

Report 2

Maximilian Fernaldy - C2TB1702

Problem 1

Find all numbers of 3 digits such that the sum of the cubes of its digits equals the number itself; an example is 153, because $1^3 + 5^3 + 3^3 = 153$.

My Line of Thinking

I wanted my solution to be somewhat unique to the ones proposed in the second and third problem. The MATLAB documentation had [this](#) page on it that showed promise, but none of the examples were exactly the one I needed. I started experimenting to see if I can write certain values that match other values in another array to a third array. This was my first attempt:

```
% Experiment 2.1 Trying out conditionally accessing array elements

% Initialize arrays A and B as random numbers
A = [2,3,4,6,7,8]
B = [1,3,5,7,9]

% C will only display numbers in A that are also in B
C = A(A == B)
```

This didn't work, and it threw an error which basically said that to compare arrays **A** and **B**, they need to be the same size.

```
>> conditionalAccess

A =

     2     3     4     6     7     8

B =

     1     3     5     7     9

error: mx_el_eq: nonconformant arguments (op1 is 1x6, op2 is 1x5)
error: called from
    conditionalAccess at line 8 column 3
```

To confirm this, I added another element to `B` and ran the program again. As expected, 3 appears. However, 7 **does not**. I suspect this is because 7 in `A` is in a different position than 7 in `B`.

```
>> conditionalAccess

A =

     2     3     4     6     7     8

B =

     1     3     5     7     9    10

C = 3 % only 3 appears, when there's also a 7 in both arrays
```

I changed up the arrays, and now `B` is a copy of `A`, with some exceptions.

```
% Experiment 2.1 Trying out conditionally accessing array elements

% Initialize arrays A and B as random numbers
A = [2,3,4,6,7,8];
B = [2,1,4,9,7,10];
% C will only display numbers in A that are also in B
C = A(A == B)
```

Output is finally as expected.

```
>> conditionalAccess

C =

     2     4     7
```

Notes from this experiment

- Logical indexing arrays work in a very specific way.
- First, the compared arrays *must* be the same size.
- Second, comparisons work in a sequenced manner, which means a number `x` in position 2 and a number `x` in position 4 will not appear as equivalent.

Script for Problem 1

Keeping the results of experiment 2.1 in mind, I started writing the program. First, initialize an array for the numbers 100 through 999, which are all the numbers containing 3 digits.

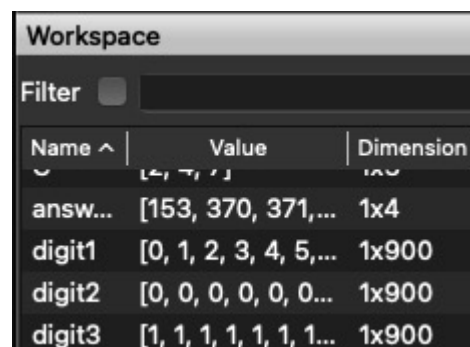
```
% Answer for problem 1

% First, we initiate the array "num" which contains all the numbers
% from 100 through 999.
num = 100:999;
```

Next, I separate the digits making up the numbers in the list. I'm following the convention used in problem 2 and 3, where `i1` is the last digit, `i2` is second to last and `i3` is the first digit. I just used different names for the digits, with `digit1`, `digit2` and `digit3`.

```
digit1 = mod(num, 10); digit2 = mod(floor(num/10), 10); digit3 = floor(num/100);
```

As the digits are now separated, we can now do operations on the individual digits. Recall that `digit1`, `digit2` and `digit3` are all arrays, not single values. This is evident by looking at the "Dimension" tab of the workspace pane.



Name ^	Value	Dimension
answ...	[153, 370, 371, ...	1x4
digit1	[0, 1, 2, 3, 4, 5, ...	1x900
digit2	[0, 0, 0, 0, 0, 0, ...	1x900
digit3	[1, 1, 1, 1, 1, 1, ...	1x900

```
% We can now get the sum of the cubes of all the numbers in the array, and
% collect them in the array "sum".
sum = digit1.^3 + digit2.^3 + digit3.^3;
% as mentioned in the ppt material for Lecture 2, .^ exponentiates each
% element of the array.
```

Now for the final step, we will use conditional/logical indexing to store values that satisfy our requirement into the array `answers`.

```
% And then we store the numbers where the sum of the cubed digits equal itself
% in the array "answers".
answers = num(num == sum);
```

To read this in non-machine language:

Into the array called `answers`, we store the numbers in `num` that are also in `sum`, and exist in the same position in both arrays.

Output

Running the complete script, we get this output:

```
>> CAPS_02_C2TB1702_threeDigitSum1

    153    370    371    407
```

Which are the correct answers.

Problem 2

Revise the script below to find these numbers:

```
for i = 100:999
    i1 = mod(i, 10);
    i2 = mod(floor(i/10), 10);
    i3 = floor(i/100);
    disp([i3 i2 i1])
endfor
```

Hint: This script scans every three-digit number and gets its three digits

My Line of Thinking

I tried running the script to get an idea of what it does. It outputs the following:

```
>> initialproblem2

    1    0    0
    1    0    1
    1    0    2
    ...
    9    9    7
```

```
9 9 8
9 9 9
% Note: I used ... to shorten the output. The actual output is hundreds
% of lines in length.
```

The program simply iteratively displays the numbers from 100 through 999, after separating each of them into three elements of an array.

I immediately thought to move the `disp` function inside an `if` statement so that it only displays the numbers we want. The condition, of course, being that the sum of the cubes of all three digits equal the number itself.

Script for Problem 2

```
% Answer for problem 2

for i = 100:999
    i1 = mod(i, 10);
    i2 = mod(floor(i/10), 10);
    i3 = floor(i/100);
    if i1^3 + i2^3 + i3^3 == i
        disp([i3 i2 i1])
        % we only display the numbers if the condition is satisfied.
    endif
endfor
```

Running the above script returns this output:

```
>> CAPS_02_C2TB1702_threeDigitSum2

1 5 3
3 7 0
3 7 1
4 0 7
```

As expected, we get the values 153, 370, 371 and 407, which are the same numbers we got in Problem 1. However, I want the output to match Problem 1. I think it's more usable in scenarios where we need them to be actual numbers instead of the digits being separated as elements in rows. In order to do this, we can just alter the way we display the answers.

```
% Answer for problem 2

for i = 100:999
    i1 = mod(i, 10);
```

```

i2 = mod(floor(i/10), 10);
i3 = floor(i/100);
if i1^3 + i2^3 + i3^3 == i
    answer = i3*100 + i2*10 + i1;
    disp(answer)
    % This way, the numbers are displayed as actual numbers.
endif
endfor

```

The output is as follows:

```

>> CAPS_02_C2TB1702_threeDigitSum2

153
370
371
407

```

Problem 3

Write a script that finds the same numbers in a different way by filling in the blanks below:

```

for i3 = 1:9
    for i2 = 0:9
        for i1 = 0:9

            endfor
        endfor
    endfor
endfor

```

With this approach, we are using nested for loops to iterate through all possible combination of digits, checking every time if the sum of the cubes of the digits equal the number itself.

Script for Problem 3

This problem is relatively similar to Problem 2. However, in this script, the variable `i` is not defined, as we are not iterating through the numbers, and through the digits instead. So in order to compare the sum of the cubes of the digits with the number, we have to first define `i` first for every combination.

```

% Answer for problem 3

for i3 = 1:9
    for i2 = 0:9
        for i1 = 0:9
            i = i3^3 + i2^3 + i1^3;
            if i3*100 + i2*10 + i1 == i
                disp(i)
            endif
            % if we land on a combination of i3, i2 and i1 that satisfies the
            % condition, we display the number.
        endfor
    endfor
endfor

```

Note that we can also just do this, but I believe defining `i` is clearer to someone reading the code.

```

% Answer for problem 3

for i3 = 1:9
    for i2 = 0:9
        for i1 = 0:9
            if i3*100 + i2*10 + i1 == i3^3 + i2^3 + i1^3
                % Right side of the equal sign in the if statement is the number i,
                % left side of the equal sign is the sum of the cube of the digits.
                disp(i3*100 + i2*10 + i1)
            endif
        endfor
    endfor
endfor

```