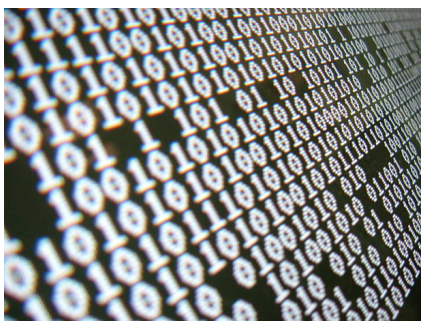
Logic and Loops

MATH-151: Mathematical Algorithms in Matlab

August 28, 2023

Formal Logic Statements

- In general, a **logical statement** is a declarative sentence to which one (and only one) of the terms "true" or "false" can be meaningfully applied
  - Air Bud is a dog that plays basketball. (True statement!)
  - $\pi = 3$ (False statement!)
  - This statement is false. (Neither! Can't meaningfully apply true or false)
  - Let's go Air Bud! (This isn't a statement, not declarative)
- Because computers "think" in 1s and 0s, logical statements are very useful in computing.
  - 1 means true. 0 means false.
- Logical statements allows us to turn parts of our code "on" and "off" using control statements



Logical Statements in Code

Great! We now know what a logical statement is, but how can we make our logical statements in Matlab? The most common ones we will use are **relational statements**

- Equality (==)
  - This returns true if the values on both sides of the == are the same, and false if not
  - Example: We can see if a value is even by seeing if dividing by two gives us an integer

```
x/2 == round(x/2)
```

- Inequalities (>, >=, <, <=)
  - Checks how one value relates to another
  - Example: Check if grade is an A

```
score >= 93
```

- Not equality ~=
  - This is the opposite of equality.
  - Example: Value is odd if dividing by two is not an integer

```
x/2 ~= round(x/2)
```

```
>> x = 5;
>> x == 5
ans =
  logical
   1
>> x == 8
ans =
  logical
   0
```

```
>> x >= 5
ans =
  logical
   1
>> x > 5
ans =
  logical
   0
>> x < 100
ans =
  logical
   1
```

```
>> x ~= 5
ans =
  logical
   0
>> x ~= 42
ans =
  logical
   1
```

If ... Else .. Statements

- Sometimes in life we have contingencies
  - If it is raining, bring an umbrella. Else, don't bring an umbrella.
- Algorithms have these as well! Consider the median
  - If there is an odd number of data values, take the center value
  - If there is an even number of data values, average the center two.
- `if ... else ...` statements in code allow us to do this. See our example for the median

```
N = 123;    % Our number of samples
if N/2 == round(N/2)    % is N even?
    % If N is even, do this
    % Average center values for median
else
    % If N is not even, do this
    % Take center value for median
end
```

Elseif Statements

- In many cases the decisions are more than just two options!
  - If it is morning, eat breakfast. If it is noon, eat lunch. If it is evening, eat dinner. Otherwise, don't eat a meal.
- We can add additional conditions between our `if` and `else` using `elseif`.
- For example my vehicle code controlled differently based on range from object, and whether or not it has detected the object yet

```
range = 50; object_detected = true;

if range > 100                          % Transit Mode
    % Command vehicle to transit to area
elseif range < 100 && ~object_detected % Search Mode
    % Command vehicle to search area
elseif object_detected && range > 1    % Approach Mode
    % Command vehicle to approach object
elseif object_detected                 % Investigate Mode
    % Command vehicle to investigate object
else                                   % Standby Mode
    % Command vehicle to standby
end
```

- The computer starts at the `if` statement, and works its way down the `elseif` statements until one of them are true.

Boolean Operators

Boolean operators are functions allowing us to link together multiple logical statements into one

- and (`p && q`)

`p && q`	p = 1	p = 0
q = 1	1	0
q = 0	0	0

- or (`p || q`)

`p || q`	p = 1	p = 0
q = 1	1	1
q = 0	1	0

- xor (`xor(p,q)`)

`xor(p,q)`	p = 1	p = 0
q = 1	0	1
q = 0	1	0

- not (`~p`)

p	1	0
~p	0	1

Why Loops?

- Suppose we want to calculate the sum of integers from 1 to 17

```
total = 0;
total = total + 1;
total = total + 2;
total = total + 3;
total = total + 4;
total = total + 5;
total = total + 6;
total = total + 7;
total = total + 8;
...
```

- That is repetitive and annoying to read and type out. There has to be a better way.
- This is where loops come into play, they allow us to tell the computer to repeat statements that follow a similar structure.

For Loops

- Let's look at that sum of the first 17 integers using a `for` loop

```
total = 0;
for ii = 1:17
    total = total + ii;
end
```

- This is much better to look at!
- Let's break down the pieces
  - `total = 0;` **initializes** our sum total at 0.
  - `for ii = 1:17` is telling the computer to repeat everything between this line and `end` for every value of ii starting with `ii=1` and ending at `ii=17`.
  - `total = total + ii;` takes our current value for `total` and add on our value of `ii` before stepping to our next **iteration**, or repeat of the code.
- Note that we indented the `total = total + ii;` line to indicate it is a line the loop is repeating.

While Loops

- Suppose instead we want to find how many numbers to add up before our sum gets greater than 100, this is a task better left for a `while` loop

```
total = 0;
ii = 0;
while total < 100
    ii = ii + 1;
    total = total + ii;
end
```

- `while total < 100` tells the computer to repeat the loop while the logical statement `total < 100` is true
- In this case we have to update our **counter** `ii` ourselves. It helps us count how many times the loop is repeated.

When to Use Each Loop

- In general, for any loop you need to perform, you could use either a `for` or `while` loop. But one is usually preferable based on the context
- The general rule of thumb is
  - Use a `for` loop when you know how many times you need to repeat your loop
  - Use a `while` loop when you are repeating the loop until some event occurs