Programmation : Fondamentaux

1 Objectif : programmer !

1.1 Notion d'algorithme

Définition

Un algorithme est une procédure se terminant en nombre *fini* d'étapes qui permet de résoudre une classe de problèmes, écrite de façon suffisamment *détaillée* pour être suivie par un humain ne possédant pas de compétence particulière et qui n'est même pas obligé de comprendre le problème qu'il est en train de résoudre.

Ainsi un algorithme fonctionne avec des *données* précisant le cas particulier du problème qu'il traite. En retour, il construit un *résultat* répondant à ce cas particulier du problème.

1.2 Notion de programme

Définition

Un *programme* est la traduction d'un algorithme dans un langage de programmation, qui est à la fois compréhensible pour l'homme et interprétable pour la machine. Il est constitué d'un assemblage d'instructions regroupées dans un fichier texte appelé *code source* du programme, qui sera exécuté par la machine dans un ordre bien défini appeléle *flot d'exécutions*.

1.3 Langage de programmation

Python est un langage de programmation, c'est à dire :

-
-
-

(nous utiliserons Python 3 dans notre cours)

Algorithme

Algorithme 1 : Test de parité

Données : $a \in \mathbb{N}^+$
Résultat : afficher si a est pair ou impair

```
1 a←valeur saisie par l'utilisateur
2 si reste(a/2)=0 alors
3 |  afficher a est pair
4 sinon
5 |  afficher a est impair
```

python

```
Test_parite.py
"""
Le programme prendra une valeur saisie
par l'utilisateur
et affichera s'il est pair ou impair.
"""
a=int(input("Saisir un entier strictement positif"))
if a%2==0 :
    print("a est pair")
else :
    print("a est impair")
```

2 Constantes et Variables

Lors de son exécution, un programme manipule des objets. Les plus simples sont les constantes (valeur connue non modifiable) : `12`, `12.3`, `-5.1`, `'bonjour'`, `True`, `False`.

Certains objets ont une ou des valeurs non constantes et/ou modifiables, appelés les variables. Une variable est définie par :

Définition

-
-
-

Définition

Identificateur

L'identificateur correspond au nom de la variable. Il doit être explicite. Pour nommer une variable, il est possible d'utiliser :

- les lettres de l'alphabet en minuscules ($\mathbf{a} \rightarrow \mathbf{z}$) ou en majuscules ($\mathbf{A} \rightarrow \mathbf{Z}$) ;
- des chiffres ($\mathbf{0} \rightarrow \mathbf{9}$) ;
- l'underscore _.

Le nom d'une variable commence par une lettre.

Définition

Typage

Le typage correspond à la nature de la variable (booléen, nombre entier, nombre réel *etc*). A chaque type est associé une convention de codage informatique.

On parle de typage 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 par le système informatique. Python appartient à cette catégorie.

2.1 Types simples

Ce sont les types les plus fondamentaux. Ils ne sont pas identiques dans tous les langages, mais présentent une base commune. En python, on retrouve notamment :

Les booléens :	type `bool`, deux états possible donc 1 bit
Les entiers relatifs :	type `int` sur au moins 64 bits
Les nombres décimaux :	type `float` (selon la norme IEEE754 double précision)
Les nombres complexes :	type `complex` codés sous la forme $a + bj$ où a et b sont deux nombres décimaux et j vérife $j^2 = -1$.

2.2 Application 1

Choisissez le type qui vous semble adapté pour représenter chacune des valeurs suivantes :

- la pointure d'un individu ;
- le nom d'un individu ;
- la valeur de pi ;
- le nombre d'êtres humains.

2.3 Types itérables ou collections

Il existe des types composés de valeurs de type simple. Ces types sont appelés itérables. Y figurent, par exemple, : les listes, les n-uplets ou *tuple*, les chaines de caractères, les dictionnaires, etc. Ils seront étudiés lors un cours dédié.

3 Instructions et expressions

Un programme est formé d'instructions. Une instruction est un morceau de code minimal qui est exécuté par la machine, et produit un effet. Une instruction mal rédigée (que la machine ne peut pas interpréter) engendre une erreur de syntaxe.

3.1 Défnitions

Définition

Expression

Une expression est une suite de caractères conduisant à l'évaluation d'un calcul. Un résultat est retourné.

Une expression peut utiliser des constantes ou des variables liées par des opérateurs ou des fonctions. A chaque type est associé des opérateurs et des fonctions spécifiques. Utiliser un opérateur inadéquat au type du reste de l'expression amène à une erreur.

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

```
>>> abs(-3)
3
>>> abs('PCSI')
TypeError: bad operand type for abs(): 'str'
```

Définition

Etat du programme

Un programme utilise un certain nombre de variable. Les valeurs de ces variables constituent l'**état du programme** (ou l'état de la mémoire).

Instruction

Une instruction produit un effet (par exemple un modification de l'état d'un programme). Une instruction peut inclure des expressions.

Instruction simple

Une instruction simple est une instruction rédigée sur une seule ligne. Une expression isolée ou une affectation sont des cas particuliers d'une instruction simple.

```
a = 1
a = 101**0.5
print(a)
del(a)
```

3.2 Déclaration et affectation

Un programme utilise des variables. Lors de son exécution, il doit définir l'identifiant, le type et la valeur de chacune des variables afin de pouvoir les gérer dans l'espace mémoire. Les deux premières opérations s'appellent déclaration et la troisième affectation.

En règle générale, en informatique, la déclaration s'effectue en début de programme. L'affectation peut avoir lieu n'importe quand.

Algorithme

Données : $a \in \mathbb{N}$; $b \in \mathbb{N}$
Résultat : Effectuer un calcul
1 a←22
2 $b \leftarrow a + a^2$

python

```
a = 22
b = a + a**2
```

Spécificité

Déclaration automatique

En python, les variables n'ont jamais un type définitif puisque le langage peut le modifier en cours d'exécution.

Affectation simultannée

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

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

L'affectation est une instruction, mais pas une expression.

Son résultat ou son effet n'est pas une valeur. Donc : `print(a=3)` entraine une erreur.

3.3 Application 2

Un programme est réalisé par l'unique ligne suivante :

```
x=x+1
```

1. x apparait deux fois. Expliquer la signification de chacune de ses apparitions.
2. Pourquoi ce programme ne fonctionne pas ? Que faut-il ajouter ?

3.4 Application 3

Proposer un algorithme qui inverse les valeurs de deux variables x et y.

3.5 Application 4

Décrire l'évolution de l'état du programme suivant pour chaque ligne de code, en précisant s'il s'agit d'une expression ou d'une instruction. L'état initial du programme est le suivant : b est défini comme un entier et sa valeur est 5.

python

```
1 a=2
2 a=b-7
3 b+1
4 b+=1
5 c=a-b
```

3.6 Entrées-sorties

On appelle entrées-sorties les opérations qui permetttent de communiquer avec un utilisateur.

- La commmande `print()` qui permet d'afficher à l'écran un texte ou la valeur d'une variable. Après un `print()`, le programme continue son exécution ;

- La commande `input()` interrompt un programme jusqu'à ce que l'utilisateur tape au clavier une séquence de caractères.

3.7 Flux d'instructions

Un programme se compose donc de lignes d'instructions qui s'exécutent les unes après les autres.

```python
# Fichier: calcul_sphere.py
# Outils importés
from math import sqrt,pi
# Dimensions en x/y/z de l'ellipsoïde.
dx = float(input("dx = "))
dy = float(input("dy = "))
dz = float(input("dz = "))
# Calcul du volume sphérique englobant:
r = sqrt(dx**2+dy**2+dz**2)
s = 4/3*pi*r**3
# Calcul du volume de l'ellipsoïde.
e = 4/3*pi*dx*dy*dz
# Rapport entre les deux:
es = e / s
# Affichage du résultat
print("ellipsoïde:",e,"sphère:",s,"rapport:",es)
```

3.8 Instructions composées ou complexes

On appelle instructions composées ou complexes :

- les instructions qui modifie l'ordre d'exécution d'un programme (sous certaines conditions) ;
- s'écrivent sur plusieurs lignes ;
- les instructions simples composant une instruction composée sont regroupées sous forme de blocs.

```
ligne d'en-tête :
    instruction 1
    instruction 2
    instruction 3
```

Algorithme

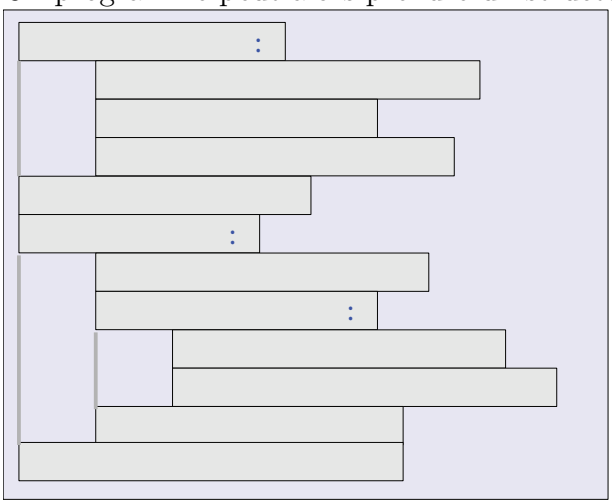
Données : $a \in \mathbb{N}$
Résultat : afficher la moitié d'un nombre pair

```
1 a←25
2 si reste(a/2)=0 alors
3 |  b←a/2
4 |  afficher a est pair
5 |  afficher sa moitié vaut b
6 sinon
7 |  afficher a est impair
```

python

```
1 a=25
2 if a%2==0 :
3     b=a/2
4     print("a est pair")
5     print("sa moitiée vaut ",b)
6 else :
7     print("a est impair")
```

Un programme peut alors prendre un structure à plusieurs niveaux :



4 Expressions : manipulation des nombres

4.1 Opérateurs sur les entiers et les décimaux

Les opérateurs sur les entiers et les décimaux sont identiques et résumés dans le tableau suivant :

Opérateur	Opération
+	Addition
-	Soustraction
*	Multiplication
/	Division flottante
//	Division entière
%	Reste de la division entière
-x	Opposé en préfixe
abs(x)	Valeur absolue
**	Exponentiation (puissance)

4.2 Précédence

La question de l'ordre dans lequel sont évaluées les expressions qui comprennent plusieurs opérateurs sans parenthèses (par ex. 2+3*4) fait l'objet de règles généralisant celles sur les priorités opératoires. On parle de précédence. Concrètement, en l'absence de parenthèses qui sont toujours évaluées en premier, on évalue dans l'ordre :

1. les exponentiations
2. les multiplications, divisions entières et modulos
3. les additions et les soustractions

En présences de fonctions, sont d'abord évaluées :

1. les expressions qui sont des paramètres des fonctions appellées,
2. les résutats des fonctions elles-mêmes,
3. enfin les règles de précédence sont appliquées.

En cas de doute :

- les expressions sont évaluées de gauche à droite,
- sauf pour pour l'exponentiation, qui est évaluée de droite à gauche.

4.3 Quelques raccourcis

Spécificité

```
a+=1                    a=a+1
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
```