PYTHON → JAVA

Memento de comparaison des syntaxes

K. Bouyarmane – INF 311 – Ecole Polytechnique

LEGENDE

```
Code python
```

```
Code Java équivalent
```

HELLO WORLD!

```
print("Hello world!")
```

```
public class MaClasse{
    public static void main(String[] args){
        System.out.println("Hello world!");
    }
}
```

COMMENTAIRES

```
# ligne commentée 1
# ligne commentée 2
# ligne commentée 3
```

```
""" Block commenté
Block commenté suite
Block commenté suite """
```

```
"""
Block commenté
"""
```

```
// ligne commentée 1
// ligne commentée 2
// ligne commentée 3
```

```
/* Block commenté
Block commenté suite
Block commenté suite */
```

```
/*
Block commenté
*/
```

OPERATEURS ARITHMÉTIQUES, LOGIQUES ET COMPARAISONS

```
True
False
a and b
a or b
not a
x==y
x!=y
x>y
x>=y
x<y
x<=y
x**y
1%2 (1)
1/2 (0.5)
1//2 (0)
```

```
true
false
a && b
a || b
!a
x==y
x!=y
x>y
x>=y
x<y
x<=y
Math.pow(x,y)
1%2 (1)
1/2 (0)
1.0/2 (0.5)
1/2.0 (0.5)
1.0/2.0 (0.5)
```

équivalence Python valable uniquement pour les types primitifs (int, double, boolean, char, etc), et pour les String dans certains cas particuliers (non instanciés par des new) pour les String en général, objets, tableaux, l'égalité de contenu se teste spécifiquement (cf methode **equals**)

VARIABLES, DECLARATIONS, TYPAGE STATIQUE

```
i=3
i=i+1
i="hello"
```
tout est ok

On ne déclare pas le type des variables
On peut affecter à une variable une valeur de n'importe quel type, même si elle contenait une valeur d'un type différent

```
i=3;
i=i+1;
i="hello";
```
i non déclarée (on ne connaît pas son type), une variable doit être déclarée une et une seule fois avec son type

```
int i=3;
i=i+1;
i="hello";
```
i déclarée

i de type int, conversion impossible

```
int i=3;
int i=i+1;
i="hello";
```
i déjà déclarée, on ne déclare pas deux fois une variable, une variable doit être déclarée une et une seule fois

```
int i;
i=3;
i=i+1;
i="hello";
```
On peut séparer la déclaration d'une variable et son initialisation, i aura une valeur même si elle n'est pas initialisée (valeur par défaut, dépendant de son type)

```
int i=3;
i=i+1;
String i="hello";
```
i déjà déclarée, on ne peut pas redéclarer une variable avec un autre type

```
int i=3;
i=i+1;
int i=4;
```
i déjà déclarée

```
int i=3;
i=i+1;
String s="hello";
```
tout est ok

LIGNES DE CODE, BLOCS DE CODE

```
<ligne de code>                                    4 blocs
<ligne de code>:
      <ligne de code>
      <ligne de code>:
            <ligne de code>
            <ligne de code>
      <ligne de code>
      <ligne de code>
<ligne de code>
<ligne de code>:
      <ligne de code>
      <ligne de code>
      <ligne de code>
      <très (longue ligne de
            code>)
      <très longue ligne de \
            code>
```

Une ligne de code se termine par un retour à la ligne
Un bloc de code commence par une indentation (suivant une ligne se terminant par un double point) et de termine par la fin de l'indentation
le retour à la ligne et l'indentation sont significatifs
un défaut d'indentation ou de retour d'indentation change le sens du programme
Un retour à la ligne dans la même instruction doit être explicite (caractère '\') ou implicite (à l'intérieur d'une expression délimitée par des parenthèses, crochets, ou accolades)
Les blocs de code délimitent les corps des classes, des fonctions, des structures if else for et while

```
<ligne de code>;                                   4 blocs
<ligne de code>{
      <ligne de code>;
      <ligne de code>{
            <ligne de code>;
            <ligne de code>;
      }
      <ligne de code>;
      <ligne de code>;
}
<ligne de code>;
<ligne de code>{
      <ligne de code>;
      <ligne de code>;
      <ligne de code>;
      <ligne de code>;
}
```

Les instructions sont délimitées par des points-virgules
Un bloc de code est délimité par une accolade ouvrante et une accolade fermante
Les blocs de code délimitent les corps des classes, des fonctions, des structures if else for et while
Les indentations et les retours à la ligne ont une utilité de lisibilité du code exclusivement

```
<ligne de code>; <ligne de code>{
<ligne de code>;
<ligne de code>
{
                  <ligne de code>;
<ligne de code>;}
      <ligne de
code>; <ligne de
            code>;}
            <ligne de code>;
<ligne de code>{<ligne de code>; <ligne
de code>; <ligne de code>;
<ligne de code>;
}
```

code strictement équivalent contenant 4 blocs

Les instructions sont délimitées par des points-virgules
Un bloc de code est délimité par une accolade ouvrante et une accolade fermante
Les retours à la ligne et les indentations ne sont pas significatifs
on peut exceptionnellement omettre les accolades pour les blocs de code (if else, while, for) qui contiennent une seule instruction, la fin du bloc est alors marquée par le premier point-virgule (cf exemple slide boucle while)

FONCTIONS, METHODES

```python
def somme(a,b):
    return a+b
```

```python
def sommeCarres(a,b):
    c=a**2
    d=b**2
    return c+d
```

```python
def estPair(a):
    if(a%2==0):
        return True
    else:
        return False
```

```python
def disBonjour():
    print("Bonjour")
```

le mot-clé def est utilisé pour définir une fonction, pas de type de retour et pas de type d'argument dans la déclaration des fonctions
attention aux indentations, le corps de la fonction est terminé dès qu'on finit l'indentation

```java
int somme(int a, int b){
    return a+b;
}
```

```java
double sommeCarres(double a, double b){
    double c=a*a;
    double d=b*b;
    return c+d;
}
```

```java
boolean estPair(int a){
    if(a%2==0){
        return true;
    }
    else{
        return false;
    }
}
```

```java
void disBonjour(){
    System.out.println("Bonjour");
}
```

Une fonction a un type de retour (void si pas de retour) et un type pour chacun de ses arguments
(attention les fonctions de cet exemple devront très probablement être déclarées **static** dans leur logique d'utilisation, cf détails classes et INF 311

CONVERSION DE TYPE (CAST)
CONVERSION EN STRING
CONCATENATION DE STRINGS

```
x = int(3.14)
y = float(3)
s = str(4)
t = str(2.5)
```

```
int x = (int) 3.14;
double y = (double) 3;
//double y = 3;
String s=""+4;
String t=""+2.5;
```

```
s = str(20) + "éléments"
t = "20"+ "éléments"

# x et y deux objets
u = str(x)
v = str(x) + str(y)
```

```
String s= 20+"éléments";

// x et y deux objets
String u = x.toString();
String v = x.toString()+y.toString();
String t = ""+ x + y;
```

ENTREES ET SORTIES TERMINAL

```
# entrer un string
nom = input("Entrer nom : ")

# entrer un entier
age = int(input("Entrer age : "))
```

```
// entrer un String en utilisant la classe TC
TC.print("Entrer nom : ");
String nom = TC.lireMot();

// entrer un entier en utilisant la classe TC
TC.print("Entrer age : ");
int age = TC.lireInt();
```

Sans la classe TC il est possible de faire des entrées avec la classe Scanner

```
# affichage avec retour à la ligne
print("Hello world!")

# affichage sans retour à la ligne
print("Hello world!", end = "")
```

```
# affichage avec retour à la ligne
System.out.println("Hello world!");

# affichage sans retour à la ligne
System.out.print("Hello world!");
```

WHILE

```
…
<avant la boucle>
while <condition>:
    <instruction 1>
    …
    <instruction n>
<sortie de la boucle>
…
```

```
…
<avant la boucle>;
while(<condition>){
    <instruction 1>;
    …
    <instruction n>;
}
<sortie de la boucle>;
…
```

```
…
<avant la boucle>;
while(<condition>){
<instruction 1>;
…
<instruction n>;
}
<sortie de la boucle>;
…
```

indentation optionnelle, mais fortement recommandée pour la lisibilité, ce qui délimite la boucle c'est non pas l'indentation du code mais les accolades

```
…
<avant la boucle>;
while(<condition>)
   <instruction 1>;
<sortie de la boucle>;
…
```

les accolades peuvent être omises en cas d'une seule instruction (non recommandé pour la lisibilité), indentation optionnelle ici aussi.

```
…
<avant la boucle>;
while(<condition>)
   <instruction 1>;
   <sortie de la boucle>;
   …
   <instruction n>;
…
```

En cas d'absence d'accolades seule la première instruction après le while (jusqu'au premier point-virgule) fait partie de la boucle, malgré les indentations

FOR

```
for <variable> in <iterable>:
    <instruction 1>
    …
    <instruction n>
```

```
for <variable> in range(<max>):
    <instruction 1>
    …
    <instruction n>

for <variable> in range(<min>, <max>):
    <instruction 1>
    …
    <instruction n>

for <variable> in range(<min>, <max>, <step>):
    <instruction 1>
    …
    <instruction n>
```

```
for(<initialisation>; <condition de continuation>; <update>){
    <instruction 1>;
    …
    <instruction n>;
}
```

utilisation courante

```
for(int i=0; i<10; i++){
    <instruction 1>;
    …
    <instruction n>;
}
<i n'existe pas en dehors de la boucle>
```

```
int i;
for(i=0; i<10; i++){
    <instruction 1>;
    …
    <instruction n>;
}
<i existe en dehors de la boucle>
```

```
int i=0;
for( ; i<10; i++){
    <instruction 1>;
    …
    <instruction n>;
}
```

```
int i=0;
for( ; i<10; ){
    <instruction 1>;
    …
    <instruction n>;
    i++;
}
```

```
int i=0;
for( ; ; ){
    if(!(i<10))
        break;
    <instruction 1>;
    …
    <instruction n>;
    i++;
}
```

FOR

```
for i in range(20):
    <instruction 1>
    …
    <instruction n>

for i in range(30, 50):
    <instruction 1>
    …
    <instruction n>

for <variable> in range(50, 100, 10):
    <instruction 1>
    …
    <instruction n>
```

```
for(int i=0; i< 20; i++){
    <instruction 1>;
    …
    <instruction n>;
}

for(int i=30; i<50; i++){
    <instruction 1>;
    …
    <instruction n>;
}

for(int i=50; i<100; i+=10){
    <instruction 1>;
    …
    <instruction n>;
}
```

IF … ELSE …

```
if <condition 1>:
     <instruction 1>
     …
     <instruction n>
elif <condition 2>:
     <instruction 1>
     …
     <instruction m>
else:
     <instruction 1>
     …
     <instruction k>
```

```
if(<condition 1>){
    <instruction 1>;
    …
    <instruction n>;
}
else if(<condition 2>){
    <instruction 1>;
    …
    <instruction m>;
}
else{
    <instruction 1>;
    …
    <instruction k>;
}
```

TABLEAUX

```
t = [45, 56, 67]

print( len(t) )
print( t[2] )
```

```
t = []

for i in range(10):
   t.append(i)
```

```
int[] t = new int[]{45, 56, 67};

System.out.println( t.length );
System.out.println( t[2] );
```

```
int[] t = new int[10];
//int t[] = new int[10];

for(int i=0; i<10; i++){
   t[i]=i;
}
```

EGALITE DE TABLEAUX

```
t1 = [1,2,3]
t2 = [1,2,4]

print(t1==t2)     //False
print(t1 is t2)   //False


t2[2]=3

print(t1==t2)     //True
print(t1 is t2)   //False
```

```
int[] t1 = new int[]{ 1, 2, 3 };
int[] t2 = new int[]{ 1, 2, 4 };

System.out.println(t1 == t2);                //false
System.out.println(t1.equals(t2));           //false
System.out.println(Arrays.equals(t1, t2));   //false

t2[2] = 3;

System.out.println(t1 == t2);                //false
System.out.println(t1.equals(t2));           //false
System.out.println(Arrays.equals(t1, t2));   //true
```

CLASSES – STRUCTURE

```
class <nom>(<nom de superclasse>):
    <variables de classe>
    <méthodes de classe>
    <methodes d'objet>
```

Les variables d'objet sont définies implicitement via le constructeur

```
<visibilité> class <nom> extends <nom de superclasse> implements <liste de noms>{
    <variables de classe>
    <méthodes de classe>
    <variables d'objet>
    <méthodes d'objet>
}
```

CLASSES – EXEMPLE

définition

```
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(self): return self.nom

    def getNote(self, i):
        return self.notes[i - 1]

    def setNote(self, i, nouvelleNote):
        self.notes[i - 1] = nouvelleNote

    def __str__(self):
        """Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        """
        resultat = self.nom + '\n'
        resultat += ''.join(map(str, self.notes))
        return resultat
```

utilisation

```
e = Etudiant('Dupont')
for i in range(1, Etudiant.NB_NOTES + 1)
    e.setNote(i, 100)
print e
```

définition

```
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public String getNom(){
        return this.nom;
    }

    public int getNote(int i){
        return this.notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        this.notes[i - 1] = nouvelleNote;
    }

    public String toString(){
        /*
        Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        */
        String resultat = this.nom + '\n';
        for (int note : this.notes)
            resultat += note + ' ';
        return resultat;
    }
}
```

utilisation

```
Etudiant e = new Etudiant("Dupont");
for (int i = 1; i <= Etudiant.NB_NOTES; i++)
    s.setNote(i, 100);
System.out.println(e);
```

CLASSES – VARIABLES D'OBJET

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(self): return self.nom

    def getNote(self, i):
        return self.notes[i - 1]

    def setNote(self, i, nouvelleNote):
        self.notes[i - 1] = nouvelleNote

    def __str__(self):
        """Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        """
        resultat = self.nom + '\n'
        resultat += ''.join(map(str, self.notes))
        return resultat
```

les variables d'objet sont définies implicitement dans le constructeur

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public String getNom(){
        return this.nom;
    }

    public int getNote(int i){
        return this.notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        this.notes[i - 1] = nouvelleNote;
    }

    public String toString(){
        /*
        Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        */
        String resultat = this.nom + '\n';
        for (int note : this.notes)
            resultat += note + ' ';
        return resultat;
    }
}
```

les variables d'objet sont déclarées explicitement indépendamment des constructeurs

CLASSES – VARIABLES D'OBJET - CONSTRUCTEURS

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(self): return self.nom

    def getNote(self, i):
        return self.notes[i - 1]

    def setNote(self, i, nouvelleNote):
        self.notes[i - 1] = nouvelleNote

    def __str__(self):
        """Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        """
        resultat = self.nom + '\n'
        resultat += ''.join(map(str, self.notes))
        return resultat
```

Le constructeur est la fonction __init__ qui prend self comme premier argument

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public String getNom(){
        return this.nom;
    }

    public int getNote(int i){
        return this.notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        this.notes[i - 1] = nouvelleNote;
    }

    public String toString(){
        /*
        Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        */
        String resultat = this.nom + '\n';
        for (int note : this.notes)
            resultat += note + ' ';
        return resultat;
    }
}
```

Un constructeur est une méthode qui n'a pas de type de retour et qui a le même nom que la classe

CLASSES – SURCHARGE DE CONSTRUCTEUR

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom=''):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(self): return self.nom

    def getNote(self, i):
        return self.notes[i - 1]

    def setNote(self, i, nouvelleNote):
        self.notes[i - 1] = nouvelleNote

    def __str__(self):
        """Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        """
        resultat = self.nom + '\n'
        resultat += ''.join(map(str, self.notes))
        return resultat
```

La surcharge de constructeur se fait en donnant des valeurs par défaut aux arguments

```python
e1 = Etudiant('Dupont')
e2 = Etudiant()
```

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public Etudiant(){
        this("");
    }

    public String getNom(){
        return this.nom;
    }

    public int getNote(int i){
        return this.notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        this.notes[i - 1] = nouvelleNote;
    }

    public String toString(){
        /*
        Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        */
        String resultat = this.nom + '\n';
        for (int note : this.notes)
            resultat += note + ' ';
        return resultat;
    }
}
```

utilisation

La surcharge de constructeur se fait en redéfinissant un autre constructeur avec un jeu d'arguments différent

```java
Etudiant e1 = new Etudiant("Dupont");
Etudiant e2 = new Etudiant();
```

CLASSES – METHODES D'OBJET

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(self): return self.nom

    def getNote(self, i):
        return self.notes[i - 1]

    def setNote(self, i, nouvelleNote):
        self.notes[i - 1] = nouvelleNote

    def __str__(self):
        """Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        """
        resultat = self.nom + '\n'
        resultat += ''.join(map(str, self.notes))
        return resultat
```

le premier argument des méthodes d'objet fait référence à l'objet appelant de la méthode
On l'a appelé self ici (par convention), mais on peut l'appeler comme on veut

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public String getNom(){
        return this.nom;
    }

    public int getNote(int i){
        return this.notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        this.notes[i - 1] = nouvelleNote;
    }

    public String toString(){
        /*
        Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        */
        String resultat = this.nom + '\n';
        for (int note : this.notes)
            resultat += note + ' ';
        return resultat;
    }
}
```

this fait référence à l'objet appelant, this suivi de . est toujours optionnel (this devient nécessaire lorsqu'on veut faire référence à l'objet en entier et non pas à une de ses variables ou méthodes d'objet, e.g. dans constructeur du slide précédent). this est défini implicement, il n'a pas d'autre noms

CLASSES – METHODES D'OBJET

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(aaa, nom):
        aaa.nom = nom
        aaa.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(bbb): return self.nom

    def getNote(ccc, i):
        return ccc.notes[i - 1]

    def setNote(this, i, nouvelleNote):
        this.notes[i - 1] = nouvelleNote

    def __str__(self):
        """Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        """
        resultat = self.nom + '\n'
        resultat += ''.join(map(str, self.notes))
        return resultat
```

code valable (non recommandé pour la lisibilité)
en particulier, rien ne nous empêche d'appeler ce premier argument this
il est fortement recommandé de conserver le nom self

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        nom = nom;
        notes = new int[NB_NOTES];
    }

    public String getNom(){
        return nom;
    }

    public int getNote(int i){
        return notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        notes[i - 1] = nouvelleNote;
    }

    public String toString(){
        /*
        Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        */
        String resultat = this.nom + '\n';
        for (int note : notes)
            resultat += note + ' ';
        return resultat;
    }
}
```

code valable (souvent utilisé en pratique)

CLASSES – METHODES D'OBJET – SIGNATURE

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(self): return self.nom

    def getNote(self, i):
        return self.notes[i - 1]

    def setNote(self, i, nouvelleNote):
        self.notes[i - 1] = nouvelleNote

    def __str__(self):
        """Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        """
        resultat = self.nom + '\n'
        resultat += ''.join(map(str, self.notes))
        return resultat
```

Une méthode n'a pas de type de retour ni de type d'argument

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public String getNom(){
        return this.nom;
    }

    public int getNote(int i){
        return this.notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        this.notes[i - 1] = nouvelleNote;
    }

    public String toString(){
        /*
        Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        */
        String resultat = this.nom + '\n';
        for (int note : this.notes)
            resultat += note + ' ';
        return resultat;
    }
}
```

une méthode doit avoir un type de retour (sauf si c'est un constructeur, type void si pas de retour) et déclarer un type pour chacun de ses arguments.

CLASSES – METHODES D'OBJET – SURCHARGE

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

def resetNotes(self, arg = 0):
    for i in range(Etudiant.NB_NOTES):
        if type(arg) == list:
            self.notes[i] = arg[i]
        else:
            self.notes[i] = arg
```

la surcharge sur le nombre d'argument se fait en donnant des valeurs par défauts aux arguments
la surcharge sur le type d'argument se fait en testant le type de l'argument

```python
e = Etudiant('Dupont')
e.resetNotes(100)
e.resetNotes()
nouvellesNotes = [85, 66, 90, 100, 73]
e.resetNotes(nouvellesNotes)
```

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public void resetNotes(){
        resetNotes(0);
    }

    public void resetNotes(int note){
        for (int i = 0; i < Etudiant.NB_NOTES; i++)
            this.notess[i] = note;
    }

    public void resetNotes(int[] notes){
        for (int i = 0; i < Etudiant.NB_NOTES; i++)
            this.notes[i] = notes[i];
    }
}
```

la surcharge sur le nombre ou sur le type d'arguments se fait en redéfinissant des nouvelles fonctions avec le même nom mais des signatures différentes

```java
Etudiant e = new Etudiant("Dupont");
e.resetNotes(100);
e.resetNotes();
int[] nouvellesNotes = {85, 66, 90, 100, 73};
s.resetNotes(nouvellesNotes);
```

CLASSES – METHODES DE CLASSES (STATIQUES)

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(self): return self.nom

    def getNote(self, i):
        return self.notes[i - 1]

    def setNote(self, i, nouvelleNote):
        self.notes[i - 1] = nouvelleNote

    @classmethod
    def getNoteLitterale(cls, note):
        if note > 89:
            return 'A'
        elif note > 79:
            return 'B'
        else:
            return 'F'
```

```python
e = Etudiant('Dupont')
for i in range(1, Etudiant.NB_NOTES + 1):
    print Etudiant.getNoteLitterale(e.getNote(i))
```

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public String getNom(){
        return this.nom;
    }

    public int getNote(int i){
        return this.notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        this.notes[i - 1] = nouvelleNote;
    }

    public static char getNoteLitterale(int note){
        if (note > 89)
            return 'A';
        else if (note > 79)
            return 'B';
        else
            return 'F';
    }

}
```

```java
Etudiant e = new Etudiant("Dupont");
for (int i = 1; i <= Etudiant.NB_NOTES; i++)
    System.out.println(Etudiant.getNoteLitterale(e.getNote(i));
```

CLASSES – VARIABLES DE CLASSES (STATIQUES)

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(self): return self.nom

    def getNote(self, i):
        return self.notes[i - 1]

    def setNote(self, i, nouvelleNote):
        self.notes[i - 1] = nouvelleNote

    @classmethod
    def getNoteLitterale(cls, note):
        if note > 89:
            return 'A'
        elif note > 79:
            return 'B'
        else:
            return 'F'
```

```python
e = Etudiant('Dupont')
for i in range(1, Etudiant.NB_NOTES + 1):
    print Etudiant.getNoteLitterale(e.getNote(i))
```

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public String getNom(){
        return this.nom;
    }

    public int getNote(int i){
        return this.notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        this.notes[i - 1] = nouvelleNote;
    }

    public static char getNoteLitterale(int note){
        if (note > 89)
            return 'A';
        else if (note > 79)
            return 'B';
        else
            return 'F';
    }

}
```

```java
Etudiant e = new Etudiant("Dupont");
for (int i = 1; i <= Etudiant.NB_NOTES; i++)
    System.out.println(Etudiant.getNoteLitterale(e.getNote(i));
```

CLASSES – REPRESENTATION STRING

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(self): return self.nom

    def getNote(self, i):
        return self.notes[i - 1]

    def setNote(self, i, nouvelleNote):
        self.notes[i - 1] = nouvelleNote

    def __str__(self):
        """Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        """
        resultat = self.nom + '\n'
        resultat += ''.join(map(str, self.notes))
        return resultat
```

```python
e1 = Etudiant('Pierre')
print(e1)
e2 = Etudiant('Marie')
print(str(e1) + '\n' + str(e2))
```

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public String getNom(){
        return this.nom;
    }

    public int getNote(int i){
        return this.notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        this.notes[i - 1] = nouvelleNote;
    }

    public String toString(){
        /*
        Format: Nom sur la première ligne
        toutes les notes sur la seconde ligne,
        séparés par des espaces.
        */
        String resultat = this.nom + '\n';
        for (int note : this.notes)
            resultat += note + ' ';
        return resultat;
    }
}
```

```java
e1 = new Etudiant("Pierre");
System.out.println(e1);
e2 = new Etudiant("Marie");
System.out.println(e1 + '\n' + e2);
```

CLASSES – EGALITE D'OBJETS

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(self): return self.nom

    def getNote(self, i):
        return self.notes[i - 1]

    def setNote(self, i, nouvelleNote):
        self.notes[i - 1] = nouvelleNote
```

```python
e1 = Etudiant('Pierre')
e2 = Etudiant('Pierre')

print(e1==e2)       //False
print(e1 is e2)     //False
```

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public String getNom(){
        return this.nom;
    }

    public int getNote(int i){
        return this.notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        this.notes[i - 1] = nouvelleNote;
    }

}
```

```java
e1 = new Etudiant("Pierre");
e1 = new Etudiant("Pierre");

System.out.println(e1==e2);          //false
System.out.println(e1.equals(e2));   //false
```

CLASSES – EGALITE D'OBJETS

```python
class Etudiant:

    NB_NOTES = 5

    def __init__(self, nom):
        self.nom = nom
        self.notes = []
        for i in range(Etudiant.NB_NOTES):
            self.notes.append(0)

    def getNom(self): return self.nom

    def getNote(self, i):
        return self.notes[i - 1]

    def setNote(self, i, nouvelleNote):
        self.notes[i - 1] = nouvelleNote



    def __eq__(self, e):
        return self.nom==e.nom and self.notes==e.notes
```

```python
e1 = Etudiant('Pierre')
e2 = Etudiant('Pierre')

print(e1==e2)       //True
print(e1 is e2)     //False
```

```java
public class Etudiant{

    public static final int NB_NOTES = 5;

    private String nom;
    private int[] notes;

    public Etudiant(String nom){
        this.nom = nom;
        this.notes = new int[NB_NOTES];
    }

    public String getNom(){
        return this.nom;
    }

    public int getNote(int i){
        return this.notes[i - 1];
    }

    public void setNote(int i, int nouvelleNote){
        this.notes[i - 1] = nouvelleNote;
    }


    public boolean estEgal(Etudiant e){
         return this.nom.equals(e.nom) && Arrays.equals(this.notes, e.notes);
    }
    //attention, ce n'est pas un override de equals de Object (cf override de
    //la méthode equals et de la méthode hashCode)

}
```

```java
e1 = new Etudiant("Pierre");
e1 = new Etudiant("Pierre");

System.out.println(e1==e2);           //false
System.out.println(e1.equals(e2));    //false
System.out.println(e1.estEgal(e2));   //true
```