Beyond the Mouse – A Short Course on Programming

2. Fundamental Programming Principles I: Variables and Data Types

Ronni Grapenthin

Geophysical Institute, University of Alaska Fairbanks

September 17, 2009

"The Uncomfortable Truths Well", http://xkcd.com/568 (April 13, 2009)

Today's schedule ...

1. Solutions to Exercises
2. How does programming work?
3. Variables and Datatypes

Today's schedule

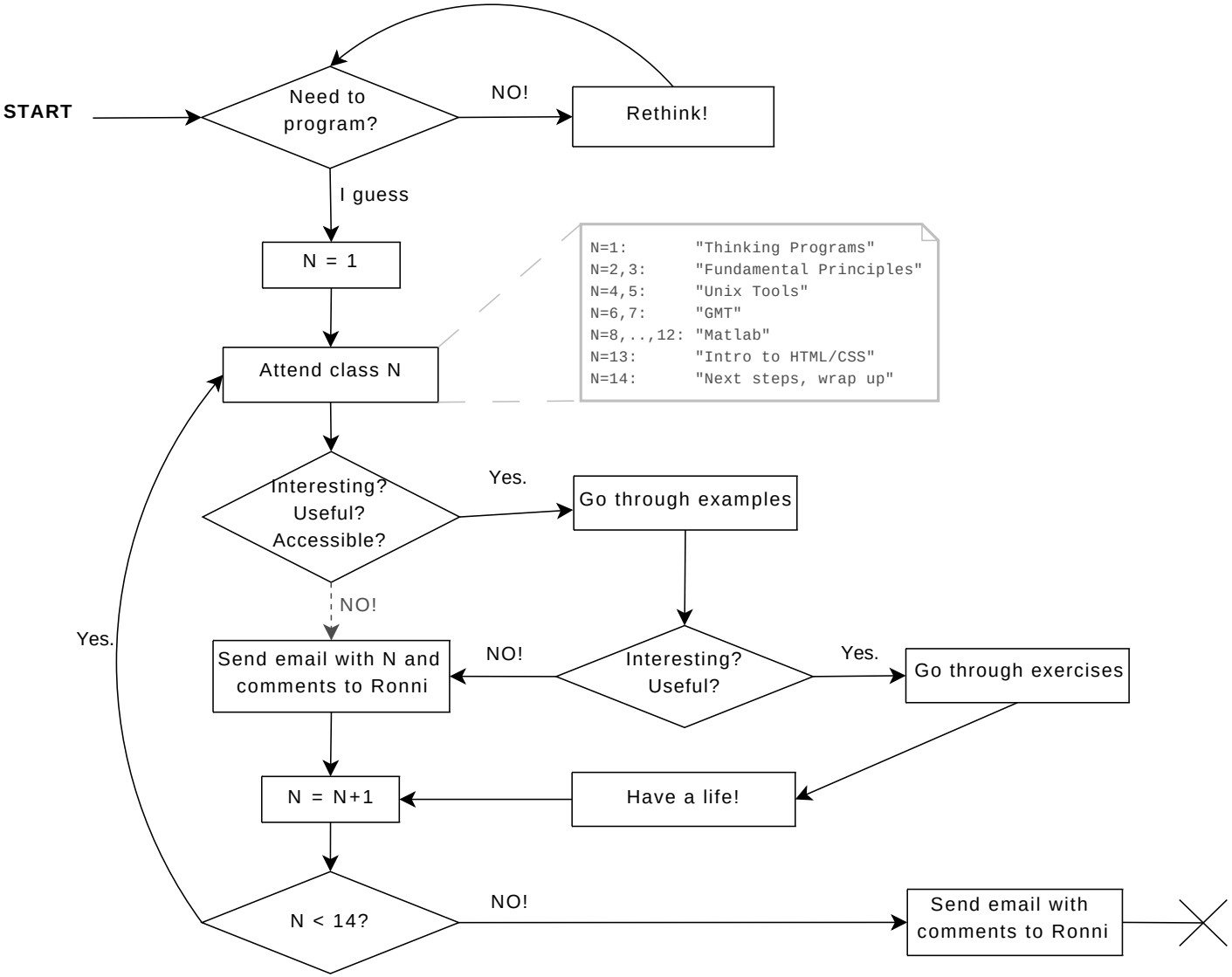
1. Solutions to Exercises
2. How does programming work?
3. Variables and Datatypes

Exercise #0

. . .

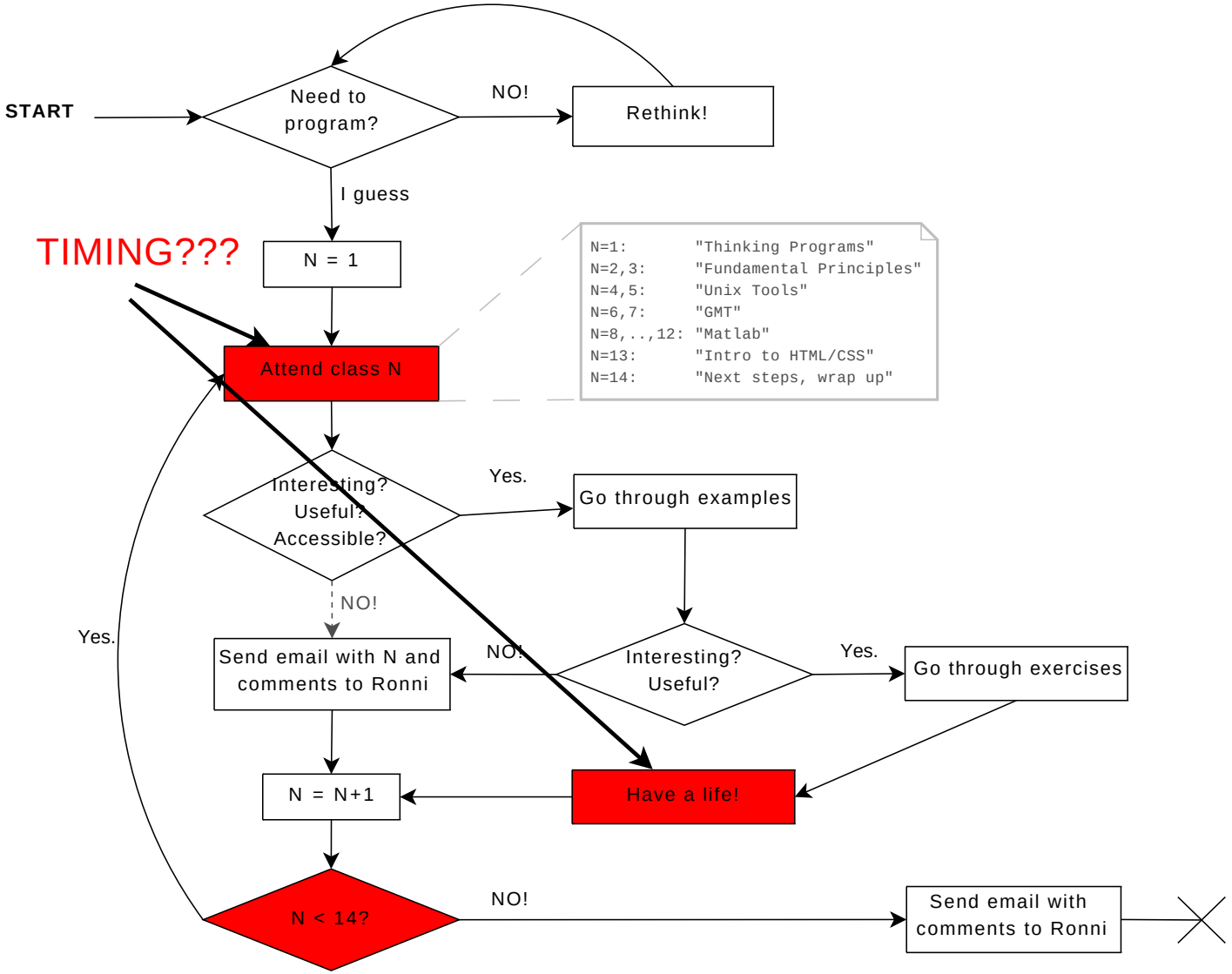
thanks for the images :)

Exercise #1



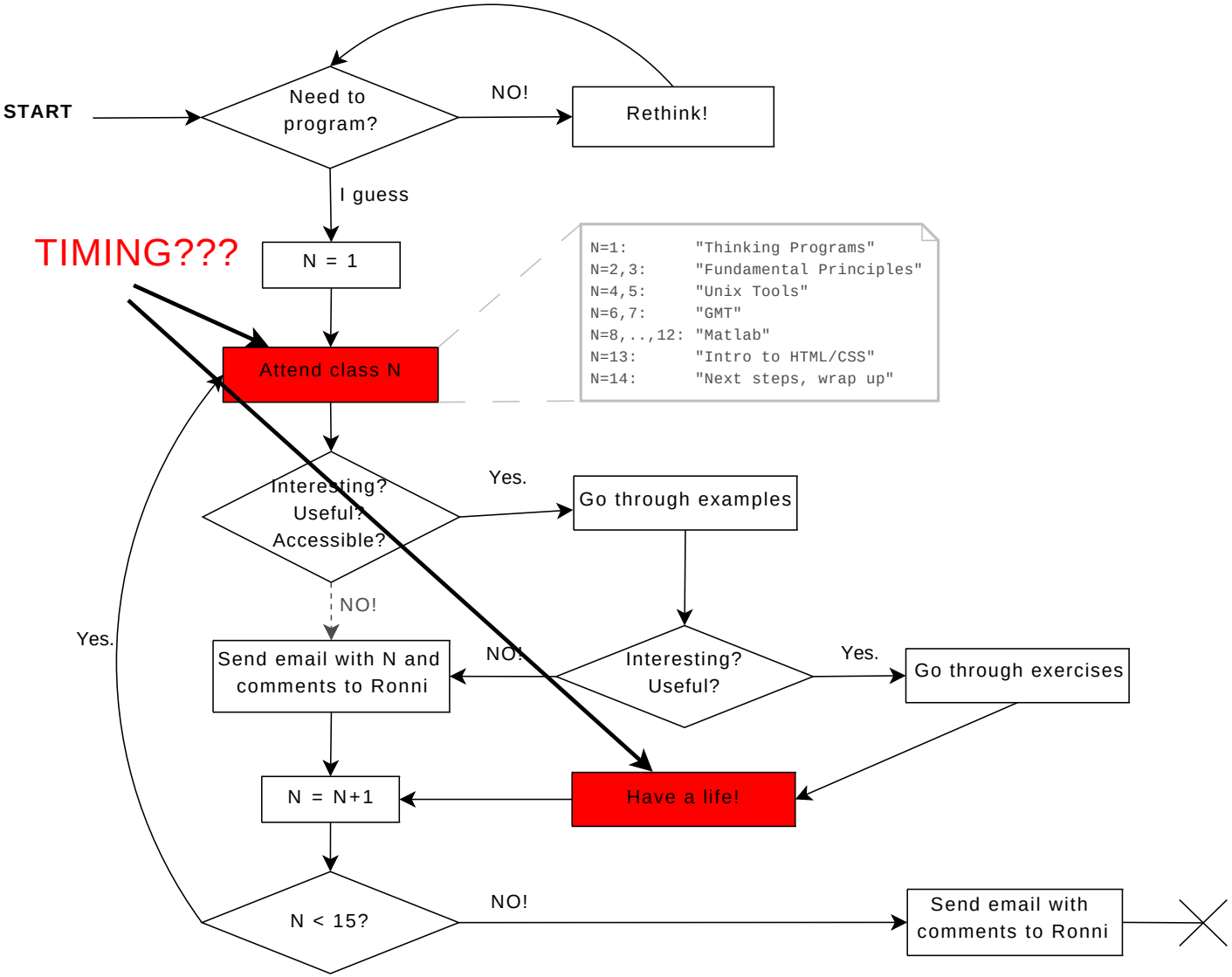
Listing 1: Seminar flow (fixed)

Exercise #1



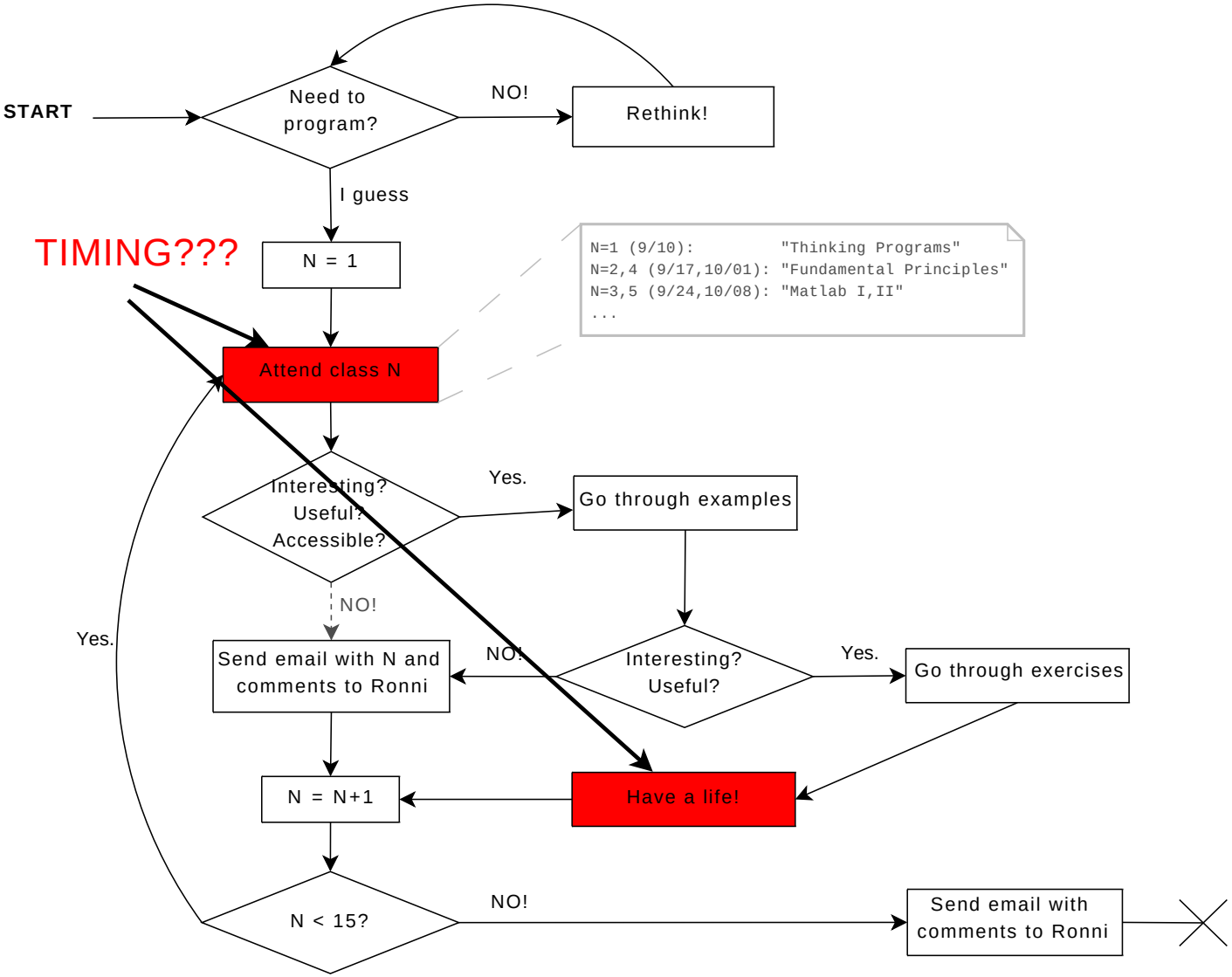
Listing 1: Seminar flow (fixed)

Exercise #1



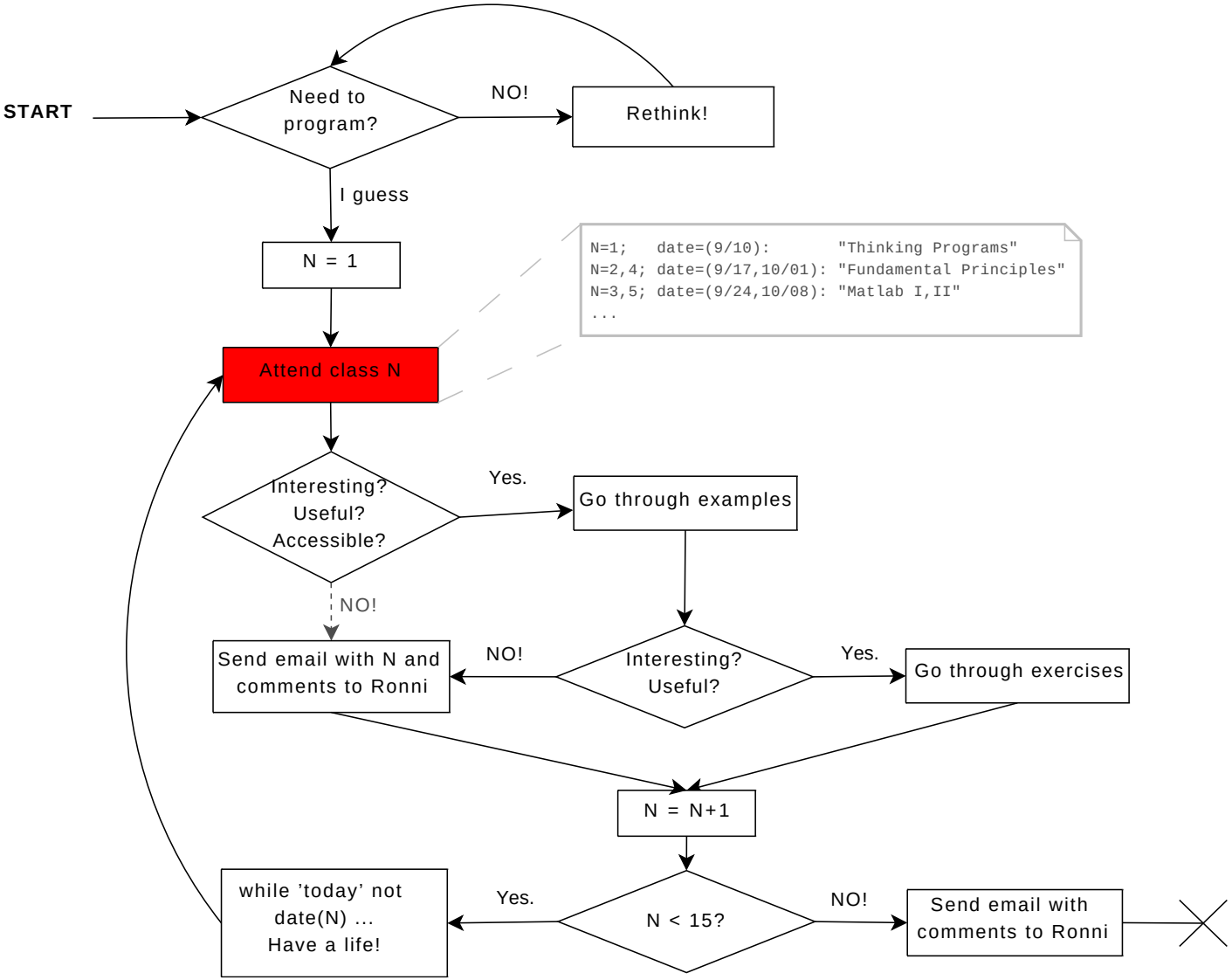
Listing 1: Seminar flow (fixed)

Exercise #1



Listing 1: Seminar flow (fixed)

Exercise #1



Listing 1: Seminar flow (fixed)

Exercise in general

I'll come back to you with individual comments on project snapshots, flow charts etc.

Today's schedule

1. Solutions to Exercises
2. How does programming work?
3. Variables and Datatypes

How does programming work?

Well, fist we should clearify terminology here!

What is a programming language?

What is a program?

Alright, what is it then?

Definitions (broad sense)

A **programming language** is an unambiguous artificial language that is made up of a set of symbols (vocabulary) and grammatical rules (syntax) to instruct a machine.

A **program** is a set of instructions in one or multiple programming languages that specifies the behavior of a machine.

Compilation or **interpretation** is the verification of a program and its translation into in the machine readable instructions of a specific platform.

Programming languages . . . continued

Two broad families can be identified:

1. **Interpreted languages**
   An interpreter program is necessary to take in commands, check syntax and translate to machine language at runtime (e.g., Matlab, Unix Shell)

Programming languages . . . continued

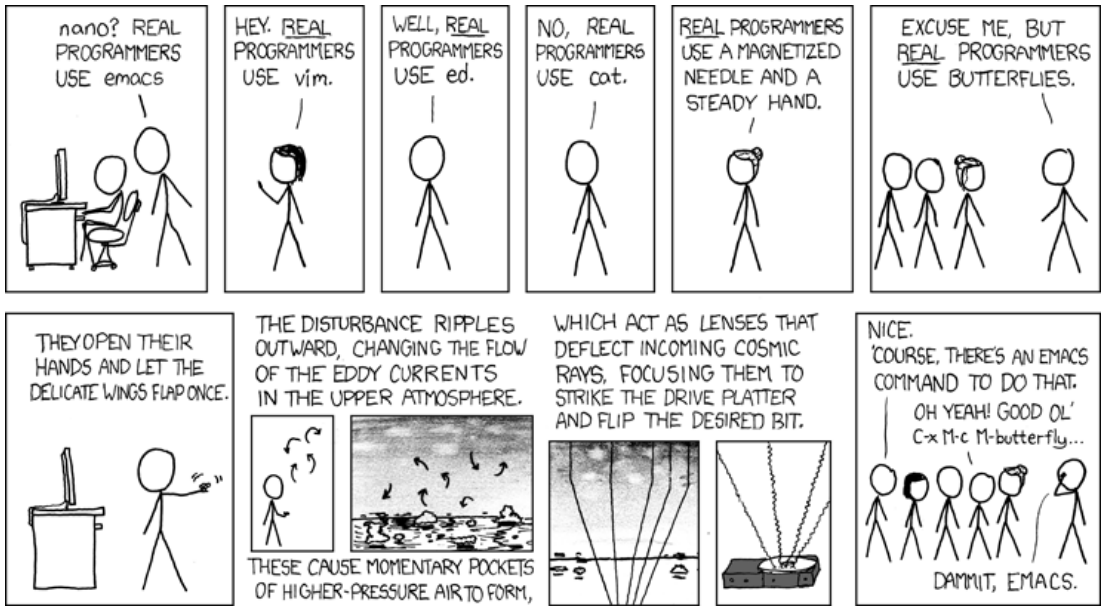
Two broad families can be identified:

1. **Interpreted languages**
   An interpreter program is necessary to take in commands, check syntax and translate to machine language at runtime (e.g., Matlab, Unix Shell)
2. **Compiled languages**
   Programs are translated and saved in machine language. At runtime no additional program is necessary (e.g., C/C++).

Now, how does programming work?

Now, how does programming work?

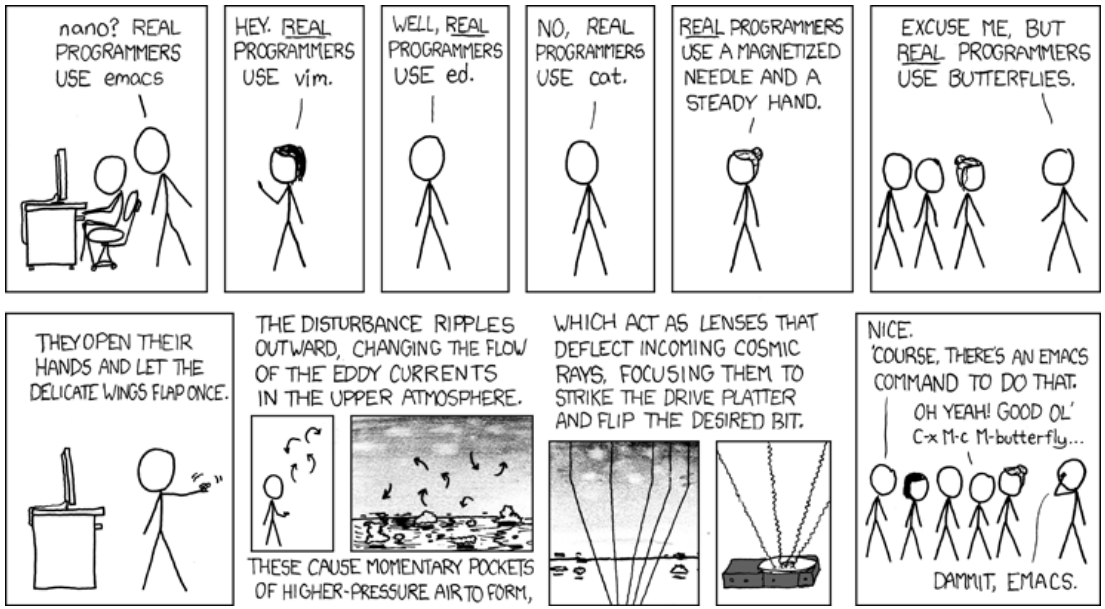
1. open **text** editor (vi, notepad, . . . , not MS Word!)



http://www.xkcd.com/378/

Now, how does programming work?

1. open **text** editor (vi, notepad, . . . , not MS Word!)

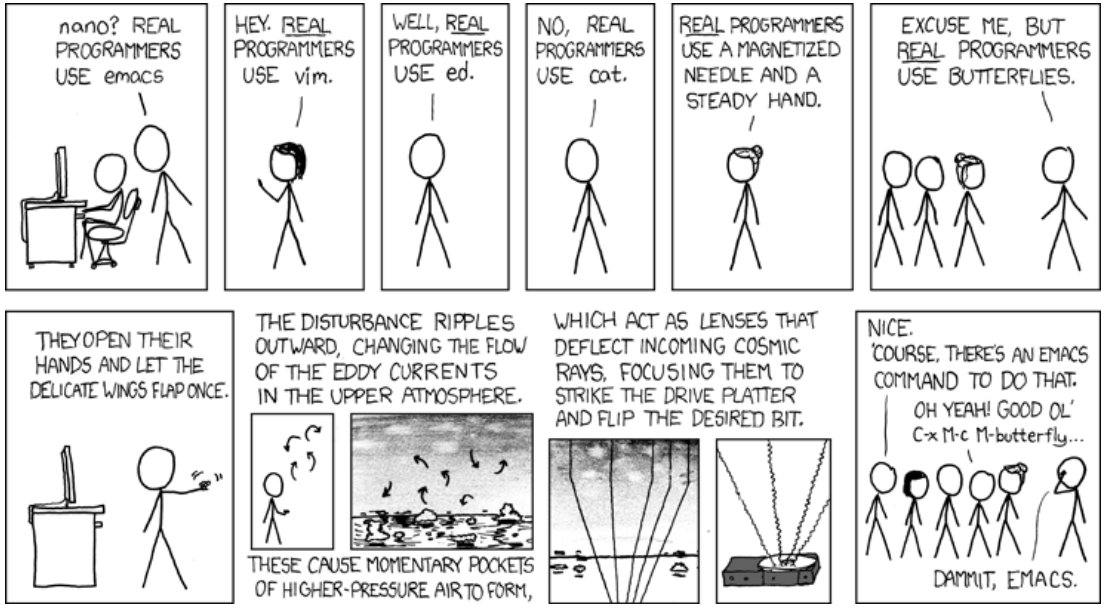


http://www.xkcd.com/378/

2. translate your (mental) flowchart into a set of instructions according to the rules of an applicable programming language

Now, how does programming work?

1. open **text** editor (vi, notepad, . . . , not MS Word!)

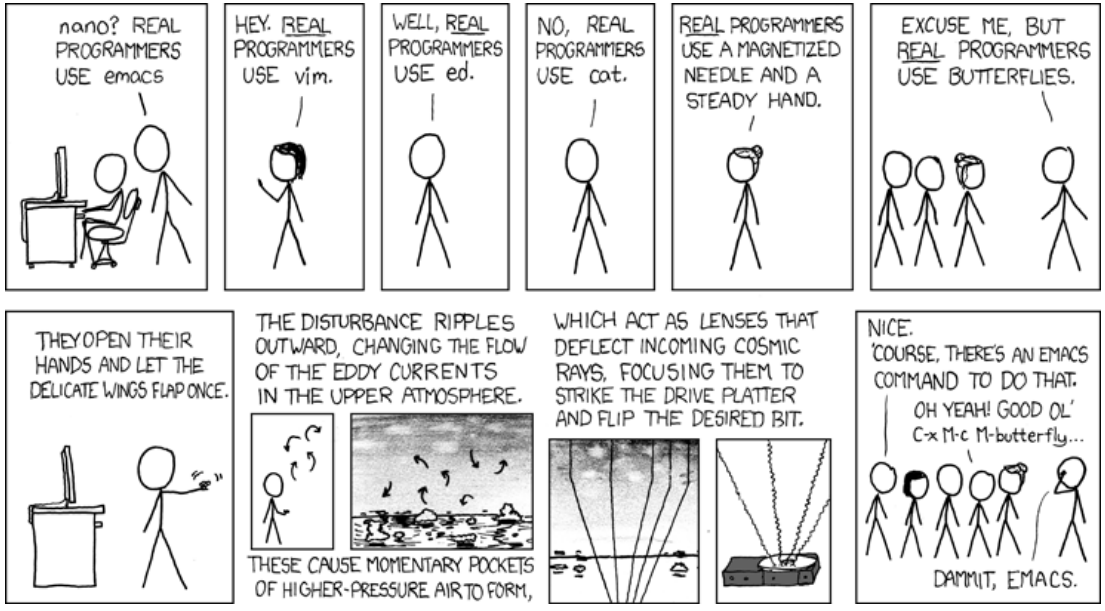


http://www.xkcd.com/378/

2. translate your (mental) flowchart into a set of instructions according to the rules of an applicable programming language
3. test your program for syntactical correctness (ask interpreter/compiler)

Now, how does programming work?

1. open **text** editor (vi, notepad, . . . , not MS Word!)

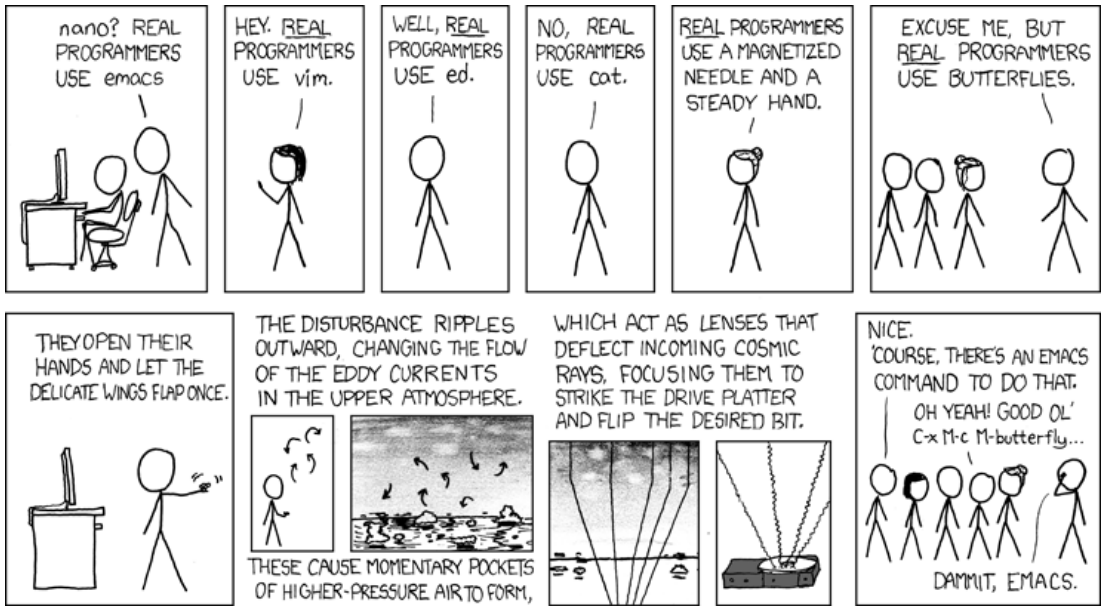


http://www.xkcd.com/378/

2. translate your (mental) flowchart into a set of instructions according to the rules of an applicable programming language
3. test your program for syntactical correctness (ask interpreter/compiler)
4. if errors, fix them and go back to (3)

Now, how does programming work?

1. open **text** editor (vi, notepad, . . . , not MS Word!)

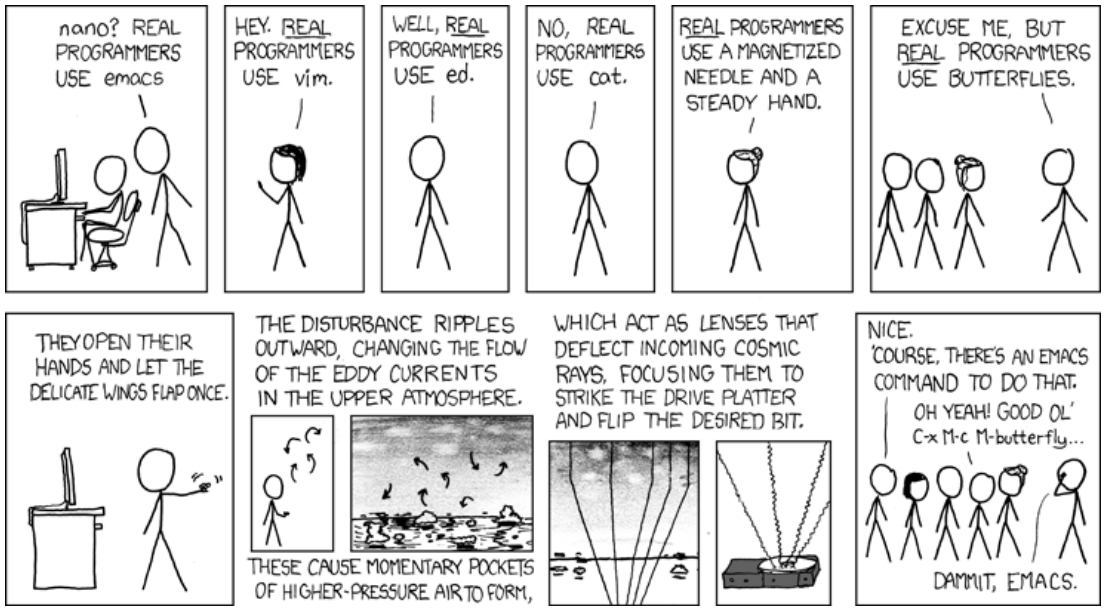


http://www.xkcd.com/378/

2. translate your (mental) flowchart into a set of instructions according to the rules of an applicable programming language
3. test your program for syntactical correctness (ask interpreter/compiler)
4. if errors, fix them and go back to (3)
5. test your program for semantical errors (the fun part!)

Now, how does programming work?

1. open **text** editor (vi, notepad, . . . , not MS Word!)



http://www.xkcd.com/378/

2. translate your (mental) flowchart into a set of instructions according to the rules of an applicable programming language
3. test your program for syntactical correctness (ask interpreter/compiler)
4. if errors, fix them and go back to (3)
5. test your program for semantical errors (the fun part!)
6. if errors, fix them and go back to (3)

Don’t even think that’s as simple as it sounds . . .

‘Hello World’ in Matlab

```
>> dsp(halo orld
??? dsp(halo orld
               |
Error: Unexpected MATLAB expression.

>> dsp('halo orld
??? dsp('halo orld
        |
Error: A MATLAB string constant is not terminated properly.

>> dsp('halo orld'
??? dsp('halo orld'
                   |
Error: Expression or statement is incorrect--possibly unbalanced (, {, or [.

>> dsp('halo orld')
??? Undefined function or method 'dsp' for input arguments of type 'char'.

>> disp('halo orld')
halo orld

% Sematically correct, if you want to say 'hi' to the world:
%
>> disp('hello world')
hello world
```

Listing 2.1: hello_world.log

Today's schedule

1. Solutions to Exercises
2. How does programming work?
3. Variables and Datatypes

Variables (1)

Definitions – a selection

Donald Knuth: A quantity that may possess different values as a program is being executed.
Mehran Sahami: A box in which we stuff things – i.e. a box with variable content.
Wikipedia: User defined keyword that is linked to a value stored in computer's **memory** (runtime).

The concept of a **variable** consists of:

Variables (1)

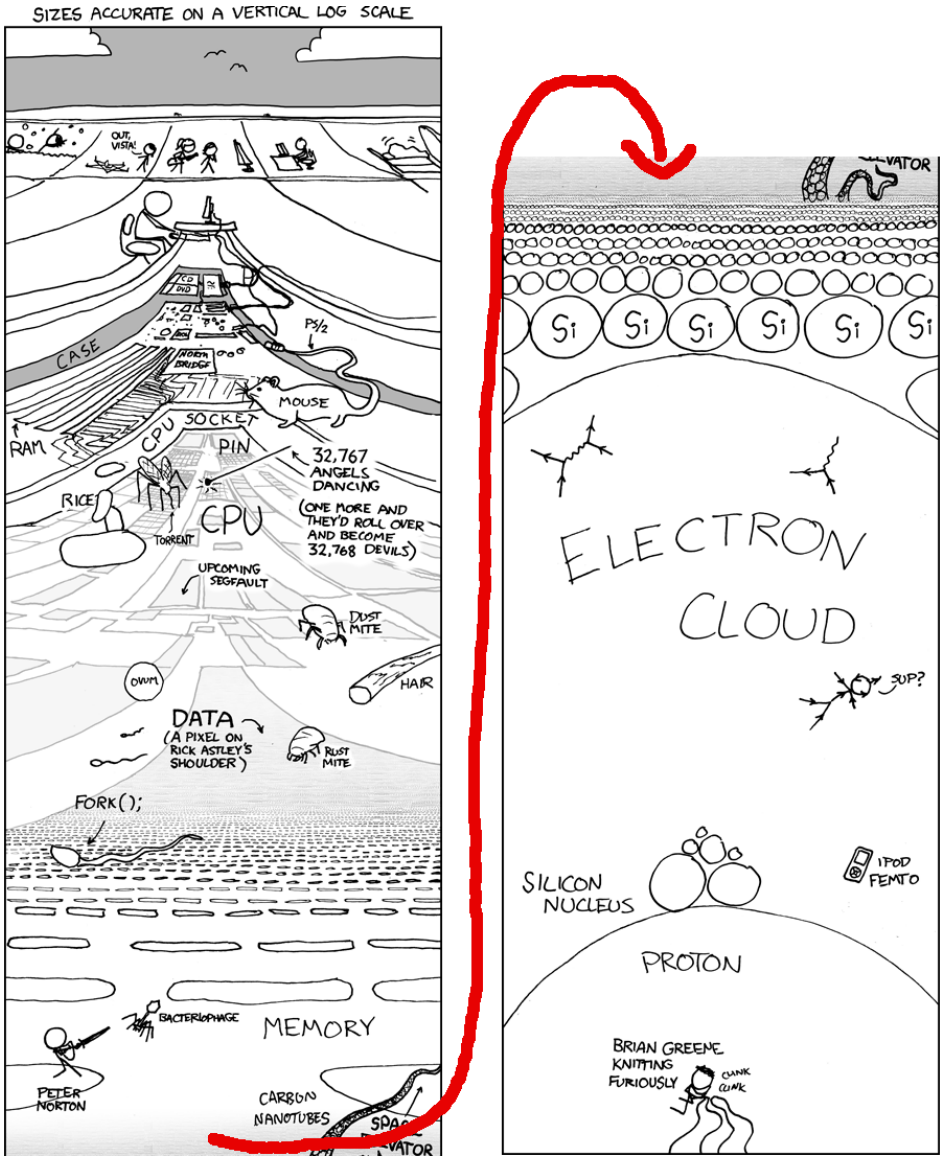
Definitions – a selection

Donald Knuth: A quantity that may possess different values as a program is being executed.
Mehran Sahami: A box in which we stuff things – i.e. a box with variable content.
Wikipedia: User defined keyword that is linked to a value stored in computer's **memory** (runtime).

The concept of a **variable** consists of:

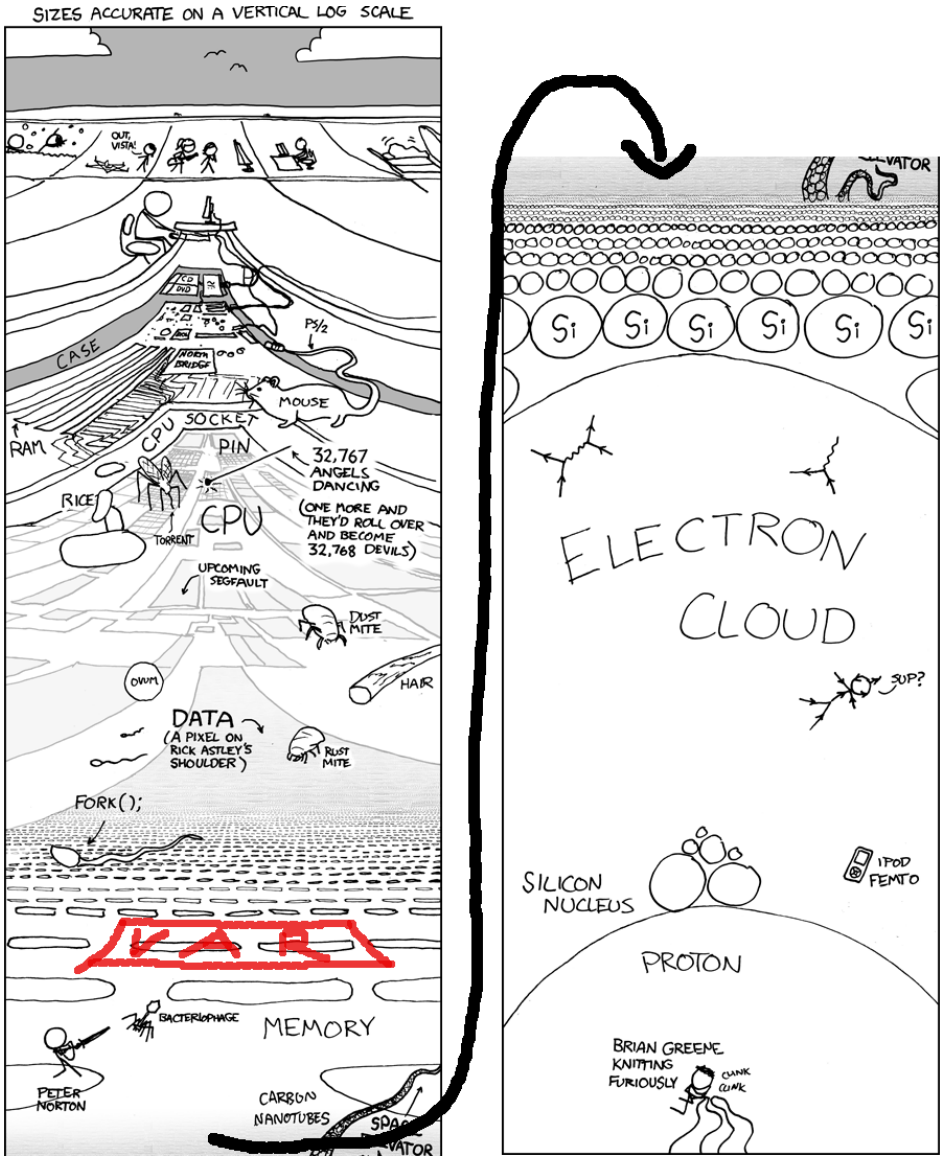
- name
- type
- value

Memory interlude



"Depth",
http://xkcd.com/485 (September 16, 2009)

Memory interlude



"Depth",
http://xkcd.com/485 (September 16, 2009)

Variables (2) – name

- USE MEANINGFUL NAMES!

Variables (2) – name

- USE MEANINGFUL NAMES!
- valid, follow programming language rules – MATLAB variable names must **begin with a letter**, followed by any **combination of letters, digits, and underscores**. MATLAB distinguishes between uppercase and lowercase. No reserved keywords!

Variables (2) – name

- USE MEANINGFUL NAMES!
- valid, follow programming language rules – MATLAB variable names must **begin with a letter**, followed by any **combination of letters, digits, and underscores**. MATLAB distinguishes between uppercase and lowercase. No reserved keywords!
- USE MEANINGFUL NAMES, i.e. names that speak:

Variables (2) – name

- USE MEANINGFUL NAMES!
- valid, follow programming language rules – MATLAB variable names must **begin with a letter**, followed by any **combination of letters, digits, and underscores**. MATLAB distinguishes between uppercase and lowercase. No reserved keywords!
- USE MEANINGFUL NAMES, i.e. names that speak:
- '`lengthGlacier`' or '`glacier_length`' **NOT NOT NOT** '`a`' – avoid ambiguity

Variables (2) – name

- USE MEANINGFUL NAMES!
- valid, follow programming language rules – MATLAB variable names must **begin with a letter**, followed by any **combination of letters, digits, and underscores**. MATLAB distinguishes between uppercase and lowercase. No reserved keywords!
- USE MEANINGFUL NAMES, i.e. names that speak:
- ‘`lengthGlacier`’ or ‘`glacier_length`’ **NOT NOT NOT** ‘`a`’ – avoid ambiguity
- use consistent formatting, i.e.: ‘`my_cool_var`’ vs. ‘`myCoolVar`’ – supports reading

Variables (2) – name

- USE MEANINGFUL NAMES!
- valid, follow programming language rules – MATLAB variable names must **begin with a letter**, followed by any **combination of letters, digits, and underscores**. MATLAB distinguishes between uppercase and lowercase. No reserved keywords!
- USE MEANINGFUL NAMES, i.e. names that speak:
- ‘`lengthGlacier`’ or ‘`glacier_length`’ **NOT NOT NOT** ‘`a`’ – avoid ambiguity
- use consistent formatting, i.e.: ‘`my_cool_var`’ vs. ‘`myCoolVar`’ – supports reading
- a gazillion style guides exist – punchline: use meaningful names, be consistent (that’s hard enough)!

Variables (3) – type

What is a type? – Think of sets of numbers in math: $\mathbb{N}, \mathbb{R}, \mathbb{Z}, \ldots$ The type refers to how numbers are being represented in a computer's memory, i.e. which bit has which meaning.

Variables (3) – type

What is a type? – Think of sets of numbers in math: $\mathbb{N}, \mathbb{R}, \mathbb{Z}, \ldots$ The type refers to how numbers are being represented in a computer's memory, i.e. which bit has which meaning.

Two kinds of Types

- primitive, built in types – for MATLAB e.g.: '`int32`', '`double`', '`boolean`'
- complex, home made types – (arrays,) structs, cell arrays (Matlab), classes/objects

Variables (3) – type

What is a type? – Think of sets of numbers in math: $\mathbb{N}, \mathbb{R}, \mathbb{Z}, \ldots$ The type refers to how numbers are being represented in a computer's memory, i.e. which bit has which meaning.

Two kinds of Types

- primitive, built in types – for MATLAB e.g.: '`int32`', '`double`', '`boolean`'
- complex, home made types – (arrays,) structs, cell arrays (Matlab), classes/objects

Types in Programming Languages

- some languages, e.g. MATLAB, Shells, Perl are weakly typed: implicit type conversions (OR one type can be treated as another)
- this is nice at first, occasionally this leads to nasty/hard to fix problems (e.g. string interpreted as number, etc.)

Variables (4) – value

Value

- a value of the type of the variable: `23`, `3.1415926...`, `false`
- i.e., the thing we stuff in the box
- can/should change during the runtime of the program, otherwise use a constant (if possible)

Variables (4) – value

Value

- a value of the type of the variable: `23, 3.1415926..., false`
- i.e., the thing we stuff in the box
- can/should change during the runtime of the program, otherwise use a constant (if possible)

Declaring a variable and Assigning a value:

In General: `(type) name = value;` or `(type) name = expression;`

Matlab: `myNewVar = 10;` **TC-Shell** (differs) `set myNewVar = 10;`
Access to the values (de-referencing):
Matlab: use `myNewVar;` **TC-Shell** (differs) use '`$`': `$myNewVar`

Variables (4) – value

Value

- a value of the type of the variable: `23, 3.1415926..., false`
- i.e., the thing we stuff in the box
- can/should change during the runtime of the program, otherwise use a constant (if possible)

Declaring a variable and Assigning a value:

In General: `(type) name = value;` or `(type) name = expression;`

Matlab: `myNewVar = 10;` **TC-Shell** (differs) `set myNewVar = 10;`
Access to the values (de-referencing):
Matlab: use `myNewVar;` **TC-Shell** (differs) use '`$`': `$myNewVar`

What's that?

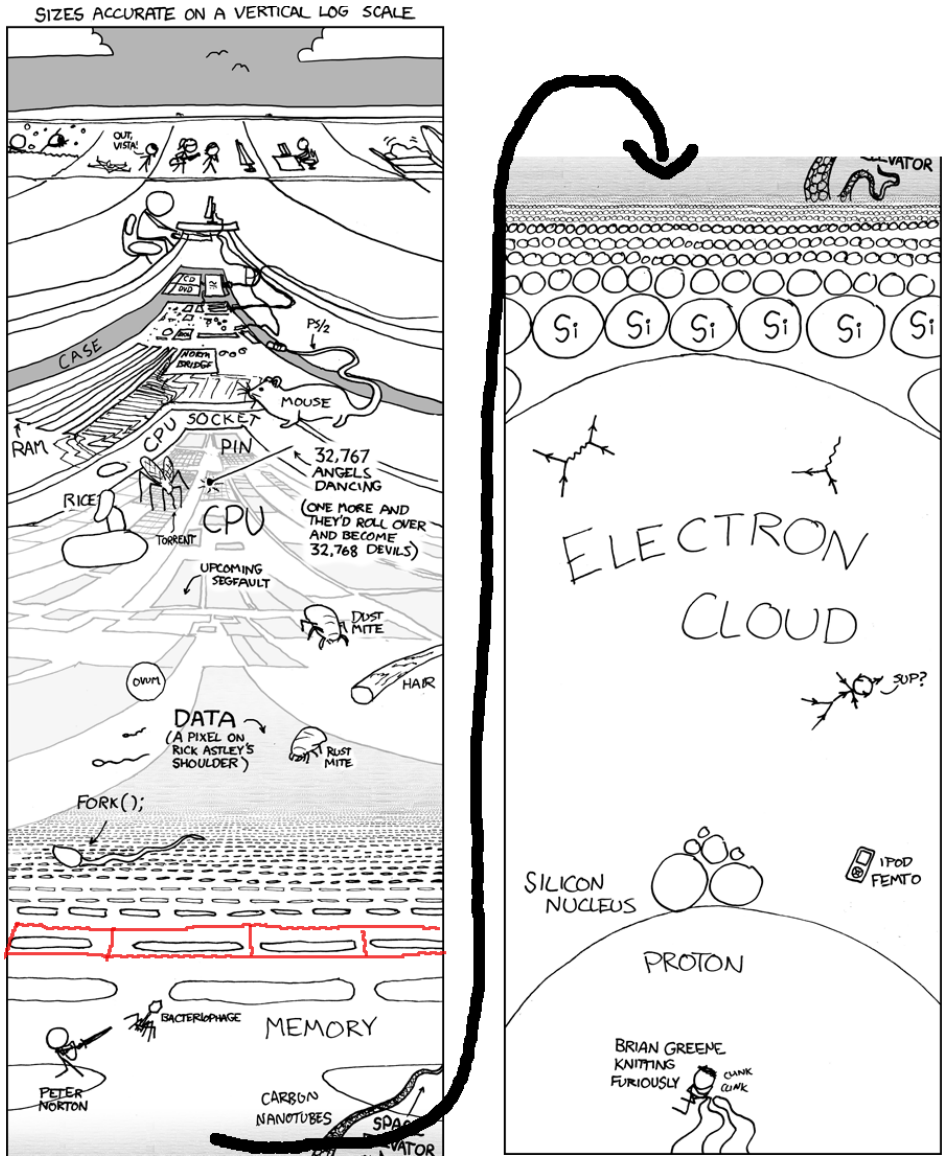
```
myNewVar = myNewVar + 1;
```

Advanced Variables: Vectors and Matrices (1)

Array variables

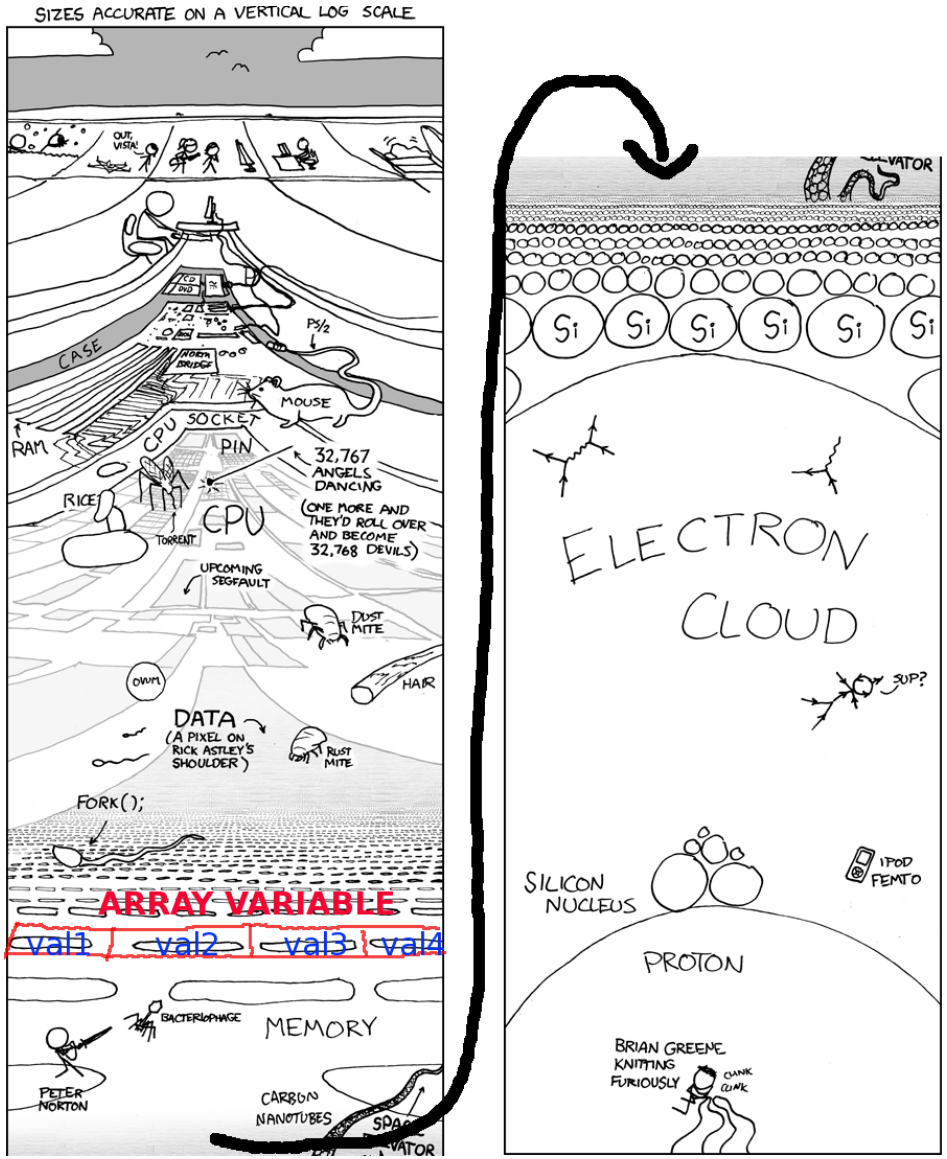
- are lists, vectors, matrices of data (1 to n dimensional – book keeping can become a hassle)
- therefore instead of one value they hold a **list of values**
- linked to a chunk of memory (a sequence of boxes)
- access by index number
- MATLAB treats everything as a matrix. Shells allow only vectors.

Memory interlude (2)



"Depth",
http://xkcd.com/485 (September 16, 2009)

Memory interlude (2)



"Depth",
http://xkcd.com/485 (September 16, 2009)

Advanced Variables: Vectors and Matrices (2)

Example

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
sting:	h	e	l	l	o		w	o	r	l	d	!	!	!	!
vector:	12	23.3	23.3	nan	nan	1	42	42.1	23	5	nan	nan	0	0	0

Advanced Advanced Variables: `struct` and Matlab `cell array` (1)

`structs, cells`

- organize and store data of different types in one variable
- these are containers you can put integers, doubles, strings, arrays of these, and other structs or cell arrays, etc ...

Advanced Advanced Variables: `struct` and Matlab `cell array` (1)

`structs, cells`

- organize and store data of different types in one variable
- these are containers you can put integers, doubles, strings, arrays of these, and other structs or cell arrays, etc ...

`cell`

- MATLAB specific data type, created using braces '{...}'
- must be used for strings of various length!
- elements are a vector in the memory – access `myCell(1)`, or `myCell{1,1}`
- can contain any other data type

`myCell`	{:,1}	{:,2}
{1,:}	[1 5]	[2.4 3.6; 4.3 1]
{2,:}	{'MacGyver' 'Bart'}	true

Access: `myCell(1)`, `myCell{1,1}` or as given in table.

Advanced Advanced Variables: `struct` and Matlab `cell array` (2)

`struct`

- more organized array type: access to fields by a name
- can contain any other data type
- excellent for representing your tables of data

`student`	(1)	(2)	(3)
.name	'Jack'	'Jo'	'Jake'
.age	21	25	30

Access: `student(1)`, `student.age`, `student.name`