Beyond the Mouse – A Short Course on Programming

2. Fundamental Programming Principles

Ronni Grapenthin

Geophysical Institute, University of Alaska Fairbanks

April 17, 2009

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

Outline

1. Solutions to Exercises
2. How does programming work?
3. Variables and Datatypes (1)
4. Control Flow
5. Good Practice

Outline

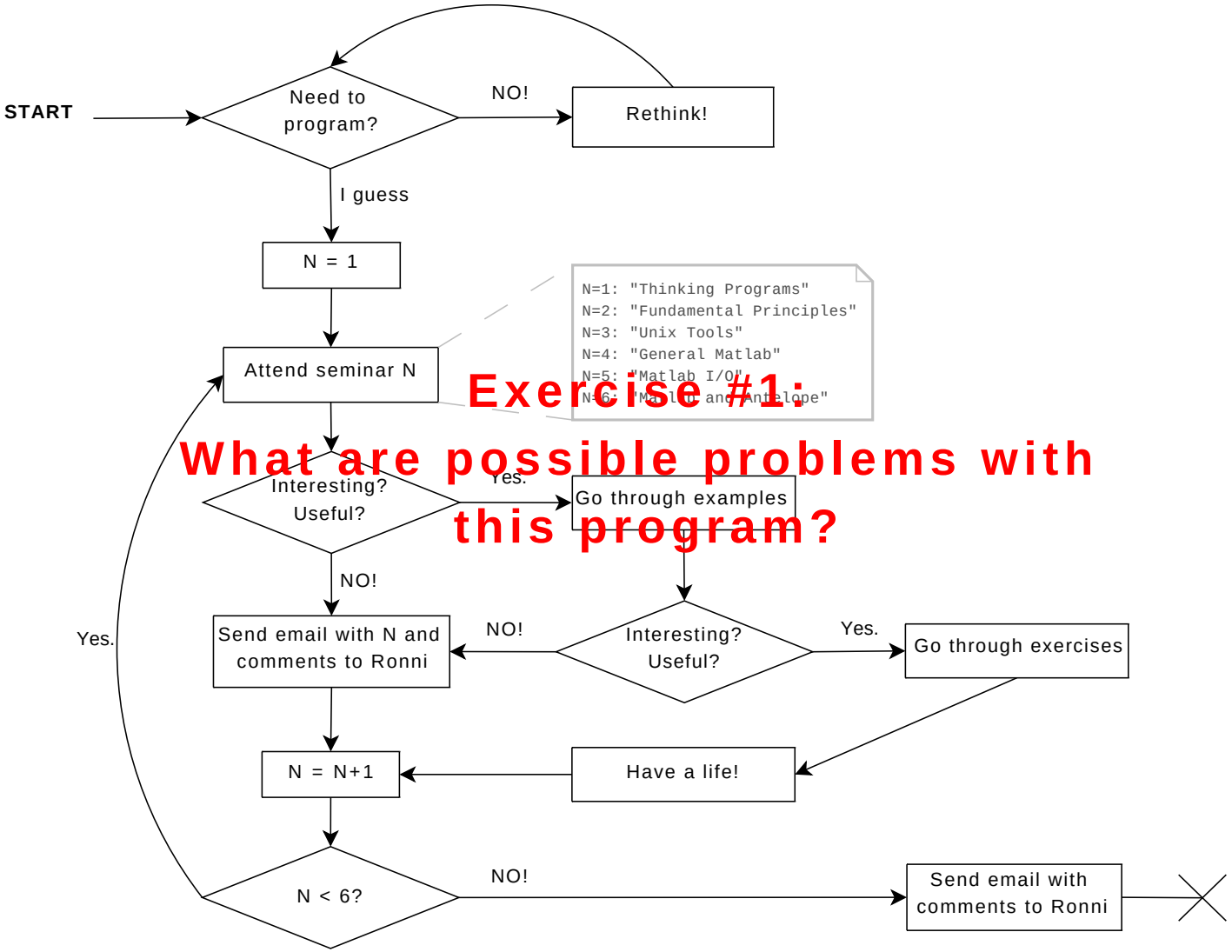
1. Solutions to Exercises
2. How does programming work?
3. Variables and Datatypes (1)
4. Control Flow
5. Good Practice

Exercise #0

...

nothing . . . :(

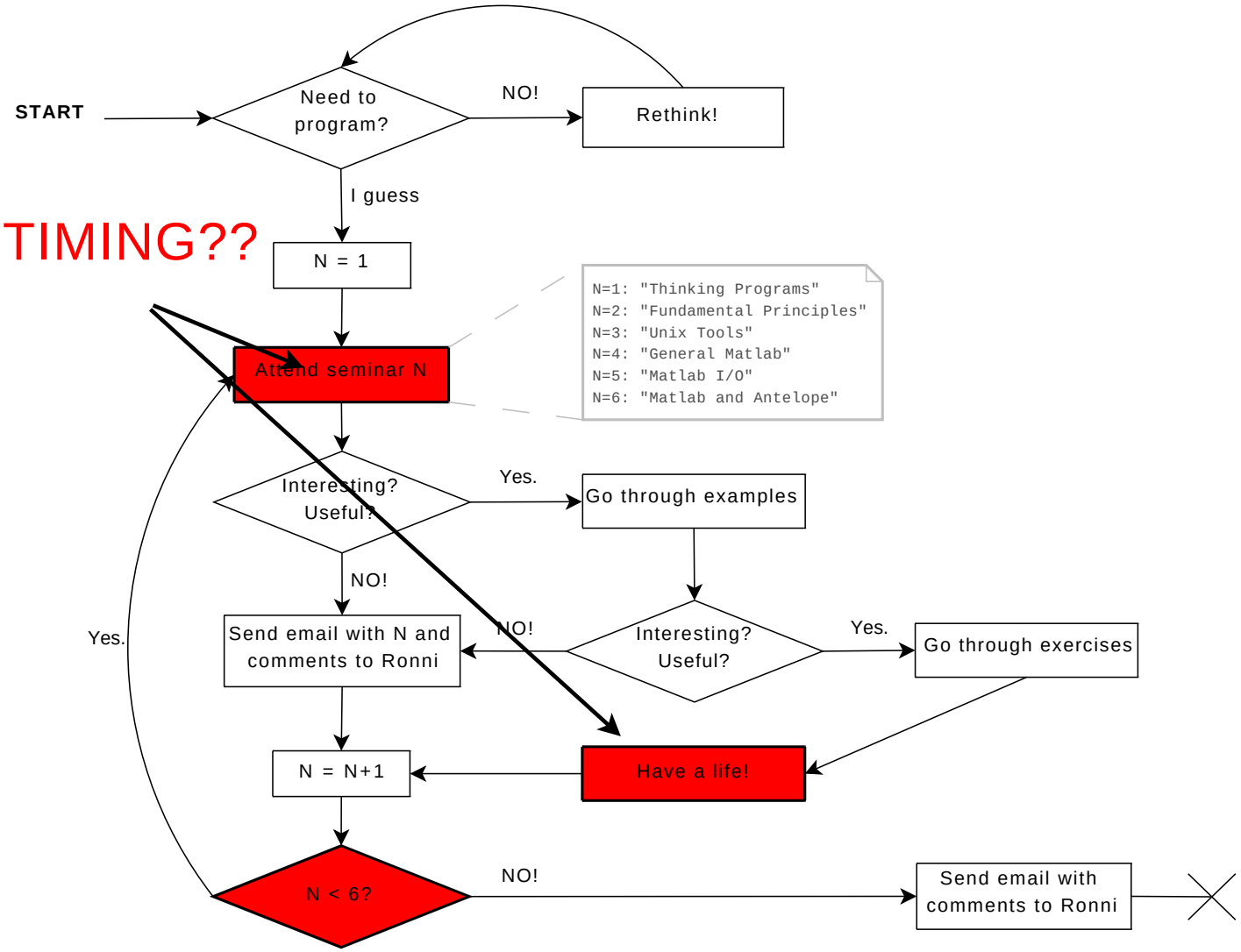
Exercise #1



Exercise #1: What are possible problems with this program?

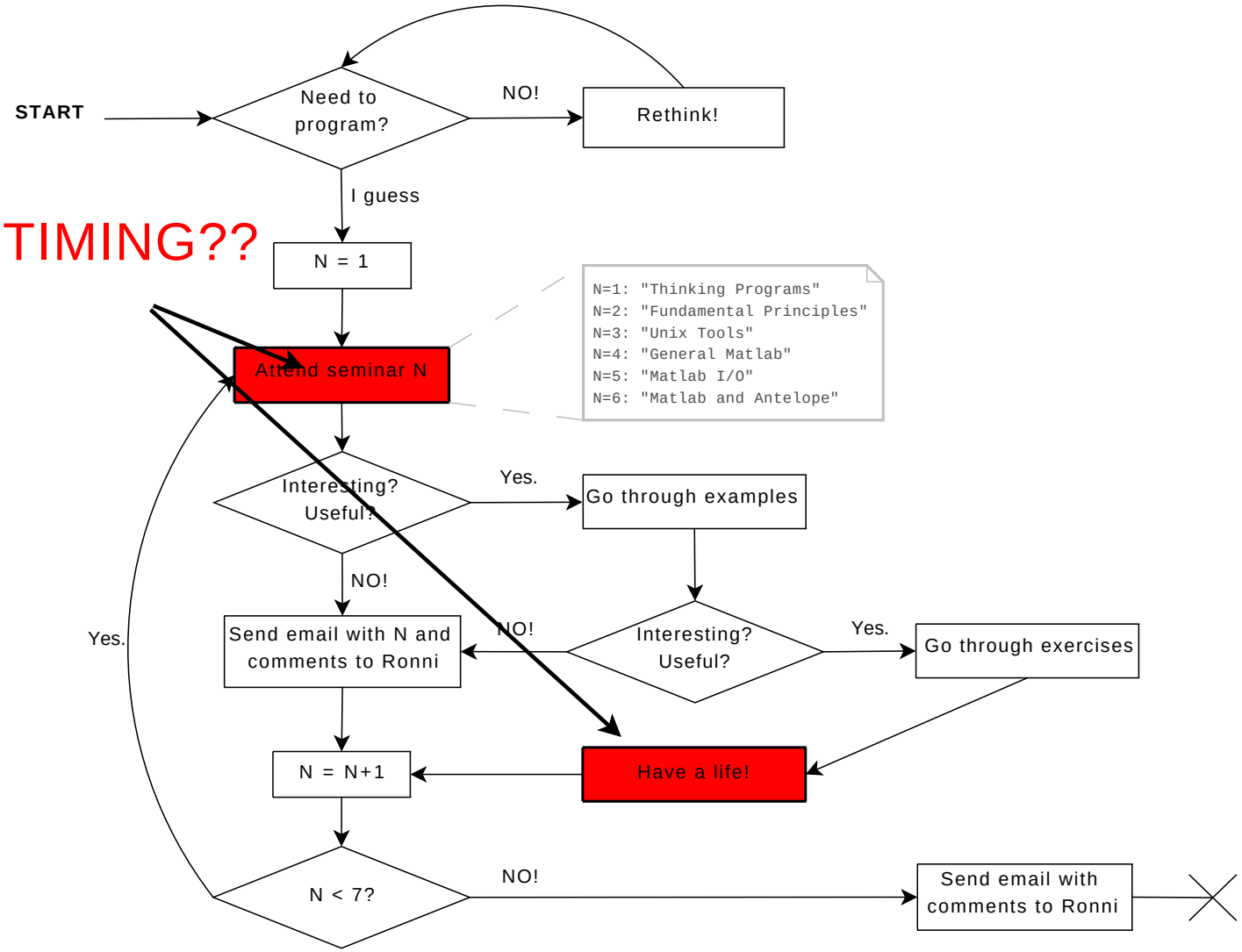
Listing 1: Seminar flow (fixed)

Exercise #1



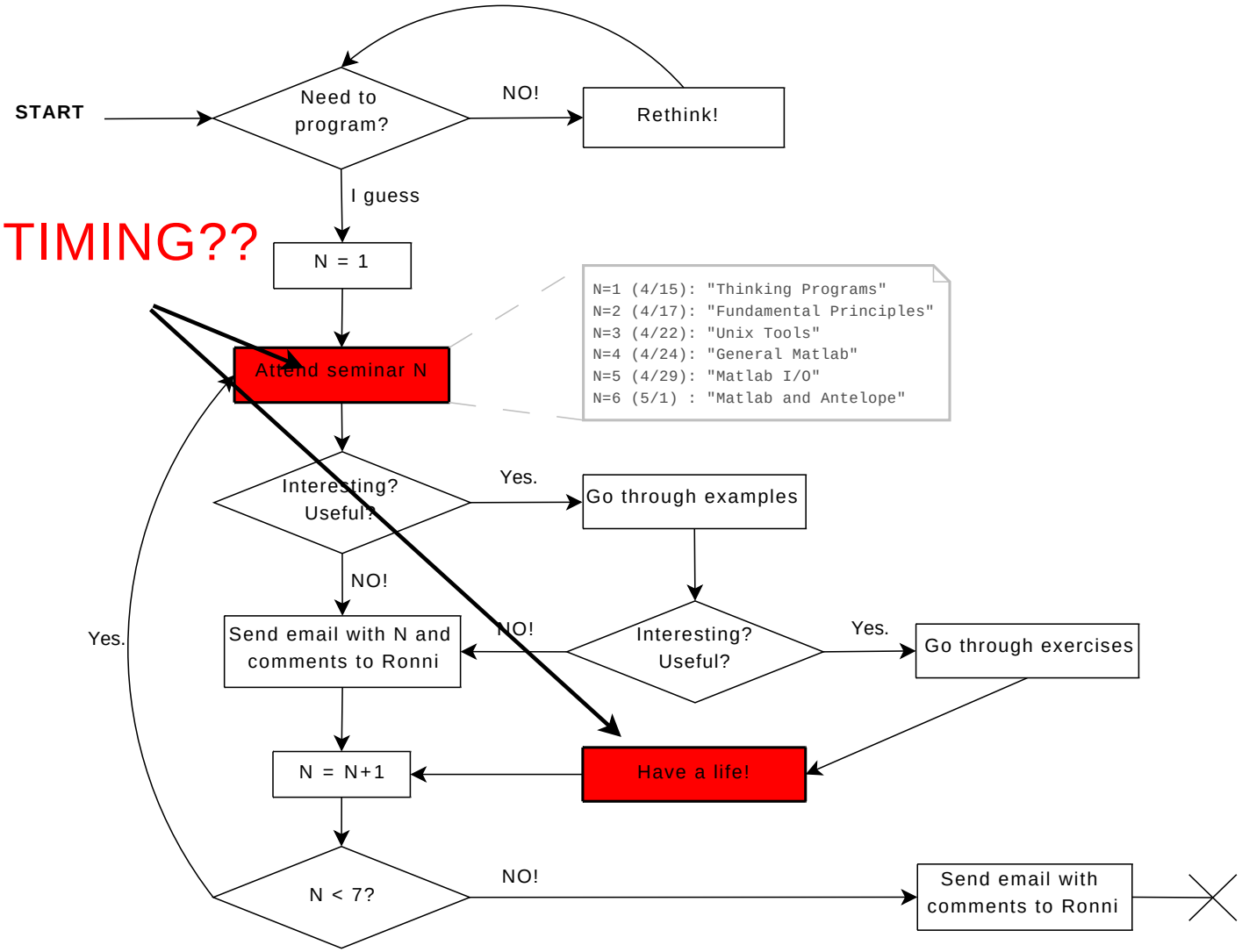
Listing 1: Seminar flow (fixed)

Exercise #1



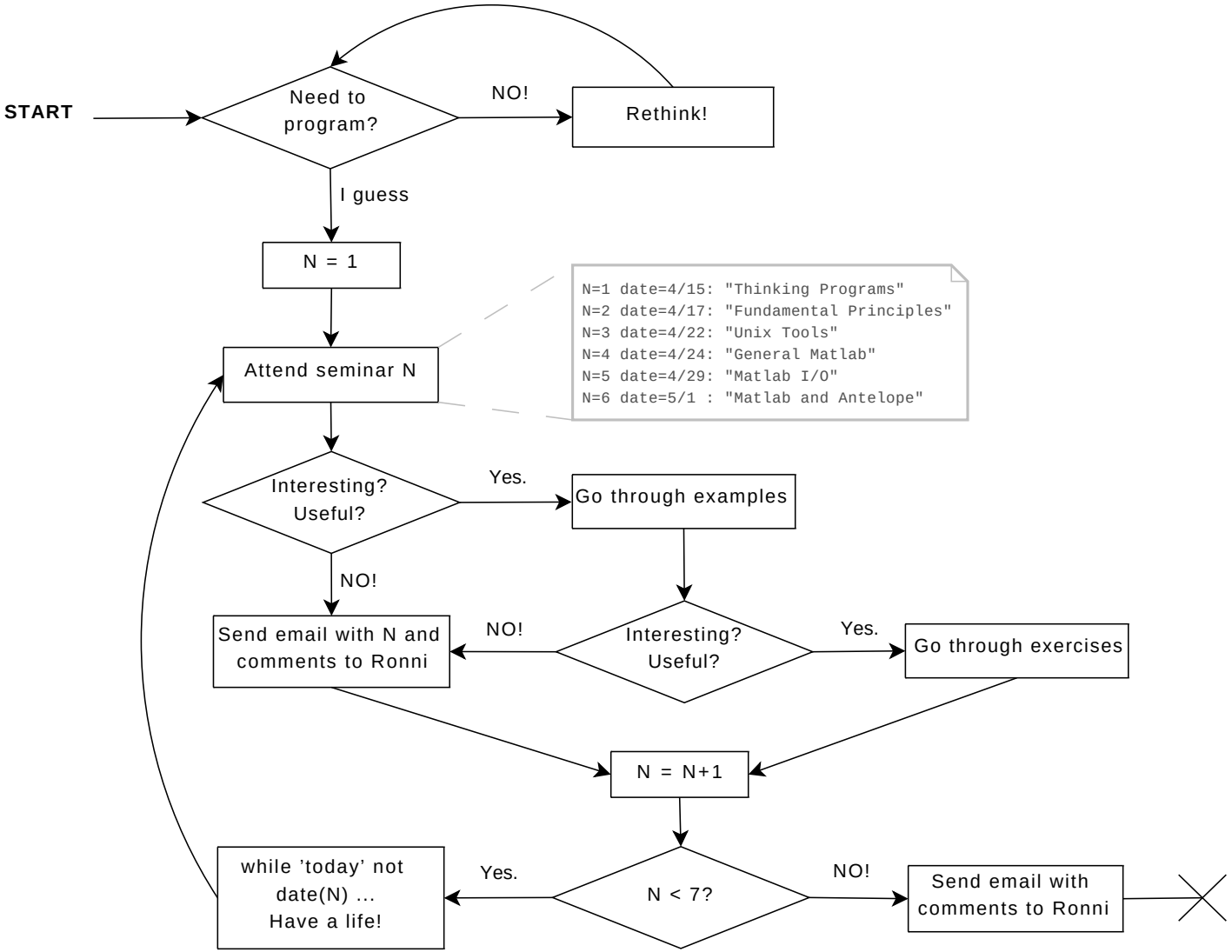
Listing 1: Seminar flow (fixed)

Exercise #1



Listing 1: Seminar flow (fixed)

Exercise #1



Listing 1: Seminar flow (fixed)

Exercise #2

so?

Outline

1. Solutions to Exercises
2. How does programming work?
3. Variables and Datatypes (1)
4. Control Flow
5. Good Practice

How does programming work?

Well, fist we should be clear what a programming language is . . .

Alright, what's a programming language 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.

Two broad families can be identified:

Alright, what's a programming language 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.

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)

Alright, what's a programming language 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.

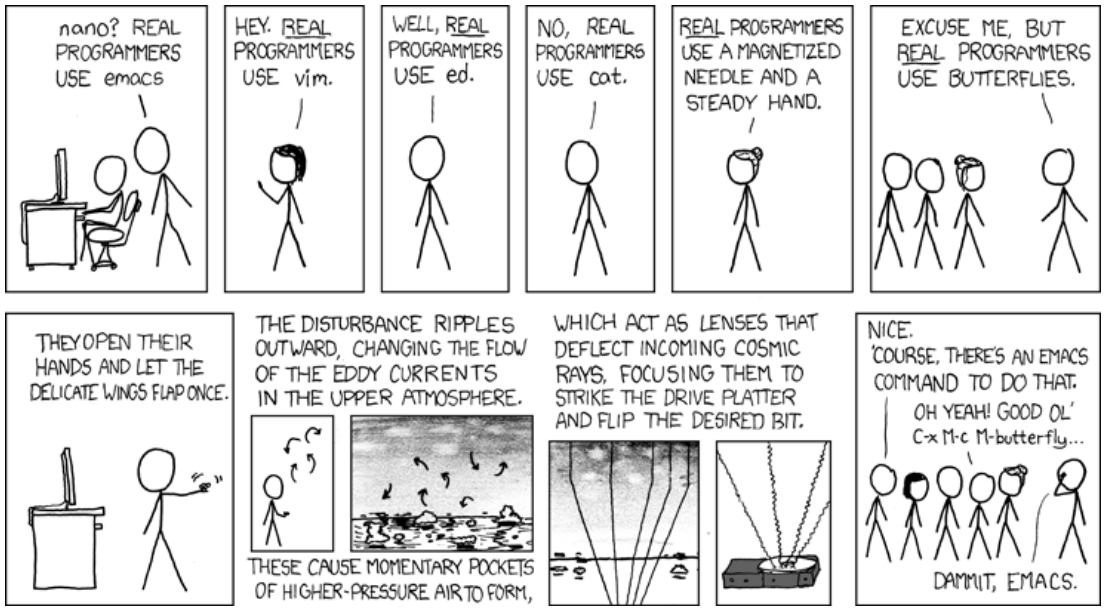
Two broad families can be identified:

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

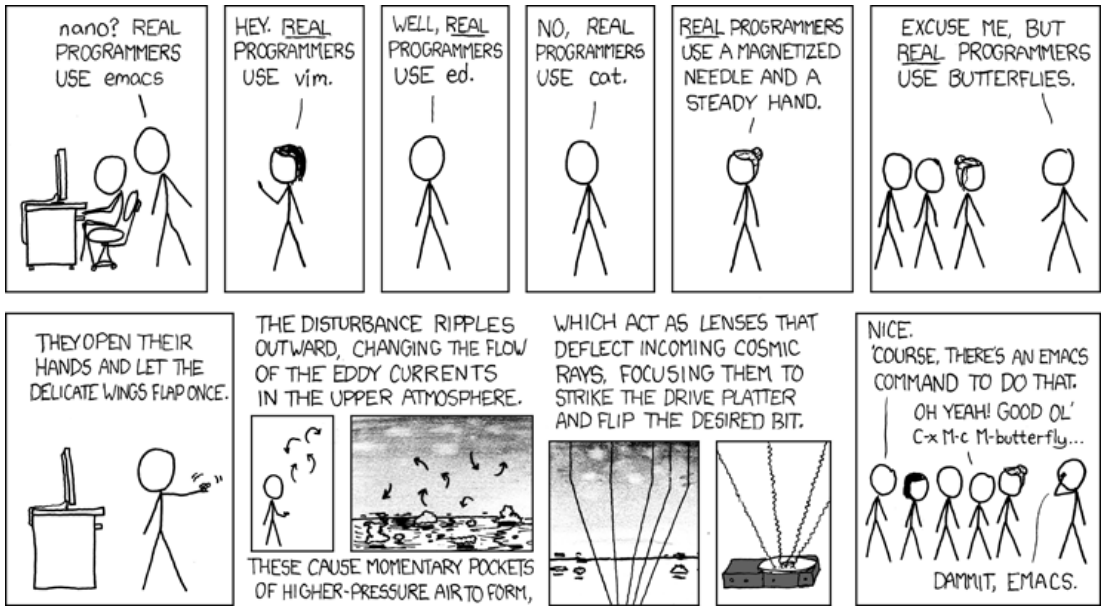
1. open text editor (vi, notepad, . . . , not MS Word!)



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

Now, how does programming work?

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

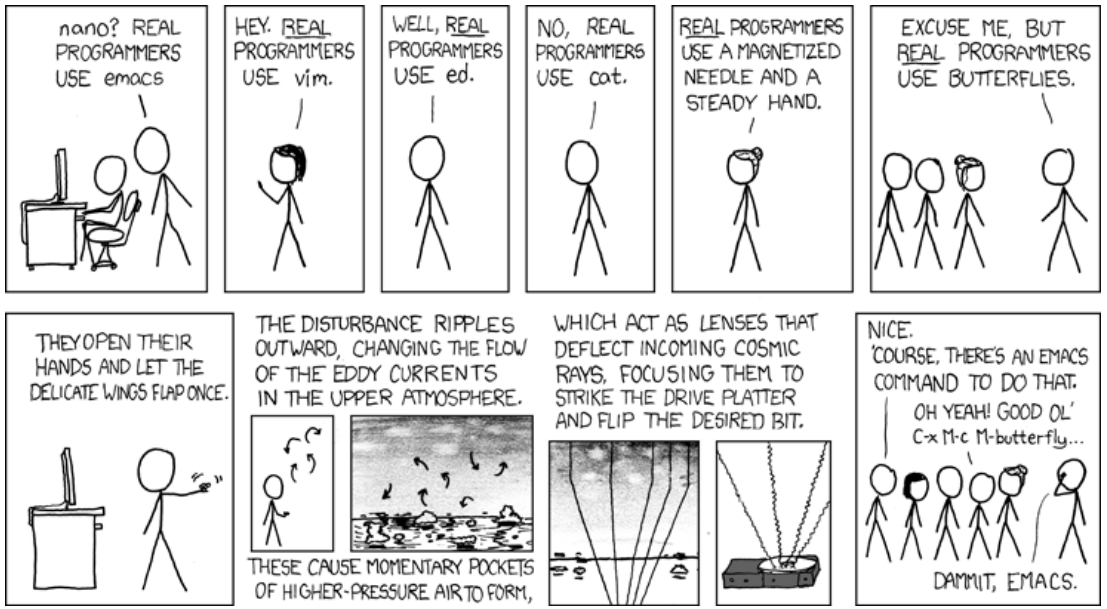


http://www.xkcd.com/378/

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

Now, how does programming work?

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

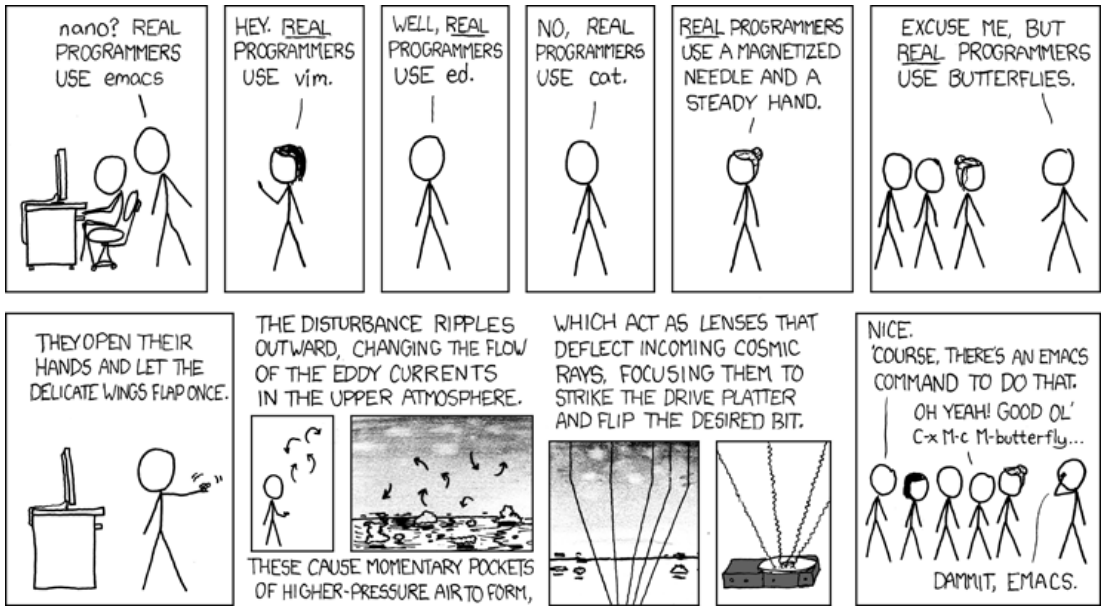


http://www.xkcd.com/378/

2. translate your (mental) flowchart into a set of instructions according to the 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!)

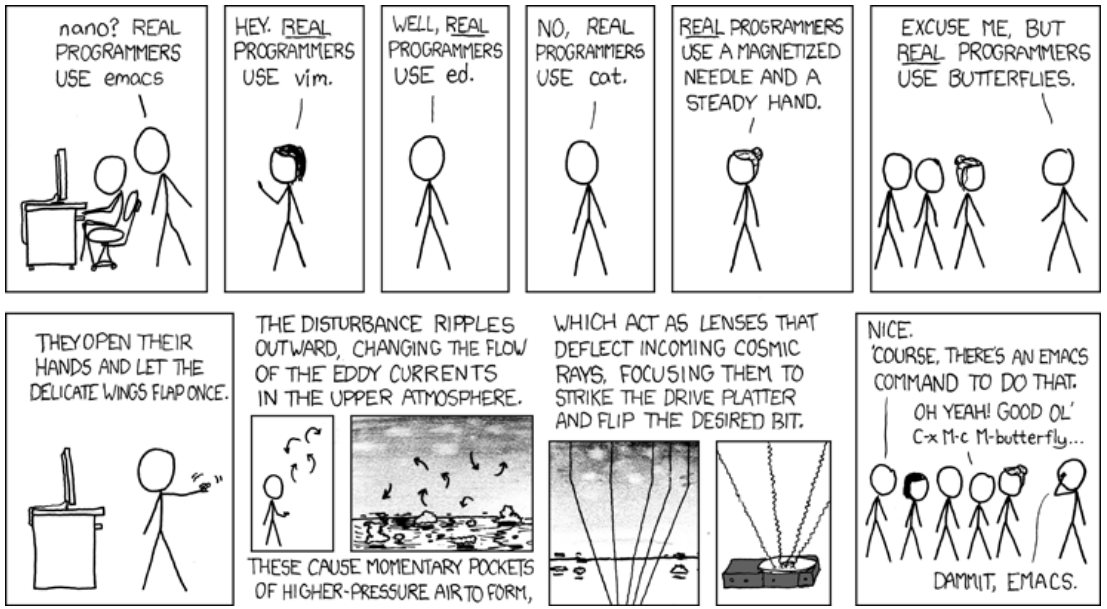


http://www.xkcd.com/378/

2. translate your (mental) flowchart into a set of instructions according to the 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!)

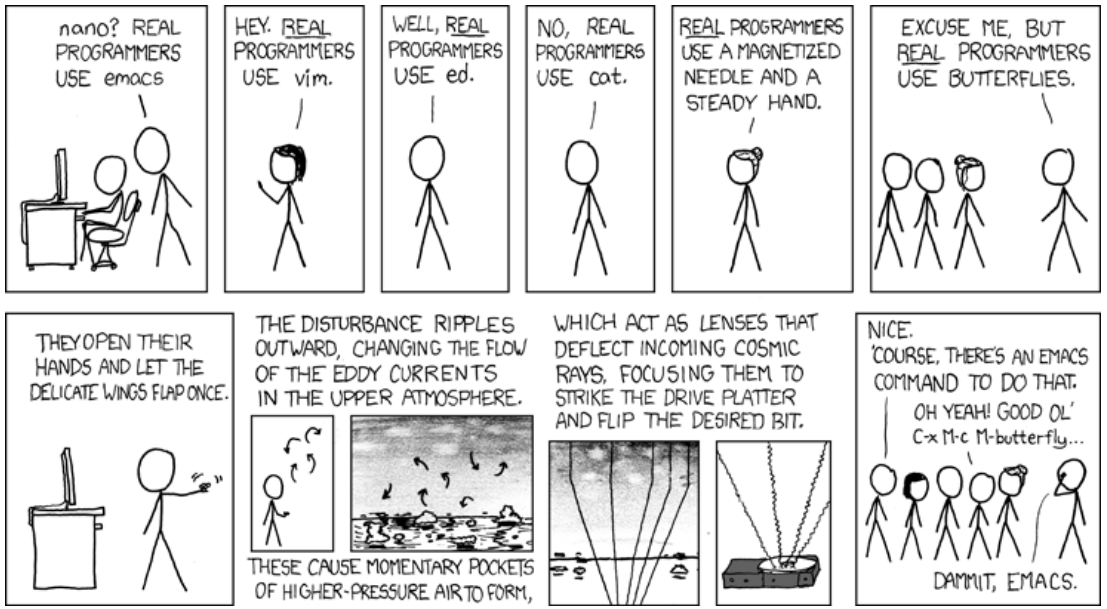


http://www.xkcd.com/378/

2. translate your (mental) flowchart into a set of instructions according to the 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 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

Outline

1. Solutions to Exercises
2. How does programming work?
3. Variables and Datatypes (1)
4. Control Flow
5. Good Practice

Variables and Data Types (1)

Definition – Variable (basic)

User defined keyword that is linked to a value stored in computer's memory (runtime). Assigning a value:
Matlab: `myNewVar = 10;` **TC-Shell** (differs) `set myNewVar = 10;`
Access to the values by de-referencing:
Matlab: `myNewVar;` **TC-Shell** (differs) `$myNewVar`

thanks to wikipedia for some help right here.

Variables and Data Types (1)

Definition – Variable (basic)

User defined keyword that is linked to a value stored in computer's memory (runtime). Assigning a value:
Matlab: `myNewVar = 10;` **TC-Shell** (differs) `set myNewVar = 10;`
Access to the values by de-referencing:
Matlab: `myNewVar;` **TC-Shell** (differs) `$myNewVar`

Defintion – Data Type (basic)

Tells the machine (well, really the interpreter/compiler) what kind of data is to be expected in a memory slot a certain variable points to. This is necessary since different data types occupy differently sized chunks of memory. Scripting languages such as Matlab and Perl are **weakly typed** and do necessary type conversions themselves if possible. Basic data types are: logic (boolean), character (char), string, integer, floating point (double precision).

thanks to wikipedia for some help right here.

Variables and Data Types (2)

Vectors and Matrices – Arrays

Array variables are lists, vectors, matrices of data. An array variable is linked to a chunk of memory that contains a series of values (usually same type) and can be dereferenced using an index number. Depending on the programming language the first value can sit at index ’0’ or ’1’. Arrays can be 1 to n dimensional (depending on the language it’s either more or less fun to keep track of that). Matlab treats everything as a matrix. Shells allow only vectors.

Variables and Data Types (2)

Vectors and Matrices – Arrays

Array variables are lists, vectors, matrices of data. An array variable is linked to a chunk of memory that contains a series of values (usually same type) and can be dereferenced using an index number. Depending on the programming language the first value can sit at index '0' or '1'. Arrays can be 1 to n dimensional (depending on the language it's either more or less fun to keep track of that). Matlab treats everything as a matrix. Shells allow only vectors.

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

Variables and Data Types (2)

Vectors and Matrices – Arrays

Array variables are lists, vectors, matrices of data. An array variable is linked to a chunk of memory that contains a series of values (usually same type) and can be dereferenced using an index number. Depending on the programming language the first value can sit at index '0' or '1'. Arrays can be 1 to n dimensional (depending on the language it's either more or less fun to keep track of that). Matlab treats everything as a matrix. Shells allow only vectors.

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

More Complex things

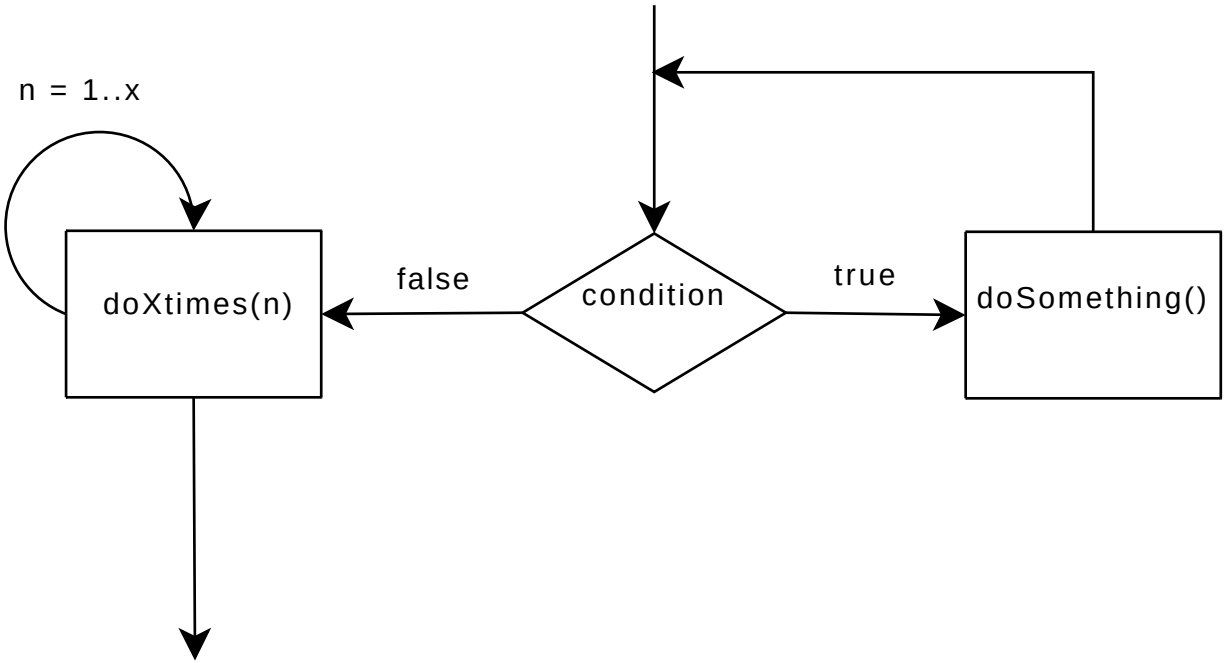
Exercise: Read Matlab help on `cell array` and `struct` (type: `>help cell` ...

Outline

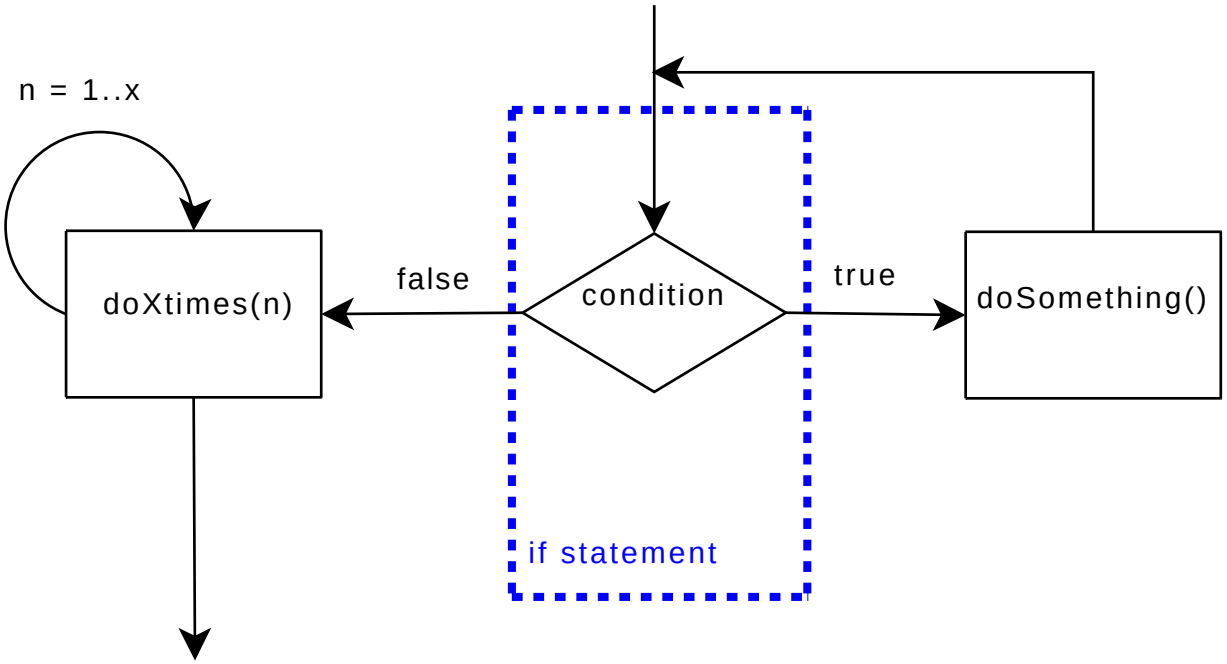
1. Solutions to Exercises
2. How does programming work?
3. Variables and Datatypes (1)
4. Control Flow
5. Good Practice

Control Flow – Redirecting the stream

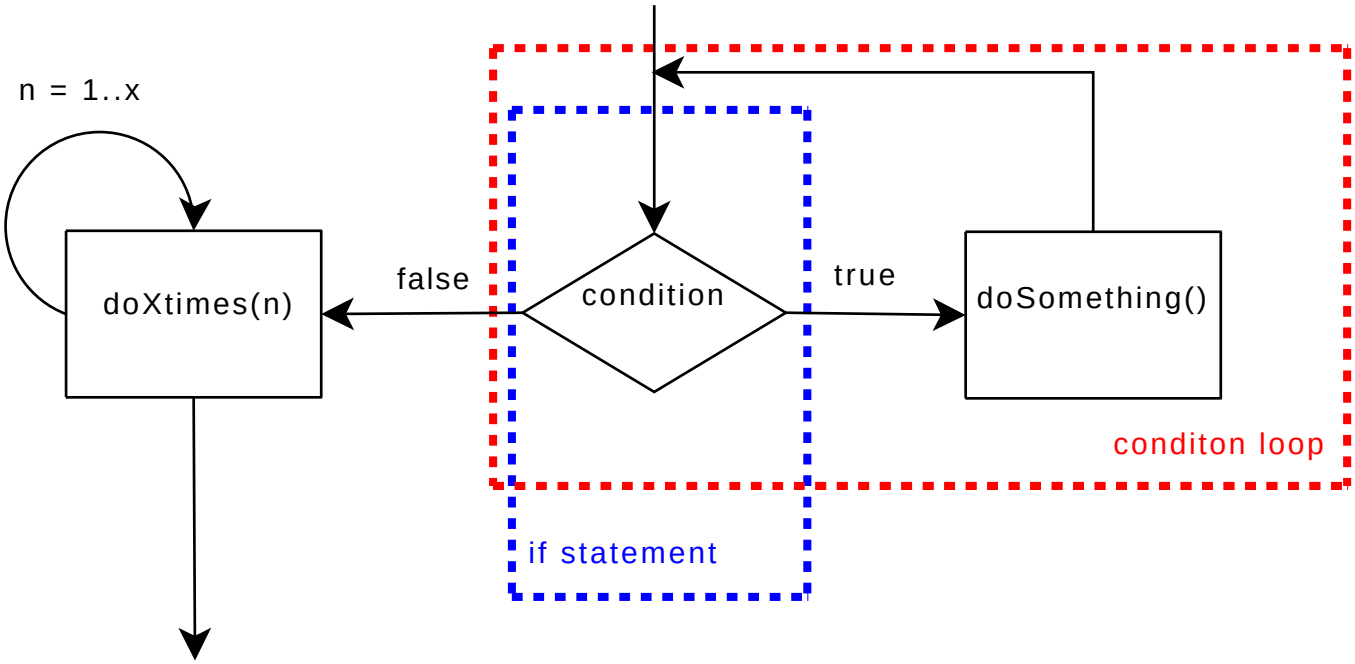
Control Flow – Redirecting the stream



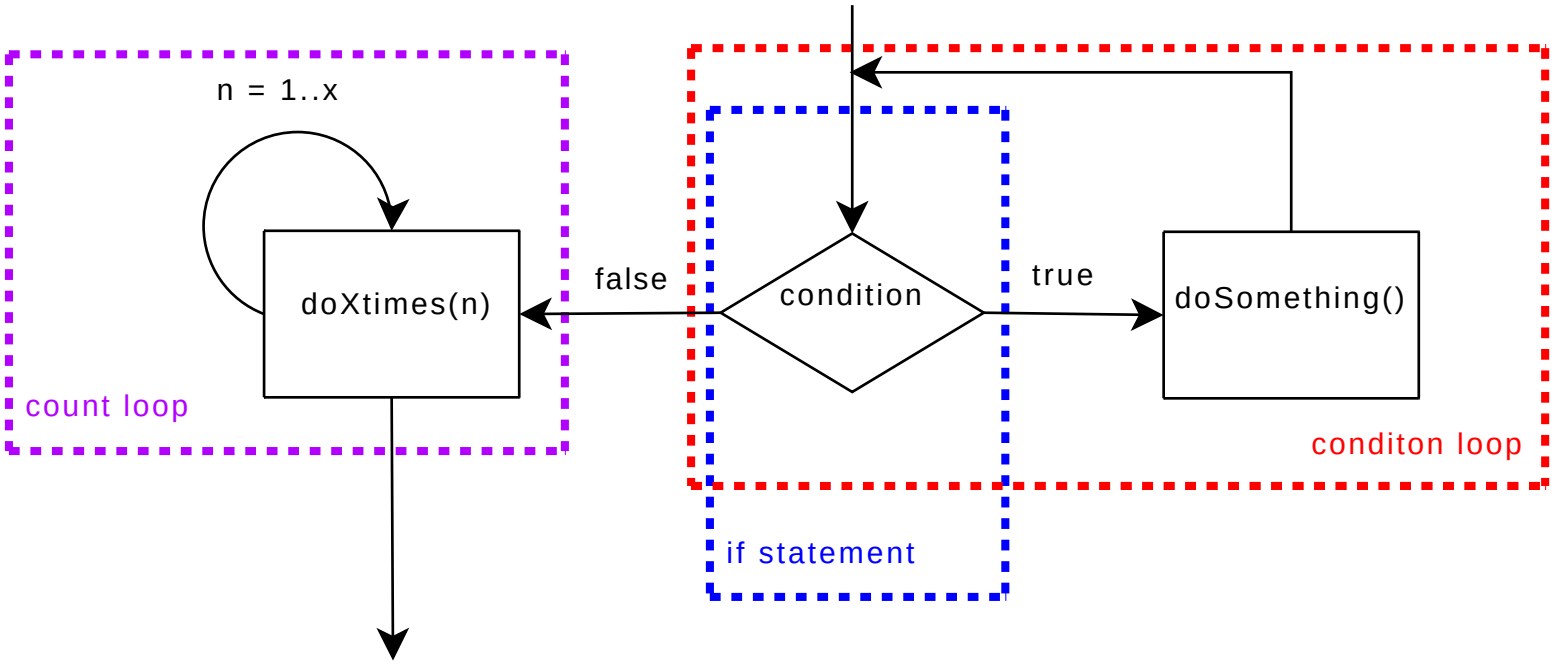
Control Flow – Redirecting the stream



Control Flow – Redirecting the stream



Control Flow – Redirecting the stream



Control Flow – Redirecting the stream

Flow control turns batch processing into programming:

- (high level) programming languages allow different behavior based on conditions **you** define – **flow control**
- A condition can be `true` (1) or `false` (0).
- You test a condition using the operators: `<, <=, >, >=, ==, !=` (find equiv. in each respective language)
- Function often give numeric return values as answer to a test. In Matlab `strcmp('compare', 'strings')` will return `false`.

Logic 101

Use logic to connect multiple conditions and test for certain cases:

Logic 101

Use logic to connect multiple conditions and test for certain cases:

‘NOT’ (“~”, ’!’):

a	!a
0	1
1	0

Logic 101

Use logic to connect multiple conditions and test for certain cases:

‘NOT’ (“~”, ’!’):

a	!a
0	1
1	0

‘AND’ (‘&&’):

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

Logic 101

Use logic to connect multiple conditions and test for certain cases:

‘NOT’ (“~”, ’!’):

a	!a
0	1
1	0

‘AND’ (‘&&’):

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

‘OR’ (‘||’):

a	b	a || b
0	0	0
0	1	1
1	0	1
1	1	1

Logic 101

Use logic to connect multiple conditions and test for certain cases:

‘NOT’ (“~”, ’!’):

a	!a
0	1
1	0

‘AND’ (‘&&’):

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

‘OR’ (‘||’):

a	b	a || b
0	0	0
0	1	1
1	0	1
1	1	1

Examples

- ‘Friday Beer’: **not** younger than 21 **and** it must be Friday. Beer today?

Logic 101

Use logic to connect multiple conditions and test for certain cases:

‘NOT’ (“~”, ’!’):

a	!a
0	1
1	0

‘AND’ (‘&&’):

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

‘OR’ (‘||’):

a	b	a || b
0	0	0
0	1	1
1	0	1
1	1	1

Examples

- ‘Friday Beer’: **not** younger than 21 **and** it must be Friday. Beer today?
- ‘Game of life’: heart beat **or** self perception. Still alive?

Control flow (0) – `statements` and such

We need a little bit of a formal definition for the following slides. Bear with me

Formal language definitions

```
<block> ::= { <statement list> }.

<statement list> ::=
        <statement>
      | <statement list> <statement>.

<statement> ::=
        <block>
      | <assignment statement>
      | <if statement>
      | <for loop>
      | <while loop>
      | <do statement>
      | . . .
```

Listing 2.1: bnf.txt

Control flow (1) – `if - then - else`

Formal

```
<if statement> ::= if (<condition>) <statement> [else <statement>].
```

Control flow (1) – `if - then - else`

Formal

```
<if statement> ::= if (<condition>) <statement> [else <statement>].
```

Matlab

```
% if ( CONDITION ) STATEMENT
% [elseif STATEMENT ]
% [else STATEMENT ]
% end.
%
% EXAMPLE: What are we gonna
% do today?
%

day=weekday(now);

if (day == 6 )
   disp('PUB!')
elseif (day == 1 || day == 7)
      disp('sleep')
else
   disp('duh.')
end
```

Control flow (1) – `if - then - else`

Formal

```
<if statement> ::= if (<condition>) <statement> [else <statement>].
```

Matlab

```
% if ( CONDITION ) STATEMENT
% [elseif STATEMENT ]
% [else STATEMENT ]
% end.
%
% EXAMPLE: What are we gonna
% do today?
%

day=weekday(now);

if (day == 6 )
   disp('PUB!')
elseif (day == 1 || day == 7)
      disp('sleep')
else
   disp('duh.')
end
```

C-Shell

```
#!/bin/tcsh
# if ( <condition> ) then <statement>
# [else <statement> ]
# endif
#
# Example: What are we gonna do today?

set day = `date | awk '{print $1}'`

if ($day == 'Fri' ) then
   echo 'PUB!'
else
   if ($day == 'Sat' || \
       $day == 'Sun') then
      echo 'sleep.'
   else
      echo 'duh.'
   endif
endif
```

Control flow (2) – condition controlled loop: `while`

Formal

```
<while loop> ::= while (<condition>) <block>.
```

Control flow (2) – condition controlled loop: `while`

Formal

```
<while loop> ::= while (<condition>) <block>.
```

Matlab

```
% while ( CONDITION )
%   STATEMENT
% end.
%
% EXAMPLE: Tell me when a new minute starts
%
clc;              %clear screen
c=clock;          %get time vector

% 6th element of c is seconds
while ( c(6) < 59.9)
    c=clock;
end
disp('start new minute of your life');
```

Control flow (2) – condition controlled loop: `while`

Formal

```
<while loop> ::= while (<condition>) <block>.
```

Matlab

```
% while ( CONDITION )
%   STATEMENT
% end.
%
% EXAMPLE: Tell me when a new minute starts
%
clc;            %clear screen
c=clock;        %get time vector

% 6th element of c is seconds
while ( c(6) < 59.9)
    c=clock;
end
disp('start new minute of your life');
```

C-Shell

```
#!/bin/tcsh
# while ( <condition> ) <block>
#
# Example: Tell me when a new minute starts

#figure out actual second value ...
set sec = `date | \
      awk '{ split($4, x, ":"); print x[3]}'`

#do that until we're starting a new minute
while ( $sec < 59 )
   set sec = `date | \
         awk '{ split($4, x, ":"); print x[3]}'`
   echo $sec
end

echo 'start new minute of your life';
```

Control flow (3) – count controlled loop: `for`

Formal

```
<for loop> ::= for (<assignment>; <condition>; <assignment>) <block>.
```

Control flow (3) – count controlled loop: `for`

Formal

```
<for loop> ::= for (<assignment>; <condition>; <assignment>) <block>.
```

Matlab

```
% for variable = expression
%   STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;                %clear screen
for n=1:10
    disp(sprintf('n=%d', n));
end
disp('done.');
```

Control flow (3) – count controlled loop: `for`

Formal

```
<for loop> ::= for (<assignment>; <condition>; <assignment>) <block>.
```

Matlab

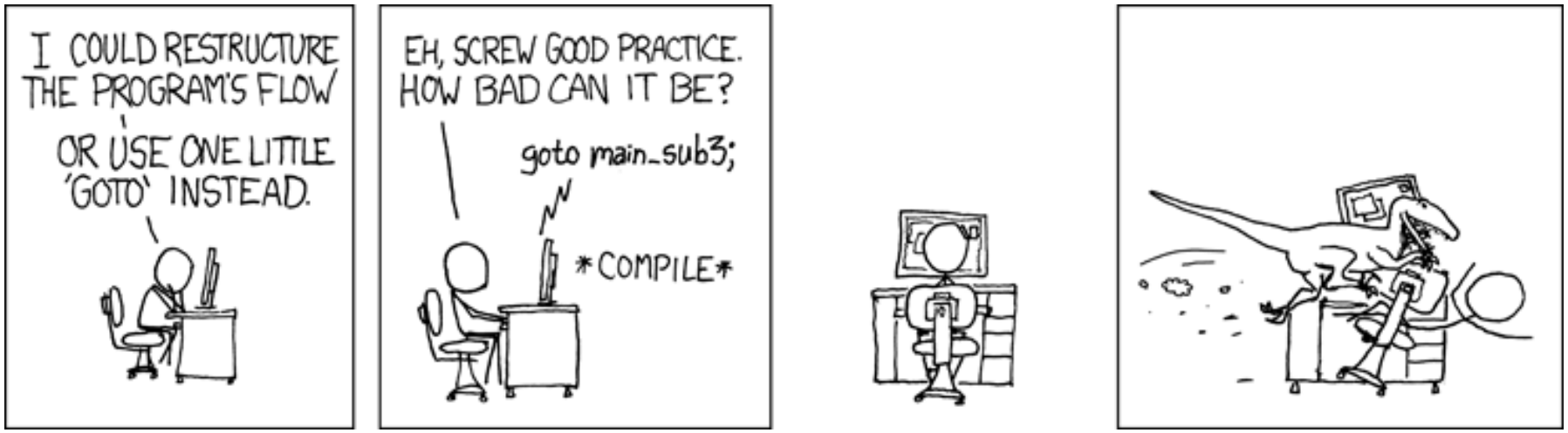
```
% for variable = expression
%   STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;              %clear screen
for n=1:10
    disp(sprintf('n=%d', n));
end
disp('done.');
```

C-Shell

```
#!/bin/tcsh
# foreach variable ( <list> ) <block>
#
# Example: list files in current
# directory (yeah, I know).

foreach x (`ls ./`)
  echo $x
end
```

Don't you ever dare to `goto`!



"GOTO", http://xkcd.com/292

Outline

1. Solutions to Exercises
2. How does programming work?
3. Variables and Datatypes (1)
4. Control Flow
5. Good Practice

Good Practice

Good Practice

How to make your code readable (language independent)

- use indentations to structure your code (align comments etc)
- use meaningful variable and function names (`sec` instead of `i` and `listFiles()` instead of `lfls()`)
- decide for one formatting and naming scheme and stick to it; no matter which one it is.
- comment your code
- do not over comment your code!
- selfstudy:
  http://www.google.com/search?hl=en&q=good+programming+style&btnG=Search