3.

Naming

The purpose of a software naming convention is to help the people who read and write the code understand the program. Understanding the code makes it easier to use, write, or modify correctly. The goal in naming is to use names that are easy to recognize and associate with their meaning and role.

The basic content idea is to use words and character sequences that help connect a name with the entity that it represents. It is a great idea to use names that are intuitive, but developers have surprisingly divergent ideas on which names are intuitive. A more realistic objective is to use names that help the reader understand the code.

The basic visual style idea is to use different visual styles for different identifier types. This makes it easier to recognize the various elements of the language. Clearly, some consistency and differentiation is important. There is a balance in differentiation because having a large number of styles makes for too much to remember. In any case, only a few variations of naming style are available in MATLAB, so the language elements have to be grouped into a smaller number of styles. There is controversy is over how best to do it.

Establishing a naming convention for a group of developers is important, but the process can become ridiculously contentious. There is no naming convention that will please everyone. Following a convention consistently is more important than the details of the convention. Inconsistent practices confuse readers and raise legitimate concerns about code quality.

https://doi.org/10.1017/CBO9780511842290.005

The choices in this guide are based on the belief that similar elements should look similar, and that common usage in the MATLAB and Java communities should be respected. This section describes a commonly used convention that is familiar to many programmers of MATLAB and other languages.

General

The names of identifiers trigger associations. Good identifier names aid comprehension of both the thing named and the associated code structure. Good names are *very* helpful to both the reader and the programmer. However, coming up with good names is not easy. Keep trying until you get them right. If you change the definition or usage of a variable or function so that its name becomes misleading or wrong, then rename it.

To reduce confusion, identifier names should look dissimilar without intensive inspection. Names should be easy to remember and easy to associate with a meaning. They should also be responsive to their environment (not too short or long). Trade-offs are likely to be required.

28. Use Meaningful Names

Use identifier names that convey a meaning consistent with the problem domain or typical interpretation. MATLAB can cope with

```
w = h * n
```

but the reader will find it much easier to follow

```
wage = hourlyRate * nHours
```

Try for descriptive but not overly verbose names.

Replace

```
s = computedYears - numberOfStudents;
```

with

```
salary, yearsToRetirement, nStudents
```

Meaningful names make the identifiers easier to understand and remember. Both attributes make the code easier to read, write, and use.

29. Use Familiar Names

If your domain convention is to use "client," do not use "customer" or "consumer" as a variable name. Put the elements of compound names in familiar order. Replace

```
lengthArm
```

with

```
armLength
```

If the software is targeted for a knowledge domain or a user group, then use names consistent with standard practice in that domain or by that group.

Replace

```
imageRegionForAnalysis
```

with

```
roi or regionOfInterest
```

30. Use Consistent Names

Simplify the reader's life by not mixing similar names such as

```
customer, customerData, customerInfo
```

for the same or related values. Organize the data so that it makes sense to use one name or multiple distinctive names.

Similarly, avoid mixing similar action terms such as

```
fetchaccount, retreivestore, returncustomer,
getinventory
```

for the same general functionality.

31. Avoid Excessively Long Names

Names that are too long can be difficult to scan and use. Long names are particularly problematic in visually parsing expressions. Because very long names decrease the readability of expressions and statements, the best names are those just long enough to be descriptive. In general, names longer than fifteen characters are not recommended.

Replace

```
applyThresholdToDataArray
```

with

```
applyThreshold
```

The goal in choosing names is not necessarily to give everything complete and proper names, but rather to give them comprehensible and trackable names.

32. Avoid Cryptic Abbreviations

Make identifiers easy to read by avoiding unnecessary abbreviation or shortening. The use of abbreviations usually makes names less pronounceable and more difficult to remember. Programmers are used to some shortening in compound words, but do not overdo it.

Using whole words reduces ambiguity and helps make the code self-documenting. Because they are often not unique, abbreviations are prone to interpretation and typing errors.

Use

```
computeArrivalTime
```

Avoid

```
comparr
```

In this case, it is unclear whether the intent is compute or compare.

Common MATLAB usage justifies some abbreviations such as col for column, dim for dimension, and h for handle. Domain-specific abbreviations are also acceptable. For example,

`shuttle` for `spaceShuttle`
`tech` for `technical`
`meds` for `medications`

Some abbreviations might benefit from a defining comment near their first appearance.

33. Treat Familiar Acronyms as Words

Domain-specific phrases that are more naturally known through acronyms should be used as acronyms rather than compound words. For example,

```
html, cpu, cm
```

Use the same convention for capitalizing acronyms as for words.

Use

```
htmlVersion, pdfParser
```

not

```
HTMLVersion, PDFParser
```

or

```
hTMLVersion, pDFParser
```

34. Avoid Names that Differ Only by Capitalization

Names that differ only by capitalization are easy to confuse, and this is an almost certain way to introduce defects. Consistent capitalization is one of the hallmarks of good style.

Do not use `FlightTime` or `Flighttime` in the same module in which you use `flightTime`.

Do not use `x` and `X` in the same module. Authors of books and papers sometimes use names such as `x` and `X` together for related but distinct entities. Programmers work in a different is context because most readers find it easier to differentiate similar names on the printed page than in code displayed on an editor screen.

35. Avoid Names that Differ Only by One Letter

Choose identifier names that will not be confused. Names within a file should differ enough to be readily distinguished. Names that are almost the same make the code difficult to scan. They also increase the likelihood of unintentional use.

Code written by novices often includes names like `a1`, `a2`, `a3` or `a`, `aa`, `aaa`. This is usually evidence that the names are not meaningful. Even names that carry some meaning can be problematic if they only differ by one number or letter. The following published code has an error in the third line that is disguised by poor variable naming:

```
total = price * qty;
   total2 = total - discount;
   total2 = total * (1 + taxRate);
```

The intent was

```
subtotal = price * quantity;
discountedTotal = subtotal - discount;
billableTotal = discountedTotal * (1 + taxRate);
```

36. Avoid Names with Hard-to-Read Character Sequences

Avoid names containing sequences such as `lll` or `OO` because they can be difficult to recognize or spell consistently. Also avoid using lowercase `l` or uppercase `O` as a single-letter name.

37. Make Names Pronounceable When You Can

Names that are pronounceable are easier to read and remember. The audio cues can make names easier to hold in short-term memory. Pronounceable names improve communication with other programmers and also reduce the likelihood of typographic errors.

38. Write Names in English

The MATLAB product is written in English, and English is the preferred language for international development.

Variables and Parameters

Variable names should be easily remembered by the programmer and have a suggestive value for readers; that is, they should help us recognize the meanings of the variables. They should also be consistent with the names of similar variables.

39. Avoid Ambiguous or Vague Names

A good name distinguishes one variable from another. A variable name that has to be deciphered should be changed. Names such as `temp1` and `vec` convey little meaning. If used, they should be limited to very small scope.

Be selective in the use of numbers at the ends of names. The appearance of numbers at the ends of variable names often indicates poor naming. Replace

```
length1, length2
```

with

```
lengthStrut, lengthSpring
```

or better

```
strutLength, springLength
```


https://doi.org/10.1017/CBO9780511842290.005

There is at least one good use for numbers at the ends of names. They are appropriate for coefficients of equations:

```
a1*x + a2*x.^2
```

40. Name According to Meaning, Not Type

The type or class of a variable is available in the Workspace browser. Including type or class in a variable name creates a headache if the type or class is changed.

Use

```
Customer, step
```

not

```
customer_structure, int_step
```

41. Use Lowercase for Simple Variable Names

This is common practice in the C++, Java, and Windows development communities. For example, use

```
linearity, delta
```

Very short variable names can be in uppercase if they are uppercase in conventional usage and if they are unlikely to become parts of compound variable names. Examples include `V` and `R` in electrical engineering and `Q` in signal processing.

42. Use lowerCamelCase for Compound Variable Names

The lowerCamelCase style starts each word in a compound name with an uppercase letter, except the first word. The use of capital letters makes it easier to recognize the individual words in the variable name. This is common practice in the C++, Java, and Windows development communities. Starting variable names with uppercase is usually reserved for objects, types, or structures in other languages. Use

```
credibleThreat, qualityOfLife
```

Some programmers prefer to use underscore to separate parts of a compound variable name. This practice is common in Unix. Although readable, this usage is generally unexpected and not common for variable names in other languages. Another issue with using underscore in variable names used in graph titles, labels, and legends is that the Tex interpreter in MATLAB will read underscore as a switch to subscript, so you will need to apply the parameter/value pair `'interpreter', 'none'` for each text string.

43. Use Meaningful Names for Variables with a Large Scope

In practice, most variables should have meaningful names. A descriptive or meaningful name is especially important when a variable is used in code locations that are far apart. This usage eases the reader's burden of remembering what quantity a variable name represents. Names that are too short are a common problem that obscures meaning; that is, it is easier to understand and remember the meaning of `salary` than `s`.

44. Limit Use of Very Short Names to Variables with a Small Scope

The use of short names should be reserved for conditions in which they clarify the structure of the statements or are consistent with intended generality. In a general-purpose function, it may be appropriate to use short generic variable names such as `x`, `y`, `z`, and `t`.

Scratch variables used for temporary storage or indices can have short names because they do not need to be remembered outside local scope. Variable names in different local scopes also do not need to be unique. A programmer reading such a variable's name should be able to assume that its value is not used outside a few lines of nearby code.

https://doi.org/10.1017/CBO9780511842290.005

45. Be Consistent With i and j

The letters `i` and `j` have long been used as both imaginary numbers and indices or loop counters. There is an inherent conflict in usage with no solution that will delight everyone.

Many code examples in MATLAB and other languages use `i` and `j` as a loop counters. Those who favor this usage can establish a different variable for imaginary numbers, such as `I`, `J`, or `jay`, or use an expression of the form `1i`, which The MathWorks recommends for speed and robustness.

Those who favor saving `i` or `j` for the imaginary number can use different loop counter variable names such as `iSample` or `I`. Avoid using both `j` and `J` in the same scope because they look similar and thus can be easily confused.

46. Use the Prefix `n` *for Variables Representing the Number of Entities*

This notation is taken from mathematics, where it is an established convention for indicating the number of items. Use

```
nFiles, nSegments
```

rather than

```
numFiles, NumberOfSegments
```

A common MATLAB-specific variation, based on common matrix notation, is the use of `m` rather than `n` as a prefix for number of rows, as in `mRows`.

47. Follow a Consistent Convention on Pluralization

Avoid having two variables with names differing only by a final letter. Be especially careful to avoid unintended mixing of the singular and plural names. Some programmers make all variable names either singular or plural, but others find

https://doi.org/10.1017/CBO9780511842290.005

that this can be awkward. The recommended practice is to use a prefix like `this` for the singular variable, together with a plural suffix for collections or arrays. For example, the singular `thisPoint` with the plural points, as in

```
thisPoint = points(iPoint,:);
```

A less common usage for the plural is to append a suffix such as `Array`. For example, the singular `point` with the plural `pointArray`, as in

```
point = pointArray(iPoint,:);
```

48. Use the Prefix `this` *for the Current Variable*

When referring to a single member of a collection, use `this` rather than `the`:

```
thisPage = pages(iPage);
```

49. Use the Suffix `No` *or Prefix* `i` *for Variables Representing a Single Entity Number*

The No notation is taken from mathematics, where it is an established convention for indicating an entity number.

Replace

```
tableNumber, employeeNumber
```

with

```
tableNo, employeeNo
```

The `i` prefix effectively makes the variable a named iterator, which is convenient for indexing. Use

```
iTable, iEmployee
```

as in

```
thisEmployee = employees(iEmployee);
```

Other suffixes that some programmers use are `Nbr` and `Num`. The most important consideration in this guideline is to distinguish clearly between a single entity number and the number of entities.

50. Prefix Iterator Variables with `i`, `j`, `k`, *etc.*

The notation is taken from mathematics, where it is an established convention for indicating iterators:

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

Many examples of `for` loops use the variable name `i` as a loop index. This practice may be suboptimal, particularly if the loop is long, because it misses an opportunity to reinforce the meaning of the loop index.

For nested loops, the iterator variables should usually be in alphabetical order. Alternatively, some mathematically oriented programmers use a variable name starting with `i` for rows and `j` for columns, independent of their position in nested loops.

Especially for nested loops, suggestive iterator variable names are helpful:

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

51. Embed `is`, `has`, *etc., in Boolean Variable Names*

Use lowerCamelCase and avoid using the word `is` as a prefix.

Replace variable names that start with `is`

```
isavalidrange, isMissingData
```

with

```
rangeIsValid, dataIsMissing
```

In some cases, other words are clearer:

```
borrowerCanQualify, lenderHasMoney
```

This convention avoids confusion with or shadowing of Boolean function names, which usually start with `is` in MATLAB software.

52. Avoid Negated Boolean Variable Names

A problem arises when such a name is used in conjunction with the logical negation operator because this usage results in a double negative. It is not immediately apparent what is meant by names such as

```
~barIsNotFound
```

Replace

```
barIsNotFound = true;
```

with

```
barIsFound = false;
```

so that you can use

```
~barIsFound
```

53. Use the Expected Logical Names and Values

A true or valid condition is usually associated with a positive integer or a logical true; a false or invalid condition is usually associated with zero or a logical false. To avoid violating expectations, replace the misleading and error-prone usage

```
valid = 0;
```

with

```
valid = true;
```

MATLAB associates any nonzero number, even a negative one, with a true condition. It is generally poor practice to rely on this somewhat confusing behavior.

54. Avoid Using a Keyword or Special Value Name for a Variable Name

Reserved words have special meaning and can only be used in specific ways. MATLAB can produce cryptic error messages or strange results if any of its reserved words or built-in special values are redefined. M-Lint will usually catch an attempt to redefine a keyword, but not if this is done inside an `eval` statement. Reserved words are listed by the command `iskeyword`. Special values are listed in the MATLAB release documentation.

Also avoid using a variable name that differs from a keyword or special value only by capitalization or a single letter. The code may work, but it can be difficult to read.

55. Avoid Hungarian Notation

A Hungarian variable name typically involves one or two prefixes, a name root, and a qualifier suffix. These names can be ugly, particularly when they are strings of contractions or abbreviations. A bigger problem occurs if a prefix, as is sometimes suggested, encodes data type. If the type needs to be changed, then all incidences of the variable name need to be changed.

Use

```
thetaDegrees
```

Avoid

```
uint8thetaDegrees
```

Because the Workspace browser lists type, Hungarian notation no longer adds value.

A related MATLAB-specific practice is the use of `vec` as a prefix or suffix. This practice can be problematic if use of the variable changes to include, for example, two-dimensional arrays.

56. Avoid Variable Names that Shadow Functions

There are several names of functions in the MATLAB product that seem to be tempting to use as variable names. Such usage in scripts will shadow the functions and can lead to errors. Using a variable and a function with the same name inside a function will probably cause an error.

Some standard function names that have appeared in code examples as variables are

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

Using a well-known function name as a variable name also reduces readability. If you want to use a standard function name such as `length` in a variable name, then you can add a qualifier, such as a units suffix, or a noun or adjective prefix:

```
lengthCm, armLength, thisLength
```

57. Avoid Reusing a Variable for Different Contents

When a variable is reused, its purpose is unlikely to be clear from its name. Reusing a temporary variable in multiple places for different contents can make reworking the code difficult. Reuse variables only if memory is a constraint. If you change the meaning of a variable, then also change its name.

Similarly, avoid giving two interpretations to a single variable, such as a hidden meaning as well as a normal meaning. It

would be poor practice to use the variable `pageCount` for a page count, except when it is negative, indicating that it is an error flag.

58. Consider a Unit Suffix for Names of Dimensioned Quantities

Using a single set of units for a project is an attractive idea that is often not implemented completely. Adding unit suffixes helps avoid the almost inevitable unintended mixed-unit expressions or unintended computed results. Do not use an ambiguous single-letter suffix. Replace

```
angleR, angleD
```

with

```
angleRadians, angleDegrees
```

If you do not use a unit suffix, then consider including the units in a comment to reduce possible confusion.

Constants

The MATLAB language does not have true constants (at least outside objects). We use naming conventions to provide a visual cue that a variable is being treated as a constant. The main goal in this case is to try to avoid unintentional redefinition of the constant. The best naming and usage practices can depend on the scope of the constant.

59. Use All Uppercase for Constant Names with Local Scope

Constants specific to a single m-file are usually defined in the code and written in uppercase. If the constant has a compound name, then use an underscore as a separator. This is common

practice in the C++ and Java development communities, when the constant is only used within a file:

```
MAX_ITERATIONS, CODE_RED
```

Never use sequential underscore characters because they are too difficult to read correctly.

Lowercase names can be used in domain-specific applications if the most common representation of the constant is in lowercase. For example, you can use `c` for the speed of light or `h` for the Planck constant.

60. Use Function Names for Constants Defined by Functions

Universal or natural constants such as pi are often defined by a function in MATLAB and written in lowercase. Each constant is the output of a function with the same name as the constant. This practice makes it easy to avoid most of the problems with global constants. It is also graceful in expressions.

```
2*pi
```

61. Use Meaningful Names for Constants

As with variable names, the goal is to make it easier for the reader to remember what the name means. Name constants based on significance, not value.

Replace

```
TEN = 10;
```

with

```
MAX_ITERATIONS = 10;
```

Very short constant names can be used if they are conventional usage and unlikely to be confusing to a reader. An example is `E` for Young's modulus.

62. Define Related Constants Based on the Relation

Defining related constants independently can lead to inconvenient problems with precision.

Replace

```
TWO_PI = 6.283185
```

with

```
TWO_PI = 2*pi;
```

63. Consider Using a Category Prefix

You can prefix the names of constants with their common category. This gives additional information on which constants belong together and what concept the constants represent:

```
CODE_RED, CODE_GREEN, CODE_BLUE
```

Structures and Cell Arrays

64. Use UpperCamelCase for Structure Names

The UpperCamelCase style starts each word in a compound name with an uppercase letter, including the first word. The use of capital letters makes it easier to recognize the individual words in the variable name. This usage is consistent with Java and C++ practice, and it helps distinguish between structures and other arrays:

```
Setup.comment = 'This is a test.';
```

65. Do Not Include the Name of the Structure in a Fieldname

The name of the structure is implicit. Repetition is superfluous in use.

Replace

`Segment.segmentWidth`

or

`Segment.widthSegment`

with

`Segment.width`

66. Use Fieldnames that Follow the Naming Convention for Variables

Structures are convenient for passing variables into and out of functions. Using fieldnames that are the same as variable names enhances readability and reduces the likelihood of typos:

```
Data.unit = unit;
function Result = convert(Data)
      unit = Data.unit;
      :
end
```

is easier to write correctly than using a capitalized fieldname:

```
Data.Unit = unit;
function Result = convert(Data)
      unit = Data.Unit;
      :
end
```

Using fieldnames that are the same as variable names also enables automated packing and unpacking of the structures.

67. Name Cell Arrays Following the Style for Variables

The names of cell arrays should follow the convention for simple variables and arrays. Remember that the goal of different naming styles is to help the reader understand the code.

Because cell arrays use numbered cells, they are more similar to arrays than structures:

```
greetings{1} = 'hello';
```

Functions

Functions are one of the most important and widespread components of MATLAB code. Making their names easy to scan and understand is critical to readability. Try to use function names that succinctly convey what the functions do and suggest how to use them.

A good function name abstracts the details of the function in a way that enhances the readability of a calling function's code without being misleading or confusing. The selection of a useful name depends on what the function does and what (if anything) it returns.

68. Give Functions Meaningful Names

The purpose of a well-named function or method can often be determined just from its name. There is an unfortunate MATLAB tradition of using function names so short that they are cryptic, possibly due to the former DOS eight-character limit for filenames. This concern is no longer relevant and the tradition should usually be avoided to improve readability.

Replace

```
compwid
```

with

```
computehalfwidth
```

or

```
computeHalfWidth
```

An exception is the use of abbreviations or acronyms widely used in mathematics or in the problem domain:

```
max,  gcd
```

Functions with such shortened names should have the complete words in header comments for clarity and to support Help browser searches.

69. Name Functions for What They Do

Functions usually perform an action. Naming the function after this action increases readability, making it clear what the function should (and possibly should not) do. Also, this usage can make it easier to keep the code clean of unintended side effects. Typically, function names should start with a verb:

```
plot, reviseforecast
```

You can also name functions for their output. This practice is appropriate if the name would otherwise begin with a common computing verb such as `compute` or `find`. Naming a mathematical or statistical function for its output is common practice in MATLAB code:

```
mean, standarderror, localmaxima
```

70. Follow a Case Convention for Function Names

Using lowercase for function names is standard practice by The MathWorks and most MATLAB authors. It works well for single-word names and reasonably well for short compound names. This practice also helps distinguish function names from lowerCamelCase variable names or UpperCamelCase object names. Using all lowercase also avoids potential filename problems in mixed operating system environments:

```
removebias, adjustbins
```

Using lowerCamelCase for compound function names is the usual practice in other modern languages. It is also becoming

more common in code written by The MathWorks and some MATLAB authors. This practice is especially popular for functions that are class methods. Function names often begin with a verb, which helps distinguish them from variable names. For longer compound names, lowerCamelCase is more readable than all lowercase:

```
removeVaryingBias, adjustHistogramBins
```

These are the preferred conventions, and their use can be mixed.

Two other function name practices are sometimes used. Some programmers use UpperCamelCase. This usage is nonstandard and can cause confusion with class and object naming. A few programmers use underscores in compound function names. This practice is nonstandard. Because it is unexpected, it can actually be more difficult to scan than lowerCamelCase. The underscores are also invisible in the hyperlinks used in reference pages and `help` output. These two practices should not be used, and, in particular, they should not be mixed with the preferred conventions.

71. Reserve the Prefixes `get/set` *for Accessing an Object Property*

This is the general practice in code written by The MathWorks and common practice in C++ and Java development. An occasional exception is the use of `set` for logical set operations:

```
getobject, setappdata
```

72. Use Expected Verbs in Expected Ways

Consistent use of verbs enhances readability and gives the reader an immediate clue about the task of the function. Consider using `compute` when something is calculated:

```
computespreads
```

The use of `comp` as a prefix should be avoided because it can be confused with compare. Consider using the word `find` for a search or lookup operation. Consistent use of the term enhances readability and it is a good substitute for the overused and possibly misleading prefix `get`:

```
findOldestRecord,  findTallestMan
```

Consider the prefix `initialize` when a variable is established. Avoid the abbreviation `init` because it could represent either initialize or initial:

```
initializeState
```

73. Use the Prefix `is` *for Boolean Functions*

This is common practice in MATLAB code as well as in C++ and Java. Replace

```
checkforoverpriced
completion
```

with

```
isoverpriced
iscomplete
```

There are a few alternatives to the `is` prefix that fit better in some situations. These include the `has` or `can` prefixes:

```
haslicense
canevaluate
```

A Boolean function can be visually distinguished from a Boolean variable in that the `is` term is a prefix for functions, but it is embedded for variables. Many Boolean functions are available in the MATLAB product. To avoid shadowing them, search the documentation for `is*`.

Avoid using the word `status` in a Boolean function name because it can easily be ambiguous.

Replace

```
licensestatus
```

with

```
haslicense
```

74. Use Complement Prefixes in Compound Names for Complement Operations

Reduce readability challenges by taking advantage of symmetry. Use prefixes such as

```
get/set, add/subtract, create/destroy, start/stop,
insert/delete, increment/decrement, begin/end,
open/close, show/hide, suspend/resume
```

75. Be Selective in the Use of Numbers at the Ends of Names

Do not use a number at the end of a function name to indicate revision. This would require continual modification of calling function code with each function name revision. Such usage would also raise the issue for the reader of whether older versions are of value. Avoid

```
foo1, foo2, foo3
```

Limit numbers at the end of function names to current functions that have different arguments, especially when the functions work for differing dimensionality. For example,

```
interp, interp2, interp3
```

76. Use Numbers Inside Function Names Only for Common Conventions

It is common practice to use the numeral `2` in place of `to`. The use of other numbers is unusual:

```
str2mat, struct2cell
```

It can be a better practice to replace functions that would use this naming convention with functions that take advantage of polymorphism. For example,

`daily2monthly, yearly2monthly`

might be better as a single function

`tomonthly`

77. Avoid Unintentional Shadowing

In general, function names should be unique. Shadowing (having two or more functions with the same name) increases the possibility of unexpected behavior or error. Some of these shadowing names that have unfortunately been used in MATLAB code examples are

`angle, contrast, length, power, rank, type`

Names can be checked for shadowing using the commands `which -all`, or `exist`.

If you do choose to use an existing function name for your function, then put it in a private folder so that execution will not depend on the order of folders in the MATLAB path.

Classes

The use of MATLAB and Java objects is becoming more common in MATLAB code. Follow the Java style conventions for objects and functions to provide consistency and ease of recognition. See *The Elements of Java Style* for additional specific guidelines.

78. Use Nouns When Naming Classes

Classes define objects or things. Use nouns, adjectives with nouns, or noun phrases for their names. Avoid using verbs. For example,

`Customer, Company, StockMarket`

79. Use UpperCamelCase for MATLAB Class and Object Names

This usage is consistent with Java practice and with recent usage by The MathWorks. For example, use the class name

```
AccountManager
```

with the object instantiation

```
Robert = AccountManager(...);
```

80. Use UpperCamelCase for Exception Names

Exceptions are classes, so use the same naming convention. If the exception is an error, then you may choose to append the suffix `Error`. This usage is consistent with Java practice and with recent usage by The MathWorks:

```
try
      :
catch KeyError
      :
end
```

81. Name Properties Like Structure Fields

Use class property names that follow the guidelines for structure fieldnames and variable names. In particular, use meaningful names written in lowerCamelCase. Do not repeat the class name in the property name.

Replace generic names such as

```
Picture.prop1
```

with meaningful names such as

```
Picture.contrast
```

Use lowerCamelCase for compound property names. Replace

```
Customer.LastOrder or Customer.last_order
```

with

```
Customer.lastOrder
```

Do not include the class name in the property name. Replace the redundant property name

```
Picture.PictureContrast
```

with

```
Picture.contrast
```

82. Name Methods Like Functions

Most methods are implemented as functions and should follow the same conventions for capitalization. A common practice is to use lowercase for short method names and lowerCamelCase for longer method names. The constructor method is a special case that uses UpperCamelCase because it must have the same name as its class.

Methods perform actions and should be named with action words:

```
Amplitude.normalize
```

Avoid redundant method names. The class name is implicit and does not need to be repeated. Replace

```
Result.updateresult
```

with

```
Result.update
```

83. Name Accessor Methods after their Properties

Getters are methods that return the value of a property. You should name a getter method by prefixing the word `get` to

the name of the property, unless it is Boolean. In that case, you prefix `is` to the name of the field instead of `get`:

```
getStamp, isValid
```

Setters are methods that modify the values of a property. You should name a setter method by prefixing the word `set` to the name of the property, regardless of its type:

```
setStamp, setComment
```

84. Use a Single Lowercase Word as the Root Name of a Package

The qualified portion of a package name should consist of a single word that captures the purpose and utility of the package. You can also use a meaningful abbreviation that is common in the project domain.

Data Files and Directories

85. Use Directory and Filenames that are Easy to Work with

Avoid embedded spaces in compound directory and filenames because they can cause difficulties or require special handling in some operating systems. Instead, use the underscore character or hyphen.

86. Use Sortable Numbering in Data Filenames

Data file and directory names often have a meaningful order, either by sequence or date. The MATLAB `dir` function and Current Folder window sort filenames in character order, which may not always be the best. For example, `data11.mat` would be displayed before `data2.mat`.

A simple way around this behavior is to add a large base number so that

`data2.mat`

becomes

`data102.mat`

and

`data11.mat`

becomes

`data111.mat`

This format is easily generated in a MATLAB `for` loop with index `iFile` as

```
base = 100;
for iFile = 1:nFiles
      fileNo = base+iFile;
      filename = ['data' int2str(fileNo) '.mat'];
      :
end
```

Another technique is to include leading zeros as needed so that

`data2.mat`

becomes

`data02.mat`

This approach requires a little more work to generate:

```
filename = ['data' ...
sprintf('%02.0f', iFile) '.mat'];
```

87. Use ISO Date Format

It is less ambiguous than other date formats because the fields are in an internationally recognized order, with a standard for embedded punctuation. ISO 8601 format sorts well; for example, 2007-01-31 comes before 2007-02-01. The embedded hyphen (-) is an allowed character in filenames.

Other date formats often use month names that vary with language and do not sort well. The slash character (/) used in some other formats is not legal in a file or directory name.

Replace

```
dataJan121999.mat or data12Jan1999.mat
```

with

```
data1999-01-12.mat or data_1999-01-12.mat
```

These ISO filenames can be easily generated in a loop for file access:

```
mainPart = 'data';
year = 1999;
month = 1;
day = [1:5];
formatSpec = 29;
for iDay = 1:length(day)
     dateNo = datenum([year month day(iDay)]);
     datePart = datestr(dateNo, formatSpec);
     filename = [mainPart datePart '.mat'];
end
```

If needed, a month name can be interpreted or produced using the month numbers to index into a cell array:

```
monthNames = {'JAN', 'FEB', 'MAR', 'APR', 'MAY',
'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC'}
```

Naming Summary

Enhance readability by using meaningful, consistent names. Make names easy to recognize and distinguish so that you are likely to use and type them correctly. Avoid names that are vague, misleading, too similar, or difficult to read.

Use capitalization to suggest the role of identifiers such as variables, functions, or classes. Follow the conventions that are common and successful in MATLAB and other languages. Use similar capitalization for similar entities such as functions and methods or variables and properties.

Append prefixes and suffixes if they clarify the meaning or use of a name. Follow common software usages. Avoid prefixes and suffixes that are likely to be distracting or cause maintenance issues.

Avoid name collisions, especially when your variable or function names would shadow existing functions. Use constant names and definitions that discourage unintentional redefinition.

Use variable names that mean what they say. Include standard prefixes and suffixes when they help communicate the role of the variable. Use lowerCamelCase for variable names.

Name functions for the action they perform or the output they produce. Use standard prefixes such as `is` or `get` in expected ways. Do not include temporary or ill-defined version numbering in function names.

Use nouns for the names of classes and objects. Write them in UpperCamelCase.