PYTHON LES PREMIERS PAS

I Python dans son environnement

Connaissance n°1 Python n'est pas un animal

Python est un langage de programmation interprété. Il a été crée par Guido van Rossum en 1989. Il doit son nom à la série *Monty Python's Flying Circus* .

Remarque n°1.

C'est un langage, nous allons donc en apprendre le vocabulaire, et la syntaxe.

Connaissance n°2 L'IDE

IDE signifie « integrated development environment » que l'on traduit en français par « environnement de développement ».
C'est un logiciel qui comporte des outils pour nous aider à programmer. Au lycée, nous utiliserons le plus souvent « Edupython ».
Les deux outils dont nous nous servirons le plus sont :
La console (ou shell) : C'est là que seront exécutés nos programmes ou scripts.
L'éditeur : C'est un éditeur de texte (comme notepad++) « intelligent » qui nous facilitera la syntaxe.

II Opérations , types ,variables et affectation

La console va nous permettre de lancer directement des instructions.

Opérations

On écrit dans la console

```
>>> 1+1
2
>>> 1-1
0
>>> 1*1
1
>>> 1/1
1.0
>>> 4.1*2
8.2
>>> 4**2
16
>>> 27//4
6
>>> 27%4
3
>>>
```

Le symbole >>> est le « prompt » (prompt primaire) qui nous indique que nous sommes dans la console.
C'est sur la dernière ligne commençant par ce symbole que nous pouvons écrire et exécuter une instruction.
On tape « 1+1 », on appuie sur « Entrée » et le résultat s'affiche à la ligne suivante (sans le >>>) .

- « plus » ou « moins » : « + » ou « - »
- « fois » ou « diviser » : « * » ou « / »
- « puissance ou plutôt exposant » : « ** »
- « quotient de la division euclidienne » : « // »
  (dans 27 =, il y a **6** fois 4 et il reste **3**)
- « reste de la division euclidienne » : « % »

Pour le quatrième calcul, la console affiche « 1.0 » à la place de « 1 ».
C'est parce que le résultat n'est pas du même type que les précédents.

Connaissance n°3 Les types

```
>>> type(14)
<class 'int'>
>>> type(7.0)
<class 'float'>
>>> type('bonjour')
<class 'str'>
>>> type("bonjour")
<class 'str'>
>>> type(bonjour)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    type(bonjour)
NameError: name 'bonjour' is not defined
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>>
```

Les principaux types sont :

- Les entiers : int
- Les flottants : float (nombres décimaux)
- Les string : str (Les chaînes de caractères)
- Les booléens : bool (« True » et « False »)
- Et bien d'autres...

Remarque n°2.

- Une chaîne de caractères est entourée soit d'apostrophes soit de guillemets.

II.1 Variables et affectation

Connaissance n°4 Définition très informelle…

En informatique, une **variable** peut être considérée comme une « petite boite » portant un nom et dans laquelle on peut « stocker » des « choses ».
Mettre quelque chose dans une boite, c'est faire une **affectation**.
En python, le symbole « = » sert à l'affection et n'a donc pas la même signification qu'en mathématiques.

Exemple n°1.

```
>>> a=5
>>> a
5
>>> b="bonjour"
>>> b
'bonjour'
>>> c='salut'
>>> c
'salut'
>>> bonjour=4
>>> bonjour
4
>>> salut
Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    salut
NameError: name 'salut' is not defined
>>>
```

- On peut mettre la valeur 5 dans la boite **a** :
« On affecte la valeur 5 à la variable **a** ».
- On peut mettre la chaîne de caractère « bonjour » dans la boite **b** :
« On affecte la chaîne de caractère « bonjour » à la variable **b** » :
- On peut mettre la valeur 4 dans la boite **bonjour** :
« On affecte la valeur 4 à la variable **bonjour**».
(Ce n'est pas une bonne idée…En effet :)
- **bonjour** est une variable qui est définie à présent, contrairement à salut.

Il ne faut pas confondre la variable **bonjour** avec la chaîne de caractère « bonjour »

Remarque n°3. N'est pas variable qui veut !

Le nom d'une variable est composé des lettres de a à z, de A à Z, et des chiffres 0 à 9, mais il ne doit pas commencer par un chiffre.
Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, etc. sont interdits, à l'exception du caractère _ (souligné). Le tiret (-) est bien sûr interdit puisqu'il correspond aussi à la soustraction.
La casse est significative : toto et Toto sont des variables différentes !
Python compte 33 mots réservés qui ne peuvent pas non plus être utilisés comme noms de variable (ils sont utilisés par le langage lui-même) :

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							

II.2 Opérations sur les variables

```
>>> a=3
>>> b=4
>>> c=a+b
>>> a
3
>>> b
4
>>> c
7
>>> a+b
7
>>> 2*a
6
>>> a/b
0.75
>>> (a-b)*c
-7
>>> a=a+5
>>> a
8
>>>
```

On peut faire toutes les opérations élémentaires avec des variables.

La dernière ligne nous montre qu'une variable informatique, ce n'est pas la même chose qu'une variable mathématique :
Il n'y aucun nombre vérifiant $a = a+5$ mais on peut mettre dans la boite **a** , la valeur qu'elle contient (déjà) augmentée de 5.

Souvenez-vous, en Python, le « = » est utilisé pour l'affectation pas pour décrire une égalité

(Dans les faits, c'est un peu plus subtil que cela mais il faudra prendre NSI pour en parler...)

III Les fonctions en python

Connaissance n°5

Pour définir une fonction en Python, on utilise les mots-clés : **def** et **return**
Elle sert, en général, à automatiser des calculs ou des suites d'instructions qui seraient utilisées souvent.

Exemple n°2.

Nous voulons automatiser le calcul de l'expression :
$(3{,}1\,x+2)(4\,x+1)-(2\,x+1)$

On écrit dans l'éditeur

```
def f(x):
    resultat = (3.1*x+2)*(4*x+1)-(2*x+1)
    return resultat
```

On définit ici une fonction qui s'appelle f et qui accepte un argument : x.

Connaissance n°6 L'indentation

On observe un décalage (de 4 espaces) afin de préciser « à Python » que les instructions font partie de la fonction f.
Ce décalage se nomme **l'indentation**.

Connaissance n°7 Exécuter un script

Une fois que l' on aura exécuté ce script (la flèche de lecture verte sur Edupython ou le mot « Run » si vous avez installé IDLE chez vous), on pourra utiliser la fonction f avec pour argument, par exemple : 4

On écrit dans la console

```
>>> f(4)
235.8
>>>
```

« resultat » est une variable dans laquelle on va stocker le résultat du calcul pour x valant 4.
Puis on renvoie (le mot-clé **return**) la variable résultat.
On a tapé « f(4) » dans la console puis on a appuyé sur « entrée »
La console affiche alors « 235.8 »

Remarque n°4.

On peut aussi être plus direct :

```
def f(x):
    return (3.1*x+2)*(4*x+1)-(2*x+1)
```

mais nous garderons ce genre de syntaxe pour plus tard.

Remarque n°5.

On peut ajouter plusieurs instructions entre le « def » et le « return » :

```
def une_fonction(une_variable) :
    resultat1 = une_variable + 2
    resultat2 = resultat1**2
    resultat3 = resultat2 // 5
    return resultat1
```

} « le corps de la fonction ».

Remarque n°6.

```
def f(x):
    resultat = (3.1*x+2)*(4*x+1)-(2*x+1)
    def g(m):
        le_calcul = m**2
        return le_calcul
    resultat = resultat + g(x)
    return resultat
```

On peut même écrire une fonction dans le corps d'une autre (même si il vaut mieux éviter) . Il faut alors penser à respecter l'indentation.

IV Les instructions conditionnelles

IV.1 Les conditions

Connaissance n°8

Une condition est une expression logique dont le résultat est soit « vrai » soit « faux ».

Connaissance n°9

Une condition est construite à l'aide d'opérateurs de comparaison :
- L'opérateur « égal à » noté = = (sans espace)
- L'opérateur « différent de » noté != ou < > (sans espace)
- Les opérateurs « inférieur à » ou « supérieur à » notés < et >
- Les opérateurs « inférieur ou égal à » ou « supérieur ou égal à » notés : <= et >=

Lorsque la situation à tester est plus compliquée, il est possible de combiner plusieurs conditions grâce aux opérateurs logiques :
- « and » qui signifie « et »
- « or » qui signifie « ou »
- « not » qui signifie « non »

Exemple n°3.

Une condition qui vérifie qu'une distance est inférieure à 40m et qu'un accès wifi est autorisé :

```
(distance < 40) and (acces == "autorisé")
```

Remarque n°7.

Suivant la valeur d'une condition (vraie ou fausse), la fonction choisit les actions à réaliser. On parle de structures conditionnelles.

IV.2 Les structures conditionnelles

Connaissance n°10 « if … else »

La structure conditionnelle « if…else » permet d'exécuter un bloc d'instructions lorsqu'une condition est vérifié et un autre bloc lorsqu'elle ne l'est pas.

Exemple n°4.

```
def peut_acceder(est_membre) :
    if est_membre == "oui" :
        reponse = "Accès autorisé"
    else :
        reponse = "Accès refusé"
    return reponse
```

Connaissance n°11 « if »

La structure conditionnelle « if » permet d'exécuter un bloc d'instruction lorsqu'une condition est vérifiée.

Exemple n°5.

```
def peut_acceder(est_membre) :
    reponse = "Accès refusé"
    if est_membre == "oui" :
        reponse = "Accès autorisé"
    return reponse
```

Connaissance n°12 « if … elif ...else »

La structure conditionnelle « if…elif…else » permet de gérer plusieurs conditions. Si une condition n'est pas validée, la
suivant est étudiée. En Python, « elif » est contraction de else if qui signifie « sinon si »

Exemple n°6.

```
def confidentalité(statut) :
    if statut == "secret" :
        reponse = "seuls les membres voient le groupe"
    elif statut == "fermé" :
        reponse = "tout le monde voit le groupe mais pas les publications"
    else :
        reponse = "tout le monde voit le groupe et les publications"
    return reponse
```

V Les boucles

Une boucle permet de répéter une ou plusieurs instructions.

V.1 Les boucles bornées

Lorsque l'on connaît le nombre de répétitions, on utilise une boucle bornée.

Connaissance n°13 Les boucles bornées

L'instruction Python correspondant à une boucle bornée est :
« **for** indice **in range**( ) : »

Remarque n°8.

« indice » est juste le nom de l'indice qui permettra de parcourir la boucle. On peut utiliser n'importe quel nom de variable à la place.

Connaissance n°14

« **range** » crée en quelque sorte une liste de nombres entiers dans (**in**) laquelle l'indice prendra ses valeurs.
« **range(0)** » donnera « une liste qui est vide »
« **range(1)** » donnera « une liste » contenant 1 entier à partir de 0 : c'est à dire 0.
« **range(2)** » donnera « une liste » contenant 2 entiers à partir de 0 : c'est à dire 0 et 1.
« **range(10)** » donnera « une liste » contenant 10 entiers à partir de 0 : c'est à dire 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 et 9.
etc..

« **range(2,10)** » donnera « une liste » allant de 2 jusqu'au 10^{e} entier à partir de zéro : c'est à dire 2 , 3 , 4 , 5 , 6 , 7 , 8 et 9.

« **range(2,10,3)** » donnera « une liste » allant de 2 jusqu'au 10^{e} entier à partir de zéro avec un « pas » de 3 : c'est à dire 2 , 5 et 8.

Exemple n°7.

La fonction

```
def somme_des_entiers(un_entier):
    somme = 0
    for nombre in range(un_entier) :
        somme = somme + nombre
    return somme
```

Quelques utilisations

```
>>> somme_des_entiers(1)
0
>>> somme_des_entiers(2)
1
>>> somme_des_entiers(3)
3
>>> somme_des_entiers(4)
6
```

V.2 Les boucles non bornées

Lorsque le nombre de répétitions n'est pas connu à l'avance, on utilise une boucle non bornée. Elle permet de répéter un bloc d'instruction tant qu'une condition est vérifiée.

Connaissance n°15

La structure correspondant à une boucle non bornée est « while condition : »

Remarque n°9.

Si la condition est toujours réalisée, la boucle se répétera indéfiniment. Il est donc important de vérifier que la condition cesse d'être vraie à terme afin que la boucle se termine.
Le nombre de passage dans une boucle non bornée étant inconnu au départ, il peut être nécessaire de créer une variable comptant le nombre de tours. Elle peut indiquer à partir de quand une condition n'est plus vérifiée.
Une variable servant de compteur permet de compter le nombre de passages dans la boucle dans une boucle non bornée.

Exemple n°8.

Une fonction sans argument qui donne le nombre d'années pour que la consommation d'internet atteigne 6000TWh/an.
Celle-ci est multipliée par 1,2 tous les ans. La variable « annees » sert de compteur.

La fonction

```
def nb_annee() :
    consommation=1500
    annees=0
    while consommation < 6000:
        consommation=consommation*1.2
        annees=annees+1
    return annees
```

Son utilisation

```
>>> nb_annee()
8
```

----------Ici s'arrête le programme de seconde----------

VI Les listes

Connaissance n°16 Définition très informelle…

Une *liste* est un tableau de valeurs qui peuvent être de différents types. Les « cases » de ce tableau ou plutôt les *éléments de cette liste* sont numérotés à partir de zéro. On reconnaît une liste au fait qu'elle est entourée de crochets et que ses éléments sont séparés par une virgule.

Exemple n°9.

À la variable **A**, nous avons affecté une liste comportant 5 éléments.
Puis nous avons « appelé » **A**, la console a alors affiché la liste.

```
>>> A = [5, "bonjour",5.1,-4,"fin"]
>>> A
[5, 'bonjour', 5.1, -4, 'fin']
>>>
```

Remarque n°10. Liste vide

On peut générer une *liste vide* et l'affecter à la variable **B** :

```
>>> B = []
>>> B
[]
>>>
```

Connaissance n°17 Connaître la longueur d'une liste

On peut obtenir la longueur de la liste **A** :

```
>>> len(A)
5
>>>
```

Connaissance n°18 Accéder à un élément d'une liste

Pour obtenir l'élément numéro (de rang) 2 de la liste **A** :
Attention, comme on commence à zéro, l'élément de rang 2 est le troisième de la liste...

```
>>> A[2]
5.1
>>>
```

Remarque n°11. Le premier et le dernier élément

Le premier élément a pour rang zéro et le dernier a pour rang -1 .

```
>>> A
[5, 'bonjour', 5.1, -4, 'fin']
>>> A[0]
5
>>> A[-1]
'fin'
>>>
```

Connaissance n°19 Ranger une liste

En général, vos listes contiendront des éléments du même type.
Il est alors possible de les ranger par ordre croissant.
Attention, vous rangez les éléments mais pas la liste elle-même.
Si vous souhaitez conserver ce tri alors il faut l'affecter à une variable.

```
>>> C = [9,5,8,6,1,3]
>>> C
[9, 5, 8, 6, 1, 3]
>>> sorted(C)
[1, 3, 5, 6, 8, 9]
>>> C
[9, 5, 8, 6, 1, 3]
>>> D = sorted(C)
>>> D
[1, 3, 5, 6, 8, 9]
>>>
```

Connaissance n°20 ***Modifier un élément***

On peut modifier un élément d'une liste en faisant une affectation.

```
>>> C
[9, 5, 8, 6, 1, 3]
>>> C[4] = 25
>>> C
[9, 5, 8, 6, 25, 3]
>>>
```

Connaissance n°21 ***Fusionner deux listes***

On peut fusionner deux listes avec « + ».

```
>>> E = ['a','b','c']
>>> F = ['d','e']
>>> E+F
['a', 'b', 'c', 'd', 'e']
>>>
```

Remarque n°12. ***Ajouter un élément à la fin d'une liste***

« + » peut servir à ajouter un élément à la fin d'une liste.

Attention, si vous souhaitez conserver l'ajout, il faut penser à affecter la nouvelle liste.

```
>>> E
['a', 'b', 'c']
>>> E + ['un de plus']
['a', 'b', 'c', 'un de plus']
>>> E
['a', 'b', 'c']
>>> E = E + ['un de plus']
>>> E
['a', 'b', 'c', 'un de plus']
>>>
```

Remarque n°13. ***Une autre façon d'ajouter un élément***

On peut utiliser la *méthode* append

Attention à la syntaxe !
Cette fois-ci la liste est directement modifiée...

```
>>> F
['d', 'e']
>>> F.append('un de plus')
>>> F
['d', 'e', 'un de plus']
>>>
```

Connaissance n°22 ***Supprimer un élément***

On peut supprimer l'élément de rang k .
(ici $k=1$)

On peut aussi supprimer le dernier élément

```
>>> F
['d', 'e', 'un de plus']
>>> del(F[1])
>>> F
['d', 'un de plus']
>>> E
['a', 'b', 'c', 'un de plus']
>>> del(E[-1])
>>> E
['a', 'b', 'c']
>>>
```

Remarque n°14. ***Copier une liste***

Il faut faire attention quand on copie une liste.

```
>>> C
[9, 5, 8, 6, 25, 3]
>>> M = C
>>> M
[9, 5, 8, 6, 25, 3]
>>> M[4] = 'haha'
>>> M
[9, 5, 8, 6, 'haha', 3]
>>> C
[9, 5, 8, 6, 'haha', 3]
>>>
```

```
>>> C = [9,5,8,6,1,3]
>>> C
[9, 5, 8, 6, 1, 3]
>>> M = C[:]
>>> M
[9, 5, 8, 6, 1, 3]
>>> M[4] = 'haha'
>>> M
[9, 5, 8, 6, 'haha', 3]
>>> C
[9, 5, 8, 6, 1, 3]
>>>
```

VII « print » et « input »

Ces deux instructions ne sont pas abordées dans ce cours alors qu'elles sont très utiles…La raison est la suivante : Nous utilisons Python afin de travailler l'algorithmique et n'avons pas vocation à faire de la programmation pour de la programmation…Néanmoins nous en parlerons en TD ...