Les Variables :

Introduction : Nous allons aborder dans ce chapitre une notion fondamentale de tout Langage de Programmation. Les Variables sont un moyen de retenir, de modifier et de réutiliser des données de tout type lors de l'exécution de votre Programme.

Les Variables :

Pour illustrer l'objet de ce Chapitre, prenons un exemple. Vous souhaitez écrire un programme capable d'additionner, de soustraire, de multiplier ou de diviser deux nombres choisit par l'utilisateur, c'est ce que nous allons apprendre à faire dans la suite. Il nous faut d'abord pouvoir enregistrer les nombres en question. Voyons un premier exemple :

```
NombreUn = 15
NombreUn = 17
NombreDeux = 13
NombreTrois = NombreUn+NombreDeux
NouvelleVariable = 15
```

Que vient on de faire ? Analysons ce programme ligne à ligne. D'abord nous trouvons l'instruction : NombreUn = 15, Cette ligne c'est ce qu'on appelle une Affectation de Variable (Soyez en fiers, c'est votre première vraie Instruction de Programmeur ! :P) Elle permet de relier le nom NombreUn à la valeur 15. On trouve ensuite : NombreUn = 17, On modifie simplement la valeur désignée par NombreUn qui ne vaut plus 15 mais 17. La Ligne Suivante : NombreDeux = 13 Est une nouvelle Affectation. Puis on lit : NombreTrois = NombreUn+NombreDeux Il faut l'interpréter comme suit : NombreTrois désigne la valeur égale à la somme des valeurs désignées par NombreUn et NombreDeux, Ici comme NombreUn et NombreDeux valent 17 et 13.NombreTrois vaut donc 30. Pour finir on trouve une nouvelle Affectation avec : NouvelleVariable = 15

L'idée qui se cache derrière le nom même de variable c'est celle que la quantité désignée par le nom (En jargon Informatique on préfère dire l'Identificateur ;)) de la Variable peut changer, peut varier. Ainsi NombreUn valait 15 au début du programme et 17 à la fin. Ce mécanisme est très pratique et même indispensable, vous vous en rendrez vite compte !

Règles de Nommage :

L'écriture des Identificateurs est soumis à certaines règles . Les Voici :

1 : Un Identificateur ne peut être constitué que de lettres, de chiffres et du caractère souligné .

2 Un Identificateur ne peut pas commencer par un chiffre

3 : Il existe certains mots réservés du langage (Ils sont une trentaine) que vous ne pouvez pas utiliser comme identificateurs. En voici la liste : and, as, assert, break, class, continue, def, del, elif, else, except, false, finally, for, from, global, if, import, in, is, lambda, none, nonlocal, not, or, pass, raise, return, true, try, while, with, yield.

Par ailleurs les Identificateurs sont soumis à la casse, ils différencient donc les majuscules et minuscules. Si vous écrivez :

```
Variable = 15
VARIABLE = 20
```

Vous avez défini deux variables différentes

Tandis que si vous écrivez :

```
Variable = 15
Variable = 20
```

Vous avez défini une Variable mais vous en avez changé la valeur en cours de route ;)

Les Types De Variables :

Nous avons vu quelques premiers exemples élémentaires, nous avons affecté à des variables des valeurs entières, 15,17,13,20 ... Mais Python offre beaucoup plus de possibilités ! Nous pouvons par exemple utiliser les nombres à virgule, voici une illustration :

```
NombreAVirgule = 1.5
DeuxiemeNombreAVirgule = 7.897567
```

Nous avons défini deux Variables (On peut aussi dire que nous avons déclaré deux Variables) dont les Identificateurs sont : NombreAVirgule et DeuxiemeNombreAVirgule désignant respectivement les valeurs 1,5 et 7,897567 (Remarquez qu'on substitue le point à la virgule, nous sommes en notation Anglo-Saxonne). De même, il est tout à fait possible d'utiliser des nombres négatifs :

```
NombreNegatif = -518787
Zero = -0
SecondZero = 0
NombrePositif = -NombreNegatif
```

Et plus fort encore, des nombres négatifs décimaux ! :P

```
NombreNegatifDecimal = -78778787.8755878787
MoinsPi = -3.14159265358979
```

Ainsi, nous avons vu les différents types de nombres que l'on peut utiliser nativement avec Python à notre niveau, enchaînons sur les chaînes de caractères justement ! N'ayez pas peur, chaîne de caractères ce n'est rien d'autre qu'un mot compliqué pour dire du texte ;)

Pour stocker du texte il nous suffit de l'enregistrer entre guillemets :

```
MonTexte = "Encore une fois sur la Brèche mes amis... !"
MonSecondTexte = "Je travaille dur donc je réussi :D"
```

On trouve aussi d'autres syntaxes, entre apostrophes, entre triples apostrophes ou entre triples guillemets ! :

```
NouveauTexte = 'Je comprends Python avec ce super bouquin :P'
EncoreDuTexte = """A ce rythme je vais devenir un expert du Python
X)"""
CaEnFaitDuTexte = '''Just Do It ! Yes You Can ;)'''
```

On peut se demander à quoi servent ces deux dernières syntaxes, la réponse est simple, imaginez que vous vouliez enregistrer du texte qui contient un apostrophe ou un guillemet :

```
Test = 'C'est Problématique Hmmm'
```

Et oui, quand Python va trouver votre apostrophe il va penser qu'elle ferme la chaîne de caractères et il ne saura pas quoi faire de la suite ! Idem pour un guillemet ;) Pour ça on peut donc utiliser les délimiteurs faits de trois guillemets ou de trois apostrophe ou alors on peut "échapper" le guillemet ou l'apostrophe problématique en le faisant précéder d'un backslash :

```
Test = 'C\'était Problématique Hmmm :P Le Problème a disparu :)'
```

Je vais profiter de cette partie pour introduire un nouvel outil, la fonction type ! Si vous ne savez pas ce qu'est une fonction, pas d'inquiétude, on part de zéro ici ! L'idée est qu'une fonction est une suite d'instructions qui vont travailler, éventuellement sur une ou plusieurs données et éventuellement donner un résultat.

Sachez qu'on consacrera un chapitre entier aux fonctions dans quelques temps. Si vous voulez un exemple dites vous qu'une fonction pourrait par exemple calculer la somme de deux nombres. On pourrait aussi avoir une fonction qui donne le coût d'un article en magasin en fonction de la TVA ;)

Si vous n'êtes pas encore tout à fait sûr de comprendre ce n'est pas grave, on y reviendra et on va faire un exemple !

Les fonctions aussi ont des identificateurs, des petits noms qui nous permettent de les appeler et d'exécuter les instructions qu'elles renferment ! Ici on va voir la fonction type :

```
Entier = 42
Decimal = 1.21
Texte = "9.21 Gigowatts ? ! Incroyable !"
type(Entier)
type(Decimal)
type(Texte)
```

Voyons voyons :

```
<class 'int'>
<class 'float'>
<class 'str'>
```

Ne prêtez pas attention aux mentions "class" (Elles ne vous la rendraient sans doute jamais !) Ce que nous voyons la c'est le type de la donnée sur laquelle on travaille : int pour integer, entier (42 est un Entier, enfin je crois), float pour Flottant c'est à dire nombre Décimal (1.21 étant Décimal) et str pour string c'est à dire chaîne (de caractères non mais oh ! x).

On remarquera qu'un :

```
Texte1 = "7"
Texte2 = "787.56"
type(Texte1)
type(Texte2)
```

Nous affiche :

```
<class 'string'>
<class 'string'>
```

Et oui Python considère que tout ce qui est entre apostrophes, guillemets, triples guillemets ou triples apostrophes est du texte et ça a son importance !

Les Opérations Autorisées :

Pour commencer, on peut bien sur ajouter deux nombres, qu'ils soient positifs, négatifs, décimaux ou non. Pour cela il suffit d'utiliser le symbole +. Seul petit bémol concernant ces derniers à cause de certains écueils relatifs à l'encodage des nombres à virgule : la précision mathématique n'est pas absolue (3.11+2.08 Ne donnera pas un résultat exact). Nous nous en contenterons mais j'évoquerai quelques solutions potentielles à ce problème, lorsque nous serons déjà de fiers développeurs confirmés, les cheveux flottants au vent, la classe x). La soustraction fonctionne de la même manière, remplacez simplement le symbole + par le symbole - ! Passons à la multiplication, nous utiliserons le symbole *. Pour la division c'est le symbole / qui prévaut (Essayez de diviser un nombre par zéro et observez ce qu'il se passe. (pas de panique c'est normal :P) Enfin une

opération peut être nouvelle pour vous, le modulo, il s'agit simplement du reste de la division euclidienne, on le note %. Par exemple 5%2 = 1, 10%5 = 0, 1000%10 = 0... Enfin la division entière, identique à la division ordinaire mais son résultat est tronqué à sa partie entière. On utilise // pour l'exécuter. On a 10//3 = 3, 25//5 = 5 30//5 = 6

A savoir que, lorsque l'on mélange nombre flottant et nombre entier dans un produit, une somme ou une soustraction le résultat est un nombre décimal (Même si sa partie décimale est nulle (Python différencie 2 et 2.0, par exemple) Le reste d'une division quant à lui est toujours un nombre décimal (Peu importe si sa partie décimale est à nulle).

Pour les chaînes de caractères on définit deux opérations principales : la somme (qui correspond à la concaténation) et le produit par un entier. Si vous faites :

```
Texte = "Ceci Est"+"Un Essai"
Texte2 = "Encore un Texte Zzzzzz..."*3
```

Alors Texte contiendra "Ceci Est Un Essai" et Texte2 Contiendra "Encore un Texte Zzzzzz...Encore un Texte Zzzzzz...""Encore un Texte Zzzzzz..." ;). Il se peut que vous vouliez ajouter dans votre Texte des caractères tels que le saut de ligne ou la tabulation. Pour cela il ne suffit pas d'insérer les dits caractères comme on le ferait dans un traitement de texte. On utilise en fait des séquences d'échappement, par exemple \n correspond à un saut de ligne, \t à une tabulation et \a à un signal sonore. Par ailleurs les chaînes de caractères sont manipulables d'une façon assez particulière : imaginez que chaque caractère soit numéroté à partir de 0. Ainsi dans la chaîne : "ABCD", A est le caractère numéro 0, B le caractère numéro 1, C le numéro 2 et D le numéro 3. Python nous permet de sélectionner une partie de notre chaîne. Si nous faisons "ABCD"[0] Nous trouvons "A", "ABCD"[1] Nous trouvons "B" Et ainsi de suite... Nous pouvons aussi faire "ABCD"[0:2] Pour sélectionner les caractères de la première incluse à la troisième exclue par exemple. On a donc "ABCD"[0:4] = "ABCD" ;)

Nous reviendrons par ailleurs sur les chaînes de caractères dans un chapitre ultérieur. Dernière fonctionnalité, on peut essayer de convertir une donnée d'un type en un autre en la faisant précéder du type cible et en l'entourant de parenthèse.

```
Decimal = 7.88
Entier = int(Decimal)
SecondEntier = 7
Flottant = float(SecondEntier)
TroisiemeEntier = 189
Texte = str(ToisiemeEntier)
Texte = Texte*3
```

Alors Entier vaudra 7 (On prend la partie Entière ;)) Flottant vaudra 7.0 (Que Python différencie de 7) Et Texte vaudra "189189189"

On écrit souvent des opérations du type :

```
Variable = Variable+Valeur
Variable = Variable*Valeur
Variable = Variable/Valeur
Variable = Variable-Valeur
...
```

De manière générale cela équivaut à Variable Opérateur= Valeur. Par exemple Variable = Variable+8 équivaut à Variable += 8 (On augmente la valeur de Variable de 8), Variable = Variable*2 équivaut à Variable *= 2 ...

J'ajoute que pour afficher un élément à l'écran on dispose là encore d'une fonction, la fonction print !

Indiquez simplement entre parenthèses ce que vous souhaitez afficher :

print("Du Texte")

print(515)

...

Vous pouvez aussi indiquer plusieurs éléments à afficher en les séparant par une virgule

(Avec plusieurs print nous aurions des sauts de ligne, nous apprendrons à résoudre ce problème à l'avenir)

Les Conditions

Introduction : Les Conditions constituent un moyen de casser la linéarité de notre programme. Jusqu'ici, nous n'avions aucun moyen d'influer sur le déroulement de celui ci mais ce temps est révolu ! Nous allons voir comment Python nous permet d'exécuter certaines instructions dans un cas et d'autres dans un autre cas. Vous êtes prêts ? C'est parti !

Les Variables Booléennes :

Nous allons introduire un nouveau type de données qui va s'ajouter à ceux que nous avons déjà mentionnés lors du dernier chapitre. Ce type c'est celui des Variables Booléennes. Rassurez vous rien de compliqué ! Une Variable Booléenne n'aura en fait qu'une valeur de vérité, elle sera soit vraie soit fausse.
Illustrons ce principe :

```
LaTerreEstPlate = False
UnEtUnFontDeux = True
```

C'est aussi simple que ça, le False correspond à Faux et le True à Vrai !

Les Opérations sur les Variables Booléennes :

Nous avons vu au précédent chapitre que chaque Type était muni d'opérations spécifiques, les nombres peuvent être multipliés, divisés, soustraits, ajoutés… Les Chaînes de Caractères ne peuvent être qu'ajoutées entre elles ou multipliées par un nombre, en outre il est impossible de les diviser, par exemple. Le Type booléen lui est muni de 3 opérations principales : and, or et not. Ces opérations nous permettent de combiner des variables Booléennes entre elles, mais trêve de bavardages voyons un exemple.

```
LaTerreEstPlate_ET_UnEtUnFontDeux = False and True
TrueLaTerreEstPlate_Ou_UnEtUnFontDeux = False or True
LaTerreEsTNonPlate = not LaTerreEstPlate
```

L'opérateur And nous permet d'avoir une expression booléenne qui n'est vraie que si tous ses opérandes sont vrais. L'opérateur Or quant à lui nous permet d'obtenir une expression booléenne qui est vraie si au moins l'un de ses opérandes est vrai. Finalement l'opérateur Not nous permet d'avoir la négation de son unique opérande.

Ici : LaTerreEstPlate_ET_UnEtUnFontDeux vaut False and True c'est à dire False (Il aurait fallu avoir True and True pour qu'elle désigne True) LaTerreEstPlate_Ou_UnEtUnFontDeux vaut False or True, c'est à dire True. Finalement LaTerreEsTNonPlate vaut not False et donc True. ;)

Par ailleurs nous pouvons affecter à des Variables Booléennes des valeurs dépendantes d'autres Variables, exemple :

```
Entier = 7
EntierEstSuperieurA8 = Entier > 8
EntierEstSuperieurOuEgalA3 = Entier >=3
```

(Ici les deux Booléens valent respectivement False et True)
La syntaxe se comprend aisément, pour ce type de comparaisons on dispose des opérateurs :

```
> Supérieur
>= Supérieur ou Egal
< Inférieur
<= Inférieur ou Egal
== Egal
```

Vous remarquerez que l'égalité se teste avec un double égal (Pour ne pas confondre avec l'affectation de Variable) ne vous trompez donc pas.

Les Mots-Clés If, Elif et Else :

```
if LaTerreEstPlate :
     print("C'est la Fin du monde!")
else :
     print("Tout va bien")
```

N'attendons pas plus longtemps avant que je ne vous révèle ce que ces mots signifient.
If nous permet d'exécuter un bloc d'instructions si une condition est vérifiée :
Nous y avons ajouté un Else, celui ci permet tout simplement d'exécuter des instructions après un if si la condition qui accompagnait ce dernier était Fausse. Il correspond à un "Sinon".
Finalement on a le Elif qui permet d'implémenter le "Sinon si". Exemple :

```
Entier = 8
if Entier == 9 :
        print("Entier vaut 9 !")
elif Entier == 7 :
        print("Entier vaut 7")
else :
        print("Entier ne vaut ni 9 ni 7")
```

Ici notre If teste d'abord si Entier vaut 9, si ce n'est pas le cas il teste si il vaut 7, et si ce n'est toujours pas le cas nous entrons dans la clause du else.

L'Opérateur Ternaire :

Dans le cas ou nous avons un bloc conditionnel de la forme :

```
if Condition :
     Instructions
else :
     NouvellesInstructions
```

Il peut être plus compact d'écrire (ValeurSiFaux,ValeurSiVrai)[Condition] Attention, il faut bien indiquer une Valeur dans chaque cas, ne faites donc pas : (A = 5, A = 9)[Condition] mais plutôt A = (5,9)[Condition].

Voyons une nouvelle fonction très utile, la fonction : Input. Cette fonction vous permettra de demander à l'utilisateur de saisir une donnée, faites simplement :

```
SaisieUtilisateur = input()
```

Néanmoins, par défaut le type de la valeur donnée par input est toujours celui d'une chaîne de caractères. Si vous demandez à l'utilisateur de saisir autre chose n'oubliez pas d’effectuer la

conversion :

```
Entier = input()
Entier = int(Entier)
```

Ce qui équivaut d'ailleurs à :

```
Entier = int(input())
```

Les Boucles

Introduction : Ce nouveau chapitre est consacré à une fonctionnalité qui vous sera indispensable dans votre future vie de développeur. Les Boucles. Avec elles nous pourront répéter des instructions autant de fois que nous le voudrons et à l'issu de ce chapitre vous aurez rédigé votre premier vrai programme fonctionnel.

Les Boucles :

Introduisons sans plus attendre la première forme de boucle que nous allons voir. La boucle "While" (Tant Que) Cette boucle permet d'exécuter une portion de code tant qu'une condition est vérifiée, voyons un exemple.

```
Compteur = 0
while Compteur < 5 :
        print("Bonjour")
        Compteur += 1
```

Nous définissons d'abord un Compteur initialisé à 0. Puis : Tant que ce compteur est inférieur à 5 Nous affichons Bonjour et nous Augmentons Compteur de 1 (Si nous n'avions pas ajouté cette dernière ligne la condition aurait été éternellement vérifiée et la boucle ne se serait jamais arretée (On parle de boucle infini dans le Jargon du Programmeur)). Sans utiliser les boucles il nousv aurait fallut écrire :

```
print("Bonjour")
print("Bonjour")
print("Bonjour")
print("Bonjour")
print("Bonjour")
```

Autant dire que pour un grand nombre de répétitions les boucles deviennent très vite indispensables. Nous avons vu un premier type de boucles, Python nous en fournit un second, la boucle for. La boucle for nous permet plus spécifiquement de parcourir des séquences, par exemple une Chaine de Caractère est une séquence de Caractères. (Rappellez vous par exemple qu'on a "ABCD"[0:2] == "AB" Ou "DEF"[0] == "D" ...). Ainsi si avec une boucle while nous devrions procéder ainsi :

```
SaisieUtilisateur = input()
Compteur = 0
while Compteur < len(SaisieUtilisateur) :
        print(SaisieUtilisateur[Compteur])
        Compteur += 1
```

La fonction len nous permet de connaitre la longueur d'une chaine de caractère, len("ABC") == 3, len("F") == 1, len("176.E,4") == 7). On retrouve la fonction input mentionnée dans le chapitre précédent, rappellez vous qu'elle permet de récupérer ce que tape l'utilisateur sous forme de chaine de caractères. Tandis qu'avec for nous n'avons qu'à faire :

```
SaisieUtilisateur = input()
for Lettre in SaisieUtilisateur :
        print(Lettre)
```

Lettre va parcourir SaisieUtilisateur et nous l'afficherons à chaque fois, nous aurons ainsi un résultat identique mais beaucoup plus lisible et simple.

Les Mots Clés Break et Continue :

Il peut etre intéressant de mettre fin à la boucle en cours, pour cela vous pouvez utilisez le mot clé break :

```
Compteur = 0
while Compteur < 15 :
        if Compteur == 10 :
                break
        print(Compteur)
        Compteur += 1
```

Une fois que Compteur aura atteint la valeur 10 notre if va exécuter le break et la boucle s'achèvera. De meme on peut l'utiliser pour mettre fin à une boucle for. Exemple :

```
SaisieUtilisateur = input()
for Lettre in SaisieUtilisateur :
        if Lettre == "Q" :
                break
```

Ce code ne s'arretera donc que si la Saisie de l'Utilisateur contient un Q Majuscule. Enfin, le mot clé Continue permet d'interrompre un tour de Boucle pour passer directement au suivant :

```
Compteur = 0
while Compteur < 100 :
        Compteur += 1
        continue
        print("Bonjour")
```

Ce programme n'affichera donc rien puisque le print("Bonjour") est précédé du mot clé continue faisant éternellement reboucler le programme.

Les Fonctions

Introduction : Nous avons déjà vu quelques exemples de Fonctions dans notre apprentisage du Python, il est désormais temps de leur consacrer un chapitre. Les Fonctions vont nous permettre d'empaqueter du code pour le réutiliser à volonté. Nous n'aurons ainsi plus qu'à appeller la Fonction par son nom pour exécuter le code qu'elle renferme.

Définition d'une Fonction :

Pour définir une Fonction, nous faisons appel au mot clé : Def. Nous précisons ensuite le nom de notre Fonction et indiquerons, entre parenthèse si elle attend des paramètres. Les paramètres sont en fait des données sur lesquelles notre Fonction va pouvoir travailler, notre premier exemple n'en utilisera pas, nous laisserons donc ces parenthèses vides.

```
def MaFonction() :
     print("Bonjour, Ceci est ma première Fonction !")
```

Les règles de nommage d'une Fonction sont identiques à celles des Variables (Uniquement des lettres, des chiffres, le caractère underscore et le nom de notre Fonction ne peut commencer par un Chiffre) Nous n'avons ensuite plus qu'à appeler notre Fonction par son nom pour executer le code qu'elle renferme : Le message "Bonjour, Ceci est ma première Fonction !" s'affiche alors à l'écran. Naturellement, nous pouvons mettre beaucoup plus d'instructions dans le corps de notre Fonction que nous ne l'avons fait.

```
def MaFonction() :
        print("Cette fois ci c'est un peu plus compliqué !")
        Compteur = 0
        while Compteur <= len("ABCD") :
                print(Compteur)
                Compteur += 1
```

Si nous appelons notre Fonction nous verrons le message indiqué s'afficher ainsi que les nombres de 0 à 4. Une Fonction peut aussi travailler sur des Paramètres, des données qu'on lui passe et sur lesquelles elle va effectuer des traitements. Il suffit pour cela de les indiquer entre parenthèses, voici un exemple.

```
def Somme(NombreUn,NombreDeux) :
        print(NombreUn+NombreDeux)
```

On aura alors qu'à appeler notre Fonction en lui précisant la valeur de ces paramètres :

```
Somme(15.75,137)
```

Affichera 152.137. On peut définir autant de paramètres que l'on veut :

```
def FonctionEtrange(NombreUn,NombreDeux,ChaineDeCaracteres,VariableInutile) :
     print(NombreUn*NombreDeux,ChaineDeCaracteres*(NombreUn+NombreDeux)
     if NombreUn == NombreDeux :
           print("Zzzzzzz....")
```

Définit par exemple une fonction tout à fait valide Python permet aussi d'affecter une valeur à chaque paramètre dans le désordre, on utilise pour cela la syntaxe suivante :

```
Fonction(Paramètre = Valeur, Autre Paramètre = Autre Valeur) ...
```

On pourrait ici appeler notre Fonction de cette façon :

```
Somme(NombreDeux = 175, NombreUn = 13, "ABCD", 17)
```

Remarquez que si certains paramètres n'ont pas de valeur Python nous signale une erreur :

```
def Fonction(Entier, SecondEntier) :
     print(Entier/SecondEntier)
```

Si vous faites :

```
 Fonction(1)
```

Alors Python rencontre un problème, nous pouvons fixer une valeur par défaut à chaque paramètre qui ne sera modifiée que dans le cas ou une autre valeur est précisée lors de l'appel de la Fonction :

```
def Fonction(Entier, SecondEntier = 1) :
     print(Entier/SecondEntier)
```

Alors si vous exécutez :

```
Fonction(17)
```

Python affiche 17. Mais si vous faites :

```
Fonction(17,3)
```

Alors Python nous affiche 5.6666666666667...
Il se peut aussi que nous souhaitions récupérer une valeur à la fin de l'execution de la fonction. Par exemple si nous avons le programme suivant :

```
def Produit(NombreUn,NombreDeux,NombreTrois) :
     print(NombreUn*NombreDeux*NombreTrois)
Produit(17,15,6)
```

Il se pourrait que nous voulions récupérer la valeur du produit de ces trois nombres pour travailler dessus, or ici nous ne faisons que l'afficher. Nous pouvons pour cela utiliser le mot clé return.

```
def Produit(NombreUn,NombreDeux,NombreTrois) :
     return NombreUn*NombreDeux*NombreTrois
Resultat = Produit(17,15,6)
```

Ici Resultat contiendra le produit de nos trois nombres, nous pouvons bien sur garder notre print dans la définition de notre fonction. Si nous n'avions pas utilisé return il nous aurait été impossible de stocker le résultat de notre fonction dans une variable. Il est à noté que l'instruction return met fin à la fonction, le code qui pourrait éventuellement suivre cette instruction et se trouver dans le corps de la Fonction ne sera pas exécuté (Excepté dans un cas particulier que nous verrons à l'avenir). A savoir que nous pouvons tout à fait faire travailler une fonction sur le résultat d'une autre fonction, c'est d'ailleurs ce que nous faisons quand nous écrivons :

```
Entier = int(input())
```

Input renvoie une chaine de caractère qui est alors convertie en int.

Les Fonctions Lambdas :

Si nous utilisons une fonction particulièrement courte, une syntaxe existe pour la définir en une seule ligne :

```
Fonction = lambda X : X*(X+1)
```

On indique en premier lieu le ou les paramètres, on ajoute deux points puis on indique ce que la fonction renvoie.

Les Classes

Introduction : Nous entrons avec ce chapitre dans le monde de la POO, La Programmation Orientée Objet qui nous permettra d'envisager l'écriture de nos programmes et la logique qui les sous tends d'une façon nouvelle. En effet, nous allons découvrir une nouvelle méthode de Programmation, un nouveau Paradigme qui nous permettra d'organiser notre code d'une manière plus logique et cohérente.

Programmation Orientée Objet :

Pour présenter le paradigme de la Programmation Orientée Objet prenons l'exemple suivant, vous souhaitez modéliser un Personnage caractérisé par sa Taille, son Age et son Sexe. Jusqu'ici nous aurions sans doute obtenu quelque chose de ce genre ci :

```
Nom = ""
Sexe = ""
Age = 0
```

Très bien, supposons maintenant que vous souhaitiez ajouter une seconde Personne. Vous procéderez probablement de cette manière :

```
Nom = ""
Nom2 = ""
Sexe = ""
Sexe2 = ""
Age = 0
Age2 = 0
```

Si vous devez gérer un univers dans lequel 10 000 De ces Personnages évoluent le moins que l'on puisse dire c'est que cela vous sera très pénible ! Pourtant, nous avons pu dégager des caractéristiques communes à chacun de ces personnages, tous sont décrits par un Nom, un Sexe et un Age. L'Idée est donc de créer un modèle, un moule en quelque sortes qui nous permette de modéliser une personne, nous n'aurons plus qu'à utiliser ce moule à chaque fois que nous voudrons créer un Personnage. Passons à la pratique !

```
class Personne :
     def __init__(self) :
           self.Nom = ""
            self.Sexe = ""
            self.Age = 0
```

Bien, la première ligne que nous trouvons : "class Personne" Indique nous allons créer une classe, un modèle de Personne. Viens ensuite une ligne quelque peu déroutante : "def __init__(self) :" Nous reconnaissons ici la syntaxe de la définition d'une fonction Cette fonction c'est le constructeur de notre classe, elle va etre appelée quand nous allons vouloir créer une Variable modélisant une Personne. Elle porte toujours le nom __init__, Par ailleurs elle prend un paramètre "self" Celui ci correspond en quelque sorte à l'objet que nous sommes en train de créer.
Viennent ensuite trois lignes :

```
self.Nom = ""
self.Sexe = ""
self.Age = 0
```

Rappelez vous de ce que nous disions il y a quelques chapitres, le "." marque l'appartenance quand nous écrivons self.Nom nous désignons le Nom de l'Objet que nous créons.

Pour créer une Variable (En Programmation Orientée Objet on Préfère le terme, d'Objet...) du type Personne il suffit de procéder ainsi :

```
PremierePersonne = Personne()
```

Cette syntaxe nous permet d'ailleurs de définir des Variables des types déjà connus de cette façon :

```
Entier = int()
Chaine = str()
NombreDecimal = float()
```

Et oui, rappelez vous lorsque nous faisions :

```
print(type(Entier))
```

Nous voyions 'class <int>' s'afficher, les Entier, les Nombres Décimaux, les Chaines de Caractères sont en fait construits sur des classes !

Bien, Nous avons défini notre premier objet ! Malheureusement son Nom, son Sexe et son Age sont encore indéfinis, remédions à cela en améliorant un peu notre constructeur !

```
class Personne :
        def __init__(self,Nom,Sexe,Age) :
                self.Nom = Nom
                self.Sexe = Sexe
                self.Age = Age
```

Désormais si nous faisons :

```
PremierePersonne = Personne("Victor Hugo","Masculin","37")
print(PremierePersonne.Nom,PremierePersonne.Sexe,PremierePersonne.Age)
```

Nous voyons s'afficher "Victor Hugo Masculin 37" Nous pouvons bien sur modifier ces données, il nous suffit pour cela de leur affecter une nouvelle valeur. Notez d'ailleurs que les Variables d'un Objet sont appelées ses Attributs en Programmation Orientée Objet. Désormais que nous avons vu que nous pouvions définir les attributs d'un Objet, intéressons nous à ses méthodes. Les méthodes d'un objet ne sont rien d'autres que des Fonctions propre à cet objet et qui peuvent éventuellement agir sur ses attributs. Prenons l'exemple d'une classe destinée à modélisée des Voitures :

```
class Voiture :
        def __init__(self,VitesseMaximale,Longueur,Largeur,Couleur) :
                self.VitesseMaximale = VitesseMaximale
                self.Longueur = Longueur
                self.Largeur = Largeur
                self.Couleur = Couleur
                self.VitesseActuelle = 0
```

Au moment de sa création, la Vitesse Actuelle de notre véhicule est nulle mais il se pourrait qu'elle augmente à l'avenir, que la Voiture accélère.
Pour permettre cela créons une fonction Accélérer :

```
class Voiture :
        def __init__(self,VitesseMaximale,Longueur,Largeur,Couleur) :
                self.VitesseMaximale = VitesseMaximale
                self.Longueur = Longueur
                self.Largeur = Largeur
                self.Couleur = Couleur
                self.VitesseActuelle = 0
        def Accelerer(self,NouvelleVitesse)
                self.VitesseActuelle = NouvelleVitesse
```

Bien sur nous aurions obtenu un résultat similaire en modifiant directement la valeur de l'attribut VitesseActuelle de notre Objet lors de l'excution de notre programme. Malgré tout, nous pouvons ainsi entrevoir un mécanisme qui nous permettra de définir des fonctions bien plus complexes. Ces fonctions définies à l'intérieur de classes sont d'ailleurs appelées des Méthodes nous utiliserons ce nouveau Vocabulaire très fréquemment désormais.

Dict et Dir :

Lorsque vous créez un Objet, Python vous permet d'en explorer tous les attributs au moyen du dictionnaire __dict__.

```
class Animal :
        def __init__(self,Espece,Taille,Couleur) :
                self.Espece = Espece
                self.Taille = Taille
                self.Couleur = Couleur
Perroquet = Animal("Perroquet",1,"Rouge")
print(Perroquet.__dict__)
```

A l'exécution ce code vous affichera : {'Espece': 'Perroquet', 'Taille': 1, 'Couleur': 'Rouge'}Vous pouvez naturellement accéder à la valeur d'un attribut en particulier avec Perroquet.__dict__[Attribut] De meme que vous pouvez le modifier. Python vous offre une fonction parfois fort utile pour lister les noms de tous les attributs et méthodes de votre Objet, La Fonction Dir.

```
dir(Perroquet)
```

Affichera :

```
['Couleur', 'Espece', 'Taille', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
```

Vous voyez ici des attributs et des méthodes que vous n'avez jamais définies, sachez que Python les crées automatiquement et qu'elles lui sont utiles, en tant que programmeurs nous n'avons pas à nous en soucier ;).