به نام خدا

آموزش متلب

مدرس:

دکتر فریده قریشی، زهرا مهدور

راه‌های ارتباطی:

Email : zahra.mahdevar@yahoo.com

Telegram : Z_Mhdr

جلسه چهارم: کلیدواژه‌ها، بردارسازی و مدیریت اجرای حلقه‌ها

مقدمه

در این جلسه، ابتدا با برخی از کلمات کلیدی آشنا می‌شویم که در زبان متلب به‌صورت پیش‌فرض تعریف شده‌اند و به همین دلیل نمی‌توان از آن‌ها برای نام‌گذاری متغیرها استفاده کرد. در ادامه، روش‌های مختلف ساخت بردارهای عددی یک‌بعدی را بررسی می‌کنیم و با دستورات مربوط به تولید بازه‌های عددی، بازه‌های هم‌فاصله و بازه‌های لگاریتمی آشنا می‌شویم. سپس نحوه‌ی نوشتن توضیحات درون کد (کامنت‌گذاری) برای مستندسازی برنامه را فرا می‌گیریم و در پایان، تفاوت دو دستور پرکاربرد "توقف" و "ادامه" در حلقه‌ها را مورد بررسی قرار می‌دهیم

بخش اول: بررسی کلمات کلیدی در متلب

در متلب، کلمات کلیدی کلماتی هستند که توسط زبان متلب رزرو شده‌اند و نمی‌توان از آن‌ها به‌عنوان نام متغیر، تابع یا فایل استفاده کرد. برای بررسی اینکه یک کلمه خاص کلمه کلیدی رزرو شده است یا نه، از تابع زیر استفاده می‌کنیم

```
iskeyword('word')
```

```
ans = logical

0
```

مثال

```
iskeyword('for')
```

```
ans = logical

1
```

دریافت لیست کامل کلمات کلیدی

```
iskeyword
```

```
'break'

'case'
```

```
'catch'
'classdef'
'continue'
'else'
'elseif'
'end'
'for'
'function'
'global'
'if'
'otherwise'
'parfor'
'persistent'
'return'
'spmd'
'switch'
'try'
'while'
```

تـوجـه : استفاده از کلمات رزرو شده به‌عنوان نام متغیر باعث بروز خطا یا رفتارهای غیرمنتظره در کد می‌شود

```
% if = 1
```

Incorrect use of '=' operator. Assign a value to a variable using '=' and compare values for equality using '=='.

```
% case = 1
```

Illegal use of reserved keyword "case".

بخش دوم: ساخت بردارهای عددی

در متلب، بردارهای عددی یک‌بعدی (چه سطری و چه ستونی) یکی از مهم‌ترین ساختارهای داده‌ای محسوب می‌شوند که برای ذخیره‌سازی مجموعه‌ای از اعداد پشت سر هم به کار می‌روند. ساخت این بردارها به روش‌های مختلفی ممکن است که در ادامه مهم‌ترین آن‌ها را همراه با مثال بررسی می‌کنیم

1. تعریف دستی بردارها

ساده‌ترین روش تعریف بردار، وارد کردن عناصر به‌صورت دستی است

```
v1 = [1 2 3 4 5]       % بـردار سطری
v2 = [1; 2; 3; 4; 5]   % بـردار ستـونـی
```

```
v1 = 1×5

1     2     3     4     5

v2 = 5×1

1

2

3

4

5
```

در متلب، فاصله یا ویرگول , عناصر را به‌صورت سطری در کنار هم قرار می‌دهد، ولی برای ایجاد بردار ستونی باید از ; بین عناصر استفاده کنیم

1. : (colon)

ایجاد بازه‌های هم‌فاصله در یک بازه مشخص

```
% start:step:end
```

اگر طول گام ذکر نشود، مقدار پیش‌فرض آن برابر با ۱ است

```
v3 = 1:5
v4 = 0:0.5:2
v5 = 5:-1:1
```

```
v3 = 1×5

1     2     3     4     5

v4 = 1×5

0    0.5000    1.0000    1.5000    2.0000

v5 = 1×5

5     4     3     2     1
```

2. linspace

این تابع برای ساخت بردارهایی با تعداد مشخصی عنصر بین دو مقدار ابتدایی و انتهایی به کار می‌رود

```
% linspace(a, b, n)
```

ورودی‌های این تابع به ترتیب مقدار ابتدایی، مقدار انتهایی و تعداد عناصر بردار است

```
v6 = linspace(0, 1, 5)
```

```
v6 = 1×5

0    0.2500    0.5000    0.7500    1.0000
```

3. logspace

برای تولید بردارهایی که فاصلهٔ لگاریتمی بین عناصر دارند، از این تابع استفاده می‌کنیم

```
% logspace(a, b, n)
```

دو ورودی اول آن توان‌های 10 هستند و سومی تعداد نقاط است

```
v7 = logspace(1, 3, 3)
```

```
v7 = 1×3

10         100        1000
```

نکته: اگر بخواهیم یک بردار ستونی از بردار سطری بسازیم، می‌توانیم از ' (ترانهاده) استفاده کنیم:

```
v1 = [1 2 3 4 5];
v_column = v1'
```

```
v_column = 5×1

1

2

3

4

5
```

این روش‌ها پایه‌ای‌ترین ابزارهای تولید لیست‌های عددی در متلب هستند و در مسائل عددی، ترسیم نمودار، حل معادلات و بسیاری از کاربردهای دیگر بسیار مهم‌اند

Practice 1: Oscillating Pattern with Combined Steps

Create a vector that follows the pattern below:

[1 2 3 2 1 2 3 2 1]

Without typing the elements manually—use functions and loops!

Practice 2: Creating a Vector Conditionally

Create a vector that contains only the numbers between 1 and 100 that are divisible by both 3 and 5.

(*Without using a loop!*)

بخش سوم: کامنت‌گذاری و بلوک‌های کامنتی

کامنت متنی است که در کد نوشته می‌شود تا برای برنامه‌نویس یا دیگران توضیح بدهد که کد چه کاری انجام می‌دهد، بدون اینکه در اجرای برنامه تأثیری داشته باشد

⟵ برای نوشتن یک کامنت تک‌خطی، از علامت % در ابتدای خط استفاده می‌کنیم

⟵ برای نوشتن توضیحات چندخطی (مثل مقدمه‌ی یک تابع یا توضیح الگوریتم) می‌توانیم از ابزار بلوک کامنتی استفاده کنیم

◁ استفاده از % پشت سرهم

```
% 25.1.1404
% Zahra.Mahdevar
% Lecture 4
```

نکته: فقط همان خطی که بعد از % نوشته می‌شود کامنت می‌شود

◁ انتخاب چند خط و فشردن دکمه های میان بر در نوار ابزار منوی ویرایشگر

```
% 25.1.1404
% Zahra.Mahdevar
% Lecture 4
```

◁ استفاده از علامت % و آکولاد با هم

```
%{
25.1.1404
Zahra.Mahdevar
Lecture 4
%}
```

نکته بسیار مهم: کامنت چندخطی حتماً باید با %} بسته شود. اگر فراموش شود، متلب ادامه کد را نیز به‌عنوان کامنت در نظر می‌گیرد و اجرا نمی‌کند

بخش چهارم: کنترل اجرای حلقه ها

break --> اجرای حلقه را به‌طور کامل متوقف می‌کند

ساختار کلی

```
% for i = 1:N
%     if شرط_خاص
%         break;
%     end
%     % سایر عملیات
% end
```

مثال ۱: یافتن اولین عددی که بر 7 بخش‌پذیر باشد

```
nums = [13, 22, 15, 43, 28, 33, 14];
for i = 1:length(nums)
    if mod(nums(i), 7) == 0
        fprintf('The first number divisible by 7: %d\n', nums(i))
        break;
    end
end
```

```
The first number divisible by 7: 28
```

مثال: حل یک معادله به روش نیوتن رافسون

```
% تعریف تابع و مشتق آن
f  = @(x) x - cos(x);
df = @(x) 1 + sin(x);
root = 0.7390851332;
y_root = f(root);

% تنظیمات اولیه
x0 = 0.5;              % حدس اولیه
tol = 1e-4;            % تلورانس
maxIter = 40;          % حداکثر تعداد تکرار
x_vals = x0;           % ذخیره مسیر تکرارها

for i = 1:maxIter
    fx = f(x0);
    dfx = df(x0);

    if dfx == 0
        error('Derivative became zero. The algorithm has stopped.');
    end

    x1 = x0 - fx / dfx;
    x_vals(end+1) = x1;

    fprintf('Step %d: x = %.10f\n', i, x1);

    if abs(x1 - x0) < tol
        fprintf('\nThe root is approximately: %.10f\n', x1);
        break;
    end

    x0 = x1;
end

% رسم نمودار تابع
x = linspace(0, 2, 400);
y = f(x);

figure;
plot(x, y, 'b-', 'LineWidth', 2); hold on;
plot(x_vals, f(x_vals), 'yo-', 'LineWidth', 2); hold on;
plot(root, y_root, 'r*', 'MarkerSize', 16, 'MarkerFaceColor', 'r')
yline(0, '--k');
xlabel('x');
ylabel('f(x)');
xlim([0.4 0.8]);
ylim([-0.6 0.2]);
```

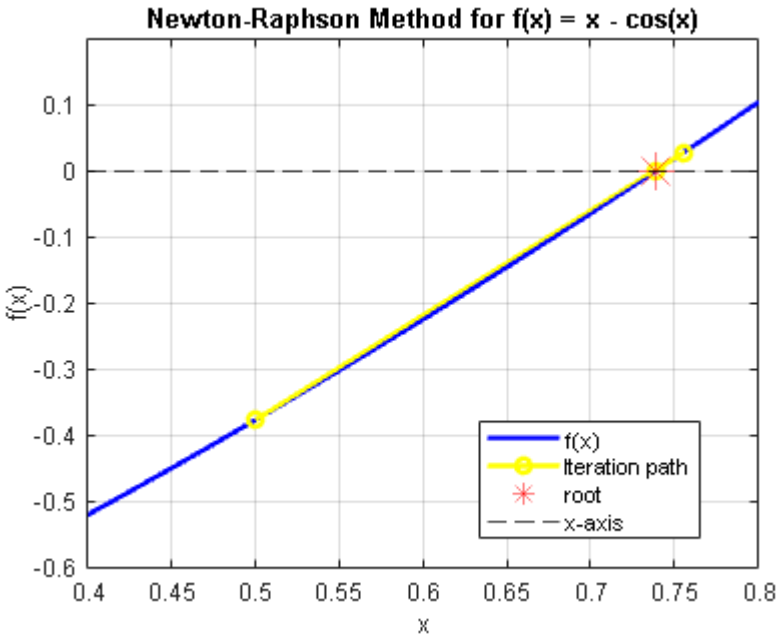
```
% علامتگذاری ریشه با دایره قرمز
title('Newton-Raphson Method for f(x) = x - cos(x)');
legend('f(x)', 'Iteration path', 'root', 'x-axis', 'Location', 'best');
grid on;
```

```
Step 1: x = 0.7552224171

Step 2: x = 0.7391416661

Step 3: x = 0.7390851339

The root is approximately: 0.7390851339
```



continue **در حلقه اجرای تکرار جاری را رد می‌کند و به تکرار بعدی می‌رود <--**

ساختار کلی

```
% for i = 1:N
%     if شرط_خاص
%         continue;
%     end
%     % سایر عملیات
% end
```

مثال: چاپ فقط اعداد فرد از یک آرایه

```
A = 1:10;
for i = 1:length(A)
    if mod(A(i), 2) == 0
        continue; % اگر زوج است، از این تکرار بگذر
    end
    fprintf('Odd number: %d\n', A(i))
end
```

```
Odd number: 1

Odd number: 3

Odd number: 5

Odd number: 7

Odd number: 9
```

مثال: برنامه‌ای بنویسید که یک آرایه عددی شامل مقادیر معتبر، مقادیر' منفی' و مقدارهای 'عدد نیست' را پردازش کند و میانگین فقط مقادیر معتبر را محاسبه نماید

```
data = [12, 15, NaN, -3, 20, 25, NaN, 30, -1];
i = 1;
sum = 0;
count = 0;

while i <= length(data)
    if isnan(data(i)) || data(i) < 0
        i = i + 1;
        continue;  % مقدار نامعتبر، ادامه نده
    end

    sum = sum + data(i);
    count = count + 1;
    i = i + 1;
end

mean_value = sum / count;
fprintf('Mean of valid values: %.2f\n', mean_value)
```

```
Mean of valid values: 20.40
```

Practice 3: Print Numbers Not Divisible by 3 Between 1 and 50

Use a **for loop** to iterate through the numbers from 1 to 50.

If a number is divisible by 3, skip it using **`continue`**.

Print all other numbers.

Practice 4: Check for a **Perfect Number** Using **`break`**

Write a program that checks whether a given number `n` is a *perfect number* or not.

While summing up the positive divisors of `n` (excluding `n` itself), if the sum exceeds `n`, stop the process early using **`break`**—because the number can no longer be perfect.

A **perfect number** is a number whose sum of positive divisors (excluding itself) is equal to the number itself. Examples: 6 and 28.

Exercise 1: Gradient Descent Algorithm with a Stopping Condition Using `break`

Implement a simple gradient descent algorithm for the function

$$f(x) = x^3 - 6x^2 + 9x + 1,$$

such that the algorithm continues until the absolute value of the gradient is less than or equal to $\varepsilon = 0.01$.

Use $\alpha = 0.1$ as the learning rate, and once the stopping condition is met, exit the loop using the `break` statement.

Hint:

Gradient descent updates x in the direction that reduces the function. The update rule is:

$$x_{\text{new}} = x_{\text{old}} - \alpha \nabla f(x),$$

where α is the learning rate and $\nabla f(x)$ is the gradient of the function.

Exercise 2: Bubble Sort Algorithm with a Stopping Condition

mplement the Bubble Sort algorithm and add a condition to stop the loop if no changes were made to the array in a complete iteration. Use `continue` to skip unnecessary iterations.

Test it with the following array:

```
[38, 27, 43, 3, 9, 82, 10, 56, 45, 71, 92, 12, 56, 24, 87, 18, 63, 55, 28, 41]
```

The array should be sorted in ascending order by the end of the algorithm.

Hint:

The Bubble Sort algorithm is a simple sorting technique that compares adjacent elements in the list and swaps them if they are in the wrong order. After each pass, the largest unsorted element "bubbles" to its correct position at the end.

To optimize the algorithm, if no swaps are made in a complete iteration, it means the array is already sorted, and the algorithm can stop early. Use the `continue` statement to avoid unnecessary iterations when the array is partially sorted.