Introduction à la programmation

Renaud Costadoat
Lycée Dorian

La programmation

Definition

- L'immense majorité des programmes qui s'exécutent sur nos ordinateurs, téléphones et autres outils électroniques sont écrits dans des langages de programmation dits impératifs : **les lignes du programme sont exécutées les unes après les autres**,
- Chaque ligne du programme effectue soit une opération simple, soit exécute une fonction qui est elle-même une suite d'opérations simples,
- Ces opérations sont des **instructions** ou des **expressions** qui manipulent des **variables**.

Les variables

Une variable permet de stocker des informations. Une variable est définie par :

- un identificateur,
- un type,
- une valeur,
- une référence.

Identificateur (nom de la variable) (lettres, chiffres, underscore)

Remarque

En python, les noms de variables suivants sont interdits :

and	as	assert	break	class	continue	def	True	try
del	elif	else	except	False	finally	for	while	
from	global	if	import	in	is	lambda	with	
None	nonlocal	not	or	pass	raise	return	yield	

```
>>> Booleen = True
>>> Entier = 2
>>> Reel = 3.456
>>> chaine = "coucou"
```


Les variables

Typage

Le typage correspond à la nature de la variable (*booléen, nombre entier, nombre réel, etc...*).

Le typage peut être:

- **statique** lorsqu'il est nécessaire de définir le type d'une variable lors de sa création. On parle de typage,
- **dynamique** lorsque, par exemple, le type le mieux adapté est choisi automatiquement lors de l'assignation d'une variable.

```
# type permet de determiner le type d'une variable
>>> type(nbBooleen)
   <class 'bool'>
```

Référence

La référence permet de créer un alias pointant directement vers l'adresse mémoire d'une variable.

Les variables

Opérations

Une opération est une combinaison arithmétique, relationnelle, logique,... de deux ou plusieurs variables. Le résultat dépend du type de variable.

Les principales opérations sont les suivantes :

- l'addition : `+`,
- la soustraction : `-`,
- la multiplication : `*`,
- l'exposant : `**`,
- la division : `/`,
- la division entière : `//`,
- le modulo : `%`,
- la comparaison : `==`.

Types de variables

- les entiers,
- les réels,
- les booléens,
- les chaînes de caractères.

```
>>> a = 64 # affectation d'un entier
>>> a = 64.64 # affectation d'un reel
>>> a = True # affectation d'un booleen
>>> a = "a" # affectation d'un caractere
>>> 0b1000000 # Conv. binaire>decimal
   64
>>> 0x40 # Conv. hexa. > decimal
   64
```

Les chaînes de caractères

Remarque

- En raison des différences d'encodages entre les différents systèmes d'exploitation, des problèmes peuvent se poser lors de l'affichage des caractères spéciaux tels les accents, les cédilles ...
- Séquences d'échappements :
  - `\n` provoque un retour à la ligne (retour chariot),
  - `\t` provoque une tabulation,
  - `\a` provoque une bip système,
  - `\"` et `\?` permettent d'écrire un guillemet sans ouvrir ou fermer une chaîne de caractère,
  - `\\` permet d'écrire un antislash.

```
>>> a = 75011 ; b = "Lycee Dorian"
>>> print(a,": \t",b)
   75011 Lycee Dorian
```


Les listes et les tableaux

Liste

Une liste est une collection de plusieurs éléments qui peuvent avoir un type différent.

```
>>> x=[1,"b",3,"coucou"] # On affecte une liste à la variable x
>>> x[0] # Accès (affichage) a une variable
>>> x[0:2] # Accès (affichage) de x[0] a x[1]
>>> x.append(5) # Ajouter un element en fin de liste
>>> del(x[2]) # Supprime la valeur 3 de la liste x
```

Expression

Une expression est l'évaluation d'une opération. Un résultat est retourné.

```
>>> 1+1
   2
>>> 'a'+'a'
   'aa'
>>> 1 == 1
   True
```


Instructions

Une instruction est une action utilisée dans un algorithme ou dans un programme. Une instruction peut inclure une expression.

```
>>> a = 1
   1
>>> a = 2*3 # on affecte à a le résultat de l'expression
   6
>>> print(a)
   6
>>> type(a)
   <class ' int '>
```

Remarque: il est impossible d'affecter une valeur à une expression, il est donc impossible de mettre une expression à gauche du signe =.

L'affectation

Affectation simple

À cette variable on fait correspondre un identificateur (nom de la variable), une valeur, un type (booléen, entier, flottant ...) et une adresse mémoire. Tant que la variable n'est pas réaffectée, l'adresse mémoire reste inchangée.

```
>>> a=2
>>> id(a) # Permet de connaitre l'adresse memoire
   15590464
```

Affectation multiple

L'affectation multiple permet l'affectation simultanément plusieurs variables.

```
>>> a,b=1,2                    >>> b
>>> a                             2
   1
```


Affectation multiple

Les variables comme les tableaux ne peuvent pas être copiées aussi simplement que les variables simples :

```
>>> a,b=1,2
>>> tab1=[a,b]
>>> tab2=tab1
>>> id(tab1);id(tab2)
   19282320
   19282320
```

Lors de la création de `tab2`, python n'a pas créé un nouvel espace mémoire. Il a juste créé la variable `tab2` et lui a adressé le même espace mémoire que `tab1`. En conséquence, si on change un champ de `tab1`, le même champ de `tab2` sera modifié.

Affectation multiple

En général, ce comportement n'est pas souhaité :

```
>>> tab1;tab2                    >>> tab1;tab2
   [1, 2]                           [0, 2]
   [1, 2]                           [0, 2]
>>> tab1[0]=0
```

Ainsi, une méthode spéciale permet de recréer une nouvelle variable avec son adresse mémoire.

```
>>> tab2=tab1.copy()             >>> tab1[0]=4;tab1;tab2
>>> id(tab1);id(tab2)               [4, 2]
   19282320                         [0, 2]
   20335832
```


Affectation externe

Lors de l'exécution d'un programme, il est possible de demander à l'utilisateur de saisir une donnée. Pour cela il existe des instructions permettant de communiquer avec l'utilisateur.

Dans Python, en utilisant la fonction input, les données saisies par l'utilisateur sont converties en chaîne de caractère.

```
>>>a=input()
```

Sorties à l'écran

Lors de l'exécution d'un programme il est souvent nécessaire que ce dernier renvoie des informations à l'utilisateur pour, par exemple, donner le résultat d'une opération ou encore donner l'avancement dans le programme.

```
>>> print("Coucou") # Afficher une chaine de caract.
   Coucou
>>> i = 2
# Afficher une phrase composee.
>>> print("La valeur de i est ", i ,".", sep="")
# sep=" permet de supprimer l'espace entre 2 et le point
   La valeur de i est 2.
```


Exemple de programme

L'algorithme d'Euclide

Savoir

1. Pour a, b deux entiers, on dit que b divise a si a s'écrit $k.b$ avec k un entier (ex : 3 divise 75),
2. Pour $a, b \in \mathbb{N}$, non tous deux nuls, on note $PGCD(a, b)$ le plus grand diviseur commun à a et à b, (ex : PGCD(75,40) = 5),
3. Pour tout entier naturel a, $PGCD(a, 0) = a$ (tout entier divise 0).

Soient a, b deux entiers naturels, avec b non nul. On note q le quotient et r le reste dans la division euclidienne de a par b.

On a : $a = b.q + r$, et on sait que $a = k.a_1$ et $b = k.b_1$, ainsi:
$k.a_1 = k.b_1.q + r$, avec a_1, b_1, k, q, r entiers naturels, ainsi, il existe un r_1 tel que $k.a_1 = k.b_1.q + k.r_1$, tel que $PGCD(a, b) = PGCD(b, r)$

Exemple de programme

Calcul de *PGCD*(5810, 1125)

1. .....,
2. .....,
3. .....,
4. .....

```
def Euclide_PGCD(a,b): # nom de la fonction et ses variables
   r=a%b           # on calcule le reste dans
                   # la division de a par b
   while r!=0:     # tant que ce reste est non nul :
         a=b       # b devient le nouveau a
         b=r       # r devient le nouveau b
         r=a%b     # on recalcule le reste
   return(b)       # une fois la boucle terminée,
                   # on retourne le dernier b
pgcd(5810,1125)  # resultat retourné par la fonction
```