

Revisiting **for** and **while** loops:

For loops are better when you need to do something N times.

While loops are better when you have a test condition for breaking the loop.

```
for index = initial value : increment size : final value
    statements, calculations
end
```

```
while (test condition)
    statements, calculations
end
```

Example:

```
N = 4;
x = 0;
for i = 1 : N
    x = x + 0.5;
    disp([i, x])
end
```

After execution:

```
1    0.5
2     1
3    1.5
4     2
```

A **while** loop can do the same task.

```
i = 1;
N = 4;
x = 0;
while( i <= N )
    x = x + 0.5;
    disp( [i , x] )
end
```

Example:

```
for x = 0 : 0.5 : 2
    disp(x)
end
```

After execution:

```
0
0.5
1
1.5
2
```

A **while** loop can do the same task.

```
x = 0;
while( x <= 2 )
    disp(x)
    x = x + dx;
end
```

Example: Determine $y(x) = x^2$ from $x=1$ to 2 in steps of 0.2.

In the past we would do the following,

```
x = [1 : 0.2 : 2];
y = x.^2;
table = [x ; y];
disp(table')
```

Let's do the same task but with a **for** loop.

```
for x = 1 : 0.2 : 2
    y = x^2
    disp([x y])
end
```

It gives the same display but doesn't store the values of x and y for later use.

```
i = 1;
for x = 1 : 0.2 : 2
    y = x^2;
    xval(i) = x;
    yval(i) = y;
    i = i + 1;
end
table = [xval ; yval];
disp(table')
```

A **while** loop can do the same task.

```
i = 1;
x = 1;
dx = 0.2;
while(x >= 2)
    y = x^2;
    xval(i) = x;
    yval(i) = y;
    x = x + 0.2;
    i = i + 1;
end
table = [xval ; yval];
disp(table')
```

Example: There is a sequence of numbers,
1, 2, 4, 8, 16, 32,...

Calculate the first N numbers of this sequence, store them in memory, and display them to the screen. Notice that the sequence is as follows,

$$\text{number}(i) = \text{number}(i-1) * 2$$

```
number(1) = 1
for i=2:N
    number(i) = number(i-1)*2
end
disp(number')
```

Beware of infinite loops!

```
x = 0;
i = 1;
while( x <= 2 )
    i = i + 1;
end
```

This loop will never end. Type ctrl+C to end the program.