

Offset	Topic
00:17	<ul style="list-style-type: none"> • Intro
	<ul style="list-style-type: none"> • Giving thanks
04:41	<ul style="list-style-type: none"> • Listener Feedback
	<ul style="list-style-type: none"> • Katy suggest Software Engineering Radio • Natalie talks about sensor meshes • James talks about the cycle of up and down moods
11:39	<ul style="list-style-type: none"> • Word of the Week: core
	<ul style="list-style-type: none"> • http://www.catb.org/jargon/html/C/core.html
12:35	<ul style="list-style-type: none"> • Inner Chapter: Programming Paradigms 1
	<ul style="list-style-type: none"> • Programming at its simplest <ul style="list-style-type: none"> • Data and instructions • In fact, CPU caches come in exactly those two flavors • A processor has elements, which execute instructions <ul style="list-style-type: none"> • Direct instructions for CPU are machine code • Elements typically do math, similar low level operations • Arithmetic Logic Unit, Floating Point Unit • CPU has registers, that hold data <ul style="list-style-type: none"> • Additional instructions read, store data • To and from registers, to and from memory, disk • Data can be simple, just data, or complex • Complex data is a struct, usually named fields whose value is simple or another struct • Some flavors of assembly introduce structs, more commonly seen in higher level languages • Higher refers to language being closer to the way people think about problems, solutions • For more on CPU architecture basics, see Writing Great Code, Volume 1 • With these parts alone, programs would be highly repetitive, error prone • Flow of control instructions add decisions <ul style="list-style-type: none"> • Some CPUs support directly, like a conditional jump • Usually these are added with languages above machine code • Simplest is the if statement • Next simplest and rarely exposed directly is goto <ul style="list-style-type: none"> • Arbitrary goto is hard to maintain because it lacks context • Goto just jumps to some arbitrary point • Specifying where to go is where context is useful • Next flow hackers learn after if, loop, is just a special case of an if and a goto

- Loops goto top or bottom of some arbitrary set of instructions
- They also introduce the idea of scope, a block of instructions handled together for flow
- Even with flow of control, code can be very expansive, repetitive, error prone
- Procedural programming
 - What do you do when you want to repeat the same set of operations?
 - Could do with a goto but, again, lacks context
 - Doesn't reveal your intent for goto, also relies on correct return with another goto
 - A procedure is a special block of code, a scope, that can be called by name
 - Procedures take arguments, or parameters, to work on
 - At their simplest, they return a result
 - For instance a square root procedure takes a number and returns its root
 - For some languages, they can alter the arguments passed in, this is called a side effect
 - Side effect relies on method of passing arguments, either by value or by reference
 - By value creates a copy so changes are thrown away when the procedure returns
 - By reference means the procedure is working on the same data that the calling code has
 - Changes by reference outlive procedure call
 - The point is the procedure gives first real tool for organizing code
 - When I have talked about functional decomposition, it is merely the decisions on what procedures to write
- Procedural introduced first discussion of paradigms
 - Also can be contrasted to functional programming
 - Functional is essentially a mathematical approach
 - Variables, that is pointers to data, cannot be changed
 - Calls to functions always produce a result, never a side effect
 - Just like the way you write out mathematical formulae, no concept of changing a value, always producing something new
- To me, different paradigms are like schools of art
 - Some sense of increasing sophistication with each new one
 - Not necessarily, though
 - Often the same scene or subject can be addressed by different schools with similar effect
 - Sometimes the differences are substantive, others not
 - Newer schools often revive techniques, ideas, from older schools
 - The school you are taught often shapes your approach, even when you learn other schools

Offset

Topic

- These statements are also generally true of programming paradigms
- Flexibility of approach is even truer, because of universal aspect of computers
- Any universal program, e.g. a programming language interpreter or compiler, can emulate any other
- There are often differences in performance, efficiency but the point holds
- Once I realized this analogy, I actually relaxed considerably about what paradigms I know, use
- Found it easier to be open minded
- Much of the fundamentals, like data, procedures, flow of control and basic operations are the same for all
- Will explore object oriented and aspect oriented programming in future installments

36:50

• **Outro**

- Contact me
 - Email to feedback@thecommandline.net
 - Web site at <http://thecommandline.net/>
 - IM to command.line@skype
 - Listener comment line is 240-949-2638
 - del.icio.us tag is "for:cmdln"
 - <http://twitter.com/cmdln>
- I'd like to thank libsyn.com for AAC hosting and Wouter de Bie for MP3 hosting
- These notes and the show audio and music are covered by a Creative Commons license
 - <http://creativecommons.org/licenses/by-nc-sa/3.0/us/>
 - Attribution, non-commercial, share alike