CI 2 : Algorithmique & Programmation

Chapitre 1 – Introduction à la programmation

Savoir

Savoirs :
- Variables : notion de type et de valeur d'une variable, types simples
- Expressions et instructions simples : affectation, opérateurs usuels, distinction entre expression et instruction

Ce document évolue. Merci de signaler toutes les erreurs ou coquilles rencontrées.

1 Variables

1.1 Exemple d'algorithme : L'algorithme d'Euclide

Rappels

1. Pour a, b deux entiers, on dit que b divise a si a s'écrit kb avec k un **entier** (ex : 5 divise 45).
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(45,30) = 15).
3. Pour tout entier naturel a, PGCD(a,0) = a (tout entier divise 0).

Propriété

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

On a : PGCD(a,b) = PGCD(b,r)

Exemple

Calcul de PGCD(1525, 755).

Exemple

$$1525 = 755 \times 2 + 15 \quad \text{Donc } PGCD(1525, 755) = PGCD(755, 15)$$
$$755 = 15 \times 50 + 5 \quad \text{Donc } PGCD(755, 15) = PGCD(15, 5)$$
$$15 = 5 \times 3 + 0 \quad \text{Donc } PGCD(15, 5) = PGCD(5, 0) = 5$$

Remarque

1. Le procédé précédent est un algorithme : «ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations».
2. Pour coder des algorithmes nous utiliserons :
   (a) des variables (ici a, b, r) qui auront des valeurs assignées ;
   (b) des instructions : des affectations, des opérations,... ;
   (c) des expressions logiques : ici dire si le reste est nul ou non ;
   (d) des boucles : (ici une boucle Tant que) pour répéter des instructions ;
   (e) une rédaction sous forme de fonction : pour pouvoir appeler cet algorithme à d'autres endroits de notre programme.

Codage en Pythonde l'algorithme d'Euclide :

```
def Euclide_PGCD(a,b): # on définit le nom de la
                       # fonction et ses variables
                       #d'entrées/d'appel
    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
print(pgcd(1525,755))  # on affiche le résultat
                       # retourné par la fonction
```

Pseudo Code

Fonction PGCD : algorithme d'Euclide

Données : a et b : deux entiers naturels non nuls tels que a > b
Résultat : le PGCD de a et b

Euclide_PGCD(a,b)
  Répéter
    r ← a mod b
    a ← b
    b ← r
  Jusqu'à r == 0
  Retourner a

A la nième itération, on a la relation a $a_n = q_n \times b_n + r_n$ où q_n est un entier.

1.2 Définitions

Définition

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

- un identificateur (nom de la variable) ;
- un type ;
- une valeur ;
- une référence ;
- des opérations.

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 (**a** → **z**) ou en majuscules (**A** → **Z**) ;
- des chiffres (**0** → **10**) ;
- l'underscore _.

Le nom d'une variable commence par une lettre.

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	

Exemple

Lister les identificateurs utilisés dans l'algorithme d'Euclide.

Définition

Affectation

L'affectation permet d'assigner une valeur à une variable.

Exemple

Pseudo Code

```
nbBooleen ← True
nbEntier ← 2
nbReel ← 3.456
chaine ← "coucou"
```

python

```
nbBooleen = True
nbEntier = 2
nbReel = 3.456
chaine = "coucou"
```

Scilab

```
nbBooleen =%T
nbEntier =2
nbReel =3.456
chaine ="coucou"
```

Définition

Typage

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

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 lors de l'assignation d'une variable.

Exemple

python

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

Définition

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

Définition

Opérations
Une opération est une combinaison arithmétique 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 valeur absolue : **abs**.

1.3 Types de variables

1.3.1 Types simples

Définition

- les entiers
- les réels
- les booléens
- les caractères

Exemple

python
```
>>> 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
```

Scilab
```
-->a =64 // affectation d'une constante
-->a =int8(64) // aff. un entier sur 8 bits
-->a =int16(64) // aff. un entier sur 16 bits
-->a =%T // affectation d'un booleen
-->a ="a" // affectation d'un caractere
```

Remarque

En programmation, le type entier (int – integer) désigne les entiers **relatifs**.

Exemple

Il est aisé de convertir des nombres depuis une base n vers la base décimale :

python
```
>>> 0b1000000 # Conv. binaire>decimal
        64
>>> 0x40 # Conv. hexa. > decimal
        64
```

1.3.2 Les chaînes de caractères

Définition

Les chaînes de caractères sont une succession 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 ...

Remarque

Séquences d'échappements L'utilisation d'un antislash
dans une chaîne de caractère peut entraîner un comportement particulier de cette chaîne de caractère :

- **\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.

Exemple

```
>>> a = 64 ; b = "Pyrenees Atlantiques"
>>> print(a,": \t",b)
        64          Pyrenees  Atlantiques
```

1.3.3 Les listes et les tableaux

Définition

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

Exemple

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

1.3.4 Les dictionnaires

Définition

Les dictionnaires sont des collections de clés auxquelles sont associées des valeurs.

Exemple

```
>>> dep = {"Ain":1}
>>> dep["Aisne"]=2
>>> print(dep)
        {'Aisne': 2, 'Ain': 1}
>>> print(dep['Ain'])
        1
```

2 Expressions et instructions simples

2.1 Expressions

Définition

Expression
Une expression est l'évaluation d'un calcul. Un résultat est retourné.

Exemple

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

```
-->1+1
        ans = 2
-->'a'+'a'
        ans = aa
-->1 ==1
        ans = T
```

2.2 Instructions

Définition

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

Exemple

Pseudo Code

$a \leftarrow 1$
Afficher(a)
Type_De(a)

```
>>> a = 1
        1
>>> print(a)
        1
>>> type(a)
        <class 'int'>
```

```
-->a =1
        ans = 1.
-->mprintf("%f",a)
        1.0000
-->typeof(a)
        ans = constant
```

Remarque

En python, les expressions ou les instructions sont séparés par des ; ou par des retours à la ligne.

2.3 L'affectation

2.3.1 L'affectation simple

On a vu précédemment, qu'il était possible d'affecter assez simplement une valeur à une variable. Lors d'une affectation, un espace mémoire est réservé dans la mémoire de l'ordinateur. Cet espace mémoire est situé à une certaine adresse.

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.

Remarque

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

Exemple

Affectation simple

Lorsqu'on copie une variable simple, un nouvel espace mémoire avec une nouvelle adresse est créé.

Pseudo Code

$a \leftarrow 1$
$b \leftarrow a$

```
>>> a=2
>>> id(a)
    15590464
```

2.3.2 L'affectation multiple

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

Exemple

Affectation multiple

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

2.3.3 Problèmes liés à l'affectation de variables composites

Exemple

Copie de variables composites

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. En général, ce comportement n'est pas souhaité :

Exemple

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

Ainsi, pour copier un tableau, une liste ou un dictionnaire, il est nécessaire d'utiliser une méthode spéciale qui permettra de recréer une nouvelle variable avec une nouvelle adresse mémoire. l

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

Remarque

En Python, le passage des éléments se fait toujours par référence. Il faudra donc prendre garde à la manipulation des variables, notamment lorsqu'on leur applique des fonctions.

2.3.4 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.

Exemple

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

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

```
-->a=input("Saisir un nombre : ")
```

2.4 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.

Exemple

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

```
print(%io(2),a)      // affiche le contenu de la variable a a l ecran
write(%io(2),a)     // fonction similaire
disp(a)               // affiche le contenu de a sans faire figurer a =
xinfo('message')  // affiche un message dans la barre d information
```

2.5 Gestion des exceptions

Certaines parties de programmes sont susceptibles de produire des erreurs. C'est par exemple le cas d'une entrée-sortie hasardeuse (saisie au clavier, lecture d'un fichier volumineux) ou bien de l'utilisation d'une opération instable dont le résultat peut provoquer un dépassement…

Pour gérer cette situation, on peut utiliser une structure : try – except en Python ou try – catch avec Scilab.

except permet de spécifier une action de remplacement en cas d'erreur.

```
try :
    nombre = int(input("Sasir un nombre : "))
    print ("Le carré de", nombre, "est égale à", nombre*nombre)
except :
    print("Vous n'avez pas donné de nombre correct, nous ne pouvons donc pas donner le carré")
```

try fonctionne toujours avec except.

Ce programme demande donc un nombre et affiche son carré. Mais s'il y a eu une erreur lors de la récupération de ce nombre, il affiche qu'il est impossible de donner le carré.

Une solution plus complète serait de redemander un nombre tant que la valeur saisie n'est pas un nombre. Pour cela on peut implémenter les lignes de code suivantes :

```
nombreIncorrect = True
while nombreIncorrect == True :
    try :
        nombre = int(input(" Saisir un nombre : "))
        nombreIncorrect = False
    except :
        print ("Vous n'avez pas donné de nombre correct, nous ne pouvons donc pas donner le carré")
print ("Le carré de", nombre, "est égale à", nombre*nombre)
```

On crée ici une variable ayant la valeur True pour indiquer que la proposition effectuée est incorrecte. Tant que l'utilisateur ne saisit pas de nombre, nombreIncorrect reste égal à True ; on continue donc à demander un nombre et on spécifie l'erreur à l'utilisateur.

Comme on a initialisé la variable à True, on entre dans la boucle. Si l'utilisateur saisi réellement un un nombre, on indique avec "nombreIncorrect = False" que l'on ne doit pas refaire un tour de boucle. La question n'est donc pas reposée.

On sort donc de la boucle lorsqu'on est sûr que la variable nombre contient un nombre. On peut donc afficher le carré.

On peut aussi utiliser la méthode montrée en exemple dans la documentation de python :

```python
while True :
    try :
        nombre = int(input("Un nombre s'il vous plait : "))
        break
    except :
        print ("Vous n'avez pas donné de nombre correct, nous ne pouvons donc pas donner le carré")
print ("Le carré de", nombre, "est égale à", nombre*nombre)
```

While True : ne s'arrête jamais, c'est une boucle infinie. On demandera toujours un nombre, sauf si on arrête la boucle avec break. Comme le break est dans le try, la boucle s'arrêtera seulement s'il n'y a pas d'erreur, c'est-à-dire que le nombre n'est pas incorrect.

Les exceptions permettent donc de gérer, dans un certaine mesure, les erreurs.

De la même manière, sous scilab :

```
try
        <instructions "normales">
catch
        <instructions executees en cas d'erreur>
end
[message_erreur, numero_erreur] = lasterror (%t) // enregistrement de l erreur
<suite du script>
```

Scilab exécute le code entre les commandes try et catch ; si aucune erreur ne se produit, il va tout de suite après le end. Si une erreur se produit entre le try et le catch, il exécute directement les instructions entre le catch.

3 Notions de programmation orientée objets

Définition

Classes, objets et méthodes Une classe est une structure particulière de programmation. Par instanciation d'une classe, il est alors possible de créer des objets.

Une classe définit les attributs et des méthodes qui pourront être appliquées à l'objet.

Exemple

Nous allons créé une classe permettant de gérer des points dans $\mathbb{R}^3$.

```python
class Point3d(object):
        """Creation d'un point de l'espace"""

p = Point3d()
```

Définition

Un attribut est une donnée propre à l'objet.

Exemple

Le point p a comme attribut ses 3 coordonnées. On pourrait donc créer un point à partir de ses coordonnées.

```python
class Point3d(object):
    """ Point de l'espace $\mathbb{R}^3$"""
    def __init__(self,coordx,coordy,coordz):
        """Creer un point a partir de 3 coordonnees x, y et z"""
        self.x = coordx
        self.y = coordy
        self.z = coordz

>>> p = Point3d(1,2,3)
>>> p.x
        3
```

Définition

Méthodes

Exemple

Le point p a comme attribut ses 3 coordonnées. On pourrait donc créer un point à partir de ses coordonnées.

```python
import math

class Point3d(object):
    """ Creation d'un point de l'espace"""

    def __init__(self,coordx,coordy,coordz):
        """Creer un point a partir de 3 coordonnees x, y et z"""
        self.x = coordx
        self.y = coordy
        self.z = coordz

    def distance(self,pt):
        """ Calcule la distance entre 2 points"""
        dist = math.sqrt((self.x-pt.x)**2
                         +(self.y-pt.y)**2
                         +(self.z-pt.z)**2)
        return dist

>>> p1=Point3d(0,0,0)
>>> p2=Point3d(1,1,1)
>>> p1.distance(p2)
        1.7320508075688772
```

Références

[1] *Gérard Swinnen*, Apprendre à programmer avec Python 3.

[2] *Robert Cordeau*, Introduction à Python 3.

[3] *Patrick Beynet*, Cours d'informatique en CPGE, Lycée Rouvière Toulon, UPSTI.

[4] *Adrien Petri et Laurent Deschamps*, Cours d'informatique en TSI 1, Lycée Rouvière Toulon.