

## How To ...

Reading: Thompson Ch.4

### COMP1100 — Introduction to Programming and Algorithms

Clem Baker-Finch

Australian National University  
Semester 1, 2006

## Understand the Problem

**Analyse** the problem.

Think about all the details. Is the problem fully specified?

For example, we want our program to say numerals **properly**, not like telephone numbers etc.

Do we want it to work for *all* integers? No matter how big? Negative as well as positive?

There is no *right* answer to these questions. It's up to the person **specifying the requirements** — that is, the “customer.”

For this exercise, we will only deal with positive numbers between 0 and 100.

(It will be fairly easy to extend, once we have finished.)

## How To Design a Function

We will go through some very basic ideas about how to come up with a function definition to solve a particular problem.

When learning to program, the biggest hurdle is knowing how and where to start.

**The running example will be to design and define a function to convert integers to English phrases corresponding to how we say numerals.**

(The script we are going to develop will be different to `NumWords.hs` used in Week 2 lab classes.)

## What is a good *name* for the function?

I chose `convert`.

Not a very informative name — can you suggest a better one?

## What is the Function's *Type*?

What are its inputs and outputs?

```
convert :: Int -> String
```

## Abstraction — Reducing Complexity

You can't figure it out all at once!  
This is the **KEY ACTIVITY** in software development!

1. Does the problem break down into smaller parts?
2. Is there a simpler version you can do first?
3. Have you seen a similar or related problem?

## What Tools Do We Have?

- **What does the programming language give you that might be useful?**
  - In Haskell, the Prelude functions and other libraries.
  - (This knowledge improves with practice.)
- **Do you know of any other programs or functions that may be similar or otherwise useful?**
  - I *routinely* cut and paste code from programs I have written earlier.
  - I *routinely* look at old programs for clues and details.

For our exercise, thinking about (1) and (2) in the list, we can start by working on the problem of converting *single digit numerals*: 0 to 9.

(Why do I think that will help? Common sense and experience ...)

## Test As You Go!

Writing a whole program means writing lots of different self-contained bits of code (functions) ...

because we have broken the problem down into simpler parts, or ...

because we have tackled a simpler problem first.

**Make sure those parts work *correctly* before proceeding!**

## ABSTRACTION

The key to reducing complexity is to focus on a well-defined and well understood sub-problem . . .

while temporarily ignoring the rest of the problem.

When we have solved that subproblem, we can forget about *how it works* and only think about *what it does*.