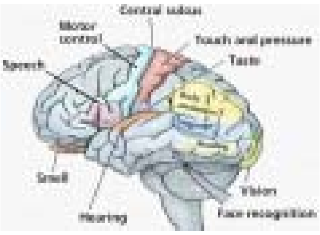

Functional Areas of the Brain

More on Functions

Lecture 8 (Feb ~~15~~ 20)
CS100M – Spring 2007

Announcements

- Prelim 1
  - Feb 22 at 7:30pm
  - Room: Statler Auditorium
  - Reminder: You *must* contact Kelly Patwell (see website) if you have any scheduling difficulties due to other exams
  - Prelim 1 topics: Everything through today
    - Material introduced after today will not appear on the prelim
  - Sample exam questions are available on the course website (solutions will appear shortly)
- Clicker registration
  - If you register and it tells you that you have a duplicate number then you need to exchange your clicker at the Campus Store

Topics

- Reading: No new reading
- Plans for today
  - Continue with user-defined functions
  - Brief review

General Form for a User-Defined Function

```
function outputArg = functionName(arg1, arg2, …)
% One line comment describing the function
% Additional description of function
<executable code which at some point assigns to outputArg>
…
```

- The function definition is stored in the file **`functionName.m`**
- What if the filename and the function name are different?
  - Matlab finds and uses the function by looking at the *filename*
  - The name in the function heading can be different from the filename, but don't do this!
    - Mismatch implies that the name in the function heading is *ignored*; the filename is used

Example: Printing Coin Flips

- You can have a function that returns no value at all
  - Function header: function functionName(arg1, arg2, …)
  - Example calling code: printFlips(10);
- Goal: Create a function printFlips(n) that prints the result (e.g., HTTHT) of n coin flips

```
function printFlips(n)
for k = 1:n
  if rand(1) > 0.5
        fprintf('H');
  else
        fprintf('T');
end
fprintf('\n');
```

Return Values & Function Parameters

- One return value, two parameters
  - function returnValue = myFunction(argOne, argTwo)
    - Usage: x = myFunction(x, 5);
    - Usage: y = 7 + myFunction(44, x);
- Zero return values, one parameter
  - function myFunction(argOne)
    - Usage: myFunction(17)
- Two return value, zero parameters
  - function [retA, retB] = myFunction()
    - Usage: [x, y] = myFunction()

Helper Functions

- For the most part, each of your functions lives in its own file
  - But sometimes you just need a simple helper function
- You can include multiple functions in a single M-file
  - The first function listed in the file behaves normally
    - And its name should match the filename
  - Any remaining functions are accessible only from within this M-file
  - In Matlab, these helper functions are called *subfunctions*
  - The next example uses such a helper function, called diceRoll

Example: Simple Game

- Description
  - Two players take turns rolling a pair of dice
  - The winner is the first player to roll doubles
- Goal: Write a function that plays the game and then reports
  - The winner (Player 1 or Player 2) and
  - The number of dice rolls used
- Which works?
  - roll = round(1 + 5*rand(1));
  - roll = ceil(6*rand(1));

Algorithm

- From the Goal, we can tell that the function should have the following header

```
function [winner, rolls] = game()
```

- Guts of the algorithm

```
while no winner yet
   Roll dice
```

- We have to keep track of
  - Whose turn it is
  - How many rolls have occurred

Questions to Resolve

- How do we change players between Player 1 and Player 2?
  - We want to swap back and forth between 1 and 2
  - How about: player = 3 - player
- How do we test if doubles are rolled?

```
d1 = diceRoll();       % First die
d2 = diceRoll();       % Second die
Test: d1 == d2
```

Putting the Pieces Together

Function header	function [winner, rolls] = game()
	player = 1;
Initialization	d1 = diceRoll();
	d2 = diceRoll();
	rolls = 1;
while d1 ~= d2	while d1 ~= d2
Change player	player = 3 – player;
Roll again	d1 = diceRoll(); d2 = diceRoll();
Increment rolls	rolls = rolls + 1;
Report winner & rolls	end
	winner = player;

Global Variables

- Sometimes it's useful to have a variable that's shared by all of your functions
  - Example
    - In order to implement a computer game, you create a large number of functions
    - All (or almost all) of these functions need access to the game board
    - You can either (1) include the game board as an argument for each function or (2) make the game board *global*
- Each function that uses the game board must include a statement of the form **`global gameBoard`**
  - This statement must appear *before* the first use of gameBoard in the function
- In general, you can use **`global var1 var2 var3 …`**
- It is considered bad programming style to use a large number of global variables

Persistent Variables

- A *persistent variable* is a function variable that is preserved unchanged between calls to the function
- You can create persistent variables with the following statement

```
persistent var1 var2 var3 …
```

- An example use: Can use a persistent variable to count the number of times that a function is called
- Note that a persistent variable is stored outside a function's workspace since a function's workspace is deleted when we leave the function

Walking Randomly

- Write a function that performs a "random walk" in the plane
  - Possible moves are left, right, up, or down
  - Input parameters are the number of steps, n, and the initial coordinates, x0, y0
  - Return the final coordinates xFinal, yFinal

Prelim 1 Topics

- Variables (scalar)
- Assignment statements
- Built-in functions: max, min, abs, rand, sin, cos, tan, asin, acos, atan, exp, log, log2, log10, round, floor, ceil, fix, mod
- Selection: if, if-else, if-elseif-else
- Iteration: for-loop, while-loop
- User-defined functions
- Good programming style

- Material from
  - Lectures (through today)
  - Sections (Exercises 1-5)
  - Reading (Chapters 1-4)
  - Homework (Projects 1 & 2)
- You don't have to memorize the built-in functions
  - The names of any built-in functions that you need will be listed on the prelim
  - You are expected to know *how* to use them