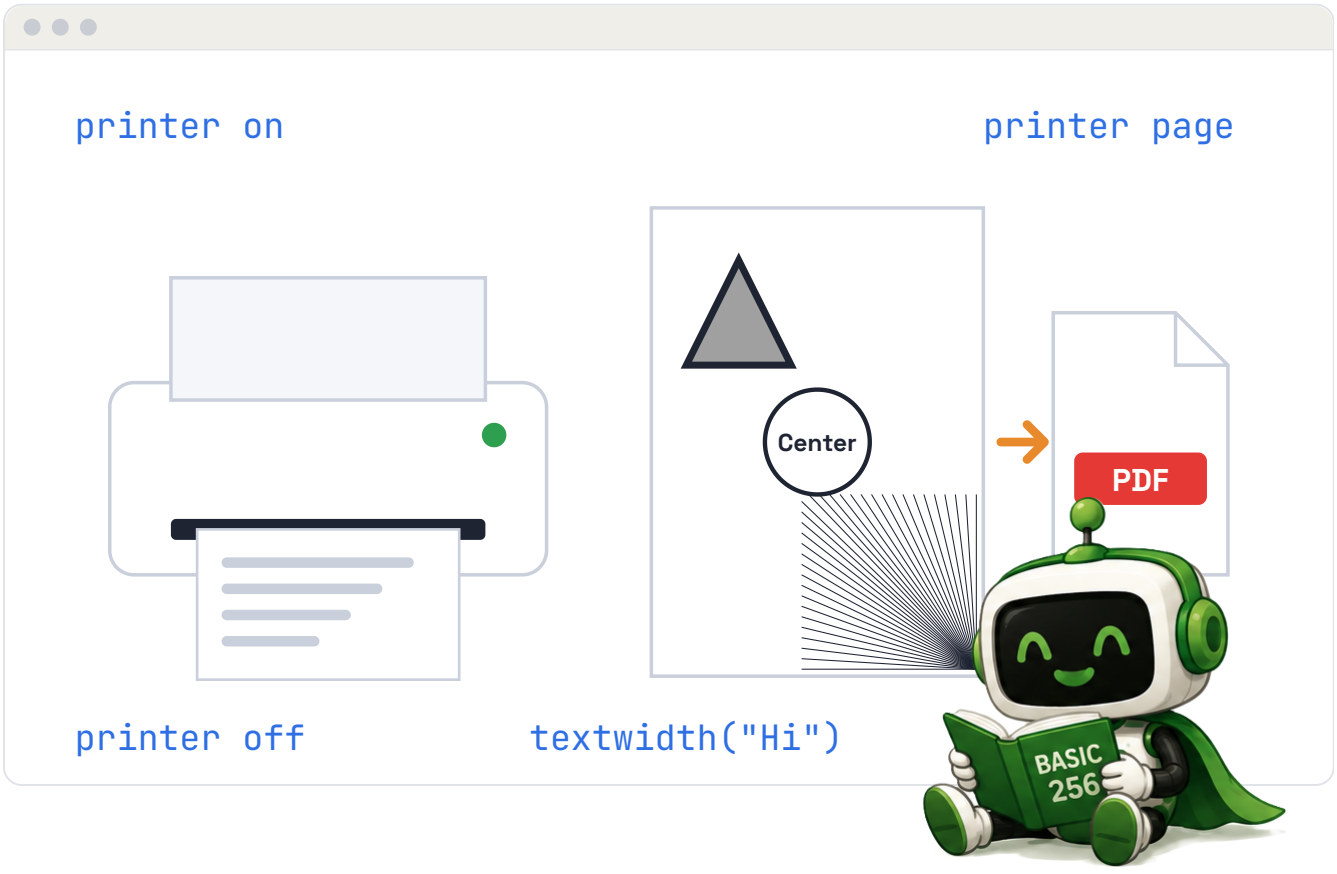
LEARN TO CODE · PAGES AND PDF FILES

Chapter 14

Printing

Everything your programs have drawn so far stayed on the screen. In this chapter they draw on paper. A printed page works like a big graphics output: you draw on it with the same **text**, **line**, **rect**, **circle**, **poly** and **stamp** statements you already know. The pages can go to a printer, or into a PDF file that you can open, send by email, or print later. At the end you print a complete multiplication table.



© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 1

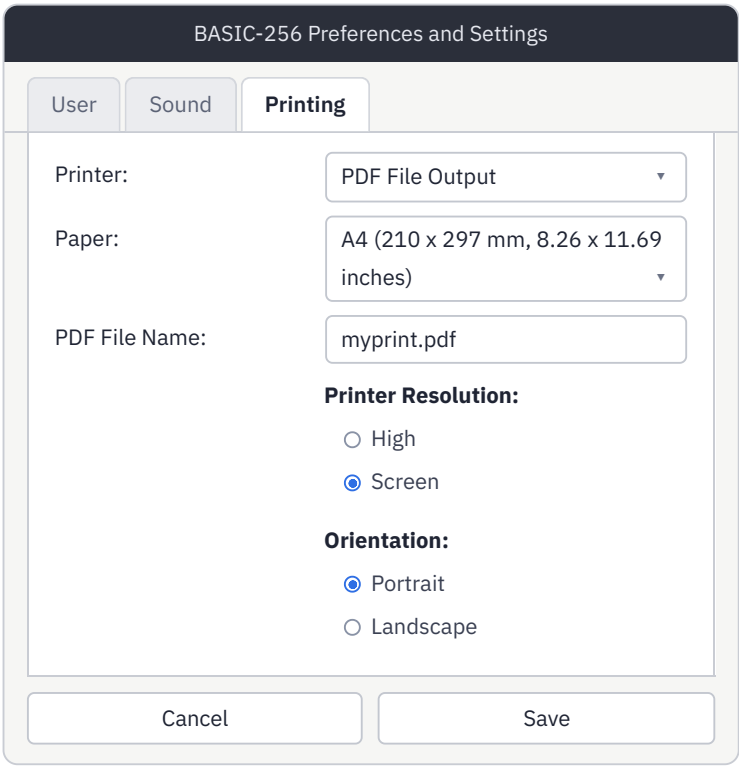
Where do the pages go?

Before you print, tell BASIC-256 where the pages go: to a printer or to a PDF file.

A program doesn't choose the printer itself. You choose it once, in BASIC-256's preferences, and every program that prints uses that choice. Choose **Edit › Preferences...** and click the **Printing** tab.

The Printing tab, set up for this chapter: the pages go into the PDF file *myprint.pdf*, on A4 paper, in Screen resolution, upright.

All the pages in this chapter were printed into PDF files on A4 paper, in Screen resolution. The programs measure the page as they run, so they work on Letter paper, or any other size, as well.



SETTING	WHAT IT CHOOSES
Printer	a printer connected to your computer, or **PDF File Output** to save the pages in a PDF file instead. Choose PDF File Output to try the programs in this chapter: it costs no paper.
Paper	the size of the paper, for example A4 or Letter.
PDF File Name	the name of the PDF file, for example *myprint.pdf*. A name without a folder is saved in the folder of your program. Each run replaces the file of the run before.
Printer Resolution	**Screen** (about 96 pixels per inch) or **High** (the printer's own resolution: 1200 pixels per inch for a PDF file). More about this in Section 3.
Orientation	**Portrait** (upright, taller than wide) or **Landscape** (turned on its side, wider than tall).

Set it up before you run the programs. When you have never changed these settings, BASIC-256 uses your computer's standard printer, so the programs in this chapter would print on real paper straight away. Choose **PDF File Output**, check that **Paper** is the size you use (A4 or Letter), and type a name in **PDF File Name**: if that box is empty, no file is saved at all. Then click **Save**.
After a run, open the PDF file from your program's folder to see the page. Close it again before the next run: while a PDF viewer holds the file open, BASIC-256 can't replace it, and **printer on** stops the program with the error *Unable to open printer*.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 2

Turning printing on and off

`printer on` starts a new page, and `printer off` finishes the printing.

To print, a program turns the printer on with **printer on**. From then on the drawing statements draw on the page instead of the graphics output. When the program has drawn everything, **printer off** finishes: the page goes to the printer, or is saved in the PDF file.

Program 93 writes ten lines of text on a page. The variable *y* holds how far down the page the next line goes. After each line, **textheight** tells the program how high a line of text is, and *y* goes down by that much.

printpage.kbs BASIC-256

```
# printpage.kbs - print a page with text

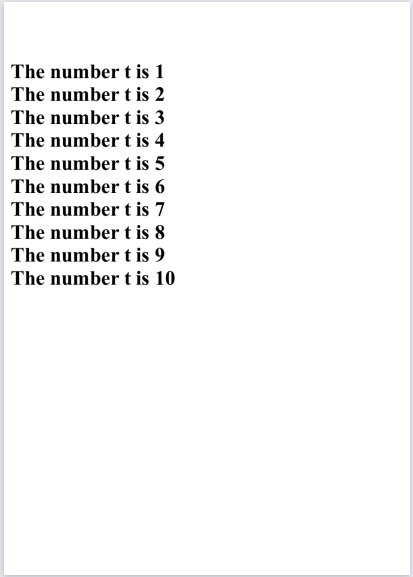
printer on

font "Times New Roman", 30, 700
y = 100      # first line: 100 pixels down
for t = 1 to 10
   text 0, y, "The number t is " + t
   y += textheight()
next t

printer off
print "The page is printed."
```

Program 93 · Printing a Page with Text

PDF · myprint.pdf



The page Program 93 printed.

TEXT OUTPUT

```
The page is printed.
```



NEW CONCEPT

`printer on`

printer on starts a new page. Until **printer off**, all drawing goes to the page: **plot**, **line**, **rect**, **circle**, **arc**, **poly**, **stamp**, **text**, **imgload** and **clg**. And **graphwidth**, **graphheight**, **textwidth** and **textheight** measure the page instead of the graphics output. The graphics output on the screen stays as it was. The color, fill color, pen width and font you chose before stay in use on the page too.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout



NEW CONCEPT

```
printer off
```

printer off finishes the printing. With a printer, the pages are sent to it and it starts printing. With PDF File Output, the pages are saved in the PDF file.

print or printer? `print` (Chapter 1) writes in the text output, also while the printer is on: the last line of Program 93 tells you in the text output that the page is done. Only `printer` has to do with paper. Turning the printer on twice, or turning it off when it isn't on, stops the program with an error: *Printing is already on* or *Printer is not on*. *Unable to open printer* means the printer, or the PDF file, can't be used.

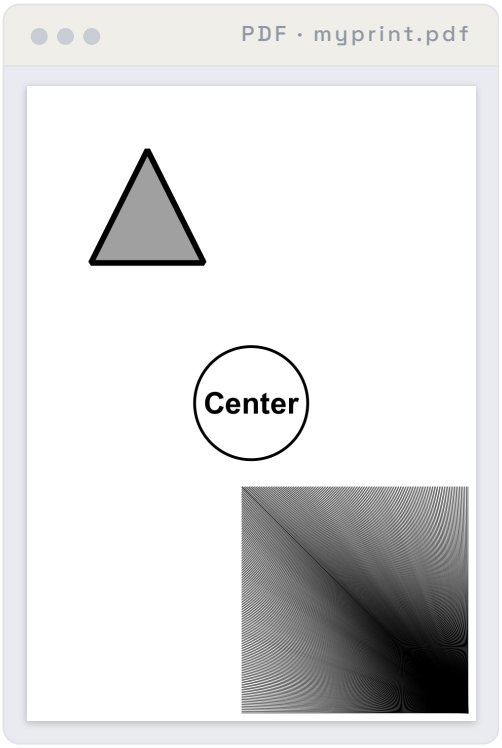
© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 3

Drawing on the page

Use the size of the page and the size of your text to put things exactly where you want them.

Program 94 draws on the page with the statements from Chapter 9: the word *Center* in the middle of the page with a circle around it, a gray triangle, and a **moiré** pattern of thin lines in the bottom right corner. A moiré pattern shows rings that nobody drew: they appear where the lines lie close together.



The word is in the middle of the page, whatever its size.

drawpage.kbs BASIC-256

```
# drawpage.kbs - draw on the page

printer on
w = graphwidth     # the size of the page
h = graphheight

# put the word in the center of the page
color black
font "Arial", 40, 700
words = "Center"
x = (w - textwidth(words)) / 2
y = (h - textheight()) / 2
text x, y, words

# draw a circle around the word, filled with clear
color black, clear
penwidth 5
circle w / 2, h / 2, 100
```

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

```
# draw a triangle with poly
color black, grey
penwidth 10
poly {200, 100, 300, 300, 100, 300}

# draw a moiré pattern in the bottom right corner
color black
penwidth 1
for t = 0 to 400 step 3
   line w, h, w - 400, h - t
   line w, h, w - t, h - 400
next t

printer off
print "The page is printed."
```

Program 94 · Printing a Page with Graphics

Right after **printer on**, the program saves the size of the page in *w* and *h*. That keeps the long lines short. It also matters where you do it: before **printer on**, **graphwidth** would still give you the width of the graphics output, 500.

To put the word in the middle, the program needs to know how big it is. **textwidth** gives the width of the word in the current font. Take that away from the width of the page and half of what is left goes on each side: that is *x*. In the same way, *y* uses **textheight** and the height of the page. **text** places the top left corner of the word at *x*, *y*.



NEW CONCEPT

```
textwidth(string)
textheight()
```

textwidth returns how many pixels wide a string is when **text** draws it in the current font. **textheight** returns the height of a line of text in the current font, the same for every string. Both measure on the page when the printer is on, and on the graphics output when it isn't. Choose the font first, then measure.

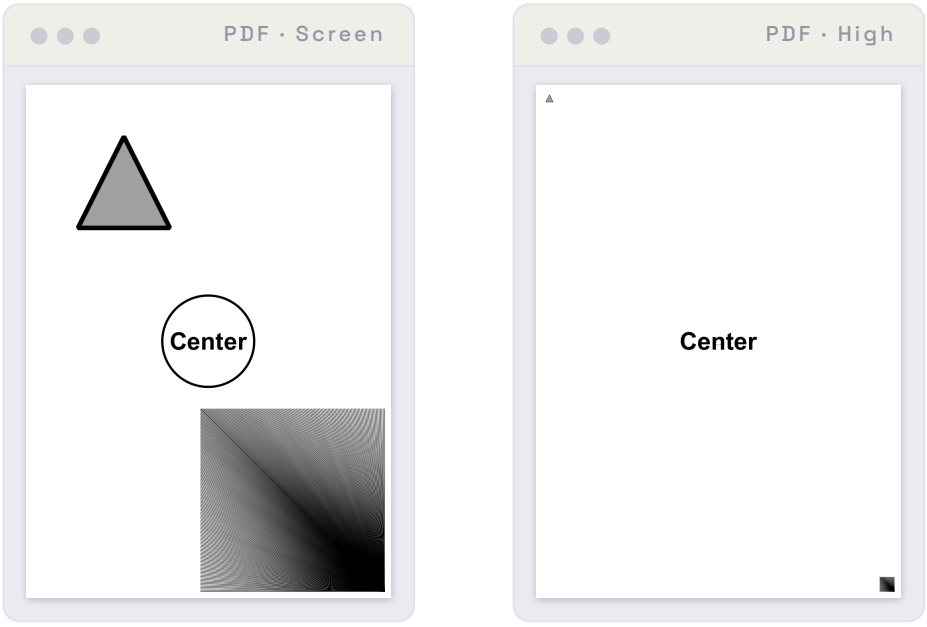
Pixels on paper

On paper, a pixel is not a dot on your screen, so how big is it? That depends on the **Printer Resolution** in the preferences:

RESOLUTION	PIXELS PER INCH	A4 PAGE	LETTER PAGE
Screen	about 96 (0.26 mm a pixel)	767 × 1096	790 × 1030
High, PDF file	1200 (0.021 mm a pixel)	9582 × 13699	9866 × 12866

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

In **Screen** resolution, a pixel on the page is about as big as on your screen, so the numbers you know from the graphics output work on paper too. An A4 page is 767 pixels wide. In **High** resolution the page has many more, much smaller pixels: 12.5 times as many in each direction for a PDF file. The page $x = 0$, $y = 0$ is not at the very edge of the paper but at the top left corner of the area that can be printed, a few millimeters in.



Program 94 printed twice: in Screen resolution (left) and in High resolution (right).

In High resolution, the triangle at 200, 100 and the 400-pixel moiré square shrink to tiny shapes in the corners, and the circle around the word is too small to see. The word itself stays the same size, because **font** measures text in **points**, not pixels. A point is 1/72 of an inch (about 0.35 mm), whatever the resolution: a 12-point line of text has the same size on every page. So a program written for Screen resolution prints its drawings tiny in High resolution. Program 94 still centers its word in both, because it measures the page and the text instead of guessing.



WHY USE HIGH RESOLUTION?

Small pixels draw fine detail: smooth curves and thin lines that are only 1/1200 of an inch wide. But then every number in your program must be about 12 times bigger. A program that works out its positions from **graphwidth**, **graphheight**, **textwidth** and **textheight** doesn't need to know which resolution it is printing in.



TRY IT

Change the Orientation to **Landscape** in the preferences and run Program 94 again. The word is still in the middle, and the moiré pattern still sits in the bottom right corner, now of a page that is wider than it is tall.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 4

More than one page

`printer page` starts a new page, and `printer cancel` stops printing.

A document can have as many pages as you like. **printer page** finishes the page you are drawing on and starts a new, empty one. Program 95 prints three pages, each with a big page number in the middle and a small line at the bottom.

pages.kbs BASIC-256

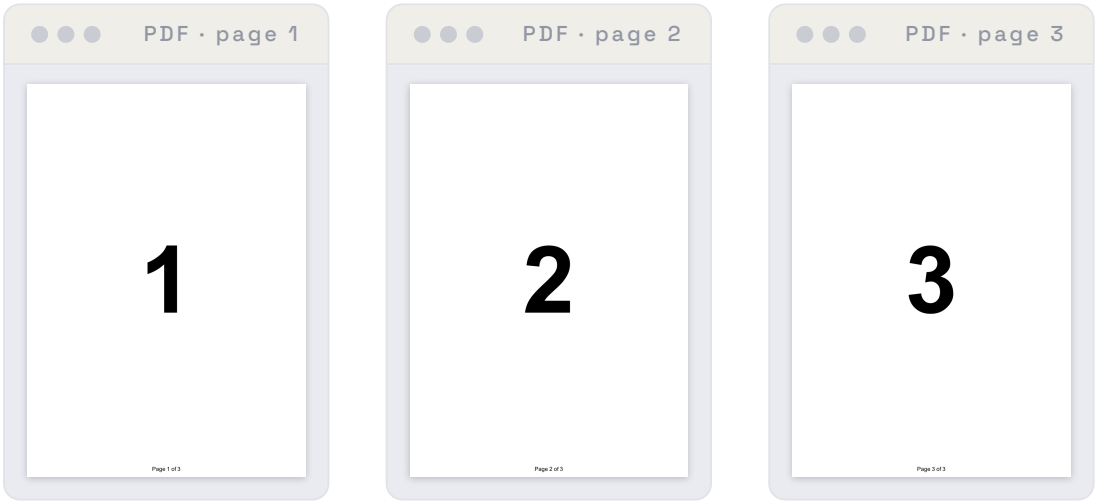
```
# pages.kbs - print three pages, each with a big number

printer on
for p = 1 to 3
   if p > 1 then printer page   # start a new page

   # a big page number in the middle
   font "Arial", 200, 700
   n = "" + p
   text (graphwidth - textwidth(n)) / 2, (graphheight - textheight()) / 2, n

   # a small line at the bottom
   font "Arial", 12, 400
   foot = "Page " + p + " of 3"
   text (graphwidth - textwidth(foot)) / 2, graphheight - textheight(), foot
next p
printer off
print "3 pages are printed."
```

Program 95 · Printing Three Pages



The three pages of the PDF file.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

printer on has already started page 1. So the program starts a new page only from page 2 on: `if p > 1 then printer page`. If it used **printer page** after each page instead, the document would end with an empty fourth page. The number *p* is turned into a string with `"" + p` (Chapter 6) so that **textwidth** can measure it.



NEW CONCEPT

```
printer page
```

printer page finishes the current page and starts a new, empty one. Everything drawn after it goes on the new page.



NEW CONCEPT

```
printer cancel
```

printer cancel turns printing off without finishing, if you started a document but decide you don’t want it. With a printer, nothing is printed. After **printer cancel**, drawing goes to the graphics output again.

With PDF File Output, a canceled document is saved anyway. In BASIC-256 2.3, **printer cancel** doesn’t remove the PDF file: it still holds the pages drawn so far. The same happens when a program stops while the printer is on, because of an error or because you pressed Stop. So end every document with **printer off**, and don’t count on **printer cancel** to throw a PDF file away.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout



BIG PROGRAM

A Multiplication Table

Print a 12 × 12 multiplication table with a title, a shaded header row and column, and every number centered in its cell.

The table has 14 columns: the header column and the columns for 0 to 12. Dividing the width of the page by 14 gives the width of a cell, *w*, so the table always fills the page from side to side. A cell is two lines of text high. The grid starts at *top*, below the title.

The header row and column are two light gray rectangles, drawn first so that the grid lines and numbers come on top of them. One loop draws all the grid lines: each time a vertical line and a horizontal one.

Every number is centered in its cell, the same way Program 94 centers its word: **textwidth** gives the room it needs, and half of what is left goes on each side. The headers are bold (weight 700), the products normal (weight 400).

PDF · myprint.pdf

Multiplication Table

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12
2	0	2	4	6	8	10	12	14	16	18	20	22	24
3	0	3	6	9	12	15	18	21	24	27	30	33	36
4	0	4	8	12	16	20	24	28	32	36	40	44	48
5	0	5	10	15	20	25	30	35	40	45	50	55	60
6	0	6	12	18	24	30	36	42	48	54	60	66	72
7	0	7	14	21	28	35	42	49	56	63	70	77	84
8	0	8	16	24	32	40	48	56	64	72	80	88	96
9	0	9	18	27	36	45	54	63	72	81	90	99	108
10	0	10	20	30	40	50	60	70	80	90	100	110	120
11	0	11	22	33	44	55	66	77	88	99	110	121	132
12	0	12	24	36	48	60	72	84	96	108	120	132	144

multtable.kbs BASIC-256

```
# multtable.kbs - print a 12 x 12 multiplication table

printer on
color black

# a title, centered at the top of the page
font "Arial", 24, 700
title = "Multiplication Table"
text (graphwidth - textwidth(title)) / 2, 0, title

# the size of a cell: 14 columns fill the width of the page
font "Arial", 12, 700
w = graphwidth / 14
h = textheight() * 2
top = 80        # the grid starts 80 pixels down the page

# shade the header row and the header column
color rgb(220, 220, 220)
rect 0, top, 14 * w, h
rect 0, top, w, 14 * h

# draw the grid
color black
penwidth 2
```

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

```
for i = 0 to 14
   line i * w, top, i * w, top + 14 * h
   line 0, top + i * h, 14 * w, top + i * h
next i

# the numbers in the header row and column, in bold
for i = 0 to 12
   n = "" + i
   x = (i + 1) * w + (w - textwidth(n)) / 2
   y = (i + 1) * h + (h - textheight()) / 2
   text x, top + (h - textheight()) / 2, n
   text (w - textwidth(n)) / 2, top + y, n
next i

# the products, in normal type
font "Arial", 12, 400
for row = 0 to 12
   for col = 0 to 12
      n = "" + (row * col)
      x = (col + 1) * w + (w - textwidth(n)) / 2
      y = top + (row + 1) * h + (h - textheight()) / 2
      text x, y, n
   next col
next row

printer off
print "The table is printed."
```

Program 96 · Big Program — Multiplication Table

TEXT OUTPUT

The table is printed.

The text output. The table itself is in the PDF file.



TRY IT

Make the table fill the whole page from top to bottom. Work out *h* from the room left under the title: `h = (graphheight - top) / 14`.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout



PRACTICE

Exercises

WORD SEARCH

d r e t n i r p h o t s l l
h t e x t w i d t h u h l t
m d f f j a l e h w r p i b
a a w d l e c n a c e n s m
r s b p v p o i n t p i y g
g a j v v s y u d w a v r w
i f t p a g e e w c p i c i
n p r e f e r e n c e s j w
b e t e x t h e i g h t e h
e w r e p a c s d n a l t l
p o r t r a i t c b o i s d
b e g e g c r l j p b s r r
i g r v h t d i w h p a r g
n o i t u l o s e r v l k i

Find: cancel, graphwidth, landscape, margin, page, paper, pdf, point, portrait, preferences, printer, resolution, textheight, textwidth



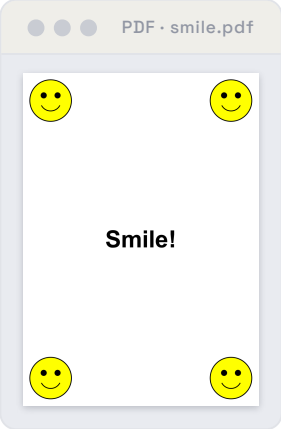
Problems

Try these — they only use what this chapter and the earlier ones covered.

1. Take your program from Problem 1 or 2 of Chapter 6, the playground rhyme or “Old MacDonald”, and have it print the verse on a page after the user has typed in the words. Keep a variable with how far down the page the next line goes, as Program 93 does.

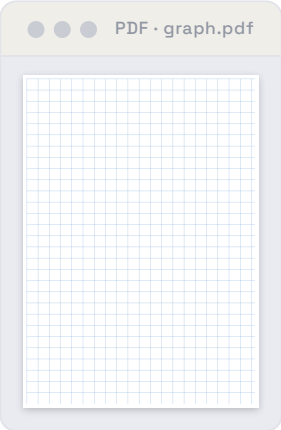
© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

2 Use the smiling face subroutine from Problem 1 of Chapter 10 to print a page with a smiling face in each of the four corners and “Smile!” in the middle of the page. Make the faces bigger than on the screen: give the subroutine the radius as a third number.



3 Ask the user for a number with **inputinteger** (Chapter 6) and print its times table on a page: *7 × 1 = 7* to *7 × 12 = 84*, one line each, with a title. Then change the program to print the tables of 1 to 12, each on its own page. Make sure the document doesn’t end with an empty page.

4 Print a page of graph paper, with a light blue line every centimeter across and down. In Screen resolution there are 96 pixels in an inch, and an inch is 2.54 cm. How many pixels is a centimeter? Measure your printed page with a ruler to check.



Chapter 14 · Printing. Modernised layout of the BASIC-256 beginner’s guide, updated for BASIC-256 2.3. Programs are numbered 93–96, continuing from Chapter 13. Corrected for 2.3: the statement is `printer on` (the old text said “print on”); font weights use Qt 6’s 1–1000 scale, so the old 100 (now very thin) became 700 for bold and the old 50 became 400 for normal; a point is 1/72 inch, about 0.35 mm (not 3.5 mm); the preferences are under Edit › Preferences..., where the Printing tab now comes after Sound; with PDF File Output `printer cancel`, or a program that stops while printing, still saves the PDF file. The Big Program’s cells were 700/13 pixels wide for 14 columns; they are now worked out from `graphwidth`, and the numbers are centered with `textwidth`, under a title and shaded headers. Program 93 uses *y* (not *x*) for the distance down the page and `+=`; Program 94 keeps the page size in variables. Problem 1 now points to the rhymes of Chapter 6. New: Section 1 on choosing a printer or a PDF file first, the page sizes in Screen and High resolution with the same program printed in both, the margin, Program 95 with `printer page` (and why it doesn’t come after the last page), and Problems 3 and 4. Every page shown was printed by the program into a PDF file with BASIC-256 2.3, on A4 paper in Screen resolution unless marked High. Original text © 2019 James M. Reneau, licensed under CC BY-NC-SA 3.0 US. This adaptation is shared under the same licence. Mascot: BitBot.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout