Notes de lecture python Swynnen.

I À l'école de sorciers.

II Premiers pas.

Affectation ou assignation.

$n = 7$

Afficher la valeur d'une variable.

print() une fonction qui était dans les versions antérieures de python une instruction. Permet d'afficher plusieurs valeurs séparées par des virgules.

Typage des variables.

Automatique. Typage dynamique contraire C++ et Java.
$n = 3$. Type nombre entier.
$msg =$ "*testdemessage*". Type chaîne de caractères.
$pi = 3.1415$. Type nombre à virgule flottante.
type() fonction donnant le type d'une variable ou d'une expression.

Affectations multiples.

$a,b = 4,4$. Elles sont simultanées (ce qui permet l'affectation parallèle ou échange : $a,b = b,a$.

Opérateurs et expressions.

Les valeurs et variables sont combinées entre elles avec des opérateurs constituant ainsi des expressions.
Opérateurs : $+, -, *, /$.
// division entière (quotient dans la division euclidienne).
** exponentiation.
% reste de la division euclidienne.

Priorité des opérations.

Règle usuelle : PEMDAS

Composition.

Nous avons vu : variables, expressions (variables combinées avec des opérateurs), instructions (commandes implémentées de base dans python).

Python langage de programmation de haut niveau. Il est donc possible de combiner des instructions, des expressions, etc. : $print(20 + 3)$.

III Contrôle du flux d'exécution.

Flux d'exécution : chemin suivi par python dans le programme.

Trois structures de contrôle du flux d'exécution : séquence, sélection (conditionnel) et répétition.

Séquence d'instructions.

Les commandes dans un programme s'exécutent dans l'ordre ou elles sont écrites. Un bloc d'instructions qui se succèdent est appelé une séquence.

Sélection ou exécution conditionnelle.

Les deux petits points et l'indentation (quatre espaces ou tabulation) sont obligatoires

```
if condition :
        instruction
```

Il vaut mieux que les indentations soient constantes.

Si l'on souhaite effectuer une instruction lorsque la condition n'est pas vérifiée.

```
if condition :
        instruction 1
else :
        instruction 2
```

Il est même possible de distinguer trois cas :

```
if condition 1 :
        instruction 1
elif condition 2 :
        instruction 2
else :
        instruction 3
```

Opérateurs de comparaison.

Pour faire des conditions : == (égale), ! = (différent), >, <, >=, <=.

Instructions composées, blocs d'instructions.

Une instruction composée est une instruction qui s'applique à un bloc d'instructions. if est un exemple d'instruction composée.

ligne d'en-tête de l'instruction :
    instruction 1
    ⋮
    instruction n

Instructions imbriquées.

Pas de problème.

Quelques règles de syntaxe Python.

Les commentaires commencent par #.
Les indentations font partie du langage de programmation.

IV Instructions répétitives.

Réaffectation.

Une variable peut prendre différentes valeurs.
Affectation parallèle pour échanger les valeurs des variables : a,b=b,a.

Répétitions en boucle : l'instruction while.

while *condition* :
    instruction

Premiers scripts, ou comment conserver nos programmes.

Pour lancer un script : python3 monscript.py
Les commentaires dans un script sont précédés de #.

V Principaux types de données.

Les données numériques.

Le type *integer*.
Pour les entiers python3 donne des valeurs exactes même si les nombres sont très grands. Le temps de calcul est simplement augmenté.

Maximum en valeur absolu d'un entier codé avec 32 bits sous python3 2 147 483 647. Les plus grands nombres ou les réels nécessitent un travail de codage et décodage supplémentaire.

Le type *float* : il s'agit des nombres décimaux

Pour qu'un nombre soit vu comme du type float il faut qu'il contienne le point décimal ou l'exposant 10 : 3.14, 10., .001, $1e100$, $3.14e-10$

Les nombres de types float peuvent être écrit avec autant de chiffres que les nombres entiers cependant dès que la limite des nombres entiers est dépassée, la notation devient scientifique. Les nombres de type float sont, en valeur absolue, compris entre 10^{-308} et 10^{308}. Ainsi : 12 chiffres significatifs et un exposant de 10 (entre -308 et 308).

Les données alphanumériques.

Mots, phrases, nombres, mélange de tout ceci. Cet objet est appelé une chaîne de caractères.

Les chaînes de caractères sous python3 sont des objets du type *string*.

Pour créer un objet de type string il faut le mettre entre apostrophes (simple quote) ou guillemets (dubble quote) $text =' "Truc" etmachin'$ ou $text =' Bidule'$ ou $text = "zoup"$.

Pour faire du verbatim et mettre ce qu'on veut dans une chaîne de caractère nous utiliserons le triple quote : """machin bizarre, présentation ou caractères"""

Les chaînes font partie, dans python3, des séquences lesquelles sont des collections ordonnées d'éléments.

Pour appeler les éléments de la chaine $ch =$ "*testdetruc*" nous ferons $ch[0]$, $ch[3]$. La numérotation de la liste de caractères commence à 0.

Opérations sur les chaînes :

1. concaténation (assembler des chaînes) : $chaineconcatenee = ch1 + ch2$.
2. fonction longueur de la chaîne (nombre de caractères) : $len()$.
3. convertir une chaîne en nombre : $ch =' 87'$, ch n'est pas un nombre, pour le rendre nombre utilisons la fonction $int()$: $n = int(ch)$ ou $n = int(ch[2])$.

les *listes* sont des collections d'éléments séparés par des virgules et encadrées de crochets. Elles peuvent contenir tous les types de données *integer*, *float*, chaîne de caractères et *listes*.

Il est possible d'appelé un éléments d'une liste : $liste[0]$, $liste[2]$. Contrairement aux chaînes de caractères les éléments d'une liste sont modifiables individuellement : $liste[0] = 3$ (modification du premier terme de la liste. Donc aussi interchangeables. La fonction $len()$ peut être utilisée sur les listes.

$del()$ est une fonction permettant de supprimer un élément de la liste.

Pour ajouter un élément à une liste nous utilisons le fait que la liste est un *objet* dont nous allons utiliser une *méthode* : $liste.append('32')$ rajoute la chaîne

de caractères $'32'$ à la liste *liste*. *append*() est la méthode appliquée à l'objet *liste* avec l'argument $'32'$.

Il est possible de créer une liste vite : $liste = [\,]$

VI Fonctions prédéfinies.

Les fonctions sous python sont des sous-programmes que l'on peut appeler dans un autre programme. Les fonctions admettent différents arguments et paramètres.

La fonction *print*().

Une fonction comme *print* admet de nombreuses variables appelées arguments qui peuvent être des données d'entrée comme des paramètres de réglage, de configuration de la figure.

sep est argument (paramètre) de la fonction *print*() il indique la façon dont sont séparées les différentes variables affichées : $print("test",3,sep = "?")$ affiche *test*?6.

end est argument (paramètre) de la fonction *print*() il indique la façon dont s'enchainent des affichage. $\begin{matrix} print("test",end = "") \\ print(6) \end{matrix}$ permet de mettre les deux affichage sur la même ligne. Très pratique pour les affichages en boucle qu'on souhaite aligner.

Interaction avec l'utilisateur : la fonction *input*().

input() ou *input*("*textdescriptif*"). Le résultat de *input*() est une chaîne que l'on peut enregistrer. Pour obtenir un nombre il faut transformer la chaîne avec les fonctions *int*() ou *float*().

Importer un module de fonctions.

Un module de fonctions est un package avec des fonctions déjà définies et utilisables. Autrement dit c'est un ensemble de sous-programmes. Le module *math* regroupe de nombreuses fonctions (au sens mathématique) de fonctions.

Pour appelé le module il faut écrire en entête de script

from math import $*$

$*$ signifie qu'il faut importer toutes les fonctions du module *math*.

Un peu de détente avec le module turtle.

Véracité ou fausseté d'une expression.

La valeur zéro indique faux les autres valeurs signifient vrai.

Pour python le vrai est enregistré avec la valeur un.

Une chaîne de caractères est vraie si elle n'est pas vide.

Révisions.

L'association des fonctions *eval*() et *list*() permet de convertir en liste toute chaîne de valeurs séparées par des virgules : $list(eval(ch))$.

Il est possible d'utiliser & comme connecteur logique à la place de *and*.

Il est possible de prendre la négation d'une proposition logique a en écrivant *not a*.

VII Fonctions originales.

Une *fonction* sous python est un sous programme.

Nous verrons deux autres types de sous programmes : les *classes* et les *objets*.

```
def nomdelafonction(liste de paramètres) :
    instruction
```

La liste de paramètres peut rester vide mais il faut laisser les parenthèses.

def est une instruction comme *while*, *if*.

Fonction simple sans paramètres.

Une fonction peut utiliser une autre fonction. Les fonctions permettent d'éviter des répétition de code dans un programme, simplifier le corps principal d'un programme en délégant une partie du code.

Les fonctions sont des sous-programme.

Les fonctions sont des instructions personnalisées.

Fonction avec paramètre.

Les paramètres à choisir pour définir la fonction sont appelés des *arguments*.

La variable locale (propre à la fonction) qui reçoit l'argument s'appelle le *paramètre*.

Utilisation d'une variable comme argument.

Il est possible de donner en argument à une fonction une variable du corps principal du programme.

Variables locales, variables globales.

Les variables données en paramètres lors de la définition de la fonction sont des variables locales. Il est impossible de les appeler dans le reste du programme.

Les autres variables, celles qui sont accessibles dans le programme sont appelées des variables globales.

Si une variable locale d'une fonction porte le même nom qu'une variable globale cela n'a pas d'importance car ce sont deux variables de facto distinctes et dès que le programme sort de la fonction la variable locale a été effacée.

Il cependant possible d'imposer l'utilisation d'une variable globale dans la définition de la fonction.

```
def monter():
        global a
        a=a+1
        print(a)
```

Nous renverra successivement les valeurs a+1, a+2, a+3 , etc.

Vraies fonctions et procédures.

Les fonctions définies ci-dessus sont en fait des *procédures* en ce qu'il s'agit de sous programmes qui ne renvoient pas forcément une valeur (integer, float, chain, list) contrairement aux *fonctions*. Pour cela il faut utiliser l'instruction *return* en sortie de fonction qui définie la valeur retournée.

```
def cube(w):
        return w*w*w
```

il est alors possible d'affecter cube(3) a une variable globale du programme.

return peut également être utilisé sans aucun argument, à l ' intérieur d ' une fonction, pour provoquer sa fermeture immédiate. La valeur retournée dans ce cas est l ' objet None (objet particulier, correspondant à « rien »).

la méthode methode4() d'un objet *objet3* , à l'aide d'une instruction du type : *objet3.methode4()* , c'est-à-dire le nom de l'objet, puis le nom de la méthode, reliés l'un à l'autre par un point. Ce point joue un rôle essentiel : on peut le considérer comme un véritable *opérateur*.

Utilisation des fonctions dans un script.

Il faut définir les fonctions avant de les appeler dans le programme.

Le programme principal (main) est donc placé à la fin du script après les définitions des fonctions.

Modules de fonctions.

Les fonctions sont en général définies dans un module et les programmes dans un autre.

Possibilité d'intégrer dans une fonction des commentaires observables avec certains éditeurs : il suffit de mettre une chaine (ente guillemets) à la place d'une instruction dans la définition de la fonction. Ces commentaires sont associé à l'attribut

`__doc__`

de la fonction.

`print(fct.__doc__)`

affichera les commentaires de la fonction fct.

Essayer de créer un module.

Typage des paramètres.

Encore dynamique du coup il est possible de mettre de nombreux arguments dans les fonctions.

Valeurs par défaut pour les paramètres.

Il est possible de donner des valeurs par défaut au paramètre en précisant dans la définition de la fonction.

```
def fonction(argument1,argument2='ValeurParDefaut')
```

les arguments acceptant des valeurs par défauts doivent être données en dernier.