufallszahlen von 1 bis 10 erzeugt.

If-Then – Zahlenvergleich

Den If-Befehl gibt es in fast jeder Programmiersprache. If ist Englisch und heißt auf Deutsch “Falls”. Hier ein Beispielprogramm, wie der Befehl funktioniert:

```
Print “Gib irgendeine Zahl bis 10 ein.”
Input z
If z>10 then
print “Die Zahl ist zu groß”
End if
```

Die Erklärung

Der erste Print-Befehl sagt, was eingegeben werden soll.

Der Inputbefehl fragt nach einer Zahl und nennt sie z.

Die Zeile mit If überprüft, ob z größer ist als 10.

Falls z größer ist als 10 gibt es eine Warnung.

Das End if schließt den If-Befehl ab.

Hier sind alle Möglichkeiten, wie man mit Basic256 zwei Zahlen vergleichen kann (Vergleichsoperatoren).

=	gleich
<	kleiner als
>	größer als
<=	kleiner oder gleich
>=	größer oder gleich
<>	ungleich

If-Then – Wissensquiz

Schreibe ein kleines Frage-Antwort-Programm. Hier ein Vorschlag, was das Programm machen könnte:

Wie viele Meter hoch ist der höchste Berg der Welt?
Falls Antwort kleiner 8848 → “Das ist zu wenig.”
Falls Antwort gleich 8848 → “Stimmt, sehr gut!”
Falls Antwort größer 8848 → “Das ist zu hoch.”

Leerzeile

Wie viele Beine hat eine Krabbe?
Falls Antwort kleiner 8 → “Nein, sie hat mehr.”
Falls Antwort gleich 8 → “Ja, Krabben haben acht Beine.”
Falls Antwort größer 8 → “Nein, so viele sind es nicht.”

Leerzeile

Wie viele Tage hat ein normales Jahr
Falls Antwort kleiner 365 → “Es sind mehr.”
Falls Antwort gleich 365 → “Richtig. Ein normales Jahr hat 365 Tage.”
Falls Antwort gleich 366 → “Nur Schaltjahre haben 366 Tage.”
Falls Antwort größer 366 → “Das sind zu viele.”

If-Then – Einspluseins

Wir schreiben jetzt einen kleinen Kopfrechentrainer für Erstklässler. Das Programm "denkt" sich selbst eine kleine Plusaufgabe aus und sagt, ob die Antwort stimmt.

Das Programm	Kurzerklärung
a = Int(10*Rand)	a = Zufallszahl von 0 bis 9
b = Int(10*Rand)	b = Zufallszahl von 0 bis 9
print "Was gibt ";	
print a;	Schreibt so etwas wie...
print " + ";	... Was gibt 4 + 3 ?
print b;	
print " ?"	
print	
input z	
if z=a+b then	Ist dieAntwort richtig?
print "richtig!"	Falls ja →
else	Ansosten ...
print "leider nicht richtig"	Falls nein
end if	Ende

Im oberen Programmteil ist das Semikolon ; wieder wichtig. Es sorgt dafür, dass die Rechenaufgabe in eine Zeile ausgedruckt wird. Im unteren Programmteil ist das wichtige Wort das "Else". Der If-Befehle wird damit erweitert:

if ... then	falls ... dann
...	was dann passieren soll
else	ansonsten
...	was ansonsten passieren soll
end if	Ende des If-Konstrukts

Programmideen

Die folgenden Programme unten kann man alleine mit den bisher gelernten Befehlen umsetzen:

- Man gibt zwei Zahlen ein, der Computer sagt, welche die größere ist
- Man gibt drei Zahlen ein, der Computer sagt, welche die größte ist
- Man gibt fünf Zahlen ein, der Computer sortiert sie aufsteigend
- Dreizeilenprogramm schreibt Zahlen von 1000 rückwärts bis 1
- Der Computer löst die Gleichung 2493 = 3x+297 durch Probieren
- Man gibt eine Strecke in Seemeilen ein und der Computer rechnet sie in Kilometer um
- Man gibt eine Strecke in Kilometern ein und der Computer rechnet sie in Seemeilen um
- Man gibt zwei Zahlen ein und der Computer findet ihre Mitte
- Man gibt eine Zahl ein und der Computer teilt sie im Verhältnis 2 zu 1 auf
- Der Computer erzeugt eine Umwandlungstabelle für Grad Celsius und Grad Fahrenheit
- Mister X: Der Comptuter denkt sich eine ganze Zufallszahl zwischen 0 und 100 aus. Der Spieler darf immer eine Zahl sagen (also eingeben). Der Computer sagt dann, ob die geratene Zahl zu groß, zu klein oder richtig war. Er zählt die Versuche, die man bis zum Erraten der richtigen Zahl brauchte und gibt sie am Ende aus.
- Einmaleinstrainer: Der Computer denkt sich Aufgaben zum Einmaleins aus. Der Spieler nennt die Lösung und der Computer sagt, ob sie richtig war oder nicht.

Zu den Programmideen gibt es keine Lösungen.

Zwischenresümee und Lösungen

Die bis jetzt gelernten Befehle reichen aus, um die Flugbahn einer Rakete zum Mars zu berechnen, die Zinseszinsen für ein Konto für 100 Jahre im Voraus zu ermitteln, Rechentrainer für alle möglichen Kopfrechenaufgaben zu schreiben und und und.

Im online-Lexikon von Mathe-AC findest du beispielhafte Lösungen zu allen besprochenen Programmen:

=> Basic256 Programme Gewitterformel

=> Basic256 Programme Vollbremsung

=> Basic256 Programme Tsunamiformel

=> Basic256 Programme Rumpfgeschwindigkeiten

=> Basic256 Programme Wissensquiz

=> Basic256 Programme Einspluseins

Sonstiges

- Die Raute # in Programmen steht am Anfang von Kommentarzeilen. Kommentarzeilen findet man in Programmen oft. Es sind sozusagen Notizzettel für den Programmier.
- a=4 nennt man beim Programmieren eine Zuweisung. Der Computer merkt sich, dass die Variable a den Wert 4 haben soll.
- Der Befehl End beendet ein Programm sauber. Man sollte am Ende immer den End-Befehl setzen.
- Mit dem Befehl Cls kann man die Textausgabe komplett löschen.

Im zweiten Teil dieses Tutorials geht es jetzt um kleine 2D-Grafiken.

Teil II – 2D-Grafikprogramme

Color – Rect - Circle

Tippe das folgende Programm ein und starte es mit Run:

```
Graphsize 400,200
Color white
Rect 0,0,400,200
Color yellow
Circle 200,100,10
```

Erklärung

Unterhalb der Textausgabe liegt die Grafikausgabe (Graphics Output). Um diese geht es jetzt. Mit Graphsize sagt man, wieviele Pixel breit und hoch die Grafikausgabe sein soll.

Denke dir die Graphikausgabe wie ein umgedrehtes Koordinaten-system aus der Schule: ganz oben links liegt der Punkt (0|0). Die x-Werte werden dann nach rechts größer, die y-Werte werden nach unten größer.

Color legt die Farbe fest, die ab jetzt benutzt werden soll. Das Rechteck, was gleich gezeichnet wird, wird also weiß werden. Damit haben wir jetzt unsere ganze Graphikausgabe weiß gemacht (normalerweise ist sie grau).

Rect zeichnet ein Rechteck. Die erste Zahl ist der x-Wert der linken oberen Ecke. Die zweite Zahl ist der y-Wert der linken oberen Ecke. Die dritte Zahl ist die Breite des Rechtecks in Pixeln. Die vierte Zahl ist die Höhe des Rechtecks in Pixeln.

Der nächste Color-Befehle bewirkt, dass der Kreis mit Circle gleich gelb wird. Die erste Zahl nach dem Circle-Befehl ist der x-Wert des Kreismittelpunktes. Die zweite Zahl ist der y-Wert des Kreismittel-punktes. Die dritte Zahl ist der Radius des Kreises.

Color – Farben und RGB

Mit Basic256 kannst du die folgenden Farben benutzen. Gib immer das englische Wort ein. Wie fast alle Programmiersprachen, ist auch Basic256 ganz auf Englisch geschrieben.

white (weiß)	clear (wie Radiergummi)
black (schwarz)	---
red (rot)	darkred (dunkelrot)
green (grün)	darkgreen (dunkelgrün)
blue (blau)	darkblue (dunkelblau)
cyan (ähnlich türkis)	darkcyan (dunkeltürkis)
purple (violett)	darkpurple (dunkelviolett)
yellow (gelb)	darkyellow (fast bräunlich)
orange (orange)	darkorange (auch bräunlich)
gray (grau)	darkgray (dunkelgrau)

Es gibt noch eine Möglichkeit, die Farben anzugeben: An einem Bildschirm werden alle Farben aus den Grundfarben Rot, Grün und Blau gemischt, den sogenannten RGB-Farben. 0 heißt, dass es die Farbe gar nicht gibt, 255 ist ihre maximale Stärke:

Color 255,0,0 → gibt Knallrot

Color 0,255,0 → gibt grelles Grün

Color 0,0,255 → gibt sattes Blau

Erste Zahl nach Color: rot-Anteil

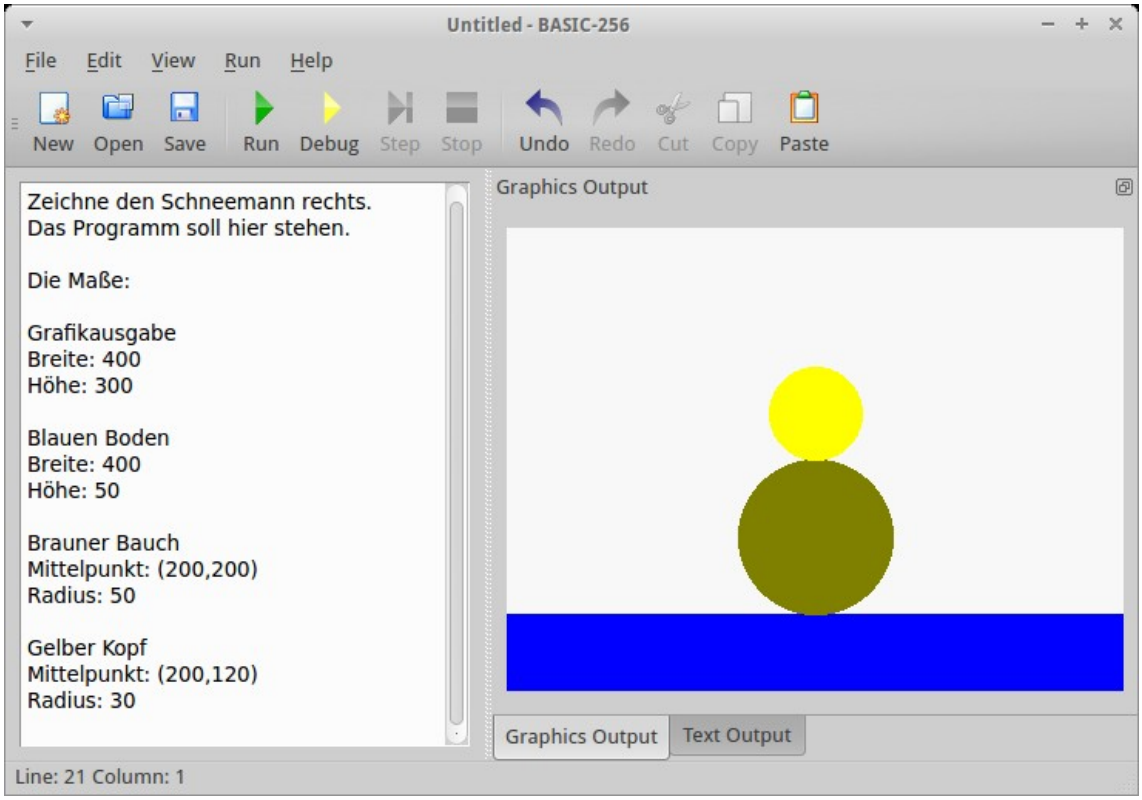
Zweite Zahl nach Color: grün-Anteil

Dritte Zahl nach Color: blau-Anteil

- Probiere aus: color 100,100,100
- Kannst du dir gelb mischen?

Color – Rect – Circle - Schneemann

Mit den Grafikbefehlen von der vorherigen Seite kannst du jetzt einen Schneeman zeichnen:



Tipp: Zeichne dir vielleicht erst das weiße Grafikfeld auf ein Blatt Papier. Schreibe die wichtigen Koordinaten und Radien auf das Papier. Mit so einem Plan lassen sich Grafiken sehr gut planen.

While – Pinselstrich

Mit dem While-Befehl kriegen wir Bewegung in die Graphik. Wir lassen jetzt einen kleinen Pinsel von links nach rechts über das Grafikfenster (Grafikausgabe) wandern. Das geht so:

```
graphsize 400,300
x=30
while x<370
circle x,200,5
x=x+1
end while
```

Erklärung

In dem Circle-Befehl ist der x-Wert des Kreismittelpunktes jetzt nicht mehr fest. Stattdessen steht dort der Buchstabe x. Dieser Variablen x können wir nun während des Programmes dauernd einen anderen Wert geben (variabel heißt veränderlich). Gehen wir das Programm zeilenweise durch:

graphsize 400,300	Bildschirmgröße (diesmal nur grau)
x=30	Am Anfang ist die Kreismitte bei x=30
while x<370	Hier beginnt die Schleife (mit x=30)
circle x,200,5	Solange x kleiner 370 ist: Kreis zeichnen
x=x+1	Macht x eins größer (x ist jetzt also 31)
end while	Geht solange zum While-Befehl zurück, wie x noch kleiner ist als 370.

While & Clg – Tennisball

In dem nächsten Programm machen wir aus dem Pinselstrich einen fliegenden Ball:

```
graphsize 400,300
x=30
while x<370
clg
circle x,200,5
x=x+1
end while
```

In der vierten Zeile steht jetzt der Befehl Clg. Das ist die Abkürzung für Clear Graph und heißt auf Deutsch: Lösche alles in der Grafikausgabe. Der alte Ball wird also weggewischt, bevor der neue Ball eins weiter rechts (x=x+1) gezeichnet wird. Dadurch entsteht der Eindruck, dass ein Ball von links nach rechts über den Bildschirm wandert.

Das Flackern kriegt man bei Grafikprogrammen mit den Befehlen fastgraphics und refresh weg. Das macht man so, dass man ganz am Anfang eines Grafikprogrammes fastgraphics setzt. Innerhalb einer Schleife schreibt man dann ganz am Ende, wenn alle Zeichnungen für die aktuelle Runde fertig sind das Wort refresh. So sieht dann das flackerfreie Programm aus:

```
graphsize 400,300
x=30
while x<370
clg
circle x,200,5
x=x+1
refresh
end while
```

Key – Pinselsteuerung

Jetzt kommen wir zum letzten Befehl unseres Tutorials. Mit dem Key-Befehl kann man Grafiken mit Hilfe der Tastatur steuern. Das Programm unten lässt einen orangenen Pinsel wandern. Man steuert den Pinsel mit den WASD-Tasten (oft so bei alten Spielen):

W → nach oben
A → nach links
S → nach unten
D → nach rechts

Um das Programm zu beenden drückt man auf die rote Taste “Stop” ganz oben im Benutzermenü von Basic256.

```
graphsize 400,300
color orange
x=0
y=0
taste=0
while taste <> 80
taste=key
if taste=87 then y=y-1
if taste=65 then x=x-1
if taste=83 then y=y+1
if taste=68 then x=x+1
circle x,y,5
end while
print "Bis zum nächsten Mal!"
end
```

In die Variable “taste” wird über Key der Code der aktuell gedrückten Taste eingegeben. Darüber kann der Pinsel bewegt werden. Auf der nächsten Seite findest du die Codes für die Tasten.

Key - Tastaturwerte

Basic256 Tastaturwerte

Down Arrow=16777237
Up Arrow=16777235
Left Arrow=16777234
Right Arrow=16777236
ESC=16777216
Space (Leertaste)=32

0=48	A=65	K=75	U=85
1=49	B=66	L=76	V=86
2=50	C=67	M=77	W=87
3=51	D=68	N=78	X=88
4=52	E=69	O=79	Y=89
5=53	F=70	P=80	Z=90
6=54	G=71	Q=81	
7=55	H=72	R=82	
8=56	I=73	S=83	
9=57	J=74	T=84	

Programmideen

Mit den hier vorgestellten Grafik-Befehlen lassen sich richtige kleine 2D-Spiele erstellen. Ab hier ist alles nur Phantasie und Probieren. Gut ist es, wenn man jemanden kennt, den man ab und zu fragen kann. Hier sind noch einige Idee für Programme, nach unten hin werden sie schwieriger:

- Strich von unten links nach oben rechts
- Lasse eine Programm einen Roboter zeichnen
- Lasse ein Programm einen Schneemann zeichnen
- Lasse ein Programm ein gleichseitiges Dreieck zeichnen
- Ein Kreis wächst auf dem Bildschirm
- Mit der Tastatur verändert man die Größe eines Kreises

- Ein Ball fliegt zwischen linken und rechtem Rand hin und her.
- Ein Schneemann fliegt von links nach rechts über den Schirm
- Vom linken Bildschirmrand kommen auf zufälliger Höhe Bälle nach rechts geflogen. Man muss sie rechts mit einem Schläger treffen, den man über die Tastatur hoch und runter bewegen kann.
- Wer Trigonometrie von Mathe her kennt: lasse einen Ball auf einer Kreisbahn fliegen.

Dateinamen

Du kannst deine Programme mit “Speichern unter” oder “Save as” (ist das Gleiche) speichern. Nun gibt es für Computer verschiedene Betriebssysteme:

→ PCs haben oft Windows.
→ Server und manche PC haben Linux.
→ Mobile Geräte haben oft Android.

Mit einigen Zeichen gehen die Betriebssysteme unterschiedlich um. Dazu gehören Sonderzeichen wie ! %, & und vor allem das Leerzeichen. Aber auch Großbuchstaben, das Eszet (ß) und die deutschen Umlaute ä, ö und ü. Es ist gut, diese Zeichen in Dateinamen zu meiden. Hier nun einige Tipps:

- Verwende nur Kleinbuchstaben
  Gut: schneeball.kbs
  Schlecht: Schneeball.kbs
- Tiefstrich statt Leerzeichen
  Gut: zahlen_raten.kbs
  Schlecht: statt zahlen raten.kbs
- Keine Umlaute, kein Eszet
  Gut: masszahl.kbs
  Schlecht: maßzahl.kbs
- Keine Sonderzeichen
  Gut: meteorschauer.kbs
  Schlecht: meteor-schauer.kbs
- Kurze Namen
  Gut: pingpong.kbs
  Schlecht: ball_fliegt_hin_und_her.kbs
- Sprechende Namen
  Gut: mitte_von_zwei_zahlen.kbs
  Schlecht: mvzz.kbs

Ressourcen

Ressourcen nennt man beim Programmieren alles, was irgendwie hilft. Für Basic256 gibt es Dokumentationen und Beispielprogramme auf:

=> Homepage von Basic256

=> Basic-256 auf Wikipedia

=> In einem Schülerlexikon

Download

Die jeweils aktuellste Version dieses Lexikons befindet sich auf dem Rhetos Lernlexikon der Mathe-AC Lernwerkstatt in Aachen:

Download

Copyright

Hinweise & Tipps:
Any suggestions:

Mathe-AC Lernwerkstatt Aachen
Sabine & Gunter Heim
Wilhelmstraße 54
52070 Aachen
Germany
2021

= Schluss =