“Early in the morning, at the break of day, in all the freshness and dawn of one’s strength, to read a book – I call that vicious!”

Friedrich Nietzsche, *Ecce Homo*, 1908
Computers are the most complex objects human beings have ever created.

But, in a fundamental sense, they're remarkably simple.

If you drew a full wiring diagram for a modern computer, it could fill all the books in a library.

No one would have the patience to read it – and no one needs to! Why?

Such a diagram is unnecessary because of the regularity of a computer's design.

Computers are built up in a hierarchy of parts, with each part repeated many times over.

All you need to understand a computer is an understanding of this hierarchy.

Another principle that makes computers easy to understand is the nature of interactions among parts:

Simple, well-defined

Usually one-directional – either a cause or an effect

Because of this, the inner workings of a computer are more comprehensible than, say, a car engine.
The ideas of computers have almost nothing to do with the electronics they’re built out of.

Present day computers are made out of transistors and wires.
They could be built, according to the same principles, from valves and water pipes...

...or sticks and strings.

The principles are the essence of what makes a computer compute.

One of the most remarkable things about computer is that their essential nature transcends technology.

The structure of the computer is an example of applying functional abstraction over and over again.

We can focus on what’s happening at one level of the hierarchy without worrying about the details of what’s going on at lower levels.

Functional abstraction is what decouples the ideas from the technology.
With the right programming, a computer is
  a movie theater
  a musical instrument
  a reference book
  a chess opponent
  ...
No other entity in the world has such an adaptable, universal nature – except a human.

The magic of a computer is its ability to become almost anything you can imagine...
...as long as you can explain exactly what that is.

A programming language is a set of building blocks for constructing computer software.
  Like a human language, a programming language has a vocabulary and a grammar.
  How's it different?
We use a programming language to write programs that tell a computer what to do.

Why should that be hard?

Everything needs to be described precisely.

Say you tell an accounting program to bill your clients the amount each owes.

Should the computer send a weekly bill for $0.00 to clients who owe nothing?

If you tell the computer to send a threatening letter to clients who haven’t paid, then clients who owe nothing will receive threatening letters until they send you a payment of $0!

There are many programming languages due to

- history
- habit
- taste

But different languages are also good at describing different kinds of things.
Each language has its own syntax you need to learn.

But the syntax – like spelling and punctuation in human language – isn’t fundamental to the expressive power of the language.

What’s important to the expressive power are the vocabulary – the primitive of the language – and the way they can be combined to define new concepts.

The earliest computer languages were designed primarily to manipulate numbers and sequences of characters.

Latter-day programming languages can manipulate words, pictures, sounds, and even other computer programs.

Programming languages provide ways of

- reading the data’s elements into the computer
- taking the data apart
- putting them together
- modifying them
- comparing them
- giving them names.
Although Logo is simple enough to be used by a ten-year-old, it embodies many of the features of the most sophisticated computer languages. Including the ability to write programs that manipulate other programs!
Drawing pictures this way is fun but a lot of typing. Here’s where it gets interesting: We can define new words.

Once the word “square” has been defined, it becomes part of the computer’s vocabulary and can then be used to define other words, e.g.,

```plaintext
to window
  to square
  square
  square
  square
end
```
What if we want *big* squares and *little* squares?

The only thing that needs to change is the size, “10”

to square :size
    forward :size
    right 90
    forward :size
    right 90
    forward :size
    right 90
    forward :size
end

Other programs can call square; it doesn’t matter whether it’s a user-defined word or one of the language’s primitives.

In extending the language, the programmer uses the power of functional abstraction to *create new building blocks.*

What if we insert a word inside its own definition?

to design
    square
    right 10
    design
end

design
Cool, right?
Not cool.

The recursive definition of “design” has a problem: it goes on forever.

A guru claimed the Earth was sitting on the back of a giant turtle.

“And what is the turtle sitting on?” asked a student.

“Another turtle”, replied the guru.

“And that turtle?” asked the student, beginning to grow skeptical.

“It’s no use asking”, said the guru. “It’s turtles all the way down.”

This particular infinite loop is easily avoided by writing the program with a parameter specifying how many squares to draw:

to design :number
  square
  right 10
  if :number = 1 stop else design :number -1
end
Recursion is powerful

Recursive definitions are convenient for specifying operations on recursive data.

Many of the types of data we like to manipulate—in particular, computer programs themselves—have recursive structures.

The typical recursive definition has two parts:

What’s to happen in a particular simple case

How a more complex case can be reduced to something simpler

Other programming languages differ from Logo in details of vocabulary and syntax, but they can all express the same kinds of procedures.

In the rest of this course, we’ll focus on a language called Racket.

But you can imagine turtles.
The computer is a device that accelerates and extends our processes of thought.

The true power of the computer is that it is capable of manipulating not just the expression of ideas, but also the ideas themselves.

A computer can hold the contents of all the books in a library — but it can also notice relationships between the concepts described in the book!