

# Logical Operators

- Assume  $x = 0$ .
- How about  $1 < x < 3$ ? (Surprising!)

Logical operator	Name	Description
& Example: $A \& B$	AND	Operates on two operands ( $A$ and $B$ ). If both are true, the result is true (1), otherwise the result is false (0).
 Example: $A   B$	OR	Operates on two operands ( $A$ and $B$ ). If either one, or both are true, the result is true (1), otherwise (both are false) the result is false (0).
~ Example: $\sim A$	NOT	Operates on one operand ( $A$ ). Gives the opposite of the operand. True (1) if the operand is false, and false (0) if the operand is true.

# Truth Table<sup>1</sup>

- Let  $A$  and  $B$  be two logical variables.
- Then you can find the truth table for logical operators as follows:

A	B	$\sim A$	A&B	A   B
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

---

<sup>1</sup>Note that the basic instructions, such as the plus operator, are implemented by **logic gates**. See any textbook for digital circuit design.

## Exercise: & vs. ==<sup>2</sup>

```
1 >> u = [0 2 0 4];
2 >> v = [0 0 3 4];
3 >> u == v
4
5 ans =
6
7     1     0     0     1
8
9 >> u & v
10
11 ans =
12
13     0     0     0     1
```

<sup>2</sup>Thanks to a lively class discussion (MATLAB-237) on April 16, 2014.

## Precedence of Operators<sup>3</sup>

Operators	Precedence
parentheses: ( )	Highest
transpose and power: ' , ^ , . ^	
unary: negation ( - ) , not ( ~ )	
multiplication, division * , / , \ , . * , . / , . \	
addition, subtraction + , -	
relational < , <= , > , >= , == , ~=	
element-wise and &	
element-wise or	
and && (scalars)	
or    (scalars)	
assignment =	Lowest

<sup>3</sup>See Table 1.2 Operator Precedence Rules in Attaway, p. 25.

```
1 >> Lecture 2
2 >>
3 >>           -- Programming Basics
4 >>
```

*“If debugging is the process of **removing** software bugs, then programming must be the process of **putting** them in.”*

– Edsger W. Dijkstra (1930–2002)

# Flow Controls

- We wish the computers could **make decision** on their own.
- Also, the computers should **repeat** actions for a specified number of times or until the stopping condition is satisfied.
  - As known as **loops**.
- These two features facilitate the usefulness of computers.
  - Think about the max algorithm.

# Building Blocks

- **Sequential operations:** be executed **in order**.
- **Selections:** check which condition is satisfied and then execute the actions accordingly.
- **Repetitions:** repeat some instructions and stop while the termination condition is satisfied.



# Selections

- We start with `if` followed by a logical expression.
- If true, then do the corresponding statements; otherwise, leave the structure.
- You can also use `else` to specify the actions if the condition is false.
- For both cases, you need the `end` statement to finish the selection.

## Example: Circle Area

- Write a program which takes a number as input.
  - We use the function **input** which takes a number from the keyboard.
- If the input is positive, then output the resulting circle area.

```
1 clear; clc;
2
3 r = input("Enter r? ");
4 if r > 0
5     A = pi * r ^ 2;
6     disp("The circle area is " + A + ".");
7 else
8     disp(num2str(r) + " is negative.");
9 end
```

## Example: Nested Conditional Statements

```
1 clear; clc;
2
3 s = input("Enter r? ", "s");
4 r = str2num(s);
5 if isempty(r)
6     disp(s + " is not a number.");
7 else
8     if r > 0
9         A = pi * r ^ 2;
10        disp("The circle area is " + A + ".");
11    else
12        disp(s + " is negative.");
13    end
14 end
```

- Use **str2num** to convert from a string to a number.
- Use **isempty** to check if the variable is null.

## Example: if-elseif-else

```
1 clear; clc;
2
3 s = input("Enter r? ", "s");
4 r = str2num(s);
5 if isempty(r)
6     disp(s + " is not a number.");
7 elseif r >= 0
8     A = pi * r ^ 2;
9     disp("The circle area is " + A + ".");
10 else
11     disp(s + " is negative.");
12 end
```

- More clear!

## Exercise

- Write a program which converts centesimal points to GPA.
- Let  $x$  be the grade as input.
- For simplicity,
  - If  $90 \leq x \leq 100$ , then  $x$  is converted to 4.
  - If  $80 \leq x < 90$ , then 3.
  - If  $70 \leq x < 80$ , then 2.
  - If  $60 \leq x < 70$ , then 1.
  - If  $x < 60$ , then 0.

```
1 clear; clc;
2 x = input("Enter your score? ");
3 if 90 <= x && x <= 100
4     disp("4");
5 elseif 80 <= x && x < 90
6     disp("3");
7 elseif 70 <= x && x < 80
8     disp("2");
9 elseif 60 <= x && x < 70
10    disp("1");
11 else
12    disp("0");
13 end
```

- Note that we use && to join two criterion in Line 3.

## Short-Circuit Evaluation: `&&` and `||`

- Let A and B be two logical results.
- Consider A `&&` B.
- If A returns false, then B won't be evaluated.
- This facilitates time-saving.
- The case of A `||` B is similar.
- We need to guarantee that the condition is a scalar.

## Another Selection Structure: **switch-case**


```
1 clear; clc;
2
3 city = input("Enter a city name: ", "s");
4 switch city
5     case {"Taipei", "New Taipei"}
6         disp("Price: $100");
7     case "Taichung"
8         disp("Price: $200");
9     case "Tainan"
10        disp("Price: $300");
11    otherwise
12        disp("Not an option.");
13 end
```



## Equivalence between if and switch<sup>4</sup>

```
1 clear; clc;
2
3 city = input("Enter the city name: ", "s");
4 if city == "Taipei" || city == "New Taipei"
5     disp("Price: $100.");
6 elseif city == "Taichung"
7     disp("Price: $200.");
8 elseif city == "Tainan"
9     disp("Price: $300.");
10 else
11     disp("Not an option.");
12 end
```

---

<sup>4</sup>Thanks to a lively class discussion (MATLAB-244) on August 20, 2014. 

## Quantifiers<sup>5</sup>

- The function **all** determines if **all** elements are true.
- The function **any** determines if there is **any** true element in the array.

```
1 >> scores = [50 60 70];
2 >> all(scores >= 60)
3 ans =
4
5     0
6
7 >> any(scores >= 60)
8 ans =
9
10    1
```

---

<sup>5</sup>See [https://en.wikipedia.org/wiki/Quantifier\\_\(logic\)](https://en.wikipedia.org/wiki/Quantifier_(logic)).

## More Logical Functions

### Logical function

<code>ischar(A)</code>	Returns a 1 if <b>A</b> is a character array and 0 otherwise.
<code>isempty(A)</code>	Returns a 1 if <b>A</b> is an empty matrix and 0 otherwise.
<code>isinf(A)</code>	Returns an array of the same dimension as <b>A</b> , with 1s where <b>A</b> has 'inf' and 0s elsewhere.
<code>isnan(A)</code>	Returns an array of the same dimension as <b>A</b> with 1s where <b>A</b> has 'NaN' and 0s elsewhere. ('NaN' stands for "not a number," which means an undefined result.)
<code>isnumeric(A)</code>	Returns a 1 if <b>A</b> is a numeric array and 0 otherwise.
<code>isreal(A)</code>	Returns a 1 if <b>A</b> has no elements with imaginary parts and 0 otherwise.

- NaN: *Not A Number*, caused by  $\frac{\infty}{\infty}$  and  $\infty - \infty$ .<sup>6</sup>

---

<sup>6</sup>See [NaN](#).

*"Logic is the anatomy of thought."*

– John Locke (1632–1704)

*"This sentence is false."*

– anonymous

*"I know that I know nothing."*

– Plato

(In Apology, Plato relates that Socrates accounts for his seeming wiser than any other person because he does not imagine that he knows what he does not know.)

# Repetitions

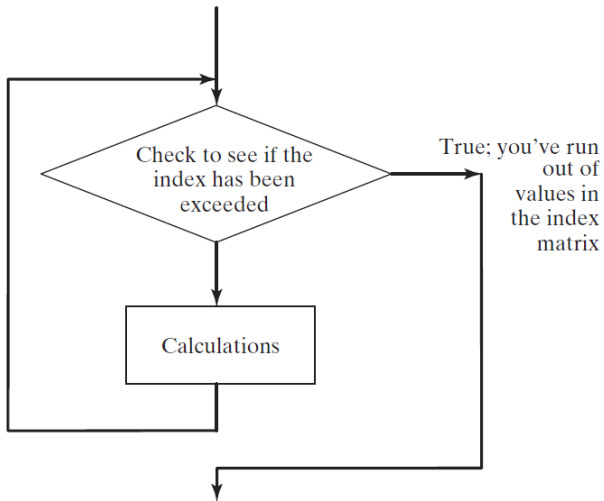
- If some instructions are **potentially** repeated, you should wrap those in a **loop**.
- All loops can be done in the following three parts:
  - find the **repeated pattern** for each iteration;
  - wrap them by a proper loop;
  - set the continuation condition by defining a **loop variable** with some **criterion**.
- MATLAB has two types of loops: **for** loops and **while** loops.
  - Use **for** loops if you know the number of iterations.
  - Otherwise, use **while** loops.

## for Loops

- A **for** loop is the easiest choice when you know how many times you need to repeat the loop.

```
1 for loopVar = someArray
2     % body
3 end
```

- Particularly, we often use **for** loops to manipulate arrays (data)!



## Examples

- Print 1 to 10.

```
1 for i = 1 : 10
2     disp(i);
3 end
```

- How to show the odd integers from 1 to 9?

```
1 stock_list = ["tsmc", "apl", "goog"];
2 for stock = stock_list
3     disp(stock);
4 end
```

- Clearly, MATLAB has **for-each** loops, which is an enhanced one compared to the naive one in C.



## Example: Find Maximum (Revisited)

```
1 clear; clc;
2
3 data = [4 9 7 2 -1 6 3];
4 result = data(1);
5 for item = data(2 : end)
6     if result < item
7         result = item;
8     end
9 end
10 result
```

- Use **max** in your future work.<sup>7</sup>
- Can you find the location of the maximum element?
- Try to find the minimum element and its location.

---

<sup>7</sup>Don't repeat yourself.

## Exercise: Where is Maximum?

- Write a program which indicates where the maximum is.

```
1 clear; clc;
2
3 data = [4 9 7 2 -1 6 3];
4 loc = 1;
5 for i = 2 : length(data)
6     if data(i) > data(loc)
7         loc = i;
8     end
9 end
10 loc
```

- Note that **max** could return the index of maximum as the second output.

## Example: Running Sum

- Write a program which calculates the sum of data.
- Use **randi** to generate a random integer array as testing data.

```
1 clear; clc;
2
3 n = 5;
4 data = randi(100, 1, n)
5
6 sum = 0;
7 for i = 1 : n
8     sum = sum + data(i); % running sum
9 end
10 sum
```

- Of course, you could use **sum** for the same functionality.

## Digression: Programming feat. Math

- To sum the sequence  $1, 2, \dots, n$ , we could write

$$\text{sum} = 1 + 2 + \dots + n = \sum_{i=1}^n i.$$

- Recall that you write down a loop to add  $i$  from 1 to  $n$  one by one to an accumulator, say `sum`.
- See? **A summation is realized by a loop!**
- From now, you know how to program when you meet a formula like above.

## Numerical Example: Monte Carlo Simulation

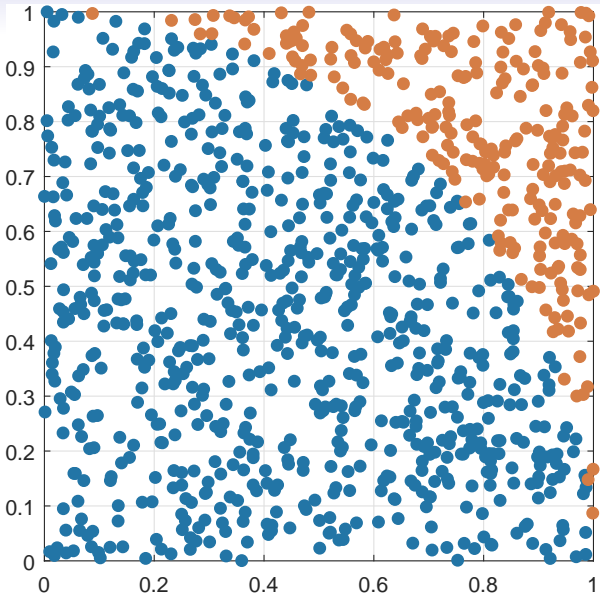
- Let  $m$  be the number of sample points falling in the region of the quarter circle shown in the next page,  $n$  be the total number of sample points.
  - Use **rand** to generate a value between 0 and 1 (exclusive).
- Write a program which estimates  $\pi$  by

$$\hat{\pi} = 4 \times \frac{m}{n}.$$

- Note that  $\hat{\pi} \rightarrow \pi$  as  $n \rightarrow \infty$  by the law of large numbers (LLN).<sup>8</sup>

---

<sup>8</sup>See [https://en.wikipedia.org/wiki/Law\\_of\\_large\\_numbers](https://en.wikipedia.org/wiki/Law_of_large_numbers).



```
1 clear; clc;
2
3 n = 1e5;
4 m = 0;
5
6 for i = 1 : n
7
8     x = rand(1);
9     y = rand(1);
10
11     if x ^ 2 + y ^ 2 < 1
12         m = m + 1;
13     end
14
15 end
16 result = 4 * m / n
```

- Try to vectorize this program.

## Exercise: Vectorization of MC Simulation for $\pi$

```
1 clear; clc;
2
3 n = 1e5;
4 x = rand(n, 1);
5 y = rand(n, 1);
6 m = sum(x.^2 + y.^2 < 1);
7 result = 4 * m / n
```

- More clear and faster!!!

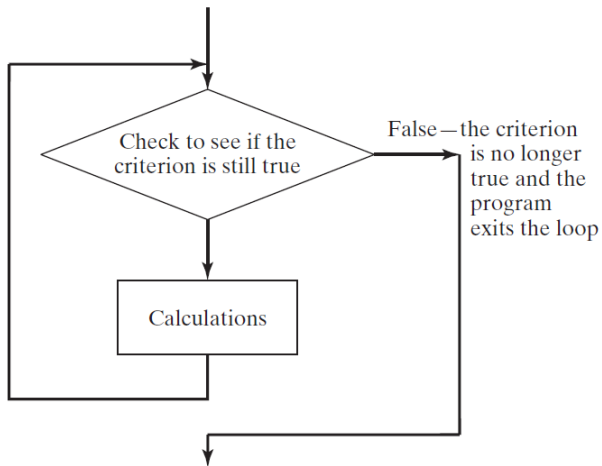


## while Loops

- The `while` loops are preferred when you need to keep repeating the instructions **until a continuation criterion is not satisfied**.

```
1 while criterion
2     % body
3 end
```

- Be aware that `if` statement executes only once; you should use `while` statement if you want to repeat some actions.



# Example: Compounding

- Let  $balance$  be the initial amount of some investment, and  $r$  be the annualized return rate.
- Write a program which calculates the holding years when this investment doubles its value.

## Solution

- In this case, we don't know how many iterations we need before the loop.

```
1 clear; clc;
2
3 balance = 100;
4 r = 0.01;
5 goal = 200;
6
7 holding_years = 0;
8 while balance < goal
9     balance = balance * (1 + r);
10    holding_years = holding_years + 1;
11 end
12 holding_years
```

- Note that the criterion is evaluated to continue the loop.

# Infinite Loops

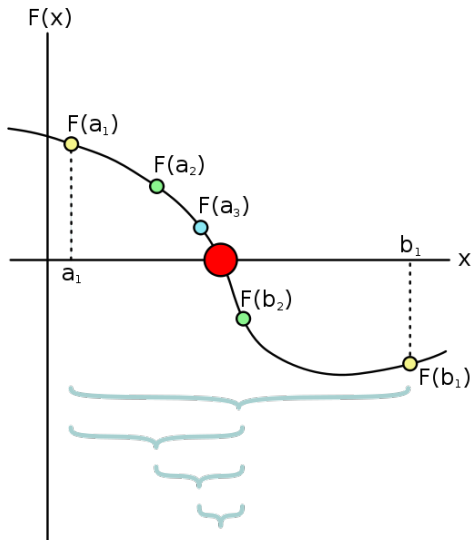
```
1 while true
2     disp("Press ctrl+c to stop me!!!");
3 end
```

- Note that your program can terminate the program by pressing **ctrl+c**.

## More Exercises (Optional)

- Let  $a > b$  be two any positive integers.
- Write a program which calculates the remainder of  $a$  divided by  $b$ .
  - Do not use **mod**( $a, b$ ).
- Write a program which determines the greatest common divisor (GCD) of  $a$  and  $b$ .
  - Do not use **gcd**( $a, b$ ).

# Numerical Example: Bisection Method for Root-Finding



# Problem Formulation

---

## Input

- Target function  $f(x) = x^3 - x - 2$ .
- Initial search interval  $[a, b] = [1, 2]$ .
- Error tolerance  $\epsilon = 1e - 9$ .

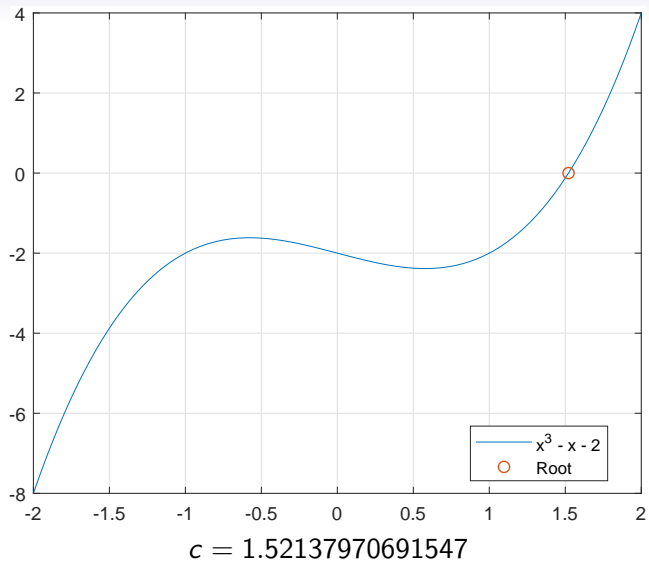
## Output

- The approximate root  $\hat{r}$ .
-



## Solution

```
1 clear; clc;
2
3 a = 1; b = 2; eps = 1e-9;
4
5 while b - a > eps
6
7     c = (a + b) / 2;
8     fa = a * a * a - a - 2;
9     fc = c * c * c - c - 2;
10
11     if fa * fc < 0
12         b = c;
13     else
14         a = c;
15     end
16
17 end
18 root = c
19 residual = fc
```



*“All science is dominated by the idea of **approximation**.”*

– [Bertrand Russell](#) (1872–1970)

# Jump Statements

- A **break** statement terminates a **for** or **while** loop immediately.
  - Aka **early termination**.
- A **continue** statement skips instructions behind it and start the next iteration.
  - Directly jump to the very beginning of the loop; still in the loop.
- Notice that the **break** and **continue** statements must be conditional.

## Example: Primality Test<sup>9</sup>

- Let  $x$  be any positive integer larger than 2 as input.
- Then  $x$  is a **prime** number if  $\forall y \in \{2, 3, \dots, x - 1\}$ ,  $y$  is not a divisor of  $x$ , denoted by  $y \nmid x$ .
- In other words,  $x$  is called a **composite** number if  $\exists y \in \{2, 3, \dots, x - 1\}$ ,  $y \mid x$ .
- Now write a program which determines if  $x$  is a prime number.

---

<sup>9</sup>Also see Manindra Agrawal, Neeraj Kayal, Nitin Saxena (2002).

```
1 clear; clc;
2
3 x = input('Enter x > 2? ');
4 isPrime = true; % a flag, true if the number is prime
5 for y = 2 : sqrt(x)
6     if mod(x, y) == 0
7         isPrime = false;
8         break;
9     end
10 end
11
12 if isPrime
13     disp([num2str(x) ' is a prime number.']);
14 else
15     disp([num2str(x) ' is a composite number.']);
16 end
```

## Equivalence: for and while Loops

- Whatever you can do with a **for** loop can be done with a **while** loop, and vice versa.

```
1 clear; clc;
2
3 balance = 100; goal = 200; r = 0.01;
4
5 for years = 1 : inf % inf: a huge but finite integer
6     balance = balance * (1 + r);
7     if balance >= goal
8         break;
9     end
10 end
11 years
```

- For another example,

```
1 clear; clc;
2
3 x = input("Enter x > 2? ");
4
5 isPrime = true; y = 2;
6 while isPrime && y < x
7     isPrime = mod(x, y);
8     y = y + 1;
9 end
10
11 if isPrime
12     disp(num2str(x) + " is a prime number.");
13 else
14     disp(num2str(x) + " is a composite number.");
15 end
```



# Nested Loops

- Write a program which outputs the following patterns:

```
*           *****           *           *****
**          ****           **          ****
***         ***           ***         ***
****        **           ****        **
*****      *           *****      *
```

(a)

(b)

(c)

(d)

- You may use `fprintf("*")` and `fprintf("\n")` to print a single star and break a new line, respectively.

```
1 clear; clc;
2
3 % case (a)
4 for i = 1 : 5
5     for j = 1 : i
6         fprintf("*");
7     end
8     fprintf("\n");
9 end
```