Cours Algorithmique SNT/NSI

S. GIBAUD

2019-2020

1 Variables

Définition 1.1. Une variable est caractérisée par :

- **Un identificateur** : Il peut contenir des lettres, chiffres, _ (underscore) mais il ne peut pas commencer par un chiffre. Minuscule et Majuscule sont différenciées. C'est le nom d'une case mémoire. Il est donc unique.
- **Un type** : C'est une information sur le contenu de la variable qui indique la manière de manipuler la variable.

Définition 1.2 (Type (en python))**.**

- *int* : nombre entier : -1 , 2 , 0 , 18021990
- *float* : nombre décimal : 1.2 3.4 2.133 18.021990. On adopte les notations anglo-saxonnes : On ne met pas de virgule dans les nombres mais des points.
- *boolean* : C'est un type Vrai Faux : True, False, $1 > 3$, $2<2.1$ $1==0$, $1 =! 2$
- *string* : C'est une chaîne de caractères. Ca sert à faire du texte. "Kikou", "Choupette lâche le stylo".
  NB : Un string se comporte comme une chaine par exemple. str = "Kikou". str[0] est K, str[1] est i
- *list* : C'est un ensemble de données ordonnées.
  - liste = [3, 2.1, "babar", "Gibaud"]
  - liste[0] est 3
  - liste[2] est "babar"
- *dictionnaire* : Un dictionnaire est une liste de couples nom :définition. Le nom s'appelle la clé, la définition s'appelle une valeur.
  - dict = { "a" : 1, "b" :2, "Gibaud" : "Meilleur prof du monde" }
  - dict["b"] est 2

Définition 1.3 (Affectation)**.** Quand l'on veut mettre une valeur dans une variable on dit que l'on *affecte* une valeur à une variable. On l'écrit en python via = et sur papier $\leftarrow$.

Exemple.
A $\leftarrow$ 2 : On met dans la variable A, la valeur 2. Le type est int.
B = 3.1 : On met dans la variable B, la valeur 3.1 . Le type est float.
dicop = { "a" :1, "k" :7 } : On met dans la variable dicop la valeur { "a" :1, "k" :7 }. Cette valeur est de type dictionnaire.
C = 3, C = C+1. On met dans la variable C la valeur 3. Puis on met dans la variable C la valeur de C + 1. Donc on a mis 4 dans C.

Exercice (A refaire avec légère modif 5 fois). Affecter dans des variables votre nom, prénom, jour de naissance et dans un dictionnaire ces informations avec comme clé "nom", "prénom", "jour de naissance".
Donner les types de ces variables. Puis afficher ces variables grâce à la commande print()

Important : Connaître le type des variables que l'on utilise est très important pour pouvoir coder sans trop d'erreur.

Exercice. Pour les élèves de NSI. Affecter à A une valeur, puis affecter à B la valeur A. Modifier la valeur de B. Afficher la valeur de A. Que remarquez vous ?

Pour éviter que des valeurs soient couplés, on peut utiliser la commande deepcopy (voir google)

2 Instructions Conditionnelles :Si ..... Alors ....

2.1 Booléen et Conditions

Un booléen est une valeur vrai ou faux. On génère des booléens simples de 3 façons différentes.

- A = True, A = False. On affecte à A directement le booléen Vrai ou Faux.
- 1<2 , 4>5, 1<=1, 3>=2. On teste directement une inéquation (que veut dire la troisième inéquation). En exercice, quelles sont les inéquations vraies ou fausses.
- 1 == 1, 2 =!2, A ==3 , "bla" == "bla" : On peut tester des égalités (comme le simple = sert à l'affectation, on utilise le double == pour tester une égalité). Que veut dire :! ?

Pour faire des conditions plus compliquées que celle précédentes on peut soit utiliser des fonctions (comme *isdir* qui vérifie si un objet est un dossier) ou utiliser les opérateurs logiques ET OU.

Voici les tables de vérités de ET et OU.

A	B	A et B	A ou B	Un seul Vrai	2 Vrais
Vrai	Vrai	Vrai	Vrai	Oui	Oui
Vrai	Faux	Faux	Vrai	Oui	Non
Faux	Vrai	Faux	Vrai	Oui	Non
Faux	Faux	Faux	Faux	Non	Non

2.2 Si ... Alors ...

Principe. Un père dit à son enfant : "**Si** tu finis tes légumes, **Alors** tu auras un dessert". L'enfant ne finit pas ses légumes mais le père donne quand même à son enfant un dessert. Le père a-t-il menti ?

La réponse est non. Et ce n'est pas pour des raisons d'éducation ou de morale. Le père a dit ce qu'il ferait si son enfant finit son dessert mais il n'a rien dit si il ne finissait pas son dessert.
Le *If* informatique fonctionne de la même façon. Lorsque l'on écrit :

$$If \ldots \text{Condition} \ldots \quad Then \ldots \text{Instruction} \ldots$$

L'ordinateur vérifie la condition, si cette condition est vérifiée alors l'ordinateur effectue l'instruction sinon rien ne se passe. On peut aussi utiliser des instructions *Sinon* et *Sinon si*.

En python, les conditions et les instructions sont séparés par des " :" et un quadruple espace appelé **Indentation**.

FIGURE 1 – Code Python avec un Si (en exercice que ce passe-t-il ?)

```
if "Gibaud" == "Prof":
    print("Alors je m'affiche ??")
```

FIGURE 2 – Code Python avec un Si, Sinon (en exercice que ce passe-t-il ?)

```
if "Gibaud" == "Prof":
    print("Alors je m'affiche ?")
else:
    print("En fait je m'affichais pas")
```

FIGURE 3 – Code Python avec un Si, Sinon Si et Sinon (en exercice que ce passe-t-il ?)

```
if "Gibaud" == "Prof":
    print("Alors je m'affiche ?")
elif "Gibaud" == "Gibaud":
    print("Mais oui je testais une chaine...")
else:
    print("En fait je m'affichais pas")
```

Exercice.

1. Faire un code en python qui affiche "Quatre" lorsque le nombre A est un multiple de 4. Affecter à A 2,4,8,17, 18021990. Le programme marche-t-il ? (Oui ⇒ Cool, Non ⇒ réparer le !)
2. Faire un code en python qui affiche "Quatres" lorsque le nombre A est un multiple de 5 et qui affiche "Pas Cinq" Sinon. Affecter à A 2,4,8,17, 18021990. Le programme marche-t-il ? (Oui ⇒ Cool, Non ⇒ réparer le !)

3. Faire un code en python qui affiche "Quatre" lorsque le nombre A est un multiple de 4. Sinon si A est un multiple de 5 il affiche "Cinq". Sinon afficher "Ben c'est un nombre".

3 Boucles

3.1 Boucle conditionnelle : Tant Que

Une boucle **Tant Que** :

While Condition :*Instruction*

fonctionne de la manière suivante : Tant que la condition est vérifiée on applique l'instruction.
Le problème est que l'on peut avoir des boucles infinies. Que fait la boucle suivante :

FIGURE 4 – Boucle infinie

```
while 1 == 1:
    print("Je suis une boucle infinie")
```

De même que fait la boucle suivante :

FIGURE 5 – Exemple de Boucle

```
i = 0
while i < 5:
    print("i est plus petit que 5")
    print("i est égal à {}".format(i))
    i = i + 1
```

Exercice.

1. En utilisant le package random (taper *import random*) et la commande random.randint(a,b) qui tire un nombre au hasard en a et b. Faites une boucle conditionnelle tirant au hasard un nombre entre 0 et 50 jusqu'à être le numéro 18. On affiche le numéro dans la boucle. (Pour les NSI, cette boucle est elle infini et pourquoi ?)
2. Faire une boucle qui tire des noms au hasard jusqu'à avoir un nombre pair.
3. Faire une boucle où le nombre augmente de 3 en 3 jusqu'à dépasser le numéro 1802.

3.2 Boucle de taille définie : Pour

Une boucle **Pour** :

For compteur *in* Ensemble : Instruction

fonctionne de la manière suivante : On crée une variable compteur (que l'on appellera ici i). Cette variable a pour valeur la première valeur de l'ensemble. Ensuite on réalise l'instruction. Puis on incrémente le compteur (on l'augmente).

Que fait la boucle suivante ($range(0, 10)$ est l'ensemble des nombres de 0 à 10) ?

FIGURE 6 – Boucle for simple

```
for i in range(0,10):
    print(i)
```

En utilisant la figure suivante :

FIGURE 7 – Répétition d'un texte

```
>>> "Gibaud "*2
'Gibaud Gibaud '
```

que fait la boucle suivante ?

FIGURE 8 – Boucle J'aime l'informatique

```
for i in range(0,13):
    print("j'aime l'informatique  "*i)
```

et que fait celle ci ?

FIGURE 9 – Boucle avec une liste

```
L = ["i est moi", "puis moi", "puis toujours
moi", "bon j'en ai marre", "et voilà on s'arrete
là", "non c'était une blague", "bon Stop !"]

for mot in L:
    print(mot)
```

Exercice.

1. Faire une boucle qui affiche les noms de toute votre famille
2. Faire une boucle qui compte tous les nombre de 1 à 1000
3. Faire une boucle qui dit le nombre de lettres pour le nom de chaque personne de votre famille (utiliser la fonction len())

4 Fonctions

Principe.

1. Comment un informaticien/mathématicien fait bouillir de l'eau avec une casserole vide, un robinet et une plaque chauffante ?
   Il remplit la casserole, il fait chauffer la casserole jusqu'à ébullition.
2. Comment un informaticien/mathématicien fait bouillir de l'eau avec une casserole vide, un robinet et une plaque chauffante ?
   Il vide la casserole et ré-applique la réponse de la question 1.

En informatique, on essaie d'être assez fainéant mais productif, c'est à dire lorsque l'on a fait quelque chose on essaie de le réutiliser. Pour ce faire on utilise des fonctions.
Comme en mathématiques, *une fonction c'est juste quelque chose en entrée et qui donne quelque chose en sortie.* En informatique parfois ce quelque chose c'est rien.

Que fait le bout de code suivant :

FIGURE 10 – Boucle suite arithmétique

```
A = 25
B = 0
for i in range(0,A):
    B = B + i
```

Maintenant si je ne veux pas recopier cette boucle à chaque fois mais que je veux juste le résultat B à la fin, je peux faire la fonction suivante :

FIGURE 11 – Fonction : Suite arithmétique

```
def SuiteArithmetique(A):
    B = 0
    for i in range(0,A):
        B = B + i
    return B
```

Exercice. Transformer en fonction tous les bouts de codes précédemment tapés.

FIGURE 12 – Une nouvelle classe : Eleve

```
class Eleve:
```

Principe (Fonction récursive)**.** Parfois une fonction peut s'appeler elle-même. On appelle cela la récursivité. C'est un mécanisme très puissant mais un peu difficile à conceptualiser. Cela permet notamment d'explorer des arbres (de données).

Exercice. Écrire une fonction à n associe $n * (n-1) * (n-2) * \cdots * (2) * 1$.

5 Programmation Orientée Objet : Les Classes et les Objets (En NSI)

5.1 Concept : Programmation Orientée Objet

Principe. Il consiste en la définition et l'interaction de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. Il possède une structure interne et un comportement, et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.

Beaucoup des types que vous connaissez sont considérés comme des **Classes** en Python. Par exemple, les listes, dictionnaires, string.

À la place de manipuler des Classes/types directement donnés par Python. On peut créer directement les objets/types qui nous intéressent.

Pour créer une nouvelle Classe, on fait : (la nouvelle classe est Eleve)

5.2 Attributs

Comme dit précédemment, **les Classes/Objets** sont utiles pour réaliser des modélisations. On va prendre le problème de modélisation suivant : on veut modéliser des élèves de Seconde qui veulent partir en Première générale, afin de pouvoir faire un programme qui propose des répartitions d'élèves dans des classes.

Première question : Que faut-il pour caractériser un tel Élève ... On posera qu'il faut connaître :

— Son nom
— Son genre/sexe
— Ses voeux de spécialité.
Ces caractéristiques sont appelés **Attributs** de la Classe.

Définition 5.1. Un objet est une instance d'une classe *i.e.* tout comme [1,2,3] est une instance d'une Liste. Un objet sera une instance d'une classe.

Pour créer un objet, on aura besoin d'un constructeur. Ce constructeur est la fonction/méthode : **___init___**. Donc pour créer un objet Élève avec les attributs : nom, genre, voeux on écrit :

FIGURE 13 – Classe avec Attributs

```
class Eleve:
    def __init__(self):
        self.nom =  "Moi"
        self.genre  = "Adolescent"
        self.voeux = []
```

On sait que **___init___** est une fonction. Donc elle prend une entrée. Ici l'entrée est *self*. Ce dernier représente l'objet.

Ainsi pour construire un objet de type/classe Eleve je vais faire écrire dans la console : (le nom de l'Objet sera Kevin)

FIGURE 14 – Création d'un Objet

```
>>> Kevin = Eleve()
```

Pour voir les attribut de l'Objet Kevin je fais :
On remarque que le nom de l'objet Kevin est "Moi". Alors que l'Objet s'appelle Kevin. Pourquoi cela ?
On a ce phénomène car lorsque l'on fait : Eleve(). La fonction qui est appelé est le constructeur **___init___** et cette fonction pose en attribut nom de l'objet : "Moi", en genre "Adolescent" et en voeux la liste vide. Pour modifier ces attribut (et donc les attribut de Kevin) on fait :

Exercice. L'objectif de cet exercice est de faire une classe Prof et de créer deux Objets : Gorce et Gibaud avec les bons attributs.

1. Trouvez les caractéristiques d'un professeur de lycée (sur papier).
2. Définir la classe Professeur (avec son constructeur)
3. Créer deux variables de type Professeur. Une variable sera Gibaud, l'autre Gorce.

FIGURE 15 – Voir Attribut d'un Objet

```
>>> Kevin.nom
'Moi'

>>> Kevin.genre
'Adolescent'

>>> Kevin.voeux
[]
```

FIGURE 16 – Redéfinition Attribut

```
>>> Kevin.nom = "Kévin"

>>> Kevin.genre = "H"

>>> Kevin.voeux = ["NSI","Math", "Geopolitique"]
```

4. Changer les attributs de ces deux fonctions pour que Gibaud et Gorce aient les bons attributs

Définition 5.2 (L'attribut Spécial : **___dict___**)**.** Cet attribut spécial vous donne tous les attributs d'un objet sous la forme d'un dictionnaire.

5.3 Les Méthodes

Toute la beauté de la programmation orientée objet est que les objets ont :

— des caractéristiques appelés **Attributs**

— des actions/fonctions appelés **Méthodes**

Les méthodes sont justes des fonctions internes à une Classe. Cela permet aux objets d'agir et d'interagir.

Pour faire une méthode on fait :

Nota Bene. **Toutes les méthodes prennent au moins self en entrée**

FIGURE 17 – Attribut Spécial

```
>>> Kevin.__dict__
{'nom': 'Kévin', 'genre': 'H', 'voeux': ['NSI', 'Math',
'Geopolitique']}
```

FIGURE 18 – Avec une méthode Simple : DirePresent

```
class Eleve:
    def __init__(self):
        self.nom = "Moi"
        self.genre = "Adolescent"
        self.voeux = []

    def DirePresent(self):
        str = self.nom + " présent ! "
        return str
```

Pour appeler cette méthode, on doit déjà créer l'objet puis appeler la méthode. (on va changer l'attribut d'abord). On fait :

FIGURE 19 – Appel de DirePresent dans la console

```
>>> Kevin = Eleve()

>>> Kevin.nom  = "Kévin"

>>> Kevin.DirePresent()
'Kévin présent ! '
```

Nota Bene. Pour qu'un objet appelle une méthode on met un • entre l'objet et la méthode. Comme la méthode est une fonction on met des parenthèses avec les arguments après la méthode. Si la méthode ne prend que self on met des parenthèses vides.

Cependant les méthodes peuvent être plus compliquées et faire des actions plus complexes. Par exemple **___init___** est une méthode ou **append** qui est une méthode pour les liste et qui permet d'ajouter un élément à la fin d'une liste.

On pourrait avoir le suivant :

Ainsi si je veux utiliser cette fonction je fais : (je reprend l'objet Kevin de la figure 19)

5.4 Héritage de Classe

Beaucoup d'autres choses sont possibles avec les Classes et les Objets. Pour cela n'hésitez pas à consulter Google quand vous voulez manipuler des objets ou faire des choses particulières. De plus beaucoup de packages amènent leurs objets que vous devrez manipuler.

FIGURE 20 – Une méthode pour faire l'appel

```
class Eleve:
    def __init__(self):
        self.nom = "Moi"
        self.genre = "Adolescent"
        self.voeux = []

    def DirePresent(self):
        str = self.nom + " présent ! "
        return str

    def Appel(self,L):
        L.append(self.nom)
        return L
```

FIGURE 21 – Faire l'Appel dans la console

```
>>> Presence = [] #Liste vide

>>> Presence = Kevin.Appel(Presence)

>>> Presence
['Kévin']
```

6 Paquets, Package : La grande communauté des informaticiens partagent tout (ou presque)