

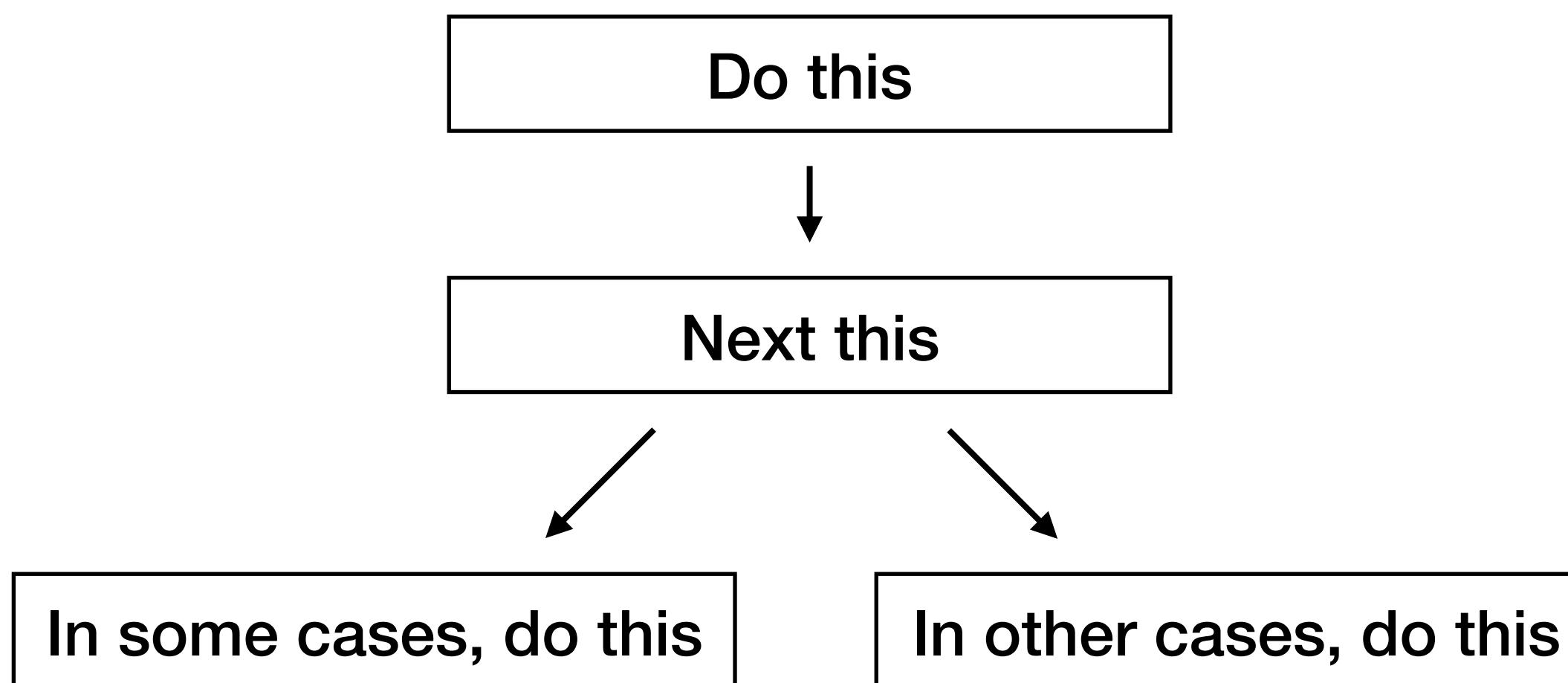
Control Flow II: Loops

PNI Summer Internship 2020
Mai Nguyen

Control flow

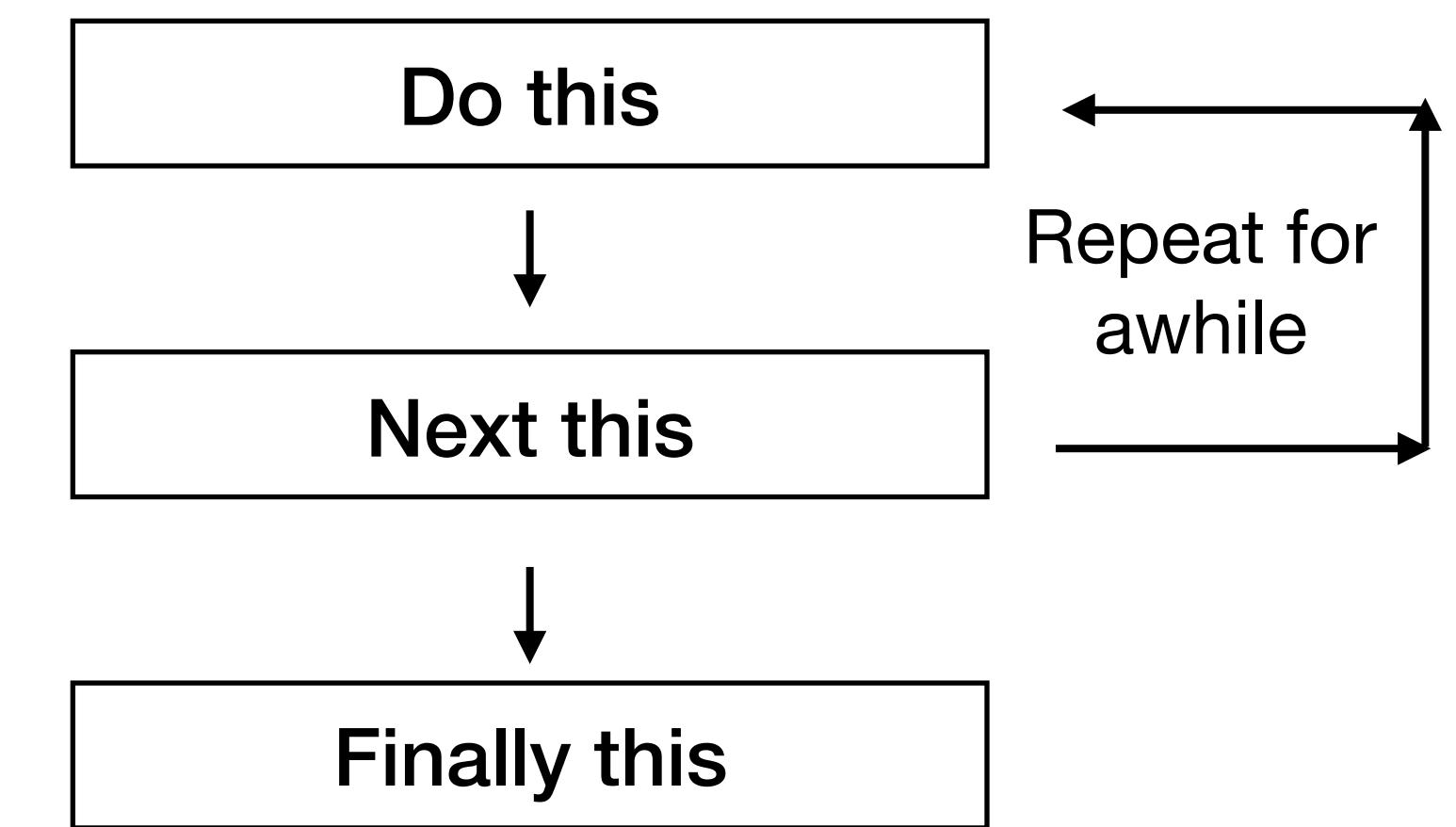
- Control flow allows us to control which code is run (conditionals) and how many times (loops)

Conditionals



if/else, switch

Loops

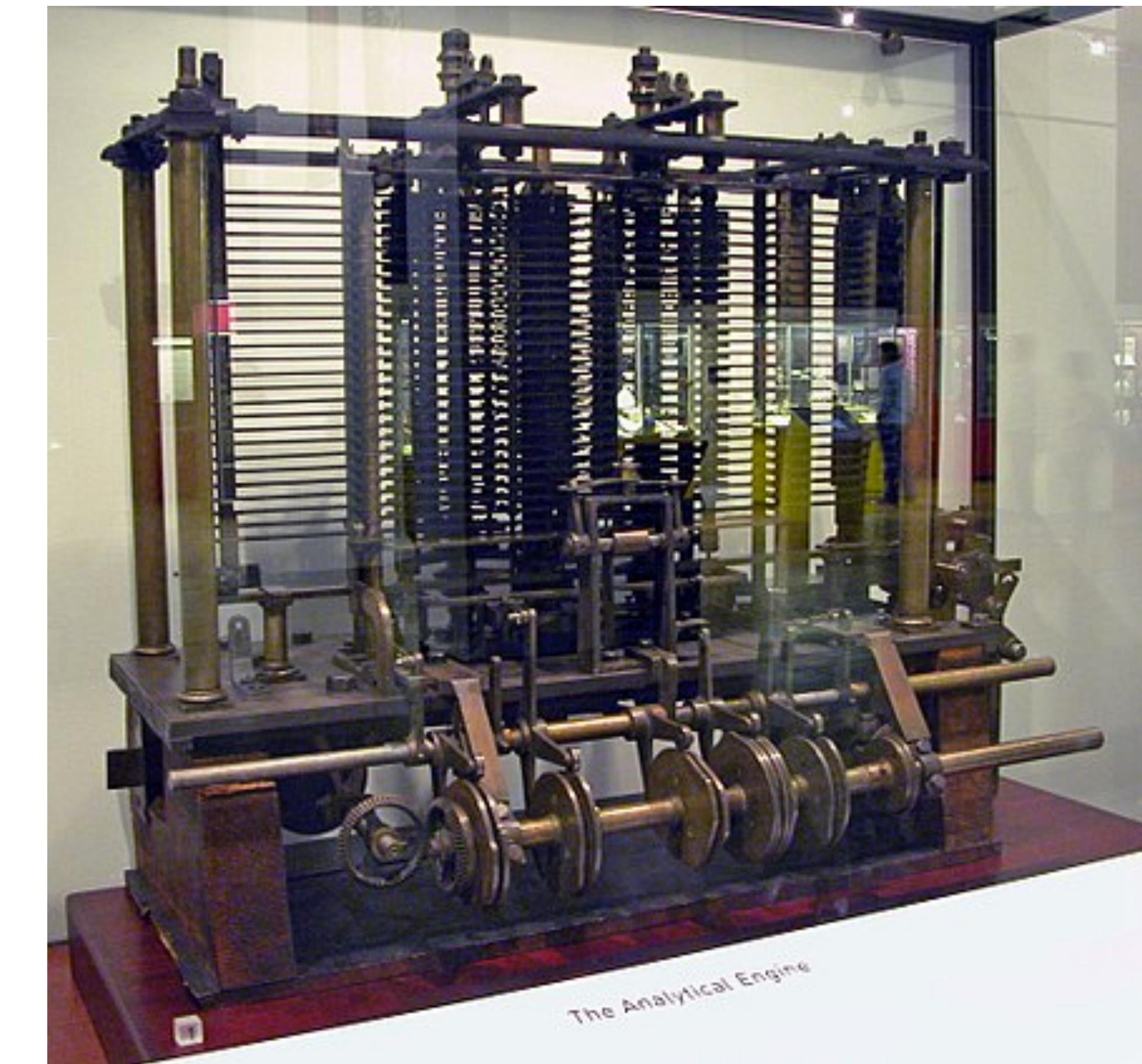


for loops, while loops

Ada Lovelace and the first software loop



Ada Lovelace, 1815-1852



Model of Charles Babbage's Analytical Engine

For loops

- Allows for repeating a chunk of code for a specific number of times
- Print the numbers 1-5

Print each number individually

```
disp(1)  
disp(2)  
disp(3)  
disp(4)  
disp(5)  
1  
2  
3  
....
```

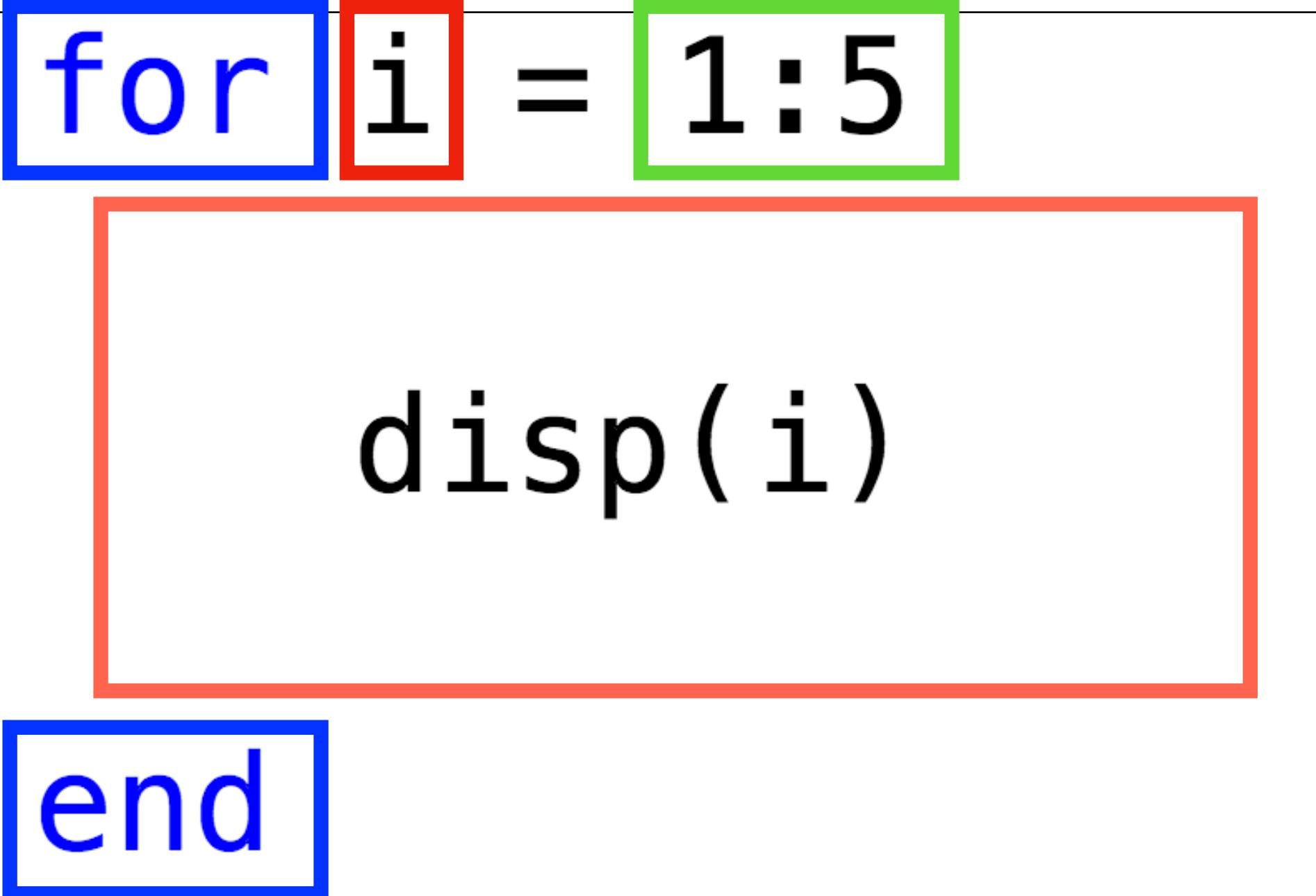
Use for loop

```
for i = 1:5  
    disp(i)  
end  
1  
2  
3  
....
```

Now print
numbers 1-1000

For loops: syntax

```
9 - %  
10 for i = 1:5  
11     disp(i)  
12 end  
13 - %
```



- Start with keyword **for**
- **Loop index**, in this case **i** but can be anything
- **Vector counter**, on each iteration of the loop, the loop index is assigned the next value in the vector
- Close with keyword **end**
- **Loop body** is executed each iteration

Python aside:

MATLAB

```
for i = 1:5  
    disp(i)  
  
end
```

Python

```
for i in range(5):  
    print(i+1)
```

For loops: loop index and vectors

Count 1-5

```
>> % an example
for i = 1:5
    fprintf([num2str(i) ' '])
end
```

1 2 3 4 5

Count vector of random nums

```
>> % an example
for n = [1 3 4 99 0]
    fprintf([num2str(n) ' '])
end
```

1 3 4 99 0

Count through vec variable

```
>> % an example
vec = [-1 1 3 5];
for val = vec
    fprintf([num2str(val) ' '])
end
```

-1 1 3 5

Exercises 1

- **Ex 1.1:** Write a for loop that counts backwards from 10 to 1.
- **Ex 1.2:** Write a for loop that squares each values in a 1x5 vector of random numbers and displays it
- **Ex 1.3:** On your own: Write a for loop takes the square root (use `sqrt()` function) for the even numbers from 1 to 10 and displays it

For loops: assign values to an array

- Often times use loops to fill values in an array

```
>>  
for i = 1:10  
  
    my_vec(i) = i * 2;  
  
end
```

For loops: assign values to an array

- Better practice: pre-define an vector of nans or zeros and fill

```
my_vec = NaN(1,10);
for i = 1:length(my_vec)

    my_vec(i) = i*2;

end
```

For loops: assign values to an array

- Fill matrix rows

```
>> my_matrix = NaN(5,10);

for i = 1:size(my_matrix,1)
    my_matrix(i,:) = i;

end
```

Python aside: assign values to an array

- Typically you don't do this kind of “pre-allocate and fill” arrays or numpy arrays in Python
- Instead, use `append()` to add values to end of an array

Exercise 2

- **Exercise 2.1**
 - Write a for loop that increments i from 1 to 10. On each iteration, get a random number from 1 to i and store to a vector.
 - Modify the for loop to pre-allocate memory in a vector. Display the final vector
- **Exercise 2.2 (on your own)**
 - Write a loop that increments n from 7 to 2. On each iteration, calculate n^n and store to a pre-allocated vector.
 - Display the final vector.

For loops: nested loops

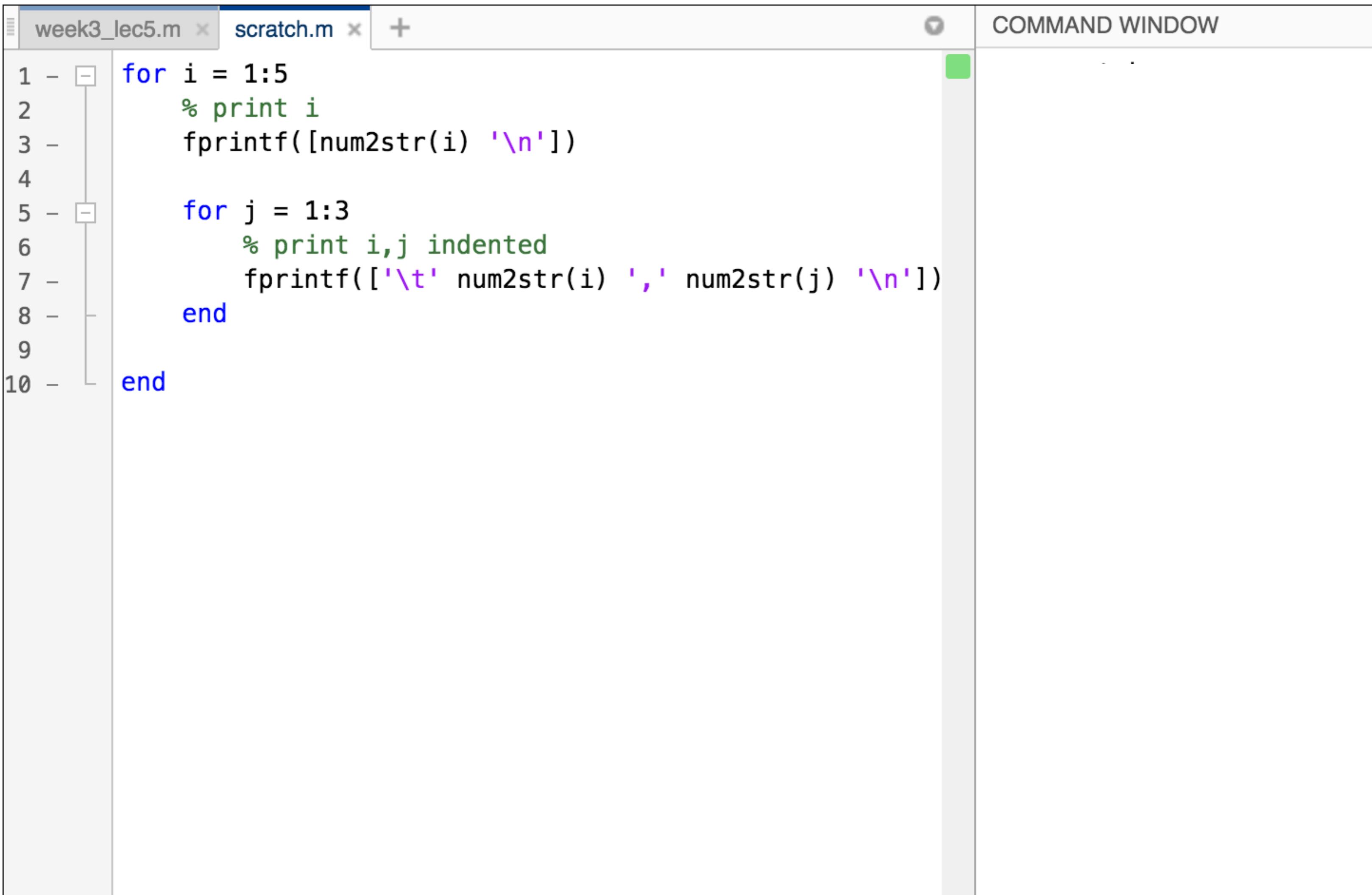
The image shows a screenshot of a MATLAB code editor window titled "scratches.m". The code contains two nested for loops. The outer loop (lines 1-3) iterates over i = 1:5, printing each value on a new line. The inner loop (lines 5-8) iterates over j = 1:3, printing the values of i and j on each line, with j being indented under i. The code is highlighted with blue and red boxes around the respective loops.

```
1 - for i = 1:5
2 - % print i
3 - fprintf([num2str(i) '\n'])
4 -
5 - for j = 1:3
6 - % print i,j indented
7 - fprintf(['\t' num2str(i) ',' num2str(j) '\n'])
8 - end
9 -
10 - end
```

- Outer loop

- Inner loop

For loops: nested loops



The image shows a MATLAB IDE interface. The left pane displays the code in a file named 'scratch.m'. The code contains two nested for loops. The outer loop iterates over i = 1:5, printing each value on a new line using fprintf. The inner loop iterates over j = 1:3, printing the values of i and j on the same line, separated by a tab character (\t). The right pane is the 'COMMAND WINDOW' where the output of the script will be displayed.

```
week3_lec5.m x scratch.m x +
1 - for i = 1:5
2 - % print i
3 - fprintf([num2str(i) '\n'])
4 -
5 - for j = 1:3
6 - % print i,j indented
7 - fprintf(['\t' num2str(i) ',' num2str(j) '\n'])
8 - end
9 -
10 - end
```

For loops: nested loops

The image shows a MATLAB IDE interface. On the left, there are two tabs: 'week3_lec5.m' and 'scratch.m'. The 'scratch.m' tab is active, displaying the following MATLAB code:

```
1 - for i = 1:5
2   % print i
3   fprintf([num2str(i) '\n'])
4
5   for j = 1:3
6     % print i,j indented
7     fprintf(['\t' num2str(i) ',' num2str(j) '\n'])
8   end
9
10 end
```

To the right is the 'COMMAND WINDOW' pane, which displays the output of the code. The output is highlighted with a red box:

```
>> scratch
1
1,1
1,2
1,3
```

Annotations on the right side of the image explain the output:

- "On first iteration of outer loop, i = 1" points to the number 1.
- "Then iterates through the inner loop, j = 1, j = 2, j = 3" points to the three lines of output starting with 1,1.

For loops: nested loops

The image shows a MATLAB IDE interface. On the left, there are two tabs: 'week3_lec5.m' and 'scratch.m'. The 'scratch.m' tab is active, displaying the following MATLAB code:

```
1 - for i = 1:5
2   % print i
3   fprintf([num2str(i) '\n'])
4
5   for j = 1:3
6     % print i,j indented
7     fprintf(['\t' num2str(i) ',' num2str(j) '\n'])
8   end
9
10 end
```

To the right is the 'COMMAND WINDOW' pane, which shows the output of the code. The output is:

```
>> scratch
1
1,1
1,2
1,3
2
2,1
2,2
2,3
```

A red box highlights the output for the second iteration of the outer loop, where $i = 2$. This visual cue indicates that the code has just completed its second iteration of the outer loop and is now in the process of executing the inner loop for $j = 1, 2, 3$.

On second iteration of outer loop, $i = 2$

Then iterates through the inner loop, $j = 1, j = 2, j = 3$

Exercises 3

- **Ex 3.1:** Create a 3x5 matrix of NaNs. Using nested for loops, fill each element of the matrix with the product of the row and column number.
- **Ex 3.2:** [on your own] Create a 3x4 matrix of 0s. Using nested for loops, fill each element of the matrix with the square of the sum of the row and column $(\text{row}+\text{col})^2$

While loops

- **For loops** iterate for a specific number of times specified in the loop counter
- **While loops** continue until a condition is met

While loops

Increment counter until reach 5

```
>> counter = 0;  
  
while counter < 5  
  
    disp(counter)  
    counter = counter + 1;  
  
end
```

0

1

2

3

4

Calculate $5!$ ($5 \times 4 \times 3 \times 2 \times 1$)

```
>> n = 5;  
factorial = 1;  
  
while n > 1  
    factorial = factorial * n;  
    n = n - 1;  
end  
disp(factorial);  
120
```

While loops: syntax

```
3 counter = 0;
4
5 while counter < 5
6     disp(counter)
7     counter = counter + 1;
8
9
10 end
```

- Start with keyword **while**
- Some **condition** that evaluates to true or false; continue repeating until this is false
- Close with keyword **end**
- **Loop body**, repeat this every iteration
- **Modify value** of loop condition
- **Initialize value** of loop condition

While loops: syntax

MATLAB

```
counter = 0;  
  
while counter < 5  
    disp(counter)  
    counter = counter + 1;  
  
end
```

Python

```
i = 1  
while i < 5:  
    print(i)  
    i = i + 1
```

Exercises 4

- **Ex 4.0:** Use a while loop to display the integers up to 10
- **Ex 4.1:** Use a while loop to count up the number of consecutive integers starting from 1 that you need to sum together to get at least 100.
- **Ex 4.2:** [on your own] Use a while loop to find the next prime number greater than a 204. Use the `isprime()` built-in function.

The infinite loop

A screenshot of the MATLAB Online interface. On the left, the MATLAB Drive browser shows a folder structure with 'inf_while.m' selected. The code editor window displays the following MATLAB script:

```
1 while (2+2 == 4)
2     disp('hi i''m infinite')
3 end
```

The right side of the interface is the 'COMMAND WINDOW'. It shows the output of the script: a continuous stream of the string 'hi i'm infinite'. At the bottom of the window, there is a toolbar with a 'Stop' button, which is highlighted with a red rectangle.

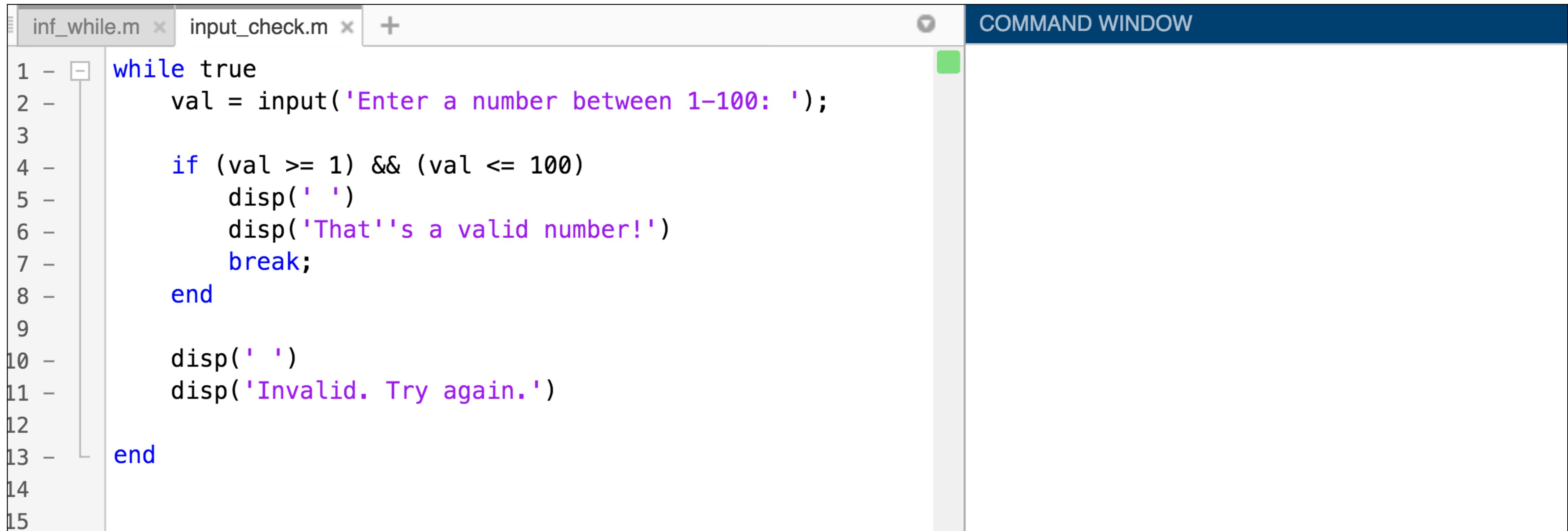
MATLAB desktop: **ctrl + c**

Breaking out of a loop: break

```
6 - count = 1;
7 -  while count < 500
8 -
9 -     if fraction > 100
10 -         break;
11 -     end
12 -
13 - end
14 - disp(count)
```

- **break** allows you to exit the loop early

Breaking out of a loop: break

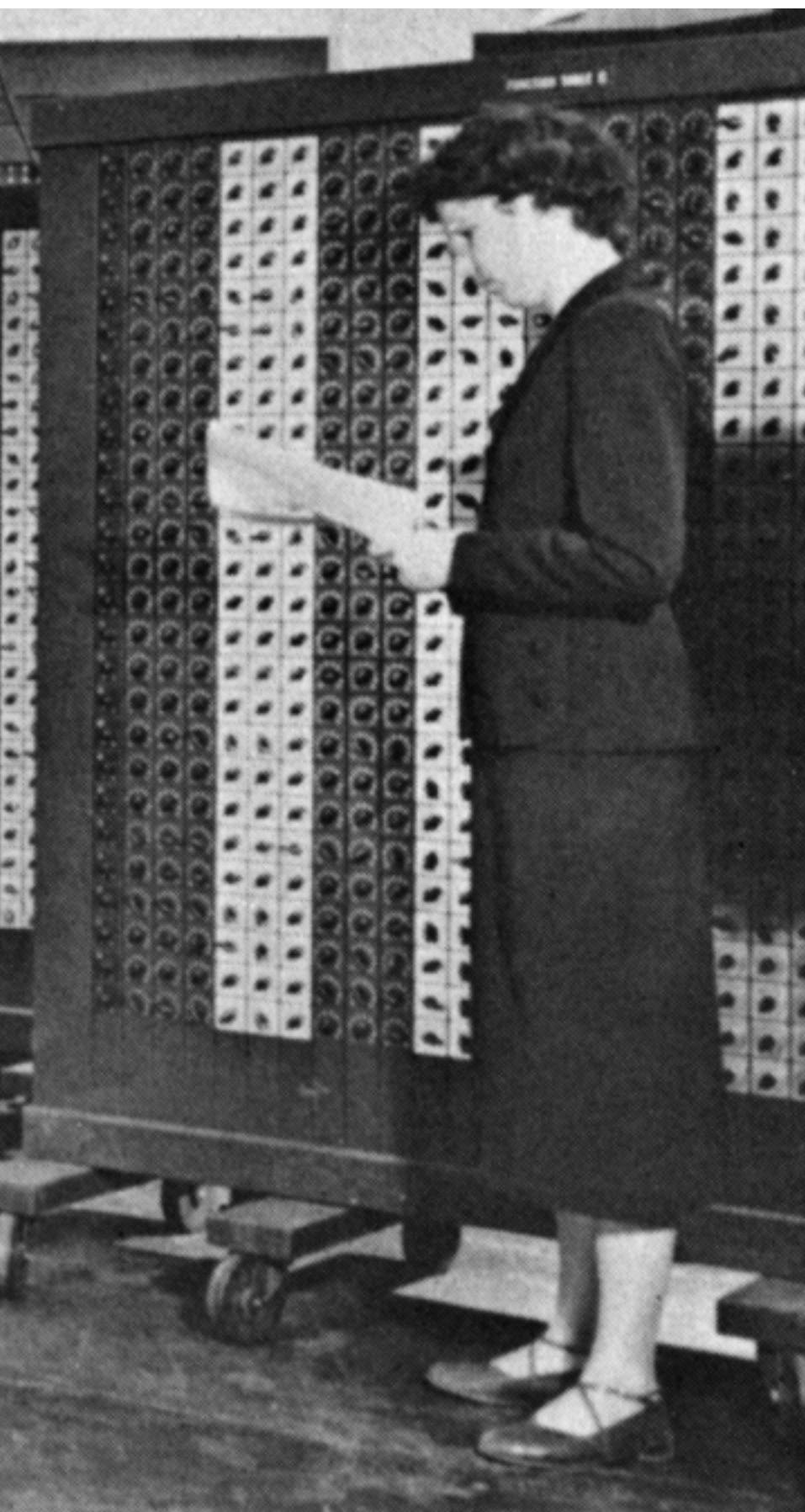


The image shows a MATLAB IDE interface. On the left, there are two tabs: 'inf_while.m' and 'input_check.m'. The 'inf_while.m' tab is active, displaying the following MATLAB code:

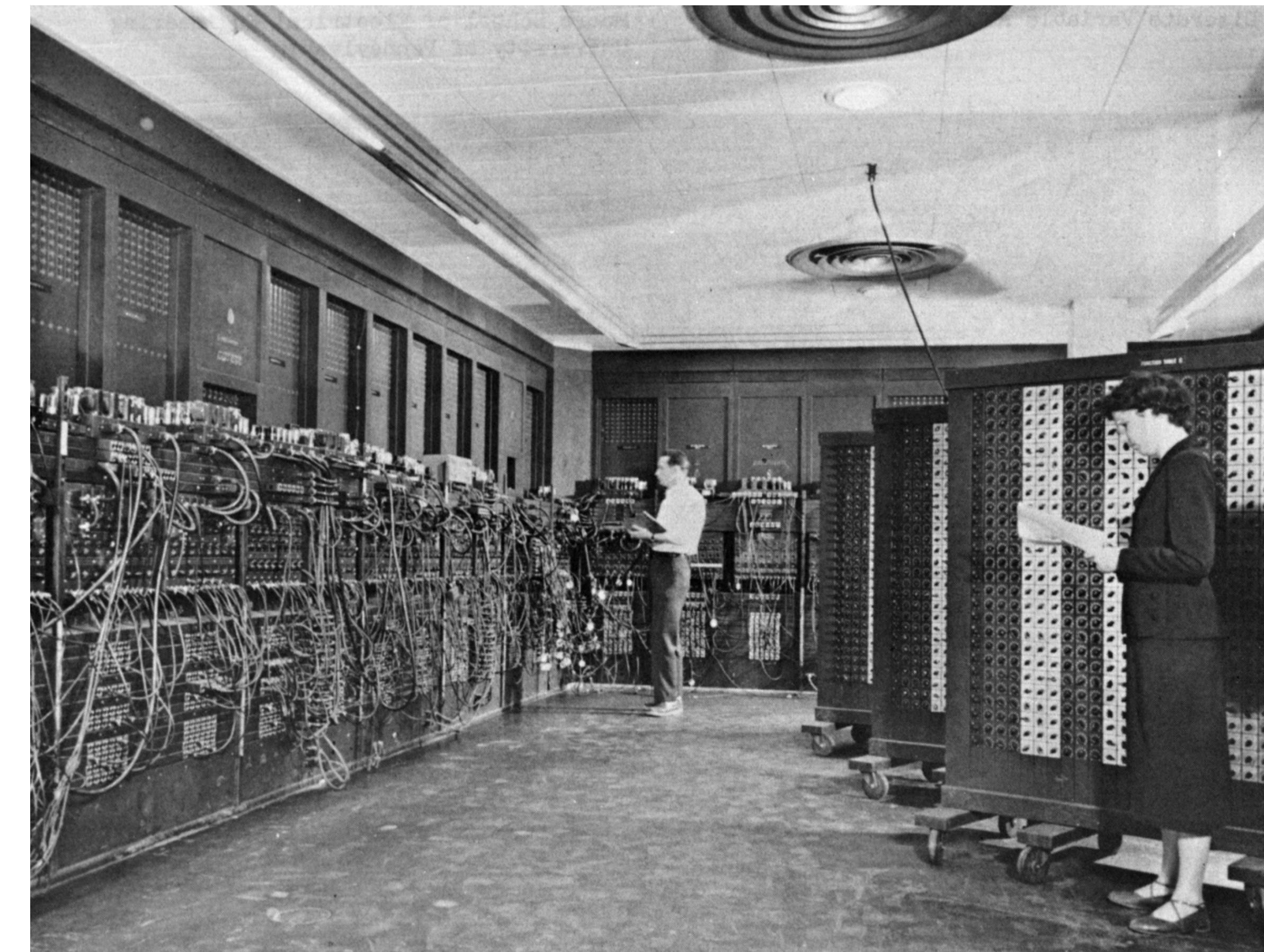
```
1 - while true
2 -     val = input('Enter a number between 1-100: ');
3 -
4 -     if (val >= 1) && (val <= 100)
5 -         disp(' ')
6 -         disp('That''s a valid number!')
7 -         break;
8 -     end
9 -
10 -    disp(' ')
11 -    disp('Invalid. Try again.')
12 -
13 -end
14
15
```

The 'COMMAND WINDOW' tab is visible on the right, showing a green progress bar.

Betty Holberton and the first breakpoint

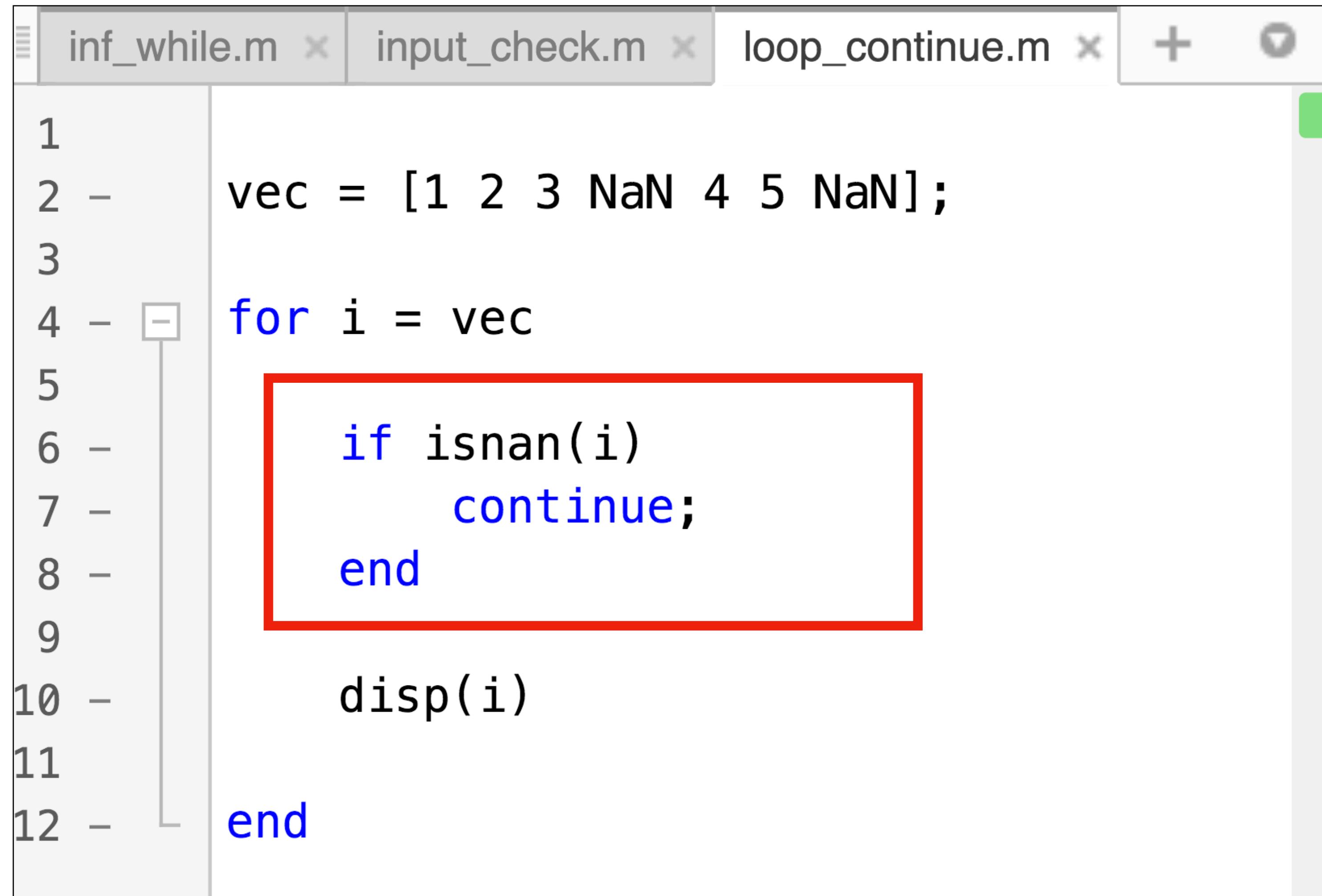


Betty Holberton, 1917-2001



ENIAC ca. 1947-1955

Skip an iteration: continue



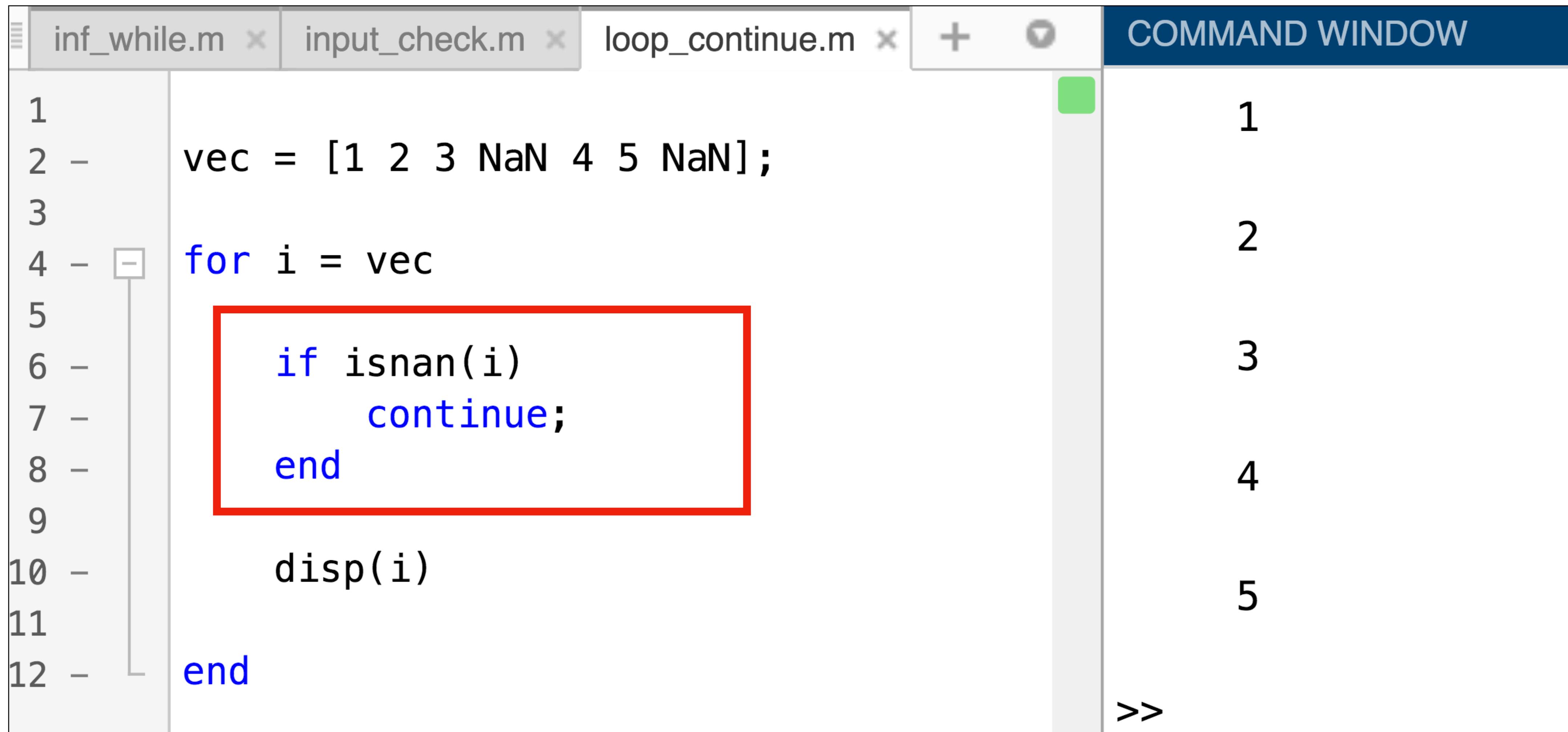
The screenshot shows a MATLAB interface with three tabs at the top: 'inf_while.m', 'input_check.m', and 'loop_continue.m'. The 'loop_continue.m' tab is active. The code in the editor is:

```
1 vec = [1 2 3 NaN 4 5 NaN];
2
3 for i = vec
4
5     if isnan(i)
6         continue;
7     end
8
9     disp(i)
10
11 end
```

A red rectangular box highlights the code block from line 5 to line 8, which contains the condition and the `continue` statement.

- **continue** skips the rest of the loop and goes back to the top

Skip an iteration: continue



The screenshot shows the MATLAB IDE interface with three tabs: 'inf_while.m', 'input_check.m', and 'loop_continue.m'. The 'loop_continue.m' tab is active. The code in the editor is:

```
1 vec = [1 2 3 NaN 4 5 NaN];
2
3 for i = vec
4     if isnan(i)
5         continue;
6     end
7
8     disp(i)
9
10    end
```

A red rectangular box highlights the code block from line 5 to line 8, which contains the condition for skipping iterations with NaN values using the `continue;` statement.

In the 'COMMAND WINDOW' pane, the output is displayed as:

```
1
2
3
4
5
>>
```

For loops versus while loops

Use for loops

- Already know number of iterations
- Iterating through specific values

While loops

- Don't know number of iterations i.e. searching for something

Review

- For loop: `for i = vector`
- While loop: `while cond_is_true`
- `break`
- `continue`