Faculté des sciences et techniques de Mohammedia

Filière ILISI

ShankScript
Un Langage de Programmation

Auteur:
ELKAISSI Souhail
LAGHEZALI Mohamed Réda

Encadrant:
A. BEKKHOUCHA

April 20, 2017

Abstract

ShankScript est un langage de programmation multi-paradigme et multiplateformes , il est doté d'une gestion automatique de la mémoire ,il est ainsi similaire à des langages de programmations comme Python ,C,C+ ...

Le Langage ShankScript est placé sous une licence libre ,il peut fonctionner dans la plateforme windows et aussi dans les systèmes d'exploitations UNIX (Ubuntu ,Fedora ,Kali ...) . Il est conçu pour rependre au besoin des débutant de la programmation et pour augmenter la productivité des programmeurs en offrant des outils de haut niveau et d'un syntaxe simple à utiliser .

Notre but dans l'avenir est qu'il soit apprécié par les pédagogies d'apprentissage et on veut qu'il soit l'un des piliers de la programmation dans le monde de développement .

Contents

Chapter 1

Introduction à ShankScript

1.1 Qu'est ce que ShankScript ?

En informatique, un langage de programmation est une notation conventionnelle destinée à formuler des algorithmes et produire des programmes informatiques qui les appliquent. D'une manière similaire à une langue naturelle, un langage de programmation est composé d'un alphabet, d'un vocabulaire, de règles de grammaire et de significations .

Les langages de programmation permettent de décrire d'une part les structures des données qui seront manipulées par l'appareil informatique, et d'autre part d'indiquer comment sont effectuées les manipulations, selon quels algorithmes. Ils servent de moyens de communication par lesquels le programmeur communique avec l'ordinateur, mais aussi avec d'autres programmeurs ; les programmes étant d'ordinaire écrits, lus, compris et modifiés par une équipe de programmeurs .

Un langage de programmation est mis en œuvre par un traducteur automatique : compilateur ou interpréteur. Un compilateur est un programme informatique qui transforme dans un premier temps un code source écrit dans un langage de programmation donné en un code cible qui pourra être directement exécuté par un ordinateur, à savoir un programme en langage machine ou en code intermédiaire, tandis que l'interpréteur réalise cette traduction « à la volée ».

Les langages de programmation offrent différentes possibilités d'abstraction, et une notation proche de l'algèbre, permettant de décrire de manière concise et facile à saisir les opérations de manipulation de données et l'évolution du déroulement du programme en fonction des situations. La possibilité d'écriture abstraite libère l'esprit du programmeur d'un travail superflu, notamment de prise en compte des spécificités du matériel informatique, et lui permet ainsi de se concentrer sur des problèmes plus avancés.

Chaque langage de programmation supporte un ou plusieurs styles de programmation – paradigmes. Les notions propres au paradigme font partie du langage de programmation, permettant au programmeur d'exprimer dans le langage de programmation une solution qui a été imaginée selon ce paradigme.

Les premiers langages de programmation ont été créés dans les années 1950. De nombreux concepts de l'informatique ont été lancés par un langage, avant d'être améliorés et étendus dans les langages suivants. La plupart du temps la conception d'un langage de programmation a été fortement influencée par l'expérience acquise avec les langages précédents.

Dans nôtre cas , ShankSscript est un peu ceci et cela ,avec le manque du temps qu 'on a ,et pour cette première version le langage et paradigmes ,est il est interprété ;vous pouvez valider commande par commande comme dans le cas de python sinon vous avez la possibilité d'écrire dans un fichier Bash avec l'extension ".sks" et par la suite le compiler .

Pour Programmer avec Notre langage , on a conçu un IDE spéciale qui gère les deux mode d'utilisation ; mode interpréteur ,et mode Bash .

Remarque : Cette plateforme et utilisé à la fois dans les systèmes Unix et Windows .

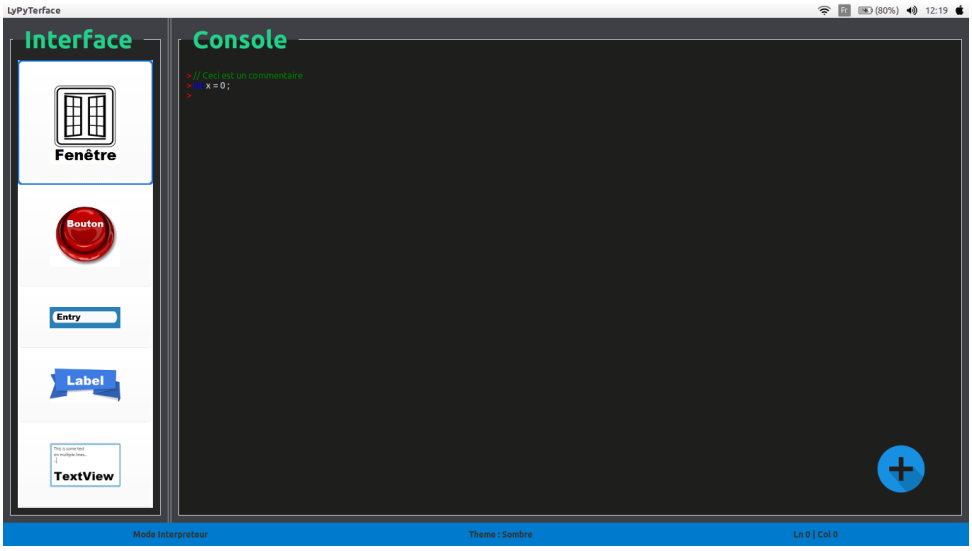

Figure 1.1: IDE de ShankSscript

1.2 Premiers pas avec l'interpréteur de commandes ShankScript

Après les premières notions théoriques et l'installation de ShankScript, il est temps de découvrir un peu l'interpréteur de commandes de ce langage. Même si ces petits tests vous semblent anodins, vous découvrirez dans ce chapitre les premiers rudiments de la syntaxe du langage et je vous conseille fortement de me suivre pas à pas, surtout si vous êtes face à votre premier langage de programmation.

Comme tout langage de programmation, ShankScript a une syntaxe claire : on ne peut pas lui envoyer n'importe quelle information dans n'importe quel ordre. Nous allons voir ici ce que ShankScript mange... et ce qu'il ne mange pas.

1.2.1 L'interpréteur

Eh bien, cet interpréteur de commandes va nous permettre de tester directement du code. Je saisis une ligne d'instructions, j'appuie sur la touche Entrée de mon clavier, je regarde ce que me répond ShankScript (s'il me dit quelque chose), puis j'en saisis une deuxième, une troisième... Cet interpréteur est particulièrement utile pour comprendre les bases de ShankScript et réaliser nos premiers petits programmes. Le principal inconvénient, c'est que le code que vous saisissez n'est pas sauvegardé (sauf si vous l'enregistrez manuellement, mais chaque chose en son temps).

Dans la fenêtre que vous avez sous les yeux, l'information qui ne change pas d'un système d'exploitation à l'autre est la série de trois chevrons qui se trouve en bas à gauche des informations : >. Ces trois signes signifient : « je suis prêt à recevoir tes instructions ».

Comme je l'ai dit, les langages de programmation respectent une syntaxe claire. Vous ne pouvez pas espérer que l'ordinateur comprenne si, dans cette fenêtre, vous commencez par lui demander : « j'aimerais que tu me codes un jeu vidéo génial ». Et autant que vous le sachiez tout de suite (bien qu'à mon avis, vous vous en doutiez), on est très loin d'obtenir des résultats aussi spectaculaires à notre niveau.

Tout cela pour dire que, si vous saisissez n'importe quoi dans cette fenêtre, la probabilité est grande que ShankScript vous indique, clairement et fermement, qu'il n'a rien compris.

Si, par exemple, vous saisissez « Salut ShankScript », vous allez avoir un message d'erreur qui dit que vous avez une erreur de syntaxe .

l'interpréteur vous indique qu'il a trouvé un problème dans votre ligne d'instruction. Il vous indique le numéro de la ligne (en l'occurence la première), qu'il vous répète obligeamment (ceci est très utile quand on travaille sur un programme de plusieurs centaines de lignes).

1.2.2 Première commande

On va commencer par la saisie des nombre et opération arithmétique ,Nous allons donc essayer d'obtenir les résultats de calculs plus ou moins compliqués:

1.2.2.1 Saisir un nombre

Vous avez pu voir sur notre premier (et à ce jour notre dernier) test que ShankScript n'aimait pas particulièrement les suites de lettres qu'il ne comprend pas. Par contre, l'interpréteur adore les nombres. D'ailleurs, il les accepte sans sourciller, sans une seule erreur .
Si vous entrer un nombre dans l'interpréteur ,alors sans problème il va l'afficher . ce simple retour indique que l'interpréteur a bien compris et que votre saisie est en accord avec sa syntaxe. De même, vous pouvez saisir des nombres à virgule.

Remarque : on utilise ici la notation anglo-saxonne, c'est-à-dire que le point remplace la virgule. La virgule a un tout autre sens pour ShankScript, prenez donc cette habitude dès maintenant.

Il va de soi que l'on peut tout aussi bien saisir des nombres négatifs (vous pouvez d'ailleurs faire l'essai).

1.2.2.2 Opérations courantes

Les opérations disponibles en ce moment sont :Addition, soustraction, multiplication, division .

Pour effectuer ces opérations, on utilise respectivement les symboles +, -, * et /.

On a aussi ajouté l'opération Modulo qui permet de connaître le reste de la division.

1.3 Les variables

1.3.1 C'est quoi une variable ?

Une variable est une donnée de votre programme, stockée dans votre ordinateur. C'est un code alpha-numérique que vous allez lier à une donnée de votre programme, afin de pouvoir l'utiliser à plusieurs reprises et faire des calculs un peu plus intéressants avec. C'est bien joli de savoir faire des opérations mais, si on ne peut pas stocker le résultat quelque part, cela devient très vite ennuyeux.

Voyez la mémoire de votre ordinateur comme une grosse armoire avec plein de tiroirs. Chaque tiroir peut contenir une donnée ; certaines de ces données seront des variables de votre programme.

1.3.2 Comment cela fonctionne-t-il ?

Le plus simplement du monde. Vous allez dire à Python : « je veux que, dans une variable que je nomme age, tu stockes mon âge, pour que je puisse le retenir (si j'ai la mémoire très courte), l'augmenter (à mon anniversaire) et l'afficher si besoin est ».

Comme on vous l'a dit, on ne peut pas passer à côté des variables. Vous ne voyez peut-être pas encore tout l'intérêt de stocker des informations de votre programme et pourtant, si vous ne stockez rien, vous ne pouvez pratiquement rien faire.

En ShankScript, pour donner une valeur à une variable, il suffit d'écrire nom de la variable egal valeur.

Une variable doit respecter quelques règles de syntaxe incontournables :

1. Le nom de la variable ne peut être composé que de lettres, majuscules ou minuscules, de chiffres et du symbole souligné (appelé underscore en anglais).
2. Le nom de la variable ne peut pas commencer par un chiffre.

3. Le langage Python est sensible à la casse, ce qui signifie que des lettres majuscules et minuscules ne constituent pas la même variable (la variable AGE est différente de aGe, elle-même différente de age).

Au-delà de ces règles de syntaxe incontournables, il existe des conventions définies par les programmeurs eux-mêmes. L'une d'elles, que j'ai tendance à utiliser assez souvent, consiste à écrire la variable en minuscules et à remplacer les espaces éventuels par un espace souligné.

Et d'après de ce qu 'on a vu avant ,vous pouvez les même opérations arithmétiques sur les variables .

Remarque : Certains mots-clés de ShanskScript sont réservés, c'est-à-dire que vous ne pouvez pas créer des variables portant ce nom. Comme ("var","str","return","class","function" ...).

1.3.3 Les types de données en ShankScript

Là se trouve un concept très important, que l'on retrouve dans beaucoup de langages de programmation.

Qu'entend-on par « type de donnée » ?

Jusqu'ici, vous n'avez travaillé qu'avec des nombres. Et, s'il faut bien avouer qu'on ne fera que très rarement un programme sans aucun nombre, c'est loin d'être la seule donnée que l'on peut utiliser en ShankScript. À terme, vous serez même capables de créer vos propres types de données, mais n'anticipons pas.

ShankScript a besoin de connaître quels types de données sont utilisés pour savoir quelles opérations il peut effectuer avec. Dans ce chapitre, vous allez apprendre à travailler avec des chaînes de caractères, et multiplier une chaîne de caractères ne se fait pas du tout comme la multiplication d'un nombre. Pour certains types de données, la multiplication n'a d'ailleurs aucun sens. Python associe donc à chaque donnée un type, qui va définir les opérations autorisées sur cette donnée en particulier.

Les différents types de données

Type entier et flottante

Si on voit les langage de tels que C ,et va remarquer que chaque type est mis à part ,cela et bien mais notre but est de rendre la programmation facile , on veut que l'utilisateur ne se soucie de rien qu'on il va commencer à programmer .

Alors on a rassemblé les deux types en un seul type qui est « var » , il peut contenir soit un flottant soit un entier par rapport à la situation .

Les chaînes de caractère

Heureusement, les types de données disponibles en ShnkScript ne sont pas limités aux seuls nombres, bien loin de là. Le dernier type « simple » que nous verrons dans ce chapitre est la chaîne de caractères. Ce type de donnée permet de stocker une série de lettres, pourquoi pas une phrase.

On peut écrire une chaîne de caractères de cette façon :
entre guillemets ("ceci est une chaîne de caractères") .

1.4 Les structures conditionnelles

Jusqu'à présent, nous avons testé des instructions d'une façon linéaire : l'interpréteur exécutait au fur et à mesure le code que vous saisissiez dans la console. Mais nos programmes seraient bien pauvres si nous ne pouvions, de temps à autre, demander à exécuter certaines instructions dans un cas, et d'autres instructions dans un autre cas.

Dans ce chapitre, on va vous parler des structures conditionnelles, qui vont vous permettre de faire des tests et d'aller plus loin dans la programmation.

Les conditions permettent d'exécuter une ou plusieurs instructions dans un cas, d'autres instructions dans un autre cas.

1.4.1 La condition simple IF

En anglais, le mot « si » se traduit parif. C'est celui qu'on utilise en langage C pour introduire une condition.

Écrivez donc unif. Ouvrez ensuite des parenthèses : à l'intérieur de ces parenthèses vous devrez écrire votre condition.

Ensuite, ouvrez une accoladeet fermez-la un peu plus loin. Tout ce qui se trouve à l'intérieur des accolades sera exécuté uniquement si la condition est vérifiée.

1.4.2 Le ElSE pour dire sinon

Maintenant que nous savons faire un test simple, allons un peu plus loin : si le test n'a pas marché (il est faux), on va dire à l'ordinateur d'exécuter d'autres instructions.

Il suffit de rajouter le motelseaprès l'accolade fermante du if.

ALors si on applique la logique si l'action n'est pas inclus dans la condition <if> on swap directement à ELSE .

1.4.3 LE ELSE IF pour dire sinon si

On a vu comment faire un « si » et un « sinon ». Il est possible aussi de faire un « sinon si » pour faire un autre test si le premier test n'a pas marché. Le « sinon si » se place entre leifet leelse.

L'ordinateur fait les tests dans l'ordre.

1. D'abord il teste le premierif: si la condition est vraie, alors il exécute ce qui se trouve entre les premières accolades.
2. Sinon, il va au « sinon si » et fait à nouveau un test : si ce test est vrai, alors il exécute les instructions correspondantes entre accolades.
3. Enfin, si aucun des tests précédents n'a marché, il exécute les instructions du « sinon ».

Remarque : Leelseet leelse ifne sont pas obligatoires. Pour faire une condition, seul unifest nécessaire (logique me direz-vous, sinon il n'y a pas de condition !).

1.4.4 La condition Switch

La conditionif... elseque l'on vient de voir est le type de condition le plus souvent utilisé. En fait, il n'y a pas 36 façons de faire une condition en C. Leif... elsepermet de gérer tous les cas.

Les informaticiens détestent faire des choses répétitives, on a eu l'occasion de le vérifier plus tôt.

Alors, pour éviter d'avoir à faire des répétitions comme ça quand on teste la valeur d'une seule et même variable, ils ont inventé une autre structure que leif... else. Cette structure particulière s'appelle switch.

L'idée c'est donc d'écrireswitch (maVariable)pour dire « je vais tester la valeur de la variablemaVariable». Vous ouvrez ensuite des accolades que vous refermez tout en bas.

Ensuite, à l'intérieur de ces accolades, vous gérez tous les cas :case 2,case 4,case 5,case 45...

Remarque : Vous devez mettre une instructionbreak;obligatoirement à la fin de chaque cas. Si vous ne le faites pas, alors l'ordinateur ira lire les instructions en dessous censées être réservées aux autres cas !

L'instructionbreak;commande en fait à l'ordinateur de « sortir » des accolades.

Enfin, le casdefaultcorrespond en fait auelsequ'on connaît bien maintenant. Si la variable ne vaut aucune des valeurs précédentes, l'ordinateur ira lire ledefault.

1.5 Les boucles

1.5.1 Qu'est-ce qu'une boucle ?

une boucle est une structure qui permet de répéter les mêmes instructions plusieurs fois.

Tout comme pour les conditions, il y a plusieurs façons de réaliser des boucles. Au bout du compte, cela revient à faire la même chose : répéter les mêmes instructions un certain nombre de fois.

Voici ce qu'il se passe dans l'ordre :

1. l'ordinateur lit les instructions de haut en bas (comme d'habitude) ;
2. puis, une fois arrivé à la fin de la boucle, il repart à la première instruction ;
3. il recommence alors à lire les instructions de haut en bas...
4. ... et il repart au début de la boucle.

Le problème dans ce système c'est que si on ne l'arrête pas, l'ordinateur est capable de répéter les instructions à l'infini ! Il n'est pas du genre à se plaindre, vous savez : il fait ce qu'on lui dit de faire... Il pourrait très bien se bloquer dans une boucle infinie, c'est d'ailleurs une des nombreuses craintes des programmeurs.

Et c'est là qu'on retrouve... les conditions ! Quand on crée une boucle, on indique toujours une condition. Cette condition signifiera « Répète la boucle tant que cette condition est vraie ». [0.5cm] Comme je vous l'ai dit, il y a plusieurs manières de s'y prendre. Voyons voir sans plus tarder comment on réalise une boucle de type while .

1.5.2 La boucle while

C'est aussi simple que cela. while signifie « Tant que ». On dit donc à l'ordinateur « Tant que la condition est vraie, répète les instructions entre accolades ».

L'ordinateur lit la condition ,après il teste si la condition est vrai ,si la condition est vrai il execute le programme et refais le teste,sinon il sort .

Lorsque vous créez une boucle, assurez-vous toujours qu'elle peut s'arrêter à un moment ! Si la condition est toujours vraie, votre programme ne s'arrêtera jamais !

1.5.3 La boucle do... while

Ce type de boucle est très similaire à while, bien qu'un peu moins utilisé en général. La seule chose qui change en fait par rapport à while, c'est la position de la condition. Au lieu d'être au début de la boucle, la condition est à la fin .

Qu'est-ce que ça change ?

Pour la boucle do... while, c'est différent : cette boucle s'exécutera toujours au moins une fois. En

Il est donc parfois utile de faire des boucles de ce type, pour s'assurer que l'on rentre au moins une fois dans la boucle.

Remarque : Il y a une particularité dans la boucle do... while qu'on a tendance à oublier quand on débute : il y a un point-virgule tout à la fin ! N'oubliez pas d'en mettre un après le while, sinon votre programme plantera à la compilation !

1.5.4 La boucle for

En théorie, la boucle while permet de réaliser toutes les boucles que l'on veut.
Toutefois, tout comme le switch pour les conditions, il est dans certains cas utile d'avoir un autre système de boucle plus « condensé », plus rapide à écrire.

Les boucles for sont très très utilisées en programmation. Je n'ai pas de statistiques sous la main, mais sachez que vous utiliserez certainement autant de for que de while, si ce n'est plus, il vous faudra donc savoir manipuler ces deux types de boucles.

Comme je vous le disais, les boucles for sont juste une autre façon de faire une boucle while.

1.6 Les entrées et les sorties

(En cours ...)

1.7 Les fonctions

(En cours ...)

1.8 Les packages

(En cours ...)

1.9 Interfaçage et les packages graphiques

(En cours ...)

Chapter 2

L'analyse de l'interpréteur

En informatique, un interprète, ou interpréteur , est un outil ayant pour tâche d'analyser, de traduire et d'exécuter les programmes écrits dans un langage informatique. On qualifie parfois, et abusivement, les langages dont les programmes sont généralement exécutés par un interpréteur de langages interprétés.

Un interpréteur se distingue d'un compilateur par le fait que, pour exécuter un programme, les opérations d'analyse et de traductions sont réalisées à chaque exécution du programme (par un interpréteur) plutôt qu'une fois pour toutes (par un compilateur).

2.1 Introduction au principe

L'interprétation repose sur l'exécution dynamique du programme par un autre programme (l'interprète), plutôt que sur sa conversion en un autre langage (par exemple le langage machine) ; elle évite la séparation du temps de conversion et du temps d'exécution, qui sont simultanés.

On différencie un programme dit script, d'un programme dit compilé :

- Un programme script est exécuté à partir du fichier source via un interpréteur de script ;
- Un programme compilé est exécuté à partir d'un bloc en langage machine issu de la traduction du fichier source.

Le cycle d'un interprète est le suivant :

- lire et analyser une instruction (ou expression) ;
- si l'instruction est syntaxiquement correcte, l'exécuter (ou évaluer l'expression) ;
- passer à l'instruction suivante.

Ainsi, contrairement au compilateur, l'interprète exécute les instructions du programme (ou en évalue les expressions), au fur et à mesure de leur lecture pour interprétation. Du fait de cette phase sans traduction préalable, l'exécution d'un programme interprété est généralement plus lente que le même programme compilé. La plupart des interprètes n'exécutent plus la chaîne de caractères représentant le programme, mais une forme interne, telle qu'un arbre syntaxique.

En pratique, il existe une continuité entre interprètes et compilateurs. La plupart des interprètes utilisent des représentations internes intermédiaires (arbres syntaxiques abstraits, ou même code octet) et des traitements (analyses lexicale et syntaxique) ressemblant à ceux des compilateurs. Enfin, certaines implémentations de certains langages (par exemple SBCL pour Common Lisp) sont interactifs comme un interprète, mais traduisent dès que possible le texte d'un bout de programme en du code machine directement exécutable par le processeur. Le caractère interprétatif ou compilatoire est donc propre à l'implémentation d'un langage de programmation, et pas au langage lui-même.

L'intérêt des langages interprétés réside principalement dans la facilité de programmation et dans la portabilité. Les langages interprétés facilitent énormément la mise au point des programmes car ils évitent la phase de

compilation, souvent longue, et limitent les possibilités de bogues. Il est en général possible d'exécuter des programmes incomplets, ce qui facilite le développement rapide d'applications ou de prototypes d'applications. Ainsi, le langage BASIC fut le premier langage interprété à permettre au grand public d'accéder à la programmation, tandis que le premier langage de programmation moderne interprété est Lisp.

La portabilité permet d'écrire un programme unique, pouvant être exécuté sur diverses plates-formes sans changements, pourvu qu'il existe un interprète spécifique à chacune de ces plates-formes matérielles.

Un certain nombre de langages informatiques sont aujourd'hui mis en œuvre au moyen d'une machine virtuelle applicative. Cette technique est à mi-chemin entre les interprètes tels que décrits ici et les compilateurs. Elle offre la portabilité des interprètes avec une bonne efficacité. Par exemple, des portages de Java, Lisp, Scheme, Ocaml, Perl (Parrot), Python, Ruby, Lua, C, etc. sont faits via une machine virtuelle.

L'interprétation abstraite (inventée par Patrick et Radhia Cousot) est une technique et un modèle d'analyse statique de programmes qui parcourt, un peu à la manière d'un interprète, le programme analysé en y remplaçant les valeurs par des abstractions. Par exemple, les valeurs des variables entières sont abstraites par des intervalles d'entiers, ou des relations algébriques entre variables.

Revenant à notre cas ,on ne veut pas sortir du cadre générale , on a procédé par la structurisation usiale .

On a commencé par Construire un analyseur Lexical ,après on a fait Analyseur syntaxique et puis Sémantique ,pour que par la suite la traduction se fait de façon simple et facile .

2.2 Analyse Lexical

2.3 Analyse syntaxique

(En cours ...)

2.4 Analyse sémantique

(En cours ...)

2.5 Traduction

(En cours ...)

Chapter 3

Conclusion

Pour conclure ,le développement du langage est encore en cours , on pourra ajouter plein de chose ,et ces aussi le cas pour la suppression , on va essayer de corriger toutes les erreurs de cette version pour que la version finale soit meilleur .