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 19, 2011

YOU'LL NEVER FIND A PROGRAMMING LANGUAGE THAT FREES YOU FROM THE BURDEN OF CLARIFYING YOUR IDEAS.
BUT I KNOW WHAT I MEAN!

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

Today's schedule . . .

1. How does (computer) programming work?

2. Variables and Datatypes

Today's schedule

1. How does (computer) programming work?

2. Variables and Datatypes

How does (computer) 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 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

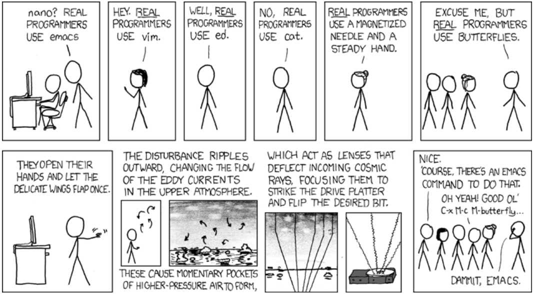
Two broad families can be identified:

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!)

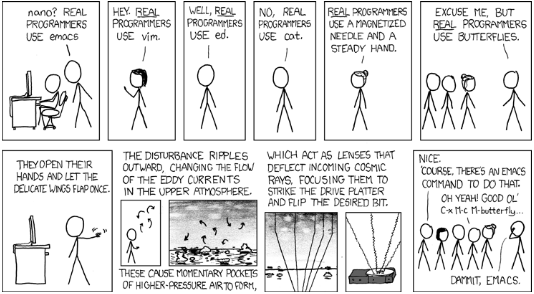


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

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

Now, how does programming work?

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

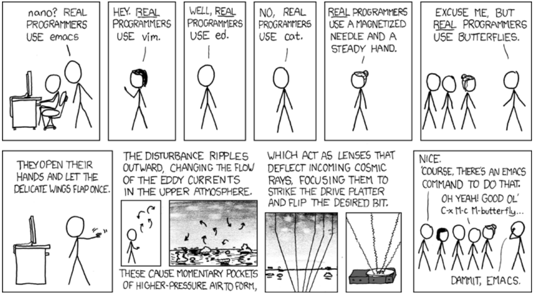

nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

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!)

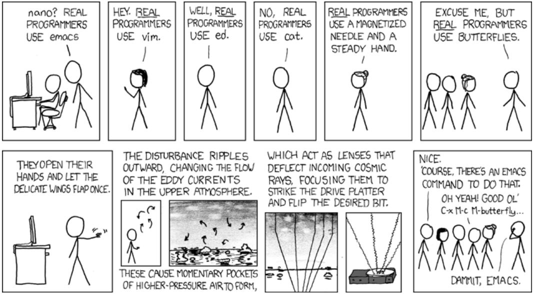

nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

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!)



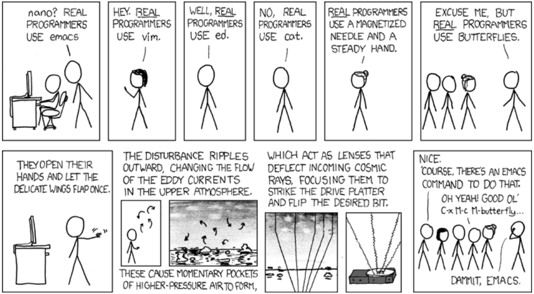
nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

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!)

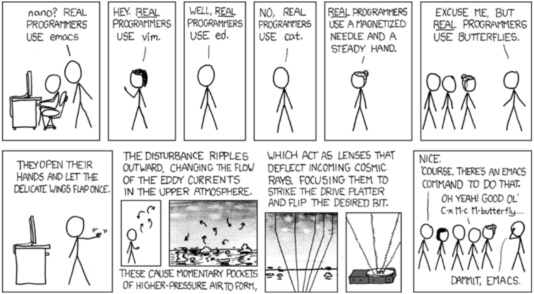

nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

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!)


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

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. How does (computer) programming work?

2. 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)

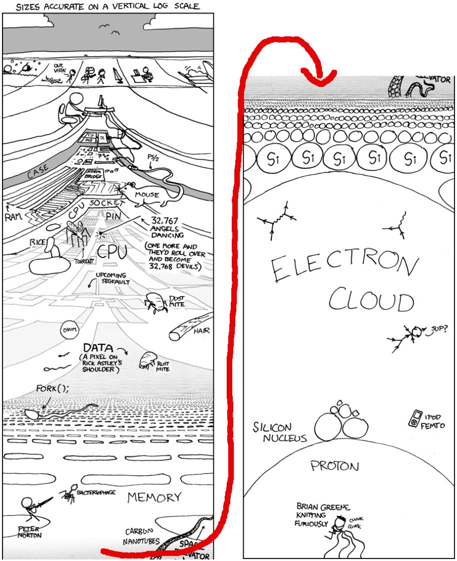
Definitions – a selection

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:

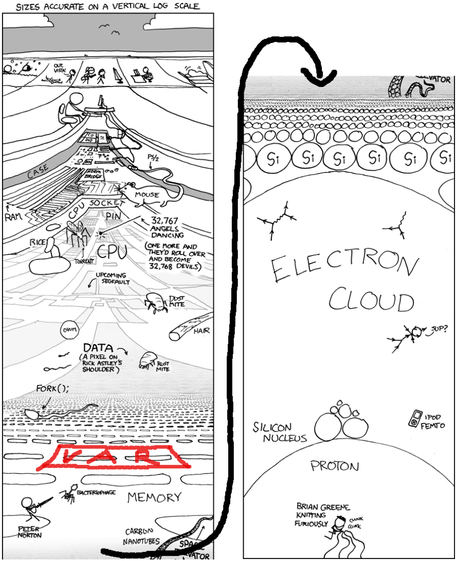
- name
- type
- value

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, and how many bits are necessary

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, and how many bits are necessary

Two kinds of Types

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

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, and how many bits are necessary

Two kinds of Types

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

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?

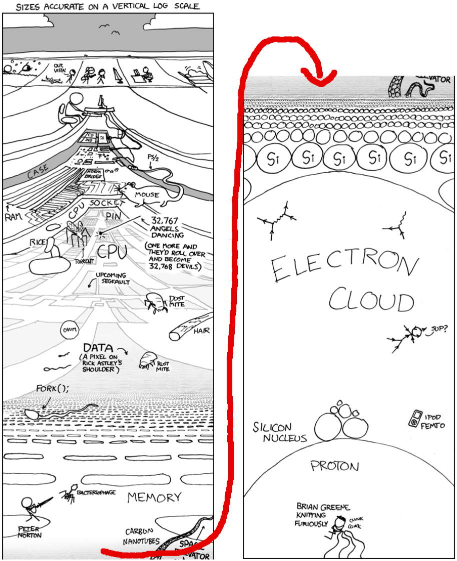
```
myNewVar = myNewVar + 1;
```

Advanced Variables: Vectors and Matrices (1)

Array variables

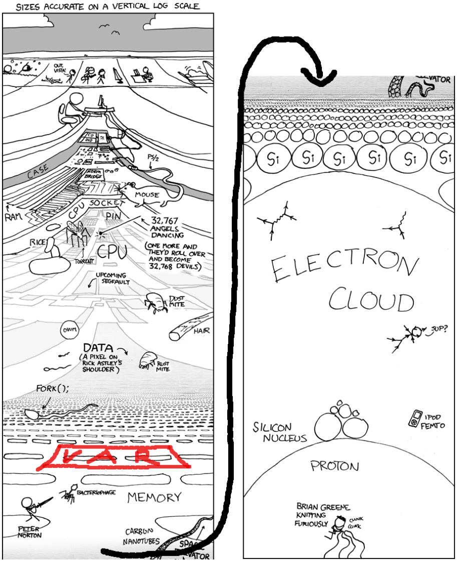
- are lists, vectors, matrices of data (1 to n dimensional – book keeping can become a hassle)
- therefore instead of one value they hold a **list of values**
- linked to a chunk of memory (a sequence of boxes)
- access by index number
- MATLAB treats **everything** as a matrix
- Shells allow only vectors.

Memory interlude (2)



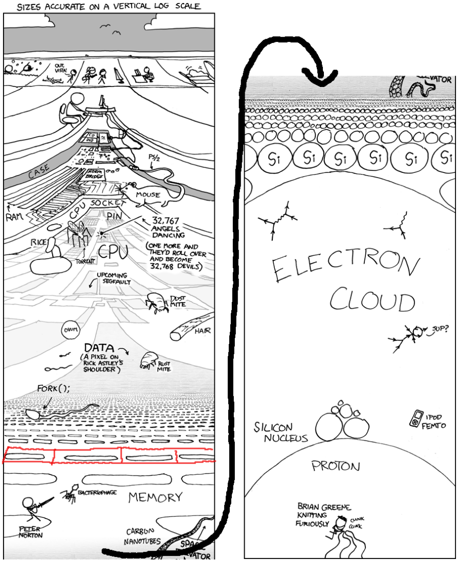
“Depth”,
http://xkcd.com/485 (September 16, 2009)

Memory interlude (2)



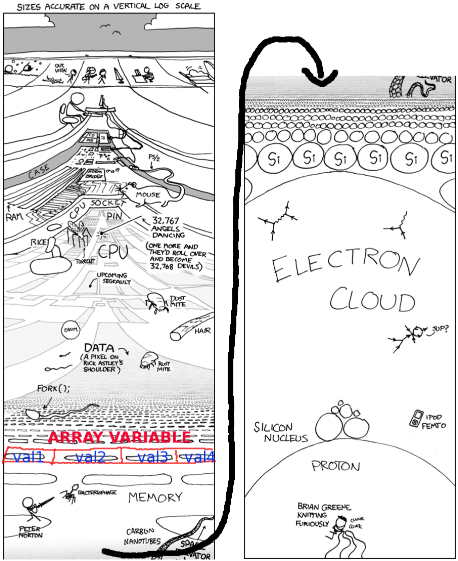
"Depth",
http://xkcd.com/485 (September 16, 2009)

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: Numeric Vector

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
vector:	12	23.3	23.3	nan	nan	1	42	42.1	23	5	nan	nan	0	0	0

Advanced Variables: Vectors and Matrices (2)

Example: Numeric Vector

index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
vector:	12	23.3	23.3	nan	nan	1	42	42.1	23	5	nan	nan	0	0	0

Example: String

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	!	!	!	!

Advanced Variables: Vectors and Matrices (3)

Setting up a numeric Matrix: Equinox marathon pacing tables

index	**Mile**
1	1
2	5
3	10
4	15
5	20
6	25
7	26.2

Advanced Variables: Vectors and Matrices (3)

Setting up a numeric Matrix: Equinox marathon pacing tables

index	**Mile**	**record**
1	1	0:05:55
2	5	0:30:01
3	10	0:59:56
4	15	1:35:01
5	20	2:04:59
6	25	2:32:19
7	26.2	2:40:00

Advanced Variables: Vectors and Matrices (3)

Setting up a numeric Matrix: Equinox marathon pacing tables

index	Mile	record	well trained
1	1	0:05:55	0:08:42
2	5	0:30:01	0:44:06
3	10	0:59:56	1:28:01
4	15	1:35:01	2:19:33
5	20	2:04:59	3:03:34
6	25	2:32:19	3:43:43
7	26.2	2:40:00	3:55:00

Advanced Variables: Vectors and Matrices (3)

Setting up a numeric Matrix: Equinox marathon pacing tables

index	Mile	record	well trained	mildly trained
1	1	0:05:55	0:08:42	0:10:55
2	5	0:30:01	0:44:06	0:55:21
3	10	0:59:56	1:28:01	1:50:29
4	15	1:35:01	2:19:33	2:55:05
5	20	2:04:59	3:03:34	3:50:26
6	25	2:32:19	3:43:43	4:40:50
7	26.2	2:40:00	3:55:00	4:55:00

Example: Equinox marathon pacing table in Matlab

```
% UAF/GI   Beyond the mouse, fall 2010, Ronni Grapenthin
% EXAMPLE: 2D matrix (Table), prints list of times that can be used for optimal
% Equinox 2011 preparation
% parameter: miles — miles you've run

function pace_table = pacing_table(miles)

% Set up pacing table: Give miles as numbers and times as strings (requires a cell array,
% hence the curly braces)
pace_table = {  1     '0:05:55' '0:08:42' '0:10:55';
                5     '0:30:01' '0:44:06' '0:55:21';
                10    '0:59:56' '1:28:01' '1:50:29';
                15    '1:35:01' '2:19:33' '2:55:05';
                20    '2:04:59' '3:03:34' '3:50:26';
                26.2  '2:40:00' '3:55:00' '4:55:00'};


% Since I'm lazy and didn't want to type all the miles, a mile does not equal the index,
% hence we'll have to do some math. Index is rounded number of miles divided by 5. Since
% Matlab indices start at 1, we have to add a 1. Otherwise everything smaller than 2.5 miles
% would result in an error
idx = round(miles/5)+1;

% lame output
pace_table(idx,:)
pause

% fancy output:
disp('␣');
disp('␣␣␣miles␣␣␣␣record␣␣␣␣well␣trained␣␣mildly␣trained');
disp('␣␣------------------------------------------------------------');
disp(pace_table(idx,:));
end
```

Listing 2.2: pacing_table.m