While-Loop

In MATLAB what does

$$a < b < c$$

really mean?

Convention in Math:

$a < b < c$ mean $a < b$ and $b < c$

MATLAB does under the hood:

$$(a < b) < c$$

$a < b$

- if $a < b$, then "true".
- otherwise, then "false".

Contents

Pop Quiz

Understanding Loops

Question 1

How many lines of output are produced by the following script?

```
for k = 100:200
    disp(k)
end
```

A 99

B 100

C 101

D 200

Understanding Loops

Question 2

How many lines of output are produced by the following script?

```
for k = 100:200
    if mod(k,2) == 0
        disp(k)
    end
end
```

→ Q. Is k an even number?

	Print
100	100
101	
102	102
103	
⋮	104
	⋮
200	200

even #s

A 50 B 51 C 100 D 101

mod(k, n) : remainder upon dividing k by n.

- mod(5, 3) = 2 $\quad 5 = 3 \cdot 1 + 2$
- mod(7, 5) = 2 $\quad 7 = 5 \cdot 1 + 2$

FOR-Loop: Tips

- Basic loop header:

```
for <loop var> = 1:<ending value>
```

- To adjust starting value:

```
for <loop var> = <starting value>:<ending value>
```

- To adjust step size:

```
for <loop var> = <starting value>:<step size>:<ending value>
```

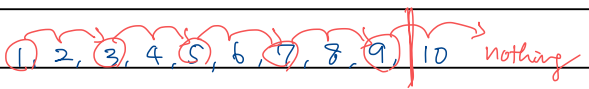
for k = 1 : 5 same as for k = 1 : 1 : 5

Examples

- To iterate over 1, 3, 5, …, 9: [*step size = 2*]

```
for k = 1:2:9
```

or




```
for k = 1:2:10
```

- To iterate over 10, 9, 8, …, 1: [*negative step size*]

```
for k = 10:-1:1
```

Try:

```
for k = 1 : -2 : 10
    disp(k)
end
```

Introduction to `WHILE`-Loop

Need for Another Loop

- `For`-loops are useful when the number of repetitions is known in advance.

  "Simulate the tossing of a fair coin 100 times and print the number of Heads."

  # of rep.

- It is not very suitable in other situations such as

  "Simulate the tossing of a fair coin until the gap between the number of Heads and that of Tails reaches 10."

  We need another loop construct that terminates as soon as $|N_{\mathrm{H}} - N_{\mathrm{T}}| = 10$.

Toss:	H	H	H	T	T	H	T	T	T	...
N_H:	1	2	3	3	3	4	4	4	4	
N_T:	0	0	0	1	2	2	3	4	5	
Gap:	1	2	3	2	1	2	1	0	1	

WHILE-Loop Basics

WHILE-loop is used when a code fragment needs to be executed repeatedly *while* a certain condition is true.

```
while <continuation criterion>
    <code fragment>
end
```

must evaluate to a logical variable (T/F)

- if T: run <code frag>
- if F: go outside of loop

- The number of repetitions is *not* known in advance.
- The continuation criterion is a boolean expression, which is evaluated at the start of the loop.
  - If it is true, the loop body is executed. Then the boolean expression is evaluated again.
  - If it is false, the flow of control is passed to the end of the loop.

Simple WHILE-Loop Examples

```
k = 1; n = 10;
while k <= n
    fprintf('k = %d\n', k)
    k = k+1;
end
```

print

k = 1
k = 2
⋮
k = 10

Q. What would happen if modified to

```
while k <= n
    k = k+1;
    fprint( -- )
end
```

```
k = 1;
while 2^k < 5000
    k = k+1;
end
fprintf('k = %d\n', k)
```

Q. If modified to "while 2^k <= 5000", would it make any difference?

⋮

$k=10$: $2^{10} = 1024 < 5000$ ✓

$k=11$: $2^{11} = 2048 < 5000$ ✓

$k=12$: $2^{12} = 4096 < 5000$ ✓

$k=13$: $2^{13} > 5000$ ✗

k = 13

FOR-Loop to WHILE-Loop

A `for`-loop can be written as a `while`-loop. For example,

FOR

```
s = 0;
for k = 1:4
    s = s + k;
    fprintf('%2d %2d\n', k, s)
end
```

WHILE

```
k = 0; s = 0;
while k < 4
    k = k + 1; s = s + k;
    fprintf('%2d %2d\n', k, s)
end
```

$s = 1 + 2 + 3 + 4 = \sum_{k=1}^{4} k$

- Note that `k` needed to be initialized before the `while`-loop.
- The variable `k` needed to be updated inside the `while`-loop body.

Examples

Up/Down Sequence

(3n+1) problem, Collatz conjecture / Ulam / Kakutani --- / Hailstone ---

Question

Pick a random integer between 1 and 1,000,000. Call the number n and repeat the following process:

- If n is even, replace n by $n/2$.
- If n is odd, replace n by $3n + 1$.

Does it ever take more than 1000 updates to reach 1?

Examples

- 7, 22, 11, 34, 17, ---
- 32, 16, 8, 4, 2, 1

- To generate a random integer between 1 and k, use `randi`, *e.g.*,

```
randi(k)
```

; randi([3, 10]) : rand. int. btw 3 & 10 (inclusive)

- To test whether a number n is even or odd, use `mod`, *e.g.*,

```
mod(n, 2) == 0
```

checks whether n is even or odd
↓ true ↓ false.

Attempt Using FOR-Loop

n initialized before this.

<main frag>

```
for step = 1:1000
    if mod(n,2) == 0
        n = n/2;
    else
        n = 3*n + 1;
    end
    fprintf(' %4d %7d\n', step, n)
end
```

- Note that once n becomes 1, the central process yields the following pattern:

$$1, 4, 2, 1, 4, 2, 1, \ldots$$

- This program continues to run even after n becomes 1.

Solution Using WHILE-Loop

```
step = 0;
while n > 1
    if mod(n,2) == 0
        n = n/2;
    else
        n = 3*n + 1;
    end
    step = step + 1;
    fprintf(' %4d %7d\n', step, n)
end
```

loop header

$n = 1 \longrightarrow (n > 1)$ evaluates to "false"

control is passed to the end of the loop.

- This shuts down when n becomes 1!

Exercise: Gap of 10

Question

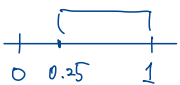
Simulate the tossing of a fair coin until the gap between the number of Heads and that of Tails reaches 10.

biased coin

	1	0
	H	T
Prob	75%	25%

Suggestion

- rand()
- ~~ceil, floor~~
- if-statement




```
if x > 0.25,
    x = 1
else
    x = 0
end
```

Summary

- `For`-loop is a programming construct to execute statements repeatedly.

```
for <loop index values>
    <code fragment>
end
```

- `While`-loop is another construct to repeatedly execute statements. Repetition is controlled by the termination criterion.

```
while <termination criterion is not met>
    <repeat these statements>
end
```