**What is abstraction?**

Abstracting something means to give names to things, so that the name captures the core of what a function or a whole program does. *Reducing information and detail to focus on essential characteristics.*

**One example :**

Suppose we’re working with a graphing robot, and a common operation we need is to draw squares. “Draw a square” is an abstraction, or a mental chunk, of a number of smaller steps. So let’s write a function to capture the pattern of this “building block”:

Think of drawing a square. If I tell you to draw a square (on paper), you immediately know what to do:

* draw a square => draw a rectangle with all sides of the same length.

You can do this without further questions because you know by heart what a square is, without me telling you step by step. Here, the word square is the abstraction of "draw a rectangle with all sides of the same length".

**Abstractions run deep**

But wait, how do you know what a rectangle is? Well, that's another abstraction for the following:

* rectangle => draw two lines parallel to each other, of the same length, and then add another two parallel lines perpendicular to the other two lines, again of the same length but possibly of different length than the first two.

Of course it goes on and on - lines, parallel, perpendicular, connecting are all abstractions of well-known concepts.

Now, imagine each time you want a rectangle or a square to be drawn you have to give the full definition of a rectangle, or explain lines, parallel lines, perpendicular lines and connecting lines -- it would take far too long to do so.

**The real power of abstraction**

That's the first power of abstractions: they make talking and getting things done much easier.

The second power of abstractions comes from the nice property of composability: (water is composed of hydrogen and water. ) once you have defined abstractions, you can compose two or more abstractions to form a new, larger abstraction: say you are tired of drawing squares, but you really want to draw a house. Assume we have already defined the triangle, so then we can define:

* house => draw a square with a triangle on top of it

Next, you want a village:

* village => draw multiple houses next to each other

Oh wait, we want a city -- and we have a new concept street:

* city => draw many villages close to each other, fill empty spaces with more houses, but leave room for streets
* street => (some definition of street)

and so on...

**How does this all apply to programmming?**

If in the course of planning your program (a process known as analysis and design), you find good abstractions to the problem you are trying to solve, your programs become shorter, hence easier to write and - maybe more importantly - easier to read. The way to do this is to try and grasp the major concepts that define your problems -- as in the (simplified) example of drawing a house, this was squares and triangles, to draw a village it was houses.

In programming, we define abstractions as functions (and some other constructs like classes and modules, but let's focus on functions for now). A function essentially names a set of single statements, so a function essentially is an abstraction -- see the examples in your book for details.

**The beauty of it all**

In programming, abstractions can make or break productivity. That's why often times, commonly used functions are collected into libraries which can be reused by others. This means you don't have to worry about the details, you only need to understand how to use the ready-made abstractions. Obviously that should make things easier for you, so you can work faster and thus be more productive:

Example:

Imagine there is a graphics library called "nicepic" that contains pre-defined functions for all abstractions discussed above: rectangles, squares, triangles, house, village.

Say you want to create a program based on the above abstractions that paints a nice picture of a house, all you have to write is this:

import nicepic

draw\_house()

So that's just two lines of code to get something much more elaborate. Isn't that just wonderful?

**Definition for Abstraction:**

***In computer science, abstraction is the process by which data and programs are defined with a representation similar in form to its meaning, while hiding away the implementation details.***

So, abstraction tries to reduce and factor out details so that the programmer can focus on a few concepts at a time. (which is by the way a good life skill, simplify and break down a problem)

examples? Karel – buildTower, makePancakes, other functions you wrote

HW: <https://www.youtube.com/watch?v=_f-qkGJBPts>