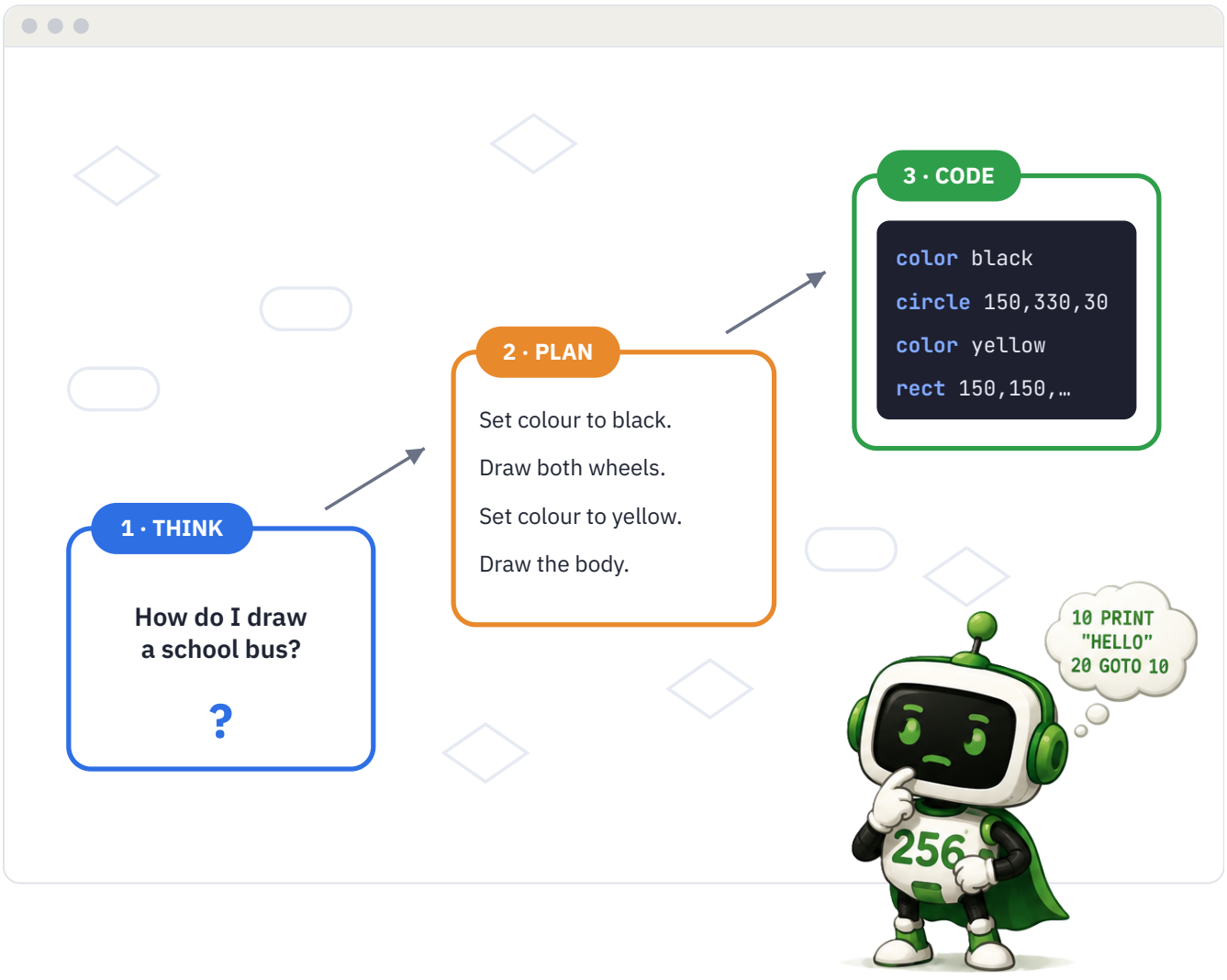
LEARN TO CODE · PROBLEM SOLVING

Chapter 5

Thinking Like a Programmer

Before you write a single line of code, you need a plan. In this chapter you will learn two tools that programmers use to work out a plan: pseudocode, which writes the steps in plain English, and flowcharts, which draw them as a picture.



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

SECTION 1

Think first, code later

One of the hardest things to learn is how to think like a programmer.

No book or class can turn you into a programmer by itself: a programmer grows from the inside. Becoming a good programmer takes a passion for technology, teaching yourself new things, and the drive to create and explore.

You are like the great explorers Neil Armstrong, the first person to walk on the Moon, and Yuri Gagarin, the first person in space. Inside the computer there is a whole universe for you to explore and to build in. The only limits on where you can go are your creativity and your willingness to learn.

A game or a useful application can easily run to several thousand lines of code. That can quickly become overwhelming, even for the most experienced programmer. So programmers often tackle a big problem in three steps:

1 Think

Think about the problem. What should the program do?

2 Plan

Break the problem into small pieces and write them down clearly.

3 Code

Turn each piece into statements of the programming language you are using.

Step 2 is where this chapter comes in. You will meet two ways of writing down a plan: **pseudocode** in Section 2 and **flowcharts** in Section 3.

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

SECTION 2

Pseudocode

Pseudocode means writing out, step by step, what your program needs to do.



NEW CONCEPT

Pseudocode is a step-by-step description of a program, written in short, plain sentences. The word comes from the Greek prefix *pseudo-*, meaning “false”, and *code*, the statements of a programming language. It is not meant for the computer: it is meant for **you**, to help you understand a problem and break it into meaningful pieces.

There is no single best way to write pseudocode. Dozens of styles exist, and each one suits a particular kind of problem. In this chapter we will simply use short English sentences.

How would you write a program to draw a school bus like this one?

GRAPHICS OUTPUT



The school bus we want to draw.

Let’s break the problem into two steps: draw the **wheels**, then draw the **body**. Now break those steps into smaller pieces and write them down as pseudocode:

PSEUDOCODE

1. Set the colour to black.
2. Draw both wheels.
3. Set the colour to yellow.
4. Draw the body of the bus.
5. Draw the front of the bus.

Now that the plan is worked out, all we need to do is write it. Each line of pseudocode becomes one or two BASIC-256 statements:

PSEUDOCODE	BASIC-256
Set the colour to black.	`color black`
Draw both wheels.	`circle 150,330,30` `circle 380,330,30`
Set the colour to yellow.	`color yellow`
Draw the body of the bus.	`rect 150,150,300,150`
Draw the front of the bus.	`rect 50,225,100,75`

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

schoolbus.kbs BASIC-256

```
# schoolbus.kbs - draw a school bus
clg

# draw the wheels
color black
circle 150,330,30
circle 380,330,30

# draw the body of the bus
color yellow
rect 150,150,300,150

# draw the front of the bus
rect 50,225,100,75
```

Program 31 · School Bus

Run it, and you get exactly the bus at the start of this section. Look at the finished program and you will see that its remarks are the steps from the pseudocode. They remind the programmer, and anyone who reads the program later, of how the problem was solved.

There were many ways to break up this problem. You could have drawn the body first and the wheels last, or the front before the back. There are dozens of ways to tackle even a simple problem like this one.

There is no wrong way to approach a problem. Some ways are better than others: they need fewer statements, or they are easier to read. But the important thing is that you solved the problem.



TRY IT

Draw the wheels *after* the body: move the `color black` line and the two `circle` lines to the end of the program and run it again. Does the picture change? Why not?

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

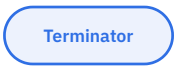
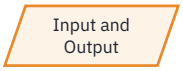
SECTION 3

Flowcharting

A flowchart draws a picture of what a program, or any process, has to do.

Following the old saying that “a picture is worth a thousand words”, programmers sometimes draw a diagram of the logic of a program. Flowcharting is one of the oldest and most widely used ways of showing what a program is supposed to do.

This short introduction covers only a small part of what flowcharts can do, but with a few simple symbols and connectors you can model very complex processes. It is a technique that will help you not only in programming but with many other problems too. Here are the essential symbols:

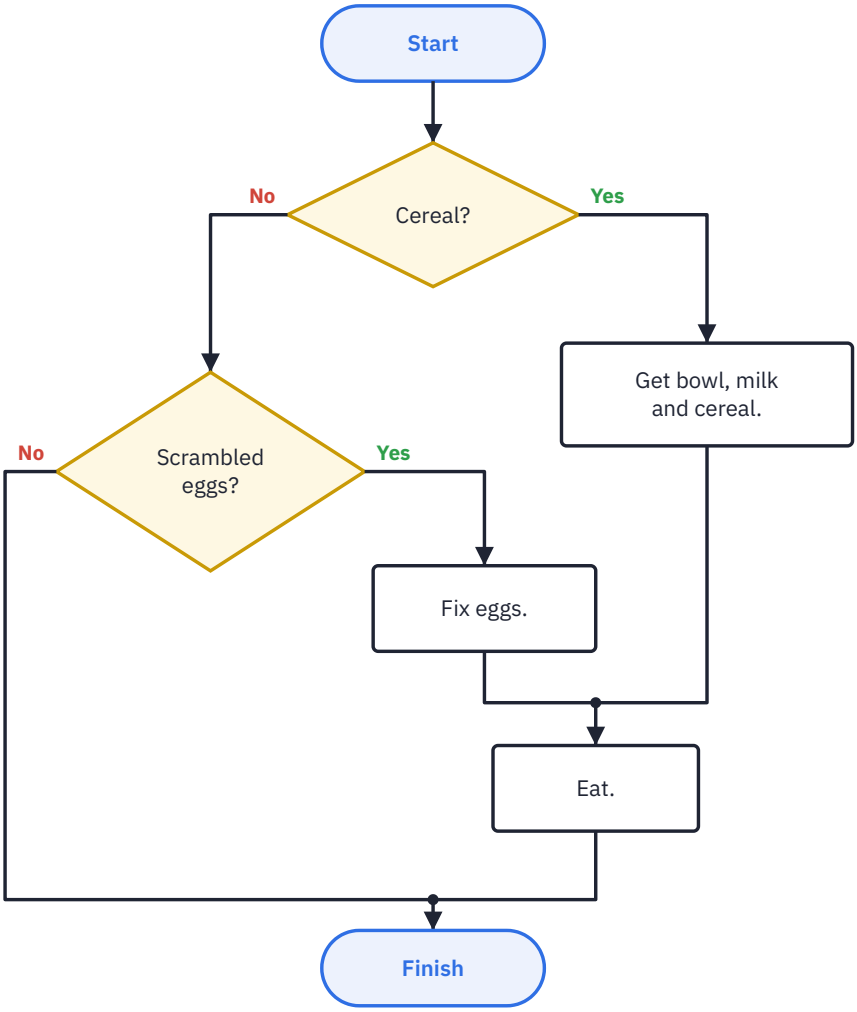
SYMBOL	NAME AND MEANING
	**Flow line** An arrow shows the move from one symbol, or step, to the next. Always follow the direction of the arrowhead.
	**Terminator** Shows where the flowchart starts and where it finishes. Every flowchart has a **Start** and a **Finish**.
	**Process** An action or activity: something the program, or the person, does. Only one arrow leaves a process.
	**Input and Output** Data going into the program, like a key being typed or a file being loaded, or coming out of it, like text on the screen or a file being saved.
	**Decision** Asks a simple question with two possible answers: yes or no, true or false. Two arrows leave a decision, and the answer decides which one you follow.

The best way to learn flowcharting is to look at some examples, and then try your own hand at it.

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

Example one: breakfast

You have just rolled out of bed and your parent offers you two choices for breakfast: your favourite cold cereal or a scrambled egg. If you don't choose either of them, you go to school hungry.



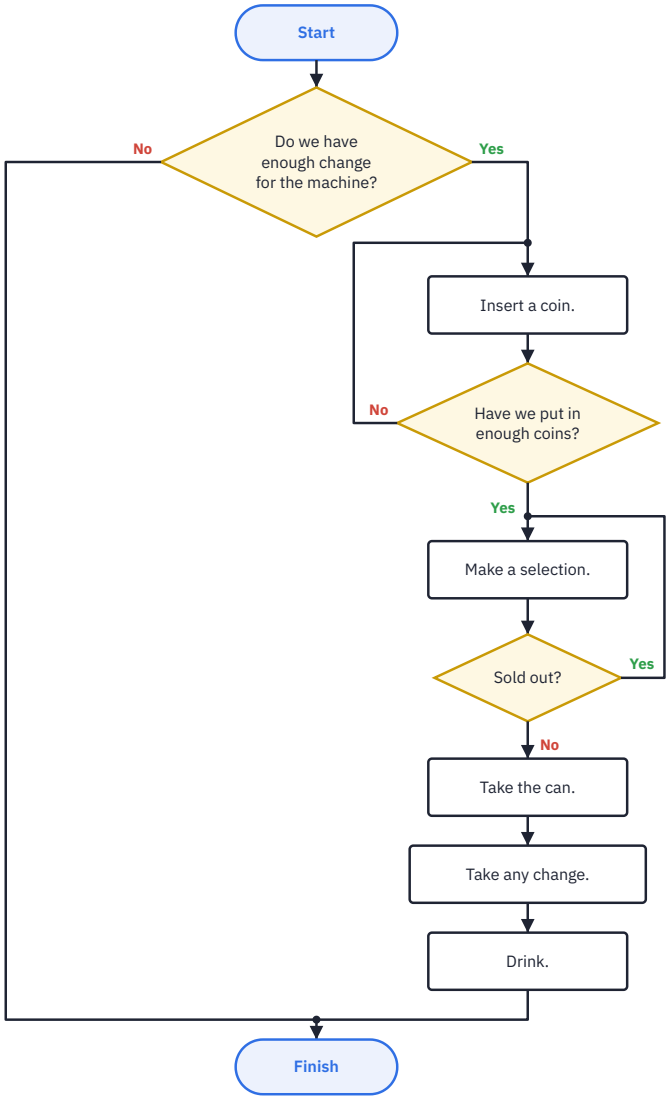
Breakfast as a flowchart. Each decision has a Yes and a No arrow.

Follow the arrows from **Start**. Can you find all three ways to reach **Finish**? Notice how the two paths with food join up again before **Eat**, so that step only has to be drawn once.

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

Example two: a drink from the machine

You are thirsty and want a can from the drinks machine. Take a look at this flowchart.



Buying a drink from a machine. Two of the arrows lead back up the chart.

This flowchart has two places where a step can happen again: we keep inserting coins until we have put in enough, and if our choice is sold out we go back and make another selection. The flowchart shows these **loops** as arrows that lead back up the chart.



COMING UP

The decisions and loops in these flowcharts are exactly the tools a program needs too. You haven't seen how to write them in BASIC-256 yet, but they are covered in the next few chapters.

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



PRACTICE

Exercises

WORD SEARCH

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

Find: decision, flowchart, input, output, problem, process, programming, pseudocode, steps, symbol, terminator



Problems

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

1 In complete sentences, write out the steps to make a peanut butter and jelly sandwich. Assume that the peanut butter jar, the jelly jar, a loaf of bread, a plate and the cutlery are on the table in front of you. Could someone who has never seen this sandwich make one by following your directions?

2 Draw a flowchart (or a similar diagram) of what you do to open the front door of your house or apartment. Do you have your keys? Is the door locked? Is it already open?

3 In pseudocode (short statements), write directions from your school or workplace to the nearest restaurant or petrol station. Don't cheat by looking them up online! Will the same directions get you back, or do they need to be changed?

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

> **4** Write pseudocode for drawing a house: a square front, a door, two windows and a roof. Then turn each step into BASIC-256 statements, using `rect`, `line` and `color`, and put remarks in the program that match your pseudocode.

Chapter 5 · Thinking Like a Programmer. Modernised layout of the BASIC-256 beginner's guide, updated for BASIC-256 2.3. The school bus is redrawn for the 500 × 500 graphics area used since Chapter 2, its remarks now follow the pseudocode step by step, and Program 31 continues the numbering from Chapter 4. In the flowcharts, "Eat" and "Drink" are now process boxes: they are actions, not input or output. Typos in the text and the problems were corrected, and a fourth problem was added. The picture of the school bus was made by running Program 31 in BASIC-256 2.3. 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.