DOCUMENTATION INTERNE

MATHON Tristan
RASOAMANANA Nathanaël

Année 2023-2024

● Structure conventionnelle du programme

- SORTIE -

. *valid* : Indique l'état de la validation.
plusieurs états sont à considérer, les voici :
Si tout est ok : 2
En cas d'erreur (cf CODE ERREURS)

. *message* : Dépend de *valid*, si *valid* est différent de 2 alors le message indique le type d'erreur.
Si *valid* est 2 alors cela dépend de la fonction.

- CODES ERREURS -

Si une fonction retourne une erreur, elle sera toujours sous la forme ci-dessous. (S'applique notamment dans le modules liés à l'interface avec le prouveur, le module de vérification et de validation du tableau, le module de gestion de la zone textuelle).

1.1 : erreur de syntaxe
(ex : [1.1, message, ligne, charac])

1.2 : erreur de typage.
(ex : [1.2, message, var])

1.3 : variable déjà définie..
(ex : [1.3, message])

1.4 : variable déjà initialisée.
(ex : [1.4, message])

1.5 : erreur inconsistante..
(ex : [1.5, message])

1.6 : élément DOM inexistant.
(ex : [1.6, message])

1.7 : Out of bound
(ex : [1.7, message])

1.9 : erreur interne.
(ex : [1.9, message])

- INFORMATIONS SUR LES FONCTIONS -

. Description

Décris ce que fait la fonction.

. Fonction.s appelée.s

Liste des fonctions qu'elle appelle en interne au module (sauf si c'est indiqué).

. Fonction.s appelante.s

Liste des fonctions qui l'appelle en interne au module (sauf si c'est indiqué).

. Définition des arguments en entré

Définition des arguments théoriques en entrée de la fonction.

. Retour théorique

Retour sans erreur de la fonction illustrée par des valeurs théoriques.

. Définition des valeurs en sortie

Définitions des valeurs retournées par la fonction.

. Exemple d'utilisation

Un exemple de cas pratique sans erreur.

. Retour erreur

Un exemple de cas pratique avec erreur.

- Modules de gestion d’événements DOM

https://code.jquery.com/jquery-3.6.0.min.js : bibliothèque jQuery pour la manipulation du DOM et la gestion des événements.
https://code.jquery.com/ui/1.12.1/jquery-ui.min.js: bibliothèque jQuery UI utilisée pour la fonctionnalité d'autocomplétion.

1.1 Relations d’utilisation avec d’autres modules

Ces modules interagissent avec les modules de gestion internes, notamment la gestion de la saisie de commandes textuellement et le module d’affichage de l’aide sur les commandes disponibles. Ils permettent principalement l’autocomplétion lors de la saisie d’une commande.

1.2 Les modules utilisés par ce module et ceux utilisant ce module.

Modules utilisés par ce module :

- Le module de la gestion de la saisie des commandes.
- Le module d’affichage de l’aide.
- Les modules de visualisation de l’interface.

Modules utilisant ce module :

- Le module de la gestion de la saisie des commandes.
- Le module d’affichage de l’aide.
- Les modules de visualisation de l’interface.

● Modules de visualisation

Les fichiers correspondant aux différents modules dans le code source:

interface.html : module du visuel principal.
styles.css : module d'habillage de l'interface.

Ce module gère tout l'aspect visuel de l'interface du programme, il contient notamment la structure de base du tableau et toutes les zones dédiées à chaque module de gestion interne.

1.1 Relations d'utilisation avec d'autres modules

Ce module interagit avec les modules de gestion internes, notamment la vérification et la validation du tableau ainsi que la saisie de commandes textuelles.

1.2 Les modules utilisés par ce module et ceux utilisant ce module.

Modules utilisés par ce module :

- Les modules de gestion internes.
- Les modules de gestion d'événements DOM

Modules utilisant ce module :

- Les modules de gestion internes.
- Les modules de gestion d'événements DOM

- Modules liés à l'interface avec le prouveur
  (développé dans le cadre du projet par les encadrants)

Les fichiers correspondant aux différents modules dans le code source:

tabvar.js : Module tabvar contenant les différentes fonctions dans l'API.
tabvar-addons.js : Module contenant d'autres fonctions de l'API ajoutées en dehors du module tabvar.

Ces modules ont été développés au cours du projet par notre encadrant Martin Bodin. Il est codé en langage OCaml mais ensuite compilé en JavaScript pour pouvoir être utilisé par le programme. Nous avons contribué à la documentation des fonctions incluses dans ce module. Ces fonctions permettent notamment la vérification et la validité du tableau de variation.

1.1 Relations d'utilisation avec d'autres modules

Ce module interagit avec les modules de gestion internes, notamment la vérification et la validation du tableau ainsi que la saisie de commandes textuelles.

1.2 Les modules utilisés par ce module et ceux utilisant ce module.

Modules utilisés par ce module :

- Aucun.

Modules utilisant ce module :

- Gestion de la vérification des données du tableau et des entrées de fonctions.
- Gestion de la zone textuelle.

1.3 Procédures :

1.3.1 Les fonctions utilisées par le programme

init()

. Description

Permet d'initialiser le module "tableau de variation".

. Définition des arguments en entré

Pas d'argument

. Retour théorique

→ *[valid, new_env, message]*

. Définition des arguments en sortie

valid : indique l'état de la validation
new_env : Nouvel environnement
message : "Require Import Tabvar." pour indiquer l'importation correct de TabVar ou retourne "Already initialised" si TabVar a déjà été importé.

. Cas pratique

init()
→ *[2, env, "Require Import Tabvar."]*

. Retour error

init()
→ *[1.4, "Already initialised"]*

validate(env, expr)

. Description

Permet de vérifier si une fonction est correctement écrite et ainsi elle peut être lisible par l'API.

. Définition des arguments en entré

env : correspond à l'environnement actuel.
expr : (string) Expression à valider.

. Retour théorique

→ *[valid, message]*

. Définition des valeurs en sortie

valid : indique l'état de la validation
message : (string) retourne l'expression si cette dernière est correcte, sinon retourne un message d'erreur.

. Cas pratique

*validate(env, "5*x+8")*
→ *[2, "5×x + 8"]*

. Retour error

validate(env, "5x+8")
→ *[1.1, "Parser error at line 1, character 3", 1, 3]*

deriv(env, expr, var)

. Description

Retourne une dérivé de la fonction mis en argument en fonction d'une variable

. Définition des arguments en entré

env : correspond à l'environnement actuel.
expr : (string) Expression à dériver.
var : (string) Variable qui permet de dériver l'expression en fonction d'elle.

. Retour théorique

→ *[valid, message]*

. Définition des arguments en sortie

valid : indique l'état de la validation.
message : (string) Affiche la dérivé de la fonction *expr* (argument en entrée).

. Cas pratique

*deriv(env,"5*x+6","x")*
→ [2, "5"]

compare_values(env, expr1, expr2)

. Description

Compare la valeur dans la première chaîne de caractère à la valeur de la seconde chaîne de caractère. Comparer *expr1* à *expr2* en retournant le résultat de comparaison qui peut être "<", ">", "=" ou "?".

. Définition des arguments en entré

env : correspond à l'environnement actuel.
expr1 : (string) Expression comparée.
expr2 : (string) Expression comparante.

. Retour théorique

→ *[valid, symb]*

. Définition des arguments en sortie

valid : indique l'état de la validation
symb : symbole de comparaison de la première expression par rapport à la deuxième

. Cas pratique

compare_values(env, "sqrt(1)+48", "sin 42 - 2")
→ [2, ">"]

develop(env, expr)

. Description

Développer une expression arithmétique en une expression équivalente.

. Définition des arguments en entré

env : correspond à l'environnement actuel.
expr : (string) Expression à traiter.

. Retour théorique

→ *Array [valid, message]*

. Définition des arguments en sortie

valid : indique l'état de la validation
message : (string) expression développée

. Cas pratique

develop(env, "((a+b)*c)*d")
→ Array [2, "a×c×d + b×c×d"]

is_positive(env, expr)

. Description

Indique si le signe d'une expression est positif (zéro étant considéré positif).

. Définition des arguments en entré

env : correspond à l'environnement actuel
expr : (string) expression à traiter

. Retour théorique

→ *Array [valid, message]*

. Définition des arguments en sortie

valid : indique l'état de la validation
message : (string) affiche true si l'expression e (en entrée) est positive, false si c'est négatif et "?" si on ne sait pas.

. Cas pratique

is_positive(env, "-9+5")
→ *Array[2, false']*

1.3.2 Les fonctions utilisées par le programme (avec problèmes)

(expliquer les problèmes)

function_definition(env, f, var, expr)

. Description

Permet de définir la fonction principale sur laquelle on se base pour remplir le tableau de variation. Permet aussi de définir la variable mise en paramètre

(i.e pas besoin de définir la variable de la fonction principal grâce à la fonction suppose_var()).

. Définition des arguments en entré

env : correspond à l'environnement actuel.
f : (string) Nom de la fonction.
var : (string) Le nom de l'argument de la fonction.
Remarque: On ne peut pas utiliser x comme argument car c'est déjà déclaré comme variable globale.
expr : (string) Expression de la fonction.

. Retour théorique

→ [valid, *new_env* , expr]

. Définition des arguments en sortie

valid : indique l'état de la validation
new_env : nouvel environnement
expr : expression qui est maintenant défini

. Cas pratique

function_definition(env, "f", "t","2*t*t-4*t+2")
→[2, Array(6), 'Definition f (t : real) := 2×t×t − 4×t + 2.']

. Retour error

function_definition(0, "f", "x", "3+e")
→ [1.2, "Unknown variable e.", "e"]

interval(env, expr)

. Description

Détermine (une sur-approximation) l'intervalle d'une expression.

. Définition des arguments en entré

env : correspond à l'environnement actuel
expr : (string) Expression à traiter

. Retour théorique

→ [valid, message]

. Définition des arguments en sortie

valid : indique l'état de la validation
message : (string) intervalle dans laquelle l'expression est définie

. Cas pratique

*interval(env,"15*x-8")*
→ [2, ']–∞ ; +∞ [']

suppose_var(env, x)

. Description

Déclare une variable.

. Définition des arguments en entré

env : correspond à l'environnement actuel
x : (string) Variable à déclarer

. Retour théorique

→ [valid, new_env, message]

. Définition des arguments en sortie

valid : indique l'état de la validation
new_env : Nouvel environnement
message : (string) indique que la variable *x* (argument en entrée) est bien un réel.

. Cas pratique

suppose_var(env, "y")
→ [2, Array(6), 'Variable y : real.']

print_env(env)

. Description

Affiche l'environnement actuel (pour aider le débogage).

. Définition des arguments en entré

env : correspond à l'environnement actuel

. Retour théorique

→ [valid, message]

. Définition des arguments en sortie

valid : indique l'état de la validation
message : (string) affiche tous les éléments de l'environnement.

. Cas pratique

print_env(env)
→ [2, 'x : real (]–∞ ; +∞[) ; ']

defined(env, expr)

. Description

Indique si une expression est bien définie (y) dans l'environnement, si elle n'est pas définie (n), ou si on ne sait pas (?).

. Définition des arguments en entré

env : correspond à l'environnement actuel
expr : (string) Expression à traiter

. Retour théorique

→ [valid, message]

. Définition des arguments en sortie

valid : indique l'état de la validation
message : (string) affiche "y" si l'expression est définie, "n" si elle n'y est pas et "?" si on ne sait pas.

. Cas pratique

*defined(env, "12*x*x-6*x+15")*
→[2, 'y']

suppose(env, prop)

. Description

Suppose une propriété comme étant vraie dans l'environnement.

. Définition des arguments en entré

env : correspond à l'environnement actuel
prop : (string) propriété

. Retour théorique

→ [valid, new_env, message]

. Définition des arguments en sortie

valid : indique l'état de la validation
new_env : Nouvel environnement
message : (string) affiche la propriété ou l'hypothèse.

. Cas pratique

suppose(env, "x = 5")
→ [2, Array(6), 'Hypothesis H : x = 5.']

proof_assistant(env, cmd)

. Description

Exécute des commandes comme si on les avait données à l'assistant de preuve.

. Définition des arguments en entré

env : correspond à l'environnement actuel
cmd : (string) commande d'assistant de preuve à traiter

. Retour théorique

→ *[valid, new_env, mess]*

. Définition des arguments en sortie

valid : indique l'état de la validation
new_env : Nouvel environnement
mess : Retour sous forme de message que l'assistant de preuve aurait fait

. Cas pratique

tabvar.proof_assistant (env, "Variable y : real. Hypothesis (H1 : y > 3) (H2 : y < 4). Lemma prop : (y > 2). Proof. Qed.")
→[2, Array(6), Array(4)]
Détail de ce que contient le "Array (4)"

0: "y has been successfully assumed."
1: "H1 has been successfully assumed."
2: "H2 has been successfully assumed."
3: "prop has been successfully defined."

1.3.3 Les fonctions non utilisées par le programme (n'apparaissent pas ou ont des erreurs internes)

valeur_interdite(env, v, f)

. Description

Indique si une valeur définie est une valeur interdite de la fonction.

. Définition des arguments en entré

env : correspond à l'environnement actuel
v : (string) valeur à vérifier
f : (string) fonction à traiter

. Retour théorique

→[valid, message]

. Définition des arguments en sortie

valid : indique l'état de la validation
message : (string) affiche true si la valeur v (en entrée) est une valeur interdite, false si ça n'en est pas une et "?" si on ne sait pas.

extremum_local(env, v, f)

. Description

Indique si une valeur définie est un extremum local de la fonction.

. Définition des arguments en entré

env : correspond à l'environnement actuel
v : (string) valeur à vérifier
f : (string) fonction à traiter

. Retour théorique

→[valid, message]

. Définition des arguments en sortie

valid : indique l'état de la validation
message : (string) affiche true si l'expression e (en entrée) est un extremum local, false si ça n'en est pas un et "?" si on ne sait pas.

signe_deriv(env, v, f)

. Description

Indique le signe de la dérivée de la fonction à une valeur définie.

. Définition des arguments en entré

env : correspond à l'environnement actuel
v : (string) valeur à vérifier
f : (string) dérivée de la fonction à traiter

. Retour théorique

→[valid, message]

. Définition des arguments en sortie

valid : indique l'état de la validation
message : (string) affiche "+" si positif, "-" si négatif, "?" si on ne sait pas.

roots(env, x, e)

. Description

Essaye de trouver des valeurs qui annulent l'expression.

. Définition des arguments en entré

env : correspond à l'environnement actuel
x : (string) valeur à vérifier
e : (string) expression à traiter

fresh(env, x)

. Description

Renvoie une variable fraîche dans l'environnement.

. Définition des arguments en entré

env : correspond à l'environnement actuel
x : (string) Variable à déclarer

define_new_function(f, expr)

. Description

Définit une nouvelle fonction à ajouter à tabvar lors de l'initialisation.

. Définition des arguments en entré

f : nom de la fonction
expr : (string) expression de la fonction

intersting_values(env, x, expr)

. Description

Renvoie des valeurs potentiellement intéressantes (par exemple des valeurs interdites ou des points d'annulation).

. Définition des arguments en entré

env : correspond à l'environnement actuel
x : (string) Valeurs dont on s'intéresse
expr : (string) Expression à traiter

replace(env, expr, x, ex)

. Description

Remplace une variable par une autre dans l'expression.

. Définition des arguments en entré

env : correspond à l'environnement actuel
expr : (string) Expression à traiter
x : (string) Variable à remplacer
ex : (string) Nouvelle variable remplaçant la précédente

- Modules de gestions internes

Les fichiers correspondant aux différents modules dans le code source:

gestion_tab.js : Gestion de l'interaction avec le tableau de variation
verif_tab.js : Gestion de la vérification des données du tableau et des entrées de fonctions
gestion_zone_texte.js : Gestion de la zone textuelle
gestion_aide.js : Gestion de la zone dédiée à l'aide sur les fonctions saisies dans la zone textuelle

1. Gestion de l'interaction avec le tableau de variation.

Ce module a pour objectif de gérer les interactions et les manipulations du tableau de variation ainsi que la saisie des fonctions à traiter.

1.1 Relations d'utilisation avec d'autres modules

Ce module interagit principalement avec le DOM pour manipuler les éléments HTML du tableau et des input pour les fonctions.

1.2 Les modules utilisés par ce module et ceux utilisant ce module.

Modules utilisés par ce module :

- Manipulations du DOM (éléments HTML)
- Événements JavaScript (addEventListener, onclick, etc.)

Modules utilisant ce module :

- Modules de visualisation de l'interface.

1.3 Procédures :

1.3.1 Les fonctions utilisées par le programme

showValues() :

. Description

Récupère et affiche les valeurs et les symboles transformés en texte des lignes 1, n-1 et n du tableau (n est le nombre de lignes).

. Fonction.s appelée.s

Pas de fonction

. Fonction.s appelante.s

getValues()

. Retour théorique

→[Arr1, Arr2, Arr3]

. Définition des valeurs en sortie

Arr1 : (Array) Contenus de la 1ère ligne du tableau
Arr2 : (Array) Contenus de l'avant dernière ligne du tableau
Arr3 : (Array) Contenus de la dernière ligne du tableau

. Exemple d'utilisation

showValues()
→ ['-80', '-0.9', '1.5', '80']
→ ['neg', 'VI', 'pos', '0', 'neg']
→ ['down', 'up', 'down']

. Retour erreur

Erreur d'accès à l'ID du tableau pour la récupération du tableau.
Erreur d'index (le tableau doit contenir au moins 3 lignes).

getValues(index) :

. Description

Récupère les valeurs spécifiques du tableau à partir des retours de *showValues*() selon l'index fourni.

. Fonction.s appelée.s

showValues()

. Fonction.s appelante.s

Pas de fonction.

. Définition des arguments en entrée

index : valeur en paramètre
index = 1 pour récupérer les valeurs de x
index = 2 pour récupérer les signes de la dérivée
index = 3 pour récupérer les variations de la fonction

. Retour théorique

→[Arr]

. Définition des valeurs en sortie

Arr : (Array) Contenus selon l'index en paramètre.

. Exemple d'utilisation

getValues(1)
→ ['-80', '0', '0', '80']

. Retour erreur

Erreur de validité sur les valeurs retournée par showValues().
Erreur dans les paramètres (si index est différent de 1, 2, 3).

1.3.2 L'entrée de la fonction

changeVarFun() :

. Description

Change le nom de la fonction principale et de la variable.

. Fonction.s appelée.s

adjustInputWidth(input)

. Fonction.s appelante.s

Appelée dans le module de l'interface HTML.

. Retour théorique

Pas de retour mais des changements de comportement dans le tableau.

. Définition des valeurs en sortie

Pas de valeurs de sortie mais des effets (affichage dans le tableau).

. Exemple d'utilisation

Utilisation spécifique sur les zones de texte dédiées.

. Retour erreur

Erreur d'existence des champs d'entrées spécifiques.
Erreur de destination (l'emplacement du changement à effectuer n'existe pas).

adjustInputWidth(input) :

. Description

Ajuste la largeur d'un champ de saisie en fonction de son contenu.

. Fonction.s appelée.s

Pas de fonction.

. Fonction.s appelante.s

changeVarFun()

. Définition des arguments en entrée

input : (String) Valeur saisie dans la zone de texte.

. Retour théorique

Pas de retour mais des changements de comportement de la taille de la zone de texte.

. Définition des valeurs en sortie

Pas de valeurs de sortie.

. Exemple d'utilisation

Utilisation spécifique sur les zones de texte dédiées.

1.3.3 Les cellules du tableau

changeValue(link) :

. Description

Permet de saisir une valeur dans le bouton value.

. Fonction.s appelée.s

updateButtonValue(button, newValue)

. Fonction.s appelante.s

addColumn(button)

. Définition des arguments en entrée

link : Elément DOM (lien cliqué correspondant).

. Retour théorique

Pas de retour mais des changements de comportement directement sur le DOM. (Changement du contenu de la cellule en champ de saisie).

. Exemple d'utilisation

Utilisation directement dans le tableau.

showDropdown(element) :

. Description

Affiche le menu déroulant lorsque la souris passe dessus.

. Fonction.s appelante.s

createDropdownCell()
addExtraRow()

. Définition des arguments en entrée

element : Elément DOM de l’interface avec la classe : dropdown-content

. Retour théorique

Pas de retour mais des changements de comportement sur les éléments DOM spécifiés.

. Exemple d’utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

hideDropdown(element) :

. Description

Masque le menu déroulant lorsque la souris quitte l'élément.

. Fonction.s appelante.s

createDropdownCell()
addExtraRow()

. Définition des arguments en entrée

element : Elément DOM de l’interface avec la classe : dropdown-content

. Retour théorique

Pas de retour mais des changements de comportement sur les éléments DOM spécifiés.

. Exemple d’utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

updateButtonValue(button, newValue) :

. Description

Met à jour le texte du bouton avec la nouvelle valeur saisie.

. Fonction.s appelante.s

changeValue(button)

. Définition des arguments en entrée

button : Bouton dans lequel se trouve la valeur à changer.
newValue : Nouvelle valeur saisie.

. Retour théorique

Pas de retour mais des changements de comportement sur les éléments DOM spécifiés.

. Exemple d’utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

toggleMore(link, value) :

. Description

Affiche l'identité du bouton MoreLess (bouton avec les options + ou - pour les signes de la dérivée).

. Fonction.s appelante.s

addColumn(button)
addExtraRow()

. Définition des arguments en entrée

link : Elément DOM (lien cliqué où doit se faire l'affichage).
value : Valeur choisie (- ou +).

. Retour théorique

Pas de retour mais des changements de comportement sur les éléments DOM spécifiés.

. Définition des valeurs en sortie

Pas de valeurs de sortie.

. Exemple d'utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

toggleIncrease(link, value) :

. Description

Affiche l'identité du bouton Increase.

. Fonction.s appelante.s

addColumn(button)
addExtraRow()

. Définition des arguments en entrée

link : Elément DOM (lien cliqué où doit se faire l'affichage).
value : Valeur choisie (croissant ou décroissant)

. Retour théorique

Pas de retour mais des changements de comportement sur les éléments DOM spécifiés.

. Exemple d'utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

toggleVI(link, value)

. Description

Affiche l'identité du bouton VI.

. Fonction.s appelante.s

addColumn(button)
addExtraRow()

. Définition des arguments en entrée

link : Elément DOM (lien cliqué où doit se faire l'affichage).
value : Valeur choisie (valeur interdite, zéro, non défini)

. Retour théorique

Pas de retour mais des changements de comportement sur les éléments DOM spécifiés.

. Exemple d'utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

toggleEmptySymbol() :

. Description

Gère l'affichage du symbole "vide" dans l'avant-dernière ligne du tableau.

. Fonction.s appelante.s

addColumn(button)
addExtraRow()

. Retour théorique

La fonction ne retourne aucune valeur.

. Définition des valeurs en sortie

Il n'y a pas de valeurs de sortie, mais l'état du DOM est modifié.

. Exemple d'utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

. Retour erreur

TypeError : Si un élément attendu est null ou undefined.
Absence de l'élément table : Si l'élément avec l'ID tabvar n'existe pas dans le DOM.

createCell() :

. Description

Créer une nouvelle cellule de tableau.

. Fonction.s appelante.s

addColumn(button)

. Retour théorique

La fonction retourne une nouvelle cellule de tableau HTML (<td>) avec la classe additional-column.

. Définition des valeurs en sortie

La valeur de sortie est un élément HTML <td> avec une classe CSS nommée additional-column.

. Exemple d'utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

createDropdownCell(buttonOrDiv, buttonText, buttonClass, buttonOnClick, contentHTML) :

. Description

Créer un nouvel élément avec un menu déroulant.

. Fonction.s appelée.s

showDropdown(element)
hideDropdown(element)

. Fonction.s appelante.s

addColumn(button)
addExtraRow()

. Définition des arguments en entrée

buttonOrDiv (string) : Détermine si un bouton ou un div est créé ("button" pour un bouton, autre chose pour un div).
buttonText (string) : Texte affiché sur le bouton ou le div.
buttonClass (string) : Classe CSS appliquée au bouton ou au div.
buttonOnClick (function) : Fonction appelée lors du clic sur le bouton ou le div.
contentHTML (string) : HTML à inclure dans le contenu du menu déroulant.

. Retour théorique

La fonction retourne un élément HTML représentant le menu déroulant, avec le bouton/div et son contenu.

. Définition des valeurs en sortie

La valeur de sortie est un élément HTML avec une structure de menu déroulant comprenant un bouton ou un div et un contenu.

. Exemple d'utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

. Retour erreur

Erreur de type : Si un argument requis n'est pas fourni ou a une valeur incorrecte.

1.3.4 Les colonnes du tableau

addColumn(button) :

. Description

Ajoute deux nouvelles colonnes à un tableau HTML spécifié (lorsqu'on clique sur le bouton + de la première ligne du tableau). Elle gère l'ajout de cellules pour chaque ligne du tableau en utilisant des menus déroulants personnalisés.

. Fonction.s appelée.s

createCell()
createDropdownCell(buttonOrDiv, buttonText, buttonClass, buttonOnClick, contentHTML)
changeValue(element)
deleteColumns(element)
toggleIncrease(element, text)
toggleVI(element, text)
toggleMore(element, text)
toggleEmptySymbol()

. Définition des arguments en entrée

button : Le bouton qui déclenche l'ajout de colonnes, utilisé pour déterminer la position où les nouvelles colonnes seront insérées (s’applique sur les boutons “+” de la première ligne du tableau).

. Retour théorique

Pas de retour. Modification directe sur le DOM.

. Exemple d’utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

. Retour erreur

Erreur de type : Si button n'est pas un élément HTML valide.

deleteColumns(link) :

. Description

Supprime supprimer la colonne contenant le lien cliqué ainsi que la colonne située immédiatement à droite dans une table HTML.

. Définition des arguments en entrée

link : Le lien cliqué à partir duquel la suppression des colonnes doit commencer.

. Retour théorique

Pas de retour. Modification directe sur le DOM.

. Exemple d’utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

. Retour erreur

TypeError : Si l'élément link n'est pas contenu dans une cellule de tableau <td>, l'appel à link.closest('td') retournera null

Invalid Table Structure : Si le tableau n'est pas bien formé ou que les indices de cellule sont incorrects

1.3.5 Les lignes du tableau

addExtraRow() :

. Description

Ajoute une nouvelle ligne en deuxième position du tableau en dupliquant les éléments à la deuxième ligne précédente.

. Fonction.s appelée.s

createDropdownCell(tagName, text, className, onClickFunction, innerHTMLContent)
changeValue(element)
deleteRow(element)
showDropdown(element)
hideDropdown(element)
toggleMore(element, text)
toggleIncrease(element, text)
toggleVI(element, text)
adjustInputWidth(input)
toggleEmptySymbol()

. Fonction.s appelante.s

Appelée par un événement de clic sur le bouton approprié.

. Retour théorique

Pas de retour. Modification directe sur le DOM.

. Exemple d’utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

. Retour erreur

ReferenceError : Si les fonctions appelées ne sont pas définies.

TypeError : Si la table n'existe pas ou si une des cellules de la deuxième ligne est inaccessible, cela peut entraîner une erreur de type.

removeMiddleRows() :

Description

Supprime toutes les lignes sauf la première, l'avant-dernière et la dernière. (Toutes les lignes supplémentaires pour les sous-fonctions).

. Retour théorique

Pas de retour. Modification directe sur le DOM.

. Exemple d'utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

. Retour erreur

ReferenceError : Si l'élément avec l'ID "tabvar" n'existe pas dans le DOM.
TypeError : Si l'accès aux propriétés ou méthodes des éléments du tableau échoue.

deleteRow(link) :

. Description

Supprime une ligne spécifique. La fonction deleteRow permet de supprimer une ligne spécifique lorsque l'utilisateur clique sur un lien associé à cette ligne (option “Supprimer” au survol de la première colonne).

. Fonction.s appelante.s

Appelée par un événement de clic sur un lien ("Supprimer") situé dans la première colonne de la ligne du tableau ajoutée.

. Définition des arguments en entrée

link : Un élément HTML à partir duquel la ligne parente (tr) doit être identifiée et supprimée.

. Retour théorique

Pas de retour. Modification directe sur le DOM.

. Exemple d'utilisation

Utilisation spécifique sur les éléments DOM spécifiés.

. Retour erreur

TypeError : Si link n'est pas un élément HTML valide ou si closest('tr') échoue à trouver un élément <tr> parent, cela peut entraîner une erreur de type.
ReferenceError : Si l'élément parent (<tr>) n'existe pas ou si l'élément avec l'ID spécifié n'est pas trouvé dans le DOM.

1.3.6 Sauvegarde des données dans un fichier .json

saveValuesToJSON() :

. Description

Sauvegarde les données de la 1ère, avant dernière et de la dernière lignes du tableau dans un fichier JSON.

. Fonction.s appelée.s

showValues()

. Retour erreur

Erreur interne :
→ [1.9, Erreur interne : Les valeurs retournées par showValues sont invalides]
→ [1.9, Erreur interne : Erreurs lors de la sauvegarde des valeurs dans le fichier JSONS]

2. Gestion de la vérification et de la validation des données du tableau de variation.

Ce module valide et vérifie ce que l'utilisateur a fait dans le tableau de variation.

2.1 Relations d'utilisation avec d'autres modules

Ce module permet de faire la liaison entre l'interface graphique et l'API.

2.2 Les modules utilisés par ce module et ceux utilisant ce module.

Modules utilisés par ce module :

- Manipulations du DOM (éléments HTML)
- tabvar
- tabvar-addons

Modules utilisant ce module :

- Interface utilisateur : l'utilisateur s'en sert pour valider et vérifier son tableau de variation

2.3 Procédures :

checkAll()

. Description

Permet les vérifications et les validations du tableau de variation mais aussi l'affichage des validation et des erreurs dû à ce dernier.

. Fonction.s appelée.s

clearMessageBox2()
validation_funMain(env)
validation_funDeriv(env)
verification_funDeriv(env)
validation_subFun(env)
compareExpressionsInRow(env)
messageDisplay(message)

. Fonction.s appelante.s

Aucune

. Définition des arguments en entré

Pas d'argument

. Retour théorique

→ *[valid, message]*

. Définition des valeurs en sortie

valid : indique l'état de la validation

message : "vérification et validation ok" si il n'y a pas d'erreur lors des vérification et des validations sinon renvoi "Erreur interne, veuillez nous excuser".

. Exemple d'utilisation

checkAll()

→ *[2, "verification et validation ok"]*

. Retour erreur

checkAll()

→ *[1.9, "Erreur interne, veuillez nous excuser"]*

clearMessageBox2()

. Description

Permet de vider la zone de message dans la box2, utiliser pour afficher les message de validation et de vérification.

. Fonction.s appelée.s

Aucune.

. Fonction.s appelante.s

checkAll()

. Définition des arguments en entré

Pas d'argument

. Retour théorique

→ *[valid, message]*

. Définition des arguments en sortie

valid : indique l'état de la validation

message : "élément existant" si l'élément du DOM a vidé est existant, "L'element n'existe pas" si l'élément du DOM n'existe pas, sinon "Erreur interne, veuillez nous excuser".

. Exemple d'utilisation

clearMessageBox2()
→ [2, ""élément existant*"]*

. Retour error

clearMessageBox2()
→ [1.6, "Element inexistant"]

messageValidExpr(message_tab)

. Description

Permet d'afficher un message en fonction de la validation de l'expression grâce, en amont, de l'appel de la fonction *tabvar.validate()*.

. Fonction.s appelée.s

messageDisplay()

. Fonction.s appelante.s

validation_funMain(env), *validation_funDeriv(env)*

. Définition des arguments en entré

message_tab: tableau de retour de la fonction *tabvar.validate().*

. Retour théorique

→ [valid , message]

. Définition des arguments en sortie

valid : indique l'état de la validation
message : "message expression ok" si une expression est valide et affichée, sinon "Erreur interne, veuillez nous excuser".

. Exemple d'utilisation

messageValidExpr([2, "5×x"])
→ [2 , "message expression ok"];

. Retour error

messageValidExpr([2, "5×x"])
→[1.9, "Erreur interne, veuillez nous excuser"];

messageDisplay(message)

. Description

Permet d'afficher le message qui est entré en paramètre.

. Fonction.s appelée.s

Aucune

. Fonction.s appelante.s

checkAll()
messageValidExpr()
verification_funDeriv()
validation_subFun()

. Définition des arguments en entré

message : chaîne de caractères.

. Retour théorique

→ *[valid, message]*

. Définition des arguments en sortie

valid : indique l'état de la validation
message : "élement existant" si l'élément existe dans le DOM, "Element inexistant" si l'élément n'existe pas, sinon "Erreur interne, veuillez nous excuser".

. Exemple d'utilisation

messageDisplay("Bonjour")
→ *[2, "*élement existant*"]*

. Retour error

messageDisplay("Bonjour")
→ *[1.6, "Element inexistant"]*

validationFunMain()

. Description

Permet de valider l'expression de la fonction principale.

. Fonction.s appelée.s

tabvar.suppose_var()
tabvar.validate()
messageValidExpr()

. Fonction.s appelante.s

checkAll()

. Définition des arguments en entré

env: environnement actuel.

. Retour théorique

→ *[valid, new_env, expr]*

. Définition des arguments en sortie

valid : indique l'état de la validation
new_env : nouvel environnement
expr : (string) Expression de la fonction valider.

. Exemple d'utilisation

validation_funMain(env)
*→ [2, env, "5*x*x"]*

. Retour error

validation_funMain(env)
→ [1.6, "Element inexistant"]

validation_funDeriv(env)

. Description

Permet de valider l'expression de la dérivé de la fonction principale.

. Fonction.s appelée.s

tabvar.suppose_var()
tabvar.validate()
messageValidExpr()

. Fonction.s appelante.s

checkAll()

. Définition des arguments en entré

env: environnement actuel.

. Retour théorique

→ [valid, expr]

. Définition des arguments en sortie

valid : indique l'état de la validation
expr : Chaine de caractère de l'expression de la fonction validée.

. Exemple d'utilisation

validation_funDeriv(env)
*→ [2, "10*x"]*

. Retour error

validation_funDeriv(env)
→ [1.9, "Erreur interne, veuillez nous excuser"]

verification_funDeriv(env)

. Description

Permet de vérifier si le dérivé est bien la dérivé de la fonction principale.

. Fonction.s appelée.s

tabvar.deriv()
tabvar.compare_values()
messageDisplay()

. Fonction.s appelante.s

checkAll()

. Définition des arguments en entré

env: environnement actuel.

. Retour théorique

→ [valid, message]

. Définition des arguments en sortie

valid : indique l'état de la validation
expr : (string) Expression de la fonction valider.

. Exemple d'utilisation

validation_funDeriv(env)
*→ [2, env, "10*x"]*

. Retour error

validation_funDeriv(env)
→ [1.9, "Erreur interne, veuillez nous excuser"]

validation_subFun()

. Description

Permet de valider les expressions des sous fonctions qui ont été ajoutées. Et coloris les sous fonction en fonction de leur validées, si une fonction n'est pas valide alors elle sera coloriage en rose et sinon en vert. De plus, si il n'y a pas de sous fonction alors le message "Pas de sous fonction" apparaîtra sinon il sera affiché uniquement les fonctions valides.

. Fonction.s appelée.s

tabvar.validate()
messageDisplay()

. Fonction.s appelante.s

checkAll()

. Définition des arguments en entré

env: environnement actuel.

. Retour théorique

→ *[valid, message]*

. Définition des arguments en sortie

valid : indique l'état de la validation

message : "Sous fonction(s) validée(s)" si les dérivé existantes sont passées par la validation sans erreur et de meme si il n'y a pas de sous fonction, "Element inexistant" si l'élément n'existe pas, sinon "Erreur interne, veuillez nous excuser".

. Exemple d'utilisation

validation_subFun(env)

→ *[2, env, "10*x"]*

. Retour error

validation_subFun(env)

→ *[1.9, "Erreur interne, veuillez nous excuser"]*

compareValuesInRow(env)

. Description

Permet de comparer les valeurs de "x" deux a deux. Et colorier ces valeurs en fonction de leur résultat de comparaison. Si la premiere valeur est inferieur a la seconde a lors la derniere sera coloriage en rose, si elle est égale ou que l'le module tabvar peut lire la valeur mais peut pas la compérer alors le seconde valeur sera coloriage en jaune. Si la première est inférieure à la deuxième alors la plus grande sera en vert et sinon si la valeur n'est pas reconnue alors les deux seront en rouge.

. Fonction.s appelée.s

tabvar.compare_values()

. Fonction.s appelante.s

checkAll()

. Définition des arguments en entré

env: environnement actuel.

. Retour théorique

→ *[valid, message]*

. Définition des arguments en sortie

valid : indique l'état de la validation

message : "Les valeurs ont étaient comparées" si les valeurs présentent dans la premiere ligne du tableau ont bien été toutes comparées, "Element inexistant" si l'élément n'existe pas, sinon "Erreur interne, veuillez nous excuser".

. Exemple d’utilisation

compareValuesInRow(env)

*→ [2, env, "10*x"]*

. Retour error

compareValuesInRow(env)

→ [1.9, "Erreur interne, veuillez nous excuser"]

3. Gestion de la zone textuelle.

Le module "gestion de la zone textuelle" a pour objectif de gérer les interactions de l'utilisateur avec une zone de texte dans une interface utilisateur. Il fournit des suggestions de complétion automatique pour les commandes saisies, de gérer les événements liés à l'input et à la validation des commandes, ainsi que d'exécuter des fonctions spécifiques définies dans un tableau de méthodes (tabvarMethods1).

3.1 Relations d'utilisation avec d'autres modules

Ce module interagit principalement avec le module tabvar, qui contient les définitions des fonctions à exécuter lors de la validation des commandes. Il utilise également des bibliothèques externes comme jQuery pour manipuler le DOM et gérer les événements, ainsi que jQuery UI pour la fonctionnalité de complétion automatique.

3.2 Modules utilisés par ce module et ceux utilisant ce module

Modules utilisés par ce module :

- tabvar: Ce module contient les fonctions à exécuter pour les différentes commandes saisies par l'utilisateur.
- jQuery: Pour la manipulation du DOM et la gestion des événements.
- jQuery UI: Pour la fonctionnalité d'autocomplétion.

Modules utilisant ce module :

- Interface Utilisateur: Ce module est utilisé par l'interface utilisateur pour gérer la saisie et la validation des commandes dans une zone de texte.

3.3 Procédures

Exécute la commande saisie lorsque l'utilisateur appuie sur la touche Entrée. Vérifie si la commande est valide et l'exécute en appelant ExecutApi(command).

parseFunctionString(funcStr):

. Description

Cette fonction analyse une chaîne de caractères représentant une fonction et ses paramètres, et retourne un objet contenant le nom de la fonction et ses paramètres.

. Fonction.s appelante.s

ExecutApi(command)

. Définition des arguments en entrée

funcStr : La chaîne de caractères représentant la fonction et ses paramètres (par exemple, "fonction(param1, param2)").

. Retour théorique

Un objet contenant le nom de la fonction et un tableau de paramètres.

. Définition des valeurs en sortie

null : si la chaîne ne correspond pas au modèle attendu.
{ functionName: String, params: Array } : si la chaîne correspond au modèle.

. Exemple d'utilisation

```
const result = parseFunctionString("function_name(param1,param2)");
console.log(result);
→ { functionName: "function_name", params: ["param1", "param2"] }
```

. Retour erreur

Retourne null en cas de chaîne mal formée.

ExecutApi(command):

. Description

Cette fonction exécute une commande saisie dans la zone de texte en appelant la fonction correspondante dans le module tabvar. Elle utilise la fonction parseFunctionString pour analyser la commande.

. Fonction.s appelée.s

parseFunctionString(funcStr)
insertAtSecondLine(text)
Fonctions du module tabvar

. Fonction.s appelante.s

onEnter(event)

. Définition des arguments en entrée

command : La commande à exécuter, incluant le nom de la fonction et ses paramètres (par exemple, "init()", "validate(param1, param2)").

. Retour théorique

Aucun (la fonction exécute des actions en réponse à la commande).

. Définition des valeurs en sortie

La fonction appelle la méthode appropriée de tabvar et insère le résultat à la deuxième ligne de la zone de texte.

. Exemple d'utilisation

ExecutApi("validate(x, y)");

. Retour erreur

→ [1.1, Erreur : Paramètres manquants]

insertAtSecondLine(text):

. Description

Cette fonction insère du texte à la deuxième ligne de la zone de texte spécifiée (#command-textarea). Si la deuxième ligne n'existe pas, elle est créée. La deuxième ligne est non-éditable par l'utilisateur.

. Fonction.s appelante.s

ExecutApi(command)

. Définition des arguments en entrée

text : Le texte à insérer à la deuxième ligne.

. Retour théorique

La fonction modifie directement le contenu de la zone de texte.

onEnter(event):

. Description

Cette fonction est déclenchée lorsqu'un utilisateur appuie sur la touche Entrée dans la zone de texte. Elle vérifie si la commande est valide et l'exécute en appelant *ExecutApi(command)*.

. Fonction.s appelée.s

ExecutApi(command)

. Fonction.s appelante.s

Gestionnaire d'événement de la zone de texte (#command-textarea).

. Définition des arguments en entrée

event : L'événement de touche.

. Retour théorique

La fonction exécute des actions en réponse à l'événement.

. Exemple d'utilisation

Utilisée comme gestionnaire d'événement.

. Retour erreur

→ *[1.1 Erreur : Commande inexistante.];*

4. Gestion de la zone dédiée à l’aide sur les fonctions.

Ce module a pour objectif de fournir des descriptions détaillées et des informations d'utilisation pour les différentes commandes disponibles. Il permet d’afficher le descriptif de l’API.

4.1 Relations d’utilisation avec d’autres modules

Ce module utilise des bibliothèques externes comme jQuery pour manipuler le DOM et gérer les événements, ainsi que jQuery UI pour la fonctionnalité de complétion automatique.

4.2 Modules utilisés par ce module et ceux utilisant ce module

Modules utilisés par ce module :

- jQuery: Pour la manipulation du DOM et la gestion des événements.
- jQuery UI: Pour la fonctionnalité d'autocomplétion.

Modules utilisant ce module :

- Interface Utilisateur: Ce module est utilisé par l'interface utilisateur pour gérer la saisie et la validation des commandes dans un champ de saisie, ainsi que pour afficher les descriptions d'aide correspondantes.

4.3 Procédures

showAide():

. Description

Affiche des informations détaillées sur une commande spécifique en fonction du texte saisi dans la zone de texte *#command-input.*

. Fonctions appelantes

Gestionnaires d'événements.

. Arguments en entrée

Valeur actuelle de *#command-input.*

. Retour erreur

→ *[1,1, Erreur : Commande inexistante.]*