5

Python en mode calculatrice

5.1 Compétences exigées

— Objectifs —

La capacité évaluée dans cette partie de la formation est :

- choisir un type de données en fonction d'un problème à résoudre.

5.2 Manipuler des nombres

En première approche, le langage *Python* peut être vu comme un ensemble de commandes simples à utiliser. En ce sens, il peut jouer le rôle d'une calculatrice scientifique classique. Pour s'en convaincre il suffit de lancer la séquence triviale suivante :

```
>>> a=1
>>> b=2
>>> c=a+b
```

Pour avoir le résultat de l'opération, on prolongera simplement par :

```
>>> print (c)
3
```

— Remarque —

La fonction `print()` sera vu prochainement (page 56) : elle permet d'afficher un résultat mais n'est pas indispensable en mode calculatrice.

Plus simplement, en console, on peut faire :

```
>>> c
3
```

5.3 Affectation

5.3.1 Affectation simple

On vient de le voir, l'affectation se fait avec le signe d'égalité = qui n'a rien à voir avec le signe d'égalité mathématique. En effet, les instructions :

```
x = 0
x = x + 1
```

n'ont aucune signification mathématique alors qu'en *Python*, on incrémente (on augmente) de 1 la variable x.

5.3.2 Affectations simultannées ou multiples

Il est possible d'effectuer des affectations simultanées :

```
>>> a,b,c = 5,9,12
>>> a
5
>>> b
9
>>> c
12
```

ou multiples :

```
>>> x = y = 2
>>> y = y + 2
>>> y
4
>>> x
2
```

⚠ N'utilisez pas l'affectation multiple pour les listes car celles-ci seraient alors stockées à la même adresse, ce qui aurait pour conséquence de changer les 2 listes lors de la modification d'une des deux : cf page 52.

5.4 Variables

L'essentiel du travail effectué par un programme d'ordinateur consiste à manipuler des données. Ces données peuvent être très diverses, mais dans la mémoire de l'ordinateur, elles se ramènent toujours en définitive à une suite finie de nombres binaires.

Pour pouvoir accéder aux données, le programme d'ordinateur (quel que soit le langage dans lequel il est écrit) fait abondamment usage d'un grand nombre de variables de différents types.

Une variable apparaît dans un langage de programmation sous un nom de variable à peu près quelconque, mais pour l'ordinateur il s'agit d'une référence désignant une adresse mémoire, c'est-à-dire un emplacement précis dans la mémoire vive.

À cet emplacement est stockée une valeur bien déterminée. C'est la donnée proprement dite, qui est donc stockée sous la forme d'une suite de nombres binaires, mais qui n'est pas nécessairement un nombre aux yeux du langage de programmation utilisé. Cela peut être en fait à peu près n'importe quel « objet » susceptible d'être placé dans la mémoire d'un ordinateur, par exemple : un nombre entier, un réel, un complexe, un vecteur, une chaîne de caractères typographiques, un tableau, une fonction, · · ·

Sous *Python*, les noms de variables doivent obéir à quelques règles simples :

- Un nom de variable est une séquence de lettres (de a à z et de A à Z) et de chiffres (0 à 9), qui doit toujours commencer par une lettre.

- Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, %, $\cdots$ sont interdits, à l'exception du caractère _ (souligné donné par la touche 8).
- La casse est significative (les caractères majuscules et minuscules sont distingués). Ainsi, `Toto`, `toto`, `TOTO` et `ToTo` sont donc des variables différentes.

Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules (y compris la première lettre). Il s'agit d'une simple convention, mais elle est largement respectée. N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans `tableDesMatieres`, par exemple.

Évitez si possible les caractères ℓ, 1 et I que l'on peut confondre ainsi que les 0 que l'on peut confondre avec des O.

En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme noms de variables les 33 « mots réservés » ci-dessous (ils sont utilisés par le langage lui-même) :

mot	utilisation	page
`and`	opérateur logique `ET`	43
`as`	associé à `import` pour utiliser les bibliothèques	225, 229
`assert`	permet de rajouter des contrôles pour le débogage d'un programme	non étudié
`break`	interrompt une boucle `while` ou `for`	62
`class`	utilisée en programmation orientée objet (POO)	non étudié
`continue`	saute l'étape suivante dans une boucle `while` ou `for`	62
`def`	définit une fonction	65, 128
`del`	supprime un ou plusieurs éléments d'une liste à partir de leur index	51
`elif`	instruction conditionnelle `SINON SI`	58
`else`	instruction conditionnelle `SINON`	58
`except`	gestion d'erreurs	76
`False`	booléen `FAUX`	46
`finally`	gestion d'erreur	76
`for`	boucle `POUR`	60
`from`	importation de bibliothèques	229
`global`	définit des variables globales dans une fonction	71
`if`	instruction conditionnelle `SI` 58	
`import`	importation de bibliothèques	229
`in`	test d'appartenance	51
`is`	test d'égalité	47
`lambda`	définit une fonction par une expression	128
`None`	objet qui ne vaut "rien"	67
`nonlocal`	gestion des variables (locales et globales)	71
`not`	opérateur logique `NON`	59
`or`	opérateur logique `OU`	43
`pass`	instruction vide	59
`raise`	levée d'exception	77
`return`	valeur de retour d'une fonction	65
`True`	booléen `VRAI`	46
`try`	gestion d'erreur	76
`while`	boucle `TANT QUE`	61
`with`	simplification d'écriture	non étudié
`yield`	remplace `return` dans un générateur	non étudié

TABLE 5.1 – Mots réservés

Vous retrouverez cette liste en annexe page 310.

Les noms commençant par une majuscule ne sont pas interdits, mais l'usage veut qu'on le réserve plutôt aux variables qui désignent des classes (le concept de classe ne sera pas abordé en CPGE).

Une variable sert d'espace de stockage pour les résultats intermédiaires d'un calcul. Elle possède un certain nombre de caractéristiques :

- **identificateur** : c'est son nom. Il doit être **bien choisi** – surtout dans un programme long – et respecter les règles précédentes.
- **type** : Dans certains langages, le type d'une variable peut changer en cours d'exécution du programme. C'est le cas de *Python*.
- **portée** : On ne peut utiliser le nom d'une variable que dans la fonction ou la procédure (en réalité le bloc) qui la contient. La durée de vie de la variable correspond à la durée d'exécution de la fonction ou de la procédure (du bloc). Sans ce principe, les programmes longs deviendraient incompréhensibles et difficiles à corriger.

Attention à ne pas confondre le symbole $=$, utilisé comme symbole d'affectation avec le symbole $=$ d'égalité mathématique qui indique que deux valeurs sont égales. L'équation mathématique $a = a+1$ est par exemple sans intérêt en mathématiques, alors que la ligne de programme $a = a+1$ indique que la variable a doit être incrémentée de 1.

5.5 Types numériques

En mathématiques, les nombres appartiennent à des ensembles tels que $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ ou $\mathbb{C}$. En *Python* (comme dans beaucoup de langages informatiques) les nombres seront du type :

- `int` (integer) : correspond à un entier ($\mathbb{N}$ ou $\mathbb{Z}$). Sa taille est limitée par la mémoire de l'ordinateur.
- `float` : le type flottant, permet de représenter des nombres à virgule. Il est codé en mémoire sur 8 octets (64 bits). Il est représenté sous la forme nombre $= \pm 1.\text{mantisse} * 2^{\pm \text{exposant}}$. La mantisse est écrite avec 52 chiffres binaires, et l'exposant avec 10 ; il y a deux bits de signe. La précision maximale est donc de l'ordre de 2×10^{-16}. Des valeurs spéciales permettent de représenter $-\infty$, $+\infty$ et `Nan`, *not a number*, souvent issu d'une forme indéterminée..
  Les réels, que l'on manipule en mathématiques, n'existent pas en *Python* : ils sont remplacés par les nombres à virgule flottante. Cela implique que tous les calculs sur ordinateur sont intrinsèquement faux ...
- `complex` : le type complexe correspond à une structure naturellement composée de deux flottants (partie réelle et imaginaire) sur 2 x 8 = 16 octets.
- `bool` : le type booléen correspond à l'algèbre booléenne et ne prend que deux valeurs, `True` / `False`. Il est codé sur un bit.

En pratique, pour des applications basiques, le typage est la plupart du temps transparent pour l'utilisateur car *Python* interprète dynamiquement le type des variables.
Par exemple sur une machine 32 bits, on peut lancer la séquence suivante, illustrant le type *entier int* :

Exemple pour les entiers :

```
>>> c=2**10000
>>> c
199506311688075838488374216268358508382349683188619245485200894985294388302219466319199616840361945978993311294232091242715564913494137811175937859320963239578557300467937945267652465512660598955205500869181933115425086084606181046855090748660896248880904898948380092539416332578506215683094739025569123880652250966438744410467598716269854532228685381616943157756296407628368807607322285350916414761839563814589694638994108409605362678210646214273333940365255656495306031426802349694003359343166514592977732796657756061725820314079941981796073782456837622800373028854872519008344645814546505579296014148339216157345881392570953797691192778008269577356744441230620187578363255027283237892707103738028663930314281332414016241956716905740614196543423246388012488561473052074319922596117962501309928602417083408076059323201612684922884962558413128440615367389514871142563151110897455142033138202029316409575964647560104058458415660720449628670165150619206310041864222759086709005746064178569519114560550682512504060075198422618980592371180544447880729063952425483392219827074044731623767608466130337787060398034131971334936546227005631699374555082417809728109832913144035718775247685098572769379264332215993998768866608083688378380276432827751722736575727447841122943897338108616074232532919748131201976041782819656974758981645312584341359598627841301281854062834766490886905210475808826158239619857701224070443305830758690393196046034049731565832086721059133009037528234155397453943977152574552905102123109473216107534748257407752739863482984983407569379556466386218745694992790165721037013644331358172143117913982229838458473344402709641828510050729277483645505786345011008529878123894739286995408343461588070439591189858151457791771436196987281314594837832020814749821718580113890712282509058268174362205774759214176537156877256149045829049924610286300815355833081301019876758562343435389554091756234008448875261626435686488335194637203772932400944562469232543504006780272738377553764067268986362410374914109667185570507590981002467898801782719259533812824219540283027594084489550146766683896979968862416363133763939033734558014076367418777110553842257394991101864682196965816514851304942223699477147630691554682176828762003627772577237813653316111968112807926694818872012986436607685516398605346022978715575179473852463694469230878942659482170080511203223654962881690357391213683383935917564187338505109702716139154395909915981546544173363116569360311222499379699992267817323580231118626445752991357581750081998392362846152498810889602322443621737716180863570154684840586223297928538756234865564405369626220189635710288123615675125433383032700290976686505685571575055167275188991941297113376901499161813151715440077286505731895574509203301853048471138183154073240533190384620840364217637039115506397890007428536721962809034779745333204683687958685802379522186291200807428195513179481576244482985184615097048880272747215746881315947504097321150804981904558034168269497871413160632106863915116817743047925967093765
>>> type(c)
<class 'int'>
```

— Remarque importante —

Si un programme est "planté" ou met trop de temps à s'exécuter, on peut arrêter son exécution par la commande CTRL - C.

Pour les booléens (| et & indiquent respectivement *OU* et *ET*), on peut lancer la séquence :

```
>>> a=True
>>> b=False
>>> a|b
True
>>> a&b
False
```

— Remarques —

Pour les booléens :

- $a|b$ est équivalent à a or b
- $a\&b$ est équivalent à a and b

mais en règle générale, ce n'est pas vrai :
http://python.developpez.com/cours/DiveIntoPython/php/frdiveintopython/power_of_introspection/and_

On peut représenter la table de vérité suivante :

a	b	a and b	a or b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

TABLE 5.2 – table de vérité

Pour les nombres complexes, on peut tester :

```
>>> a=1.5+3j
>>> b=1.5
>>> a-b
3j
>>> a*b
(2.25+4.5j)
>>> c=1+1j
>>> a*c
(-1.5+4.5j)
>>> (1+4j)**(1/(2+3j))
(1.67585169460166-0.20706889169199882j)
>>> 1j * complex(0,1)
(-1+0j)
```

En réalité, il est possible d'utiliser la structure des nombres complexes de façon plus évoluée. Par exemple :

```
>>> a=1.5+3j
>>> a.real
1.5
>>> a.imag
3.0
>>> a.conjugate()
(1.5-3j)
```

— Remarque —

a.conjugate est appelée une fonction ou encore mieux, une méthode.

Enfin, il est possible de modifier explicitement le type des variables en utilisant les fonctions de conversion :

`int(a)`	Conversion de a en type entier
`long(a)`	Conversion a en type entier long [1]
`float(a)`	Conversion a en type flottant
`complex(a)`	Conversion a en type complexe a+0j
`complex(a,b)`	Conversion de a et de b en type complexe a+bj

TABLE 5.3 – Fonctions de conversion de type

Sous *Python*, il existe la fonction `type()` qui donne le type d'une variable. Par exemple :

```
>>> a=1
>>> type(a)
<class 'int'>
>>> a=1.
>>> type(a)
<class 'float'>
>>> a=2**1000
>>> type(a)
<class 'int'>
>>> a=3+4j
>>> type(a)
<class 'complex'>
```

1. Avec *python 2.7* mais pas avec *python 3*

5.6 Opérations sur les nombres

5.6.1 Opérateurs arithmétiques

Python ne possède que très peu d'opérations arithmétiques pour manipuler les nombres. À ce stade, *Python* se résume à une calculatrice élémentaire :

Opération	Signification	Exemple	Résultat
+	Addition	a="Les"+ " " + "chats"	a = "Les chats"
−	Soustraction	b = 8 - 2	b = 6
*	Multiplication	c = "to" * 2	c = "toto"
/	Division	d = 7 / 2	d = 3.5
**	Puissance	e = 2 ** 3	e = 8
%	Reste de division entière	f = 7 % 2	f = 1
//	Quotient de division entière	g = 7 // 2	g = 3

TABLE 5.4 – Opérateurs arithmétiques

La nature des opérations arithmétiques dépend du type des variables.

5.6.2 Raccourcis pour les opérateurs arithmétiques

Dans une séquence de code, on peut affecter une variable à une valeur, effectuer une opération arithmétique sur cette variable et finalement affecter la nouvelle valeur obtenue à la variable. Par exemple :

```
>>> a=1
>>> b=4
>>> a=a+b
>>> a
5
```

En *Python*, il est possible d'écrire l'opération arithmétique de façon plus compacte :

```
>>> a=1
>>> b=4
>>> a+=b
>>> a
5
```

Ceci est également vrai pour les autres opérateurs arithmétiques :

a+=b	a=a+b
a-=b	a=a-b
a*=b	a=a*b
a/=b	a=a/b
a**=b	a=a**b
a%=b	a=a%b

TABLE 5.5 – Raccourcis opérateurs

5.6.3 Opérateurs de comparaison

Python peut aussi vérifier si une comparaison entre deux nombres est vraie ou fausse ce qui renvoie une valeur booléenne. Les opérateurs standards de comparaison sont :

<	Plus petit que
>	Plus grand que
<=	Plus petit que ou égal à
>=	Plus grand que ou égal à
==	Égal à
!=	Différent de
is	Est

TABLE 5.6 – Opérateurs de comparaison

Quelques subtilités entre `is` et == :

```
>>> a = [1,2,3,4]
>>> b=a
>>> b is a
True
>>> b == a
True
>>> c = a[:]
>>> c is a
False
>>> c == a
True
```

5.6.4 Quelques fonctions supplémentaires

De base, Python fournit quelques fonctions numériques autres que les opérations arithmétiques :

`abs(a)`	Valeur absolue de a
`max(...)`	Plus grande valeur d'une suite de nombres
`min(...)`	Plus petite valeur d'une suite de nombres
`round(a,n)`	Arrondi de la variable `a` au niveau de la $n^{\text{ième}}$ décimale

TABLE 5.7 – Autres opérateurs arithmétiques

Par exemple :

```
>>> a=-1
>>> abs(a)
1
>>> a=1
>>> b=3.4
>>> c=10
>>> max(a,b,c)
10
>>> min(a,b,c)
1
>>> a=3.141592654
>>> round(a,4)
3.1416
>>> round(a,2)
3.14
>>> round(a,0)
3.0
```

5.7 Opérations sur les bits

Python dispose de 6 opérateurs de base pour agir directement sur les bits :

&	ET
|	OU
^	OU exclusif
∼	Inversion des bits du nombre situé à droite
≫	Décalage d'un bit à droite (division par 2)
≪	Décalage d'un bit à gauche (multiplication par 2)
is	Est

TABLE 5.8 – Opérateurs sur les bits

Exemple :

```
>>> 98&54
34
>>> bin(98)
'0b1100010'
>>> bin(54)
'0b110110'
>>> bin(34)
'0b100010'
```