Course reader: *Variables and variable types, basic math*

- *Variables* are place-holders for information in the MATLAB buffer (workspace).
  - Variables can store numbers, strings, cells, structures, and so on.
  - You don't need to declare in advance what kind of information they will store.
  - You can see what variables exist by typing `whos` in the command window, or by using the *Home* → *Layout* → *Workspace* menu option.
  - Variables are deleted by typing `clear varname` (warning: this action is permanent!). Typing `clear` will delete all variables in the workspace (also permanent!).

- You can inspect the contents of a variable by typing it into the *command window*, double-clicking on the variable name in the *workspace* window, or holding the mouse over the variable in the *editor* (you'll learn more about this in the next section).

- There are rules for how to name variables:
  - Cannot start with a number (but may contain a number, e.g., `data4me`)
  - Cannot use special characters like !, @, #, $, %, ^, &, *, (, or). The underscore _ is allowed (e.g., `var_name`).

- There are also best-practice recommendations:
  - Keep variable names short but long enough so you know what they mean. Single-letter variable names should be avoided except in very short loops (more on this in a later video).
  - Select variable names that other people could interpret (e.g., `clientName` is better than `asdf`).

- Square brackets concatenate, curly brackets make cell arrays.

- There are several *classes* of variables. The most commonly used include:
  - **double**: These are numeric (numbers). Probably most of your variables are of this type.
  - **char**: These are characters, like letter.
  - **logical**: These are Boolean and take on values of true (1) or false (0).
  - **cell**: Cells are like cages that enclose pieces of information (could be strings or numbers or even other cells). They are created using curly brackets .
  - **struct**: Structures are arguably the most flexible ways to store data in MATLAB. A structure contains fields of any class, and are indicated by a period (.) between the variable name and one of its fields.

- Be mindful of the differences between * and .* , / and ./ , and ^ and .^
  If there is a period before the operator, it indicates element-wise operations on vectors and matrices. If there is no period, then MATLAB assumes you are trying to implement matrix multiplication or power. If you are surprised by an error when typing `vect1*vect2` then you might have intended to write the element-wise `vect1.*vect2`.

Exercises

1. Creating and working with variables.
(1) Create a variable that contains your name as a string. Do you need to use (a) no quotes, (b) single quotes, or (c) double quotes?
(2) Create variables for people in your family, and assign the values to be their ages (e.g., `john=38;`). Mathematically combine (add, subtract, multiply, etc.) those ages to equal 100.

2. Create two variables `a` and `b`, set them equal to any numbers you like, and compute the following:

a) $\frac{5a}{1-b/3}$ **b)** $a^b - b^a$ **c)** $15^a \times \frac{b}{2a}$ **d)** $\frac{-b+b^2-4a}{2a}$

3. Monetary inflation means that currency now will have less value in the future (and had more value in the past). A formula to adjust the value of currency is xr^y where x is the amount (in dollars, Euros, whatever), r is the inflation rate, and y is years. For example, \$10 now has the same value as $\$10 \times 1.03^{10} = \$13.44$ ten years from now, assuming a constant 3% inflation rate. Similarly, \$10 now has the same value as $\$10 \times 1.03^{-10} = \$7.44$ ten years ago. In the problems below, assume an inflation rate of 3%.
(1) If you are currently 25 years old and have a million dollars, what will the value be when you are 65? (Note: The answer tells you why it's a good idea to invest money rather than keep it in cash!)
(2) The reason you have a million dollars is that you sold a solid-gold bar worth a million. How did you get that gold bar? Easy: You invent a time-machine to go to the past and tell your family to buy gold so you would have a million dollars today. How far back in time do you need to travel if your family had \$50,000 in their time? (Note: you can either solve this analytically or try different values until you are close.)

4. Use the colon operator and square brackets to obtain the following number sequences.

a) 1 2 3 4 8 9 10 11 12 **b)** -4 0 4 -3 0 3 -2 0 2 -1 0 1 **c)** 0 .3 .6 .9 .6 .3 0 -.3

5. Type the following into MATLAB. What do the outputs tell you about when to use curly vs. square brackets, and commas vs. semicolons vs. spaces? Some of these will produce MATLAB errors; figure out why and how to fix the code.

a) `[ 'M' 'i' 'k' 'e' ]` **b)** `[ 'M' 'i','k','e' ]` **c)** `[ 'M';'i';'k';'e' ]`
d) `[ 'M';'i','k','e' ]` **e)** `[ 'M' 'i' 'k' 4 ]` **f)** `[ 'M' 'i' 'k' '4' ]`
g) `[ 'Mi' 'k' 'e' ]` **h)** `[ 'Mi';'k';'e' ]` **i)** `{ 'M' 'i' 'k' 'e' }`
j) `{ 'M','i','k','e' }` **k)** `{ 'M';'i';'k';'e' }` **l)** `{ 'M' 'i' 'k' 4 }`
m) `{ 'M' 'i' 'k' '4' }` **n)** `{ 'Mi';'k';'e' }` **o)** `[ 'M' 'i' 'k' 'e' }`

6. Create a 3-element row vector: `v=[1 4 3]` Typing `v(4)` produces an error because there is no 4^{th} element to access. What happens if you type `v(4)=0` ? Guess the answer first, then test in MATLAB.

Answers

1. `myName = 'mike';`

2. e.g., `a=3; b=-7;`

a) `5*a/(1-b/3)`
b) `a^b - b^a`
c) `15^a * (b/2a)`
d) `(-b+b^2-4*a)/(2*a)`

3. Indeed, inflation means you should invest your money wisely, rather than keep all your savings in a bank account.
(1) `1000000*1.03^-(65-25)`
(2) `50000*1.03^(101.3482)` (thus, around 101 years ago, $50,000 was the modern equivalent of a million dollars)

4. .

a) `[1:4 8:12]`
b) `[-4:4:4 -3:3:3 -2:2:2 -1:1:1]`
c) `[0:.3:.9 .6:-.3:.3]`

5. Knowing when to use square vs. curly brackets is sometimes confusing. My recommending is when you are learning to program, just guess which one might be good and be willing to change. You'll learn quickly which are the best for which circumstances.

6. MATLAB permits this operation. It's a way of expanding a vector without recreating it. But be careful when doing this — dynamically enlarging vectors and matrices is a potential source of confusion and bugs.