Programming in Food

Ian Millington

Draft: 5 December 2013

Thanks for previewing the draft. This booklet introduces important ideas in computer science and computer programming without showing any computer code at all. I hope it can help bridge the gap between not knowing anything about the ideas and jargon of computer programming and learning a programming language. Although computer programming uses its own odd jargon, most of the ideas behind the jargon aren't unique and apply to anything where you have to give step-by-step instructions. Like recipes for cooking, for example.

This is an early draft. The biggest question is: is this useful? It isn't useful if it is too confusing, or if it takes longer to read through and understand this booklet than it would just to learn the same ideas as you go, when learning to program. That's where you come in. Let me know what was confusing, let me know if the whole thing left you going 'eh?'. At this stage there's no need to worry about typos and spelling. It will be better edited when the content is right: I've written four large textbooks and I've never got the hang of editing as I go!

Thanks again, in advance, for your feedback.

The Cake Algorithm

A recipe is a step-by-step set of instructions for cooking a dish. It is written by someone who knows how to break down the task into small steps. It is read and carried out (in computer jargon we say it is **run**) by someone who can get the correct result if they follow the steps accurately.

Computer science is the study of how to break processes into steps. Programming languages, on a computer, are ways to describe those steps, and have the computer do almost anything, from calculating tax returns to displaying your Facebook feed, to playing video games. But there's nothing special about programming a computer. Programming is just writing: writing a set of step-by-step instructions. The writer of a cookery book is a kind of programmer, and they use the same kinds of tactics and conventions that computer programmers do.

In this booklet I'll show you some of the ideas and jargon we use in computer programming. But I hope I won't be teaching you much that's new. I'll just be giving things different names.

Let's start with recipes themselves. In computing we call a recipe a **program**. If you look online or in the library for a recipe for Welsh Rarebit, you'll find lots. And they'll all bit a little bit different. Most will have some deeper similarity though, because all of them want to make a similar dish at the end of it.

This underlying similarity we call an **algorithm**. An algorithm is a method for doing something, which a programmer then turns into a program, which is the thing that is actually carried out. So there might be just a few algorithms to make Welsh Rarebit, but possibly tens of thousands of different recipes. In the same way, there are just a few algorithms for computers to work out the fastest route between two cities (in a navigation app, say), but every navigation app will have its own program to do that.

So if I ask you what your algorithm for Welsh Rarebit is, you might say "I'm going to make the sauce for the topping, toast the bread a bit on its own, then spread on the topping and grill again to finish." If I asked you to write a recipe you'd write a program based on that algorithm:

Welsh Rarebit

$\frac{1}{2}$ onion.
50 ml / 2 oz dark beer.
2 tsp English (Welsh) mustard.
125 g / 5 oz mature Welsh cheddar.
4 tsp Worcestershire sauce.
1 tbsp butter.
2 slices of thick crusty white bread.
Finely dice the onion.
Cook the onion in butter.
Take the pan off the heat.
Grate the cheese and add to the onion.
Mix in the mustard and the Worcestershire sauce.
Poor over the beer and stir until it is full incorporated.
Lightly toast the bread, until the surface is no longer soft, but it is not yet colored.
Spread the cheese mix over the bread.
Cook under the grill for a minute, or until the cheese is lightly browned and bubbling.

For Subroutines, See Page 200

If you read enough recipes, you'll see the same building-blocks appear over and over again. Many dishes may require a chicken-stock, for example. Rather than explaining how to make a chicken stock each time, a cookery book might say "you can buy good quality chicken stock from your grocery store, or follow the recipe on page 200." Let's say you aren't going to buy the chicken stock, this instruction tells you to go to page 200, follow that recipe, and come back with the final product when you're finished.

Building a recipe from other recipes is a key part of programming. In computer science these building-block recipes are called **subroutines**. Because of computer science's long association with mathematics, you might also see subroutines called **functions**. And finally, some programming languages call them **procedures**. But all three names refer to the same idea: go to another recipe, follow it, and come back when you're done.

The act of instructing the computer to go to another subroutine, and return when it is done, is known as **call**ing the subroutine.

Subroutines are really important in programming. Often the same task comes up over and over again, it is a waste of time and space to document all the steps each time.

In classical French cooking there are five famous 'mother-sauces': the Béchamel, the Velouté, the Hollandaise, the Espagnole, and the Tomato sauces. A simple recipe for the creamy white Béchamel sauce might be:

BÉCHAMEL SAUCE (to make 450 ml / 1$\frac{1}{2}$ cups, medium-thickness)
300 ml / 1 cup of full fat milk.
2 tbsp of butter.
2 tbsp of flour.
Warm the milk without bringing to the boil.
At the same time, prepare a 'roux':

> Melt the butter in a pan and add the flour.
> Cook the flour without browning for 2 min, stirring with a whisk.

Gradually add the warm milk to the roux, whisking vigorously.
Season the sauce to taste.

Once we have a Béchamel, we can use it to make further sauces:

MORNAY SAUCE (to make 600 ml / 2 cups, medium-thickness)
450 ml / 1$\frac{1}{2}$ cups, medium-thickness Béchamel sauce.
2 tbsp finely grated Parmesan cheese.
2 tbsp finely grated Gruyère cheese.

Add the cheese gradually into the warm Béchamel, stirring well.

And once we have the Mornay sauce, we can use that in further recipes:

Cod Mornay (serves 2)
2 cod fillets.
600 ml / 2 cups of milk.
1 bay leaf.
2 peppercorns.
1 tsp finely chopped chives.
600 ml / 2 cups Mornay sauce.
Put the milk, bay leaf and peppercorns into a pan and bring to the slightest simmer: do not allow the milk to boil.
Place the cod fillets into the milk, and simmer gently for 8-10 minutes, until the cod is cooked.
Remove the cod from the milk and place on a plate.
Pour over the Mornay sauce.
Sprinkle the chives on top.

In this sequence we have three recipe. The Cod Mornay calls (i.e. refers to) to the Mornay Sauce, which in turn calls the Béchamel.

There's nothing to stop us splitting things into simpler and simpler recipes, if we like. We could take the Béchamel and split out the roux:

Roux (to make 4 tbsp)
2 tbsp of butter.
2 tbsp of flour.
Melt the butter in a pan and add the flour.
Cook the flour without browning for 2 minutes, stirring with a whisk.

If we have the roux as a separate recipe, another of our French mother-sauces can use it:

Velouté Sauce (to make 450 ml / $1\frac{1}{2}$ cups, medium-thickness)
300 ml / 1 cup of clear chicken stock.
4 tbsp of roux.
Warm the chicken stock without bringing to the boil.
Gradually add the warm stock to the roux, whisking vigorously.
Season the sauce to taste.

The chicken stock, of course, is on page 200! Breaking things down to this level might seem silly. A cookbook where you have to cross-reference five pages to make poached cod in cheese sauce, is probably not going to be the first one you reach for. So cookbook writers repeat the basic building blocks over and over again through each recipe that uses it. This makes the book longer, but makes it easier on the cook.

Unlike a human cook leafing through a book, computers don't mind lots of cross-referencing. They can flick to another recipe in a fraction of a second. So computer programmers tend to use lots of subroutines, each calling other subroutines, until we end up in subroutines that do one simple thing, like making a roux.

Arguing with Recipes

In the previous section, I carefully wrote out the recipes so that the quantities would be correct, when all the recipes came together to make our final dish. But what happens if a recipe calls for a little bit more béchamel, or a little bit less roux?

We could write our recipe in this way:

Béchamel Sauce (per 300 ml / 1 cup, medium-thickness)
200 ml / $\frac{2}{3}$ of a cup of full fat milk.
2 tbsp of roux.
Warm the milk without bringing to the boil.

Gradually add the warm milk to the roux, whisking vigorously.
Season the sauce to taste.

I've only tweaked the recipe slightly, to emphasize that this subroutine for making Béchamel sauce can be used to make any quantity that a recipe needs. We can arrive at this recipe, knowing we need 4 cups of sauce, and we can increase the quantities accordingly. A more mathematical way of writing the same thing might be:

BÉCHAMEL SAUCE (to make $300x$ ml / x cups, medium-thickness)
$200x$ ml / $\frac{2}{3}x$ cups of full fat milk.
$3x$ tbsp of roux.
Warm the milk without bringing to the boil.
Gradually add the warm milk to the roux, whisking vigorously.
Season the sauce to taste.

This is just another way of saying the same thing. If you want 2 cups of sauce (so $x = 2$), you'll need 6 tbsp of roux ($3x = 6$).

Things that can change in a subroutine are called **parameters** or **arguments** in computer science (yes, arguments, really – the mathematicians who came up with the jargon of computer science in the 1940s and 1950s were sometimes a bit obtuse). In the subroutine for Béchamel, we have one argument or parameter: the quantity of sauce to make. The same recipe can be used for any amount of sauce.

For our recipes this might seem obvious, you just scale up quantities to make more. It might seem overly complex to write this with xs. But it isn't always that simple. There might be some recipes that, if you want more, you need to make it in batches: you can't just make more mixture and expect it to work. In other recipes you might be able to scale up the quantity, but the cooking time might vary in ways that aren't obvious, for example:

ROAST TURKEY (to roast a x kg / $2x$ lb turkey)
A x kg / $2x$ lb turkey.

150 g / 6 oz of unsalted butter.
$4x$ rashers of streaky bacon.
Pre-heat the oven to 220 C / 425 F
Place the turkey in a large roasting pan.
Rub butter, salt and pepper thoroughly into the skin of the bird.
Cover the turkey with the streaky bacon.
Pour a quarter-inch of water in to the bottom of the roasting pan.
Cook the turkey in the oven for 25 minutes.
Turn the oven down to 170 C / 325 F.
Remove the bacon and set aside.
Baste the bird in the juices from the roasting pan.
Cook the turkey for a further $30x$ min.

In this example, we need to be clear which values have to change and which don't. If you read a recipe for cooking a 12 lb turkey, it isn't obvious what timings you need for a 14 lb turkey. By being clear, above, we can see that the first time, to brown off the bacon, doesn't change, but the second cooking time does.

We used a number as a parameter to our roasting recipe: The number of pounds of weight of our turkey. But it isn't only just a number that can be used as a parameter. Think about this recipe:

Herb Oil (oil infused with *herb*)
1 bottle of sunflower oil.
1 bunch of fresh *herb*.
Warm the oil gently in a pan. Hot oil will cook the *herb*, and that is not what you want.
At the same time: wash the *herb* thoroughly.
Scrunch the *herb* in your hands to bruise it and begin releasing its flavor.
Put the *herb* into a jar or sealable bottle and fill with the warm oil.
Seal the container, and store in a cool dark place for one month.
Strain the oil and return to its container.

In this case we have a parameter, but it is an ingredient, not a number. With this recipe we can make oregano oil (*herb*=oregano), or thyme oil (*herb*=thyme).

So parameters can refer to anything, but each parameter has a **type**. You can't make herb oil where *herb*=12, and you can't roast a turkey where its weight, x=oregano. That is obvious nonsense to humans, but making sure parameters are the correct type is a big deal in computer programming.

You can't expect the computer to know how to vary a recipe, what values change and what stay the same, even for the simplest subroutines. So computer programmers always have to make these parameters clear. They give them names (in the same way we used x, or *herb* above) and types (x must be a number, *herb* must be a fragrant edible plant). Then the subroutines are told what changes need to be made inside the recipe depending on the values of the parameters they are given.

The final thing to notice – though it might seem obvious – is that the subroutines we've seen so far all produce something. There is an end result of the recipe. A Béchamel sauce recipe ends up with a jug of Béchamel sauce. In computing jargon, we say that the subroutine **return**s its result, ready to be used by the recipe that referred to it.

Many subroutines do produce something new to return. But there are some that, instead of creating something new, modify something that already exists. You can imagine part of a cookery book giving instructions on how to carve designs into your pastry with a knife. Those instructions are a kind of subroutine. A Beef Wellington recipe might say "the traditional design features narrow scoring, shown on page 300." So you go to page 300, follow the instructions there, and come back when you're done: just like any other subroutine. But now, rather than giving us back some new ingredient, this subroutine modifies the pie we've created. In computing this is sometimes called a **side-effect**, something that a subroutine does to modify something that already exists.

So we've seen three important properties of subroutines that are present in cookery, and are crucial in computer programming: subroutines break down the task into smaller sub-tasks that can be used in lots of different places; subroutines have parameters that can tweak the task in some way; and subroutines either return some new result, or they have side-effects, modifying something that already exists.

The Variables are in the Fridge

As the recipe progresses, you often need to prepare parts of the whole one at a time, then those parts come together later in the recipe. A recipe writer needs to refer to the results of previous steps so we can do more to them, or combine them with further ingredients.

If you look through some recipe books, you'll see writers try to do this as little as possible. They try to assume that each step follows on from the previous, using the same pan or mixing bowl. When a recipe says "add the butter and mix until you have a fine crumb," the author hasn't told you exactly what you're adding this butter *to*. Knowing the way recipes work, you just assume that you need to add it to whatever you got to in the previous step.

When this assumption would be wrong, the author has to come up with another approach. For example:

Spinach Salad and Brown Butter Dressing
150 g / 5 oz of baby spinach
3 tbsp unsalted peanuts
4 tbsp salted butter
1 shallot
1 tbsp white wine vinegar
2 tbsp honey
Crush the peanuts and toss through the spinach.
Put the salad to one side.
Melt the butter over a gentle heat.

Finely dice the shallot and cook in the butter for 4 minutes, until the butter has turned a nut-brown color.
Remove from the heat and add the vinegar and honey, stirring well.
Pour over the salad and toss.

When the recipe says "Put the salad to one side", it tells us two things: firstly that whatever we have, we're referring to as "salad", so later in the recipe when we read "pour over the salad and toss," we know we need to pick this thing up and use it; secondly it tells us that we will need to use this thing again, so we need to keep it safe. We can't throw the salad away before it is used.

Like many things we've seen so far, this is obvious. But as before, this process is crucial to how computer programming works. We call these things **variable**s. A variable has the same two properties: it gives a name to something, and it tells the computer to keep it safe, so it can be used again. The biggest difference between computers and recipes is that computers aren't good at figuring out things on their own. So while variables are not used all the time in recipes, they are everywhere in computer programs. A computer program might use variables for lots of the steps:

Spinach Salad and Brown Butter Dressing
150 g / 5 oz of baby spinach
3 tbsp unsalted peanuts
4 tbsp salted butter
1 shallot
1 tbsp white wine vinegar
2 tbsp honey
Crush the peanuts and toss through the spinach - this is the 'salad'.
Melt the butter over a gentle heat - this is the 'melted-butter'.
Finely dice the shallot and cook in the melted-butter for 4 minutes, until the butter has turned a nut-brown color - this is now the 'melted-butter'.
Remove the melted butter from the heat and add the vinegar and honey, stirring well - this is the 'dressing'.

Pour the dressing over the salad and toss.

In this recipe the steps don't follow on from one another. If you need to add something to a previous step, that is spelled out for you. For a person, this kind of naming every step is not necessary, but it is still understandable. For a computer, it is very normal to be defining lots of variables like this and using the variable names every time we do another step in the program. Computers aren't good at guessing how one step follows another, so we spell it out in detail.

I previously mentioned that variables keep things safe, they tell you not to throw something away. Imagine working in a kitchen where you never throw anything away. If you're cooking something with lots of ingredients, your chopping board will get full of onion skins, the ends of leaks, potato peelings. A good cook throws things away.

The same is true in a computer program. Most modern programming languages feature **garbage collection**. This is like a kitchen helper who comes along and cleans things up for you while you cook. How does it know what to throw away and what you will need? It looks at the variables you've defined. If you've given a name to something, it leaves that until you need it again, or until you've finished the recipe.

Potato Loops

Most recipes you follow by beginning at the first step, working through one by one, until you get to the end. But there are times when we need to jump around in the list of instructions. This section and the next show the two most common situations.

A **loop** is a section of the recipe that you carry out more than one time. Have a look at this recipe:

Fondant Potato

300 g / 10 oz butter

4 potatoes

75 ml / 3 fl oz chicken stock

```
2 sprigs of fresh thyme
1 bay leaf
Pre-heat the oven to 200C / 390F.
For each potato:
    Peel and cut into 2cm thick slices.
    Discard the end pieces, so all slices are roughly the same size.
    Trim all the way around the slice at the top and bottom. This slight rounding increases the crispy coating at the ends, and gives the classic bar-of-soap shape to the potato.
Heat the butter in the pan until foaming.
Add the potato and season generously with salt and pepper.
Cook top and bottom of the potato in the butter until golden brown.
Take off the heat.
Add the thyme, bay leaf and stock.
Roast in the oven for 12-15 minutes.
```

Here we have a set of steps that you have to carry out for each potato. It is like a mini-recipe in itself, something that you follow from top to bottom. But instead of continuing on, you jump back to the start of the loop for the next potato. Continuing only after you've prepared every one.

Recipe writers often write their recipes so these loops aren't so obvious, they might write "Peel the potatoes and into 2cm thick slices, discarding the ends. Trim around the slice at the top and bottom to make the classic bar of soap shape." This just combines all the steps in the loop into one step in the recipe. You understand there's a loop there: you certainly cook with the loop there, but it is disguised a little in the recipe.

Computers aren't so good at inferring things, so when you want something done repeatedly in a computer program you spell out the loop, as I did in the Fondant Potato recipe, above.

There are three kinds of loop that are common in programs. We've seen the first above: a loop for every item in a set, for every potato in your recipe.

The second is a loop that repeats a particular number of times. In a recipe for lasagne for example, we might have to put layers of pasta, cheese sauce and ragù. The recipe might say

---

Excerpt from a Lasagne Recipe ...

```
Repeat three times:
    Add a layer of pasta, break additional sheets into smaller pieces
    to make sure all gaps are filled.
    Add a thin layer of cheese sauce.
    Add a generous layer of ragù, spread to fill the dish.
Add a final layer of pasta and top with the rest of the cheese sauce.
```

The final kind of loop is one that continues until some condition is met. In recipes, we see this most often with the 'until' condition being 'until cooked'.

Excerpt from a Roast Chicken recipe ...

```
Repeat every ten minutes:
    Baste the chicken with the juices from the pan.
    Push a long knife into the thickest part of the leg. If the juices
    run clear, the bird is ready.
    Otherwise, ensure there is still $\frac{1}{2}$ inch of stock in the pan, top-
    ping up if necessary, and return to the oven.
```

In both previous cases, as in the Fondant Potato example, many recipe writers will phrase the loop in ways that hide what is happening a little. My examples are at the opposite extreme, I've written them to make the loop as clear as possible.

These three loops – repeating for each member of a set, repeating a certain number of times, and repeating until some condition holds – are very common in programming.

Conditional Crabs

The final element of programs I want to think about are fairly rare in recipes, but much more common in computer programs. They are sections of the recipe that only need to be carried out, if some condition holds. In computer programming they are called **conditionals**, or **if-statements**.

For example

Crab Mayonnaise

1 large crab (live or bought freshly cooked)

150 g / 5 oz mayonnaise.

$\frac{1}{2}$ lemon

1 sprig fresh parsley

If you are cooking a live crab:

> Bring a large pot of well salted water to a vigorous boil. There should be plenty enough water to completely cover the crab.
>
> Add the crab and boil for 25 minutes.
>
> Remove from the water and stand upright to allow the water to drain.
>
> Leave to stand until cool.

Remove the crab meat from the shell and claws, taking care to discard the stomach, hard membranes, and 'dead men's fingers'.

Squeeze the juice of half a lemon into the crab, and mash until the crab is in small chunks.

Add the mayonnaise and fold into the crab.

Finely chop the parsley and add to the blend.

Grate in the rind of a quarter of a lemon.

Season with salt and pepper.

Here we have several steps, almost half of the recipe, that does not need to be followed, if we bought a cooked crab.

The purpose of if-statements is to make a program or part of a program more flexible: it can change its behavior depending on its circumstances.

For subroutines, this flexibility can allow the same subroutine to be used in more situations. Our original Béchamel recipe, for example, could be made more flexible:

```
BÉCHAMEL SAUCE (to make 450 ml / 1½ cups)
300 ml / 1 cup of full fat milk.
2 tbsp of butter.
2 tbsp of flour.
If you want a thick sauce, use 3 tbsp of butter and flour.
If you want a thin sauce, use 1 tbsp of butter and flour.
Warm the milk without bringing to the boil.
At the same time, prepare a roux:
    Melt the butter in a pan and add the flour.
    Cook the flour without browning for 2 min, stirring with a
    whisk.
Gradually add the warm milk to the roux, whisking vigorously.
Season the sauce to taste.
```

The recipe is now suitable as a subroutine in a wider variety of dishes, because we know how to adapt the recipe for thickness.

Abstract Master Chefs

If you have a range of cookery books you will notice there is a range of levels to pitch your recipes. Some cookery books assume you need to be walked through every step of the process. Others assume you have good cooking skills already. So one recipe might say "cut the carrot into thin strips, then into batons about two inches long". Another recipe might just ask you to "julienne the carrots." Both are asking the same thing, but one is giving more details. A cookery book called "How to Cook" is going to be very explicit, whereas recipes shared between professionals often read more like the menu: "poach the samfire in a plain emulsion."

We call programs that are very detailed, that tell you exactly how to carry out each step **low-level**, while recipes that assume you know most techniques are **high-level**. There is a range, of course, from low- to high-level, and where a program is on this scale is called its **level of abstraction**. Notice that, regardless of the level of abstraction, you need to do just as much work to cook the dish. "Julienne the carrots" is only three words, rather than twelve in the longer version, but it takes just the same number of cuts with the knife. The level of abstraction determines how much work the programmer needs to do, not the chef.

If you think of "julienne the carrots" as being a subroutine, then one way to think about these levels is: how many subroutines can I just assume you know, and how many do I need to spell out? If you're a skilled chef, then I can skip the recipe for a Velouté, and just use it to make a mustard sauce. If you don't know how to boil an egg, then I can't just ask you to go through your boil-an-egg subroutine, I will need to step through that process.

For example, here is a recipe for "Health Soup" (Potage de Santé) taken from the cookbook of the famous French Chef Auguste Escoffier.

Health Soup from "Le Guide Cullinaire", Auguste Escoffier, 1921.

Cook quickly, in salted water, three medium-sized, peeled, and quartered potatoes. When they feel soft to the touch, drain them: rub them through a fine sieve and dilute the remaining purée with $1\frac{1}{2}$ pints of white consommé. Add 2 tbsp of sorrel cooked in butter, and finish the preparation with an ordinary thickening and 1 oz of butter.

Garnish with very thin round slices of French dinner rolls and chervil pluches.

Presumably the average reader of Escoffier would know how to make white consommé, what ordinary thickening is, and how to select a pluche of chervil from the bunch. For most cooks, we'd need subroutines.

As we saw in the last section, we can keep dividing subroutines into simpler and simpler recipes. If we took that to silly extremes, we might end up with a "how to turn on the stove" recipe. In a real recipe, there is always some dividing line between subroutines that the author has to spell out, and those they assume you know. A high-level program will assume you know more than a low-level program.

Computers are the same. You can give the computer your program in various programming languages. A big difference between languages is what subroutines they already provide for common tasks. Some languages we call high-level languages, because they already know how to do lots of complicated things: they have lots of built-in subroutines. In low-level languages, you have to spell out all the basics too, and you do that by programming all those building block subroutines before tying them all together to make the finished program.

It is not just a matter of more or fewer subroutines: cooks of different cuisines will know different things. If you ask a traditionally trained Indian chef to cook a beurre blanc, they might need the step-by-step breakdown, whereas a French chef might need the same help to cook a hot-and-sour sauce. Two high-level recipes might need to spell out different subroutines depending on where the book is published.

Programming languages are often have specialities in the same way. So not every high-level language will have the same built-in subroutines. As a programmer, choosing the right language for the task is important. You have to select a language that you can easily write, and that has the right set of subroutines to get you as close to your goal as possible. Every language makes some recipes easier to write, and others harder.

Jargon We've Met

algorithm - the method underlying a program, many different programs can have the same algorithm.

arguments - another name for parameters

call - an instruction to find a subroutine, follow its instructions, and come back when it is complete.

conditional - A section of the program that will only be followed if some condition holds.

function - another name for a subroutine.

garbage-collection - an automatic cleaner that some programming languages use to throw away anything you're not going to use again.

high-level - A programming language with many subroutines provided for you. This allows you to write your program focusing more on the overall process, rather than the details of how to carry out each step.

if-statement - Another name for a conditional, so-named because the most common conditional in many programming languages begins with the word 'if'.

level-of-abstraction - Whether a program or programming language is high-level or low-level.

loop - A section of the program that repeats.

low-level - A programming language where not many subroutines are provided for you: where you have to build your program from the smallest components.

parameters - values that are sent to a subroutine that adjust how it is carried out.

procedure - another name for a subroutine. Can sometimes be used in a different sense to refer to an algorithm, i.e. the procedure for doing something.

program - equivalent to an actual recipe in a cookery book, it gives an algorithm a form that can be run.

return - when a subroutine completes, it returns to the part of the program that called it. Often it sends back some ingredient that has been made in that subroutine: this is called returning that ingredient.

run - to follow a program from start to finish.

side-effect - a change that a subroutine makes to something that already exists. In contrast to the subroutine making something new and returning it.

subroutine - part of a program that can be reused in different situations.

type - what kind of thing a parameter or variable refers to: a number, a herb, a meat. All are types.

variable - the name used in a recipe to refer to some intermediate result of the process.