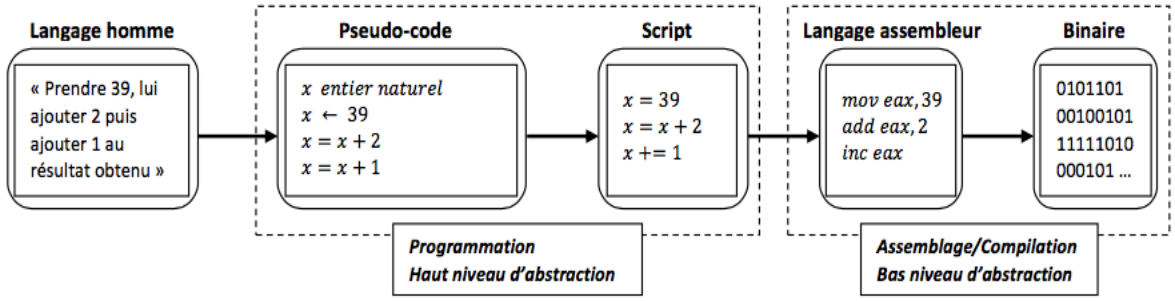
2 | Les structures de données

1 Introduction sur les niveaux d'abstractions

Python est un langage de **très haut niveau d'abstraction** (⇒ simplicité mais perte d'efficacité), **impératif** (le programme exécute des instructions successives), **orienté objet** (on agit sur des structures de données) et permettant une **approche fonctionnelle** (il est possible de décomposer un programme principal en fonctions). Il se prête très bien à l'algorithmique mais il est fréquent, dans le domaine professionnel, qu'il ne serve que de prototypage et soit ensuite suivi d'une reprogrammation en C, moins intuitive mais souvent plus efficace.



2 Structures de base d'un pseudo-code

2.1 Les expressions

Définition :

Une expression est une combinaison de valeurs, de variables, d'opérateurs et de fonctions, qui est évaluée en suivant les règles de calcul du langage de programmation pour retourner une nouvelle valeur.

Exemple : « 3 + 2 » et « True or False » sont des expressions dont les valeurs sont 5 et True.

Quelques types de valeurs en Python : int, bool et float (situé après le mot class lors de l'utilisation de la fonction « type ».

```
>>> type (3 + 2)
<class'int'>

>>> type (True)
<class'bool'>

>>> type (3.2)
<class'float'>
```

2.2 Les types de variables

On distingue plusieurs **types de variables** en programmation, définissant en partie les actions possibles sur ces variables. On peut citer notamment les types suivants (très loin d'être exhaustifs) :

Nombre			**Booléen**	**Chaîne de caractère**	**Tableaux**			**Autres objets**	
Entier	Flottant	Complexe			n-uplet	Liste		Ensemble	Autre
int	$float$	$complex$	$bool$	str	$tuple$	$list$	$array$	set	Dict, class, file, function, …
42	42.0	1+2j	True/False	''Hello word!'' "42"	(4,2)	[4,2]	array([4,2])	{4,2}	…

Remarques :

- le type $array$ est un type importé de $numpy$ mais dont on se servira fréquemment en ingénierie numérique, souvent sous forme de tableaux (notamment pour les vecteurs et les matrices).
- L'imaginaire i est remplacé par j sous Python (mais il faut mettre un nombre devant par exemple 1j).

Il est possible d'affecter un **contenu** (ou **valeur**) à une **variable**. L'**affectation** intervient en dernier, après que les instructions du membre de droite ont été traitées. Celle-ci se fait avec l'instruction $=$.

2.3 Opérations numériques, Opérateurs booléens et de comparaisons

Python	**Résultat**	**Exemple**	**Sortie (output)**
x + y	Somme de x et y	2+3	5
x – y	Différence de x et y	2-3	-1
x * y	Produit de x et y	2*3	6
x / y	Division de x par y	3/2.	1.5
x //y	Quotient de la division entière de x par y	7//2	3
x % y	Reste de la division entière de x par y	7%2	1
-x	Opposé de x		
abs(x)	Valeur absolue de x	abs(-3)	3
int(x)	Conversion de x en l'entier le plus proche de 0	int(-4.9)	-4
float(x)	Conversion de x en flottant	float(2)	2.0
divmod(x,y)	Le doublet (x//y , x % y)	divmod (7,3)	(3,1)
x**y	x puissance y	2**7	128

Python	**Résultat**	**Exemple**	**Sortie (output)**
a.conjugate( )	Conjugué d'un nombre complexe	(3-4j).conjugate	3+4j
a.real	Partie réelle du complexe a	(3-4j).real	3.0
a.imag	Partie imaginaire du complexe a	(3-4j).imag	-4.0
abs (a)	Module du complexe a	abs (3-4j)	5.0

Python	Résultat	Exemple	Sortie (output)
<	Strictement inférieur à	3 < 3	False
<=	Inférieur ou égal à	3 <= 3	True
>	Strictement supérieur à		
>=	Supérieur ou égal à		
==	Egal à	4 == 2	False
!=	Différent de	2 != 3	True

Python	Résultat	Exemple	Sortie (output)
x or y	Vrai si x ou y sont vrais (ou les deux)	0 or 1	True
x and y	Vrai si x et y sont vrais tous les deux	0 and 1	False
not x	Vrai si x est faux et inversement	not (0)	True

Mise en garde :

- La multiplication doit être explicitement indiquée. 5(3+6) renvoie une erreur : l'expression correcte est 5*(3+6).
- Le séparateur décimal des flottants est un point (et pas la virgule, qui est typiquement française).
- On confond beaucoup au début = et == qui ont des rôles très différents.

3 Les variables

Les variables ne sont rien d'autre qu'une réservation d'espace mémoire pour stocker des valeurs. L'opérateur d'affectation est le = avec à gauche le nom de la variable et à droite la valeur à stocker dans la variable. Ce symbole = n'est pas symétrique comme celui des maths !

L'ensemble des variables ainsi que les valeurs stockées dans ces variables à un instant t définissent le **« contexte »** (accessible sur Python onglet affichage).

Attention certains mots sont réservés par le langage Python, ils ne peuvent donc pas être utilisés comme nom de variable. (False, None, not, or, in, is, return, try, if ...)

Remarques :

- Pour faciliter la lecture d'un programme il est conseillé de donner des noms explicites aux variables. (h pour hauteur, l pour largeur,…)
- Il est tout à fait possible d'affecter le résultat d'une expression à une variable (a = 2.5 + 3)
- On peut affecter à une variable le résultat d'une expression dans laquelle cette variable intervient. (a = a + 2)
- La syntaxe Python introduit des opérateurs comme += afin de simplifier ce type d'expression (a = a + 2 devient alors a+=2). Le principe est le même pour la plupart des opérateur classique (-= , *= , /= , …)

4 Types Séquences

4.1 Introduction

Définition :

Les séquences sont des types informatiques qui permettent de stocker plusieurs valeurs d'une manière efficace et organisée.

Nous allons voir quatre types séquences de Python : les list, les array (tableau), les tuples, et les str (chaînes de caractères). En pratique, on utilise aussi un cinquième type : range, surtout dans les boucles for.

Définition :

Une **séquence est dite immuable** si les valeurs stockées dans la séquence ne peuvent pas être modifiée après création (exemple : les tuples et les str). L'utilisation de ce type de séquence est plus sûre car il n'est pas nécessaire de se préoccuper d'une éventuelle modification de celle-ci.

On parle de **séquence variable** dans le cas contraire : Il est possible de modifier les valeurs stockées dans la séquence et de réaliser des affectations après la déclaration et l'initialisation d'une variable (exemple : les list). Une affectation d'une variable sur une liste déjà existante se retrouvera donc modifiée si la liste de base l'est (>>> L = [1,2,3] >>> M = L >>> L[0] = 0 >>> print(M) -> [0,2,3]).

4.2 Les listes

Les listes sont des **séquences variables**. Pour créer une liste on utilise des crochets : [ ]. Une liste peut être vide (L=[]). Les valeurs à l'intérieur de la liste peuvent être de types quelconques et sont séparés par des virgules (L=["soleil", 2.5 , 4 ,"informatique"]).

Les opérations suivantes sont données avec s et t des séquences de même type, n, i, j et k des entiers et x un objet dont le type répond aux contraintes imposées par s.

Opération	Résultat
x in s	True si un élément de s est égal à x, False sinon
x not in s	False si un élément de s est égal à x, True sinon
s + t	Concaténation de s et t
s* n ou n * s	n copies de s concaténées
len(s)	Nombre d'éléments contenus dans s
min(s)	Le plus petit élément de s
max(s)	Le plus grand élément de s
s[i]	i-ème élément de s (attention les indices commencent à 0 !!)
s[i :j]	Séquence composée des éléments de s du i-ème inclus au j-ème exclus
s[i :j :k]	Séquence composée des éléments de s du i-ème inclus au j-ème exclus avec un pas k.
s[i] = x	i-ème élément remplacé par x
del s[i :j :k]	Les éléments de s[i :j :k] sont supprimés
s.append(x)	Ajoute x à la fin de la liste s
s.clear()	Supprime tous les éléments de s
s.insert(i,x)	Insère x à la i-ème place dans s
s.pop[i]	Renvoie s[i] et le supprime de s
s.remove(x)	Supprime le premier élément de s tel que s[i] == x
s.reverse()	Inverse la place des éléments de s

Pas de panique, pas besoin de tout mémoriser aujourd'hui : nous allons pratiquer, et les commandes seront souvent rappelées.

4.3 Les tableaux (array)

Pour utiliser les tableaux il est nécessaire d'importer le module numpy (*import numpy as np* ou encore *from numpy import **). Dans le premier cas, toutes les fonctions définies dans le module doivent être précédées de np.

Les tableaux sont des séquences dont la taille est fixée à la création et dont tous **les éléments contenus doivent être de même type**.

Un tableau peut être créé à partir d'une liste ou d'un n-uplet.

Il peut aussi être créé à l'aide de la fonction **arange** qui fonctionne comme la fonction range.

arange(début inclus, fin exclus, pas) , les arguments peuvent être des décimaux (float) contrairement au type range qui ne prend que des entiers (int).

```
>>> a = np.array([3,4,5,6])
>>> b = np.array((5.,4.,6.))

>>> a.dtype
dtype('int64')

>>> b.dtype
dtype('float64')

>>> np.arange(1,10,2)
array ([1, 3, 5, 7, 9])
```

Attention : quand arange est utilisé avec des flottants, il est en général impossible de prédire le nombre d'éléments obtenus à cause de la précision finie de la représentation des flottants. Il est conseillé d'utiliser la fonction **linspace** qui prend comme argument le nombre d'éléments que nous désirons avoir dans le tableau.

```
>>> np.linspace (0.1 , 1.3, 7)                    # 7 nombres de 0.1 à 1.3 inclus
array ([ 0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.3 ])
```

L'accès aux éléments se fait de la même manière que pour les listes (a[1] renvoie 4).

La dimension du tableau s'obtient par la fonction **shape** (a.shape -> (4,)).

Génération de tableaux à deux dimensions :

```
>>> a = np.arrange(12)
>>> a
array ([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ])

>>> a.reshape(3,4)
array (   [0, 1, 2, 3] ,
          [4, 5, 6, 7],
          [8, 9, 10, 11] ] )

>>> b = np.zeros ((3,4))          # np.ones((,)) fonctionne de la même façon avec des 1.
>>> b
array (   [0., 0., 0., 0.] ,
          [0., 0., 0., 0.],
          [0., 0., 0., 0.] ] )
```

4.4 Les tuples

Les tuples (n-uplets) sont des séquences immuables qui peuvent être construite de différentes manières :

- En utilisant des parenthèses pour le tuple vide ( )
- En utilisant une virgule pour un tuple constitué d'un élément a, ou (a,)
- En séparant les éléments par une virgule a, b, c ou (a, b, c)

Les opérations sont similaires à celles sur les listes, à l'exception des affectations d'éléments, puisque les tuples sont immuables (ses éléments ne peuvent être changés comme pour une liste).

4.5 Les chaînes de caractères (str)

Ce sont des séquences immuables de caractères, qui peuvent être définies avec des guillemets simples, doubles ou triples. Idéal pour manipuler du texte.

```
>>> a = '' Informatique ''
>>> a[4]
'r'

>>> a[2 :6]
'form'

>>> a[3] = 'e'  #Le type str est immuable, et ceci devrait renvoyer un message d'erreur.
TypeError: 'str' object does not support item assignment
```

4.6 Les dictionnaires

Un **dictionnaire** en python est une sorte de liste mais au lieu d'utiliser des index (c'est-à-dire des nombres qui donnent la position), on utilise des **clés** alphanumériques (par exemple une chaîne de caractères, un tuple…). Les clefs pointent vers des valeurs (ce que l'on stocke). Il faut concevoir que cette clé a l'avantage d'indiquer quelle information est stockée.

Un dictionnaire est délimité par des accolades, et ses éléments sont de la forme « clef : valeur » et séparés par des virgules. Son type est « dict ».

Exemple :

```
>>> identite={''nom'':''Popoff'',''prenom'':''Nicolas'',''age'':38}
>>>identite[''prenom'']  #On accede à l'élément dont la clé est ''prénom"
''Nicolas''
```

On ajoute très facilement un élément :

```
>>>identite[''taille'']=1.78
```

On parcourt les clés avec la méthode *.keys* et les valeurs avec la méthode *.values*, par exemple :

```
>>>for cle in identite.keys
        print cle
nom
prenom
age
taille
```

Exemple d'utilisation des dictionnaires : compter le nombre d'occurrences des lettres dans une phrase. Les clefs sont les lettres, et les valeurs sont les nombres d'occurrences (sera vu en TP).

Table des matières