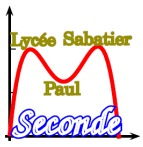


Activités pratiques proposées par vos professeurs

Table des matières

Algorithme - Python - 1

Résumé

Premier cours interactif à compléter pour :
- Faire la distinction entre un algorithme et un programme.
- Commencer à écrire des algorithme en langage naturel.

1 Pour bien commencer . . .

1.1 ATTENTION . . .

- Ceci est le premier document d'une série intitulée ***Algorithme-Python*** de niveau seconde qui a pour objectif de vous permettre de créer vos propres programmes en langage Python utilisables par exemple avec vos calculatrices et/ou ordinateurs.
- Ce cours se veut interactif et très progressif. A vous de participer au cours de chez vous à votre rythme en respectant mes consignes.
- Il est très important que vous testiez les exercices même à plusieurs reprises et surtout si cela parait évident.
- Chaque exercice interactif sera corrigé individuellement.

1.2 Mode d'emploi . . .

— Le cours version PDF contient des liens :
  - qui ouvriront des vidéos explicatives souvent pour résumer une notion. malheureusement, il risque d'y avoir des publicités (**On peut les passer au bout de quelques secondes**).
  - qui ouvriront une application par l'intermédiaire de votre ENT qui vous permettra de répondre et de tester des programmes. Un bouton *rendre ce travail* me permettra de voir vos réponses et de les commenter.

— Le cours version papier vous permettra de noter par écrit des résultats (ou autre) et de les conserver plus facilement.

2 Algorithme

Commencer par regarder cette vidéo . . . en anglais mais cliquer sur sous-titre puis sur paramètre, sous titre pour traduire automatiquement en français.
et vous verrez deux enfants (enfin surtout la fille à la fin de vidéo) réussir à écrire un algorithme.

Lien vidéo

2.1 A vous de jouer . . .

Vous pouvez accéder a vos premiers exercices interactifs :

- Après avoir cliqué sur le ***lien de l'exercice***, votre ordinateur se dirigera vers Capytale qui vous demandera comment vous connecter.
  Utiliser mon ent occitanie
- Puis après vous être connecté à l'ENT avec vos codes, autoriser la transmission de votre nom et prénom.
- Cliquer sur ***GO*** sans modifier le code.
- Vous pouvez enregistrer grâce au premier bouton (***disquette*** en haut à gauche) votre travail pour y revenir plus tard.
- En fin de travail, il faut cliquer sur ***rendre ce travail*** (clic sur bouton en haut à droite) pour que je puisse voir votre travail.
- Pour voir mes remarques, il vous faudra recliquer sur le lien.
- En cliquant sur un morceau de texte (ou plus tard un programme), un cadre apparaît entourant la zone de texte.
- Pour modifier du texte ou répondre du texte, il vous faut cliquer deux fois sur le texte puis après modification cliquer sur le bouton ***Run***
- Pour la première activité, vous pouvez inverser des zones de textes en utilisant les deux boutons flèches (en haut un peu à droite).
- Me demander en cas de soucis ou/et suivre lors des séances explicatives en classe.
- Allez c'est parti : EXO 1
- Et puis : EXO 2

2.2 A apprendre . . .

Algorithme

Un algorithme est une suite d'ordres simples et classés pour obtenir un résultat.

2.3 Utilité et exemples ...

- Une recette de cuisine est un algorithme adapté (plus ou moins) à un cuisinier débutant pour lui permettre de réussir un plat complexe.
- Une notice de montage d'un meuble "IK.." est un algorithme permettant à une personne de monter son meuble (normalement).
- Dans une chaîne de montage d'une voiture, les mouvements du robot fixant la portière avant droite sont dirigés par un algorithme.

Remarque : En mathématiques, nous écrirons des algorithmes pour un ordinateur afin de réaliser des calculs longs et/ou répétitifs. L'ordinateur étant particulièrement idiot, les algorithmes devront être précis et détaillés.

3 Codage - Programme

3.1 Mise en évidence ...

Dans ma cuisine, l'algorithme ***recette de crêpes*** m'a permis de bien manger mais ma copine anglaise, en suivant le même algorithme n'a pas réussi a faire grand chose ...
Pourquoi ?
Parce que l'algorithme était écrit en français et que je comprends le français contrairement à ma copine anglaise.

3.2 Découverte d'un code particulier ...

Le codage consiste donc de traduire des instructions dans un langage souvent peu compréhensible avec une précision extrême sous peine de blocage de la machine.
Je vous propose de découvrir un premier langage de codage (langage de programmation) le ***schtroumpfglop***. Ce langage est très peu connu mais il permet de vérifier si vous savez suivre des règles précises.
Les règles du codage sont simples : tous les verbes sont remplacés par ***schtroumpf***, tous les mots de deux lettres sont remplacés par ***glop*** et chaque ligne se termine par ***pas glop***. Pour compliquer un peu, ***schtroumpf*** s'écrit avec une (seule) majuscule mais la position de la majuscule indique la ligne de programmation.

Qui relèvera le défi et codera sans erreur en ***schtroumpfglop*** ? EXO 3

3.3 Définition

Coder ou écrire du code est le fait d'écrire un algorithme dans un langage particulier compréhensible par l'utilisateur. Si l'utilisateur est une machine (donc extrêmement bête), l'algorithme devra être très détaillé et le codage extrêmement rigoureux.
Le résultat est un programme.
Il existe une multitude de codages (langages de programmation) différents.

Programme
Un programme est un algorithme écrit dans un langage particulier exécutable par une machine.

3.4 Résumé

Lien vidéo

- Une bonne préparation de l'algorithme facilitera le codage.
- La connaissance d'un langage de programmation facilite énormément l'apprentissage des autres langages.
- Au lycée, nous vous proposons d'apprendre les bases de l'algorithmique et les bases d'un langage de programmation (Python).

4 Algorithme au lycée - Pseudo-code ou langage naturel

4.1 Objectifs ...

Notre objectif est de créer des programmes pour ordinateur ou calculatrice (bref une machine) et non pas d'écrire un livre de recettes de cuisine pour les hommes allemands célibataires de 18 à 25 ans.
Si l'on doit se faire comprendre d'une machine, il faut d'abord connaître ses caractéristiques principales.

- un ordinateur est bête mais franchement bête (le contraire d'un humain).
- un ordinateur apprend très vite (le contraire d'un humain).
- un ordinateur travaille très très vite (le contraire d'un humain).
- un ordinateur ne se fatigue pas (le contraire d'un humain).
- un ordinateur ne réfléchit pas (le contraire d'un humain).

Ne jamais oublier qu'un programme est exécuté ***par*** une machine ***pour*** un humain. Il y a deux rôles, l'ordinateur qui fait ce qu'on lui dit de faire et un humain qui utilise l'ordinateur.

4.2 Vous verrez ...

Schématiquement, il existe deux grandes familles d'erreurs en programmation :

- les erreurs d'algorithmie :
  - — des consignes qui n'aboutissent pas.
  - — des consignes trop complexes.
  - — des consignes qui ne donnent pas le bon résultat.
  - — des erreurs de variables
- les erreurs de codage :
  - — des erreurs d'écriture du code.

ANALOGIE en français :

- Les fautes de grammaire.
- Les fautes d'orthographe.

Croyez moi . . .

Si vous tentez d'écrire un programme en codant directement, toutes les erreurs vont se mélanger et il sera très vite impossible de s'y retrouver.

4.3 notion de variable

Un ordinateur sait calculer 2+3, mais si on ne lui dit pas de s'en souvenir, il va oublier le résultat . . .
Il faut lui dire de stocker toutes les informations dans des variables.
Chaque variable possède un nom et un contenu. (voir vidéo ci contre)
Remarque : Dans les écoles d'ingénieurs en informatique, il n'est pas rare que les étudiants suivent pendant plus de 6 mois plusieurs heures par semaine un cours sur juste les variables. Nous nous limiterons évidemment au lycée qu'aux bases simples.

Lien vidéo

4.4 Le langage naturel ou pseudo code, qu'est ce que c'est ?

Ecrire un algorithme en langage naturel consiste à écrire sur papier l'algorithme en ne respectant que deux ou trois règles de codage très simples. Cela permet d'éliminer les erreurs d'algorithmie et de faciliter le futur codage (je vous rassure il restera suffisamment d'erreurs pour passer de nombreuses heures à chercher pourquoi le programme ne marche pas :)).

- Les algorithmes en langage naturel sont écrits en langue française.
- On écrit les instructions les unes sous les autres sous forme de liste en utilisant les tirets pour bien marquer la chronologie.
- Chaque instruction commence par un verbe d'action.

Exemple d'un algorithme en langage naturel

L'exemple qui suit est en langage naturel mais très vite nous le simplifierons un peu plus pour être plus efficace. Cet exemple est donné pour comprendre les trois règles ci-dessus.

Algorithme

- DEMANDER ton nom
- MEMORISER le nom dans la variable N
- DEMANDER ta dernière note en math.
- MEMORISER la note dans la variable X
- CALCULER X+2
- MEMORISER le résultat du calcul dans la variable Y
- ECRIRE le texte "Bonjour N"
- ECRIRE le texte "je compte sur toi pour avoir la note Y au prochain devoir."

A vous de jouer n°1 :

Ce jeu se joue à deux. Le joueur A joue le rôle de l'ordinateur. il va suivre l'algorithme (les consignes) ci-contre. Le joueur B joue le rôle de l'utilisateur de l'ordinateur, il répondra à l'ordinateur (joueur A). Puis inverser les rôles.
Remplir le tableau ci dessous :
Si tu es le joueur A : action de l'ordinateur (avec ou sans affichage)
Si tu es le joueur B : action de l'utilisateur.

Si tu es le joueur A		Si tu es le joueur B
action que l'ordi affichera à l'écran	action que l'ordi n'afficheras à l'écran	action de l'utilisateur

A vous de jouer n°2 : Ecrire l'algorithme permettant d'additionner deux nombres quelconques. EXO 4
Conseil (rappel) : L'algorithme sera exécuté par une machine donc pour écrire un algorithme, il faut vous imaginer à la place de la machine. Vous êtes l'ordinateur, que devez vous faire ?
A vous de jouer n°3 : Ecrire l'algorithme permettant de diviser deux nombres quelconques (exercice plus difficile). EXO 5

Quelle est la différence entre l'exo 4 et l'exo 5 ?

Algorithme - Python - 2

Résumé

Découverte des premières commandes d'un langage de programmation.
- Premiers programmes exécutables par une machine.
- Les variables en Python.

1 Python : Un premier langage de programmation

1.1 Pourquoi Python

- C'est un langage de programmation relativement simple à apprendre.
- C'est un langage de programmation puissant et très utilisé dans le milieu scientifique.
- Skratch est un langage de programmation peu puissant réservé à une simple initiation de la programmation (même si on peut faire de belles choses avec skratch).

L'apprentissage d'un langage de programmation permet d'acquérir une certaine logique de programmation que l'on retrouve dans beaucoup d'autres langages. Donc l'apprentissage d'autres langages de programmation sera grandement facilité.

1.2 Python en Math

Il existe quantité de logiciels permettant d'exécuter un programme en Python. Certains programmes le faisant sont préinstallés sur les ordis région (et vous pouvez en installer d'autres).
En Math, je n'utiliserai (dans un premier temps) que Capytale par l'intermédiaire des liens de ce cours, la calculatrice Numworks et le site internet Numworks.
Schématiquement :

- Capytale pour apprendre.
- site Numworks pour faire ou modifier des programmes quand vous êtes chez vous.
- calculatrice numworks pour faire, modifier et surtout UTILISER des programmes dans le cours de mathématiques. On pourra bien sur passer des programmes des uns aux autres et même aux différents logiciels que vous utiliserez en SNT ou physique.

Pour réussir en algorithmique et avec Python

Un seul mot d'ordre : Tester, Tester, Tester ...
Ce n'est pas en regardant le tour de France que vous avez appris à faire du vélo ....

2 Premiers programmes en Python

2.1 La commande python print()

Cette commande permet à l'ordinateur d'écrire quelque chose à l'écran.
Découvrons les variables et la commande print à l'aide d'exercices.

Les liens suivants permettent d'accéder aux exercices. Comme d'habitude, vous verrez des cases "textes" mais aussi des cases avec des programmes python exécutables en cliquant sur le bouton ***Run***. Le langage Python utilise des couleurs spécifiques afin d'aider le programmeur en différenciant les commandes, les variables, etc. Quand le programme n'a pas de bug, le résultat de l'exécution s'écrit en dessous.

Affichons quelques petits calculs ...

Commençons par faire faire à l'ordinateur quelques calculs simples. EXO 1

Et si on rajoutait du texte ...

Améliorons la présentation avec un peu de texte. EXO 2

3 Introduction aux variables en Python

Ce chapitre est (à mon avis) le plus important pour la programmation. D'innombrables erreurs de programmation sont des erreurs de variables.

3.1 Quelques règles d'utilisation des variables

Une variable est une zone de stockage qui possède un nom et une valeur (qui peut changer).

Concernant le nom de la variable

- Un programme ne peut pas avoir deux variables ayant un nom identique.
- Le nom de la variable ne doit pas être un nombre.
- Le nom de la variable ne doit pas commencer par un nombre.
- Le nom de la variable ne doit pas contenir d'espace ni de signe de ponctuation.

A vous de jouer : Parmi les 10 noms suivants, entoure les noms pouvant être utilisés comme nom de variable :
N nom du joueur 4 nom_du_joueur N4 4N A340 Denis d.g alpha38

Règle d'affectation

En math : $x = 3$ ou $3 = x$ sont deux égalités identiques.
En programmation : TOUJOURS le nom de la variable à gauche et la valeur de la variable à droite.

$x = 3$ signifie que x est le nom de la variable et que 3 est la valeur de la variable.

$3 = x$ ne fonctionnera pas car 3 n'est pas un nom de variable valide.

$x = 3 + 2$ signifie qu'on met la valeur $3 + 2$ donc 5 dans la variable de nom x.

En langage naturel : on écrira $x \leftarrow 3$ qui signifie on met 3 dans la variable x.

3.2 A vous de jouer ...

Petits jeux avec les variables. EXO 3

3.3 tableau de variables

Voici l'algorithme et le programme du dernier exercice proposé.

Algorithme

- $g \leftarrow 4$
- nom $\leftarrow$ "Denis"
- $g \leftarrow g + g$
- $g \leftarrow 2g$
- $g \leftarrow g + 5$
- Afficher "bonjour",nom,"as tu trouvé", g"

programme Python

```
1 g=4
2 nom="Denis"   # mettre ton prenom entre les guillemets
3 g=g+g
4 g=2*g
5 g=g+5
6 print("bonjour ",nom,"as tu trouve",g,"comme resultat ?")
```

Un tableau des variables est comme le nom l'indique un tableau où l'on étudie l'évolution de toutes les variables. C'est une des méthodes qui permet de comprendre le fonctionnement d'un programme.
Pour l'exemple ci-dessus :

Explication ligne par ligne

Ligne 1 : La variable g prend la valeur 4
Ligne 2 : La variable nom prend la valeur Denis, g reste à 4
Ligne 3 : On calcule $g + g$ donc $4 + 4$ et on enregistre 8 dans g
Ligne 4 : On calcule $2g$ donc 2×8 et on enregistre 16 dans g
ligne 5 : On calcule $g + 5$ donc $16 + 5$ et on enregistre 21 dans g
Ligne 6 : On affiche du texte et la valeur de g soit 21

Tableau des variables

g	**nom**
4	
4	Denis
8	Denis
16	Denis
21	Denis

3.4 A vous de jouer ...

Utiliser le tableau des variables ci-contre : EXO 4

Tableau des variables

a	*b*	*c*

4 Types de variables en python

4.1 introduction

Ce qui précède nous a déjà montré qu'une variable pouvait contenir soit des nombres, soit du texte.
Quelques exercices pour approfondir les différents types de variable. EXO 5

4.2 Quelques types de variable en Python

On retrouve ces types de variable dans pratiquement tous les langages de programmation.

- Les entiers que Python appelle **INT**
- Les réels (arrondis) que Python appelle **FLOAT** (flottant)
- Les chaînes de caractères que Python appelle **STR** (string en anglais)

Plus tard dans l'année, nous verrons aussi :

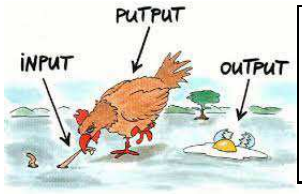
- Les listes de nombres
- Les listes de chaînes de caractères.
- Les booléens (très utilisés en programmation). Les booléens ne peuvent prendre que deux valeurs, **VRAI** ou **FAUX**

4.3 Résumé

Lien vidéo 1

Lien vidéo 2

Algorithme - Python - 3



Résumé

A la fin de cette leçon, création d'un premier jeu :
- Python : la commande Input.
- Retour sur les différences entre algorithme et programme.

1 Python : La commande input()

1.1 Introduction

algorithme

$a \leftarrow 3$
$b \leftarrow 2$
$c \leftarrow a + b$
Afficher"$3 + 2 =$",**c**

L'algorithme (ci-contre à gauche) et le programme python correspondant (ci-contre à droite) calcule très bien et très vite la somme $3 + 2$ mais ce programme ne calcule que $3 + 2$.
Si pour calculer $3 + 4$, on doit refaire un autre programme, la programmation n'aurait plus beaucoup d'intérêt.

programme Python

```
1 a=3
2 b=2
3 c=a+b
4 print("3+2=",c)
```

Quand a faire un programme (exemple : un programme qui fait l'addition de deux nombres) ; autant que ce programme puisse être utilisé dans tous les cas.
Dans notre exemple, le programme devrait additionner deux nombres quelconques au choix de l'utilisateur et non pas se limiter à $2+3$.

1.2 input()

commande Python

La commande Python **input()** permet à l'ordinateur de demander des informations à l'utilisateur du programme.

Exercices simples pour découvrir la commande input()

En suivant les exemples suivants, découvrez quelques possibilités et quelques limites de la commande **input()**. EXO 1

En conclusion :

- **input()** permet de demander des informations à l'utilisateur mais il faut le mémoriser dans une variable.
- Les informations demandées par **input()** sont de type : chaîne de caractère (impossibilité de faire des calculs avec).

Demander un nombre avec la commande input()

On se rappelle que (en simplifiant un peu) les variables sont :

- de type chaîne de caractère (**str** en Python).
- de type nombre entier (**int** en Python).
- de type nombre décimal (**float** en Python).

input() demande une chaîne de caractère, transformons la (si possible) en nombre. Quelques exemples : EXO 2

Création d'un premier jeu en python.

Comme nous commençons tout juste le langage Python et que nous n'avons pas encore découvert les modules graphiques par exemple, ce premier jeu est très simple.
Simple mais rigolo quand même (si, si !) Tellement simple que la programmation n'apportera rien au jeu mais il montrera bien les différentes étapes à suivre pour programmer (des jeux ou autres).
Avant de le programmer, je vous propose de tester ce jeu avec par exemple des copains ou des membres de votre famille. Je vous garantis un certain succès lors de votre prochaine réunion de famille.

Ce jeu demande 2 joueurs, vous et une victime. La victime va perdre (donc vous allez gagner !).

Début du jeu : Vous demandez à la victime un nombre.
S'il vous répond 5, vous répondez 6, j'ai gagné. Fin du jeu

Vous avez gagné, il a perdu.
Remarque : si la réponse de la victime avait été 7, il faut répondre 8 pour gagner, etc. !!!!!

Vous pouvez jouer 50 fois, vous aller gagner 50 fois. C'est systématique, donc un programme peut le faire. Le but de l'exercice est d'écrire ce programme (qui va toujours gagner ...).

Dans un premier temps, pour programmer ce jeu, il faut réfléchir à tout ce qui se passe y compris dans votre tête lorsque vous jouez. Quand et seulement quand les différentes étapes de tes actions seront classées, on pourra programmer l'ordinateur. Ce premier travail du programmeur n'est pas si simple, il doit se mettre à la place du futur ordinateur et surtout, il doit être aussi bête que l'ordinateur (c'est ça qui est difficile ...).

Clique sur le lien suivant quand tu penses avoir décomposé et classé complètement les différentes actions que l'ordinateur doit faire. EXO 3

ATTENTION : Ne clique sur le lien suivant que QUAND j'aurai corrigé ton travail précédent (et qu'il sera juste ...). EXO 4

1.3 Conclusion

- Il y a bien 2 phases dans le travail de programmation.
  - Une première phase avec la création de l'algorithme.
  - Une deuxième phase de codage en langage de programmation.
  - Et je rajouterai une troisième phase de recherche et correction des bugs :).
- Il existe plusieurs commandes Python pour demander des informations à l'utilisateur du programme.
  - input("quel est ton texte") : pour demander une chaîne de caractère.
  - float(input("quel est ton nombre")) : pour demander un nombre décimal.
  - int(input("quel est ton nombre")) : pour demander un nombre entier.

1.4 Exercice d'application : Le restaurant

Pour vous faire de l'argent de poche, vous trouvez comme travail d'été une place à la plonge dans le restaurant voisin.
Votre travail consiste à laver les couverts (fourchettes, couteaux et cuillères). Par curiosité, vous décidez de compter le nombre de couverts nettoyés.
Au lieu de compter un à un les couverts, vous créez un programme calculant le nombre de couverts en utilisant le nombre de clients.
Le restaurant propose des menus simples qui nécessitent 3 couverts par personne et des menus prestigieux qui en nécessitent 5 par personne.
Compléter l'algorithme puis coder le en Python pour le tester en suivant le lien : EXO 5.

algorithme

Demandernombre de menus simples servis ? S
Demandernombre de menus prestigieux servis ?
P ... ← ...
Afficher" nombre de couverts à laver :"...

2 Quelques spécificités sur le langage Python

2.1 Exemple de calcul mathématique un peu plus compliqué en Python

Je voudrais connaître le volume de glace que peut contenir un cornet. Comme je suis au régime, on fera l'hypothèse que la glace ne déborde pas du cornet (contrairement à l'image ci-contre).
Rappel : le volume d'un cône est calculé par $V = \frac{1}{3}\pi r^2 h$.

Des explications ici :

EXO 6

Algorithme et programme Python :

Algorithme

Demander : "hauteur du cône en cm :" h
Demander : "rayon de la base du cône en cm" : r
Calculer $V \leftarrow \frac{1}{3}\pi r^2 h$
Afficher : "le volume de glace est de", V, "cm cube"

programme Python

```
1 from math import *
2 h=float(input("hauteur cone ="))
3 r=float(input("rayon cone ="))
4 V=1/3*pi*r**2*h    # bizarre ces deux etoiles
5 print("le volume de glace est de",V,"cm cube")
```

2.2 Les modules importés en Python

Python est un langage de programmation puissant et très utilisé mais la version de base ne sait pas faire grand chose. Par exemple, il ne connait pas « π » .
Par contre, il apprend très vite.

Au début du programme, avec la phrase « from math import * » , je lui demande d'apprendre plein d'informations mathématiques. C'est comme s'il venait d'apprendre toute une bibliothèque de livres de math (les racines carrées, les sinus ...).
Bien sur, il n'y a pas que les mathématiciens qui utilisent Python, et donc il existe quantités de bibliothèques spécialisées dans un peu tous les domaines (modules graphiques, module random si on veut simuler des tirages au sort ...).

2.3 Les commentaires en Python

Les programmes Python vont devenir de plus en plus longs et de plus en plus compliqués. Comme le langage Python n'est pas très lisible, on risque vite de ne plus comprendre le programme lors d'une modification.
On peut laisser des commentaires en français dans le programme après le symbole #

Python ignorera les informations écrites derrière le symbole # jusqu'à la fin de la ligne.
Dans l'exemple j'ai voulu attirer votre attention sur le codage Python pour mettre au carré.

Algorithme - Python - 4

Résumé

Le simple mot SI va nous permettre de résoudre des problèmes bien plus complexes.
- Les instructions conditionnelles (algorithme et Python).
- Les tests simples (en mathématiques).

1 Python : Les programmes conditionnels

1.1 Introduction

Voir la vidéo d'introduction.
Pour l'instant nos algorithmes (et programmes) ne comportaient qu'une réponse unique à un problème. Par exemple, l'algorithme qui additionne deux nombres.
Comment écrire un algorithme (et programme) lorsque plusieurs réponses sont possibles en fonction de différents cas.

Lien vidéo

1.2 Exemples d'algorithmes conditionnels

Face à un problème, on peut écrire plusieurs algorithmes (donc programmes) différents pour le résoudre. Tous les algorithmes (ou programmes) résolvant le problème sont donc valables. Il n'y a pas qu'une seule solution. Remarque : Plus tard (étude supérieure) certains algorithmes (ou programmes) seront jugés meilleurs car plus rapides ou plus courts (par exemple).
On appelle un algorithme conditionnel (ou un programme conditionnel) un algorithme qui ne suivra pas toujours les mêmes instructions en fonction des cas.
Voici 3 exemples d'algorithmes conditionnels résolvant (presque) le même problème.

Algorithme

Demander : Risque de pluie (oui/non) ? : reponse
si *reponse=oui* **alors**
  Afficher : "prends un parapluie"
fin
si *reponse=non* **alors**
  Afficher : "ne prends pas de parapluie"
fin

Algorithme

Demander : Risque de pluie (oui/non) ? : reponse
si *reponse=oui* **alors**
  Afficher : "prends un parapluie"
sinon
  Afficher : "ne prends pas de parapluie"
fin

Algorithme

Demander : Risque de pluie (oui/non) ? : reponse
si *reponse=oui* **alors**
  Afficher : "prends un parapluie"
fin

Par l'intermédiaire de questions, les exercices du lien ci-dessous te permettrons de comprendre les petites différences entre ces 3 algorithmes résolvant à priori le même problème (prendre ou pas son parapluie ?).

Exo 1

Remarque : certains algorithmes sont totalement illisibles sans utiliser un tableau de variables.

Algorithme

Demander : Premier nombre : a
Demander : Deuxième nombre : b
Demander : Troisième nombre : c
si $a < b$ **alors**
  $d \leftarrow a$
  $a \leftarrow b$
  $b \leftarrow d$
fin
si $b < c$ **alors**
  $d \leftarrow b$
  $b \leftarrow c$
  $c \leftarrow d$
fin
si $a < b$ **alors**
  $d \leftarrow a$
  $a \leftarrow b$
  $b \leftarrow d$
fin
Afficher : a,b,c

Remplir le tableau suivant en prenant par exemple $a = 2$, $b = 15$ et $c = 8$.

a	b	c	d

à la fin de l'algorithme, l'ordinateur écrit ....
à ton avis, quelle est l'utilité de cet algorithme ? : ...

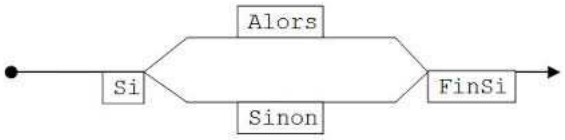
2 La commande SI-ALORS-SINON (langage naturel) ou IF- -ELSE (Python)

2.1 Cours

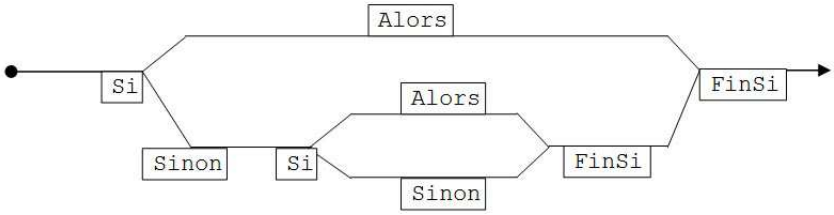
Définition

Dans un algorithme, on est parfois amené à exécuter une ou plusieurs instructions uniquement si une certaine condition est vérifiée : ce sont des instructions conditionnelles. Si la condition n'est pas vérifiée, on peut exécuter un autre bloc d'instruction ou ne rien faire. Dans tous les cas, ensuite, on exécute la suite de l'algorithme.

On peut schématiser un Si comme un aiguillage. La branche Sinon peut être inexistante dans ce cas on ne fait rien, plus exactement le programme continue. On peut aussi imbriquer plusieurs Si l'un dans l'autre.



SI-ALORS-SINON



Un premier SI avec un autre SI imbriqué dans le SINON

Le passage dans le "chemin" ALORS ou dans le "chemin" SINON est dépendant d'une condition. L'ordinateur va effectuer un test :

- si le test est VRAI, on prend le "chemin" ALORS pour effectuer quelque chose (les instructions 2 ci-contre).
- si le test est FAUX, on prend le "chemin" (facultatif) SINON pour peut être faire autre chose (les instructions 3 ci-contre).
- Dans l'exemple ci-contre, les instructions 1 et les instructions 4 sont toujours exécutées.

Algorithme

instructions 1
si ***TEST*** **alors**
  instructions 21
  instructions 22
sinon
  instructions 31
  instructions 32
fin
instructions 4

2.2 Python : Les tests classiques en mathématiques

Python n'utilise pas tout à fait les symboles mathématiques comme en math. Nous avons déjà rencontré deux exemples :

** qui veut dire PUISSANCE

= qui veut dire AFFECTER (la valeur de gauche prend la valeur de droite).

Tests les plus classique :

En Math	=	$\neq$	<	>	$\leq$	$\geq$	et	ou
En Python	==	!=	<	>	<=	>=	and	or

2.3 Python : La commande SI-ALORS-SINON en python IF- -ELSE

Dans ce chapitre, nous allons découvrir une règle essentielle pour l'utilisation de beaucoup d'autres commandes en Python. Au lieu de l'expliquer en général, prenons un exemple concret avec cet algorithme dont je me sers presque tous les jours.

Algorithme

Demander : dernière note en Math ? N
si $N > 12$ **alors**
  Afficher : Bravo
sinon
  Afficher : au travail
fin

programme Python

```
1 from math import *
2 N=float(input("note en math ?"))
3 if N>12 :
4   print("Bravo")
5 else :
6   print("au travail")
```

Fonctionnement de l'algorithme

3 questions pour savoir si tu as compris l'utilisation de la commande SI-ALORS-SINON :

- Qu'affiche l'algorithme si l'utilisateur donne 15 comme réponse à la question : ...
- Qu'affiche l'algorithme si l'utilisateur donne 8 comme réponse à la question : ...
- Qu'affiche l'algorithme si l'utilisateur donne 12 comme réponse à la question : ...

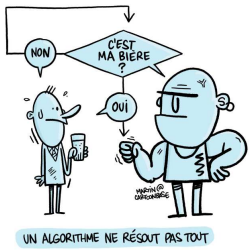
IMPORTANT Points à remarquer dans le programme Python

- Remarque la présence des deux petits points en fin de ligne 3. En simplifiant, cela signifie à l'ordinateur "si tu es là c'est que tu vas avoir des trucs à faire" et les instructions à faire (QUE DANS CE CAS LA) sont écrites dessous AVEC UN DECALAGE (on utilise le mot indentation).
- On remarque de nouveau les deux points ligne 5 et le décalage ligne 6.

2.4 A toi de jouer

Rappel : ces exercices sont obligatoires pour une bonne compréhension et ils vous permettent de comprendre vos erreurs (j'espère) car je peux voir (si vous le souhaitez) vos essais et vous conseiller. Exo 2 puis Exo 3

Algorithme - Python - 5



Résumé

Un problème très concret qui va permettre :
- de réviser le SI (une des notions les plus importantes en algorithmique).
- de découvrir la boucle TANT QUE.

1 Un problème concret très fréquent

1.1 programme un point en plus sur ta moyenne ?

Découverte de l'exercice.

Je te propose l'algorithme suivant (et le programme correspondant) qui, j'en suis sûr va t'intéresser ?

Algorithme

Demander : Un point en plus sur ta moyenne (oui/non) ? R
si $R =$ *"Oui"* **alors**
  Afficher : Pas de soucis, je rajoute un point
fin
si $R =$ *"Non"* **alors**
  Afficher : Ok, je laisse ta note telle quelle
fin
si $R \neq$ *"Oui" et* $R \neq$ *"Non"* **alors**
  Afficher : vu que tu réponds n'importe quoi, j'enlève un point
fin

programme Python

```
1 from math import *
2 R=input("veux tu un point en plus sur ta moyenne
      (oui/non) ? ")
3 if R=="Oui" :
4   print("Pas de soucis, je rajoute un point")
5 if R=="Non" :
6   print("Ok, je laisse ta note telle quelle")
7 if R!="Oui" and R!="Non" :
8   print("vu que tu reponds n'importe quoi, j'
      enleve un point")
```

Il ne te reste qu'à suivre le lien et à répondre à la question.
N'oublie pas de rendre ton travail pour que je puisse (peut-être) te rajouter un point. : EXO 1

Recherche d'une solution

En programmation, il existe très souvent plusieurs solutions pour résoudre un problème. Ce cours n'en développera qu'une.
Clique sur le lien pour y réfléchir (après avoir fait EXO 1) : EXO 2

1.2 Problème fréquent.

Ce problème de la réponse d'un utilisateur par oui ou non à une question d'un programme est très très fréquent. Par exemple :

- Veux tu quitter le jeux ?
- Prêt à jouer ?
- Es tu majeur ?
- ...

Face à la question oui ou non, le programme a 3 possibilités : soit c'est oui (avec ses différentes écritures car l'utilisateur n'est qu'un misérable humain), soit c'est non avec la même remarque sur l'utilisateur, soit c'est incompréhensible pour l'ordinateur.
Exemple : Un programme demande "Veux tu rejouer ?"

- Si oui (et ses variants), dans ce cas, on relance le programme.
- Si non (et ses variants), dans ce cas, on quitte le programme.
- Si c'est incompréhensible, très souvent dans ce cas l'ordinateur va attendre que l'utilisateur (le misérable humain) daigne répondre quelque chose de compréhensible.

Ce troisième cas, est très souvent traité avec une boucle TANT QUE (en anglais : une boucle WHILE).

2 La boucle TANT QUE ou boucle WHILE

2.1 Introduction aux boucles

Etude de 4 scènes de la vie courante :

- Scène n°1 : Pour passer votre permis de conduire, il va falloir apprendre à faire un créneau pour garer votre voiture.
- Scène n°2 : Vous devez mettre la table pour 14 personnes.
- Scène n°3 : Pour demain, il faut faire 5 exercices de mathématiques.
- Scène n°4 : Un professeur de musique apprend à chanter petit papa noël à une chorale.

Lien vidéo

Trouve ce qui est identique aux 4 scènes : ...
Des scènes 2,3 et 4, quelle est la scène qui ressemble le plus à la scène n°1 :...
La vidéo ci-contre est plus adaptée aux scènes n° ...

Conclusion.

Les boucles permettent de recommencer une partie du programme. Dans la vidéo, la scène 2 et la scène 3, on sait dès le départ combien de fois on doit recommencer alors que dans la scène 1 et la scène 4, on ne sait pas combien de fois on va recommencer.
On distingue :

- Les boucles POUR (en anglais : les boucles FOR) lorsque l'on connaît le nombre d'itérations.
- Les boucles TANT QUE (en anglais : les boucles WHILE) lorsque l'on ne connaît pas le nombre d'itérations.

2.2 Fonctionnement de la boucle TANT QUE

Dans les deux exemples de la vie courante correspondant à la boucle TANT QUE, cela donne :

- TANT QUE tu loupes les créneaux, il faut recommencer.
- TANT QUE la chanson n'est pas chantée juste, le chef d'orchestre fait répéter la chorale.

Le fonctionnement de cette boucle consiste en un **test** (égalité, différent, plus grand,...) tel que si le test est vrai, on recommence les instructions et si le test est faux, on sort de la boucle.

Initialisation des variables

Prenons l'exemple de la chorale qui apprend une chanson.

Algorithme

```
Demander : Quelle est la chanson apprise ? titre
tant que chant=bof faire
    Reprendre la répétition
fin
Afficher : "La chorale connaît la chanson",titre
```

Imaginons un ordinateur exécutant cet algorithme.

- Ligne 1, l'ordinateur affiche à l'écran, *Quelle est la chanson apprise ?* et il mémorise la réponse dans titre.
- Ligne 2, l'ordinateur regarde si on rentre dans la boucle, donc il teste chant=bof or chant n'existe pas !!!!! l'ordinateur bugue.

Il faut **AVANT** la boucle, donner des informations à l'ordinateur pour qu'il puisse faire le test. Les variables testées sont initialisées pour que le test soit vrai et donc aller dans la boucle.

Algorithme après la première correction (initialisation) :

Algorithme

```
Demander : Quelle est la chanson apprise ? titre
chant ← bof
tant que chant=bof faire
    Reprendre la répétition
fin
Afficher : "La chorale connaît la chanson",titre
```

- Ligne 1, l'ordinateur affiche à l'écran, *Quelle est la chanson apprise ?* et il mémorise la réponse dans titre.
- Ligne 2, on initialise la variable *chant*.
- Ligne 3, l'ordinateur regarde si on rentre dans la boucle, donc il teste chant=bof. VRAI donc on rentre dans la boucle.
- On fait une répétition.
- Fin de la boucle, l'ordinateur reteste pour savoir si on quitte ou si on recommence. *chant = bof*, donc on reste dans la boucle.
- On fait une répétition.
- etc,
- etc, etc

ATTENTION le gros risque des boucles TANT QUE est que l'on n'en sorte jamais. bug du programme donc du programmeur.
Dans cet algorithme, aucune instruction ne change la variable *chant* donc le test sera toujours VRAI et on restera toujours dans la boucle.

Voici un algorithme correct avec une boucle TANT QUE (variable initialisée et sortie possible).

Algorithme

```
Demander : Quelle est la chanson apprise ? titre
chant ← bof
tant que chant=bof faire
    Reprendre la répétition
    Demander : comment est le son (bof ou bien) ? : chant
fin
Afficher : "La chorale connaît la chanson",titre
```

Dans ce nouvel algorithme, après la répétition de la chorale, on demande au chef d'orchestre ce qu'il en pense.
Lorsqu'il répondra *bien*, la variable *chant* ne sera plus égale à *bof*.
Le TEST deviendra FAUX et on sortira de la boucle.
Le programme pourra continuer.

2.3 A vous de jouer ...

Plusieurs exemples de boucle TANT QUE en finissant (entre autre) le programme qui peut-être enfin te permettra d'obtenir un point en plus sur ta moyenne. : EXO 3

2.4 Le compteur de boucle

Une boucle TANT QUE répète des instructions TANT QUE le TEST est VRAI. Mais l'utilisateur ne sait pas combien de fois la boucle a été refaite. C'est une donnée qui peut être (très souvent) intéressante.
Ce problème très courant est résolu en incorporant un compteur de boucle. Cela consiste en une variable initialisée au début du programme à zéro et qui s'incrémente de 1 à chaque fois que l'algorithme (ou programme) effectue les instructions de la boucle.
A vous de jouer ... : EXO 4

Algorithme - Python - 6

Résumé

Les boucles POUR.
- On les appelle boucle FOR en Python.
- Bien faire la différence entre la boucle TANT QUE et la boucle POUR.

1 La boucle POUR en algorithmique

Différences entre boucle POUR (FOR en Python) et TANT QUE (WHILE en Python)

Dans la leçon précédente, nous avons pu remarquer :

- que les boucles permettent de répéter des instructions.
- qu'on utilise la boucle POUR quand on connaît le nombre de répétition.
- qu'on utilise la boucle TANT QUE quand on répète les instructions jusqu'à ce qu'un test devienne FAUX.
- qu'il est nécessaire d'utiliser un compteur avec les boucles TANT QUE si l'on veut connaître le nombre d'itérations.

Remarque : Si pour une boucle POUR, on doit connaître le nombre d'itérations à l'avance, l'ordinateur va compter de lui même le nombre d'itérations (pour s'arrêter quand il le faut) donc l'ordinateur utilise toujours un compteur de boucle pour les boucles POUR. C'est même le compteur de boucle qui va définir la boucle POUR.

Principe de fonctionnement des deux boucles

Boucle TANT QUE (WHILE en Python)

Algorithme

```
Instructions 1
TEST ← VRAI
compteur ← 0
tant que TEST faire
    Instructions 2.1
    Instructions 2.2
    compteur ← compteur + 1
fin
Instructions 3
```

Boucle POUR (FOR en Python)

Algorithme

```
Instructions 1.1
Instructions 1.2
pour compteur ← 0 à 8 faire
    Instuctions 2.1
    Instuctions 2.2
fin
Instructions 3.1
Instructions 3.2
```

2 La boucle FOR en Python (Exemple)

2.1 Premier exemple

Voici un exemple d'une utilisation de la boucle POUR (FOR en Python) que (j'espère) vous maîtrisez.

Algorithme

```
Table de multiplication de 5
pour compteur ← 1 à 6 faire
    mult← 5× compteur
    Afficher : compteur, "x 5 =", mult
fin
Afficher : fin de la table de multiplication
```

programme Python

```
1 from math import *
2 print("table de multiplication de 5")
3 for compteur in range (1,6) :
4   mult=5*compteur
5   print(compteur,"x 5 =",mult)
6 print("fin de la table de multiplication")
```

compteur	mult

Dans un premier temps, remplir le tableau des variables en utilisant l'algorithme. Puis suivre le lien afin de découvrir quelques particularités des boucles FOR en Python EXO 1

2.2 A vous de jouer . . .

Je vous propose 3 exercices comme application. Comme vous êtes un peu jeune pour comprendre le pourquoi du troisième exemple. Je vous propose de remonter dans le temps jusqu'en 1998 avec une vidéo explicative ci-contre.

Lien vidéo

3 applications : EXO 2

Algorithme - Python - 7

Résumé

Découverte des fonctions en algorithmique et programmation.
- pour gagner en efficacité.
- pour gagner en lisibilité.
- bref, que des avantages.

1 Les fonctions en programmation

1.1 Qu'est ce que c'est ?

Il ne faut pas confondre les fonctions mathématiques avec les fonctions en programmation. En fait en programmation (quelque soit le langage de programmation), on peut inventer nos propres commandes ou nos propres fonctions.

- print() est une fonction officielle de Python qui permet d'écrire.
- input() est une fonction officielle de Python qui permet de demander une information à l'utilisateur.
- de même pour int(), float(), ...

Ce cours montre les caractéristiques propres aux fonctions et va vous permettre sur un exemple de comprendre la création des fonctions.

1.2 Quand utiliser des fonctions

Pour les utilisateurs de windows (par exemple), n'avez vous pas remarqué que les fenêtres se ressemblent toutes ? Elles ont peut être des dimensions et des contenus différents mais elles ont la même couleur, les mêmes bordures, etc ...
Les programmeurs de windows, n'ont pas écrit des lignes de codage pour chaque fenêtre. Ils ont créé une seule fonction qu'ils *appellent* quand ils ont besoin d'une fenêtre (avec des tailles et des contenus différents).

Dans un programme, dès qu'une suite d'instructions revient plusieurs fois, il peut être intéressant de créer une fonction.

Avantage des fonctions

Les programmes sont plus courts.
Les programmes sont plus simples à lire, à corriger, à modifier, à faire évoluer (on ne corrige que la fonction et le programme restera opérationnel).

2 Révision : programmation d'un jeu de dés

Dans ce chapitre, nous allons programmer un jeu en n'utilisant que des commandes Python connues et sans utiliser de fonctions. Dans le chapitre suivant, on utilisera cet exemple pour introduire les fonctions.
Je vous conseille fortement de ne pas négliger ce chapitre afin de mieux comprendre la suite.

Règles et déroulement du jeu

Je vous propose comme sujet d'étude un jeu de dés (dés classiques à six faces). Ce jeu se joue à deux personnes (plusieurs variantes sont possibles : ici entre autres, le joueur B est avantagé mais ce n'est pas le but de ce cours :))).

- Phase 1 : Tout d'abord, les deux joueurs doivent se mettre d'accord sur l'enjeu de la partie (attention, les jeux d'argent sont interdits pour les mineurs).
- Phase 2 : Le joueur A décide du nombre de dés (comme il veut ...) et les lance une seule fois. On calcule la moyenne de points par dé.
- Phase 3 : Le joueur B décide du nombre de dés (comme il veut ...) et les lance une seule fois. On calcule la moyenne de points par dé.
- Phase 4 : Le joueur obtenant la meilleure moyenne par dé gagne le jeu.

Remarques

- Comme pour beaucoup de problèmes en programmation, il existe plusieurs solutions. Ce cours impose une solution afin de montrer l'intérêt des fonctions en programmation. La solution proposée n'est certainement pas ni la plus efficace, ni la meilleure.
- En suivant le lien, nous allons construire ce jeu pas à pas en utilisant les commandes officielles de Python. Si certaines parties ne sont pas évidentes, ne pas hésiter à me demander de l'aide.

EXO 1

3 Découverte des fonctions sur des exemples en Python

- L'utilisation d'une fonction pour remplacer une répétition de deux fois peut sembler inutile mais ceci est un exemple (imaginer pour plus de répétition). En informatique, il existe souvent de multiples façons de résoudre un problème.

- LOGIQUE : Il faut d'abord apprendre à l'ordinateur la nouvelle fonction (la nouvelle commande) avant de l'utiliser. On retrouvera donc la définition des fonctions en début de programme (ou dans un autre programme que l'on appellera comme le module math par exemple).
- En python, une fonction se définit ainsi (voir étude détaillée dans l'exemple du jeu de dés) :
  - Une première ligne : **def** *nomdelafonction*(*variablesd'entree*) :
  - suivie d'une indentation (à cause des deux petits points de la ligne précédente), avec toutes les instructions que devra effectuer la fonction.
  - Souvent une dernière ligne avec le résultat obtenu après avoir réalisé toutes les instructions de la fonction : **return**(*variablesdesortie*)

Algorithme

fonction *blabla(variableentree)*
  instruction 1
  instruction 2
  ...
fin

programme Python

```
1  def blabla(variablesentree) :
2      ...
3      ...
4      return(variablessortie)
5
```

- Pendant le déroulement de cette partie du programme, rien ne sera affiché à l'écran et l'utilisateur du programme n'interviendra pas. L'ordinateur repère les fonctions (enfin surtout, repère l'endroit où sont les instructions de la fonction) ...

3.1 Explications concrètes pour deux fonctions (jeu de dés du chapitre précédent)

- La partie du programme qui demande des informations sur les joueurs est pratiquement identique (sauf le nom des variables).
- La partie où l'ordinateur récupère les résultats des dés et où il fait les calculs est pratiquement identique.

A gauche le programme Python sans utiliser de fonction, à droite le même programme où deux fonctions sont utilisées.

- une première fonction nommée **identification** qui demande les informations des joueurs.
- une deuxième fonction nommée **demandeetcalcule** qui demande la valeur des dés de chaque lancer et qui calcule la moyenne par dé.

programme Python

```
1  from math import *
2  print("jeu de des : 2 joueurs, meilleure moyenne
       ")
3
4  prenomA=input("prenom joueur A ?")
5  nomA=input("nom joueur A ?")
6  ageA=int(input("age joueur A?"))
7
8  prenomB=input("prenom joueur B ?")
9  nomB=input("nom joueur B ?")
10 ageB=int(input("age joueur B?"))
11
12 nombredesA=prenomA+", combien de des?"
13 nbdesA=int(input(nombredesA))
14 totalA=0
15 for i in range (1,nbdesA+1) :
16   denumero="combien pour le de n"+str(i)+"?"
17   chiffrede=int(input(denumero))
18   totalA=totalA+chiffrede
19 moyenneA=totalA/nbdesA
20
21 nombredesB=prenomB+", combien de des?"
22 nbdesB=int(input(nombredesB))
23 totalB=0
24 for i in range (1,nbdesB+1) :
25   denumero="combien pour le de n"+str(i)+"?"
26   chiffrede=int(input(denumero))
27   totalB=totalB+chiffrede
28 moyenneB=totalB/nbdesB
29
30 if moyenneA<moyenneB :
31   print("le gagnant est",nomB,prenomB,ageB,"ans"
       )
32 if moyenneB<moyenneA :
33   print("le gagnant est",nomA,prenomA,ageA,"ans"
       )
34 if moyenneA==moyenneB :
35   print("Egalite, il faut recommencer le jeu")
36
```

programme Python

```
1  from math import *
2  print("jeu de des : 2 joueurs, meilleure moyenne
       ")
3
4  def identification() :
5    prenom=input("prenom joueur ?")
6    nom=input("nom joueur ?")
7    age=int(input("age joueur ?"))
8    return(prenom,nom,age)
9
10 def demandeetcalcule(nombrede) :
11   total=0
12   for i in range (1,nombrede+1) :
13     denumero="combien pour le de n"+str(i)+"?"
14     chiffrede=int(input(denumero))
15     total=total+chiffrede
16   moyenne=total/nombrede
17   return(moyenne)
18
19 prenomA,nomA,ageA=identification()
20 prenomB,nomB,ageB=identification()
21
22 nombredesA=prenomA+", combien de des?"
23 nbdesA=int(input(nombredesA))
24 moyenneA=demandeetcalcule(nbdesA)
25
26 nombredesB=prenomB+", combien de des?"
27 nbdesB=int(input(nombredesB))
28 moyenneB=demandeetcalcule(nbdesB)
29
30 if moyenneA<moyenneB :
31   print("le gagnant est",nomB,prenomB,ageB,"ans"
       )
32 if moyenneB<moyenneA :
33   print("le gagnant est",nomA,prenomA,ageA,"ans"
       )
34 if moyenneA==moyenneB :
35   print("Egalite, il faut recommencer le jeu")
36
```

3.1.1 Comparaison entre les deux programmes

- Programme n°2, la fonction *identification* est définie de la ligne 4 à la ligne 8 et est utilisée ligne 19 et ligne 20 (une fois par joueur).

- Programme n°2, la fonction *demandeetcalcule* est définie de la ligne 10 à la ligne 17 et est utilisée ligne 24 et ligne 28 (une fois par joueur).
- A part le titre, le cœur du programme n°1 se déroule de la ligne 4 à la ligne 35.
- A part le titre, le cœur du programme n°2 se déroule de la ligne 19 à la ligne 35. C'est beaucoup plus court. Le codage des deux fonctions peut être facilement sous-traité et étudié à part du reste.

3.1.2 Explication de la fonction *identification()*

- Le programme démarre, écrit le titre puis apprend la fonction **identification** et la fonction **demandeetcalcule**.
- Ligne 19 : le programme voit **identification()** et il se dit, "ah, mais ça j'ai appris tout à l'heure, il faut ..."
- Le programme fait donc ce qu'on lui a demandé de faire dans **identification()**. Ici, il doit demander du texte à mettre dans la variable *prenom*, encore du texte à mettre dans la variable *nom* et un nombre entier à mettre dans la variable *age*.
- Ensuite, le programme continue et arrive sur **return(***prenom*,*nom*,*age***)**. Il doit retourner là où il était en se rappelant les trois variables *prenom*, *nom* et *age* dans cet ordre.
- Quand le programme revient ligne 19, on lui demande de mettre les trois variables de la fonction (c'est dire, *prenom*, *nom* et *age*) dans les trois variables *prenomA*, *nomA* et *ageA* (dans le même ordre).
- Le programme passe ligne 20, voit **identification()** et il se dit, "ah, mais ça j'ai appris tout à l'heure, il faut ..."
- et on recommence tout mais en revenant, les trois variables de la fonction (c'est dire, *prenom*, *nom* et *age*) seront enregistrées dans les trois variables *prenomB*, *nomB* et *ageB*.
- Une dernière remarque sur la fonction **identification()**. Elle n'a pas de variable d'entrée (la parenthèse est vide) par contre, elle a trois variables de sortie (*prenom*, *nom* et *age*) qu'il faut se dépêcher d'enregistrer.

3.1.3 Explication de la fonction *demandeetcalcule()*

- La première différence entre la fonction **identification()** et la fonction **demandeetcalcule()** est que les parenthèses de la fonction **identification()** sont vides alors qu'il y a un nom de variable entre les parenthèse de **demandeetcalcule()**.
- La fonction **demandeetcalcule()** a besoin de connaître une variable pour être utilisée.

Voici une explication chronologique :

- ligne 10 : on indique à l'ordinateur le lieu où la fonction **demandeetcalcule()** est définie.
- le programme passe ligne 23 où l'ordinateur demande à l'utilisateur A le nombre de dés utilisés. Nombre que l'ordinateur enregistre dans la variable *nbdesA* (c'est un entier).
- ligne 24, l'ordinateur voit **demandeetcalcule()**, et l'ordinateur comprend qu'il doit aller ligne 10 avec le nombre enregistré dans *nbdesA*.
- ligne 10, le nombre contenu dans la variable *nbdesA* est affecté à la variable *nombrede*.
- l'ordinateur effectue toutes les instructions de la ligne 11 à ligne 17 (à cause de l'indentation).
- ligne 17, return indique à l'ordinateur de repartir d'où il est venu, c'est à dire ligne 24 avec une nombre dans la variable *moyenne*.
- ligne 24, on demande d'enregistrer dans la variable *moyenneA* le nombre de sortie de la fonction **demandeetcalcule()** (c'est à dire *moyenne*).
- ligne 28, l'ordinateur refera les mêmes instructions mais avec un nombre différent (le nombre de dés du joueurs B) et le résultat sera enregistré dans la variable *moyenneB*
- La fonction **demandeetcalcule()** possède une variable d'entrée (il lui faut un nombre pour fonctionner) et une variable de sortie (elle revient dans le programme avec un nombre *moyenne* qu'il faut se dépêcher d'enregistrer.

3.2 A vous de découvrir ... et de manipuler ...

Suivre les liens pour vous permettre de tester des programmes très simples utilisant des fonctions. Ces programmes n'ont comme intérêt que leur simplicité pour vous montrer quelques possibilités qu'offrent les fonctions.

Fonction et variable d'entrée : EXO 2
Variable de sortie : EXO 3

4 En conclusion

L'utilisation de fonctions en programmation est très très courante car :

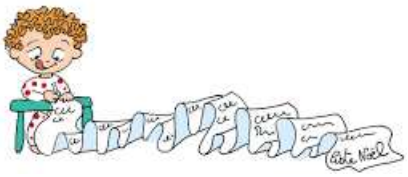
- elles permettent de diminuer la taille du programme.
- elles améliorent la lisibilité d'un programme.
- elles facilitent la recherche d'erreurs ou la modification d'un programme existant.
- elles simplifient le travail en équipes des programmeurs intervenant sur un même logiciel.

Au lycée, la programmation se limite souvent à de tous petits programmes donc l'utilisation de fonctions semble compliquer le programme, mais autant prendre de bonnes habitudes dès maintenant.

5 Bonus pour les joueurs ...

Le jeu de dés qui introduit ce cours est déséquilibré car le joueur n°1 à moins de chance de gagner que le joueur n°2.
Pour les volontaires, je tiens à votre disposition des idées de jeu à coder en Python avec une difficulté de codage mesurée et progressive. Bien entendu, ne pas hésiter à demander en cas de bug introuvable :)

Algorithme - Python - 8



Résumé

Découverte des listes en programmation :
- création de listes.
- modification de listes.
- utilisation des listes.

1 Les listes de variables dans les algorithmes

Une liste est une variable qui peut contenir plusieurs éléments. Les listes sont très pratiques à utiliser dès que des données sont répétitives.
La vidéo ci-dessous présente les tableaux et les listes de variables.
Le principe d'utilisation des listes et des tableaux est pratiquement identique. Au lycée, nous utiliserons essentiellement des listes. La différence entre les tableaux et les listes est principalement une différence de vitesse d'exécution, donc les tableaux sont souvent utilisés pour les gros, très gros programmes.

- Dans une liste, souvent les éléments sont de même type (entiers, décimaux ou chaînes de caractères).
- Dans une liste, un élément est repéré grâce à son indice.
- Très très souvent le premier indice est 0.
- Les boucles POUR sont très souvent utilisées avec les listes.

Vidéo 1

2 Les listes de variables en Python

2.1 Caractéristiques d'une liste en Python

- Une liste possède un nom : exemple la liste *blabla*
- Python utilise les crochets pour désigner une liste : L'exemple devient *blabla[]*
- Les éléments de la liste sont séparés par une virgule : Par exemple *blabla[2,"coucou",3.56]*
- L'indice du premier élément est 0. La liste *blabla[2,5,"coucou",3.56]* comporte 3 éléments (en indice 0, le nombre entier 2 suivi pour l'indice 1 du texte blabla et en dernier indice (2) le nombre décimal 3.56)
- Python autorise des listes avec des éléments de différents types.

exemple : *blabla[2,"coucou",3.56]*

2.2 Quelques manipulations utilisant des listes

Il existe quantité de manipulations avec les listes. Ce cours ne présente que les principales.

A vous de jouer : EXO 1

L'exercice ci-contre présente les actions suivantes :

- Création d'une liste (directement ou par élément) : *liste.append(rajout)*
- Affichage d'une liste ou d'un élément d'une liste : *print(liste[4])*
- Modification d'un élément de la liste
- Suppression d'un élément : *del liste[4]*
- Comptabilisation de tous les éléments d'une liste : *len(liste)*
- Comptabilisation d'un élément particulier dans une liste : *liste.count("a")*

Algorithme - Python - 9

Résumé

Découverte de 2 modules utilisés en Math :
- Le module pour remplacer le HASARD.
- Un module GRAPHIQUE.

1 Les modules Python

Python est un langage de programmation très utilisé dans des domaines très variés.
Chaque domaine (mathématiques, jeux, physique, économie, ...) a des besoins spécifiques. Python est un assemblage de plusieurs modules, chacun apportant des possibilités (relativement) spécifiques au langage Python.

L'utilisation de ces modules spécifiques permet de gagner en vitesse. Python n'a pas besoin de tout connaître pour un programme particulier, il se contente du strict minimum (défini par le programmeur).

Ce qui suit est une présentation rapide de plusieurs modules très utiles en Mathématiques au lycée (ainsi que dans les jeux ...). L'objectif est de présenter quelques commandes de base pour avoir un petit aperçu des possibilités offertes. Un ou des liens sont proposés pour des applications directes de ces nouvelles commandes (si possible sous forme de jeu).

2 Le module RANDOM

Ce module est très utilisé dans les jeux mais aussi en mathématiques pour simuler des situations car l'ordinateur calcule très vite.
Pour utiliser le module **random**, il suffit d'écrire en début de programme *from random import **

2.1 Quelques commandes ...

- *randint(2,5)* retourne un entier compris entre 2 et 5 (donc soit 2, soit 3, soit 4 ou soit 5) avec la même probabilité.
- *random()* retourne un décimal compris entre 0 et 1 (1 impossible) avec la même probabilité
- *uniform(4.5,6)* retourne un décimal au hasard compris entre 4.5 et 6
- *choice(liste)* retourne un élément d'une liste (il faut utiliser le nom de la liste).

2.2 Quelques exemples ...

Pour commencer : EXO 1 Pour progresser : Je tiens à votre disposition plein d'idées de petits programmes à réaliser tout en respectant une progressivité dans la difficulté de codage.

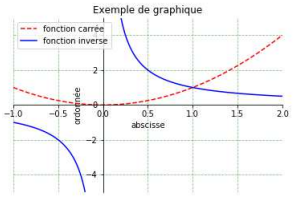
3 matplotlib et pylab

Il existe en python, beaucoup de modules différents pour dessiner sur l'écran. Chacun utilise ses propres commandes. De même, il est difficile de hiérarchiser l'ordre d'importance des différentes commandes.

3.1 Un exemple simple expliqué pas à pas.

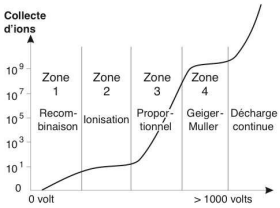
Le lien ci-dessous présente la construction pas à pas du graphique ci-contre avec le module pylab de matplotlib.

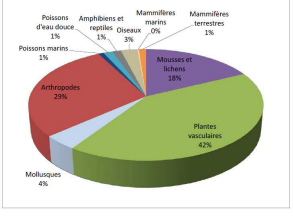
EXO 2

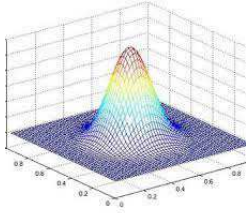


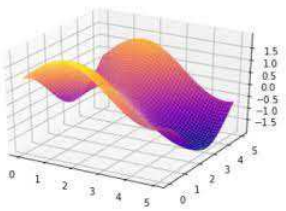
3.2 Quelques autres exemples de graphiques réalisés avec mathplotlib

Evidemment, on peut obtenir en python des graphes ou dessins bien plus complexes mais l'idée de ce cours n'est pas de faire des professionnels. Si vous voulez approfondir vos connaissances sur les possibilités graphiques de Python, vous trouverez sur la toile quantité de conseils (plus ou moins efficaces) pour vous aider.









Sites divers et variés pour compléments

Python Doctor
https://python.doctor/

Cours de Python de l'Unversité Diderot, Paris
https://python.sdv.univ-paris-diderot.fr/00_avant_propos/

Cours Python
https://courspython.com/sommaire.html

SofusPy, traducteur Scratch-Python
https://irem.univ-reunion.fr/blockly/plurialgo/blockly/extensions/sofuspy/run.html

Fiches Python pour ISN
https://fiches-isn.readthedocs.io/fr/latest/index.html

Python simple
https://python-simple.com/