Week 7: Advanced Data Types 

Objectives:

- Understand and utilize structures in MATLAB 📦.
- Learn about cell arrays and their applications 🧩.
- Explore MATLAB tables for data organization and manipulation 📊.

Topics Covered:

- Structures (`struct`) 📦
- Cell Arrays (`cell`) 🧩
- Tables (`table`) 📊

Structures in MATLAB 📦

What is a Structure?

- A **structure** is a data type that groups related data using data containers called **fields**.
- Each field can contain any type of data.

Example:

```
student.name = 'Alice';
student.age = 20;
student.grades = [90, 85, 92];
```

Creating Structures

Method 1: Using Dot Notation

```
person.name = 'Bob';
person.age = 25;
person.isStudent = true;
```

Method 2: Using the `struct` Function

```
car = struct('make', 'Toyota', 'model', 'Camry', 'year', 2020);
```

Accessing and Modifying Fields

- Access fields using dot notation.

```
name = student.name;
student.age = 21;  % Modify the age field
```

- Adding new fields:

```
student.major = 'Engineering';
```

- Removing fields:

```
student = rmfield(student, 'grades');
```

- Using Dynamic Field Names:

```
fieldName = 'age';
value = student.(fieldName);
```

Arrays of Structures

- You can create arrays where each element is a structure.

```
students(1).name = 'Alice';
students(1).age = 20;

students(2).name = 'Bob';
students(2).age = 22;
```

- Access elements:

```
students(1).name  % Returns 'Alice'
```

Nested Structures

- Structures can contain other structures.

```
company.name = 'TechCorp';
company.employee.name = 'Charlie';
company.employee.position = 'Engineer';
```

- Access nested fields:

```
position = company.employee.position;
```

Looping Through Structure Arrays

Example:

```
for i = 1:length(students)
    fprintf('Student %d: %s, Age: %d\n', i, students(i).name, students(i).age);
end
```

Practical Example: Student Database

Objective: Create a structure array to store student information.

Steps:

```
% Initialize the structure array
students = struct('name', {}, 'age', {}, 'grades', {});

% Add students
students(1).name = 'Alice';
students(1).age = 20;
students(1).grades = [90, 85, 92];

students(2).name = 'Bob';
students(2).age = 22;
students(2).grades = [88, 79, 95];

% Accessing data
avgGrade = mean(students(1).grades);
fprintf('%s\'s average grade: %.2f\n', students(1).name, avgGrade);
```

💡 **Tip**: Preallocate structure arrays for efficiency.

Gotchas with Structures

- **Field Names Are Case-Sensitive**:

```
student.Name = 'Alice';  % Different from student.name
```

- **Accessing Non-Existent Fields**:

```
% This will throw an error if 'major' doesn't exist
major = student.major;
```

Cell Arrays in MATLAB 🧩

What is a Cell Array?

- A **cell array** is a data type that allows storage of data of varying types and sizes.
- Elements can be accessed using curly braces {}.

Example:

```
C = {'Text', 123, [1, 2, 3]; true, @sin, {'Nested', 'Cell'}};
```

Creating Cell Arrays

Method 1: Using Curly Braces

```
cellArray = {1, 'Hello', [1, 2, 3]};
```

Method 2: Using the `cell` Function

```
C = cell(2, 3);  % Creates a 2x3 cell array of empty matrices
```

Accessing Cell Array Elements

- **Access Cells** (the containers) using parentheses `( )`:

```
cellRef = cellArray(1);  % Returns a 1x1 cell
```

- **Access Contents** (the data inside) using curly braces `{ }`:

```
content = cellArray{1};  % Returns the actual data
```

- **Accessing Multiple Elements**:

```
contents = cellArray{1:2};  % Returns multiple outputs
```

Modifying Cell Arrays

- **Assigning to Cells**:

```
cellArray{2} = 'World';
```

- **Appending Data**:

```
cellArray{end+1} = 3.14;
```

- **Deleting Elements**:

```
cellArray(2) = [];  % Removes the second cell
```

Common Cell Array Functions

- **`cellfun`**: Apply a function to each cell.

```
lengths = cellfun(@length, cellArray);
```

- **`iscell`**: Check if a variable is a cell array.

```
isCellArray = iscell(cellArray);
```

- **Converting Between Cell and Regular Arrays**:

```
% From cell to matrix (if possible)
mat = cell2mat(cellArray);
% From matrix to cell
C = num2cell([1, 2, 3]);
```

Practical Example: Mixed Data Types

Objective: Store and manipulate data of different types.

Example:

```
data{1} = 'Temperature';
data{2} = [72, 75, 78, 80];
data{3} = datetime('now');

% Accessing data
fprintf('%s data recorded on %s\n', data{1}, datestr(data{3}));
temps = data{2};
avgTemp = mean(temps);
fprintf('Average temperature: %.2f°F\n', avgTemp);
```

Gotchas with Cell Arrays

- **Indexing Errors**:
  - Using `( )` vs. `{}` incorrectly.

```
% Incorrect: Returns a cell, not the content
value = cellArray(1);

% Correct:
value = cellArray{1};
```

- **Combining Cells and Arrays**:
  - Concatenation requires careful handling.

```
% Horizontal concatenation
newCellArray = [cellArray, {newElement}];

% Vertical concatenation
newCellArray = [cellArray; {newElement}];
```

- **Performance Considerations**:
  - Cell arrays are flexible but can be slower than regular arrays.

💡 Tips for Using Cell Arrays

- Use cell arrays when you need to store heterogeneous data.
- Preallocate cell arrays for better performance:

```
C = cell(1, N);  % Preallocate a 1xN cell array
```

- Use `cellfun` for applying functions without explicit loops.

Tables in MATLAB 

What is a Table?

- A **table** is a data type suitable for storing column-oriented or tabular data.
- Tables can store variables of different types but with the same number of rows.

Example:

```
T = table(['Alice'; 'Bob'], [25; 30], [68; 75], 'VariableNames', {'Name', 'Age', 'Weight'});
```

Creating Tables

Method 1: From Variables

```
Name = {'Alice'; 'Bob'; 'Charlie'};
Age = [25; 30; 35];
Height = [65; 70; 68];

T = table(Name, Age, Height);
```

Method 2: Reading from Files

```
T = readtable('data.csv');
```

Accessing Data in Tables

- **By Variable Names**:

```
ages = T.Age;
```

- **By Row and Variable Indices**:

```
value = T{2, 'Height'};  % Accesses the second row, 'Height' variable
```

- **Using Dot Notation**:

```
T.Weight = [150; 180; 175];  % Adds a new variable
```

Modifying Tables

- **Adding Rows**:

```
newRow = {'David', 28, 72};
T = [T; newRow];
```

- **Adding Variables (Columns)**:

```
T.BMI = T.Weight ./ (T.Height * 0.0254).^2;
```

- **Renaming Variables**:

```
T.Properties.VariableNames{'Height'} = 'HeightInches';
```

Table Properties

- **Variable Names**:

```
varNames = T.Properties.VariableNames;
```

- **Row Names**:

```
T.Properties.RowNames = {'Row1', 'Row2', 'Row3'};
```

- **Description and Units**:

```
T.Properties.Description = 'Participant Data';
T.Properties.VariableUnits = {'', 'years', 'inches', 'kg/m^2'};
```

Table Operations

- **Sorting Rows**:

```
T = sortrows(T, 'Age');  % Sorts table by 'Age' in ascending order
```

- **Filtering Data**:

```
T_over30 = T(T.Age > 30, :);
```

- **Merging Tables**:

```
T_full = join(T1, T2, 'Keys', 'Name');
```

- **Grouping and Aggregation**:

```
% Group by a variable and compute mean
avgHeight = varfun(@mean, T, 'GroupingVariables', 'Age');
```

Practical Example: Data Analysis

Objective: Analyze participant data stored in a table.

Steps:

```
% Create a table
Name = {'Alice'; 'Bob'; 'Charlie'};
Age = [25; 30; 35];
Height = [65; 70; 68];
Weight = [130; 180; 160];

T = table(Name, Age, Height, Weight);

% Calculate BMI
T.BMI = T.Weight ./ (T.Height * 0.0254).^2;

% Filter participants with BMI over 25
overweight = T(T.BMI > 25, :);

% Display results
disp('Participants with BMI over 25:');
disp(overweight);
```

Gotchas with Tables

- **Variable Names Must Be Valid MATLAB Identifiers**:

```
% Invalid variable name
T.'First Name' = {'Alice'; 'Bob'};
% Use valid names or set 'VariableNames' property
```

- **Accessing Data with Incorrect Indexing**:

```
% Incorrect: Using parentheses returns a subtable
subTable = T(1, 'Age');

% Correct: Using curly braces returns the data
ageValue = T{1, 'Age'};
```

- **Mixing Data Types**:
  - Ensure that each variable (column) contains data of the same type.

💡 Tips for Using Tables

- Use tables for data analysis tasks involving heterogeneous data.
- Utilize table functions like `summary`, `head`, `tail` for quick insights.

```
summary(T);
```

- Tables integrate well with MATLAB's plotting functions.

```
scatter(T.Age, T.BMI);
xlabel('Age');
ylabel('BMI');
```

Comparing Structures, Cell Arrays, and Tables

Feature	Structures	Cell Arrays	Tables
Heterogeneous Data	✅	✅	✅
Named Fields/Variables	✅	❌	✅
Indexed Access	Limited	✅	✅
Size Flexibility	✅	✅	Rows only
Ideal Use Case	Complex data	Mixed types	Tabular Data

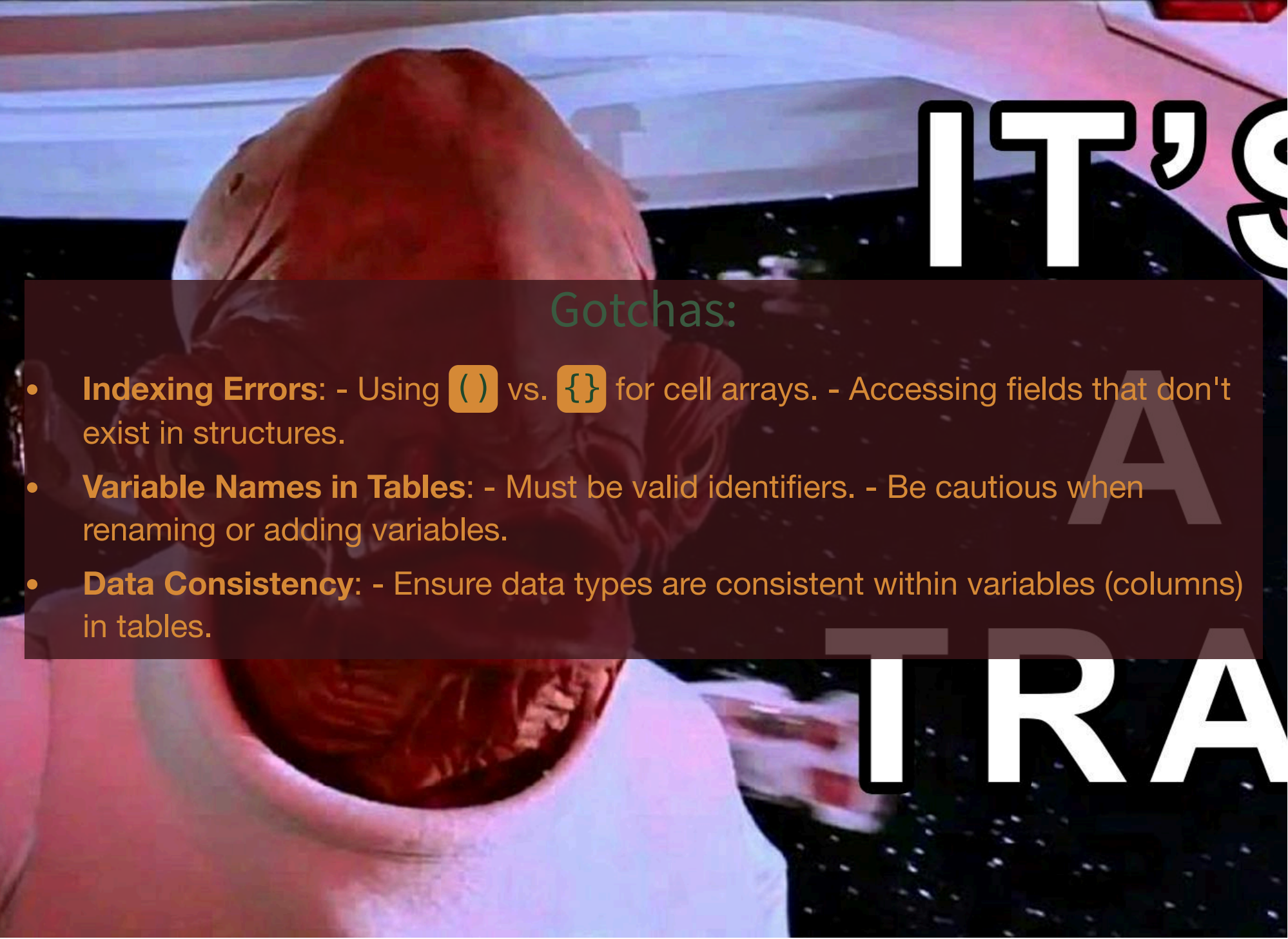
Choosing the Right Data Type

- **Structures**:
  - Use when you need to group related data with named fields.
  - Ideal for representing complex objects.
- **Cell Arrays**:
  - Use when you need indexed access to heterogeneous data.
  - Suitable for lists of mixed-type data.
- **Tables**:
  - Use for column-oriented data analysis.
  - Provides rich functionality for data manipulation.

Key Takeaways 🎓

- **Structures** are versatile containers with named fields, ideal for complex data.
- **Cell Arrays** allow storage of heterogeneous data with indexed access.
- **Tables** are powerful for organizing and analyzing tabular data with different types.
- Choose the appropriate data type based on the nature of your data and task requirements.


IT'S
A
TRA



Gotchas:

- **Indexing Errors**: - Using `( )` vs. `{}` for cell arrays. - Accessing fields that don't exist in structures.
- **Variable Names in Tables**: - Must be valid identifiers. - Be cautious when renaming or adding variables.
- **Data Consistency**: - Ensure data types are consistent within variables (columns) in tables.

Further Resources 

- MATLAB Documentation: Structures
- MATLAB Documentation: Cell Arrays
- MATLAB Documentation: Tables