```
>> Lecture 2
>>
>>                -- Programming Basics
>>
```

"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.

```
clear; clc;

r = input("Enter r? ");
if r > 0
    A = pi * r ^ 2;
    disp("The circle area is " + A + "."]);
else
    disp(num2str(r) + " is negative.");
end
```

Example: Nested Conditional Statements

```
clear; clc;

s = input("Enter r? ", "s");
r = str2num(s);
if isempty(r)
    disp(s + " is not a number.");
else
    if r > 0
        A = pi * r ^ 2;
        disp("The circle area is " + A + ".");
    else
        disp(s + " is negative.");
    end
end
```

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

Example: **if-elseif-else**

```
clear; clc;

s = input("Enter r? ", "s");
r = str2num(s);
if isempty(r)
    disp(s + " is not a number.");
elseif r >= 0
    A = pi * r ^ 2;
    disp("The circle area is " + A + ".");
else
    disp(s + " is negative.");
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**

```
clear; clc;

city = input("Enter a city name: ", "s");
switch city
    case {"Taipei", "New Taipei"}
        disp("Price: $100");
    case "Taichung"
        disp("Price: $200");
    case "Tainan"
        disp("Price: $300");
    otherwise
        disp("Not an option.");
end
```

Equivalence between if and switch[1]

```
clear; clc;

city = input("Enter the city name: ", "s");
if city == "Taipei" || city == "New Taipei"
    disp("Price: $100.");
elseif city == "Taichung"
    disp("Price: $200.");
elseif city == "Tainan"
    disp("Price: $300.");
else
    disp("Not an option.");
end
```

[1] Thanks to a lively class discussion (MATLAB-244) on August 20, 2014.

Quantifiers[2]

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

```
>> scores = [50 60 70];
>> all(scores >= 60)
ans =

     0

>> any(scores >= 60)
ans =

     1
```

[2] See https://en.wikipedia.org/wiki/Quantifier_(logic).

More Logical Functions

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

- NaN: *N*ot *A* *N*umber, caused by $\frac{\infty}{\infty}$ and $\infty - \infty$.[3]

[3] 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

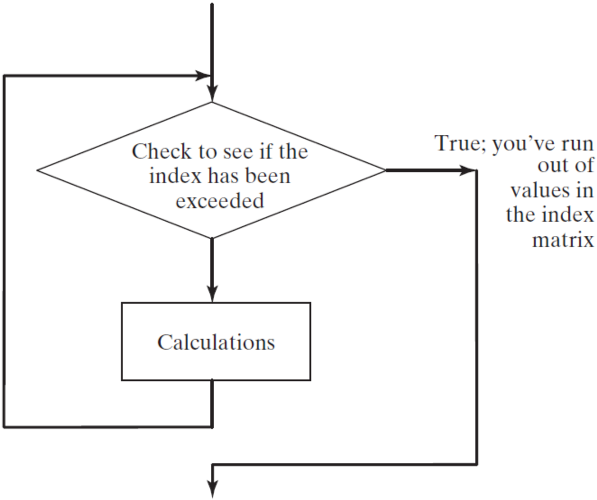
- If some instructions are potentially repeated, you should wrap those in a loop.
- All loops can be done in the following three parts:
  - find the repeated pattern for each iteration;
  - warp them by a proper loop;
  - set the continuation condition by defining a loop variable with some criterion.
- 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.

```
for loopVar = someArray
    % body
end
```

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


Check to see if the index has been exceeded
True; you've run out of values in the index matrix
Calculations

Examples

- Print 1 to 10.

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

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

```
stock_list = ["tsmc", "aapl", "goog"];
for stock = stock_list
    disp(stock);
end
```

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

Example: Find Maximum (Revisited)

```
clear; clc;

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

- Use **max** in your future work.[4]
- Can you find the location of the maximum element?
- Try to find the minimum element and its location.

[4] Don't repeat yourself.

Exercise: Where is Maximum?

- Write a program which indicates where the maximum is.

```
clear; clc;

data = [4 9 7 2 -1 6 3];
loc = 1;
for i = 2 : length(data)
    if data(i) > data(loc)
        loc = i;
    end
end
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.

```
clear; clc;

n = 5;
data = randi(100, 1, n)

sum = 0;
for i = 1 : n
    sum = sum + data(i); % running sum
end
sum
```

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

Digression: Programming feat. Math

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

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

- 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.

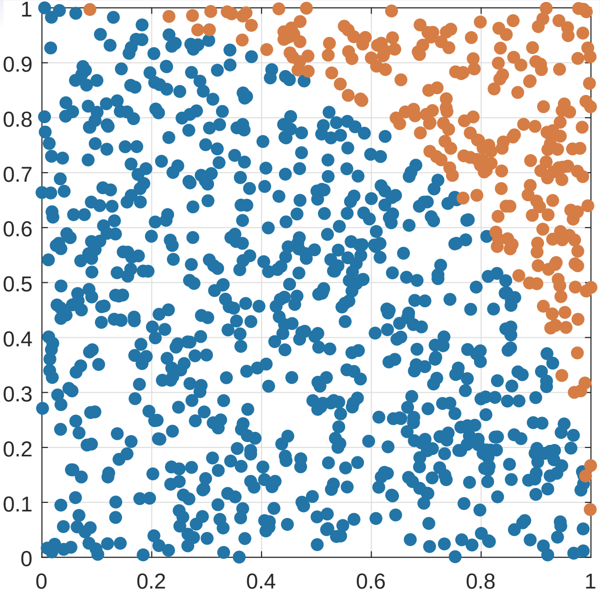
Numerical Example: Monte Carlo Simulation

- 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 π by

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

- Note that $\hat{\pi} \to \pi$ as $n \to \infty$ by the law of large numbers (LLN).[5]

[5] See https://en.wikipedia.org/wiki/Law_of_large_numbers.


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1

```
clear; clc;

n = 1e5;
m = 0;

for i = 1 : n

    x = rand(1);
    y = rand(1);

    if x ^ 2 + y ^ 2 < 1
        m = m + 1;
    end

end
result = 4 * m / n
```

- Try to vectorize this program.

Exercise: Vectorization of MC Simulation for π

```
clear; clc;

n = 1e5;
x = rand(n, 1);
y = rand(n, 1);
m = sum(x .^ 2 + y .^ 2 < 1);
result = 4 * m / n
```

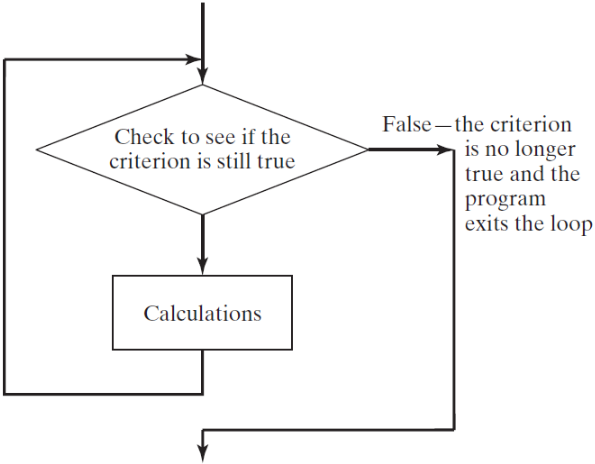
- More clear and faster!!!

while Loops

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

```
while criterion
    % body
end
```

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


Check to see if the criterion is still true
False—the criterion is no longer true and the program exits the loop
Calculations

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 it value.

Solution

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

```
clear; clc;

balance = 100;
r = 0.01;
goal = 200;

holding_years = 0;
while balance < goal
    balance = balance * (1 + r);
    holding_years = holding_years + 1;
end
holding_years
```

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

Infinite Loops

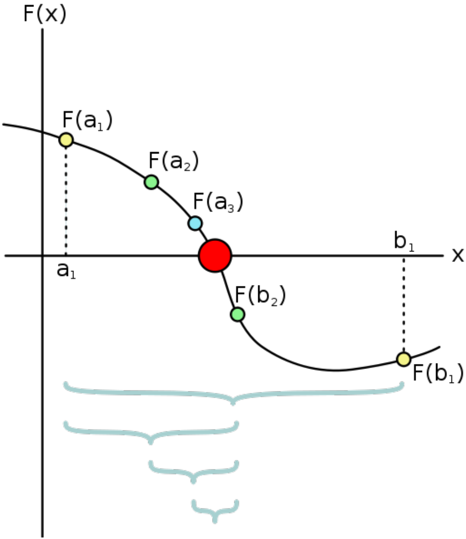
```
while true
    disp("Press ctrl+c to stop me!!!");
end
```

- 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

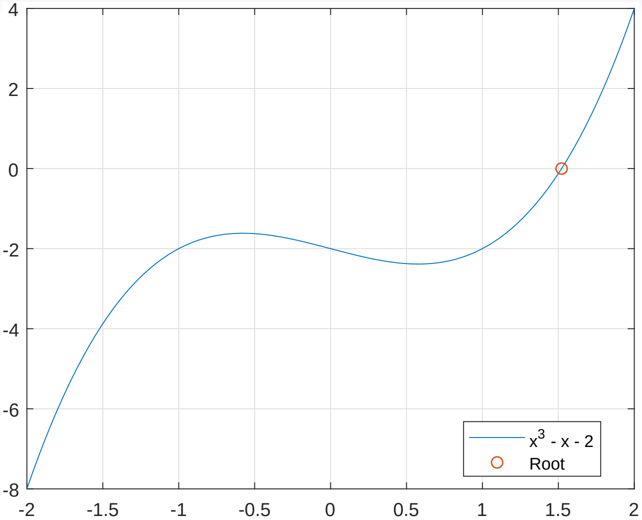
- Target function $f(x) = x^3 - x - 2$.
- Initial search interval $[a, b] = [1, 2]$.
- Error tolerance $\epsilon = 1e - 9$.

Output

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

Solution

```
clear; clc;

a = 1; b = 2; eps = 1e-9;

while b - a > eps

    c = (a + b) / 2;
    fa = a * a * a - a - 2;
    fc = c * c * c - c - 2;

    if fa * fc < 0
        b = c;
    else
        a = c;
    end

end
root = c
residual = fc
```



$$c = 1.52137970691547$$

"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[6]

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

[6] Also see Manindra Agrawal, Neeraj Kayal, Nitin Saxena (2002).

```
clear; clc;

x = input('Enter x > 2? ');
isPrime = true; % a flag, true if the number is prime
for y = 2 : sqrt(x)
    if mod(x, y) == 0
        isPrime = false;
        break;
    end
end

if isPrime
    disp([num2str(x) ' is a prime number.']);
else
    disp([num2str(x) ' is a composite number.']);
end
```

Equivalence: for and while Loops

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

```
clear; clc;

balance = 100; goal = 200; r = 0.01;

for years = 1 : inf % inf: a huge but finite integer
    balance = balance * (1 + r);
    if balance >= goal
        break;
    end
end
years
```

- For another example,

```
clear; clc;

x = input("Enter x > 2? ");

isPrime = true; y = 2;
while isPrime && y < x
    isPrime = mod(x, y);
    y = y + 1;
end

if isPrime
    disp(num2str(x) + " is a prime number.");
else
    disp(num2str(x) + " is a composite number.");
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.

```
clear; clc;

% case (a)
for i = 1 : 5
    for j = 1 : i
        fprintf("*");
    end
    fprintf("\n");
end
```