MATLAB スタイルガイドライン 2.0
MATLAB Style Guidelines 2.0

リチャード ジョンソン (川又 政征*, 八巻 俊輔†訳)
Richard Johnson, translated by Masayuki Kawamata‡ and Shunsuke Yamaki§

2017 年 7 月 28 日

訳者による序

本文書は Richard Johnson 博士著 MATLAB Style Guidelines 2.0 の日本語訳である[*1].

MATLAB のプログラムミングの原則とよい事例を探しているときに，このガイドラインを見つけた．これが求めていたものであることが一目して分かった．高級言語によるプログラミング全般に共通する書法と MATLAB の特徴を活かした書法の双方が，この文書に明快に整理されている．本文書中の番号のついた項目をながめるだけでも，プログラムの質が今日から格段に向上するはずである．

著者の Richard Johnson 博士には本文書の日本語訳とその配布について快諾をいただいた．ここに心より感謝する．

1 はじめに
Introduction

MATLAB® のコードの作成に対しての助言は通常は効率に関心があり，たとえば「ループを使うな」というような推奨をする．この文書は違う．ここでの関心は，正確性，明瞭性と一般性である．これらのガイドラインの目標は，正しく，理解可能であり，共有可能であり，かつ保守可能となるようなコードを作り出す助けとなることである．

あるコーディング法は他のコーディング法よりも優れている．それくらい単純なことである．コーディングの慣習は間違いを目に見えるようにすることで価値を高めている．Brian Kernighan いわく「良く書かれたプログラムは悪く書かれたものよりも優れている — それはエラーが少なく，デバッグと修正がより容易だからだ — だから初めからスタイルについて考えることが重要なのだ」

他人（ひと）があなたのコードを見るとき，あなたがやろうとしていることが分かるだろうか？この本の精神は「コードの書きっぱなしを避けよ」というように単純に表される．

この文書は MATLAB コーディングのための推奨事項を列挙している．この推奨事項はソフトウェア開発コミューニティにおける最良の実践と一致するものである．これらのガイドラインは，C， C++ および Java のためのものと一般的には同じであるが，MATLAB の特徴と歴史を考えての修正もある．推奨事項は他の言語のためのガイドラインに基づいており，これらのガイドラインは多数のソースと個人的経験から収集されたものである．これらのガイドラインは MATLAB を念頭に置いて書かれており，Octave，Scilab および O-Matrix のような関連のある言語のためにも役に立つはずである．

MATLAB 言語が変化するにつれ，そしてその利用がさらに広がるにつれ，スタイルについての論点がますます重要になっている．始めのころの版では変数はすべて倍精度行列であった；今では多くのデータの型が利用できる．小規模のプロトタイプコードからグループによって開発される大規模で複雑な製品のコードにまで，MATLAB の利用が成長している．Java との統合は標準となり，Java のクラスは MATLAB コードにおいても使うことができる．これらの変化のすべてにより，明快

* 東北大学 大学院工学研究科 電子工学専攻
† 東北大学 サイバーサイエンスセンター 研究開発部 先端情報技術研究部
yamaki@yoshizawa.ecei.tohoku.ac.jp
‡ Deparment of Electronic Engineering, Graduate School of Engineering, Tohoku University
§ Research and Development Divisions, Cyberscience Center, Tohoku University

ISBN
Library of Congress Control Number 3
ダウンロード
http://jp.mathworks.com/matlabcentral/fileexchange/46056-matlab-style-guidelines-2-0

なコードを書くことはさらに重要になり，さらに興味をそそるものとなっている．

ガイドラインは戒律ではない．その目標はプログラマーが良いものを書くことをただ単に手助けすることである．理由があって，これらのガイドラインからいくらか外れる組織もあるだろうが，ほとんどの組織がスタイルガイドラインをいくらかでも採用することによって利益を得るだろう．

MATLAB のスタイルと最良の開発実践のより広くかつ深い範囲を取り上げたものについては

The Elements of MATLAB Style

を参照してほしい．以下のところか **Amazon** で取り扱っている：

http://datatool.com/resources.html

MATLAB は The MathWorks, Inc. の登録商標である．本資料において MathWorks は The MathWorks, Inc. の意である．もし訂正や批評・意見があれば richj@datatool.com に連絡をしてほしい．

2 命名の慣習
Naming Conventions

ソフトウェアの命名の慣習の目的は読み手とプログラマーを助けることである．開発者のグループのために命名の慣習を作り上げることはとても重要なことであるが，その過程はばかばかしくも論争になってしまう．すべての人を喜ばせるような命名の慣習はない．

ある慣習に従うことはその慣習の詳細が何であるのかということよりも重要である．本章は共通に使われている慣習を記述する．この慣習は MATLAB と他の言語の多くのプログラマーに親しまれてる．

2.1 変数
Variables

変数の名前はその意味あるいは利用法を記述すべきである．MATLAB は

```
z = x * y
```

にうまく対処できるが，読み手にとっては

```
wage = hourlyRate * nHours
```

の方がうまく対処できるだろう．

2.1.1 小文字で始めてミクストケースで変数名を書け
Write variable names in mixed case starting with lower case.

これは他の言語にも共通する実践である[*2]．大文字で始まる名前は他の言語では通例として型あるいは構造体のために予約されている．

```
linearity, credibleThreat, qualityOfLife
```

とても短い変数名は大文字にすることができるものの，それが慣習的な使用法で大文字になっていて，合成変数名の部分となりそうもないことが条件である．その例は概して領域に特有のものである．たとえばヤング率のための E のようなものがあるが，これとて e のように誤った印象を与えやすい．

合成変数名の部分を分離するためにアンダースコアを使うことを好むプログラマーもいる．この技法は可読性があるものの，他の言語における変数名のためによく使われるわけではない．グラフの題名，ラベル，凡例にアンダースコアを使うことについて一つ配慮すべきことは，MATLAB における TEX インタープリターがアンダースコアを下付き文字への切り替えとして解釈するので，パラメータ／値の対 'interpreter', 'none' を個々のテキスト文字列に適用することが必要となることである．

2.1.2 大きなスコープの変数は意味の分かる名前を持たなければならない．小さいスコープの変数は短い名前でもよい．
Variables with a large scope should have meaningful names. Variables with a small scope can have short names.

実際上ほとんどの変数は意味の分かる名前を持たなければならない．短い名前はそれがステートメントの構造を明瞭にするか，あるいは一般的な名前を表すという意図であれば利用のために確保すべきである．たとえば一般的な目的の関数において，`x, y, z, t` のような変数の名前を使うことは適切であろう．

一時的な保管あるいはインデックスのために使われるスクラッチ変数は短くしておいてもよい．そのような変数を読むプログラマーはその値がコードの数行離れたところでは利用されないと当然考えてよいはずである．整数として使われるスクラッチ変数のための常識的名前は `k, m, n` であり，倍精度実数に対して `s, t, x, y` そ

[*2] 訳者注：ミクストケース mixed case とは，頭文字を大文字にして残りは小文字で書くことを言う．

して z である．

複素数をあつかうプログラマーは i または j，あるいは両方を -1 の平方根のためにとっておくことを選択するかもしれない．しかし The MathWorks は虚数のために 1i または 1j を使うことを推奨している．これらはより高速に動作し，書き換えられることはない．

2.1.3 対象の数を表すための変数の接頭辞 n を使え．

Use the prefix n for variables representing the number of objects.

この記法は数学からとられている[*3]．数学ではこの記法は対象の数を指定するための確立した慣習となっている．

```
nFiles, nSegments
```

MATLAB に特有の選択肢として以下のような（行列の記法に基づいて）行の数のための m の利用がある．

```
mRows
```

2.1.4 複数化についての一貫した慣習に従え．

Follow a consistent convention on pluralization.

最後の文字 s だけが異なる名前の二つの変数を使うことは避けなければならない．すべての変数の名前を単数形か複数形のどちらかにするプログラマーがいる一方で，これを不便なものと見るプログラマーもいる．

複数のものに対しての許容できる使用法は Array のような接尾辞[*4]を用いることである．

```
point, pointArray, PointList
```

単数のものに対しての許容できる使用法は以下のような接頭辞を用いることである．

```
thisPoint
```

ほとんどのプログラマーは単一の事例あるいは要素のために the を接頭辞として使うことはない．

2.1.5 単一の存在を表す変数名に接尾辞 *No* または *Num* あるいは接頭辞 *i* を使え．

Use the suffix *No* or *Num* or the prefix *i* in a variable name representing a single entity number.

No という記法は数学からとられている．それは数学では存在の数を指定するための確立した慣習である．

```
tableNo, employeeNo
```

接頭辞 i を付けると変数がうまく命名された反復子になる．

```
iTable, iEmployee
```

2.1.6 *i, j, k* などを反復子変数名の頭に付けよ．

Prefix iterator variable names with *i, j, k* etc.

この記法は数学からとられている．それは数学では反復子を指定するための確立された慣習である．

```
for iFile = 1:nFiles
   :
end
```

単一文字の変数名 i, j またはその両方を便利なループ反復子のために用いるプログラマーもいる．明示的な複素数を使うプログラマーはこの実践をきらう傾向がある．

入れ子のループのためには反復子変数は通常アルファベットの順になっていなければならない．行のために i，列のために j から始まる変数名を使う数学的指向のあるプログラマーもいる．

とくに入れ子になっているループのためには反復子変数名を使うことは役に立つ．

```
for iFile = 1:nFiles
   for jPosition = 1:nPositions
    :
   end
   :
end
```

2.1.7 否定的ブール変数名を避けよ．

Avoid negated Boolean variable names.

論理的否定演算をもつ連接（連言）[*5]にそのような名前が使われるときに問題がおきる．これが二重の否定に帰着するからである．以下のような名前が意味することは瞬間的には分からない．

```
~isNotFound
```

以下のものを使え．

```
isFound, ~isFound
```

以下のものは避けよ．

[*3] 訳者注：接頭辞 prefix とは，単語本体の前につく語のことを言う．

[*4] 訳者注：接尾辞 suffix とは，単語本体の後ろにつく語のことを言う．

[*5] 訳者注：連接（連言）とは，命題 A と B に対して 命題 A and B のことを言う．

```
isNotFound
```

2.1.8 頭字語は普通は大文字であるとしてもミクストケースあるいは小文字で書かれなければならない.

Acronyms, even if normally uppercase, should be written in mixed or lower case.

すべて大文字にしてしまうことは標準的な命名の慣習と矛盾することになるだろう[*6]. この種の変数は dVD, hTML などのように命名されなければならない，これらは明らかにあまり読みやすいものではない．このような名前が他の名前と結合されるとき可読性が極めて減少する．省略形が目立ってしまうので省略形の後に続く単語は目立たなくなる.

以下のものを使え.

```
html, isUsaSpecific, checkTiffFormat()
```

以下のものを避けよ.

```
hTML, isUSASpecific, checkTIFFFormat()
```

2.1.9 変数名にキーワードあるいは特殊な値の変数名を使うことを避けよ.

Avoid using a keyword or special value name for a variable name.

予約された単語あるいはビルトインの特別な値が再定義されるならば，MATLAB は暗号のようなエラーメッセージを出すか，あるいはおかしな結果が生じる．予約されている単語はコマンド iskeyword によって列挙される．特別な値はドキュメンテーションにおいて列挙されている.

2.1.10 領域に特有で共通の名前を用いよ.

Use common domain-specific names.

ソフトウェアがある知識領域あるいはある利用者グループを対象としているならば，標準的な実践と一致する名前を使え.

以下のものを使え.

```
roi, regionOfInterest
```

以下のものを避けよ.

```
imageRegionForAnalysis
```

2.1.11 関数をシャドウする変数名を避けよ.

Avoid variable names that shadow functions.

MATLAB が提供しているものには変数名として使いたくなるような関数名がいくつもある．スクリプト中のそのような使用法は関数をシャドウして，エラーに結びついてしまう．関数の中で同じ名前の変数と関数を使うことはたぶんエラーを引き起こすだろう.

コードの例の中で変数として使われていた標準的な関数名は以下のものである.

```
alpha, angle, axes, axis, balance, beta,
contrast, gamma, image, info, input,
length, line, mode, power, rank, run,
start, text, type
```

よく知られた関数名を変数名として使うことは可読性を減少させることにもなる．もし変数名の中で length のような標準的な関数名を使いたいならば，そのときは単位を後ろにつけるか，あるいは名詞か形容詞を前に付けた方がよい.

```
lengthCm, armLength, thisLength
```

2.1.12 ハンガリアン記法を避けよ.

Avoid Hungarian notation.

ソフトウェア開発者によって使われているハンガリアン記法には少なくとも二つの形式がある．ハンガリアン記法における変数名が典型的に必要としているものは 1 つか 2 つの接頭辞，名前本体，および修飾語である．これらの名前はひどく醜くなってしまう．それらが短縮形の文字列であるときはなおさらである．しばしば示唆されることではあるが，接頭辞がデータ型を符号化しているならば，より大きな問題が起きる．そのとき型を変更する必要がでてくれば，その変数名に関わるすべての範囲を変更する必要がある.

以下のものを使え.

```
thetaDegrees
```

以下のものを避けよ.

```
uint8thetaDegrees
```

2.2 定数

Constants

MATLAB 言語は真の定数というものは持っていない（オブジェクトにおける定数プロパティを除いて）．定数が認識されるように，かつ不用意に再定義されないよう

[*6] 訳者注：頭字語とは，アルファベットで表された複数の単語から作られた合成語の頭文字をつなげて作られる語である．たとえば， Digital Video Disc に対して DVD .

に，定数の命名と定義には標準的な実践を用いよ．

2.2.1 **(m-ファイル内の) 局所スコープを持つ定数の名前はすべて大文字としなければならず，単語を分けるためにアンダースコアを使用しなければならない．**

Constant names with local scope (within an m-file) should be all uppercase using underscore to separate words.

これは他の言語においても共通する実践である．

```
MAX_ITERATIONS, COLOR_RED
```

定数のために分かりやすい名前を使え．

以下のものを使え．

```
MAX_ITERATIONS
```

以下のものを避けよ．

```
TEN, MAXIT
```

2.2.2 **関数による出力である定数とその関数名が同じ場合には，定数はすべて小文字かミクストケースである名前にすべきである．**

Constants that are output by a function with the same name should have names that are all lowercase or mixed case.

この実践は The MathWorks によって使われている．たとえば，定数 `pi` は実際には関数である．

```
offset, standardValue
```

2.2.3 **定数には共通の型名を前においた方がよい．**

Constants can be prefixed by a common type name.

これによって定数が何に属するのか，およびどんな概念を表すのかということについての追加の情報が与えられる．

```
COLOR_RED, COLOR_GREEN, COLOR_BLUE
```

2.3 構造体

Structures

2.3.1 **構造体の名前は大文字一つで始まらなければならない．**

Structure names should begin with a capital letter.

この使用法は他の言語の場合と一致し，構造体と通常の変数を区別する助けになる．

2.3.2 **構造体の名前は暗黙のものとし，フィールド名の中に含める必要はない．**

The name of the structure is implicit, and need not be included in a fieldname.

繰り返しは使うときに余分である．

以下のものを使え．

```
Segment.length
```

以下のものを避けよ．

```
Segment.segmentLength
```

2.3.3 **フィールド名には注意せよ．**

Be Careful with Fieldnames.

構造体フィールドの値を設定するとき，もしそのフィールドがすでに存在するならば MATLAB は既存の値を置き換える．もし存在しなければ MATLAB は新しいフィールドを作り出す．もしフィールド名が一致しなければ，このとことは予期せぬ結果に至る．たとえば，構造体がフィルード

```
Acme.source = 'CNN';
```

を持っていて，これを更新しようとしているとき，

```
Acme.sourceName = 'Bloomberg';
```

とタイプしてしまったら，この構造体は今や二つのフィールドを持つようになるだろう．

2.4 関数

Functions

関数の名前はその使用法を正しく伝えなければならない．

2.4.1 **関数の名前を小文字あるいはミクストケースで書け．**

Write names of functions in lower or mixed case.

もともと MATLAB の関数名はみな小文字になっていた．

```
linspace, meshgrid
```

The MathWorks によって提供されるほとんどすべての関数はまだこの慣習に従っている．この実践は，短命であった以下の例のように，長い合成名に対しては厄介なことになってしまう．

```
isequalwithequalnans
```

他の言語では関数名に対して小文字で始まるミクストケースを使うことが常識となっている．多くのMATLABプログラマーはこの慣習に従っている．

```
predictSeaLevel, publishHelpPages
```

関数の名前にアンダースコアを用いることを好むプログラマーもいるが，しかしこの実践は常識とはなっていない．

2.4.2 意味のある関数名を使え．

Use meaningful function names.

MATLABには，短くてしばしば少し暗号がかった関数名を使うという残念な伝統がある — たぶん古いDOS8の文字の制約のせいである．このような配慮はもはや妥当ではなく，この伝統は可読性の向上のために通常は避けなければならない．

以下のものを使え．

```
computeTotalWidth
```

以下のものを避けよ．

```
compwid
```

数学において広く使用されている短縮形あるいは頭字語の使用は例外である．

```
max, gcd
```

そのような短い名前を持つ関数はヘッダーコメント行において明らかに説明しておかなければならない．

2.4.3 単一の出力を持つ関数はその出力に基づいて命名せよ．

Name functions that have a single output based on the output.

これはMathWorksコードにおける共通の実践である．

```
mean, standardError
```

2.4.4 出力引数のない関数あるいはハンドルを返すだけの関数は，それが行うことにちなんで命名せよ．

Functions with no output argument or which only return a handle should be named after what they do.

この実践は可読性を向上させ，関数が行うべきこと（そしてできれば，行ってはいけないこと）を明白にする．これによって思わぬ副作用からコードを容易に守ることができる．

```
plot
```

2.4.5 オブジェクトあるいはプロパティにアクセスする関数のために接頭辞 *get/set* を予約せよ．

Reserve the prefixes *get/set* for functions that access an object or property.

これはThe MathWorksの一般的な実践であり，他の言語においても共通する実践である．例外としては，論理的setの演算のための `set` の使用することであるが，これは仕方がない．

```
getobj, setAppData
```

2.4.6 何かを計算する関数のために接頭辞 *compute* を予約せよ．

Reserve the prefix *compute* for functions where something is computed.

この用語の守備一貫した使用は可読性を強化する．関数が行おうとしていることについての直接的な手がかりを読み手に与えよ．

```
computeWeightedAverage, computeSpread
```

`find` あるいは `make` のような，たぶん混乱を起こしそうな代案は避けよ．

2.4.7 何かを探す関数のために接頭辞 *find* を予約することを検討せよ．

Consider reserving the prefix find for functions where something is looked up.

これが必要な計算が最小である探索法であり，かつ単純であるという直接的な手がかりを読み手に与えよ．この用語の守備一貫した使用は可読性を強化し，接頭辞 `get` の過剰な使用を防ぐためのよい代案となる．

```
findOldestRecord, findTallestMan
```

2.4.8 オブジェクトあるいは変数が設定されるところでは接頭辞 *initialize* を使うことを考えよ．

Consider using the prefix *initialize* where an object or a variable is established.

ブリテン語の *initialise* よりも米語の *initialize* が好まれるはずである．省略形 `init` を避けよ．

```
initializeProblemState
```

2.4.9 ブール関数のために接頭辞 *is* を使え.

Use the prefix *is* for Boolean functions.

これは他の言語と同様に MathWorks コードにおいても共通の実践である.

```
isOverpriced, iscomplete
```

接頭辞 `is` よりもある状況にはよく適合する代案がいくつかある. これらの中には接頭辞 `has`, `can` と `should` がある:

```
hasLicense, canEvaluate, shouldSort
```

2.4.10 相補的な演算には相補的な名前を使え.

Use complement names for complement operations.

対称性によって複雑さを減少させよ.

```
get/set, add/remove, create/destroy,
start/stop, insert/delete,
increment/decrement, old/new, begin/end,
first/last, up/down, min/max, next/previous,
open/close, show/hide, suspend/resume,
```

など.

2.4.11 意図しないシャドウイングを避けよ.

Avoid unintentional shadowing.

一般的に関数名は唯一でなければならない. シャドウイング（同じ名前をもつ二つあるいはそれ以上の関数を使うこと）は予期しない振る舞いあるいはエラーの可能性を増加させる. `which -all` または `exist` を使って名前のシャドウイングの検査を行うことができる.

オーバーロード関数はもちろん同一の名前を持つだろう. 多様型関数が妥当であるときにオーバーロードの状況を作り出すな.

2.5 一般的事項について

General

2.5.1 次元付の変数と定数のために単位の接尾辞を考えよ.

Consider a unit suffix for dimensioned variables and constants.

完全に実行できることは本当にまれであるが, 開発計画のために統一した単位の集合を使うことは魅力的な考えである. 単位の接尾辞を付加することで無意識では避けがたい混ぜこぜの単位の表現を避けることができる.

```
incidentAngleRadians
```

2.5.2 名前の中の短縮形を最小限にせよ.

Minimize abbreviations in names.

単語全体を使うことは曖昧性を減らし, コード自身によるキュメンテーションを助ける.

以下のものを使え.

```
computeArrivalTime
```

以下のものを避けよ.

```
comparr
```

短縮形あるいは頭字語を通してより自然に知られている領域特有の言い回しは省略を維持しなければならない. この場合でさえ, 省略形が初めて現れるときにそれを定義するコメントを書くことで利益がもたらされるだろう.

```
html, cpu, cm
```

2.5.3 名前を発音できるようにすることを考えよ.

Consider making names pronounceable.

ほんの少しでも発音可能な名前は読んだり覚えたりすることがより容易である.

2.5.4 名前を英語で書け.

Write names in English.

MATLAB の流通資料は英語で書かれており, そして英語は国際間での開発のために好まれる唯一の言語である.

3 ステートメント
Statements

3.1 変数と定数

Variables and constants

3.1.1 メモリ制限による要求がないならば変数を再利用してはいけない.

Variables should not be reused unless required by memory limitation.

すべての概念をそれぞれ唯一に表現することを確実に行うことによって可読性を強化せよ, 誤解を生む定義からエラーが起きる機会を減らせ.

3.1.2 ファイルの開始近くのコメントの中で重要な変数をドキュメント化することを考えよ.

Consider documenting important variables in comments near the start of the file.

変数が宣言されるところで変数をドキュメント化することは他の言語において標準的な実践である.

MATLAB は変数の宣言を使うことはないので，変数の情報はコメントにおいて与えた方がよい．

```
% pointArray    Points are in rows.

THRESHOLD = 10; % Maximum noise level found.
```

3.2 大域的事項について
Globals

3.2.1 大域定数の使用を最小限にせよ．
Minimize use of global constants.

大域定数を定義するための m-ファイルあるいは mat ファイルを使え．この実践は，どこで定数が定義されるかを明らかにし，不用意な再定義を思いとどまらせる．もし m-ファイルのアクセスのオーバーヘッドが実行速度の問題を生み出すならば，関数ハンドルを使うことを考えよ．

3.2.2 大域変数の使用を最小限にせよ．
Minimize use of global variables.

大域変数の使用よりもむしろ引数の明かなパスの設定によって関数の明瞭性と保守可能性は利益を得る．大域変数の使用法には，より明らかな `persistent` または `getappdata` を用いて置き換えられるものがある．代わりとなる戦略は関数を用いて大域変数を置き換えることである．

3.3 ループ
Loops

3.3.1 ループの結果の変数をループの直前で初期化せよ．
Initialize loop result variables immediately before the loop.

これらの変数を初期化することは，ループの速度を向上させるならば，そしてすべての可能なインデックスに対してループが動作するのでなければ，偽の値を避ける助けとなる．この初期化は事前割り当てとときどき呼ばれる．ループのすぐ前に初期化操作を置くことで変数が初期化されることが容易に分かる．この実践によって，関連のあるコードをみなコピーして，どこかで利用することがより容易にもなる．

```
result = nan(nEntries,1);
for index = 1:nEntries
   result(index) = foo(index);
end
```

初期化のステートメントと `for` 行の双方における引数のために，きちんと命名された変数を用いよ．

3.3.2 ループ中の break の使用を最小限にせよ．
Minimize the use of break in loops.

このキーワードはしばしば不要である．このキーワードの使用によって構造化よりも高い可読性が得られることがはっきりと示せるのであれば，これを使ってもよい．

3.3.3 ループ中の continue の使用を最小限にせよ．
Minimize use of continue in loops.

このキーワードはしばしば不要である．このキーワードの使用によって構造化よりも高い可読性が得られることがはっきりと示せるのであれば，これを使ってもよい．

3.3.4 入れ子のループ中の end 行には識別のためのコメントを付けた方がよい．
The end lines in nested loops can have identifying comments.

長い入れ子のループの `end` 行にコメントを付加することは，どのステートメントがどのループにあるのか，これらの場所でどんなタスクが実行されているのかを明らかにする助けとなる．

3.4 条件について
Conditionals

3.4.1 複雑な条件式を避けよ．代わりに一時的な論理変数を導入せよ．
Avoid complex conditional expressions. Introduce temporary logical variables instead.

式に論理変数を割り当てることによってプログラムは自動的なドキュメンテーションを得る．この構成は読みやすく，かつデバッグも容易となるだろう．

```
if (value>=lowerLimit)&(value<=upperLimit)&~...
    ismember(value,...valueArray)
  :
end
```

は以下によって置き換えられなければならない．

```
isValid = (value >= lowerLimit) &...
  (value <= upperLimit);
isNew = ~ismember(value, valueArray);
if (isValid & isNew)
  :
end
```

3.4.2 *if else* ステートメントの if-部分には通常の事例をおいて，else-部分にまれな事例をおけ．

Put the usual case in the if-part and the unusual in the else-part of an if else statement.

まれな事例が正常な実行経路を不明瞭にすることを抑えることで，この実践は可読性を向上させる．

```
fid = fopen(fileName);
if (fid~=-1)
  :
else
  :
end
```

3.4.3 if 0 という条件式を避けよ．

Avoid the conditional expression if 0.

この使用法が実行の正常経路を不明瞭にはしないことは確実である．コードの実行を一時的に迂回するために，この式の代わりに MATLAB のエディターのブロックコメントの特色を使え．

3.4.4 *switch* ステートメントは *otherwise* の条件を含まなければならない．

A *switch* statement should include the *otherwise* condition.

`otherwise` を忘れることはよくあるエラーであり，予期しない結果につながる．

```
switch (condition)
case ABC
   statements;
case DEF
   statements;
otherwise
   statements;
end
```

3.4.5 条件が式として最も明瞭に書かれるとき *if* を使え．条件が変数として最も明瞭に書かれるとき *switch* を使え．

Use *if* when the condition is most clearly written as an expression. Use *switch* when the condition is most clearly written as a variable.

`if` と `switch` の使用の範囲には重なりがありうる．このガイドラインに従うことは一貫性を与える助けになる．

スイッチ変数は通常はストリングでなければならない．文字ストリングは，この文脈においてよく動作し，それらは番号付きの場合より通常はより分かりやすいものである．

3.5 一般的事項に関して

General

3.5.1 暗号のようなコードを避けよ．

Avoid cryptic code.

あるプログラマーたちの間には，きびきびとしているが不明瞭になってしまう MATLAB コードを書く傾向がある．これはたぶんシェークスピアの一節「簡潔さは言葉の命である」によって刺激をうけたものである：簡潔なコードを書くことはその言語の特徴を追及するための方法となりうる．しかしながら，ほとんどすべての状況において明瞭性が目標となるべきである．MathWorks の Steve Lord は「今から一月後，もし私がこのコードを見たら，それが行おうとしていることを自分で理解できるだろうか？」と書いている．

3.5.2 丸括弧を使え．

Use parentheses.

MATLAB は演算の手続きのための規則を文書化しているが，その詳細を誰が覚えたいと思うだろうか？もし何か疑問があれば，式を明確にするために丸括弧を使え．それらは長くなった論理式のためにとくに助けになる．

3.5.3 式における数の使用を最小限にせよ．

Minimize the use of numbers in expressions.

変更可能であると想定している数は，それに代わって名前のついた定数に通常はすべきである．もし数がそれ自身で明白な意味を持たないならば，それに代わり名前のついた定数を導入することによって可読性が強化される．

数の表記が現れるところをファイル中ですべて見つけて変更することよりも，定数の定義を変更することの方がかなり簡単である．

3.5.4 点の前の桁に数字を入れて小数点を書け．

Write fractional values with a digit before the decimal point.

これはシンタックスに対する数学での慣習を支持している．0.5 は .5 よりずっと読みやすい；整数 5 として読まれることはなさそうである．

以下のものを使え．

```
THRESHOLD = 0.5;
```

以下のものを避けよ．

```
THRESHOLD = .5;
```

3.5.5 浮動小数点の比較には注意せよ.

Use caution with floating point comparisons.

2 進表現は次の例にみられるように面倒なことを引き起こす.

```
shortSide = 3;
longSide = 5;
otherSide = 4;
longSide^2 == (shortSide^2 + otherSide^2)
ans =
     1
```

しかし

```
scaleFactor = 0.01;
(scaleFactor*longSide)^2 ==
((scaleFactor*shortSide)^2 + ...
(scaleFactor*otherSide)^2)
ans =
     0
```

となる.

値の間の差が十分に小さいことを判定することがより良い方法である.

3.5.6 論理式のために自然で率直な形式を使え.

Use the natural, straightforward form for logical expressions.

否定を含む論理式は理解することが難しくなってしまう. 肯定的な式を使うように努力せよ.

以下のものを使え.

```
iSample>=maxSamples;
```

以下のものを避けよ.

```
~(iSample<maxSamples);
```

3.5.7 エラーに備えよ.

Prepare for errors.

一般にエラーは下位のルーチンにおいて把握されなければならない. 解決のためにはより高いレベルのルーチンにおいて訂正されるか回避されなければならない. エラーの状態に対する保護のための有用なツールとなるステートメントは MException を使った `try catch` 構造である.

防御のためのもう一つの書き方は `if` ステートメントにおいて適切に順序づけられた式を使うことである. その結果, エラーの引き金となる式の評価をショートカットして避けることができるようになる.

3.5.8 入力を取得するために使われる関数の中に妥当性の検査を含めよ.

Include validity checking in functions used to acquire input.

妥当ではない入力は実行を停止させるエラーにつながる. 妥当性の検査はエラーに対してのより優れた取り扱いを可能とする. 有効なツールとして `validateattributes` と `inputParser` がある.

3.5.9 可能なときには *eval* の使用を避けよ.

Avoid use of *eval* when possible.

`eval` を含むステートメントは, それに代わるステートメントよりも, 正しく書くことがより難しく, 読むことがより難しく, 実行することがより遅くなる傾向がある. `eval` を使用すると M-Lint による完全な検査を維持できなくなる. `eval` を用いるステートメントは, コマンドから関数へ変更することによって, あるいは構造体のための動的なフィールド参照を `setfield` と `getfield` によって行うことによって, 改良可能である.

3.5.10 可能なときには関数としてコードを書け.

Write code as functions when possible.

関数は, 基盤ワークスペースの部分ではない内部変数を用いて計算をモジュール化する. スクリプトと比べれば関数は入力と出力の変数をより明白にし, よりきれいに, より柔軟に, よりよく設計されるようになる. スクリプトは開発をしているときに重要な役割をはたす. なぜならばスクリプトは, 変数の次元, 型, および値を直接的に見えるようにするからである.

3.5.11 自動化のためのコードを書け.

Write code for automation.

自動化された実行とテストの支援のために `keyboard` と `input` の使用を最小限にせよ.

4 レイアウト, コメントとドキュメンテーション
Layout, Comments and Documentation

4.1 レイアウト

Layout

レイアウトの目的は読み手がコードを理解することを助けることである. 構造をはっきりと示すために字下げはとくに助けになる.

4.1.1 80 カラムに内容をおさめよ.

Keep content within the first 80 columns.

80 カラムは，エディター，端末エミュレーター，プリンターおよびデバッガーのための共通の大きさである．何人かで共有されるファイルはこの制約を守らなければならない．プログラマーの間でファイルを渡すときに不用意な改行が避けられるならば，可読性が向上する．

4.1.2 分かりやすいところで長い行を分けて書け.

Split long lines at graceful places.

推奨した 80 カラムの制限を一つのステートメントが越えるとき行の分離が起きる．

一般に

コンマあるいは空白の後で改行せよ.

演算子の後で改行せよ.

（ MATLAB の）エディターは継続演算子 ... の後で字下げをする．新しい行をその前の行の式が始まるところに配置するために空白を付加することを選択肢としてもよい.

```
totalSum = a + b + c + ...
           d + e;
function (param1, param2, ...
          param3)
setText (['Long line split' ...
         'into two parts.']);
```

4.1.3 3 つか 4 つの空白で字下げせよ.

Indent 3 or 4 spaces.

優れた字下げがプログラムの構造を明瞭にするためのたぶん唯一最良の方法である．1 つの空白の字下げはコードの論理的なレイアウトを強調するためには少なすぎる．2 つの空白の字下げは入れ子のステートメントを 80 カラム以内に収めようとするための改行数を減少させるためにときどき推奨されるが，しかし MATLAB は通常は深く入れ子になることはない．4 つより長い字下げは入れ子のコードを読みにくくさせてしまう．というのは，それによって行を分離しなければならない機会が増えるからである．4 つの字下げは MATLAB エディターにおける現在の初期設定である．MATLAB エディターと一致するように字下げせよ．MATLAB エディターは字下げを行っており，これはコードの構造を明白にし， C++ と Java のために推奨されている実践と一致するものである.

4.1.4 コードの行ごとに一つの実行可能なステートメントを書け.

Write one executable statement per line of code.

この実践は可読性を向上させ，そして実行を早くできる.

4.1.5 単一の短いステートメント *if*, *for* あるいは *while* ステートメントは一行で書いた方がよい.

Short single statement *if*, *for* or *while* statements can be written on one line.

この実践はより簡潔となるが，それは字下げの形式の手がかりがないという欠点を持つ.

```
if(condition), statement; end

while(condition), statement; end

for iTest = 1:nTest, statement; end
```

4.2 空白

White Space

空白はステートメントの個々の要素を目立たせて可読性を強化する.

4.2.1 =, &&, および || の周りを空白で取り囲め.

Surround =, &&, and || by spaces.

この指定の文字の周りに空白を使うことはステートメントの左側と右側を分離するはっきりした視覚的手がかりを提供する.

2 値論理演算子の周りに空白を使うことは複雑な式を明解にできる.

```
simpleSum = firstTerm+secondTerm;
```

4.2.2 ありふれた演算子は空白で取り囲んだ方がよい.

Conventional operators can be surrounded by spaces.

この実践は議論を巻き起こすものである．ある人々はそれが可読性を向上させると信じている．他の人々はそれが式を不必要に長くすると考えている.

```
simpleAverage = (firstTerm + secondTerm) / two;

for index = 1 : nIterations
```

4.2.3 コンマの後に空白一つを続けた方がよい.

Commas can be followed by a space.

これらの空白は可読性を向上させることができる．行が分離されることを避けるために空白を取り除くプログラマーもいる.

```
foo(alpha, beta, gamma)

foo(alpha,beta,gamma)
```

4.2.4 一行中の多重コマンドのためのセミコロンあるいはコンマの後には空白一つを続けよ.

Follow semicolons or commas for multiple commands in one line by a space character.

空白を入れることは可読性を強化する.

```
if (pi>1), disp('Yes'), end
```

4.2.5 キーワードの後に空白一つを続けよ.

Follow keywords by a space.

この実践はキーワードと関数の名前を区別する助けとなる.

4.2.6 一行のブランク行を使ってブロック内のステートメントの論理的なグループ分けを行え.

Separate logical groups of statements within a block by one blank line.

一つの方法は 3 行のブランク行を使うことである. 一つのブロック内の空白行よりも, 空白をより大きくすることで, そのブロックがファイル内において目立つことになる. よりよい方法は %% によって定義される MATLAB エディターのセクション機能を使うことである.

4.2.7 可読性を強化できるところではどこでも配置に気を使え.

Use alignment wherever it enhances readability.

コードの配置は切り裂かれた式を読みやすく, かつ理解しやすくすることができる. このレイアウトはエラーを明らかにする助けとなる.

```
value = (10 * nDimes) + ...
        (5 * nNickels) + ...
        (1 * nPennies);
```

4.3 コメント

Comments

コメントの目的はコードに情報を付加することである. コメントの典型的な使用は, 使用法を説明すること, コードの目標を表現すること, 参考情報を提供すること, 判断を説明すること, 限界を記述すること, 必要な改良について言及することである. 経験から言えば, 後でコメントを付加しようとするよりもコードを書くときと同時にコメントを書いた方がよい.

4.3.1 コメントを読みやすくせよ.

Make the comments easy to read.

記号 % とコメント文の間に一つの空白がなければならない. コメントは大文字で始まり, ピリオドで終了しなければならない.

4.3.2 英語でコメントを書け.

Write comments in English.

国際的な環境では英語が好まれる.

4.3.3 ヘッダーコメント

Header comments

ヘッダーコメントとは m-ファイルの中の最初の連続的なコメントのブロックである. ファイルを使うために必要となる情報を利用者に提供するためにコメントを書け.

通常の利用では関数のためのヘッダーコメントの二つの形式がある. 伝統的な形式は関数の（定義の）行の下にコメントを入れ, たった一つの % の印だけを使う. それはコマンドウィンドウでの使用のためにもともと設計されたものであった.

より現代的な形式は関数の（定義の）行の上にコメントを書くことである. それは MATLAB マークアップをしばしば使っており, そして Help と関数ブラウザーのための HTML ファイルを作成するために設計されている.

4.3.4 ヘッダーコメントにおいて関数の文法を与えよ.

Present the function syntax in header comments.

入力と出力の引数, それらの並びと変形を知ることが利用者には必要となる.

4.3.5 ヘッダーコメントにおいて入力と出力の引数について説明せよ.

Discuss the input and output arguments in the header comments.

入力が特定の単位で表現される必要があるかどうか, あるいは特定の型の配列であるかどうかを利用者は知る必要が出てくる.

```
% completion must be between 0 and 1.
% elapsedTime must be one dimensional.
```

4.3.6 ヘッダーコメントにおいて副作用をすべて記述せよ.

Describe any side effects in the header comments.

副作用は出力変数の割り当てというよりも関数の動作である. よくある例としてプロットの生成がある. これらの副作用の記述はヘッダーコメントに含まれるように

して，ヘルプの印刷に現れるようにしなければならない．

4.3.7 関数名の実際の文字種を用いてコメント中にその名前を書け．

Write the function name in comments using its actual case.

実際の関数名はみな小文字であるにも関わらず，昔のコードのファイルはヘッダーコメントにおける関数名にすべて大文字をしばしば使っていた．カラーではないコマンドウィンドウにおいて表示したときに関数名が目立つようにしたいということが，この実践の意図であったのだろう．

ほとんどのプログラマーはいまや，エディターウィンドウ，あるいは Help あるいは関数ブラウザーにおいてヘルプ情報を見る．すべて大文字というスタイルは，このような状況においては読者の助けになるわけではない．そしてまたミクストケースの関数名がより常識になりつつあり，コメントにおいてすべて大文字を使用することは混乱を招いてしまう．

4.3.8 関数のヘッダーの表示における乱雑さを避けよ．

Avoid clutter in display of the function header.

関数ファイルの始まりの近くのコメントにおいて著作権の行と変更の履歴を含むことは常識である．これらのコメントがヘルプのドキュメンテーションにおいて表示されないようにするために，ヘッダーコメントとこれらのコメントの間に一行の空白を入れなければならない．

4.3.9 行中のコメント

Inline comments

ヘッダーとは異なりコード中のコメントは使用者ではなくプログラマーに向けられている，

4.3.10 コメントはできの悪いコードを正当化できない．

Comments cannot justify poorly written code.

適切な名前の選択と明示的な論理構造を欠くコードをコメントは埋め合わせできない．そのようなコードは書き直さなければならない．Steve McConnell いわく「コードを改良せよ，それからそれをさらに明瞭にするために説明せよ」

4.3.11 コードと一致するコメントを作れ．しかしコードの単なる再記述以上のことをせよ．

Make the comments agree with the code, but do more than just restate the code.

悪いコメント，すなわち使えないコメントは読み手の邪魔をする．N. Schryer いわく「コードとコメントが一致しないならば，そのとき両方がたぶん間違っている」コードが**何を**行うかよりも，**なぜ**あるいは**どのように**行うかについてコメントが言及することの方が通常重要である．

4.3.12 ステートメントへの言及と同様のことを字下げコードはコメントしている．

Indent code comments the same as the statements referred to.

コメントがプログラムのレイアウトを壊さないときにコードはより読みやすくなる．

4.3.13 行の終わりのコメントの使用を最小限に抑えよ．

Minimize use of end of line comments.

行の終わりのコメントの記載は典型的な 80 カラムの行の長さによって制約される．一般にそれらは変数の宣言への添え物としてのみ使われるべきである．

4.3.14 ドキュメンテーションのためのコメントの作成

Commenting for documentation.

ドキュメンテーションの目的は二つのグループに向けられている：コードを走らせたい利用者とコードを読むプログラマー．一般にヘッダーコメントは利用者に向けられており，行中のコメントはプログラマーに向けられている．

4.3.15 パブリッシュのためのコメント

Comments for publishing

MATLAB は，HTML, XML, latex, doc, ppt, および pdf のフォーマットによるパブリッシュのための特別なコメントづくりを支援している．HTML でのパブリッシュは機能的なドキュメンテ—ションのためにとても有用となりうる．doc あるいは pdf によるスクリプトのパブリッシュは基礎的な報告書を作り出すことができる．

MATLAB は，関数のために利用者が書いた HTML の参照ページを Help ブラウザー中に表示することができる．単純な MATLAB マークアップを使ったパブリッシュの特徴により，関数 m-ファイルからこれらのページを作り出すことができる．

5 ファイルと組織
Files and Organization

ファイル間およびファイル内の双方においてコードを構造化することは構造を理解可能にさせるための根幹である．考え抜かれた分割と順序付けはコードの価値を高める．

5.1 M ファイル

M Files

5.1.1 モジュール化せよ.

Modularize.

大きなプログラムを書くための最良の方法はよく設計された小さい部品（通常は関数）からプログラムを組み立てることである．この方法は，コードが行っていることを知るために読まなければならないテキストの量を減少させることによって，可読性，理解とテストを強化する．

エディタースクリーンの二つ分よりも長いコードは分割の候補である．関係する情報を同一のエディタースクリーン上にいっしょに収めておくことで，問題の形式を確実に知り，問題を正しく修正できる．よく設計された小さい関数は他の応用においても利用が進むことになる．

5.1.2 相互干渉を明らかにせよ.

Make interaction clear.

関数は入力と出力の引数および大域変数を通して他のコードと干渉する．大域変数の使用よりも引数の使用の方がほぼ常に明快である．構造体が入力あるは出力の引数の長いリストを避けるために使われる．

インターフェースの標準を作ることが関数の使用に対してより親しみやすく首尾一貫した経験をもたらす．これは適切な使用をより促進する．

5.2 分割

Partitioning

すべてのサブ関数と多くの関数は一つのことを本当に的確に行なわなければならない．すべての関数はものごととを覆ってくれる．

5.2.1 既存の関数を使え.

Use existing functions.

正しく，読むことができ，かつ合理的柔軟性のある関数を開発することは重要な課題である．必要とされる機能の一部かあるいはすべてを実現する既存の関数を見つけることが速く確実な方法であろう．

5.2.2 どんなコードのブロックでも複数の m-ファイル中に現れるならば，関数として書くことを検討しなければならない.

Any block of code appearing in more than one m-file should be considered for writing as a function.

たった一つのファイルにコードが現れるなら変更を管理することはとても容易である．コピー・アンド・ペーストよりもむしろカット・アンド・ペーストを使うことを試みよ．

5.2.3 関数の引数のために構造体を使え.

Use structures for function arguments.

関数の使いやすさは引数の数が多くなるにしたがって減少する．選択可能な引数があるときに，とくにそうである．引数のリストが 3 つを越えるときには常に構造体を使うことを考えよ．

既存の外部コードと互換性のある関数に入る値の数あるいはそのような関数からの値の数の変更を構造体は可能にすることができる．

構造体は引数を固定的な順序におく必要性を減少させる．選択可能な値にとっては変数の順序づけられた長いリストを作るよりも構造体の方がより優美である．

5.2.4 関数の中に何らかの一般性を与えよ.

Provide some generality in functions.

関数は通常十分に柔軟性をもち，入力のスカラー，ベクトル，および少なくとも 2 次元の配列を受け入れなければならない．複数の表現を通常持っている入力引数を伴う関数は，そのすべて表現に対して動作しなければならない．たとえば画像処理の関数はすくなくとも unit8 と 2 倍長変数で動作しなければならない．

5.2.5 サブ関数

Subfunctions

ある関数がたった一つの他の関数によって使われるとき，それは他の関数のファイル中でサブ関数として組み込まれなければならない．これによってコードを理解することおよび保守することがより容易になる．

MATLAB はサブ関数の m-ファイルの外からサブ関数を呼び出すことを許している．これは一般には悪い考え方である．

5.2.6 テストスクリプト

Test scripts

すべての関数のためにテストスクリプトを書け．この実践は関数の初版の質と修正版の可読性を向上させることだろう．テストが極端に困難な関数はみな書くこともたぶん極端に困難であるということを考えてほしい．Boris Beizer いわく「テストを行う活動ではなくテストを設計する活動の方がよく知られた最良の虫よけ装置の一つである」

5.3 入力と出力
Input and Output

5.3.1 入力と出力のモジュールを作れ.
Make input and output modules.

出力に対する要求は通告なしに変更されることがある．入力の形式と内容も変更はまぬがれず，それはしばしば面倒である．それらを取り扱うコードを局所化することは保守可能性を向上させる.

単一の関数において入力あるいは出力のコードと計算を混ぜこぜにすることを避けよ．ただし前処理は除く．目的が混ぜこぜの関数は再利用ができそうもない.

5.3.2 簡単に利用できるように出力を形式化せよ.
Format output for easy use.

もし人間が出力を読むことがほとんどであれば，それを自己記述的にして容易に読めるようにせよ.

もし人間よりもソフトウェアが出力を読むことがほとんどであれば，その構文解析が容易にできるようにせよ.

もし双方が重要であれば，出力を容易に構文解析できようにし，そして 書式整形関数を書いて人が読むことができるバージョンを生成せよ.

5.3.3 ファイルの読み込みのための feof を使え.
Use feof for reading files.

行あるいはデータの数え上げに依存することはファイル終了エラーか不完全入力に容易に結びついてしまう.

5.3.4 ツールボックス
Toolboxes

ツールボックスにおいてある一般性を持つように m-ファイルを組織せよ．シャドウイングが起きないように関数名を確認せよ．MATLAB のパスにツールボックスの位置を加えよ.

プロジェクトと汎用のツールボックスの双方を持つことは典型的に有用である.

6 スタイルについての引用
Style quotes

- Martin Fowler: “Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”
  「どんな愚かものでも計算機が理解できるコードを書くことができる．よいプログラマーは人間が理解できるコードを書く」
- “In matters of style, swim with the current; in matters of principle, stand like a rock.” Thomas Jefferson:
  「様式のようなことに関しては、ときの流れに乗れ；原則に関しては、岩のように立て」
- “You got to know the rules before you can break’em. Otherwise it’s no fun.” Sonny Crockett in Miami Vice
  「おまえがルールを破ってしまう前に、おまえにはルールを知ってもらう必要があった．そうでなければ面白みがない」
- Patrick Raume, “A rose by any other name confuses the issue.”
  「薔薇を別な名前で呼べば議論が混乱してしまう」
- Plato, “Nothing has its name by nature, but only by usage and custom.”
  どんなものでも必然的にその名前を持ってはいないが、使用法と習慣だけで名前を持っている.
- Unknown: “All general statements are false.”
  「一般的な主張はすべて偽りである」
- Try to avoid the situation described by the Captain in Cool Hand Luke, “What we’ve got here is failure to communicate.”
  Cool Hand Luke（1960 年代の小説、映画「暴力脱獄」）によって描かれた状況を避けようとせよ.
  「我々がここに至ったことは打合わせの誤りである」
- Kreitzberg and Shneiderman: “Programming can be fun, so can cryptography; however they should not be combined.”
  「プログラミングは楽しいものであり、暗号解読も同様である；しかし、その二つを結びつけてはならない」
- Jay Rodenberry, “Space . . . The final frontier.”
  「宇宙（スペース）. . . 最後の開拓地」
- Napoleon Hill, “First comes thought; then organization of that thought into ideas and plans; then transformation of those plans into reality.”
  “Change is inevitable . . . except from vending machines.”
  「最初に考えが現れる．それから、その考えを着想と計画によくまとめること；それから、その計画を実現に向けて変えていくこと」
  「変更 (change) は避けられない · · · 自動販売機は例外だが」[*7]

以下は訳者によるスタイルについての引用である.

[*7] 訳者注：change に 小銭・つり銭の意味もある.

- 新明解国語辞典: ぶんたい【文体】(1) その・時代（ジャンル）の文章に特有な表現様式．(2) その作者が素材をいかに形象化するかの方法．〔狭義では，表現技術の上に見られる特徴を指す〕
- 吉田 秀和（音楽評論家）: 表現の真実のために破ってはならぬ法則はない．（名曲 300 選）
- スナフキン（ムーミンの登場人物．自由と孤独，音楽を愛する旅人）: あんまり誰かを崇拝するということは，自分の自由を失うことなんだ．
  大切なのは，自分のしたいことを自分で知ってるってことだよ．
  あんまり大袈裟に考えすぎない様にしろよ．何でも大きくしすぎちゃ駄目だぜ．（Naver まとめサイト）
- 正岡 子規（俳人）: 文章は簡単ならざるべからず．最（もっとも）簡単なる文章が最（もっとも）面白き者なり．（筆まかせ 抄）
  美しき花もその名を知らずして文にも書きがたきはいと口惜し．（墨汁一滴）
- ベノワ・マンデルブロー（数学者）: ラテン語に「名づけることは知ることである」(Noemen est numen) ということわざがある．（フラクタル幾何学）
- ドナルド・クヌース（計算機科学者）: コンピュータプログラムを書くことは楽しい，またうまく書いてあるコンピュータプログラムを読むことも楽しい．人生の最大の喜びのひとつは，他の人たちに読んでもらっても，そして自分自身で読んでも，楽しいようなコンピュータプログラムを作り出すことである．
  演奏を目的として入り組んだパターンを作り出すことから，最初のうちはコンピュータプログラミングは音楽の作曲にかなり近い仕事だと思っていた．しかし最近は，それよりずっと適切なアナロジーがあることに気付いた．プログラミングを，読むことを目的とした文学作品を作り出す過程だと考えると，ぴったりする．
  プログラムとその人間との対話がより文学に近づけば，コンピュータプログラミングに関する，信頼性，携帯性，学習しやすさ，保守性，性能がよいことなど，その大きな問題はすべて改善することができる．（文芸的プログラミング）
- 三島 由紀夫（小説家）: あて名をまちがいなく書くことです．これをまちがえたら，ていねいな言葉を千万言並べても，帳消しになってしまいます．（三島由紀夫レター教室）
- 谷崎 潤一郎（小説家）: 文章に実用的と芸術的との区別はないと思います．
  最も実用的にかくということが，すなわち芸術的の手腕を要するところなので，これがなかなか容易に出来る業ではないのであります．（文章読本）
- 川端 康成（小説家）: 文章は，人と共に変り，時と共に移る．一つが消えれば，一つがあらわれる．文体の古びの方の早さは思いの外である．つねに新しい文章を知ることは，それ自身小説の秘密をしることだ．（新文章読本）
- 丸谷 才一（小説家）: ちょつと気取つてかくといふこと，あるいは，気取らないふりをして気取るということ，それこそは文体の核心にほかならない．（文章読本）
- 木下 是雄（物理学者）: 必要なことは漏れなく記述し，必要でないことは一つも書かないのが仕事の文書を書くときの第一原則である．
  私はいつも＜適当な白さ＞で書くことを最大の要件とするのである．（理科系の作文技術）
- 本多 勝一（朝日新聞記者）: 目的はただひとつ，読む側にとってわかりやすい文章を書くこと，これだけである．
  文体とかリズムのことを考慮すると，文章はあたかも精密機械や人体組織のようになってくる．完成された文体は，へたにいじると故障してしまうし，切れば血が出もしよう．（日本語の作文技術）

7 参考文献
References

1. The Elements of MATLABStyle, Richard Johnson
2. Clean Code, Robert Martin
3. Code Complete, Steve McConnell - Microsoft Press
4. The Elements of Java Style, Allan Vermeulen et al.
5. MATLAB: A Practical Introduction, Stormy Attaway
6. The Practice of Programming, Brian Kernighan and Rob Pike
7. The Pragmatic Programmer, Andrew Hunt, David Thomas and Ward Cunningham
8. Programming Style, Wikipedia

8 Copyright