Chapitre 2

Introduction à la programmation

Chapitre 2

Introduction à la programmation

I. Présentation.

I.1/ Le langage de programmation.

- **Qu'est ce qu'un programme ?** Un programme est une suite d'instructions compréhensibles par l'ordinateur réalisé les unes après les autres. A l'état brut l'ordinateur ne comprend que très peu d'instructions : écrire/lire dans une case mémoire, effectuer une addition, multiplication, une comparaison.
- **Langage de haut niveau vs bas niveau.** Le langage de programmation est un programme qui apprend à l'ordinateur des instructions plus élaborées. On distingue 2 types de langages : les langages de bas niveau et les langages de haut niveau. Les instructions supplémentaires sont peu nombreuses dans un langage de bas niveau et très proches de ce que comprend le microprocesseur. De plus le programmeur doit tenir compte des particularités de sa machine lors de la programmation comme la taille du bus, le nombre de registres... Inversement dans un langage de haut niveau, les instructions sont nombreuses et proches de ce que comprend un être humain. Le langage est aussi indépendant de la machine sur laquelle on programme. Exemple :
  - Bas niveau : assembleur
  - Haut niveau : C, C++, php, lisp, basic, python, html, ...
- **Langage compilé vs interprété.** Ensuite il y a les langages interprétés et les langages compilés. Les nouvelles instructions contenues dans le langage de programmation doivent être converties en instructions simples et compréhensibles par le microprocesseur. Cette conversion se fait soit avant l'exécution, on parle de langage compilé, soit elle se fait au moment de l'exécution, c'est un langage interprété. Les langages interprétés sont plus souples dans la programmation, les langages compilés sont plus rapides lors de l'exécution.

I.2/ Présentation de Python.

Caractéristiques de Python :

- C'est un langage de haut niveau.
- Il est partiellement compilé, partiellement interprété pour minimiser les désavantages des 2 systèmes.
- Il est gratuit.
- Il est portable, c'est-à-dire qu'il ne dépend pas du système d'exploitation choisi.
- Il est orienté objet (Cf. plus loin)

On a 2 modes d'exécution : Ligne de commande, fichier code

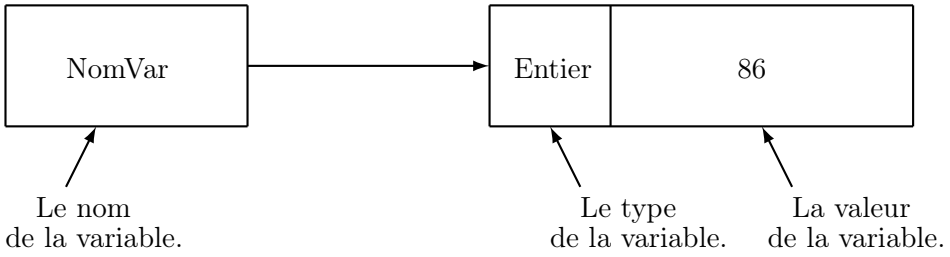
I.3/ Les erreurs.

Lorsqu'on programme, on est confronté à 3 types d'erreurs :

- Les erreurs de syntaxe : l'ordinateur ne comprend pas ce qui est écrit.
- Les erreurs d'exécution : l'ordinateur comprend ce qui est écrit, mais ne peut le réaliser (division par 0, par exemple)
- Les erreurs de sens : l'ordinateur ne réalise pas ce qui est prévu.

I.4/ Les variables.

- Dans un langage de haut niveau, on ne manipule pas les adresses mémoires directement. On a recours à des variables. Ainsi, on ne risque pas d'écrire à des endroits de la mémoire où se trouvent des données à ne pas effacer.
- Une variable est un pointeur sur une case mémoire. On la représente par :



  Ainsi une variable est définie par son nom, son type et sa valeur.
- Le nom d'une variable doit commencer par une lettre ou le caractère '_' et ne contenir que des lettres, des chiffres et le caractère '_'. Attention : on doit donner aux variables des noms significatifs. Par exemple utiliser des noms de variables à 1 caractère nuit à la lisibilité.
- Une affectation sert à modifier le type et la valeur d'une variable. La syntaxe est :

$$NomVar = Expression$$

- Exemples. Dans les exemples suivants, déterminer les nom, type et valeur des variables après affectation.

ex 1	ex 2	ex 3	ex 4	ex 5
$a = 86$	$a = 86.3$	$a =$ "$Bonjour$"	$a = 2$ $a = a \times 3 + 2$	$a =$ "Bon" $b =$ "$jour$" $c = a + b$

- Opérations :
  1) +, *, /, − sur les entiers/floats.
  2) ** sur les entiers/floats : puissance.
  3) %, // sur les entiers : reste et quotient de la division euclidienne.
  4) + sur les chaines de caractères : concaténation.
  5) >=, >, <=, < sur les entiers/floats/chaines. Renvoie un booléan.
  6) ==, != sur tous les types : égalité/différent. Renvoie un booléan.

- Plusieurs affectations en une ligne :
  » x=y=7
  » a,b=7,8

I.5/ Interaction avec l'utilisateur.

- **Afficher une information.** Syntaxe :

$$print(Expression)$$

  A noter que $\backslash n$ est un caractère spécial qui indique à print de sauter une ligne.

- **Demander une information.** Syntaxe :

$$input(Expression)$$

  L'ordinateur écrit Expression comme pour *print*, mais attend que l'utilisateur saisisse une chaine de caractères. Cette chaine est renvoyée par input. Attention l'instruction *input* renvoie toujours une chaine de caractère !

Exercice.[1] Écrire un programme qui demande à l'utilisateur son nom et affiche 'Bonjour' suivi de son nom.

I.6/ Le commentaire.

Essentiel ! Le commentaire permet d'insérer dans la source du programme des lignes sans effet sur le programme mais qui expliquent son fonctionnement. Les commentaires permettent à un autre programmeur (voire au même programmeur après un certain temps) de modifier le code afin d'éliminer un bug ou d'améliorer le programme. Syntaxe :

```
# Commentaires
```

Exemple.

```
# Demande la température en degré celsius.
strTempCel = input("Quelle est la température en degré Celsius ?")

# On change le type de la variable strTempCel en float
floTempCel = int(strTempCel)

# On convertit en degré Fahrenheit.
floTempFar = floTempCel*1.8 + 32

# On change le type de la variable FloTempFar en chaine de caractères
strTempFar = str(floTempFar)

# On affiche le résultat.
print("La température est de " + strTempFar + " degré Fahrenheit")
```

II. Structure principale de programmation.

Un programme est donc une suite d'instructions. Le langage de programmation lit la première instruction l'exécute et passe à la suivante. Il parcourt ainsi toutes les instructions de haut en bas. Dans certains cas cependant la lecture n'est plus linéaire :

II.1/ Instruction IF.

Syntaxe.

if condition **:**

Bloc 1

else :

Bloc 2

ou

if condition **:**

Bloc 1

Le langage de programmation teste la condition. Si la condition est réalisée, il exécute le bloc 1. Si elle n'est pas réalisée et que le bloc 2 est présent, il exécute le bloc 2.

Format d'un bloc. Un bloc commence toujours par le symbole : . De plus toutes les lignes du bloc sont indentées c'est-à-dire qu'elle ont un décalage sur la droite par rapport à la position du if. Ce décalage est réalisé grâce à une tabulation. Le bloc s'arrête lorsque l'indentation s'arrête.

Format d'une condition. Une condition simple est une suite de tests du type :

Expression 1	==	Expression 2	pour tester l'égalité
Expression 1	!=	Expression 2	pour tester la différence
Expression 1	<=	Expression 2	pour tester l'infériorité
Expression 1	>=	Expression 2	pour tester la supériorité
Expression 1	<	Expression 2	pour tester l'infériorité stricte
Expression 1	>	Expression 2	pour tester la supériorité stricte

séparés par des AND ou des OR. On complètera plus tard avec d'autres sortes de conditions.

Exercice.[2] Écrire un programme qui demande à l'utilisateur un nombre entier et l'ordinateur écrit s'il est pair ou impair.

II.2/ Instruction WHILE.

Syntaxe.

while condition **:**

Le langage de programmation exécute le bloc tant que la condition est réalisée. Attention, le programme peut ne pas s'arrêter !

Exercice.[3] Écrire un programme qui demande à l'utilisateur un nombre tant que celui-ci n'est pas compris entre 1 et 6.

II.3/ Instruction FOR.

Syntaxe.

for variable **in range**(deb, fin, inc) **:**

Le langage de programmation exécute le bloc pour différentes valeurs de *variable* :

variable = deb
variable = deb+inc
variable = deb+2*inc
...

C'est à dire pour toutes les valeurs deb+k*inc avec deb+k*inc<fin. A noter l'inégalité stricte !

Remarques.

1. Si range n'a que deux paramètres, ce sont les valeurs de *deb* et *fin* . On a alors *inc*=1 par défaut.
2. Si range n'a qu'un paramètre, c'est la valeurs de *fin* et *deb*=0, *inc*=1 .

Exercice.[4] Écrire un programme affichant les entiers naturels de 1 à 100.

Exercice.[5] Écrire un programme affichant les carrés parfaits de 1 à 100.

Exercice.[6] Écrire un programme affichant les lettres de 'A' à 'Z'. On pourra utiliser les instructions *Ord(...)* et *chr(...)* après avoir cherché ce qu'elles font.

III. Les fonctions.

III.1/ Comment définir une fonction ?

- On utilise une fonction pour réutiliser facilement un morceau de code.
- Les valeurs dont a besoin le code pour fonctionner sont appelées les paramètres. Les paramètres ne doivent pas être modifiés dans la fonction.
- Les informations retournées par la fonction le sont grâce à l'instruction *return*. Ces valeurs sont récupérées comme les variables.

Syntaxe.

def nomFct(Param1, Param2,...) **:**

Bloc

Exercice.[7] Écrire une fonction qui à partir d'un paramètre x renvoie la valeur x^2.

Exercice.[8]

1. Écrire une fonction *afficheCar(Car, Nb)* qui affiche le caractère *Car*, *Nb* fois sans retour à la ligne.
2. Écrire une fonction *carré(Nb)* qui affiche un carré formé d'étoiles puis une fonction *triangleRectangle(Nb)* qui affiche un triangle rectangle dont les côtés de l'angle droit ont *Nb* étoiles. Voici ce que doit donner *carré(5)* et *triangleRectangle(5)*

```
* * * * *          *
*       *          * *
*       *          * * *
*       *          * * * *
* * * * *          * * * * *
```

III.2/ Variables locales/globales.

1. Toutes les variables crées dans une fonction sont des variables utilisables uniquement dans cette fonction. On dit que la variable est locale.
2. Par opposition, les variables crées en dehors de toute fonction sont appelées des variables globales.
3. Les variables globales peuvent être utilisées et modifiées dans les fonctions à condition de le signaler au langage de programmation avec l'instruction *global NomVar* où *NomVar* est le nom de la variable globale.

Exemple. Qu'affiche le code suivant ?

```
def MaFct() :
    x=1
x=0
MaFct()
print(x)
```

Que faut-il ajouter pour que le programme affiche 1 ?

III.3/ Paramètre par défaut.

Certains paramètres d'une fonction peuvent être optionnels. Pour cela, il suffit d'ajouter *"=valeur"* après le nom du paramètre, où *valeur* est la valeur par défaut du paramètre c'est-à-dire la valeur que va prendre la variable si ce paramètre est omis dans l'appel de la fonction. Attention les paramètres optionnels doivent être après les paramètres non optionnels.

Exemple. Voici une fonction qui calcule la racine $n^{\text{ième}}$ d'un réel. Par défaut il s'agit de la racine carrée.

```
def racine(Nombre, N=2) :
    return(Nombre**(1/N))
>>> racine(16)
    4.0
>>> racine(16, 4)
    2.0
```

III.4/ Modifier les paramètres dans la fonction : prend garde !

Si vous décidez de modifiez les paramètres d'une fonction, cette modification sera parfois gardée à la sortie de la fonction, parfois non (ça dépend du caractère mutable du paramètre... Voire plus loin... ou pas...) C'est pour cela que je vous incite à ne **jamais** modifier les paramètres !

IV. Autres types de données.

IV.1/ L'objet string.

L'objet *string* représente une suite de caractères. Le premier caractère est le caractère numéro 0 Si Txt est une variable de type chaine de caractères, voici quelques informations pour la manipuler.

Initialiser	Txt = "....." Txt = '.....'
Accéder à un caractère	Txt[i] pour accéder au i^{ème} caractère Txt[$-i$] pour accéder au i^{ème} caractère en comptant de la fin
Accéder à une sous chaine	Txt[$i : j$] donne la chaine contenant les caractères numérotés de i à $j-1$. Les valeurs de i et j peuvent être négatives. La position est alors comptée à partir de la fin. Si la valeur de i est omise, la sélection commence au début de la chaine. Si la valeur de j est omise, la sélection finit à la fin de la chaine.
Longueur	len(Txt) donne le nombre de caractères de Txt.
Concaténer *Ch* et *Ch2*	Ch + Ch2
Contient une chaine *Ch*	Ch in Txt
Mettre en majuscule Mettre en minuscule	Txt.upper() Txt.lower()
Remplacer les occurrences de *Ch* par *Ch2*	Txt.remplace(Ch, Ch2)
Convertir en entier/float	int(Txt), float(Txt)
Conversation Caractère / Valeur ASCII. Txt doit être de longueur 1	ValEntier = Ord(Txt), Txt=chr(ValEntier)

Exemple. Si MaChaine = "abcdefg"
MaChaine[2] donne "c"
MaChaine[-2] donne "f"
MaChaine[:-1] donne "abcdef"

Attention. Si on veut modifier le i^{ième} caractère de la chaine *Ch*, l'instruction :
Ch[i] = "*"
ne fonctionne pas car les chaines de caractères en Python ne peuvent pas être modifiées après leur création. On dit qu'elles ne sont pas mutables. Pour changer un caractère, on peut faire :
Ch = Ch[:i] + "*" + Ch[i+1 :]

Exercice.[9] Écrire un programme demandant le nom de l'utilisateur puis écrit son nom ou la 1ère lettre sera en majuscule et les autres en minuscule.

Exercice.[10] Écrire un programme demandant le nom de l'utilisateur tant que celui-ci ne contient pas que des lettres.

Exercice.[11] Écrire une fonction qui prend en paramètre une chaine de caractères et une lettre, et donne le nombre d'occurrences de cette lettre dans la chaine de caractères.

Exercice.[12] Écrire une fonction qui prend en paramètre une liste et retourne le plus grand élément de cette liste.

IV.2/ L'objet liste.

Une liste est une suite d'objets python accessibles par leur numéro. Les listes se manipulent pratiquement comme les chaines de caractères. ? Ainsi, si Li est une liste :

Initialiser	Li = [..., ..., ...]
Accéder à un élément	Li[i] pour accéder au i$^{\text{ème}}$ élément Li[$-i$] pour accéder au i$^{\text{ème}}$ élément en comptant de la fin
Accéder à une sous liste	Li[$i : j$] donne la liste contenant les éléments numérotés de i à $j-1$. Les valeurs de i et j peuvent être négatives. La position est alors comptée à partir de la fin. Si la valeur de i est omise, la sélection commence au début de la chaine. Si la valeur de j est omise, la sélection finit à la fin de la chaine.
Longueur	len(Li) donne le nombre d'éléments de Li.
Concaténer Li et $Li2$	Li + Li2
Contient l'élément Elt ?	Elt in Li
Ajouter un élément en fin de liste	Li.append(Elt)
Enlever le premier élément ayant la valeur val	Li.remove(val)
Enlever et renvoyer l'élément en position i	Li.pop(i)

Remarques.

1. L'instruction L[i]=... fonctionne avec les listes, car les listes sont mutables. En contrepartie :
2. L'instruction Li = Li2 ne copie pas la liste Li2 dans la liste Li. Après cette instruction Li et $Li2$ sont les mêmes listes ! Si vous modifiez une liste, vous modifiez l'autre. Si ce n'est pas ce que vous voulez, essayez :

```
Li = Li2[:]
```

Exercice.[13] Écrire une fonction qui prend en paramètre une liste et retourne la somme des éléments de cette liste.

Exercice.[14] Écrire une fonction qui prend en paramètre un entier naturel non nul et retourne la liste de ses diviseurs.

IV.3/ L'objet tuple.

Les tuples sont l'équivalent des n_uplets en mathématique. Ils se manipulent comme les listes sauf qu'ils sont non mutables ce qui amène d'autres différences : ainsi, pour un tuple *Tu*,

1. Initialisation : Tu = (...)
2. Tu[i]=... est une instruction incorrect
3. *append* et *remove* ne fonctionnent pas sur les tuples.

IV.4/ Les dictionnaires.

Les dictionnaires sont une sorte de liste où l'on accède pas aux éléments avec des numéros mais avec tout objet non mutable en Python : n_uplet, chaine, entier, float. C'est l'équivalent en mathématique des familles. Ils se manipulent aussi comme les listes.

Exemple.

```
>>> Adresses = {'Julie' : '20bis rue D. Claude', 'Robert' : '11 av, des tilleul'}
>>> Adresses['Julie']
20bis rue D. Claude
>>> Adresses['Julie']='47 rue des grenouilles grises'
>>> Adresses['Julie']
47 rue des grenouilles grises
>>> len(Adresses)
2
```

IV.5/ Les boucles FOR sous un autres angle.

Syntaxe.

for variable **in** Liste **:**

Le variable prend successivement chacune des valeurs de la liste. Cela fonctionne encore en remplaçant la liste par une chaine de caractères, un tuple, un dictionnaire. A noter qu'avec un dictionnaire la variable parcourt l'ensemble des clés.

V. Introduction aux classes.

V.1/ Méthodes, attributs.

Définition. Une classe est un regroupement

1) d'attributs qui permettent de décrire l'objet,
2) de méthodes qui permettent de manipuler les attributs.

Ce regroupement (on parle d'encapsulation en informatique) se fait sous un thème commun.

Exemple. Si l'on veut créer une classe *Machine à café*, on peut avoir

1) Attributs : quantité de café/sucre/crème/chocolat/gobelet en stock, pièces présentes ...
2) Méthode : servir une boisson, rendre la monnaie, ...

Remarque. Attention, une classe définit comment sont fait les objets, c'est une sorte de moule. Cela s'apparente à un type comme integer/float/string/list/tuple/... Ensuite, il faut créer réellement un objet ayant ce profil. On parle d'instance de la classe.

Dans notre exemple plus haut, la classe *Machine à café* représente comment fonctionnent en général les machines à café. Et la machine à café de la salle des profs est une instance de cette classe.

V.2/ Création et manipulations simples de la classe

- **Créer une classe :**

```
class NomDeLaClasse :
    """ Description de la classe et des attributs """
```

Description de la classe

Moralement, on crée un nouveau type possible pour les variables.

- **Créer une instance de la classe :**

```
InstanceClasse = NomDeLaClasse()
```

Maintenant *InstanceClasse* est une variable de type *NomDeLaClasse*.

- **Accéder à un attribut ou une méthode de la classe :**

```
InstanceClasse.Méthode(...)
InstanceClasse.Attribut
```

- **Définir une méthode.** A l'intérieur de la description de la classe, on écrit :

  def NomMethode(self, Arg1, Arg2, ...) :

  Code de la méthode

  self est un argument obligatoire qui a pour valeur l'instance de la classe auquel on applique la méthode. Ainsi, pour accéder à un attribut à l'intérieur de la méthode, on écrit *self.NomAttribut*. Par contre, cet argument n'est jamais mis quand on appelle la méthode. Ainsi si on appelle cette méthode, il faut écrire *InstanceClasse.NomMethode(Arg1, Arg2, ...)* (Pas de self).

- **Infos sur les attributs.** Les attributs de la classe peuvent être utilisés/modifiés dans toutes les méthodes de la classe. La portée de ces variables est plus grande que celle des variables locales et plus petite que celle les variables globales.
  De plus, comme pour les variables classiques, pour les définir il suffit de les affecter à une valeur. C'est pour cela qu'on écrit leur liste dans la description afin d'augmenter la lisibilité du programme.

V.3/ Les méthodes spéciales.

Certaines méthodes ne sont pas faites pour être appelées directement. Elles sont appelées automatiquement lors de certains événements. Le nom de ces méthodes est **toujours** encadré par des doubles underscore. Les principales méthodes spéciales sont listées ci-dessous. On suppose que le nom de la classe est NomC et que mon instance est NomI.

Nom de la méthode	**Quand est-elle appelée ?**
__repr__(self)	print(NomI).
__init__(self, P_1,...)	NomI = NomC(P_1,...)
__del__(self)	On détruit NomI.
__getitem__(self, item)	NomI[item]
__setitem__(self, item)	NomI[item]=...
__len__(self)	len(NomI)

Nom de la méthode	**Quand est-elle appelée ?**
__eq__(self, obj) et __ne__(self, obj)	NomI == obj et NomI != obj
__gt__(self, obj) et __ge__(self, obj)	NomI > obj et NomI >= obj
__lt__(self, obj) et __le__(self, obj)	NomI < obj et NomI <= obj
__add__(self, obj)	NomI + obj
__radd__(self, obj)	obj + NomI et obj.__add__ n'a pas abouti.
__mul__(self, obj)	obj * NomI
__rmul__(self, obj)	obj * NomI et obj.__mul__ n'a pas abouti
__sub__(self, obj)	obj – NomI
__pow__(self, obj)	NomI ** obj
__truediv__(self, obj)	NomI / obj
__floordiv__(self, obj)	NomI // obj
__mod__(self, obj)	NomI % obj

V.4/ Notre première classe.

Écrire une classe *polynôme* qui devra prendre en charge la gestion par la machine des polynômes. Cette classe ne comprendra qu'un seul attribut, une liste *Coeff* qui contiendra les coefficients du polynôme (Le coefficient de X^k sera en position k dans cette liste). Les coefficients de types float et entier devront être gérés par la classe. Le programmeur pourra également s'il le souhaite gérer les coefficients de type rationnel avec la classe *Fraction* vue en cours. Cette classe *polynôme* devra entre autres :

1. gérer l'initialisation du polynôme grâce à la méthode *__init__()*
2. afficher proprement le polynôme. On écrira la méthode *__repr__()*
3. donner le degré et la valuation du polynôme. Dans le cas du polynôme nul, on renverra une chaine de caractère contenant '+oo' ou '-oo'
4. pouvoir comparer deux polynômes c'est-à-dire savoir si deux polynômes sont égaux. On écrira les méthodes *__eq__()* et *__ne__()*. On améliorera ensuite ces méthodes pour pouvoir comparer un polynôme à un float/fraction/Entier. Dans ce cas, ceux-ci seront considérées comme des

polynômes constants.

5. pouvoir donner un coefficient du polynômes en faisant *NomDuPolynôme[i]*. On écrira la méthode *__getitem__()*.
6. effectuer la somme/soustraction de deux polynômes. On écrira les méthodes *__add__()* et *__sub__()*. Comme précédemment les floats/fraction/entiers seront considérés comme des polynômes constants. Dans ce cas on écrira les méthodes *__radd__()* et *__rsub__()*
7. effectuer le produit de deux polynômes. On écrira les méthodes *__mul__()* et *__rmul__()*.
8. donner le reste et le quotient de la division euclidienne de deux polynômes. On écrira les méthodes *__floordiv__()* et *__mod__()*.
9. donner le pgcd et le ppcm de deux polynômes
10. Et j'en passe.