2024

FREDERIC VISSAULT

Pybee

20/11/2024

Manuel du langage Beetle



# Table des matières

[Table des matières 1](#_Toc185434091)

[Introduction 7](#_Toc185434092)

[Notation polonaise inversée 7](#_Toc185434093)

[Piles de données 8](#_Toc185434094)

[Console 8](#_Toc185434095)

[Interpréteur 10](#_Toc185434096)

[Définitions 11](#_Toc185434097)

[Conditionnelles 11](#_Toc185434098)

[Boucles 13](#_Toc185434099)

[Boucles finies 13](#_Toc185434100)

[Boucles infinies 14](#_Toc185434101)

[Aide 15](#_Toc185434102)

[Package ‘**core**’ 15](#_Toc185434103)

[Mot **‘bye’** (sortir de la console) 15](#_Toc185434104)

[Mot **‘words’** (obtenir la liste des mots du dictionnaire) 15](#_Toc185434105)

[Mots **‘(‘** et **‘)’** (écrire un commentaire) 15](#_Toc185434106)

[Mot ‘**variables**’ (obtenir une liste des variables et leurs valeurs) 15](#_Toc185434107)

[Mot ‘**constants**’ (obtenir une liste des constantes et leurs valeurs) 16](#_Toc185434108)

[Mot ‘**packages**’ (obtenir une liste des packages chargés dans Beetle) 16](#_Toc185434109)

[Mot ‘**!**’ (affecter une variable) 16](#_Toc185434110)

[Mot ‘**%**’ (obtenir un pourcentage d’un nombre) 16](#_Toc185434111)

[Mot ‘**\***’ (multiplication entre 2 nombres) 16](#_Toc185434112)

[Mot ‘**\*!**’ (Multiplication du contenu d’une variable avec un nombre) 16](#_Toc185434113)

[Mot ‘**\*/**’ 17](#_Toc185434114)

[Mot ‘**+**’ 17](#_Toc185434115)

[Mot ‘**+!**’ 17](#_Toc185434116)

[Mot ‘**+loop**’ 17](#_Toc185434117)

[Mot ‘**-**‘ 17](#_Toc185434118)

[Mot ‘**.**’ 17](#_Toc185434119)

[Mot ‘**.cr**’ 17](#_Toc185434120)

[Mot ‘**.s**’ 17](#_Toc185434121)

[Mot ‘**.scr**’ 17](#_Toc185434122)

[Mot ‘**/**’ 17](#_Toc185434123)

[Mot ‘**0<**’ 17](#_Toc185434124)

[Mot ‘**0<=**’ 17](#_Toc185434125)

[Mot ‘**0=**’ 17](#_Toc185434126)

[Mot ‘**0>**’ 18](#_Toc185434127)

[Mot ‘**0>=**’ 18](#_Toc185434128)

[Mot ‘**1+**’ 18](#_Toc185434129)

[Mot ‘**1-**’ 18](#_Toc185434130)

[Mot ‘**2\***’ 18](#_Toc185434131)

[Mot ‘**2+**’ 18](#_Toc185434132)

[Mot ‘**2-**’ 18](#_Toc185434133)

[Mot ‘**2>r**’ 18](#_Toc185434134)

[Mot ‘**2drop**’ 18](#_Toc185434135)

[Mot ‘**2dup**’ 18](#_Toc185434136)

[Mot ‘**2over**’ 18](#_Toc185434137)

[Mot ‘**2r>**’ 18](#_Toc185434138)

[Mot ‘**2swap**’ 18](#_Toc185434139)

[Mot ‘**:**’ 18](#_Toc185434140)

[Mot ‘**;**’ 18](#_Toc185434141)

[Mot ‘**<**’ 18](#_Toc185434142)

[Mot ‘**<=**’ 18](#_Toc185434143)

[Mot ‘**<>**’ 18](#_Toc185434144)

[Mot ‘**=**’ 19](#_Toc185434145)

[Mot ‘**>**’ 19](#_Toc185434146)

[Mot ‘**>=**’ 19](#_Toc185434147)

[Mot ‘**>r**’ 19](#_Toc185434148)

[Mot ‘**?**’ 19](#_Toc185434149)

[Mot ‘**@**’ 19](#_Toc185434150)

[Mot ‘**@r**’ 19](#_Toc185434151)

[Mot ‘ [’ 19](#_Toc185434152)

[Mot ‘]’ 19](#_Toc185434153)

[Mot ‘again’ 19](#_Toc185434154)

[Mot ‘and’ 19](#_Toc185434155)

[Mot ‘array’ 19](#_Toc185434156)

[Mot ‘base’ 19](#_Toc185434157)

[Mot ‘base!’ 19](#_Toc185434158)

[Mot ‘begin’ 19](#_Toc185434159)

[Mot ‘bl’ 19](#_Toc185434160)

[Mot ‘cell!’ 19](#_Toc185434161)

[Mot ‘cell+’ 19](#_Toc185434162)

[Mot ‘cell-’ 20](#_Toc185434163)

[Mot ‘cell=’ 20](#_Toc185434164)

[Mot ‘cell?’ 20](#_Toc185434165)

[Mot ‘cell@’ 20](#_Toc185434166)

[Mot ‘cells’ 20](#_Toc185434167)

[Mot ‘cells?’ 20](#_Toc185434168)

[Mot ‘char’ 20](#_Toc185434169)

[Mot ‘chars’ 20](#_Toc185434170)

[Mot ‘cls’ 20](#_Toc185434171)

[Mot ‘clt’ 20](#_Toc185434172)

[Mot ‘const’ 20](#_Toc185434173)

[Mot ‘cr’ 20](#_Toc185434174)

[Mot ‘create’ 20](#_Toc185434175)

[Mot ‘decimal’ 20](#_Toc185434176)

[Mot ‘do’ 20](#_Toc185434177)

[Mot ‘does>’ 20](#_Toc185434178)

[Mot ‘drop’ 20](#_Toc185434179)

[Mot ‘dump’ 20](#_Toc185434180)

[Mot ‘dup’ 21](#_Toc185434181)

[Mot ‘else’ 21](#_Toc185434182)

[Mot ‘emit’ 21](#_Toc185434183)

[Mot ‘false’ 21](#_Toc185434184)

[Mot ‘force!’ 21](#_Toc185434185)

[Mot ‘forget’ 21](#_Toc185434186)

[Mot ‘format’ 21](#_Toc185434187)

[Mot ‘help’ 21](#_Toc185434188)

[Mot ‘hex’ 21](#_Toc185434189)

[Mot ‘if’ 21](#_Toc185434190)

[Mot ‘immediate’ 21](#_Toc185434191)

[Mot ‘import’ 21](#_Toc185434192)

[Mot ‘input’ 21](#_Toc185434193)

[Mot ‘invert’ 21](#_Toc185434194)

[Mot ‘leave’ 21](#_Toc185434195)

[Mot ‘list’ 21](#_Toc185434196)

[Mot ‘load’ 21](#_Toc185434197)

[Mot ‘loop’ 21](#_Toc185434198)

[Mot ‘max’ 22](#_Toc185434199)

[Mot ‘min’ 22](#_Toc185434200)

[Mot ‘negate’ 22](#_Toc185434201)

[Mot ‘nip’ 22](#_Toc185434202)

[Mot ‘octal’ 22](#_Toc185434203)

[Mot ‘or’ 22](#_Toc185434204)

[Mot ‘over’ 22](#_Toc185434205)

[Mot ‘path’ 22](#_Toc185434206)

[Mot ‘r>’ 22](#_Toc185434207)

[Mot ‘r@’ 22](#_Toc185434208)

[Mot ‘rdrop’ 22](#_Toc185434209)

[Mot ‘rdump’ 22](#_Toc185434210)

[Mot ‘rdup’ 22](#_Toc185434211)

[Mot ‘recurse’ 22](#_Toc185434212)

[Mot ‘repeat’ 22](#_Toc185434213)

[Mot ‘rot’ 22](#_Toc185434214)

[Mot ‘rover’ 22](#_Toc185434215)

[Mot ‘rswap’ 22](#_Toc185434216)

[Mot ‘see’ 23](#_Toc185434217)

[Mot ‘space’ 23](#_Toc185434218)

[Mot ‘swap’ 23](#_Toc185434219)

[Mot ‘then’ 23](#_Toc185434220)

[Mot ‘true’ 23](#_Toc185434221)

[Mot ‘until’ 23](#_Toc185434222)

[Mot ‘var’ 23](#_Toc185434223)

[Mot ‘while’ 23](#_Toc185434224)

[Mot ‘xor’ 23](#_Toc185434225)

[Package ‘**math**’ 23](#_Toc185434226)

[Mot ‘**#fib**’ (obtenir le nième nombre de la suite de Fibonacci) 23](#_Toc185434227)

[Mot ‘**fib**’ 23](#_Toc185434228)

[Mot ‘**g**’ 23](#_Toc185434229)

[Mot ‘**h**’ 23](#_Toc185434230)

[Mot ‘**pi**’ 23](#_Toc185434231)

[Mot ‘**e**’ 23](#_Toc185434232)

[Mot ‘**c**’ 24](#_Toc185434233)

[Mot ‘**cube**’ 24](#_Toc185434234)

[Mot ‘**square**’ 24](#_Toc185434235)

[Mot ‘**abs**’ 24](#_Toc185434236)

[Mot ‘3pi/2’ 24](#_Toc185434237)

[Mot ‘3pi/2>deg’ 24](#_Toc185434238)

[Mot ‘3pi/4’ 24](#_Toc185434239)

[Mot ‘3pi/4>deg’ 24](#_Toc185434240)

[Mot ‘cbrt’ 24](#_Toc185434241)

[Mot ‘ceil’ 25](#_Toc185434242)

[Mot ‘comb’ 25](#_Toc185434243)

[Mot ‘cos’ 25](#_Toc185434244)

[Mot ‘deg’ 25](#_Toc185434245)

[Mot ‘exp’ 25](#_Toc185434246)

[Mot ‘exp2’ 25](#_Toc185434247)

[Mot ‘expm1’ 25](#_Toc185434248)

[Mot ‘fact’ 25](#_Toc185434249)

[Mot ‘floor’ 25](#_Toc185434250)

[Mot ‘log’ 25](#_Toc185434251)

[Mot ‘log10’ 25](#_Toc185434252)

[Mot ‘log1p’ 25](#_Toc185434253)

[Mot ‘log2’ 25](#_Toc185434254)

[Mot ‘pi/2’ 25](#_Toc185434255)

[Mot ‘pi/2>deg’ 25](#_Toc185434256)

[Mot ‘pi/3’ 25](#_Toc185434257)

[Mot ‘pi/3>deg’ 25](#_Toc185434258)

[Mot ‘pi/4’ 25](#_Toc185434259)

[Mot ‘pi/4>deg’ 25](#_Toc185434260)

[Mot ‘pi>deg’ 26](#_Toc185434261)

[Mot ‘pow’ 26](#_Toc185434262)

[Mot ‘rad’ 26](#_Toc185434263)

[Mot ‘remainder’ 26](#_Toc185434264)

[Mot ‘sin’ 26](#_Toc185434265)

[Mot ‘sqrt’ 26](#_Toc185434266)

[Mot ‘tan’ 26](#_Toc185434267)

[Mot ‘tau’ 26](#_Toc185434268)

[Mot ‘trunc’ 26](#_Toc185434269)

[Package ‘**file**’ 26](#_Toc185434270)

[Mot ‘**>b**’ 26](#_Toc185434271)

[Mot ‘**b>f**’ 26](#_Toc185434272)

[Mot ‘**c>**’ 26](#_Toc185434273)

[Mot ‘**cf**’ 26](#_Toc185434274)

[Mot ‘**f>**’ 26](#_Toc185434275)

[Mot ‘**l>**’ 26](#_Toc185434276)

[Mot ‘**rof**’ 26](#_Toc185434277)

[Mot ‘**wof**’ 27](#_Toc185434278)

# Introduction

Le langage Beetle est un langage proche de Forth. Il est écrit en Python. Le but de ce langage est de proposer une programmation simple en utilisant la puissance de Python.

Comme pour Forth, Beetle se base sur un dictionnaire de mots que vous pouvez enrichir à votre guise.

Vous devrez vous familiariser avec la notation polonaise inversée (qui offre des facilités de traitement des données)

En 1958, un jeune informaticien nommé Charles Moore, chargé de calculer des trajectoires de satellites, entreprend, pour faciliter son travail quotidien, la construction d’une boîte à outils sous la forme d’un interpréteur de commandes. D’année en année, cet interpréteur est amélioré et rendu indépendant de la machine hôte. En 1968, il prend le nom de Forth, et en 1970, il est suffisamment mûr pour faire l’objet d’une première publication comme « Langage pour calcul interactif ».

1971 voit la première application d’envergure : Moore utilise Forth pour développer le logiciel de pilotage du radiotéléscope de Kitt Peak (Arizona) sur deux mini-ordinateurs 16-bits. Il y est bientôt rejoint par Elizabeth Rather, qui devient le deuxième programmeur Forth. Par ses performances et sa souplesse d’emploi, l’application intéresse rapidement d’autres observatoires, et en 1976, Forth est adopté comme standard par l’Union internationale d’astronomie.

Après une première modernisation du logiciel de Kitt Peak en 1973, Moore et Rather fondent Forth, Inc., pour promouvoir le langage et ses applications. En 1976, une première version exécutable sur microprocessaurs 8-bits est disponible sous le nom de MicroFORTH.

En 1978 est créée une association, le Forth Interest Group (FIG), pour promouvoir une version « domaine public » du langage sur un maximum de processeurs, et publier la revue Forth Dimensions. Le FIG-FORTH, légèrement différent du MicroFORTH, contribuera grandement à la diffusion du langage.

Le besoin de standardisation d’un langage que chacun peut modifier à sa guise devient manifeste. Une première tentative (FORTH77) aboutit au premier standard largement diffusé, FORTH79. Ce standard évolue quatre ans plus tard en FORTH83, mais des incompatibilités entre les deux versions engendrent des problèmes de portabilité, et sont à l’origine d’un clivage dans la communauté des programmeurs FORTH.

# Notation polonaise inversée

Nommage barbare d’un outil bien pratique pour traiter les données de vos programmes Beetle. Vous allez devoir vous familiariser avec ce principe.

La **notation polonaise inversée** (NPI) (en anglais **RPN** pour Reverse Polish Notation), également connue sous le nom de **notation post-fixée**, permet d’écrire de façon non ambiguë les formules arithmétiques sans utiliser de parenthèses. Dérivée de la notation polonaise présentée en 1924 par le mathématicien polonais [Jan Łukasiewicz](https://fr.wikipedia.org/wiki/Jan_%C5%81ukasiewicz), elle s’en différencie par l’ordre des termes, les opérandes y étant présentés avant les opérateurs et non l’inverse.

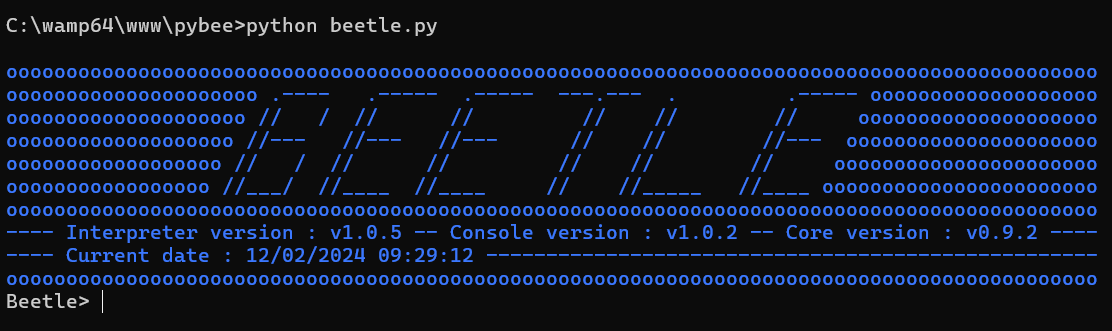
2 + 4 \* 5 va s’écrire en NPI : 4 5 \* 2 + mais aussi 5 4 \* 2 + (car la multiplication est commutative). Cela donne 22. Mais ne sera pas 2 4 + 5 \* qui donnera un autre résultat (30). Pour que le résultat soit juste, votre calcul doit prendre la forme suivante : (2 + 4) \* 5. La NPI impose un ordre d’importance aux opérateurs sans les écrire. C’est son avantage.

# Piles de données

Beetle repose sur la manipulation de piles LIFO (Last In, First Out) pour les données ou la pile de retour pour les données temporaires et sur un ensemble de mots de base. A partir de ces mots de base, il est possible d’enrichir le vocabulaire de mots de base en apportant ses propres mots.

# Console

Avec Beetle, vous allez pouvoir taper vos commandes dans un terminal. Voici une illustration de la console quand vous la lancez pour la première fois :



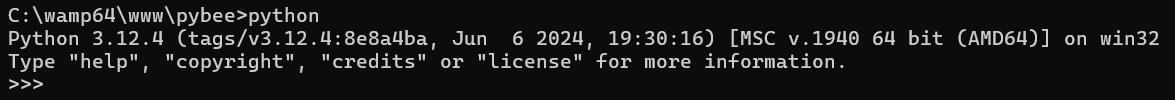
La commande qu’il faut lancer dans le terminal pour lancer Beetle est :

C:\...>python beetle.py

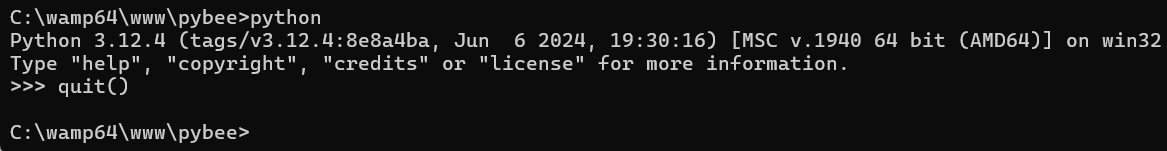
‘Beetle> ’ est le prompt de la console.

La console est une boucle infinie sur la fonction input qui vous permet de taper vos commandes. Les commandes ou mots que vous taper dans la console sont insensibles à la casse.

Ce manuel va utiliser cette console pour illustrer les mots de votre langage. C’est exactement le même principe que la console de Python. Voici la console de Python :



Vous devez taper **quit()** pour en sortir.



Dans Beetle, il existe un mot simple pour sortir de la console : **bye**

Essayer de taper bye dans la console : vous allez immédiatement sortir de la console. Voici ce que cela donne :



Vous pouvez choisir d’utiliser la console pour réaliser vos programmes. Cependant, vous pouvez rediriger la sortie standard vers une page web en utilisant l’outil mis à votre disposition qui est, dans votre navigateur préféré :

|  |
| --- |
| <http://localhost/pybee/webout.py?file=test> |

Vous aurez préalablement créé un fichier **test.btl** que vous aurez placé dans le répertoire **userarea**.

Dans le répertoire **userarea**, vous pouvez créer l’arborescence que vous souhaitez pour ranger vos fichiers ou projets.

Bien sûr, vous devez avoir lancer un serveur web de type Apache pour faire fonctionner l’interpréteur sur votre fichier

Exemple :

Si vous créez un répertoire projet1 dans **userarea** et que vous y mettez un fichier **myprogram.btl**, votre url sera la suivante :

|  |
| --- |
| <http://localhost/pybee/webout.py?file=projet1/myprogram> |

Autre exemple :

En admettant que vous avez créé un fichier **test.btl** contenant :

: newfib

dup 1 >= ( premier test )

if

dup 1 = ( deuxième test )

if

0 .cr drop

else

2 var i

1 0 2dup . . swap rot i

do ( début de la boucle )

2dup + dup . rot drop

loop ( fin de la bloucle )

2drop forget i cr

then

else

"Usage : n FIB with n > 0" .cr

then

;

50 newfib

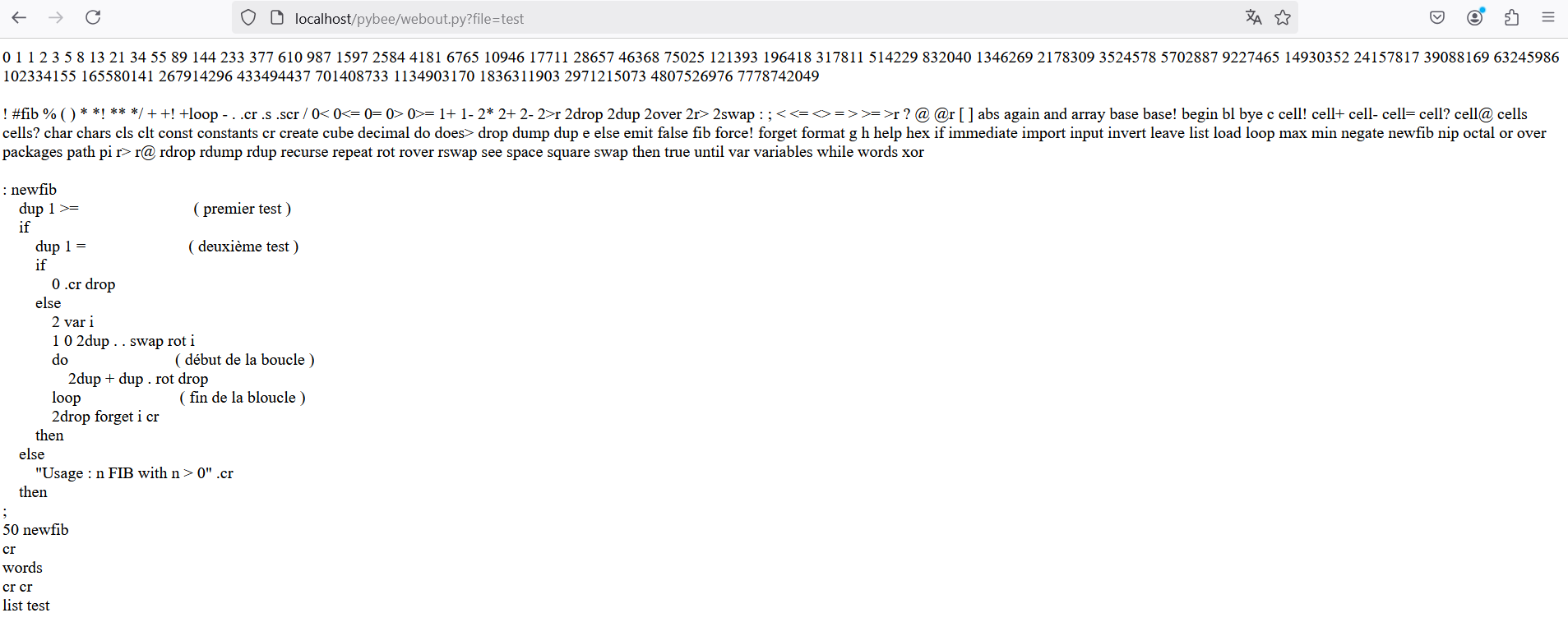
cr

words

cr cr

list test

Lorsque vous tapez : <http://localhost/pybee/webout.py?file=test> vous obtenez :



Si vous commettez une erreur de syntaxe, vous afficherez les erreurs dans la page web.

Ainsi vous pouvez utiliser un autre canal de sortie que la simple console.

# Interpréteur

L’interpréteur Beetle va interpréter l’ensemble des mots que vous taper dans la console. Si vous commettez une erreur de syntaxe, l’interpréteur va vous expliquer quoi et ou votre erreur a été faite.

Le package ‘**core**’ est automatiquement chargé dans l’interpréteur. Vous ne pouvez pas charger 2 fois le package ‘**core**’. Les mots du package ‘**core**’ sont attribués à l’interpréteur. Celui-ci va vous permettre :

* De créer des définitions
* D’inclure des conditionnelles
* D’inclure des boucles
* D’obtenir de l’aide

L’interpréteur ne fait que reconnaitre :

* Les entiers (1, 12, -5, …)
* Les nombres décimaux (1.2, 1e-10, 14E15, …)
* Les chaines de caractères ( "chaine de caractères", …)Le caractère " délimite les chaines de caractères et seulement ce caractère
* Et les mots du dictionnaire

Tous les mots doivent être séparés par le caractère ESPACE

## Définitions

Créer un mot revient à définir un mot. Ce mot peut être considérer comme une fonction dans un langage classique connu tel que C. Ce mot vient naturellement enrichir le vocabulaire existant. Je rappelle que le vocabulaire initial est le vocabulaire du package ‘**core**’. Vous pouvez ajouter autant de mots que vous le souhaitez dans le dictionnaire de mots de **Beetle** et vous pouvez utiliser vos définitions dans d’autres mots que vous définissez plus tard.

La syntaxe d’une définition est la suivante :

**:** mot[**(** aide **)**][ instructions != : ]**;**

* **:** marque le début de la définition
* **;** marque la fin d’une définition
* **( aide )** vous offre la possibilité de créer de l’aide facultative pour votre mot
* **[ ]** indique les parties facultatives de votre définition

Exemple de définition de mot :

**: carre ( Obtenir le carré d’une valeur ) dup \* ;**

Usage : **5 carre .** 🡺 affiche 25

Que se passe-t-il vraiment ?

* Le mot carre est ajouté au dictionnaire (ou vocabulaire) de Beetle
* Une entrée est ajoutée dans l’aide de Beetle avec la phrase que vous avez mis entre parenthèses

Attention : l’utilisation de ; sans : est impossible.

Attention : vous ne pouvez pas imbriquer les définitions

## Conditionnelles

Les conditionnelles vous permettent d’exécuter des instructions suivant le résultat d’une condition.

La syntaxe des conditionnelles est la suivante :

condition **IF** instructions1[ **ELSE** instructions2] **THEN**

* **condition** : un booléen représenté par TRUE ( différent de 0) ou FALSE (0)
* **condition = TRUE** : exécute les instructions de instructions1
* **condition = FALSE** : exécute les instructions de instructions2. Cette partie est facultative. Si celle-ci n’existe pas, l’interpréteur continue son exécution.

La condition peut être calculée. Un ensemble de mots sont mis à disposition pour effectuer ce calcul : **=**, **<=**, **>=**, **!=**, **<**, **>**, **0<**, **0<=**, **0=**, **0>**, **0>=**, **INVERT**, **AND**, **OR**, **XOR**, **TRUE**, **FALSE**

* **=** : égalité
* **<=** : inférieur ou égal
* **>=** : supérieur ou égal
* **!=** : différent de
* **<** : inférieur
* **>** : supérieur
* **0<** : inférieur à 0
* **0<=** : inférieur ou égal à 0
* **0=** : égal à 0
* **0>** : supérieur à 0
* **0>=** : supérieur ou égal à 0
* **INVERT** : inversion (correspond au non logique)
* **AND** : et logique
* **OR** : ou logique
* **XOR** : ou exclusif
* **TRUE** : 1
* **FALSE** : 0

Exemples de conditionnelle :

1 0 > IF "true zone" . ELSE "false zone" . THEN

Affiche 🡺 true zone

1 0 < IF "true zone" . ELSE "false zone" . THEN

Affiche 🡺 false zone

1 0 < IF "true zone" . THEN

Affiche 🡺 rien, le programme continue avec les instructions après THEN

Il n’y a pas de restrictions à l’imbrication des conditionnelles.

Exemple d’imbrication de conditionnelle :

DUP 1 >=

**IF**

DUP 1 =

**IF**

0 . DROP

**ELSE**

2 VAR i 1 0 2DUP . . SWAP ROT i DO 2DUP + DUP . ROT DROP LOOP 2DROP FORGET i CR

**THEN**

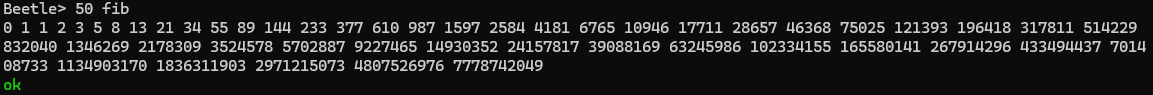
**ELSE**

"Usage : n FIB with n > 0" .CR

**THEN**

Cette séquence d’instructions correspond au mot FIB que vous trouvez naturellement dans votre dictionnaire et vous affiche tous les nombres de Fibonacci. Son usage est le suivant : 50 FIB

Cela vous donne le résultat suivant :



## Boucles

Plusieurs types de boucles sont possibles :

* Boucles finies
* Boucles infinies

### Boucles finies

Vous avez à disposition 2 types de boucles finies.

La syntaxe du premier type de boucle et la plus simple est la suivante :

limit variable **DO** instructions **LOOP**

* **limit** : fin de l’incrément de la boucle
* **variable** : début de l’incrément de la boucle dans une variable. L’incrément est stocké dans une variable que vous pouvez utiliser dans la boucle. La variable que vous utilisez pour la valeur initiale de votre boucle est automatiquement incrémentée de 1 en 1

Exemple pour le premier type de boucle :

0 VAR I

50 I DO I @ . LOOP

Affiche 🡺 tous les nombres de 0 à 49

La syntaxe du deuxième type de boucle est la suivante :

limit variable **DO** instructions incrément **+LOOP**

* **limit** : fin de l’incrément de la boucle
* **variable** : début de l’incrément de la boucle dans une variable. L’incrément est stocké dans une variable que vous pouvez utiliser dans la boucle. La variable que vous utilisez pour la valeur initiale de votre boucle est automatiquement incrémentée de 1 en 1
* **incrément**: incrément (ou pas de boucle) qui peut être positif ou négatif. Si limit < contenu de la variable, l’incrément doit être négatif sinon vous faites une boucle infinie

Exemple pour le second type de boucle :

0 VAR I

50 I DO I @ . 5 +LOOP

Affiche 🡺 tous les nombres entre 0 et 45 incluent de 5 en 5

### Boucles infinies

Vous avez le choix entre 3 types de boucles infinies

Le premier type de boucle infinie vous permet de définir une condition d’arrêt. La syntaxe de ce type de boucle est la suivante :

**BEGIN** instructions flag **UNTIL**

* **flag** : représente la condition d’arrêt de votre boucle. Si flag = false alors on continue dans la boucle. Si flag = true alors on sort de la boucle
* **instructions** : corresponds aux instructions à exécuter dans la boucle (sont comprises les instructions permettant de calculer le flag)

Exemple :

0 VAR I

BEGIN I @ . 1 I +! I @ 50 > UNTIL

Affiche 🡺 tous les nombres de 0 à 49

Le deuxième type de boucle infinie est une boucle sans condition d’arrêt. Pour sortir de cette boucle, vous devrez utiliser l’instruction LEAVE. La syntaxe de ce type de boucle est la suivante :

**BEGIN** instructions **AGAIN**

* **instructions** : corresponds aux instructions à exécuter dans la boucle

Exemple :

0 VAR I

BEGIN I @ . 1 I +! AGAIN

Affiche 🡺 tous les nombres à partir de 0 indéfiniment

Le troisième et dernier type de boucle que vous pouvez utiliser est une boucle un peu particulière. Voici sa syntaxe :

**BEGIN** instructions1 flag **WHILE** instructions2 **REPEAT**

* **flag** : représente la condition d’arrêt de votre boucle. Si flag = true alors on continue dans la boucle à partir du début de instructions1 en ayant exécuté instructions2. Si flag = false alors on sort de la boucle.
* **instructions1** : corresponds aux instructions entre BEGIN et WHILE. Le calcul du flag est inclus dans les instructions1
* **instructions2** : corresponds aux instructions entre WHILE et REPEAT

Exemple :

0 VAR I

BEGIN I @ . I @ 50 < WHILE 1 I +! REPEAT

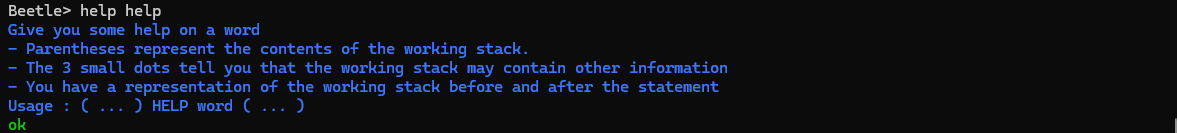
Affiche 🡺 tous les nombres de 0 à 49

Pour sortir de tous ces types de boucle, vous pouvez utiliser le mot LEAVE. L’instruction vous fait sortir de la boucle à l’endroit où il est écrit. LEAVE en dehors d’une boucle n’a aucun effet.

## Aide

La commande ‘help’ vous permet d’obtenir de l’aide sur un mot en particulier. L’aide est affichée en bleu dans la console.

Par exemple, l’aide sur le mot ‘help’ vous affichera ceci :



Ou encore l’aide sur le mot ‘dup’ vous donnera ceci :



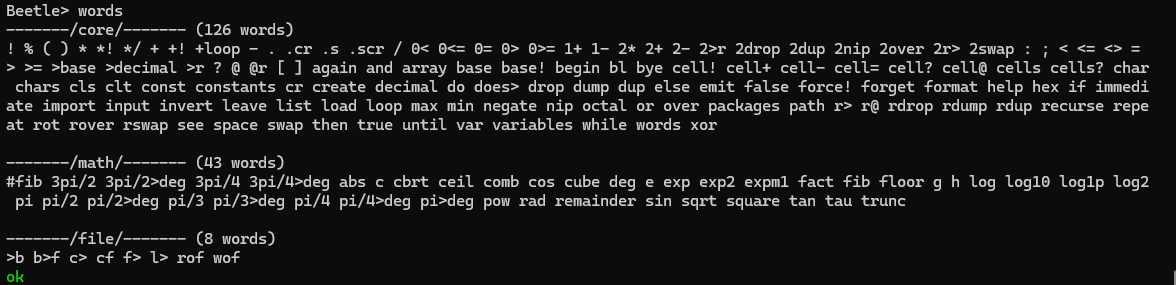
# Package ‘**core**’

## Mot **‘bye’** (sortir de la console)

Le mot bye vous permet de quitter la console de Beetle. Une illustration de ce mot a été donnée plus haut.

## Mot **‘words’** (obtenir la liste des mots du dictionnaire)

Le mot ‘words’ va vous permettre de lister l’ensemble des mots disponible dans le langage. La liste est fournie par package.



## Mots **‘(‘** et **‘)’** (écrire un commentaire)

Ces 2 mots vous permettent d’écrire un commentaire dans vos programmes comme :



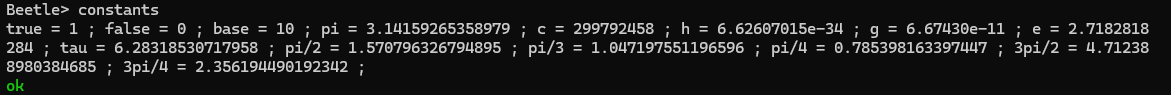
Bien sûr, cela ne fait pas grand-chose mais cela deviendra utile quand vous écrirez des programmes plus complexes.

## Mot ‘**variables**’ (obtenir une liste des variables et leurs valeurs)



Pour l’instant, vous n’avez qu’une seule variable de définie ‘path’ qui désigne le répertoire ou se trouve vos programmes Beetle.

## Mot ‘**constants**’ (obtenir une liste des constantes et leurs valeurs)



Vous pouvez constater qu’un certain nombre de constantes sont prédéfinies comme la vitesse de la lumière ou la constante de Planck, pi, etc…

## Mot ‘**packages**’ (obtenir une liste des packages chargés dans Beetle)



En tapant ‘packages’ dans la console, vous obtenez la liste des packages que vous avez chargé. Le package ‘core’ est automatiquement chargé et vous ne pouvez pas le charger une deuxième fois.

## Mot ‘**!**’ (affecter une variable)

Ce mot vous permet d’affecter une valeur à une variable. Vous pouvez affecter un entier, un float, un tableau ou une chaine de caractères.

Exemple :



## Mot ‘**%**’ (obtenir un pourcentage d’un nombre)

Comme ce mot le montre, il va vous donner le pourcentage d’un nombre.

Exemple : 32% de 225 vous donne :



## Mot ‘**\***’ (multiplication entre 2 nombres)

Pour obtenir la multiplication entre 2 nombres, voici un exemple :



## Mot ‘**\*!**’ (Multiplication du contenu d’une variable avec un nombre)

Ce mot est très pratique pour multiplier le contenu d’une variable par un nombre et affecte directement le contenu de cette même variable avec le résultat.

Exemples :

* Je créé une variable qui se nomme ‘test’ à laquelle j’affecte initialement la valeur 5.
* Je multiplie 6 au contenu de cette variable.
* J’affiche le contenu de cette variable qui est maintenant 30
* Je multiplie de nouveau le contenu de la variable par 1.2
* J’affiche le contenu de cette variable qui est maintenant 36.0 (car la dernière multiplication était effectuée avec un float)



## Mot ‘**\*/**’

## Mot ‘**+**’

## Mot ‘**+!**’

## Mot ‘**+loop**’

## Mot ‘**-**‘

## Mot ‘**.**’

## Mot ‘**.cr**’

## Mot ‘**.s**’

## Mot ‘**.scr**’

## Mot ‘**/**’

## Mot ‘**0<**’

## Mot ‘**0<=**’

## Mot ‘**0=**’

## Mot ‘**0>**’

## Mot ‘**0>=**’

## Mot ‘**1+**’

## Mot ‘**1-**’

## Mot ‘**2\***’

## Mot ‘**2+**’

## Mot ‘**2-**’

## Mot ‘**2>r**’

## Mot ‘**2drop**’

## Mot ‘**2dup**’

## Mot ‘**2over**’

## Mot ‘**2r>**’

## Mot ‘**2swap**’

## Mot ‘**:**’

## Mot ‘**;**’

## Mot ‘**<**’

## Mot ‘**<=**’

## Mot ‘**<>**’

## Mot ‘**=**’

## Mot ‘**>**’

## Mot ‘**>=**’

## Mot ‘**>r**’

## Mot ‘**?**’

## Mot ‘**@**’

## Mot ‘**@r**’

## Mot ‘ [’

## Mot ‘]’

## Mot ‘again’

## Mot ‘and’

## Mot ‘array’

## Mot ‘base’

## Mot ‘base!’

## Mot ‘begin’

## Mot ‘bl’

## Mot ‘cell!’

## Mot ‘cell+’

## Mot ‘cell-’

## Mot ‘cell=’

## Mot ‘cell?’

## Mot ‘cell@’

## Mot ‘cells’

## Mot ‘cells?’

## Mot ‘char’

## Mot ‘chars’

## Mot ‘cls’

## Mot ‘clt’

## Mot ‘const’

## Mot ‘cr’

## Mot ‘create’

## Mot ‘decimal’

## Mot ‘do’

## Mot ‘does>’

## Mot ‘drop’

## Mot ‘dump’

## Mot ‘dup’

## Mot ‘else’

## Mot ‘emit’

## Mot ‘false’

## Mot ‘force!’

## Mot ‘forget’

## Mot ‘format’

## Mot ‘help’

## Mot ‘hex’

## Mot ‘if’

## Mot ‘immediate’

## Mot ‘import’

## Mot ‘input’

## Mot ‘invert’

## Mot ‘leave’

## Mot ‘list’

## Mot ‘load’

## Mot ‘loop’

## Mot ‘max’

## Mot ‘min’

## Mot ‘negate’

## Mot ‘nip’

## Mot ‘octal’

## Mot ‘or’

## Mot ‘over’

## Mot ‘path’

## Mot ‘r>’

## Mot ‘r@’

## Mot ‘rdrop’

## Mot ‘rdump’

## Mot ‘rdup’

## Mot ‘recurse’

## Mot ‘repeat’

## Mot ‘rot’

## Mot ‘rover’

## Mot ‘rswap’

## Mot ‘see’

## Mot ‘space’

## Mot ‘swap’

## Mot ‘then’

## Mot ‘true’

## Mot ‘until’

## Mot ‘var’

## Mot ‘while’

## Mot ‘xor’

# Package ‘**math**’

## Mot ‘**#fib**’ (obtenir le nième nombre de la suite de Fibonacci)

Ce mot ne fait rien d’autre que d’afficher le nième nombre de la suite de Fibonacci.

Exemple :



## Mot ‘**fib**’

## Mot ‘**g**’

## Mot ‘**h**’

## Mot ‘**pi**’

## Mot ‘**e**’

## Mot ‘**c**’

## Mot ‘**cube**’

## Mot ‘**square**’

## Mot ‘**abs**’

## Mot ‘3pi/2’

## Mot ‘3pi/2>deg’

## 

## Mot ‘3pi/4’

## 

## Mot ‘3pi/4>deg’

## 

## Mot ‘cbrt’

## Mot ‘ceil’

## 

## Mot ‘comb’

## 

## Mot ‘cos’

## 

## Mot ‘deg’

## 

## Mot ‘exp’

## 

## Mot ‘exp2’

## 

## Mot ‘expm1’

## 

## Mot ‘fact’

## 

## Mot ‘floor’

## 

## Mot ‘log’

## 

## Mot ‘log10’

## 

## Mot ‘log1p’

## 

## Mot ‘log2’

## 

## Mot ‘pi/2’

## 

## Mot ‘pi/2>deg’

## 

## Mot ‘pi/3’

## 

## Mot ‘pi/3>deg’

## 

## Mot ‘pi/4’

## 

## Mot ‘pi/4>deg’

## 

## Mot ‘pi>deg’

## 

## Mot ‘pow’

## 

## Mot ‘rad’

## 

## Mot ‘remainder’

## 

## Mot ‘sin’

## 

## Mot ‘sqrt’

## 

## Mot ‘tan’

## 

## Mot ‘tau’

## 

## Mot ‘trunc’

# Package ‘**file**’

## Mot ‘**>b**’

## Mot ‘**b>f**’

## Mot ‘**c>**’

## Mot ‘**cf**’

## Mot ‘**f>**’

## Mot ‘**l>**’

## Mot ‘**rof**’

## Mot ‘**wof**’