Partie 1

Base de connaissance

LE SAVIEZ-VOUS ?

Ordinateur

Le mot *ordinator* existait déjà en latin. Chez les Romains, il désignait celui qui met en ordre, qui règle ou qui préside à la remise en ordre. Les premiers Chrétiens l'utilisaient pour nommer celui qui dirige les cérémonies : c'est pourquoi on parle encore de l'ordination des prêtres.

Francisé en ordinateur, il apparaît en français, dans le même sens, vers 1600 mais reste très peu employé. En 1955, la société IBM s'adresse à un latiniste, Jacques **Perret**, pour trouver un nom à ce que les Anglais appellent un computer, mot construit sur le latin *computare* qui signifie *calculer*. Celui-ci propose le terme d'ordinateur qui s'est désormais imposé dans notre langue.

On peut s'en réjouir, moins par le souci d'utiliser un mot français, que pour donner un sens plus large à la terminologie. En effet, le rôle de l'ordinateur dépasse largement celui du calculateur puisqu'il met en ordre et gère de multiples activités.

1. Types d'objets en Python

Nombres, listes, t-uplets, chaînes de caractères, ensembles

Tous les objets que manipule PYTHON ont un type défini à la fois par l'espace mémoire utilisé pour le stocker mais également par les règles de représentation, d'opération ou encore de construction qui s'y appliquent.

Les différents types

On peut distinguer deux types d'objets, les simples et les composés.

Types simples

Il s'agit essentiellement de types numériques.

`bool`	un booléen peut être *True* ou *False*
`int`	un entier relatif est exact en machine
`float`	un flottant est une valeur approchée d'un réel
`complex`	un complexe est de la forme a+jb

Types composés

Comme leur nom l'indiquent, ces objets sont des collections d'objets simples.

• On peut déterminer si un objet simple `a` appartient ou non à un objet composé `A` au moyen de la relation « `a in A` » qui renvoie un booléen. Ces objets composés des **itérables**.

• Lorsque ces objets composés sont ordonnés, on peut accéder directement à leurs composants à l'aide de « A[k] » qui renvoie l'élément de rang k. Ils sont alors dits **indexables**.

• Finalement, si on peut directement modifier les objets simples qui les composent, on dit qu'ils sont **mutables**.

Le tableau ci-dessous donne les types composés les plus fréquemment utilisés.

list	une liste est un objet mutable, indexable, itérable
tuple	un t-uplet est non mutable, indexable, itérable
str	une chaîne de caractères est non mutable, indexable, itérable
set	un ensemble est non mutable, non indexable mais itérable

Les types simples

Les tableaux ci-dessous résument les différents opérateurs sur les objets simples : opérations arithmétiques, comparaisons, constructions ou conversions.

Opérations

a+b	somme des deux nombres (int,float,complex)
a - b	différence des deux nombres (int,float,complex)
a*b	produit des deux nombres (int,float,complex)
a/b	quotient de deux flottants
a//b	quotient de la division euclidienne d'entiers
a%b	reste de la division euclidienne d'entiers
divmod(a,b)	quotient et reste de la division euclidienne
a**b pow(a,b)	a élevé à la puissance b
abs(a)	valeur absolue ou module de a

Comparaisons et opérations logiques

a==b	test d'égalité, renvoie un booléen
a !=b	test de différence, renvoie un booléen
a>b, a<b, a>=b, a<=b	tous ces tests de comparaison, respectivement $a > b$, $a < b$, $a \geqslant b$, $a \leqslant b$ renvoient un booléen
P and Q	conjonction de deux booléens
P or Q	disjonction de deux booléens
not P	négation d'un booléen

Conversions

int(a)	convertit en entier un flottant ou une chaîne
float(a)	convertit en flottant un entier ou une chaîne
complex(a,b)	convertit en complexe un couple de flottants
int(car,b)	donne la valeur de l'entier donné par son écriture car (de type chaîne) en base b
bin(a)	renvoie l'écriture en binaire de l'entier a
oct(a)	renvoie l'écriture en octal de l'entier a
hex(a)	renvoie l'écriture en hexadecimal de l'entier a

Les types composés

Définitions

• Une **liste** est une suite d'éléments quelconques (pas nécessairement de même type) séparés par des virgules et encadrée par des crochets. Par exemple [a, b, c] est une liste. Les listes sont itérables, indexables et mutables.

• Un **t-uplet** est une suite d'éléments quelconques (et pas nécessairement de même type) séparés par des virgules et encadrée par des

parenthèses (ou rien). Par exemple `(a, b, c)` et `d, e, f` sont des t-uplets. Les t-uplets sont itérables, indexables mais non mutables.

• Un **ensemble** est une collection d'objets séparés par des virgules et encadrée par des accolades. Ainsi, `{a, b, c}` est un ensemble. Dans un ensemble, ni l'ordre, ni les répétitions éventuelles des éléments ne changent l'ensemble. Les ensembles sont itérables mais non indexables et non mutables.

• Une **chaîne de caractère** est une suite de caractères quelconques encadrée par une paire de simples ou de doubles guillemets. Par exemple, `"a b c"` et `'d e f'` sont des chaînes de caractères. Comme les listes, les chaînes de caractères sont itérables, indexables mais non mutables.

Les objets séquentiels que sont les listes, les t-uplets et les chaînes, sont toujours indexés à partir de 0. `A[0]` désigne le premier élément de la séquence, `A[1]` le deuxième, etc.

Opérations

`len(A)`	retourne la longueur de `A` (liste, t-uplet, chaîne, ensemble)
`A[k]`	accède à l'élément d'indice `k` de la séquence `A` (liste, t-uplet, chaîne)
`A + B`	opération de concaténation, juxtapose les deux séquences (liste,t-uplet,chaîne)
`n * A`	opération de duplication, juxtapose `n` (entier) fois la séquence `A` (liste,t-uplet,chaîne)
`max(A)`	retourne l'objet maximal de la collection `A` (liste, t-uplet, chaîne, ensemble)
`min(A)`	retourne l'objet minimal de la collection `A` (liste, t-uplet, chaîne, ensemble)
`sum(A)`	retourne la somme des éléments d'une collection `A` de nombres (liste, t-uplet, chaîne, ensemble)

Comparaisons entre objets

a in A	teste l'appartenance d'un objet à A (liste, t-uplet, chaîne, ensemble) et retourne un booléen
a not in A	teste la non appartenance d'un objet à A (liste, t-uplet, chaîne, ensemble) et retourne un booléen
A==B	teste l'égalité de A et B (listes, t-uplets, chaînes, ensembles), retourne un booléen
A != B	teste la non égalité de A et B (listes, t-uplets, chaînes, ensembles), retourne un booléen
A < B	test d'inégalité dans l'ordre lexicographique, lorsque A et B sont des séquences (liste, t-uplet, chaîne), retourne un booléen
A < B	test d'inclusion lorsque A et B sont des ensembles, retourne un booléen

Constructions et conversions

[a,b,c]	crée la liste d'objets a, b, c
[] ou list()	crée la liste vide
(a,b,c)	crée le t-uplet d'objets a, b, c
a,b,c	crée l'ensemble d'éléments a, b, c
set()	crée l'ensemble vide
"abc" ou 'abc'	crée la chaîne de caractères a, b, c
"" ou '' ou str()	crée la chaîne de caractères vide
list(A)	convertit en liste tout type d'objet composé (liste, t-uplet, chaîne, ensemble)
set(A)	convertit en ensemble. Élimine les répétitions et range les éléments par ordre croissant, si possible
str(A)	convertit en chaîne de caractères tout type d'objet composé (liste, t-uplet, chaîne, ensemble)

Variables

Notion de variable en Python

Pour mémoriser une valeur, on la stocke en mémoire. On représente l'adresse mémoire à l'aide d'un identificateur (chaîne de caractères). Le couple (identificateur, valeur) est appelé une **variable**.

Comme nom de la variable, on peut utiliser toute chaîne de caractères alphanumériques (qui ne commence pas par un chiffre) à l'exception de quelques mots réservés.

PYTHON distingue les majuscules des minuscules. Ainsi, les chaînes 'Aa' et 'aa' sont deux identificateurs différents.

Une variable peut contenir des données de n'importe quel type déjà présenté. En PYTHON, il n'est pas nécessaire de préciser le type de valeur que l'on va placer dans une variable. En fait, le type est déterminé seulement au moment de l'évaluation, c'est ce que l'on appelle un **typage dynamique**.

Affectation d'une valeur à une variable

Déclaration d'une variable

Avant d'utiliser une variable a, il faut toujours lui donner une valeur initiale. Pour attribuer une valeur à une variable, on utlise l'**opérateur d'affectation**, noté « = » en PYTHON.

```
>>> a = 10
```

Affectation d'une variable

Pour modifier la valeur d'une variable, on utilise l'opérateur d'affectation. On ne confondra pas cet opérateur avec une égalité mathématique. En effet, l'opérateur d'affectation est non symétrique, non transitif.

```
>>> x = 1
>>> x = x + a
```

Dans une affectation, on commence d'abord par évaluer l'expression de droite, puis on affecte cette valeur à la variable de gauche.

Affectations simultanées

PYTHON permet de réaliser, des affectations simultanées de plusieurs variables, de même type ou non :

```
>>> x,y,z = 'abc',6,[]
```

En particulier, il est aisé d'échanger le contenu de plusieurs variables en PYTHON :

```
>>> x,y,z = y,z,x
```

Opérations sur les variables

Comme une variable représente un objet d'un certain type, tous les opérateurs rencontrés (opérations, comparaisons, constructions ou conversions) s'appliquent également aux variables de ce même type.

On appelle **expression** toute opération composée qui met en jeu variables et constantes. Lorsque les composants sont de même type et les opérations bien définies, l'expression finale garde le même type. Lorsque les constantes et les variables sont de types hétérogènes, l'expression prend le type minimal dans lequel toutes les opérations sont compatibles.

Opérateurs avec assignation

Il est très fréquent d'avoir à incrémenter une variable, c'est-à-dire à augmenter sa valeur de 1. PYTHON permet de raccourcir l'instruction `x = x + 1` en `x += 1`. On dit que `+=` est un opérateur avec assignation.