vendredi 17 mars 2023 12:17

MATLAB Onramp

Par contre, si vous ne connaissez pas le nom de la fonction, vous pouvez rechercher la documentation en utilisant des expressions. Essayez de rechercher dans la documentation une fonction qui crée des nombres normalement distribués (au lieu de nombres uniformément distribués) en utilisant :

```
doc nombres normalement distribués
Tracez mass2 (axe des y) par rapport à x
nom de propriété et une valeur associée.
 tracer une ligne
     traçage
Épaisseur de ligne = line width
ligne pleine (-)
entrées textuelles
      le texte est entouré de guillemets anglais (")
Étiquette = label
légende
notation avec le point : data.VariableName
Affectez le contenu de qqch à qqch
 les guillemets simples et les guillemets anglais
tableau = matrix
supérieurs/inférieurs  à
 opérateurs logiques AND (&) et OR (|)
Boucle = loop
     Au premier passage de la loop = in the first pass of the loop
Si la condition n'est pas remplie
le compteur de la boucle © : for c = 1:2
le tracé
```

Résumé de MATLAB Onramp

Syntaxe de base

Exemple	Description
`x = pi`	Créez des variables avec le signe égal (`=`). Le côté gauche (`x`) est le nom de la variable contenant la valeur sur le côté droit (`pi`).
`y = sin(-5)`	Vous pouvez fournir des entrées à une fonction en utilisant des parenthèses.

Gestion du desktop

Fonction	Exemple	Description
`save`	`save data.mat`	Enregistrez votre espace de travail actuel dans un fichier MAT.
`load`	`load data.mat`	Chargez les variables d'un fichier MAT dans l'espace de travail.
`clear`	`clear`	Effacez toutes les variables de l'espace de travail.
`clc`	`clc`	Effacez tout le texte de la fenêtre de commande.
`format`	`format long`	Modifiez le mode d'affichage de la sortie numérique.

Types de tableau

Exemple	Description
`4`	scalaire
`[3 5]`	vecteur ligne
`[1;3]`	vecteur colonne
`[3 4 5;6 7 8]`	matrice

Vecteurs uniformément espacés

Exemple	Description
`1:4`	Créez un vecteur à partir de `1` vers `4`, espacé par `1` en utilisant l'opérateur deux-points (`:`).
`1:0.5:4`	Créez un vecteur à partir de `1` vers `4`, espacé par `0.5`.
`linspace(1,10,5)`	Créez un vecteur avec `5` éléments. Les valeurs sont régulièrement espacées de `1` vers `10`.

Création de matrices

Exemple	Description
`rand(2)`	Créez une matrice carrée avec `2` lignes et `2` colonnes.
`zeros(2,3)`	Créez une matrice rectangulaire avec `2` lignes et `3` colonnes.

Indexation

Exemple	Description

`A(end,2)`	Accédez à l'élément de la deuxième colonne de la dernière ligne.
`A(2,:)`	Accédez à la totalité de la deuxième ligne
`A(1:3,:)`	Accédez à toutes les colonnes des trois premières lignes.
`A(2) = 11`	Modifiez la valeur du deuxième élément d'un tableau en `11`.

Opérations sur les tableaux

Exemple	Description

`[1 1; 1 1]*[2 2;2 2]`
`ans =`
`4 4`
`4 4`
	Effectuez une multiplication matricielle.

`[1 1; 1 1].*[2 2;2 2]`
`ans =`
`2 2`
`2 2`
	Effectuez une multiplication élément par élément.

Plusieurs sorties

Exemple	Description
`[xrow,xcol] = size(x)`	Enregistrez le nombre de lignes et de colonnes dans `x` à deux variables différentes.
`[xMax,idx] = max(x)`	Calculez la valeur maximale de `x` et sa valeur d'index correspondante.

Documentation

Exemple	Description
`doc randi`	Ouvrez la page de documentation de `randi` la fonction.

Traçage

Exemple	Description
`plot(x,y,"ro-","LineWidth",5)`	Tracez une ligne rouge (`r`) en pointillés (`--`) avec un marqueur cercle (`o`) avec une épaisseur de ligne importante.
`hold on`	Ajoutez la ligne suivante au tracé existant.
`hold off`	Créez un nouvel axe pour la prochaine ligne tracée.
`title("My Title")`	Ajoutez une étiquette à un tracé.

Utilisation des tables

Exemple	Description
`data.HeightYards`	Extrayez la variable `HeightYards` de la table data.
`data.HeightMeters = data.HeightYards*0.9144`	Dérivez une variable de table à partir de données existantes.

Logiques

Exemple	Description
`[5 10 15] > 12`	Comparez un vecteur à la valeur `12`.
`v1(v1 > 6)`	Extrayez tous les éléments dans `v1` qui sont supérieurs à `6`.
`x(x==999) = 1`	Remplacez toutes les valeurs dans `x` qui sont égales à `999` par la valeur `1`.

Programmation

Exemple	Description

`if x > 0.5`
`y = 3`
`else`
`y = 4`
`end`
	Si `x` est supérieur à `0.5`, définissez la valeur de `y` à `3`.
	Sinon, définissez la valeur de `y` à `4`.

`for c = 1:3`
`c`
`end`
	Le compteur de boucle (`c`) progresse à travers les valeurs `1`, `2` et `3`.
	Le corps de la boucle affiche chaque valeur de `c`.

Object-Oriented Programming Onramp

You can use the properties function to see the names of the *properties* (information contained by) an object in your workspace.

properties(x)

Similarly, you can use the methods function to see the names of the *methods* (functions) that work with your object.

methods(x)

orienteering even

All variables in MATLAB are arrays, even custom objects

you'll see how you can have multiple functions with the same name (and why that can be useful and make sense in the object-oriented world).

Being able to create new fields means that using a structure to hold your data could result in unexpected errors.

Using an object prevents your users from accidentally using an incorrect property name.

Any property can hold any size or type of data, even other custom objects (objects within objects).

In the OOP world, this relationship ("classA has a classB") is known as an *association*

Constrained $\ell_{p_1,p_2}^
eq:Opt-Lp1barp2

Definition \ref{eq:Opt-Lp1ba
optimisation problem, page \

Lemma \ref{lm:BUP-BS}, \gls{

In the OOP world, this relationship (classA has a classB) is known as an *association*
An association like this where the course can exist independently of the ID stick, but an ID stick contains a course, is referred to as an *aggregation*.
methods are functions intended to work on objects
Because the object is passed as an input, the method has access to all of the object's properties, using regular dot indexing
The `NaT` function creates an array of "not-a-time" values (the datetime equivalent of `NaN`).
a method of your class that happens to have the same name as that of your class.

*This method is special – it is called the **constructor** method, because it is called only for creating objects.*
However, methods are functions that are specific to their classes.
The precedence rules enable you to use whatever name makes sense for your methods, without having to worry about confusion with another function.
Deliberately reusing the same method name for different classes is known as *overloading*.
The default `display` method prints the name of the variable, an equals sign, then calls the `disp` method.
Although you can write custom methods for both `display` and `disp`, it is common to write a custom `disp` method while leaving the default `display` method.
does not cause a name conflict because of the type of what is being passed to
mat2str
How about using `mat2str(c.Waypoints)` instead of `join` and `string`
Every operator has a corresponding functional equivalent

`a + b`	`plus(a,b)`

you can call other methods from a method, including a method with the same name from another class
In cases where you want to define elementwise operations, but there is no meaningful matrix operation, you might want to define both operators to have the same (elementwise) behavior.

Property Validation

You can specify the size and type of how you want each property to be stored. You can also add checks that will be performed to ensure that property values have certain characteristics, such as numeric values being positive or text values belonging to a given set.

When you try to assign a value to a property, MATLAB will try to put it into the given type, then size, then check the characteristics. If the given value doesn't work, you will get an error.

You can also specify a default value for the property (which must satisfy the given requirements).

```
classdef classname properties PropertyName (size) type {check1, check2, ...} = defaultvalue end end
```

You can choose which (if any) of these aspects you want to specify, but those you do specify have to be in this order.

Coercion (Not as Bad as It Sounds)

Specifying a property's type determines how MATLAB will interpret and store the values for that property. What happens if you specify a value that doesn't fit that specification? MATLAB will try to *coerce* the value into the correct form.

Despite its negative meaning in normal English, in software development, coercion simply means trying to convert data in one form into another form. MATLAB has rules for converting between data types. If it knows how to convert a given value into the type specified for a property, it will. If not, then you will get an error.

This conversion is convenient, because it means you can allow your users to provide data in any way that makes sense, without you having to specify those types or how to convert them into the way you want the property stored.

However, this automatic coercion can cause unexpected behavior. For example, the `Name` property of the course object is specified as a string, but giving a name of `42` does not cause an error – it is coerced into the string `"42"`. This behavior may or may not be what you want. If it is, great! If not, you need to understand why it is happening and how to work around it. (See the lesson on *Validating Method Inputs* for a way to do this.)