Fundamental Programming Principles: Variables and Data Types

Beyond the Mouse

GEOS 436/636

Jeff Freymueller, Sep 5, 2017

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

- How does computer programming work
  - What is a programming language?
  - What is a program?
- Variables and Data Types
  - How do we store values of different kinds?
    - Numbers
    - Strings of text
    - More complicated things (like images, for example)

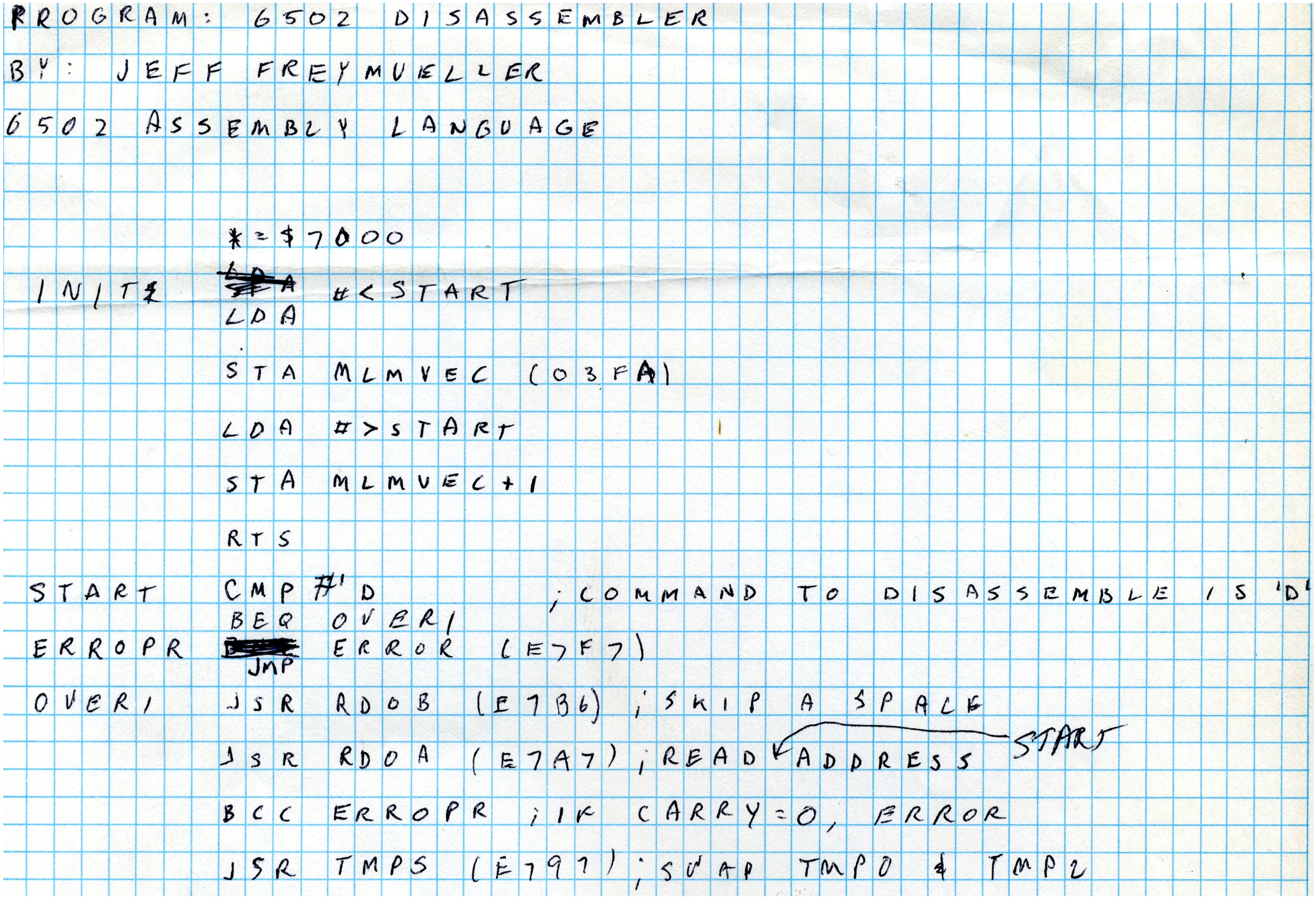
Definitions

- 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.

What Language Does the CPU Understand?

- The CPU (Central Processing Unit) actually understands only a language composed entirely of numbers, like this:
  - “157 65530 22 77 854” (this is a made-up example)
  - This means “execute instruction #157 using an argument 65530, then execute instruction #22, then execute instruction #77 using an argument 854”
  - The language definition tells the machine that instruction #157 takes one argument, but #22 does not.
- It is possible for a person to write code in this machine language, but almost nobody does it any more because it is so inconvenient.

I Actually Did This



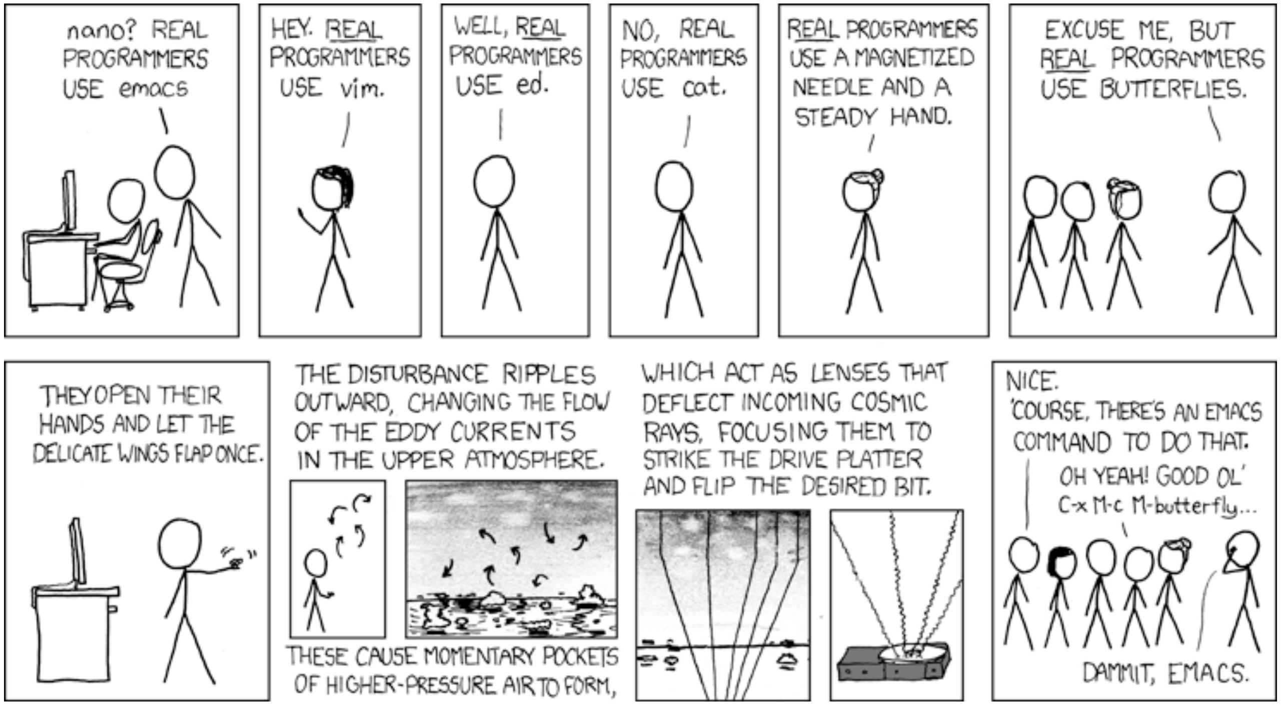
```
RROGRAM: 6502 DISASSEMBLER
BY: JEFF FREYMUELLER
6502 ASSEMBLY LANGUAGE

          *=$7000
INIT      ~~STA~~ #<START
          LDA
          STA  MLMVEC (03FA)
          LDA  #>START
          STA  MLMVEC+1
          RTS
START     CMP #'D             ; COMMAND TO DISASSEMBLE IS 'D'
          BEQ  OVER1
ERROPR    JMP  ERROR  (E7F7)
OVER1     JSR  RDOB   (E7B6) ; SKIP A SPACE
          JSR  RDOA   (E7A7) ; READ START ADDRESS
          BCC  ERROPR ; IF CARRY=0, ERROR
          JSR  TMPS   (E797) ; SWAP TMP0 & TMP2
```

Programming Languages

- Can be broken into two large families:
- **Interpreted** languages. An interpreter program takes in commands, check syntax and translates to machine language at runtime (e.g., Matlab, Unix Shell)
- **Compiled** languages. Programs are translated and saved in machine language by a compiler. At runtime no additional interpretation is necessary (e.g., FORTRAN, C/C++).
  - These generally run much faster than interpreted languages

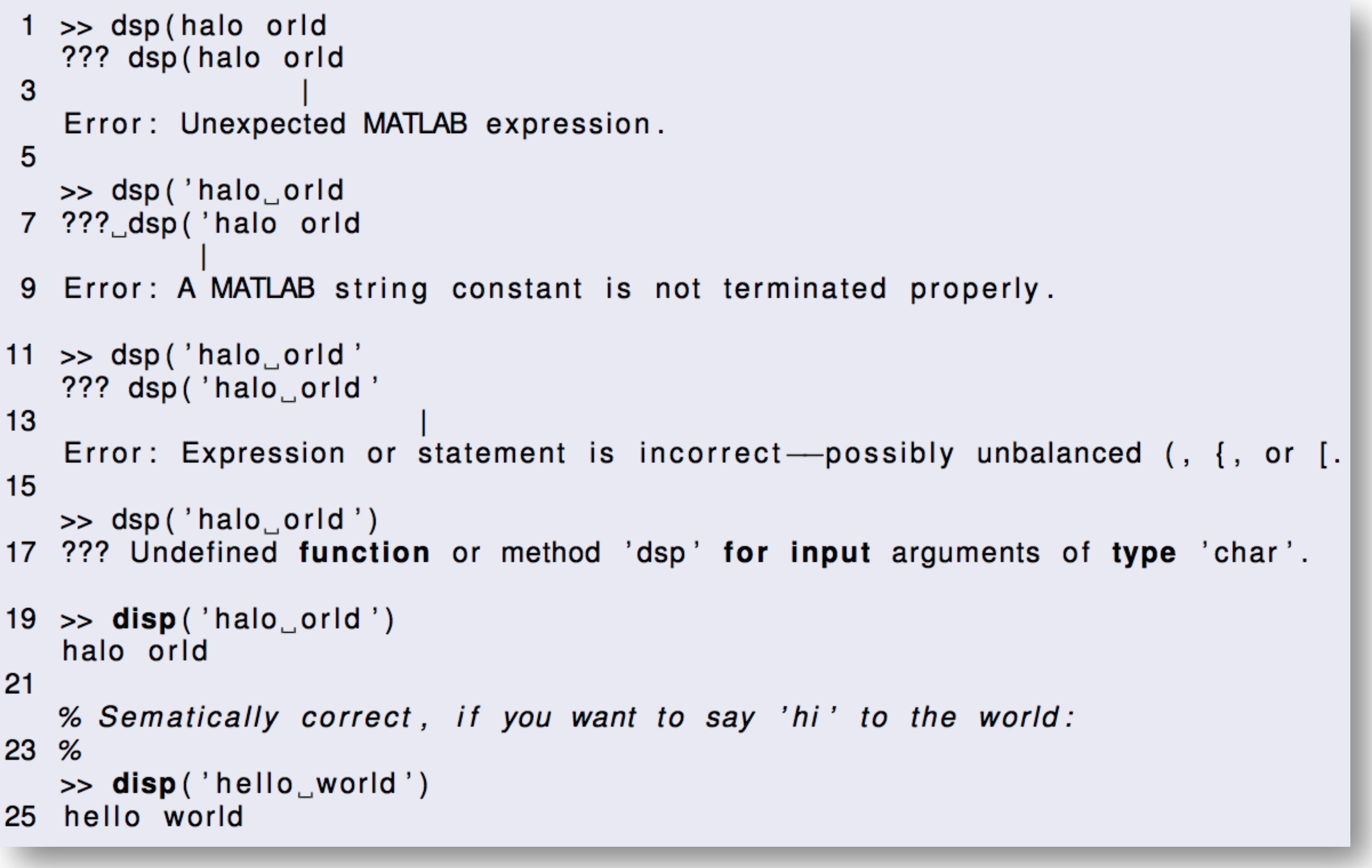
Now, How Does Programming Work?

1. Open a **text editor** (MATLAB editor, vi, notepad, Text Wrangler, … 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.

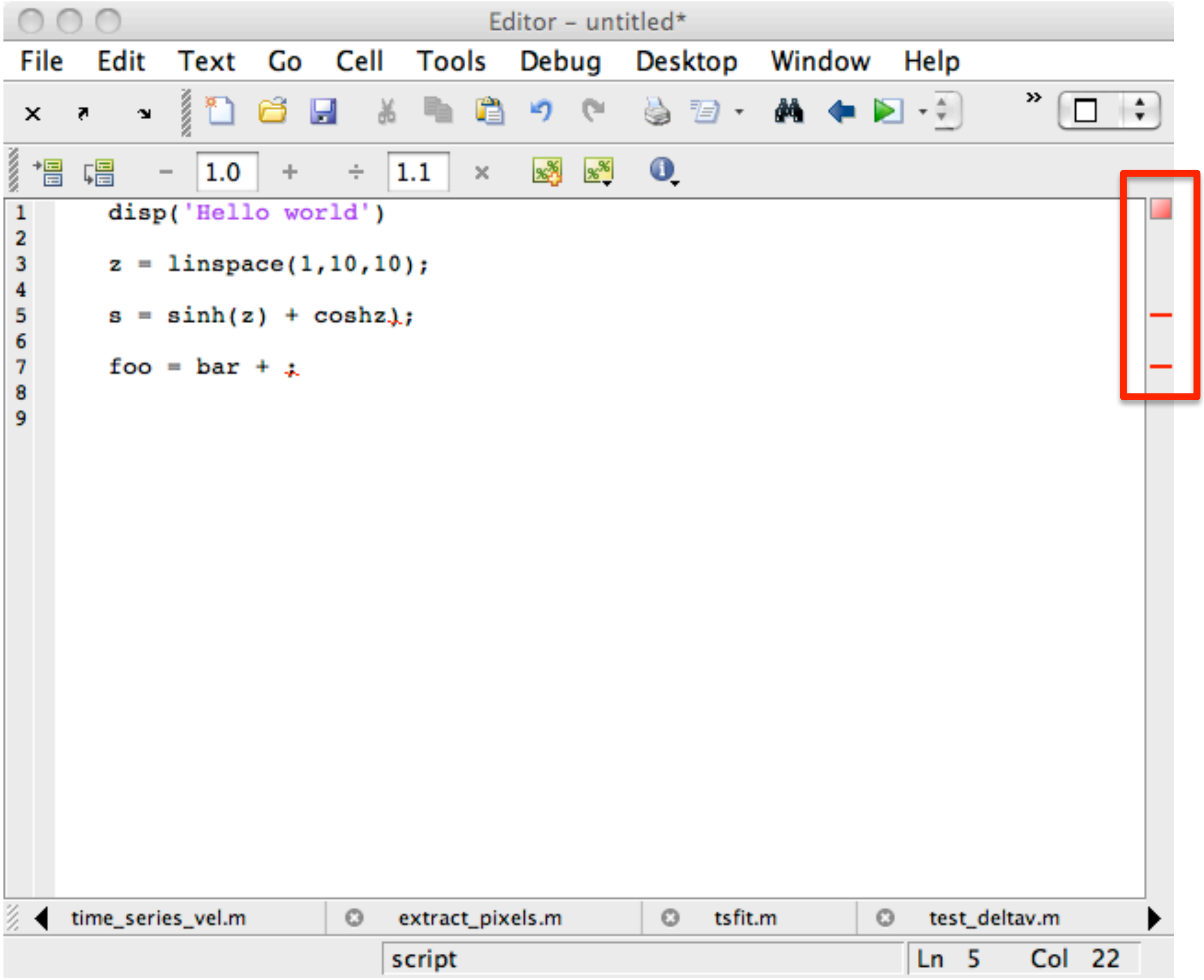
2. translate your (physical or mental) flowchart into a set of instructions according to the rules of a programming language
3. test your program for syntactical correctness (ask the interpreter/compiler)
4. if errors, fix them and go back to (3)
5. test your program for semantic errors (the “fun” part!)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

Example: Hello World



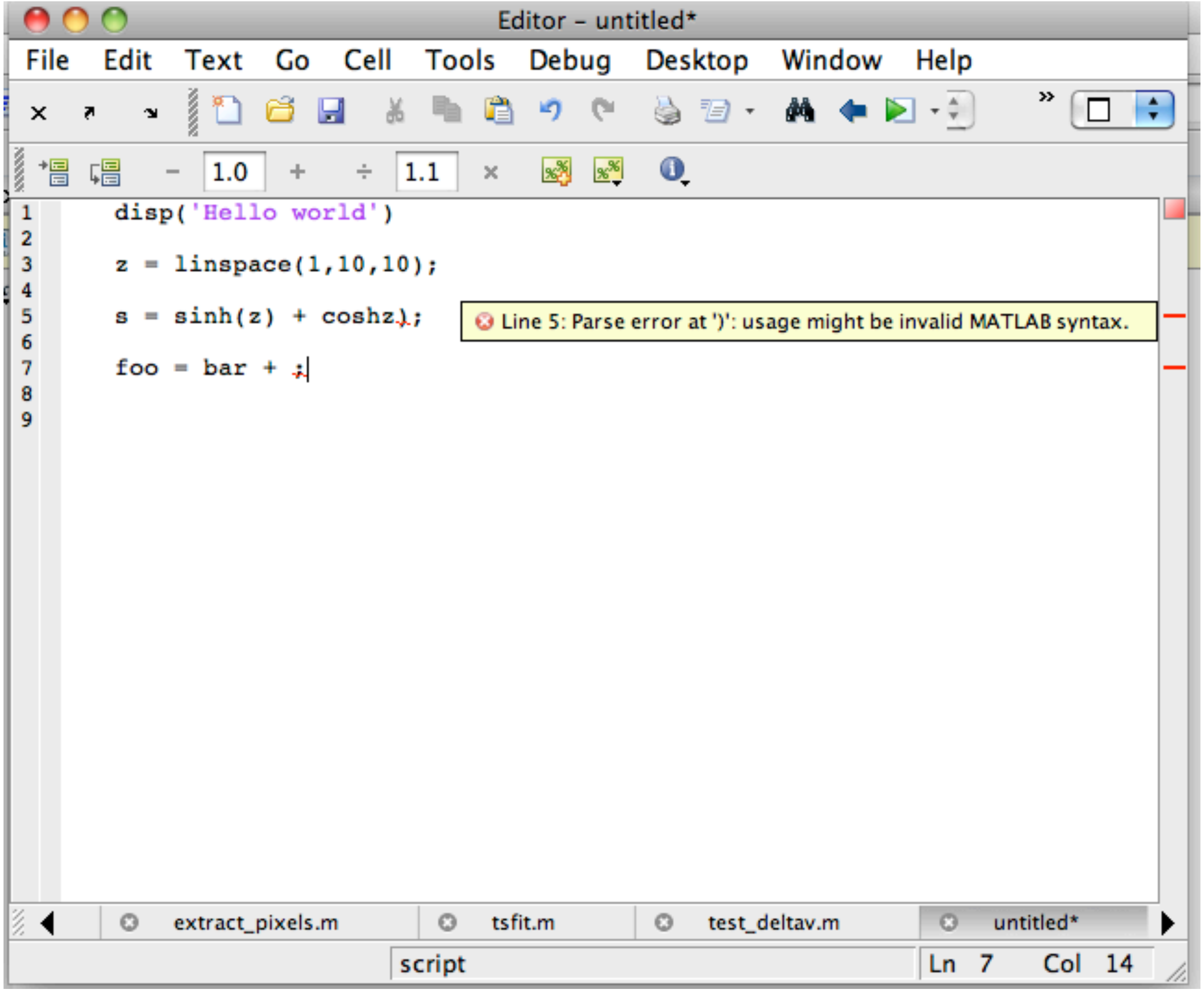
```
>> dsp(halo orld
??? dsp(halo orld
              |
Error: Unexpected MATLAB expression.

>> 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
```

The MATLAB Editor Helps You



```
disp('Hello world')

z = linspace(1,10,10);

s = sinh(z) + coshz);

foo = bar + ;
```

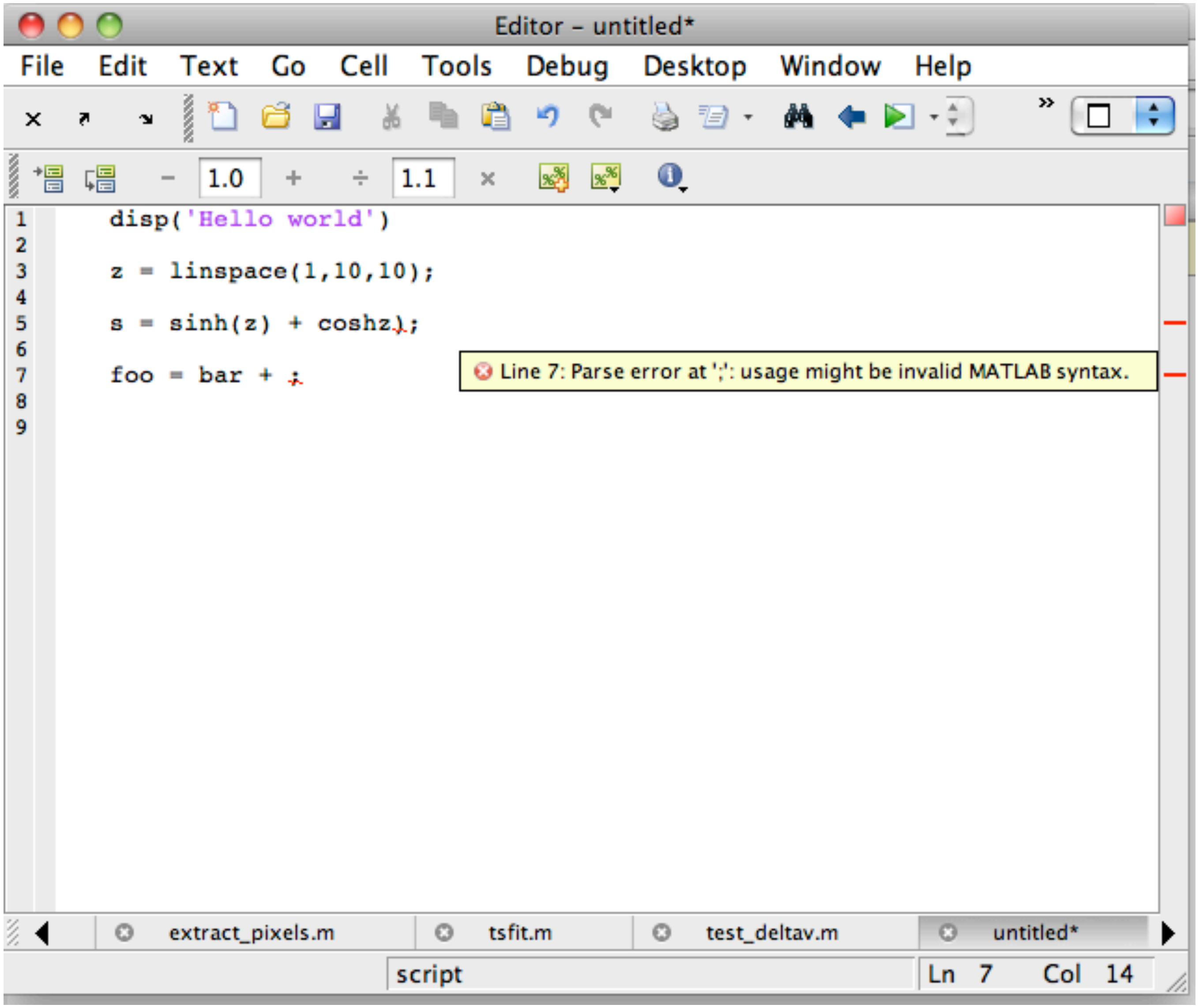
The MATLAB Editor Helps You



```
disp('Hello world')

z = linspace(1,10,10);

s = sinh(z) + coshz);

foo = bar + ;
```

Line 5: Parse error at ')': usage might be invalid MATLAB syntax.

The MATLAB Editor Helps You



```
1  disp('Hello world')
2
3  z = linspace(1,10,10);
4
5  s = sinh(z) + coshz);
6
7  foo = bar + ;
8
9
```

Line 7: Parse error at ';': usage might be invalid MATLAB syntax.

What is a Variable?

- **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

Variables: Name

- USE MEANINGFUL NAMES!
- Must 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`' or '`myCoolVar`' – this is easier to read
- a gazillion style guides exist – punchline: use meaningful names, be consistent (that's hard enough)!

Variables: Type

- What is a type? – Think of sets of numbers in math: N,R,Z, ...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
- 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: Type and Type Conversion

- some languages, e.g. MATLAB, shells, Perl are *weakly typed*: they do automatic type conversions (one type can be treated as another)
  - this is nice at first, occasionally this leads to nasty/ hard to find problems (e.g. string interpreted as number, etc.)
- Other languages are very picky and will tell you that you can't add a real number to a complex number without explicitly converting.
  - Why? It can produce more efficient machine code.
  - Picky vs loose is a design decision

Variables: Value

- A value of the type of the variable: 42, 3.1415926..., false, ‘text string’, i.e., the thing we stuff in the box
- Values can/should change during the runtime of the program. Some languages (not MATLAB) allow you to define a named **constant**, for values that can’t change.
- We need to be able to assign values to variables, and also access (dereference) the values.

Assignment and Access

- Assignment: set the value of a variable
  - MATLAB: `num_glaciers = 105`
  - tcsh scripting: `set filename = “12jun30dena.dat”`
- Access: get the value of a variable
  - MATLAB: `disp( num2str(num_glaciers) )`
  - tcsh scripting: `echo $filename`
- What does this do? (MATLAB)
  - `num_glaciers = num_glaciers + 1`

MATLAB Treats Everything as a Matrix

- Arrays or matrices are lists, vectors, matrices of data (1 to n dimensional)
- 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: `glaciers(5)`, `cov(3,2)`
- Shells allow only vectors (1-D arrays).

Example Arrays

- A numeric array:

Index	1	2	3	4	5	6	7	8	9	10
Value	0	-3.2	1000	NaN	1	5	-90	9999	3.141	0

- Values can be a mix of integers, real and complex numbers.

```
>> foo = [1; 2; 3+i; 4]

foo =
   1.0000
   2.0000
   3.0000 + 1.0000i
   4.0000
```

- You can browse these values in the variable browser within the MATLAB GUI.

Example Arrays

- A string array:

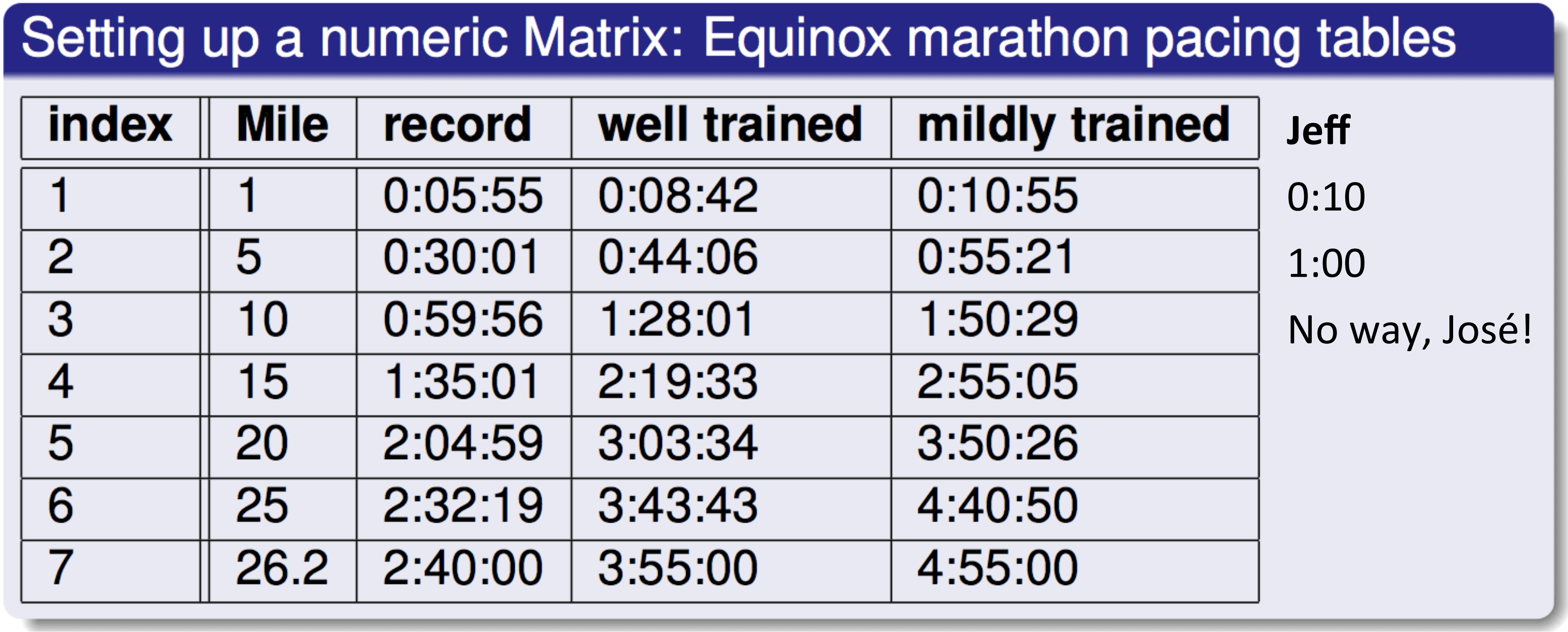
Index	1	2	3	4	5	6	7	8	9	10
Value	H	e	l	l	o		W	o	r	k

- Examples of assigning and accessing strings:

```
>> foo = ‘Hello Work’
>> foo(4)
ans =
    ‘l’
>> foo(1)
ans =
    ‘H’
>> foo(1) + 1
ans =
    73
```

What is going on here!?

An example



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

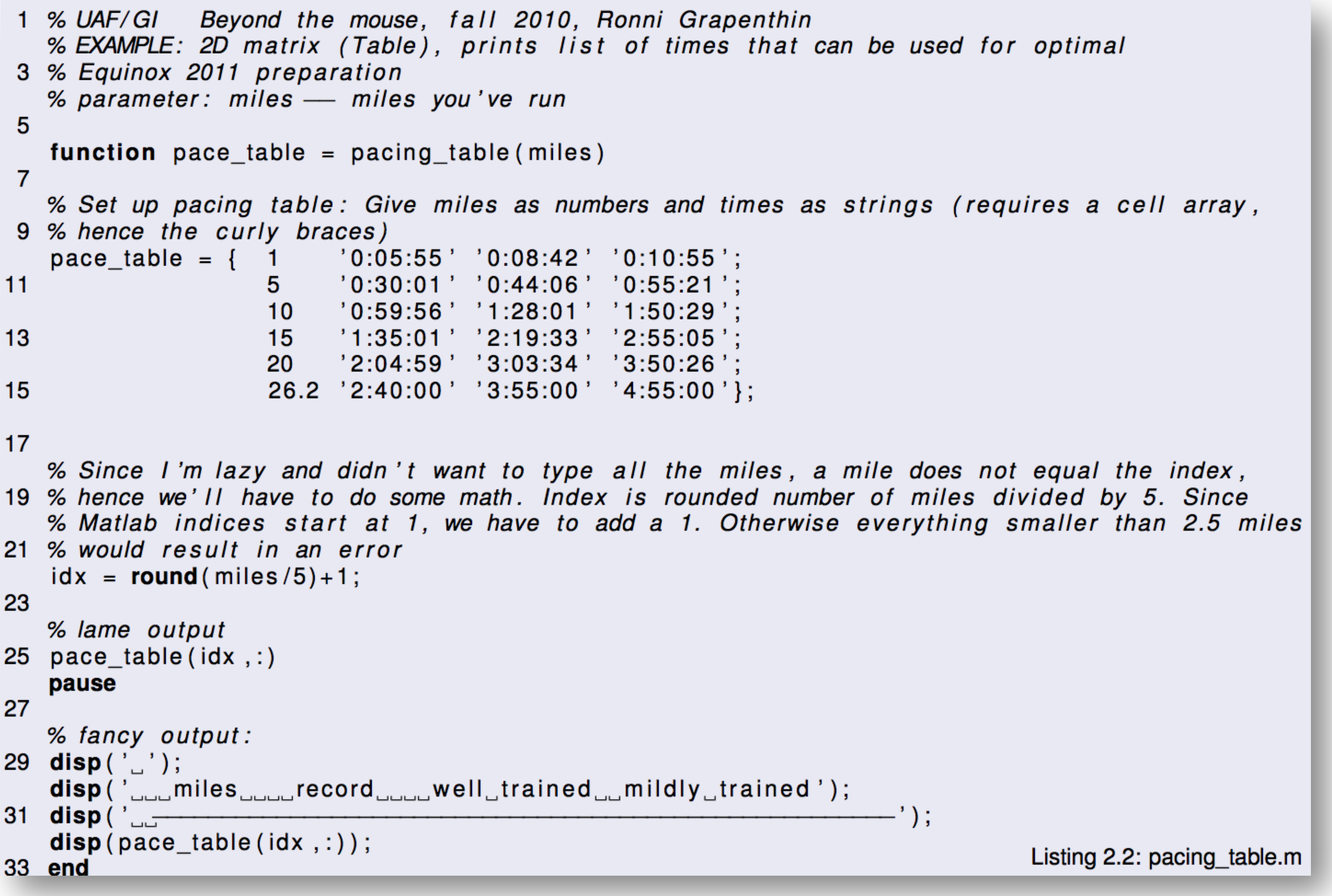
Jeff

0:10

1:00

No way, José!

How to Make the Table



```
% 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

The Importance of Playing Around

- You will learn more if you spend time playing around with the computer, trying to make it do something interesting to you.
- You can start with the exercises, typing them from the lecture notes or even doing a copy and paste
  - You do have to watch out for apostrophes: the straight apostrophe and the curly ones (‘’) are actually ***different characters***!
  - Word processors today “help” you by automatically making curly apostrophes and quotation marks because it looks fancier.