Interpréteur Serpent

Guide d'utilisation

Version 1

Auteur

Yanis Nebbaki (approfondissement, code de base HowCode)

Réalisé dans le cadre du cours Interprétation et Compilation d'une L3 Informatique au sein de l'Institut d'Enseignement à Distance de l'Université Paris 8.

Description

Le langage *Serpent* est un langage basique personnalisé ayant une syntaxe très proche du français. Il se spécialise dans les calculs trigonométriques pour la classe de troisième au collège. Il incorpore également quelques autres fonctionnalités simples. Il offre une interface en ligne de commande pour saisir les instructions. L'interpréteur est écrit en Python et utilise la bibliothèque *SLY* pour l'analyse lexicale et syntaxique.
La structure du code provient de *HowCode* (https://github.com/howCodeORG/2018-Programming-Language-Series)

Configuration requise

Le programme peut tourner sur tous les systèmes avec Python.
L'exécutable peut être lancé sur Linux.

Exécution Compilation

Pour exécuter le programme :

Depuis le fichier python (.py), il faut d'abord installer la librairie *SLY* puis exécuter le fichier .py :
python3 -m pip install sly
python3 serpent_interpreter.py

```
python3 serpent_interpreter.py
```

Depuis l'exécutable :
./serpent_interpreter

```
./serpent_interpreter
```

Fonctionnalités

Le langage n'accepte que les chiffres supérieurs ou égaux à 0. Les nombres à virgule sont acceptés.
Le langage ignore les espaces et les tabulations.

VARIABLE

Les syntaxes pour déclarer une variable sont :

- *nom_variable = nombre*

```
serpent > a = 1
serpent > a
1.0
```

- *nom_variable = chaîne de caractères*

```
serpent > b = "test"
serpent > b
"test"
```

Le nom de la variable doit respecter l'expression régulière suivant :
*[a-zA-Z_][a-zA-Z0-9_]**
Autrement dit, le nom de la variable doit commencer par une lettre majuscule ou minuscule. Les caractères suivants peuvent être une séquence vide ou non d'un nombre quelconque de caractère alphanumérique ou d'underscore _.
La chaîne de caractères commence et se termine par des guillemets : *"exemple"*.
Le contenu de la chaîne entre les guillemets peut être vide ou contenir un nombre quelconque de n'importe quel caractère.

Les variables avec les noms suivants ne sont pas utilisables :

dernier_sinus
dernier_cosinus
derniere_tangente
dernier_oppose
dernier_adjacent

dernier_hypotenuse
dernier_angle

Pour afficher ou se servir d'une variable dans un calcul, il suffit de taper le nom de la variable.

```
serpent > a = 1
serpent > 1 + a
2.0
```

```
serpent > a = "je suis a"
serpent > a
"je suis a"
```

OPÉRATIONS MATHÉMATIQUES SIMPLES

Pour faire un calcul, il faut suivre la syntaxe : *opérande opérateur opérande etc …*

```
serpent > 1 + 1
```

```
serpent > 1 + 1 + 1 + 1
4.0
```

Les opérateurs disponibles sont : addition (+), soustraction (-), multiplication (*), division (/).
Les opérandes sont des nombres supérieurs ou égaux à 0, entiers ou à virgule. Des variables peuvent être également utilisées.

```
serpent > a = 1
serpent > 1 + a
2.0
```

Le résultat s'affiche sur la ligne suivante.

```
serpent > 1 + 1
2.0
```

SI

Le test *SI* permet d'effectuer une action si une condition est remplie. Si la condition n'est pas remplie, une action différente est faite.
Un test *SI* suit la syntaxe suivante : *SI condition FAIRE action SINON action*
Pour cette version, uniquement le test d'égalité (==) est disponible.

```
serpent > a = 1
serpent > SI a == 1 FAIRE a = 2 SINON a = 3
serpent > a
2
serpent > SI a == 1 FAIRE a = 2 SINON a = 3
serpent > a
3
```

POUR

La boucle *FOR* permet d'effectuer une action jusqu'à ce qu'une condition soit remplie.
Une boucle *FOR* a pour syntaxe : *POUR assignation variable JSQ nombre FAIRE action.*
On assigne une valeur à une variable (existante ou non). On détermine une limite (un chiffre ou un nombre). On fait une action x fois, où x = limite - valeur variable.

```
serpent > POUR a = 1 JSQ 5 FAIRE a + 1
2
3
4
5
serpent > POUR x = 5 JSQ 25 FAIRE x * 5
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
```

FONCTION SANS ARGUMENT

Une fonction permet, quand elle est appelée, d'effectuer une certaine action définie au préalable. Les fonctions n'ont pas d'argument.

Pour créer une fonction il faut utiliser la syntaxe suivante : *FONC nom de la fonction () => action*
L'appel d'une fonction se fait via son nom avec des parenthèses :
nom de la fonction ()

```
serpent > FONC test () -> a = a * 10
serpent > a = 2
serpent > test()
serpent > a
20
serpent >
```

ECRIS

La commande *ECRIS* permet d'imprimer une chaîne de caractères.
La syntaxe est : *ECRIS "CHAÎNE DE CARACTÈRES"*
La chaîne de caractères doit être entre deux " et peut contenir n'importe quel caractère.

```
serpent > ECRIS "test"
"test"
serpent > ECRIS a
sly: Syntax error at line 1, token=NAME
serpent > ECRIS "qfjnkqfpioqf"
"qfjnkqfpioqf"
```

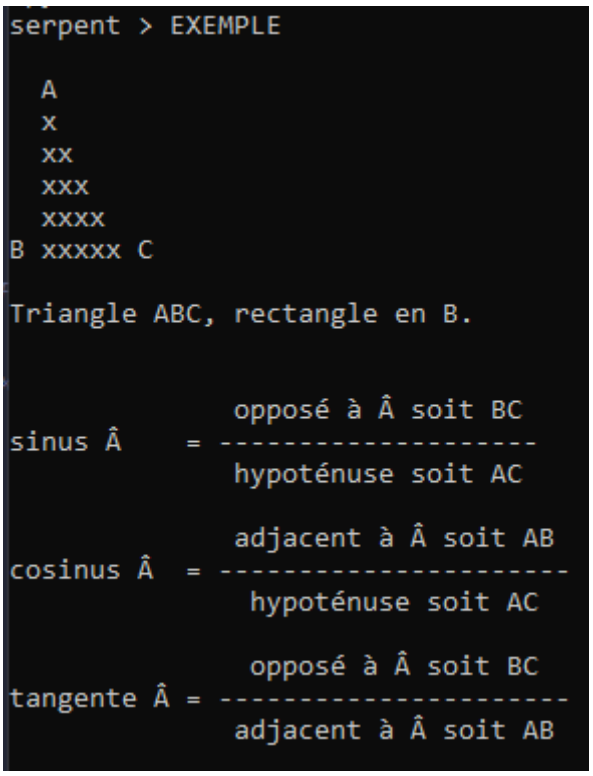
.

EFFACER

Pour effacer les lignes de la console, il suffit d'utiliser la commande *EFFACER.*

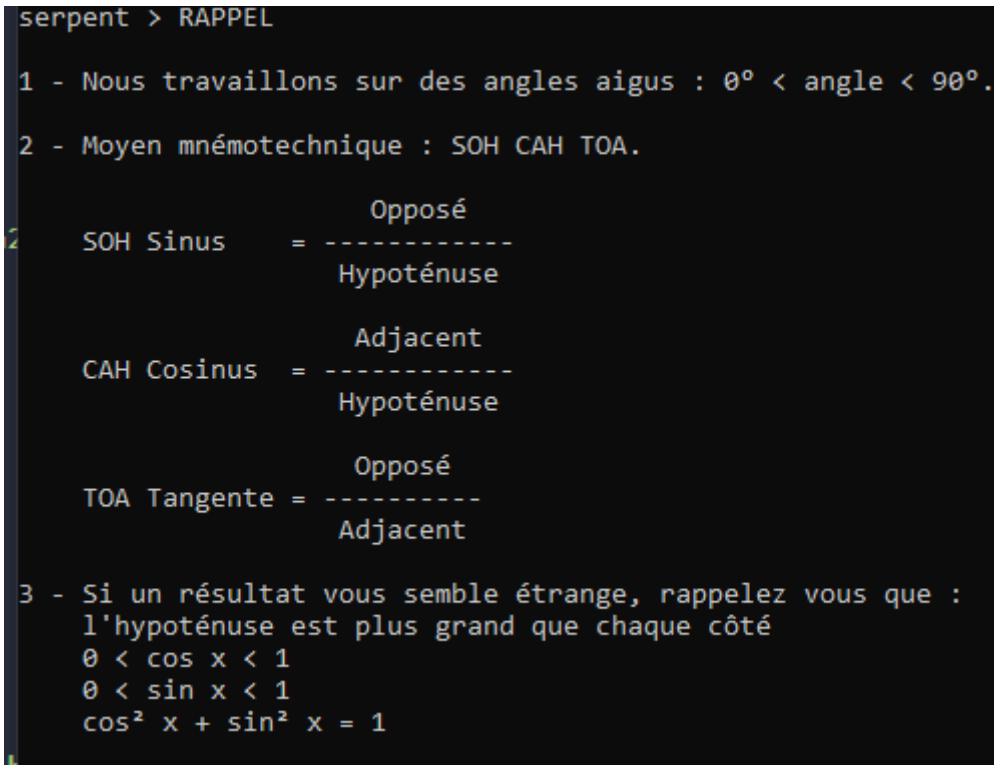
Exemple d'un triangle rectangle

Cette méthode permet d'afficher dans la console un triangle ABC rectangle en B.
A partir de ce rectangle, on affiche également les relations entre les angles et les côtés.
La syntaxe de la commande est : *EXEMPLE*
Il faut que la taille de la fenêtre soit suffisamment grande pour ne pas gêner la mise en forme.



Rappel relations trigonométriques

Cette fonction a pour but de regrouper toutes les propriétés et relations trigonométriques utiles en cours de troisième.
La syntaxe est : *RAPPEL*
Il faut que la taille de la fenêtre soit suffisamment grande pour ne pas gêner la mise en forme



CALCULS SOH CAH TOA et ses dérivés

Les calculs suivent tous la même syntaxe. La première lettre est ce qu'on souhaite calculer, la seconde et troisième les opérandes. Exemple pour SOH : sinus à calculer en utilisant l'opposé et l'hypoténuse. La syntaxe est :
SOH NOMBRE1 NOMBRE2
Le nombre 1 correspond à l'opposé (O) et le nombre 2 à l'hypoténuse (H).
Les nombres peuvent être entiers ou à virgule.
Voici la signification de chaque lettre :

- S : sinus
- O : opposé
- H : hypoténuse
- C : cosinus
- T : tangente

Si le calcul aboutit, on stocke le résultat arrondi au millième près dans une variable pour pouvoir y accéder ultérieurement. Cette variable sera écrasée à chaque fois qu'on appelle une commande qui vise à calculer la même chose (SOH et SIN partagent la même variable *dernier_sinus* par exemple).
Si le calcul n'a pas pu être fait, un message d'erreur expliquant le problème est produit comme celui-ci :

```
serpent > SOH 5 1
!!! Valeur de l'hypoténuse (1.0) inférieure/égale à celle du côté (5.0) !!!
```

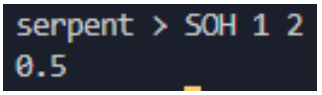
Les calculs suivants sont disponibles :

SOH

On calcule le sinus à partir de $sinus = \frac{opposé}{hypoténuse}$.
Syntaxe : *SOH NOMBRE1 NOMBRE2*
Nombre 1 : valeur de l'OPPOSÉ
Nombre 2 : valeur de l'HYPOTÉNUSE
Il y a deux vérifications :

- l'hypoténuse doit être strictement plus grande que l'opposé
- le sinus doit être compris entre 0 et 1

Le résultat est affiché sur la ligne suivante.
Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_sinus*.

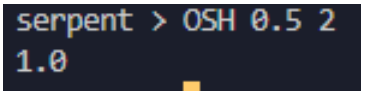


OSH

On calcule l'opposé à partir de $opposé = sinus * hypoténuse$
Syntaxe : *OSH NOMBRE1 NOMBRE2*
Nombre 1 : valeur du SINUS
Nombre 2 : valeur de l'HYPOTÉNUSE
Il y a deux vérifications :

- l'hypoténuse doit être strictement plus grande que l'opposé
- le sinus doit être compris entre 0 et 1

Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_oppose*.



HOS

On calcule l'hypoténuse à partir de $hypoténuse = \frac{opposé}{sinus}$

Syntaxe : *HOS NOMBRE1 NOMBRE2*
Nombre 1 : valeur de l'OPPOSÉ
Nombre 2 : valeur du SINUS
Il y a deux vérifications :

- l'hypoténuse doit être strictement plus grande que l'opposé
- le sinus doit être compris entre 0 et 1

Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_hypotenuse*.

```
serpent > HOS 1 0.1
10.0
```

.

SIN

On calcule le sinus à partir de l'angle.
Syntaxe : *SIN NOMBRE1*
Nombre 1 : valeur de l'angle
Il y a une vérification :

- l'angle doit être compris entre 0 et 90 degrés

Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_sinus*.

```
serpent > SIN 30
0.5
serpent > SIN 60
0.866
serpent > SIN 10
0.174
```

ASIN

On calcule l'angle à partir du sinus.
Syntaxe : *ASIN NOMBRE1*
Nombre 1 : valeur du sinus
Il y a une vérification :

- le sinus doit être compris entre 0 et 1

Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_angle*.

```
serpent > ASIN 0.5
30.0
serpent > ASIN 0.1
5.739
serpent > ASIN 0.99
81.89
```

CAH

On calcule le cosinus à partir de $cosinus = \frac{adjacent}{hypoténuse}$
Syntaxe : *CAH NOMBRE1 NOMBRE2*
Nombre 1 : valeur de l'ADJACENT
Nombre 2 : valeur de l'HYPOTÉNUSE
Il y a deux vérifications :

- l'hypoténuse doit être strictement plus grande que l'adjacent
- le cosinus doit être compris entre 0 et 1

Le résultat est affiché sur la ligne suivante.
Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier cosinus*.

```
serpent > CAH 4 10
0.4
serpent > CAH 2 41.525
0.048
serpent > CAH 2000 8888
0.225
```

ACH

On calcule l'adjacent à partir de $adjacent = cosinus * hypoténuse$
Syntaxe : *ACH NOMBRE1 NOMBRE2*
Nombre 1 : valeur du COSINUS
Nombre 2 : valeur de l'HYPOTÉNUSE
Il y a deux vérifications :

- l'hypoténuse doit être strictement plus grande que l'adjacent
- le cosinus doit être compris entre 0 et 1

Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_adjacent*.

```
serpent > ACH 0.7 2
1.4
serpent > ACH 0.44 5.5
2.42
serpent > ACH 0.0505 1.1
0.056
```

HAC

On calcule l'hypoténuse à partir de $hypoténuse = \frac{adjacent}{cosinus}$
Syntaxe : *HAC NOMBRE1 NOMBRE2*
Nombre 1 : valeur de l'ADJACENT
Nombre 2 : valeur du COSINUS
Il y a deux vérifications :

- l'hypoténuse doit être strictement plus grande que l'adjacent
- le cosinus doit être compris entre 0 et 1

Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_hypotenuse*.

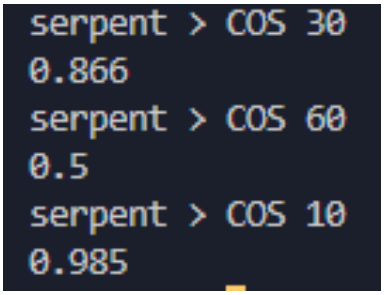
```
serpent > HAC 4 0.2
20.0
serpent > HAC 5 0.3
16.667
serpent > HAC 6 0.4
15.0
```

COS

On calcule le cosinus à partir de l'angle.
Syntaxe : *COS NOMBRE1*
Nombre 1 : valeur de l'angle
Il y a une vérification :

- l'angle doit être compris entre 0 et 90 degrés

Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_cosinus*.

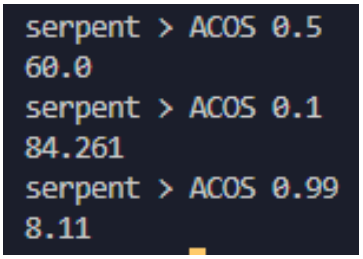


ACOS

On calcule l'angle à partir du cosinus.
Syntaxe : *ACOS NOMBRE1*
Nombre 1 : valeur du cosinus
Il y a une vérification :

- le cosinus doit être compris entre 0 et 1

Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_angle*.



TOA

On calcule la tangente à partir de $tangente = \frac{opposé}{adjacent}$

Syntaxe : *TOA NOMBRE1 NOMBRE2*
Nombre 1 : valeur de l'OPPOSÉ
Nombre 2 : valeur de l'ADJACENT
Le résultat est affiché sur la ligne suivante.
Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_tangente*.

```
serpent > TOA 50 10
5.0
serpent > TOA 8787 4
2196.75
serpent > TOA 200 777
0.257
```

OTA

On calcule l'opposé à partir de $opposé = tangente * adjacent$
Syntaxe : *OTA NOMBRE1 NOMBRE2*
Nombre 1 : valeur de la TANGENTE
Nombre 2 : valeur de l'ADJACENT
Le résultat est affiché sur la ligne suivante.
Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_oppose*.

```
serpent > OTA 1.4 4.4
6.16
serpent > OTA 2.5 0.3
0.75
serpent > OTA 150 2
300.0
```

AOT

On calcule l'adjacent à partir de $adjacent = \frac{opposé}{tangente}$

Syntaxe : *AOT NOMBRE1 NOMBRE2*
Nombre 1 : valeur de l'OPPOSÉ
Nombre 2 : valeur de la TANGENTE

Le résultat est affiché sur la ligne suivante.
Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_adjacent*.

```
serpent > AOT 2 8
0.25
serpent > AOT 3 0.5
6.0
serpent > AOT 40 80
0.5
```

TAN

On calcule la tangente à partir de l'angle.
Syntaxe : *TAN NOMBRE1*
Nombre 1 : valeur de l'angle
Il y a une vérification :

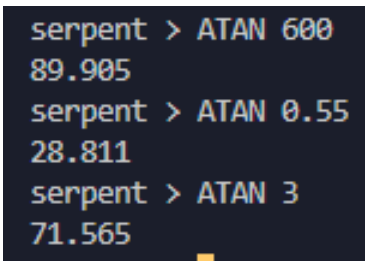
- l'angle doit être compris entre 0 et 90 degrés

Le résultat est affiché sur la ligne suivante.
Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_tangente*.

```
serpent > TAN 30
0.577
serpent > TAN 60
1.732
serpent > TAN 10
0.176
```

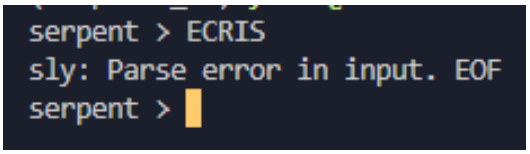
ATAN

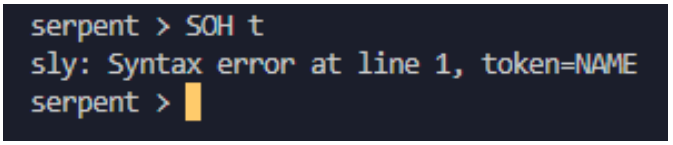
On calcule l'angle à partir de la tangente.
Syntaxe : *ATAN NOMBRE1*
Nombre 1 : valeur de la TANGENTE
Pour accéder au résultat ultérieurement, il faut utiliser la variable *dernier_angle*.



Erreurs de syntaxe

Les erreurs de syntaxe sont gérées directement par *SLY*. Voici les deux plus courantes :





La première erreur correspond à une absence d'argument. Dans cet exemple, il manque la chaîne de caractères après *ECRIS*.
La seconde erreur est due à un mauvais argument. La commande *SOH* attend en argument un nombre et ici on lui passe un caractère *t*.
Dans la documentation officielle se trouve tous les détails (https://sly.readthedocs.io/en/latest/sly.html#error-handling).