

# Le grand livre de eFORTH web

version 1.2 - 30 octobre 2023



Auteur(s)

- Marc PETREMANN

Collaborateur(s)

- xxx

## Table des matières

<b>Premiers pas avec eFORTH web.....</b>	<b>4</b>
Premiers mots et premières définitions.....	4
Voir les autres vocabulaires.....	4
Première compilation de code FORTH.....	5
Sélection mode texte ou graphique.....	5
<b>Dictionnaire / Pile / Variables / Constantes.....</b>	<b>7</b>
Étendre le dictionnaire.....	7
Gestion du dictionnaire.....	7
Piles et notation polonaise inversée.....	8
Manipulation de la pile de paramètres.....	9
La pile de retour et ses utilisations.....	9
Utilisation de la mémoire.....	10
Variables.....	10
Constantes.....	11
Valeurs pseudo-constantes.....	11
Outils de base pour l'allocation de mémoire.....	11
<b>Les nombres entiers avec eFORTH.....</b>	<b>13</b>
Empilage et dépilage des nombres entiers.....	13
Opérations arithmétiques élémentaires.....	14
Somme de deux nombres entiers.....	14
Soustraction de deux nombres entiers.....	15
Produit de deux nombres entiers.....	15
Quotient de deux nombres entiers.....	15
Produit et quotient de trois nombres.....	16
Traitement des expressions algébriques.....	16
Manipulation des données sur la pile.....	17
Passage de paramètres par la pile de retour.....	18
<b>Import de fichiers vers eFORTH web.....</b>	<b>20</b>
Edition des fichiers sources pour eFORTH.....	20
Chargement d'un fichier source vers eForth.....	20
<b>Créez des définitions FORTH en utilisant JavaScript.....</b>	<b>22</b>
Le mariage Javascript Forth.....	22
Structure d'une définition de mot avec JSWORD:.....	22
Passage des paramètres entre eFORTH et JavaScript.....	23
Extension du vocabulaire web de eFORTH.....	25
<b>Initiation au graphisme avec eFORTH pour le web.....</b>	<b>27</b>
Comprendre l'environnement graphique.....	27
Notion de tracé et remplissage.....	27
Rotations et translations.....	29
Cas pratique: tracé des heures d'une horloge.....	33
<b>Gestion des images avec eFORTH pour le web.....</b>	<b>35</b>
Chargement et affichage des images.....	35

Affichage et découpage d'une image au format jpg.....	37
Tester la couleur d'un pixel image.....	38
<b>Contenu détaillé des vocabulaires eFORTH WEB.....</b>	<b>41</b>
Version v 7.0.7.15.....	41
FORTH.....	41
ansi.....	42
asm.....	42
internalized.....	42
internals.....	42
structures.....	43
tasks.....	43
web.....	43

# Premiers pas avec eFORTH web

## Premiers mots et premières définitions

Si vous n'avez pas installé eFORTH web, vous pouvez tester notre version en ligne. Lien :

<https://eforth.arduino-forth.com/web-eforth/index/>

Pour constater le bon fonctionnement de eFORTH, tapez **words** dans l'espace de travail eFORTH:

### Essayez eFORTH avec la version en ligne

```
uEforth v7.0.7.15 - rev 564a8fc68b545ebeb3ab
Forth dictionary: 4065196 free + 79456 used = 4144652 total (98% free)
3 x Forth stacks: 16384 bytes each
ok
--> words
FORTH ok colors ms start-task task pause tasks ENDOF OF ENDCASE CASE +to
to exit ; { (local) asm words vlist order see .s startswith? str= :noname
forget dump spaces assert set-title page at-xy normal bg fg ansi ms-ticks
web structures f.s f. #fs set-precision precision fvariable fconstant fliteral
afliteral sf, internals sealed previous also only transfer{ }transfer transfer
definitions vocabulary [IF] [ELSE] [THEN] DEFINED? quit evaluate prompt
refill tib accept echo abort" abort z>s s>z r~ r| r" z" ." s" n. ? . u.
binary decimal octal hex str #> sign #s # hold <# extract pad hld cr space
emit bye terminate key? key type fill32 >name is defer +to to value throw
catch handler K J I loop +loop leave UNLOOP ?do do next for nest-depth
postpone recurse aft repeat while else if then ahead until again begin
[char] char ['] ' used remaining fdepth depth fp0 rp0 sp0 >link >link&
>size >params >name-length >flags >flags& align aligned #! \ ( CALL FP@
FP! SF@ SF! FDUP FNIP FDROP FOVER FSWAP FROT FNEGATE F0< F0= F= F< F> F<>
```

L'exécution de **words** affiche le contenu du vocabulaire FORTH. Tous ces mots sont l'équivalent de fonctions dans d'autres langages. FORTH se différencie de tous les autres langages de programmation, car en FORTH toute nouvelle définition de mot FORTH agrandit le langage FORTH. Il n'y a pas de différence entre une application en FORTH et l'extension du dictionnaire.

Pour effacer le contenu de l'espace de travail, exécutez le mot **page**.

## Voir les autres vocabulaires

**words** affiche le contenu du vocabulaire FORTH. Mais eFORTH web embarque d'autres vocabulaires. Pour voir la liste de ces vocabulaires, tapez **internals voclist**. Affiche:

```
--> internals voclist
tasks
ansi
web
structures
internalized
internals
FORTH
ok
```

```
-->
```

Parmi ces vocabulaires, celui qui va nous intéresser est le vocabulaire **web**. Pour voir les définitions de ce vocabulaire, taper **web vlist**. Affiche:

```
--> web vlist
yielding-task yielding import rm ls download cat include-file upload upload-file
session? web-key? web-key web-type scripts scripts# random button mouse
textWidth fillText font text-size! log raw-http-upload http-download raw-download
upload-success? upload-done? upload-start ms-ticks silence tone importScripts
release keyCount getKey clearItems removeItem getItem setItem smooth gpop
gpush rotate scale translate show-text keys-height mobile textRatios viewport@
window line fill stroke lineTo moveTo beginPath box lineWidth color! text
gr grmode shouldEcho? web-terminate web-key?-raw web-key-raw web-type-raw
jseval JSWORD: jsslot jseval!
ok
```

Un vocabulaire permet d'intégrer des mots utilisables dans un certain contexte. Ici, le contenu du vocabulaire web n'est défini que pour la version eFORTH web. Le vocabulaire web n'existe pas sur les autres versions de eFORTH (Windows, Linux, ESP32...).

## Première compilation de code FORTH

Si une définition FORTH est courte, vous pouvez la compiler immédiatement depuis l'invite de commande de eFORTH web:

```
: myLoop 10 0 do i . loop ;
```

Tapez ensuite **myLoop**, ce qui exécutera le code qui vient d'être compilé par eFORTH web:

```
uEforth v7.0.7.9 - rev fb3db70da6d111b1fdf0
Forth dictionary: 4068200 free + 76708 used = 4144908 total (98% free)
3 x Forth stacks: 16384 bytes each
ok
--> : myLoop 10 0 do i . loop ;
ok
--> myloop
0 1 2 3 4 5 6 7 8 9 ok
-->
```

## Sélection mode texte ou graphique

Le mot **gr** sélectionne le mode graphique. Ce mot rend visible un espace graphique de type canvas:

```
web gr
```

Pour revenir au mode text, taper **text**.

Voici un exemple de graphisme:

```
web
: grTest ( -- )
  gr 400 300 window
  $000000 color!
```

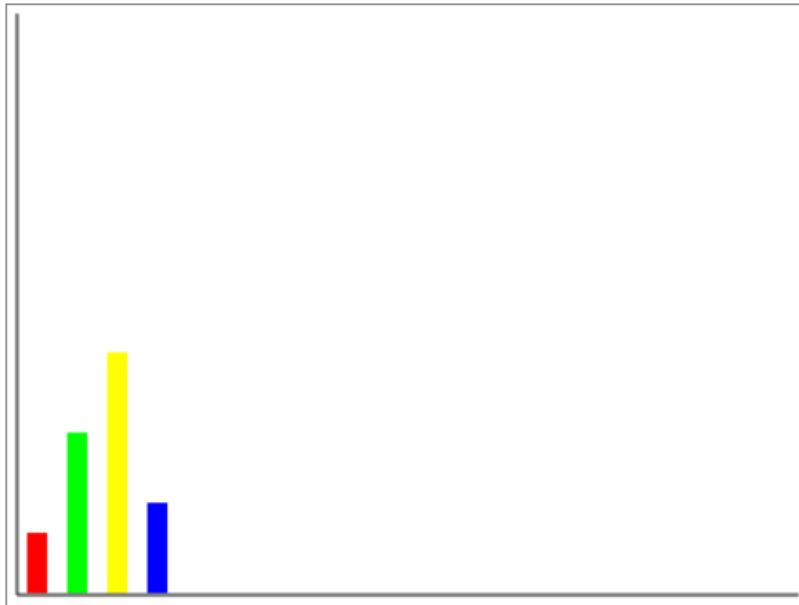
```

5 295 395 295 line
5 295 5 5 line
$ff0000 color! 10 294 10 -30 box
$00ff00 color! 30 294 10 -80 box
$ffff00 color! 50 294 10 -120 box
$0000ff color! 70 294 10 -45 box
key drop
text
;

```

L'exécution de **grTest** affiche ce graphisme:

## Try eFORTH



uForth v7.0.7.9 - rev fh3dh70da6d111h1fdf0

# Dictionnaire / Pile / Variables / Constantes

## Étendre le dictionnaire

Forth appartient à la classe des langages d'interprétation tissés. Cela signifie qu'il peut interpréter les commandes tapées sur la console, ainsi que compiler de nouveaux sous-programmes et programmes.

Le compilateur Forth fait partie du langage et des mots spéciaux sont utilisés pour créer de nouvelles entrées de dictionnaire (c'est-à-dire des mots). Les plus importants sont **:** (commencer une nouvelle définition) et **;** (termine la définition). Essayons ceci en tapant:

```
: *+ * + ;
```

Ce qui s'est passé? L'action de **:** est de créer une nouvelle entrée de dictionnaire nommée **\*+** et passer du mode interprétation au mode compilation. En mode compilation, l'interpréteur recherche les mots **et**, plutôt que de les exécuter, installe des pointeurs vers leur code. Si le texte est un nombre, au lieu de le pousser sur la pile, ESP32forth construit le nombre dans le dictionnaire l'espace alloué pour le nouveau mot, suivant le code spécial qui met le numéro stocké sur la pile chaque fois que le mot est exécuté. L'action d'exécution de **\*+** est donc d'exécuter séquentiellement les mots définis précédemment **\*** et **+**.

Le mot **;** est spécial. C'est un mot immédiat et il est toujours exécuté, même si le système est en mode compilation. Ce que fait **;** est double. Tout d'abord, il installe le code qui renvoie le contrôle au niveau externe suivant de l'interpréteur et, deuxièmement, il revient du mode compilation au mode interprétation.

Maintenant, essayez votre nouveau mot :

```
decimal 5 6 7 *+ . \ affiche 47 ok<#,ram>
```

Cet exemple illustre deux activités principales de travail dans Forth: ajouter un nouveau mot au dictionnaire, et l'essayer dès qu'il a été défini.

## Gestion du dictionnaire

Le mot **forget** suivi du mot à supprimer enlèvera toutes les entrées de dictionnaire que vous avez faites depuis ce mot:

```
: test1 ;  
: test2 ;  
: test3 ;  
forget test2 \ efface test2 et test3 du dictionnaire
```

## Piles et notation polonaise inversée

Forth a une pile explicitement visible qui est utilisée pour passer des nombres entre les mots (commandes). Utiliser Forth efficacement vous oblige à penser en termes de pile. Cela peut être difficile au début, mais comme pour tout, cela devient beaucoup plus facile avec la pratique.

En FORTH, La pile est analogue à une pile de cartes avec des nombres écrits dessus. Les nombres sont toujours ajoutés au sommet de la pile et retirés du sommet de la pile. ESP32forth intègre deux piles: la pile de paramètres et la pile de retour, chacune composée d'un certain nombre de cellules pouvant contenir des nombres de 16 bits.

La ligne d'entrée FORTH:

```
decimal 2 5 73 -16
```

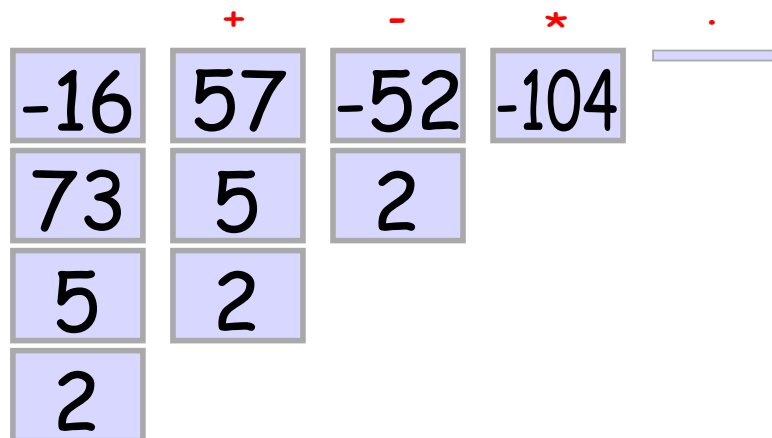
laisse la pile de paramètres dans l'état

Cellule	contenu	commentaire
0	-16	(TOS) Sommet pile
1	73	(NOS) Suivant dans la pile
2	5	
3	2	

Nous utiliserons généralement une numérotation relative à base zéro dans les structures de données Forth telles que piles, tableaux et tables. Notez que, lorsqu'une séquence de nombres est saisie comme celle-ci, le nombre le plus à droite devient *TOS* et le nombre le plus à gauche se trouve au bas de la pile.

Supposons que nous suivions la ligne d'entrée d'origine avec la ligne

```
+ - * .
```



Les opérations produiraient les opérations de pile successives:

Après les deux lignes, la console affiche :

```
decimal 2 5 73 -16 \ affiche: 2 5 73 -16 ok
+ - * .           \ affiche: -104 ok
```



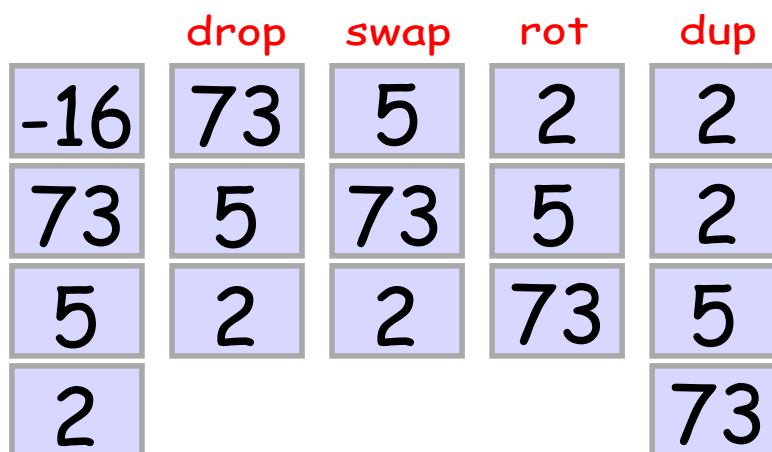
Notez que ESP32forth affiche commodément les éléments de la pile lors de l'interprétation de chaque ligne et que la valeur de -16 est affichée sous la forme d'entier non signé 32 bits. En outre, le mot `.` consomme la valeur de données -104, laissant la pile vide. Si nous exécutons `.` sur la pile maintenant vide, l'interpréteur externe abandonne avec une erreur de pointeur de pile `STACK UNDERFLOW ERROR`.

La notation de programmation où les opérandes apparaissent en premier, suivis du ou des opérateurs est appelée Notation polonaise inverse (RPN).

## Manipulation de la pile de paramètres

Étant un système basé sur la pile, ESP32forth doit fournir des moyens de mettre des nombres sur la pile, pour les supprimer et réorganiser leur ordre. On a déjà vu qu'on peut mettre des nombres sur la pile simplement en les tapant. Nous pouvons également intégrer les nombres dans la définition d'un mot FORTH.

Le mot **drop** supprime un numéro du sommet de la pile mettant ainsi le suivant au sommet. Le mot **swap** échange les 2 premiers numéros. **dup** copie le nombre au sommet, poussant tout les autres numéros vers le bas. **rot** fait pivoter les 3 premiers nombres. Ces



actions sont présentées ci-dessous.

## La pile de retour et ses utilisations

Lors de la compilation d'un nouveau mot, ESP32forth établit des liens entre le mot appelant et les mots définis précédemment qui doivent être invoqués par l'exécution du nouveau mot. Ce mécanisme de liaison, lors de l'exécution, utilise la pile de retour (rstack). L'adresse du mot suivant à invoquer est placée sur la pile de retour de sorte que, lorsque le mot courant est terminé en cours d'exécution, le système sait où passer au mot suivant. Comme les mots peuvent être imbriqués, il doit y avoir une pile de ces adresses de retour.

En plus de servir de réservoir d'adresses de retour, l'utilisateur peut également stocker et récupérer à partir de la pile de retour, mais cela doit être fait avec soin car la pile de retour est essentielle à l'exécution du programme. Si vous utilisez la pile de retour pour le

stockage temporaire, vous devez la remettre dans son état d'origine, sinon vous ferez probablement planter le système ESP32forth. Malgré le danger, il y a des moments où l'utilisation de pile de retour comme stockage temporaire peut rendre votre code moins complexe.

Pour stocker dans la pile, utilisez **>r** pour déplacer le sommet de la pile de paramètres vers le haut de la pile de retour. Pour récupérer une valeur, **r>** déplace la valeur supérieure de la pile de retour vers le sommet de la pile de paramètres. Pour supprimer simplement une valeur du haut de la pile, il y a le mot **rdrop**. Le mot **r@** copie le haut de la pile de retour dans la pile de paramètres.

## Utilisation de la mémoire

Dans eForth WEB, les nombres 32 bits sont extraits de la mémoire vers la pile par le mot **@** (fetch) et stocké du sommet à la mémoire par le mot **!** (store). **@** attend une adresse sur la pile et remplace l'adresse par son contenu. **!** attend un nombre et une adresse pour le stocker. Il place le numéro dans l'emplacement de mémoire référencé par l'adresse, consommant les deux paramètres dans le processus.

Les nombres non signés qui représentent des valeurs de 8 bits (octets) peuvent être placés dans des caractères de la taille d'un caractère. cellules de mémoire en utilisant **c@** et **c!**.

```
create testVar
  cell allot
  $f7 testVar c!
testVar c@ . \ affiche 247
```

## Variables

Une variable est un emplacement nommé en mémoire qui peut stocker un nombre, tel que le résultat intermédiaire d'un calcul, hors de la pile. Par exemple:

```
variable x
```

crée un emplacement de stockage nommé, **x**, qui s'exécute en laissant l'adresse de son emplacement de stockage au sommet de la pile:

```
x . \ affiche l'adresse
```

Nous pouvons alors aller chercher ou stocker à cette adresse :

```
variable x
3 x !
x @ . \ affiche: 3
```

## Constantes

Une constante est un nombre que vous ne voudriez pas changer pendant l'exécution d'un programme. Le résultat de l'exécution du mot associé à une constante est la valeur des données restant sur la pile.

```
\ définit les pins VSPI
19 constant VSPI_MISO
23 constant VSPI_MOSI
18 constant VSPI_SCLK
05 constant VSPI_CS

\ définit la fréquence du port SPI
4000000 constant SPI_FREQ

\ sélectionne le vocabulaire SPI
only FORTH SPI also

\ initialise le port SPI
: init.VSPI ( -- )
  VSPI_CS OUTPUT pinMode
  VSPI_SCLK VSPI_MISO VSPI_MOSI VSPI_CS SPI.begin
  SPI_FREQ SPI.setFrequency
;
```

## Valeurs pseudo-constantes

Une valeur définie avec `value` est un type hybride de variable et constante. Nous définissons et initialisons une valeur et est invoquée comme nous le ferions pour une constante. On peut aussi changer une valeur comme on peut changer une variable.

```
decimal
13 value thirteen
thirteen .      \ display: 13
47 to thirteen
thirteen .      \ display: 47
```

Le mot **to** fonctionne également dans les définitions de mots, en remplaçant la valeur qui le suit par tout ce qui est actuellement au sommet de la pile. Vous devez faire attention à ce que **to** soit suivi d'une valeur définie par **value** et non d'autre chose.

## Outils de base pour l'allocation de mémoire

Les mots **create** et **allot** sont les outils de base pour réserver un espace mémoire et y attacher une étiquette. Par exemple, la transcription suivante montre une nouvelle entrée de dictionnaire **graphic-array** :

```
create graphic-array ( --- addr )
  %00000000 c,
  %00000010 c,
```

```
%00000100 c,  
%00001000 c,  
%00010000 c,  
%00100000 c,  
%01000000 c,  
%10000000 c,
```

Lorsqu'il est exécuté, le mot **graphic-array** poussera l'adresse de la première entrée.

Nous pouvons maintenant accéder à la mémoire allouée à **graphic-array** en utilisant les mots de récupération et de stockage expliqués plus tôt. Pour calculer l'adresse du troisième octet attribué à **graphic-array** on peut écrire **graphic-array 2 +**, en se rappelant que les indices commencent à 0.

```
30 graphic-array 2 + c!  
graphic-array 2 + c@ .      \ affiche 30
```

# Les nombres entiers avec eFORTH

## Empilage et dépilage des nombres entiers

Le langage FORTH stocke les nombres sur une pile nommée pile de données ou pile paramétrique. L'empilage d'un nombre est très simple:

```
55
```

empile le nombre 55. Pour dépiler ce nombre, il y a plusieurs méthodes. La plus simple consiste à afficher le contenu du dernier élément empilé. Le mot `.` (point) affiche 55.

```
55      \ display ok
.       \ display 55 ok
```

Si vous déposez plusieurs nombres sur la pile de données, voici ce qui se passe:

```
11 45 6543
```

Les nombres sont mis sur la pile de données dans l'ordre de frappe. Le dernier nombre entré est toujours au sommet de la pile:

valeur empilée	haut -- pile -- fond
11	11
45	45 11
6543	6543 45 11

Le dépilage successif des nombres affiche ceux-ci dans l'ordre inverse de leur empilage:

```
11 45 6543      \ display ok
.               \ display 6543 ok
.               \ display 45  ok
.               \ display 11  ok
```

Pour mieux visualiser le mécanisme d'empilage et de dépilage des nombres, pensez à une pile d'assiettes: la dernière assiette déposée sur la pile sera la première reprise.

A tout moment vous pouvez prendre connaissance du contenu de la pile de données sans avoir à provoquer le dépilage des valeurs qui y sont stockées en utilisant le mot `.S`:

```
1 2 3 .S affiche 1 2 3
```

Ce principe de rangement est appelé également pile LIFO (Last In, First Out) dans certains ouvrages écrits en anglais pour désigner une pile dont le mécanisme est: "dernier entré, premier sorti".

Avec la majorité des versions du langage FORTH, la quantité de nombres empilables est assez élevée, mais reste limitée. Si vous empilez trop de nombres, vous saturerez la pile de données. De même, toute tentative pour dépiler un nombre alors que la pile de données est vide, affichera un message d'erreur.

Chaque nombre empilé ne peut être qu'un nombre entier au format simple précision. Selon les cas, ce nombre peut être considéré signé ou non signé:

- signé ou non signé: paramètre de calcul arithmétique
- non signé: adresse simple précision

En format simple précision signé, le bit de poids le plus fort indique le signe du nombre.

La valeur -1, déposée sur la pile en tant que valeur entière a la représentation binaire suivante:

```
11111111111111111111111111111111 (simple précision 32 bits) .
```

Cette même valeur peut être affichée en valeur absolue en utilisant le mot **U.** à la place du mot **.** (point):

```
-1 U. \ affiche 4294967295
```

## Opérations arithmétiques élémentaires

Les opérateurs arithmétiques **+** **-** **\*** et **/** agissent sur les deux valeurs situées au sommet de la pile de données. Les valeurs traitées sont toujours des entiers simple précision signés.

### Somme de deux nombres entiers

Pour additionner deux nombres, il faut d'abord les déposer sur la pile de données:

```
22 44 + . \ affiche 66
```

Une fois empilés 22 et 44, le mot **+** opère l'addition de ces deux valeurs et le mot **.** affiche le résultat:

```
55 1 + 3 + . \ affiche 59 et peut aussi s'écrire  
55 1 3 + + .
```

L'addition est commutative: les valeurs peuvent être déposées sur la pile de données dans n'importe quel ordre:

```
55 22 +  
22 55 +
```

Ce principe de calcul est appelé NOTATION POLONAISE INVERSE (RPN dans la littérature anglaise, pour Reverse Polish Notation). On peut aussi additionner deux nombres entiers non signés, à condition de visualiser le résultat à l'aide du mot **U.** au lieu de **.**:

```
35000 10 + U.
```

Il est également possible de faire la somme de deux nombres de signe différent:

```
10 -5 + . \ affiche 5  
-5 10 + . \ affiche aussi 5
```

FORTH dispose également du mot **1+** qui incrémente la valeur située au sommet de la pile de données de 1 unité:

```
10 1+      \ est équivalent à
10 1 +
```

## Soustraction de deux nombres entiers

Soit deux nombres a et b. La différence de deux nombres sera écrite en FORTH sous la forme:

```
a b - pour a-b
```

La soustraction n'est pas commutative:

```
10 3 - .      \ affiche      7
3 10 - .      \ affiche     -7
```

Le mot **1-** décrémente la valeur située au sommet de la pile de données de 1 unité:

```
10 1-      \ est équivalent à
10 1 -
```

## Produit de deux nombres entiers

Soit deux nombres a et b. Le produit de deux nombres sera écrit en FORTH sous la forme:

```
a b *      \ pour a*b
```

La multiplication est commutative:

```
7 5 * .      \ ou
5 7 * .      \ affiche    35
```

Le mot **2\*** multiplie la valeur située au sommet de la pile de données par deux:

```
5 2*      \ est équivalent à
5 2 *
```

## Quotient de deux nombres entiers

Pour la division, seul le quotient entier est conservé sur la pile de données:

```
22 7 / .      \ affiche    3
```

La division n'est pas commutative:

```
15 5 / .      \ affiche    3
5 15 / .      \ affiche    0
```

Le reste de la division peut être obtenu en appliquant la fonction modulo:

```
22 7 MOD .      \ affiche    1
```

La fonction modulo peut servir à déterminer la divisibilité d'un nombre par un autre:

```

: DIV? ( n1 n2 ---)
  OVER OVER MOD CR
  IF
    SWAP . ." n'est pas "
  ELSE
    SWAP . ." est "
  THEN
    ." divisible par " . ;

```

Le mot **/MOD** combine les actions de **/** et de **MOD**:

```

22 7 /MOD . . \ affiche 3 1

```

## Produit et quotient de trois nombres

Le mot **\*/** combine les opérations de multiplication et de division.

Exemple, soit à calculer le prix TTC d'une marchandise (TVA à 20 %), on définira le mot TTC comme suit:

```

: TTC ( n1 --- n2)
  DUP 200 1000 */ + ;
442 TTC . \ affiche 530

```

Les valeurs traitées étant exprimées en centimes.

Le mot **\*/MOD** a les mêmes propriétés que **\*/**, mais délivre le quotient et le reste de l'opération.

A titre d'exemple, pour donner une application immédiate et pratique des notions déjà exprimées, est la conversion des degrés Fahrenheit et Celsius:

- la conversion des degrés Fahrenheit en degrés Celsius obéit à la formule  
 $^{\circ}\text{C} = (^{\circ}\text{F} - 32) * 5/9$
- la conversion des degrés Celsius en degrés Fahrenheit obéit à la formule  
 $^{\circ}\text{F} = 9/5 * ^{\circ}\text{C} + 32$

```

: C>F ( °C --- °F)
  9 5 */ 32 + ;
: F>C ( °F --- °C)
  32 - 5 9 */ ;
37 C>F . \ affiche 98 (les résultats sont arrondis)

```

Cet exemple fonctionne sur toutes les versions du langage FORTH.

## Traitement des expressions algébriques

Les opérations peuvent être chaînées, mais une opération en notation algébrique comportant des parenthèses doit être convertie en notation RPN en tenant compte de



l'ordre de priorité des opérations. FORTH n'utilise pas les parenthèses dans les opérations arithmétiques:

- soit l'expression algébrique  $(2 + 5) * (7 - 2)$
- elle s'écrit en FORTH `2 5 + 7 2 - *`

Lors d'une opération de conversion de notation algébrique infixée en notation polonaise inverse, commencez toujours par le niveau de parenthèse le plus imbriqué et par la gauche. Ecrivez la transcription en notation polonaise inverse de chaque opération sur des lignes séparées, successivement de haut en bas, en les mettant dans le prolongement de l'expression algébrique exprimée dans la formule initiale.

C'est choquant? Mais tous les interpréteurs/compilateurs travaillent ainsi lorsqu'ils ont à évaluer une formule algébrique. En notation algébrique, les parenthèses ne servent qu'à isoler une expression sous forme de sous-expression qui devient membre d'une expression plus générale.

En informatique comme en arithmétique, un opérateur travaille toujours sur deux opérandes et seulement deux opérandes simultanément. Le résultat d'une opération portant sur deux opérandes délivre une valeur qui peut devenir à son tour opérande d'un autre opérateur. L'ordre d'exécution des opérandes et des opérateurs est fondamental:

notation algébrique	polonaise inverse
$(2+3)*5$	2 3 + 5 *
$2+(3*5)$	2 3 5 * + or 3 5 * 2 +

Tous les problèmes arithmétiques peuvent être résolus de cette manière, ce n'est qu'une question d'habitude. L'exemple donné précédemment illustre parfaitement la rigueur dont doit faire preuve le programmeurs Forth. Cette rigueur garantit un fonctionnement sans ambiguïté des programmes, quel que soit leur niveau de complexité.

## Manipulation des données sur la pile

La pile de données est l'élément fondamental du langage FORTH pour le traitement de données. Son fonctionnement est identique à celui de la pile gérée par le micro-processeur. Dans certaines situations, les données traitées par les différentes définitions doivent être réordonnées ou dupliquées.

Le mot **DUP** duplique le contenu du sommet de la pile de données:

```
10 DUP . . \ affiche 10 10
5 DUP * . \ affiche 25
```

Le mot **OVER** duplique le second élément de la pile de données:

```
5 15 OVER . . . \ affiche 5 15 5
```

Le mot **SWAP** inverse les deux éléments du sommet de la pile de données:

```
1 3 SWAP . . \ affiche 1 3
```

Le mot **ROT** effectue une rotation sur les trois éléments situés au sommet de la pile de données:

```
1 2 3 ROT . . . \ affiche 1 3 2
```

Voici quelques exemples d'utilisation de ces manipulateurs de pile de données:

```
: AU-CARRE ( n --- n2)
  DUP * ;
```

Le mot **AU-CARRE** élève un nombre entier quelconque au carré:

```
2 AU-CARRE . \ affiche 4
3 AU-CARRE . \ affiche 9
4 AU-CARRE . \ affiche 16

: AU-CUBE ( n --- n3)
  DUP DUP * * ;
```

Le mot **AU-CUBE** élève un nombre entier quelconque au cube:

```
2 AU-CUBE . \ affiche 8
3 AU-CUBE . \ affiche 27
```

Attention, n'utilisez pas des valeurs trop élevées, car un résultat dépassant 2 puissance 32, pour une pile de données 32 bits, devient faux. Ayez toujours à l'esprit que les données traitées sont des entiers, donc de capacité limitée.

## Passage de paramètres par la pile de retour

A côté de la pile de paramètres, il existe dans FORTH une deuxième pile, appelée pile de retour parce qu'elle sert à l'interpréteur interne à retrouver l'adresse de retour à chaque appel d'une procédure.

Il y a parfois des cas extrêmes où on peut être amené à stocker un ou plusieurs paramètres ailleurs que sur la pile de données, ceci pour simplifier quelque peu certaines manipulations scabreuses. La solution la plus commode, parmi d'autres, est la pile de retour. Cette pile est accessible par les mots **>R** et **R>** moyennant quelques précautions pour ne pas compromettre le fonctionnement de cette pile interne.

Le mot **>R** transfère un nombre entier de la pile de données vers la pile de retour.

Le mot **R>** transfère un nombre entier de la pile de retour vers la pile de données.

Une opération **>R R>** est nulle. En fin de définition, il doit y avoir autant de **>R** que de **R>** sous peine de perturber quelque peu le déroulement normal de votre définition. Exemple d'utilisation:

```
: AU-CARRE ( n --- n^2)          \ élévation au carré
  DUP >R                        \ transfert de la copie sur pile de
retour                           retour
  CR ." Le carré de " .         \ affichage valeur initiale
```

<code>."</code> est " <code>R&gt; DUP * . ;</code> retour	<code>\ récupération valeur déposée sur pile</code> <code>\ et affichage de son carré</code>
--------------------------------------------------------------	-------------------------------------------------------------------------------------------------

## Import de fichiers vers eFORTH web

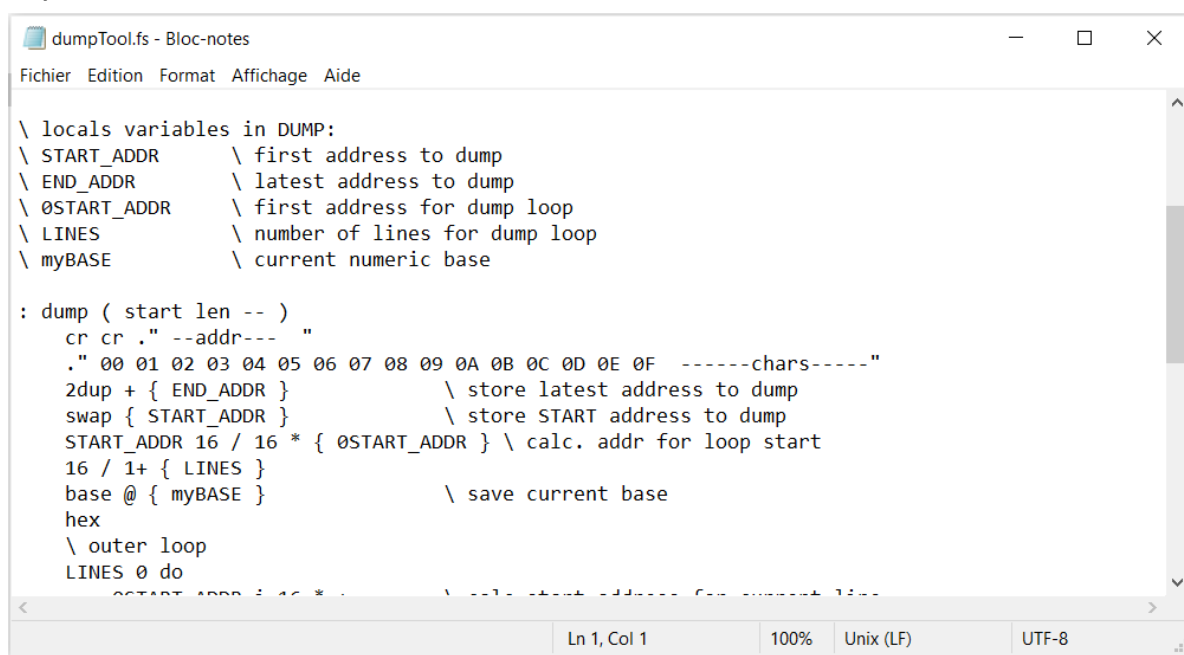
Vous avez installé eFORTH sur votre site web? Ou en local au pire des cas?

OK.

Voyons donc les choses sérieuses. FORTH est intéressant si on développe de vrais programmes.

## Edition des fichiers sources pour eFORTH

Les fichiers sources pour eFORTH peuvent être édités avec n'importe quel éditeur texte: Wordpad, bloc-notes...



```
\ locals variables in DUMP:
\ START_ADDR      \ first address to dump
\ END_ADDR        \ latest address to dump
\ ØSTART_ADDR     \ first address for dump loop
\ LINES           \ number of lines for dump loop
\ myBASE          \ current numeric base

: dump ( start len -- )
  cr cr ." --addr--- "
  ." 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  -----chars-----"
  2dup + { END_ADDR }          \ store latest address to dump
  swap { START_ADDR }         \ store START address to dump
  START_ADDR 16 / 16 * { ØSTART_ADDR } \ calc. addr for loop start
  16 / 1+ { LINES }
  base @ { myBASE }           \ save current base
  hex
  \ outer loop
  LINES 0 do
    ØSTART_ADDR + 16 * + \ calc. start address for current line
```

Ici, édition du fichier **dumpTool.fs** avec le bloc-notes.

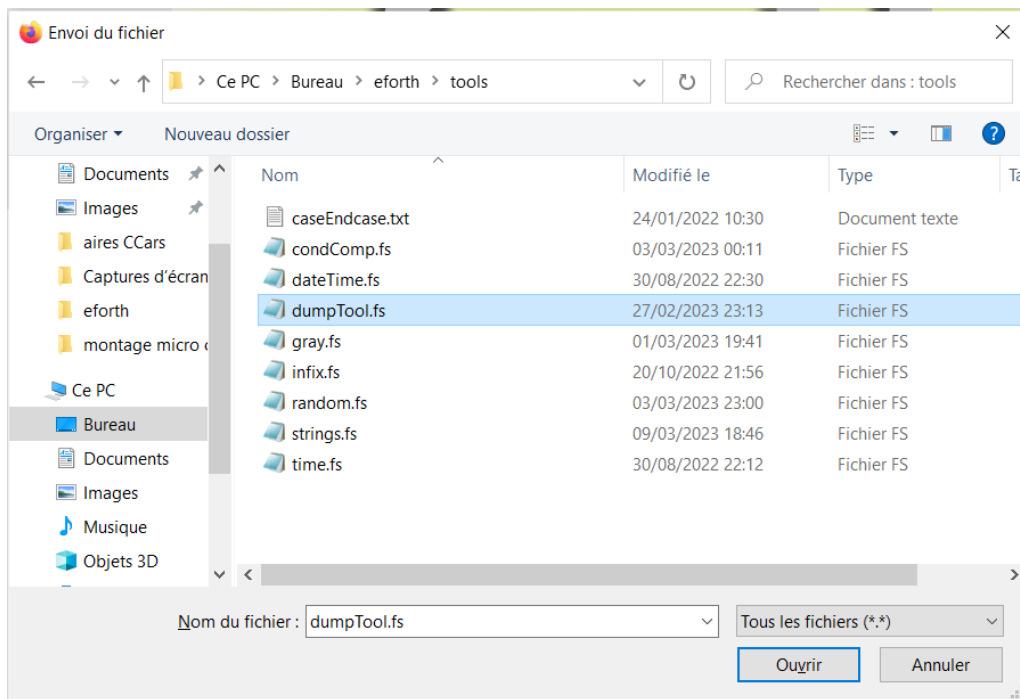
Sauvegardez vos fichiers source dans le répertoire de votre choix.

## Chargement d'un fichier source vers eForth

Pour charger le contenu de notre fichier **dumpTool.fs**, il suffit d'exécuter ces deux mots depuis l'invite de commande de eFORTH:

```
web import
```

La fenêtre de sélection de fichier s'ouvre sur votre ordinateur:



Ici, on sélectionne le fichier **dumpTool.fs** précédemment édité et sauvegardé.

L'envoi du fichier provoque immédiatement le traitement de son contenu. Ici, on a compilé le mot **dump** qu'il suffit ensuite de tester:

```
ok
--> here 100 dump
--addr--- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F -----chars-----
0001-F450 D4 90 01 00 00 00 00 00 00 00 00 00 00 00 00 .....
0001-F460 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0001-F470 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0001-F480 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0001-F490 00 00 00 00 00 00 00 00 00 00 00 30 30 30 31 2D .....0001-
0001-F4A0 46 34 33 33 00 00 00 00 00 00 00 00 00 00 00 F400.....
0001-F4B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
ok
-->
```

Un fichier source pour eFORTH peut contenir des mots à compiler, mais aussi des commandes, dont l'exécution des mots précédemment compilés. Cette facilité permet de gagner un temps considérable en temps de développement.

Seule différence, avec eFORTH pour windows, les fichiers source ne peuvent pas contenir un appel de fichier par include.

# Créez des définitions FORTH en utilisant JavaScript

## Le mariage Javascript Forth

eFORTH web fonctionne dans tous les navigateurs web, sur tous les systèmes Windows, Linux, MacOS, Android... Comment est-ce possible?

Le secret de cette extraordinaire polyvalence est très simple: un mariage réussi entre Javascript et FORTH.

La plupart des primitives du langage FORTH sont écrites dans un pseudo-assembleur ou en FORTH. Nous allons voir comment agrandir le vocabulaire web en définissant le mot `ellipse`:

```
web definitions

\ draw ellipse in graphic mode
JSWORD: ellipse { x y rx ry angle div }
  context.ctx.ellipse(x, y, rx, ry, Math.PI * 2 * angle / div, 0, 2 *
Math.PI);
~
forth definitions
```

Ici, le mot `ellipse` est défini après le mot de création `JSWORD:`. Le délimiteur de fin de définition est marqué par le caractère `~`. Voyons en détail la structure de définition de notre mot `ellipse`.

### Structure d'une définition de mot avec JSWORD:

Commençons par analyser la première ligne de cette définition de `ellipse`:

```
JSWORD: ellipse { x y rx ry angle div }
  context.ctx.ellipse(x, y, rx, ry, Math.PI * 2 * angle / div, 0, 2 *
Math.PI);
~
```

- le mot `JSWORD:` marque le début d'une définition utilisant le code Javascript. Ce mot de définition est immédiatement suivi du mot à définir, ici `ellipse`;
- le mot `ellipse` est le mot en cours de définition. Il est suivi d'un passage de paramètres par variables locales;
- les paramètres sont passés par `{ x y rx ry angle div }`.

La ligne suivante contient le code Javascript:

```
JSWORD: ellipse { x y rx ry angle div }
  context.ctx.ellipse(x, y, rx, ry, Math.PI * 2 * angle / div, 0, 2 *
Math.PI);
~
```

Le code Javascript peut être écrit sur plusieurs lignes:

```
web definitions
JSWORD: time@ { -- h m s }
  let date = new Date();
  var hh = date.getHours();
  var mm = date.getMinutes();
  var ss = date.getSeconds();
  return [hh, mm, ss];
~

JSWORD: date@ { -- y m d }
  let date = new Date();
  var yy = date.getFullYear();
  var mm = date.getMonth()+1;
  var dd = date.getUTCDate();
  return [yy, mm, dd];
~
forth definitions
```

Nous allons voir en détail comment s'effectue le passage des paramètres.

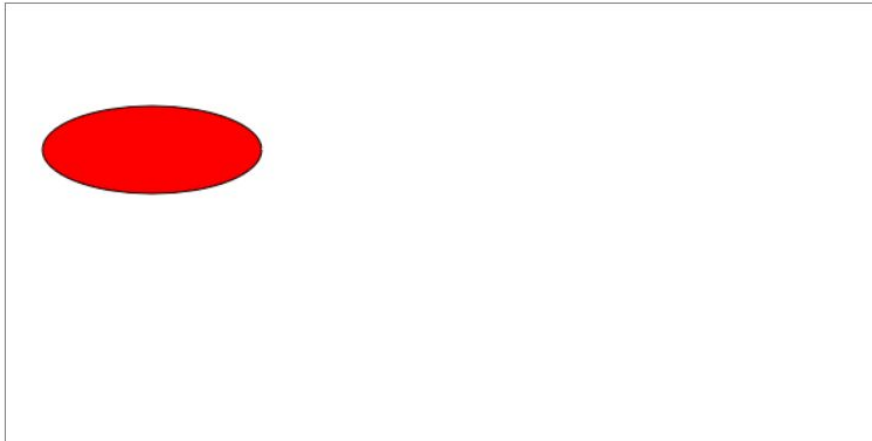
## Passage des paramètres entre eFORTH et JavaScript

Le passage de paramètres exploite les variables locales. Ces variables sont définies entre le mot `{` et le caractère `}`. Reprenons l'exemple de la définition du mot `ellipse`:

```
JSWORD: ellipse { x y rx ry angle div }
  context.ctx.ellipse(x, y, rx, ry, Math.PI * 2 * angle / div, 0, 2 *
Math.PI);
~
```

Ici nous passons cinq paramètres. Ces paramètres sont ceux utilisés ensuite dans une définition en langage FORTH:

```
web gr
600 300 window
$ff0000 color!
100 100 75 30 0 360 ellipse fill
0 color! stroke
```



Il y a quatre cas de passage de paramètres à un mot défini par **JSWORD** :

1) Appel à une fonction JavaScript sans passage de paramètre. Les accolades sont vides:

```
JSWORD: beginPath { }  
    context.ctx.beginPath();  
~
```

2) Appel à une fonction JavaScript avec passage de paramètres en entrée seulement:

```
JSWORD: lineTo { x y }  
    context.ctx.lineTo(x, y);  
~
```

3) Appel à une fonction JavaScript avec récupération de paramètres en sortie seulement:

```
JSWORD: viewport@ { -- w h }  
    return [context.width, context.height];  
~
```

4) Et enfin, passage de paramètres en entrée et en sortie:

```
JSWORD: web-type-raw { a n -- yld }  
    if (globalObj.write) {  
        var text = GetString(a, n);  
        write(text);  
        return 0;  
    } else {  
        var newline = false;  
        for (var i = 0; i < n; ++i) {  
            var ch = u8[a + i];  
            if (ch == 10) { newline = true; }  
            context.Emit(ch);  
        }  
        if (newline) {  
            context.Update();  
        }  
        return newline ? -1 : 0;  
    }
```



```
}  
~
```

Dans ce dernier cas, on sépare les paramètres d'entrée et de sortie avec `--`. En temps normal, dans une définition eFORTH classique, ces caractères et ceux qui suivent sont ignorés. Pour **JSWORD**, le passage des paramètres doit impérativement respecter une des quatre formes énumérées ci-avant.

Pour les paramètres de sortie, il n'est pas nécessaire d'utiliser des variables JavaScript. Reprenons le code de **time@**:

```
web definitions  
JSWORD: time@ { -- h m s }  
  let date = new Date();  
  var hh = date.getHours();  
  var mm = date.getMinutes();  
  var ss = date.getSeconds();  
  return [hh, mm, ss];  
~  
forth definitions
```

Ce code peut être réécrit comme suit:

```
web definitions  
JSWORD: time@ { -- h m s }  
  let date = new Date();  
  return [date.getHours(), date.getMinutes(), date.getSeconds()];  
~  
forth definitions
```

La fonction JavaScript **return()** doit simplement énumérer les valeurs à passer à eFORTH. Les `--` doivent être suivis d'autant de paramètres que ceux renvoyés par **return()**.

Il est facile d'étendre eFORTH en rajoutant des fonctions de tout type: file, HTTP, etc...

## Extension du vocabulaire web de eFORTH

Le vocabulaire web est très incomplet. Vous pouvez l'étendre en récupérant le contenu du fichier **additionalDefs.fs** dans le fichier **eForthWEB-book.zip**.

Installez ce fichier dans le répertoire de votre choix. Pour notre part, nous avons une préférence pour le nom de répertoire **fs** (pour Forth Script). Pour appeler le contenu de ce fichier **additionalDefs.fs** depuis votre application web:

```
<div id="ueforth"></div>  
<script src="../js/ueforth.js"></script>  
<script type="text/forth" src="fs/additionnalDefs.fs"></script>  
<script type="text/forth">  
  \ here additionnal Forth Code
```

```
</script>
```

# Initiation au graphisme avec eFORTH pour le web

## Comprendre l'environnement graphique

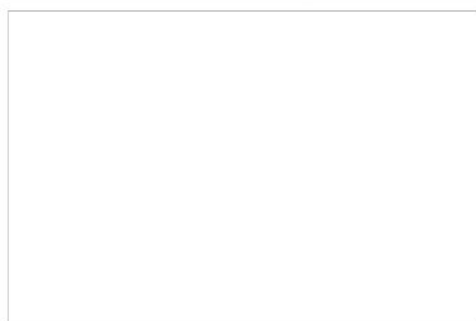
eFORTH utilise l'élément HTML **<canvas>** pour gérer des formes graphiques. Cet élément est activé en exécutant le mot `gr` qui est défini dans le vocabulaire `web`:

```
web gr
600 400 window
```

On définit ensuite la taille de cet élément en utilisant **window**. Dans notre exemple, ceci définit une fenêtre graphique de 600 pixels de large par 400 pixels de hauteur.

Il est convenu que tous nos exemples commenceront avec ces deux lignes d'initialisation sans qu'il soit nécessaire de les réécrire pour chaque exemple.

Try eFORTH online version



```
uFORTH v2.0.7.9 rev 4b5b70da5d111b1fd90
Forth dictionary: 4068180 free 1 76/28 used = 4344908 total (98% free)
3 x Forth stacks: 16384 bytes each
ok
--> web gr 600 400 window
ok
--> 
```

Le système de coordonnées de **<canvas>** commence toujours à l'angle supérieur gauche. Il est défini en points. Dans notre cas, chaque point correspondra à un pixel. Cette échelle peut être modifiée avec **scale**. Nous verrons l'utilisation de ce mot plus tard.

## Notion de tracé et remplissage

Les fonctions graphiques associées à **<canvas>** sont simples, mais nécessitent une certaine méthode pour être efficaces.

Un tracé, c'est un chemin. On marque le début d'un tracé avec le mot **beginPath**. Puis un tracé est déterminé par sa couleur avec **color!**. Et enfin, on réalise le tracé avec **moveTo** et **lineTo**.

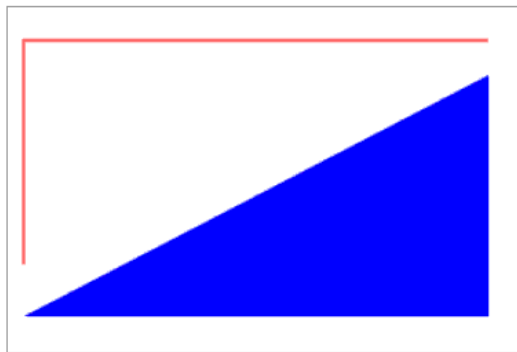
Le tracé s'achève avec **stroke** si on ne souhaite afficher que les traits du tracé:

```
$ff0000 constant red
beginpath red color! 10 150 moveTo 10 20 lineto 280 20 lineto stroke
```

Pour réaliser un tracé plein, on remplace **stroke** par **fill** :

```
$ff0000 constant red
$0000ff constant blue
beginpath  red color!  10 150 moveTo  10  20 lineto  280 20 lineto
stroke
beginpath  blue color!  10 180 moveTo  280 180 lineto  280 40 lineto fill
```

## Try eFORTH online version



ok

La couleur peut aussi être définie en fin de tracé. Dans ce cas, elle s'appliquera à tout le tracé depuis l'exécution de **beginPath** :

```
$ff0000 constant red
$0000ff constant blue

web gr
400 300 window

beginpath  10 150 moveTo  10  20 lineto  280 20 lineto
red color!  stroke
4 linewidth
beginpath  10 180 moveTo  280 180 lineto  280 40 lineto
blue color!  fill
red color!  stroke
```

Ici, ce sont les deux dernières lignes de code qui nous intéressent. On exécute d'abord le tracé de notre figure, puis c'est seulement après ce tracé qu'on sélectionne la couleur de fond avec **blue color! fill** et ensuite la couleur du contour avec **red color! stroke**.

La coloration du contour n'apparaît que sur la partie de contour tracée.

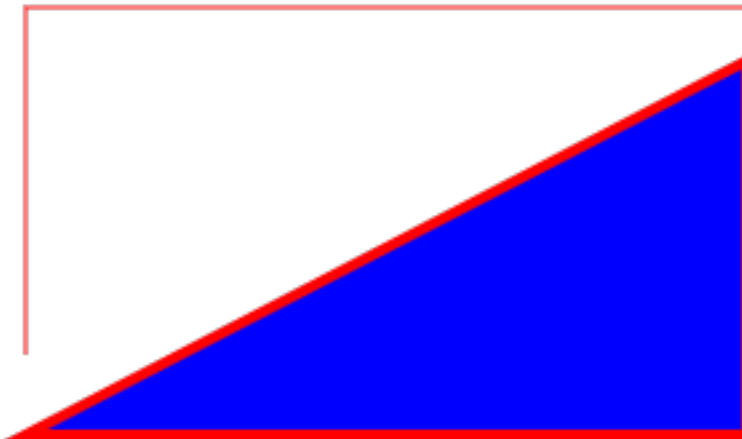
Pour fermer un tracé, on peut utiliser le mot **closePath**. Voici comment définir ce mot qui n'existe pas d'origine dans le vocabulaire web :

```
web definitions
JSWORD: closePath { }
  context.ctx.closePath();
```

```
~  
forth definitions
```

Et maintenant, on peut modifier le code de traçage :

```
4 linewidth  
beginpath 10 180 moveTo 280 180 lineto 280 40 lineto closePath  
blue color! fill  
red color! stroke
```



## Rotations et translations

Le système de coordonnées de canvas commence dans l'angle supérieur gauche de notre fenêtre graphique. Les coordonnées x sont croissantes vers la droite, les coordonnées y sont croissantes vers le bas.

Ceci est vrai tant que l'on n'effectue aucune transformation. Pour bien assimiler ceci, nous allons tracer une vraie grille dans l'espace de travail :

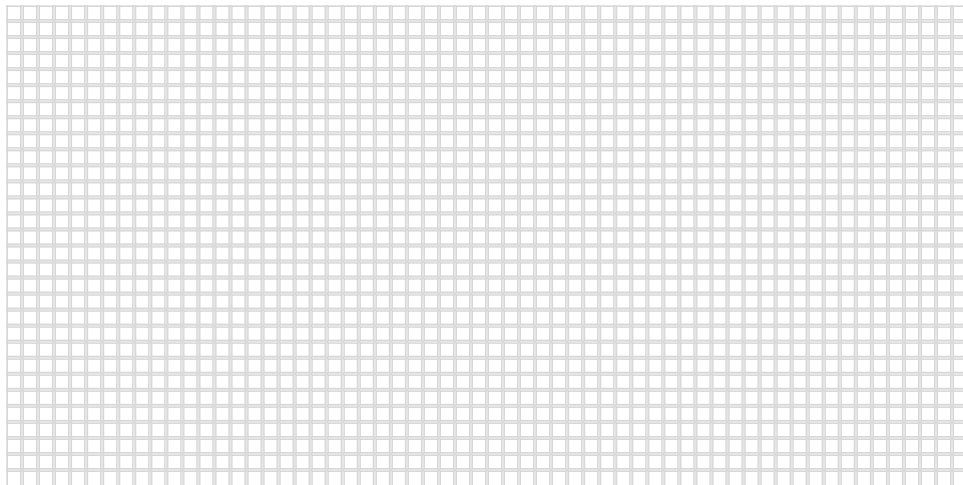
```
600 constant gridWidth  
300 constant gridHeight  
10 constant gridStep  
  
$cfcfcf constant lightGrey  
$afaffff constant lightBlue  
  
web  
: drawGrid ( -- )  
  \ draw vertical lines  
  gridWidth 0 do  
    beginPath  
    i 0 moveTo  
    i gridHeight lineTo  
    stroke
```

```

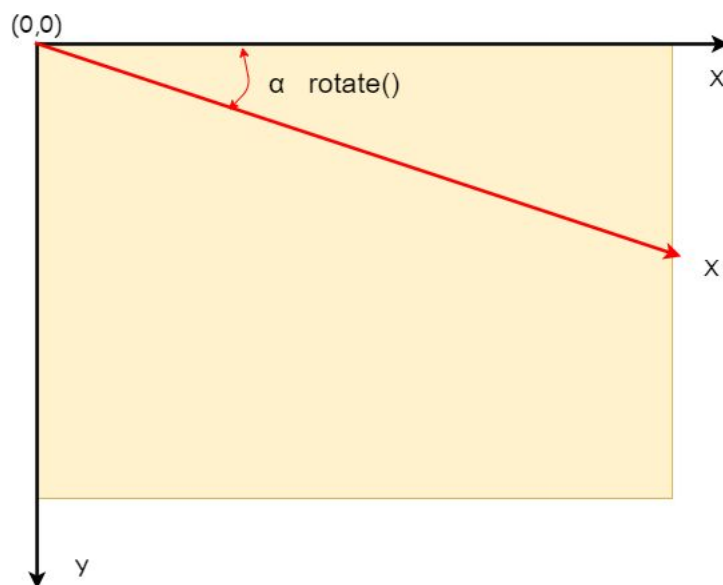
gridStep +loop
\ draw horizontal lines
gridHeight 0 do
  beginPath
  0 i moveTo
  gridWidth i lineTo
  stroke
gridStep +loop
;

web gr
gridWidth gridHeight window
lightGrey color! drawGrid

```



Nous allons effectuer une rotation de  $30^\circ$ . La rotation s'effectue dans le sens horaire et s'applique depuis l'axe horizontal supérieur et au point de départ de traçage qui est dans l'angle supérieur gauche :



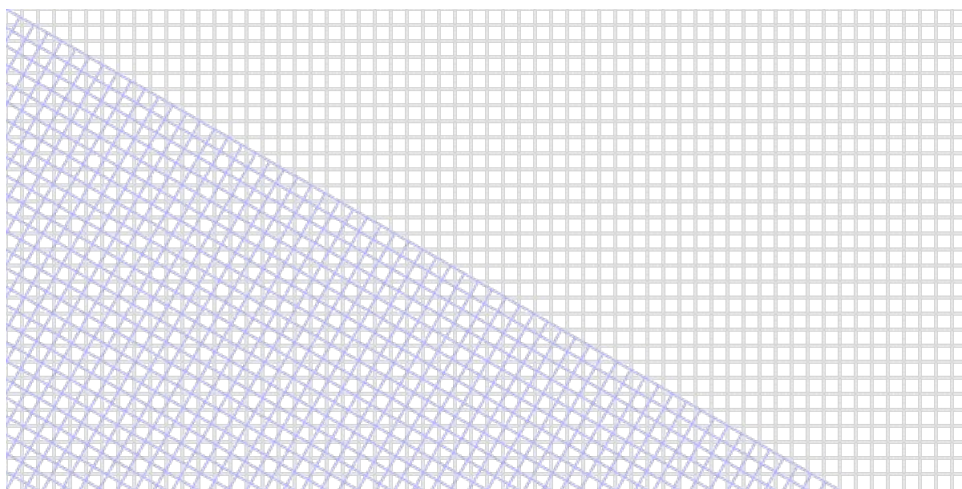
Avec eFORTH, on utilise le mot **rotate** qui a besoin de deux paramètres :

- la valeur de déplacement angulaire
- le diviseur appliqué à cette valeur angulaire

Pour effectuer une rotation de 90 degrés, on peut taper **90 360 rotate**. Mais on peut aussi taper **1 4 rotate** qui aura exactement le même effet.

Nous allons effectuer une rotation de 30° et tracer une autre grille colorée en bleu clair :

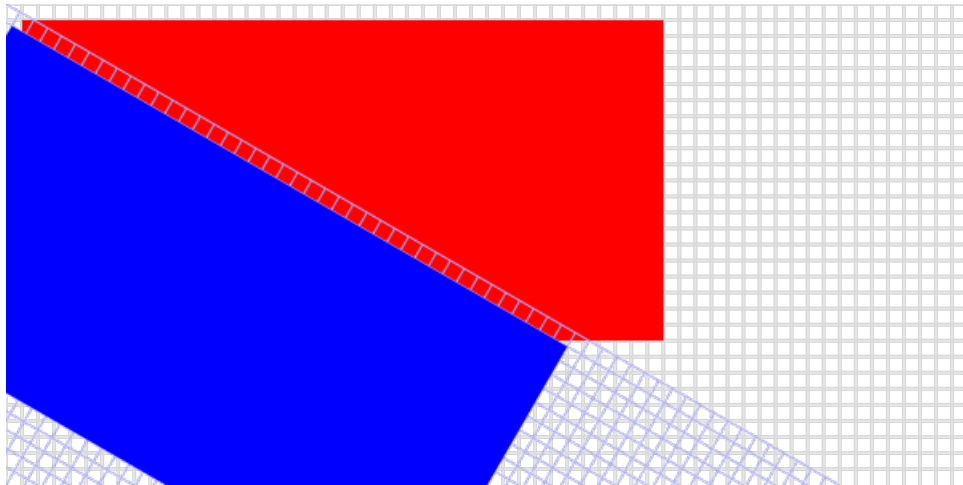
```
web gr
gridWidth gridHeight window
lightGrey color! drawGrid
1 12 rotate
lightBlue color! drawGrid
```



Mmmmmm.... très intéressant! Les tracés peuvent sortir de l'espace de travail sans faire planter eFORTH...

Voyons ce que ça donne si on trace un rectangle dans chacune de ces grilles:

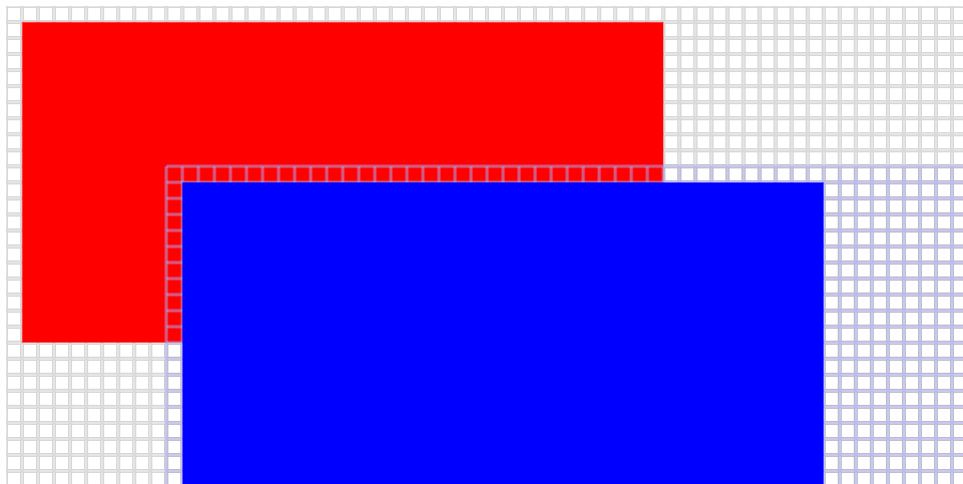
```
: myBox ( color -- )
  color! 10 10 400 200 box
;
web gr
gridWidth gridHeight window
lightGrey color! drawGrid
fullRed myBox
1 12 rotate
lightBlue color! drawGrid
fullBlue myBox
```



La rotation est cumulative. Pour faire tourner de trente autre degrés, on exécutera à nouveau **1 12 rotate**.

La translation est réalisée par le mot **translate**. Ce mot a besoin de deux paramètres, x et y, indiquant le déplacement horizontal et vertical dans l'espace de travail :

```
web gr
gridWidth gridHeight window
lightGrey color! drawGrid
fullRed myBox
300 100 translate
lightBlue color! drawGrid
fullBlue myBox
```



Une fois **translate** exécuté, le nouveau système de coordonnées est situé au point de destination de cette translation. Les translations et rotations sont cumulatives :

```
: myBox ( color -- )
  color! 10 10 400 200 box
;
```



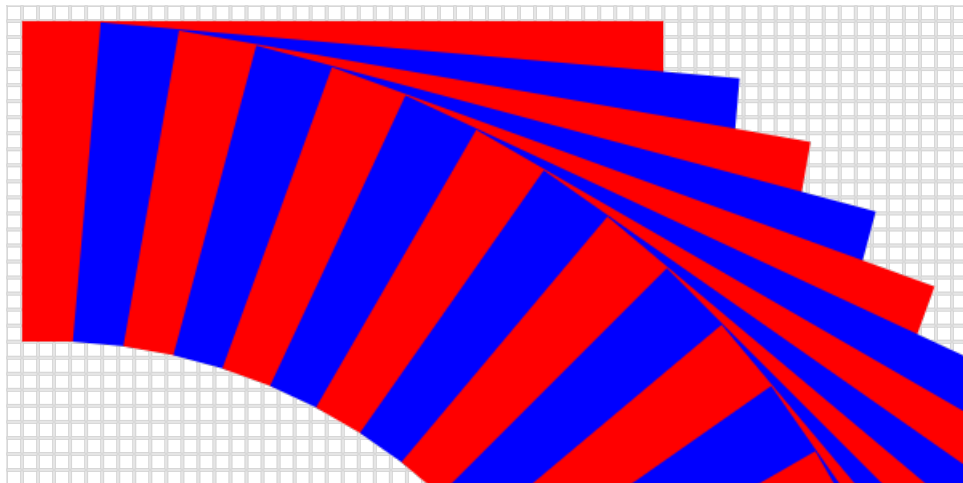
```

: addTranslateRotate ( -- )
    50 0 translate
    5 360 rotate
;

: myBoxes ( -- )
    12 for
        aft
            fullRed myBox
            addTranslateRotate
            fullBlue myBox
            addTranslateRotate
        then
    next
;

web gr
gridWidth gridHeight window
lightGrey color! drawGrid
myBoxes

```



## Cas pratique: tracé des heures d'une horloge

Voici un exemple pratique, le traçage des heures d'une pendule :

```

600 constant gridWidth
300 constant gridHeight
10 constant gridStep

gridHeight 2/ 90 100 */ value clockRadius
gridHeight 2/ 10 /      value face_width

$cfcfcf constant lightGrey
$afafff constant lightBlue
$ff0000 constant fullRed

```

```

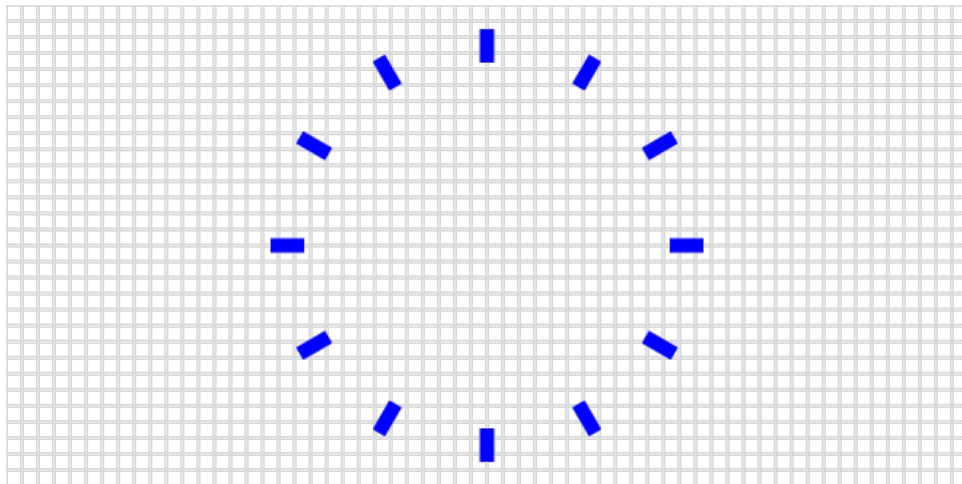
$0000ff constant fullBlue

web
: drawHrMin { size }
  beginPath
  clockRadius size 100 */ 0 moveTo
  clockRadius 0 lineTo
  stroke
;

\ draw hours
: drawHours ( --)
  fullBlue color!
  face_width 3 5 */ lineWidth
  12 0 do
    1 12 rotate
    85 drawHrMin
  loop
;

web gr
gridWidth gridHeight window
lightGrey color! drawGrid
gridWidth 2/ gridHeight 2/ translate
drawHours

```



Bonne programmation.

# Gestion des images avec eFORTH pour le web

## Chargement et affichage des images

La gestion des images est réalisée à partir du mot **drawImage**. ce mot n'étant pas défini dans eFORTH, voici comment le créer dans le vocabulaire web :

```
web definitions
JSWORD: drawImage { a n x y }
  let img = new Image();
  img.addEventListener('load', function() {
    context.ctx.drawImage(img, x, y);
  }, false);
  img.src = GetString(a, n);
~
forth definitions
```

Voici une image, au format GIF, qui va nous servir pour ces premiers tests.

Et voici comment utiliser **drawImage** pour l'intégrer à canvas:

```
600 constant ctxWidth
300 constant ctxHeight

web
gr
ctxWidth ctxHeight window

s" greenPencil.gif" 20 10 drawImage
```



Pour centrer cette image, il faut récupérer ses dimensions. Pour ce faire, définissons ce mot :

```
web definitions
JSWORD: imageSize { a n -- w h }
```

```

let img = new Image();
img.src = GetString(a, n);
if(img.complete){
    return [img.naturalWidth, img.naturalHeight];
}
~
forth definitions

```

Voici un code non optimisé montrant comment calculer la position de l'image pour qu'elle soit positionnée au centre de notre espace délimité par canvas.

```

forth definitions

600 constant ctxWidth
300 constant ctxHeight

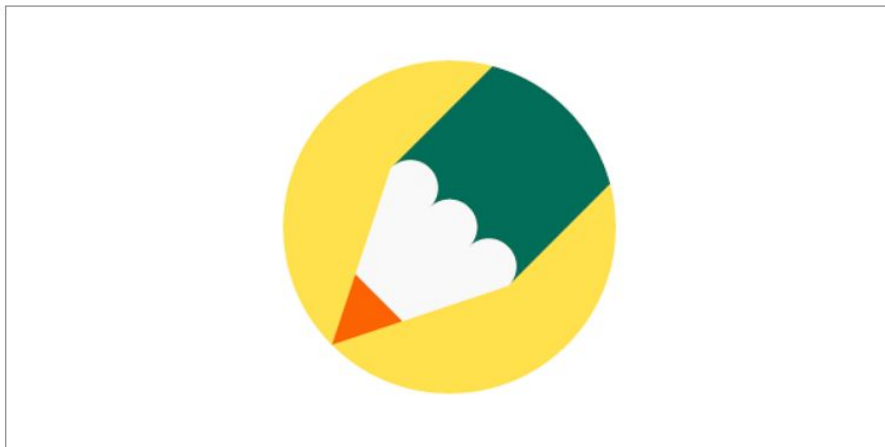
web
gr
ctxWidth ctxHeight window

0 value imageWidth
0 value imageHeight

s" greenPencil.gif" imageSize
  to imageHeight
  to imageWidth

s" greenPencil.gif"
  ctxWidth imageWidth - 2/
  ctxHeight imageHeight - 2/ drawImage

```



## Affichage et découpage d'une image au format jpg

Le mot **drawImage** sait afficher divers format d'images: gif, png, jpg, svg. Voyons comment afficher et découper une image au format jpg. Mais avant de voir en détail ce découpage d'image, définissons ces mots:

```
web definitions
JSWORD: getImageData { addr len x y w h }
    var myString = GetString(addr, len);
    const imageData = context.ctx.getImageData(x, y, w, h);
    Object.assign(context.ctx, {[myString]:imageData});
~
\ usage:
\   s" motorhome" 20 30 100 200 getImageData
\   create copie from canvas content, named "motorhome"

JSWORD: putImageData { addr len x y }
    var myString = "context.ctx." + GetString(addr, len);
    const imageData = eval(myString);
    console.info(imageData);
    context.ctx.putImageData(imageData, x, y);
~
\ usage:
\   s" motorhome" 300 30 putImageData

\ create in dictionnay and save part of image
\ execution of x y display saved part of image
: getImage: ( comp: x y w h -- | exec: x y )
    create
        >r >r >r >r
        latestxt dup ,
        >name r> r> r> r> getImageData
    does>
        -rot >r >r
        @ >name r> r> putImageData
;
forth definitions
```

Fonctionnement de ces trois nouvelles définitions :

- **getImageData** récupère une partie d'image déjà affichée et stocke les données de cette partie d'image dans un objet Javascript;
- **putImageData** affiche à la position x y une partie d'image précédemment stockée par getImageData et désignée par son nom d'objet Javascript;
- **getImage**: crée un mot dans le vocabulaire FORTH qui pointe vers l'objet Javascript de même nom et stockant une partie d'image. L'exécution de ce mot, précédé des coordonnées x y affiche le contenu de cette partie d'image.

Voici l'utilisation du mot **getImage::**

```
: imgCCar ( -- addr len )
  s" ccar.jpg"
  ;

web gr
ctxWidth ctxHeight window

imgCCar 5 5 drawImage
500 ms

30 5 80 232 getImage: motorhome

380 5 motorhome
465 5 motorhome
```

Notre image a comme nom **ccar.jpg**. Le mot **imgCCar** est donc une sorte de constante de chaîne alphanumérique.

La séquence de code **30 5 80 232 getImage: motorhome** va copier une partie de l'image affichée dans canvas, à partir de x et y (30 5) et de 80 pixels de large et 232 pixels de haut. Le mot **getImage:** crée le mot **motorhome**.



L'exécution de **motorhome** précédé des coordonnées x y d'affichage affiche la partie d'image copiée au moment de créer **motorhome**.

## Tester la couleur d'un pixel image

Pour tester la couleur d'un pixel, on définit une variante du mot **getImageData**, mais qui teste seulement un pixel dans l'espace graphique canvas :

```
web definitions
\ get pixel color at x y
JSWORD: getPixelColor { x y -- c }
```

```

var pixel = context.ctx.getImageData(x, y, 1, 1);
console.info(pixel);
return pixel.data[0]*256*256 + pixel.data[1]*256 + pixel.data[2];
~
forth definitions

```

Voici un cas d'école. Tracer trois boîtes rectangulaires. L'astuce, ici, c'est de tracer ces trois boîtes en bleu, mais avec une couleur bleue adaptée à chaque boîte :

```

$0000ff constant colorBox1
$0001ff constant colorBox2
$0100ff constant colorBox3

web gr
ctxWidth ctxHeight window
colorBox1 color!
10 10 100 50 fillRect
colorBox2 color!
120 10 300 120 fillRect
colorBox3 color!
400 140 180 120 fillRect

```

Voici une définition **testBoxes** qui va tester le pixel pointé par la souris. La couleur testée permet de savoir quelle boîte est pointée :

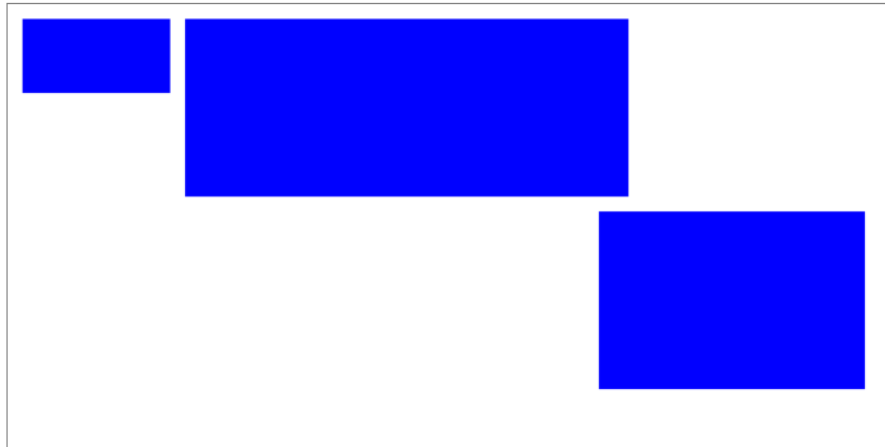
```

: waitButtonDown ( -- )
  begin
    button 0=
  until
  ;

: testBoxes
  begin
    button if
      getMousePosition getPixelColor
      case
        page
          colorBox2 of ." BOX 2 selected" cr endof
          colorBox3 of ." BOX 3 selected" cr endof
          colorBox1 of ." BOX 1 selected" cr endof
      endcase
      200 ms
    then
      key? until
  ;

." click on any blue box / space to STOP" cr
testBoxes

```



Voici la définition de **getMousePosition** qui n'est pas défini dans le vocabulaire web d'origine :

```
web definitions
\ The word mouse delivers the position of
\ the mouse pointer from the origin x y (0, 0)
\ of the HTML page and not from the origin of the canvas.
\ The word getMousePosition retrieves and recalculates
\ the relative position of the
\ mouse pointer from the origin of the canvas.
JSWORD: getMousePosition { -- mousex mousey }
  var offset = {x: 0, y: 0};
  var node = context.ctx.canvas;
  while (node) {
    offset.x += node.offsetLeft;
    offset.y += node.offsetTop;
    node = node.offsetParent;
  }
  return [context.mouse_x-offset.x, context.mouse_y-offset.y];
~
forth definitions
```

Cette définition est à utiliser en remplacement de **mouse**.



# Contenu détaillé des vocabulaires eFORTH WEB

eFORTH WEB met à disposition de nombreux vocabulaires :

- **FORTH** est le principal vocabulaire ;
- certains vocabulaires servent à la mécanique interne pour eFORTH WEB, comme **internals**, **asm...**

Vous trouverez ici la liste de tous les mots définis dans ces différents vocabulaires. Certains mots sont présentés avec un lien coloré :

[align](#) est un mot FORTH ordinaire ;

**CONSTANT** est mot de définition ;

**begin** marque une structure de contrôle ;

**key** est un mot d'exécution différée ;

**handler** est un mot défini par **constant**, **variable** ou **value** ;

**web** marque un vocabulaire.

Les mots du vocabulaire **FORTH** sont affichés par ordre alphabétique. Pour les autres vocabulaires, les mots sont présentés dans leur ordre d'affichage.

Vous pouvez accéder à la documentation d'un mot FORTH s'il est souligné, simplement en cliquant sur ce mot.

## Version v 7.0.7.15

### FORTH

<a href="#">=</a>	<a href="#">-rot</a>	<a href="#">_</a>	<a href="#">.</a>	<a href="#">:</a>	<a href="#">:noname</a>	<a href="#">!</a>	<a href="#">?</a>
<a href="#">?do</a>	<a href="#">?dup</a>	<a href="#">_</a>	<a href="#">_."</a>	<a href="#">.s</a>	<a href="#">'</a>	<a href="#">(local)</a>	<a href="#">[</a>
<a href="#">[']</a>	<a href="#">[char]</a>	<a href="#">[ELSE]</a>	<a href="#">[IF]</a>	<a href="#">[THEN]</a>	<a href="#">l</a>	<a href="#">{</a>	<a href="#">}transfer</a>
<a href="#">@</a>	<a href="#">*</a>	<a href="#">*/</a>	<a href="#">*/MOD</a>	<a href="#">/</a>	<a href="#">/mod</a>	<a href="#">#</a>	<a href="#">#!</a>
<a href="#">#&gt;</a>	<a href="#">#fs</a>	<a href="#">#s</a>	<a href="#">#tib</a>	<a href="#">+</a>	<a href="#">+!</a>	<a href="#">+loop</a>	<a href="#">+to</a>
<a href="#">&lt;</a>	<a href="#">&lt;#</a>	<a href="#">&lt;=</a>	<a href="#">&lt;&gt;</a>	<a href="#">=</a>	<a href="#">≥</a>	<a href="#">&gt;=</a>	<a href="#">&gt;BODY</a>
<a href="#">&gt;flags</a>	<a href="#">&gt;flags&amp;</a>	<a href="#">&gt;in</a>	<a href="#">&gt;link</a>	<a href="#">&gt;link&amp;</a>	<a href="#">&gt;name</a>	<a href="#">&gt;name-length</a>	
<a href="#">&gt;params</a>	<a href="#">&gt;R</a>	<a href="#">&gt;size</a>	<a href="#">0&lt;</a>	<a href="#">0&lt;&gt;</a>	<a href="#">0=</a>	<a href="#">1-</a>	<a href="#">1/F</a>
<a href="#">1+</a>	<a href="#">2!</a>	<a href="#">2@</a>	<a href="#">2*</a>	<a href="#">2/</a>	<a href="#">4*</a>	<a href="#">2drop</a>	<a href="#">2dup</a>
<a href="#">4/</a>	<a href="#">abort</a>	<a href="#">abort"</a>	<a href="#">abs</a>	<a href="#">accept</a>	<a href="#">afliteral</a>	<a href="#">aft</a>	<a href="#">again</a>
<a href="#">ahead</a>	<a href="#">align</a>	<a href="#">aligned</a>	<a href="#">allot</a>	<a href="#">also</a>	<a href="#">AND</a>	<a href="#">ansi</a>	<a href="#">ARSHIFT</a>
<a href="#">asm</a>	<a href="#">assert</a>	<a href="#">at-xy</a>	<a href="#">base</a>	<a href="#">begin</a>	<a href="#">bg</a>	<a href="#">binary</a>	<a href="#">bl</a>
<a href="#">blank</a>	<a href="#">bye</a>	<a href="#">c,</a>	<a href="#">C!</a>	<a href="#">C@</a>	<a href="#">CALL</a>	<a href="#">CASE</a>	<a href="#">catch</a>
<a href="#">CELL</a>	<a href="#">cell/</a>	<a href="#">cell+</a>	<a href="#">cells</a>	<a href="#">char</a>	<a href="#">cmove</a>	<a href="#">cmove&gt;</a>	<a href="#">colors</a>
<a href="#">CONSTANT</a>		<a href="#">context</a>	<a href="#">cr</a>	<a href="#">CREATE</a>	<a href="#">current</a>	<a href="#">decimal</a>	<a href="#">defer</a>
<a href="#">DEFINED?</a>		<a href="#">definitions</a>	<a href="#">depth</a>	<a href="#">do</a>	<a href="#">DOES&gt;</a>	<a href="#">DROP</a>	<a href="#">dump</a>
<a href="#">DUP</a>	<a href="#">echo</a>	<a href="#">else</a>	<a href="#">emit</a>	<a href="#">ENDCASE</a>	<a href="#">ENDOF</a>	<a href="#">erase</a>	<a href="#">evaluate</a>
<a href="#">EXECUTE</a>	<a href="#">EXIT</a>	<a href="#">exit</a>	<a href="#">extract</a>	<a href="#">F-</a>	<a href="#">f.</a>	<a href="#">f.s</a>	<a href="#">F*</a>

<a href="#">F**</a>	<a href="#">E/</a>	<a href="#">F+</a>	<a href="#">F&lt;</a>	<a href="#">F&lt;=</a>	<a href="#">F&lt;&gt;</a>	<a href="#">F=</a>	<a href="#">F&gt;</a>
<a href="#">F&gt;=</a>	<a href="#">F&gt;S</a>	<a href="#">F0&lt;</a>	<a href="#">F0=</a>	<a href="#">FABS</a>	<a href="#">FATAN2</a>	<a href="#">fconstant</a>	<a href="#">FCOS</a>
<a href="#">fdepth</a>	<a href="#">FDROP</a>	<a href="#">FDUP</a>	<a href="#">FEXP</a>	<a href="#">fq</a>	<a href="#">fill</a>	<a href="#">fill32</a>	<a href="#">FIND</a>
<a href="#">fliteral</a>		<a href="#">FLN</a>	<a href="#">FLOOR</a>	<a href="#">FMAX</a>	<a href="#">FMIN</a>	<a href="#">FNEGATE</a>	<a href="#">FNIP</a>
<a href="#">for</a>	<a href="#">forget</a>	<a href="#">FORTH</a>	<a href="#">forth-builtins</a>	<a href="#">FOVER</a>	<a href="#">FP!</a>	<a href="#">FP@</a>	
<a href="#">fp0</a>	<a href="#">FROT</a>	<a href="#">FSIN</a>	<a href="#">FSINCOS</a>	<a href="#">FSQRT</a>	<a href="#">FSWAP</a>	<a href="#">fvariable</a>	<a href="#">handler</a>
<a href="#">here</a>	<a href="#">hex</a>	<a href="#">hld</a>	<a href="#">hold</a>	<a href="#">I</a>	<a href="#">if</a>	<a href="#">IMMEDIATE</a>	<a href="#">internals</a>
<a href="#">invert</a>	<a href="#">is</a>	<a href="#">J</a>	<a href="#">K</a>	<a href="#">key</a>	<a href="#">key?</a>	<a href="#">L!</a>	<a href="#">latestxt</a>
<a href="#">leave</a>	<a href="#">literal</a>	<a href="#">loop</a>	<a href="#">LSHIFT</a>	<a href="#">max</a>	<a href="#">min</a>	<a href="#">mod</a>	<a href="#">ms</a>
<a href="#">ms-ticks</a>		<a href="#">n.</a>	<a href="#">negate</a>	<a href="#">nest-depth</a>	<a href="#">next</a>	<a href="#">nip</a>	<a href="#">nl</a>
<a href="#">normal</a>	<a href="#">octal</a>	<a href="#">OF</a>	<a href="#">ok</a>	<a href="#">only</a>	<a href="#">OR</a>	<a href="#">order</a>	<a href="#">OVER</a>
<a href="#">pad</a>	<a href="#">page</a>	<a href="#">PARSE</a>	<a href="#">pause</a>	<a href="#">PI</a>	<a href="#">postpone</a>	<a href="#">precision</a>	<a href="#">previous</a>
<a href="#">prompt</a>	<a href="#">quit</a>	<a href="#">r"</a>	<a href="#">R@</a>	<a href="#">R&gt;</a>	<a href="#">r </a>	<a href="#">r~</a>	<a href="#">rdrop</a>
<a href="#">recurse</a>	<a href="#">refill</a>	<a href="#">remaining</a>	<a href="#">repeat</a>	<a href="#">rot</a>	<a href="#">RP!</a>	<a href="#">RP@</a>	<a href="#">rp0</a>
<a href="#">RSHIFT</a>	<a href="#">s"</a>	<a href="#">S&gt;F</a>	<a href="#">s&gt;z</a>	<a href="#">sealed</a>	<a href="#">see</a>	<a href="#">set-precision</a>	
<a href="#">set-title</a>		<a href="#">sf,</a>	<a href="#">SF!</a>	<a href="#">SF@</a>	<a href="#">SFLOAT</a>	<a href="#">SFLOAT+</a>	<a href="#">SFLOATS</a>
<a href="#">sign</a>	<a href="#">SL@</a>	<a href="#">SP!</a>	<a href="#">SP@</a>	<a href="#">sp0</a>	<a href="#">space</a>	<a href="#">spaces</a>	<a href="#">start-task</a>
<a href="#">startswith?</a>		<a href="#">state</a>	<a href="#">str</a>	<a href="#">str=</a>	<a href="#">structures</a>	<a href="#">SW@</a>	<a href="#">SWAP</a>
<a href="#">task</a>	<a href="#">tasks</a>	<a href="#">terminate</a>	<a href="#">then</a>	<a href="#">throw</a>	<a href="#">tib</a>	<a href="#">to</a>	<a href="#">transfer</a>
<a href="#">transfer{</a>		<a href="#">type</a>	<a href="#">u.</a>	<a href="#">U/MOD</a>	<a href="#">UL@</a>	<a href="#">UNLOOP</a>	<a href="#">until</a>
<a href="#">used</a>	<a href="#">UW@</a>	<a href="#">value</a>	<a href="#">VARIABLE</a>	<a href="#">vlist</a>	<a href="#">vocabulary</a>	<a href="#">W!</a>	<a href="#">web</a>
<a href="#">while</a>	<a href="#">words</a>	<a href="#">XOR</a>	<a href="#">z"</a>	<a href="#">z&gt;s</a>			

## ansi

```
terminal-restore terminal-save show hide scroll-up scroll-down clear-to-eol
bel esc
```

## asm

```
code, code4, code3, code2, code1, callot chere code-at code-start
```

## internalized

```
flags'or! LEAVE LOOP +LOOP ?DO DO NEXT FOR AFT REPEAT WHILE ELSE IF THEN
AHEAD UNTIL AGAIN BEGIN cleave
```

## internals

```
DOFLIT S>FLOAT? 'heap 'context 'latestxt 'notfound 'heap-start 'heap-size
'stack-cells 'boot 'boot-size 'tib 'argc 'argv 'runner 'throw-handler NOP
BRANCH 0BRANCH DONEXT DOLIT DOSET DOCOL DOCON DOVAR DOCREATE DODOES ALITERAL
LONG-SIZE S>NUMBER? 'SYS YIELD EVALUATE1 'builtins internals-builtins cases
\(+to\) \(to\) --? }? ?room scope-create do-local scope-clear scope-exit local-op
scope-depth local+! local! local@ <>locals locals-here locals-area locals-gap
locals-capacity ?ins. ins. vins. onlines line-pos line-width size-all size-
vocabulary
vocs. voc. voclist voclist-from see-all >vocnext see-vocabulary nonvoc?
see-xt ?see-flags see-loop see-one indent+! icr see. indent mem= ARGS_MARK
-TAB +TAB NONAMED BUILTIN_FORK SMUDGE IMMEDIATE_MARK dump-line ca@ cell-shift
cell-base cell-mask CALLCODE #f+s internalized BUILTIN_MARK zplace $place
free. boot-prompt raw-ok \[SKIP\] \[SKIP\] ?stack sp-limit input-limit tib-setup
raw.s $@ digit parse-quote leaving, leaving )leaving leaving( value-bind
```

```
evaluate&fill evaluate-buffer arrow ?arrow. ?echo input-buffer immediate?  
eat-till-cr wascr *emit *key notfound last-vocabulary voc-stack-end xt-transfer  
xt-hide xt-find& scope
```

## structures

```
field struct-align align-by last-struct struct long ptr i64 i32 i16 i8  
typer last-align
```

## tasks

```
.tasks main-task task-list
```

## web

```
yielding-task yielding import rm ls download cat include-file upload upload-file  
session? web-key? web-key web-type scripts scripts# random button mouse  
textWidth fillText font text-size! log raw-http-upload http-download raw-download  
upload-success? upload-done? upload-start ms-ticks silence tone importScripts  
release keyCount getKey clearItems removeItem getItem setItem smooth gpop  
gpush rotate scale translate show-text keys-height mobile textRatios viewport@  
window line fill stroke lineTo moveTo beginPath box lineWidth color! text  
gr grmode shouldEcho? web-terminate web-key?-raw web-key-raw web-type-raw  
jseval JSWORD: jsslot jseval!
```



## Index lexical

1-.....	15	getMousePosition.....	40	SWAP.....	17
1+.....	15	getPixelColor.....	38	tasks.....	43
2*.....	15	imageSize.....	35	translate.....	32
ansi.....	42	import.....	20	U.....	14
asm.....	42	internalized.....	42	value.....	11
beginPath.....	27	internals.....	42	variable.....	10
c!.....	10	JSWORD:.....	22	web.....	20, 43
c@.....	10	lineTo.....	27	window.....	27
closePath.....	28	mémoire.....	10	.....	7
color!.....	28	MOD.....	15	.....	7
constant.....	11	moveTo.....	27	.S.....	13
drawImage.....	35, 37	OVER.....	17	@.....	10
drop.....	9	pile de retour.....	9	*.....	15
dup.....	9	putImageData.....	37	*/.....	16
DUP.....	17	R>.....	18	*/MOD.....	16
fill .....	28	ROT.....	18	/.....	15
forget.....	7	rotate.....	31	/MOD.....	16
FORTH.....	41	stroke.....	27	+.....	14
getImage:.....	37	structures.....	43	>R.....	18