"**The Elements of MATLAB Style**"

- by Richard K. Johnson (Cambridge, 2011)

47. Follow a Consistent Convention on Pluralization
48. Use the Prefix `this` for the Current Variable
49. Use the Suffix `No` or Prefix `i` for Variables Representing a Single Entity Number
50. Prefix Iterator Variables with `i`, `j`, `k`, etc.
51. Embed `is`, `has`, etc. in Boolean Variable Names
52. Avoid Negated Boolean Variable Names
53. Use the Expected Logical Names and Values
54. Avoid Using a Keyword or Special Value Name for a Variable Name
55. Avoid Hungarian Notation
56. Avoid Variable Names that Shadow Functions
57. Avoid Reusing a Variable for Different Contents
58. Consider a Unit Suffix for Names of Dimensioned Quantities

Constants

59. Use All Uppercase for Constant Names with Local Scope
60. Use Function Names for Constants Defined by Functions
61. Use Meaningful Names for Constants
62. Define Related Constants Based on the Relation
63. Consider Using a Category Prefix

Structures and Cell Arrays

64. Use UpperCamelCase for Structure Names
65. Do Not Include the Name of the Structure in a Fieldname
66. Use Fieldnames that Follow the Naming Convention for Variables
67. Name Cell Arrays Following the Style for Variables

Functions

68. Give Functions Meaningful Names
69. Name Functions for What They Do
70. Follow a Case Convention for Function Names
71. Reserve the Prefixes `get`/`set` for Accessing an Object Property
72. Use Expected Verbs in Expected Ways
73. Use the Prefix `is` for Boolean Functions
74. Use Complement Prefixes in Compound Names for Complement Operations
75. Be Selective in the Use of Numbers at the Ends of Names
76. Use Numbers Inside Function Names Only for Common Conventions
77. Avoid Unintentional Shadowing

Classes

78. Use Nouns When Naming Classes
79. Use UpperCamelCase for MATLAB Class and Object Names
80. Use UpperCamelCase for Exception Names
81. Name Properties Like Structure Fields
82. Name Methods Line Functions
83. Name Accessor Methods after their Properties
84. Use a Single Lowercase Word as the Root Name of a Package

Data Files and Directories

85. Use Directory and Filenames that are Easy to Work with
86. Use Sortable Numbering in Data Filenames
87. Use ISO Data Format

DOCUMENTATION

General

88. Provide Well-Written Code
89. Document Each Module Before or During Its Implementation
90. Document the Interface for Those Who Will Use It
91. Document the Design and Implementation for Those Who Will Maintain It
92. Consider Documenting Some Changes Header Comments
93. Generate HTML Reference Pages
94. Integrate the Reference Pages with the Help Browser
95. Consider Providing PDF Documentation

Comments

96. Make Comments Useful
97. Be Sure that Comments Agree with the Code

98. Revise Comments to be Correct When the Code is Changed
99. Put Restrictions in the Code, Not the Comments
100. Clean up Commented Out Code before Release
101. Make Comments Easy to Read
102. Write Comments for the Publish Feature
103. Minimize Use of End-Line Comments
104. Consider End-of-Loop Comments
105. Consider Documenting Important Variables Near the Start of the File
106. Consider Documenting Constant Assignments Where They are Defined
107. Use Voice and Person Appropriately
108. Use Present Tense to Describe Code
109. Use Complete Sentences in Descriptive Blocks
110. You Can Use Incomplete Sentences in One-Liners
111. Use Short Words
112. Eliminate Cute Comments
113. Minimize the Use of ASCII Art
114. Write All Comments in English

Header Comments

115. Format the Header Comments for Easy Publishing as Documentation
116. Put the Header Comments in the Right Place
117. Use Title Markup for the Function Name
118. Document the Function Interface in the Syntax Section
119. Use the Actual Function Name Case in Comments
120. Describe the Function Arguments in the Description Section
121. Describe Any Function Side Effects
122. Describe Any Custom Exception that May be Generated
123. Include Examples in the Header Comments
124. Include See also in the Header Comments
125. Avoid Clutter in the Reference Page
126. Format Header Comments of `classdef` Files for the Help Browser
127. Clarify Subclass Methods

Block Comments

128. Indent Block Comments to Match Code
129. Place a Blank Line or Cell Break before a Block Comment

Minimize y within constraints on x

130. Do Not Use Comment Blocks for Block Comments
131. Use Comment Block Syntax to Temporarily Bypass a Block of Code

Interspersed or Inline Comments

132. Indent Comments with the Code Block
133. Do Not Follow a Single-Line Comment with a Blank Line
134. Resolve TODO/FIXME Comments

PROGRAMMING

General

135. Avoid Cryptic Code
136. Avoid Off-By-One Mistakes
137. Attend to NaN Results
138. Consider Using `isfinite`
139. Use Programming Patterns or Idioms

Variables and Constants

140. Do Not Reuse Variable Names Unless Required by Memory Limitation
141. Beware of Mistyping Variable Names
142. Minimize the Use of Literal Numbers in Statements
143. Write Floating Point Values with a Digit Before the Decimal Point
144. Avoid Showing Excessive Decimal Places
145. Avoid Mixing Units within a Program
146. Use Caution with Floating Point Comparison
147. Limit Boolean Variable Values to True or False
148. Do Not Assume Array Size
149. Use Appropriate Numerical Class Conversions
150. Minimize the Use of Global Variables

151. Minimize the Use of Global Constants
152. Define Local Constants Only Once
153. Do Not Declare Unrelated Variables in a Single Line

Character Strings

154. Consider Using `strcmpi`
155. Use `lower` or `upper` When You Cannot Use `strcmpi`
156. Use `isempty`
157. Use `fullfile`
158. Compute with Numbers for Speed
159. Consider Using Character Arrays for Speed
160. Consider Using `unique`

Structures

161. Use Structures for Associated Data
162. Use Structures for Metadata
163. Organize a Structure Based on How It Will be Accessed
164. Put Structure Field in a Helpful Order
165. Be Careful with Fieldnames

Cell Arrays

166. Consider Using Cell Arrays for String
167. Look Out for Cells within Cells
168. Consider Using Cell Arrays for Ragged Arrays

Expressions

169. Write Short Expressions
170. Put Numbers Before the Multiplication Operator
171. Make the Denominator Clear
172. Use Parentheses
173. Use a Clear Form for Relational Expressions
174. Use `&&` or `||` by Default for Scalar Operands

Statements

175. Write Short Statements
176. Avoid Use of `eval` When Possible

Loops

177. Initialize Loop Variables Immediately Before the Loop
178. Initialize Using `nan` or `false` Rather Than `zeros`
179. Do Not Change the Loop Index Variable Inside a Loop
180. Minimize the Use of `break` in Loops
181. Minimize the Use of `continue` in Loops
182. Avoid Unnecessary Computation within Loops
183. Be Careful of Infinite `while` Loops
184. Be Careful of Count Errors in `while` Loops

Conditionals

185. Avoid Complicated Conditional Expressions
186. In General, Include an `else` Statement with `if`
187. Put the Normal Branch in the `if` Part
188. Avoid Unnecessary `if` Constructs
189. Use a Practical Order for `elseif` Conditions
190. Avoid Unnecessary `elseif` Blocks
191. Avoid Unnecessary Levels of Nesting in Control Structures
192. Try to Simplify Nested `if` Constructs
193. Avoid the Conditional Expression `if 0`
194. Include `otherwise` with `switch` Statements
195. Consider Using a Cell Array to Simplify a `switch` Construct
196. Use if When the Condition `is` Most Clearly Written as an Expression
197. Use `switch` When the Condition is Most Clearly Written as a Value
198. When Either `if` or `switch` Can Work, Use the More Readable One

Logical Functions

199. Use `logical` to Cast Variables
200. Use `true` or `false` Functions and Values

201. In General Use `isequal` Rather Than `==`

Vectorization

202. Be Thoughtful with Vectorization
203. Use `repmat`

Functions

204. Modularize
205. Write Code as Functions When Possible
206. Write Simple Functions
207. Design for Loose Coupling
208. Use Subfunctions Appropriately
209. Hide Implementation Details
210. Write for High Cohesion
211. Use Existing Functions
212. Eliminate Overlapping Functions
213. Provide Some Generality in Functions
214. Write a Function at One Level of Abstraction
215. Write Convenience Functions
216. Make Interaction Clear
217. Name All Input Arguments
218. Write Boolean Functions to Return `true` or `false`
219. Make Logical Output and Function Name Consistent
220. Minimize Input Flag Arguments
221. Write Arguments in Useful Order
222. Use Lazy Evaluation
223. Make Input and Output Arrays Consistent
224. Use a Structure to Replace a Long List of Function Arguments
225. Consider an Options Structure
226. Consider `varargin` and `varargout`
227. Use Parameter-Value Pairs for Optional, Unordered Input Arguments
228. In General, Use Caller Variable Names Consistent with the Function Argument Names
229. Define Imports Where They are Easy to Find
230. Use Anonymous Functions Rather Than Inline Functions
231. Use Function Handles
232. Avoid Including Hidden Side Effects
233. Refactor

Input and Output

234. Write Input Functions
235. Provide Some Input Validity Checking
236. Expect NaN Values in Data
237. Use `feof` for Reading Files
238. Make Output Modules
239. Format Output for Easy Use
240. Provide for Automation

Classes and Objects

241. Keep Classes Simple
242. Avoid Classes with Ambiguous Overlap
243. Construct Valid Objects
244. Follow Constructor Conventions
245. Define Small, Simple Methods
246. Write Methods That You Can Unit Test
247. Do Not Write a Method That Can Produce an Invalid Property
248. Avoid Incomplete Public Methods
249. Try to Make Properties Private
250. Do Not Expose Properties with Behavior
251. Avoid Writing Methods with Many Input Arguments
252. Validate Method Argument Values
253. Check for a Property's Existence Before Using It
254. Refactor Repeated Code Into Methods
255. Overload Standard Functions When Possible

256. Avoid Unconventional Usage of Overload Names
257. Do not Overload `&&` or `||`
258. Do not Get Carried Away with Inheritance
259. Place Method Functions in Folders Consistently
260. Use Java Syntax for Java Methods

Exceptions, Errors, and Warnings

261. Use Appropriate Error Handling
262. Prepare for Errors
263. Make Error Messages Informative
264. Use Message IDs with Errors and Warnings
265. Use Exceptions
266. Support Error Returns
267. Use Appropriate Assertions

Output Style

268. Learn to Use `strintf`
269. Learn to Use Tex

Tests

270. Write at Least One Test for Every Function or Method
271. Write Small Tests
272. Write Uncoupled Tests
273. Write Tests with Boolean Outputs
274. Test for Expected Exceptions
275. Write Tests You Can Automate
276. Use Related Names in the Function and the Test Code
277. Identify Test Files by Name
278. Develop Test Patterns
279. Consider Tests for External Software

Data Files

280. Make Use of mat Files
281. Follow Database Conventions
282. Follow the MATLAB Convention for Data Array Orientation

FILES AND ORGANIZATION

Toolboxes

284. Organize General-Purpose m-Files in Toolboxes
285. Put Test Files in a Separate Directory
286. Consider Writing Demo Files
287. Use a Consistent Toolbox Folder Organization
288. Provide for Integration with MATLAB
289. Provide a Reference Page for Every Public Function
290. Integrate the Reference Pages with the Help Browser
291. Do not Make Your Toolbox a Subfolder of the MATLAB Folder

Project Files

292. Organize Your Project-Specific Files by Project Directory
293. Organize Your Data Directory for Ease of Access

DEVELOPMENT

Design

294. Expect Change
295. Include Appropriate Flexibility
296. Use the Right Algorithm
297. Program by Contract
298. Write for Automation
299. Make Associated Data Easy to Use
300. Recompute When Data Changes
301. Leave Code Optimization for Last or Never
302. Consider Coding Standards

Development Practices

303. Develop in Small Steps
304. Develop in the Editor
305. Use Version Control
306. Run Tests Often

307. Run all Tests Before Release

MATLAB IDE Tools

308. Try the MATLAB Editor
309. Use Smart Indent
310. Use Find and Replace
311. Pay Attention to M-Lint
312. Use Consistent Preference Settings
313. Use the Debugger Effectively
314. Use The TODO/FIXME Report
315. Use the Profiler
316. Use the Dependency Report
317. Publish During Development
318. Find a Desktop Layout That Works for You