

Langage et algorithmique

A. Malek TOUMI

toumiab@ensta-bretagne.fr

2015/2016

ENSTA Bretagne



Sommaire

1 Fonction

Déclaration d'une fonction

Paramètres par défaut

Règles de portée

Notion de pile d'appel

2 Retour sur les tableaux

Recherche dichotomique

Diviser pour régner

3 Utilisation des classes de l'API Java

Quelques classes de l'API

Consultation de la documentation



Sommaire

1 Fonction

Déclaration d'une fonction

Paramètres par défaut

Règles de portée

Notion de pile d'appel

2 Retour sur les tableaux

Recherche dichotomique

Diviser pour régner

3 Utilisation des classes de l'API Java

Quelques classes de l'API

Consultation de la documentation



Intérêt des fonctions

- Découpage du programme en sous-programmes : fonctions
- Avantage : modularité, réutilisabilité
- Évite les copier/coller !
- Structure le code
- Rend le code plus lisible, plus compact



Intérêt des fonctions

- Découpage du programme en sous-programmes : fonctions
- Avantage : modularité, réutilisabilité
- Évite les copier/coller !
- Structure le code
- Rend le code plus lisible, plus compact

Syntaxe :

```
def nomFonction (arg1,arg2, ... ,argn):  
    <instruction>  
    return <valeur>
```



Intérêt des fonctions

- Découpage du programme en sous-programmes : fonctions
- Avantage : modularité, réutilisabilité
- Évite les copier/coller !
- Structure le code
- Rend le code plus lisible, plus compact

Syntaxe :

```
def nomFonction (arg1,arg2, ... ,argn):  
    <instruction>  
    return <valeur>
```

Exemples :

- Fonctions de calcul
- Fonctions d'analyse des fichiers



Syntaxe

Diagram illustrating the syntax of a Python function definition with annotations:

```
def proportion ( chaîne, base ) :  
    """Fréquence d'apparition de base dans chaîne.  
    chaîne est une chaîne de caractères,  
    base doit être un caractère unique.  
    Attention: Chaîne ne doit pas être vide  
    """  
    n = len(chaîne)  
    k = chaîne.count(base)  
    return k / n
```

Annotations:

- mot réservé**: points to `def`
- nom de la fonction**: points to `proportion`
- paramètres formels**: points to `chaîne, base`
- parenthèses (obligatoires)**: points to the parentheses around the parameters
- chaîne de documentation (docstring)**: points to the triple-quoted string block
- indentation**: points to the indentation of the function body
- variables locales**: points to `n` and `k`
- mot réservé**: points to `return`
- résultat de la fonction**: points to `k / n`
- la fin de la définition de la fonction est indiquée par le retour à l'indentation normale**: points to the end of the function body



Syntaxe

- définition

```
def proportion ( chaine, base ) :  
    n = len(chaine)  
    k = chaine.count(base)  
    return k /n
```

paramètres ou paramètres formels

- appel

```
...  
y = proportion ( adn, "t" )  
f = proportion  
y = f ( adn, "t" )  
...
```

arguments ou paramètres effectifs



Syntaxe

- définition

```
def proportion ( chaine, base ) :  
    n = len(chaine)  
    k = chaine.count(base)  
    return k/n
```

paramètres ou paramètres formels

- appel

```
...  
y = proportion ( adn, "t" )  
...
```

```
    n = len(adn)  
    k = chaine.count("t")  
    return k/n
```

arguments ou paramètres effectifs

chaine et base sont substitués par adn et "t"

Remarques

- Utiliser un nom qui commence par un minuscule
Exemple : uneFonctionBienNommee
- La correspondance des paramètres effectifs/formels est positionnelle par défaut
- pour l'appel, les paramètres et arguments doivent coïncider en nombre par défaut



Fonction sans valeur de retour

- Pas de return
- Valeur de retour conventionnelle : None

Exemple

```
def ajouElemSpecial(liste,element):  
    nouvElem=[element] * 2  
    liste.append(nouvElem)  
    print(liste)
```

```
lst =[1, 2]  
ajouElemSpecial(lst, 4)
```



Fonction avec valeur(s) de retour

Exemple

```
def operations(ch, val):  
    if ch.count(str(val)) and val:  
        return int(ch)/val  
    else return 0  
  
s = '25'  
divR = operations(s,2) # => divR=12.5
```



Fonction avec valeur(s) de retour

Exemple

```
def operations(ch, val):  
    if ch.count(str(val)) and val:  
        return int(ch)/val, int(ch)//int(val)  
    else return 0, 0 # retourne (0,0)  
  
s = '25'  
divR, divE = operations(s,2) # => divR=12.5  
                             # et divE=12
```



Sommaire

1 Fonction

Déclaration d'une fonction

Paramètres par défaut

Règles de portée

Notion de pile d'appel

2 Retour sur les tableaux

Recherche dichotomique

Diviser pour régner

3 Utilisation des classes de l'API Java

Quelques classes de l'API

Consultation de la documentation



Paramètres formels/paramètres effectifs

Exemple (paramètre formel/paramètre effectif)

```
public void m1(int parametreFormel) {  
    int i = parametreFormel + 22;  
    System.out.println(i);  
}  
public void m2() {  
    int parametreEffectif = 20;  
    m1(parametreEffectif);  
    m1(20);  
}
```



Passage de paramètres

Deux types de passages de paramètres :

- paramètres de type simple : passage par valeur
- paramètres de type composite : passage par variable



Passage de paramètres

Exemple

```
public void uneMethode(int param) {  
    param++;  
    System.out.println("dans uneMethode : "+param);  
}  
public void autreMethode() {  
    int i=0;  
    System.out.println("autreMethode : "+i);  
    uneMethode(i);  
    System.out.println("fin de autreMethode : "+i);  
}
```



Passage de paramètres

Deux types de passages de paramètres :

- paramètres de type simple : passage par valeur
- paramètres de type composite : passage par variable
- Résultat :

```
autreMethode : 0  
dans uneMethode : 1  
fin de autreMethode : 0
```



Paramètres de type composite

Exemple (Paramètres de type composite)

```
public void modifieVariable1(int tableau[]) {  
    tableau[0]=42;  
}  
public void modifieVariable2(int tableau[]) {  
    tableau = new int[10];  
    tableau[0]=3;  
}
```



Paramètres de type composite

Exemple (Paramètres de type composite)

```
public void utiliseMethode() {  
    int tab[] = new int[5];  
    tab[0] = 1;  
    System.out.println(tab.length+" "+tab[0]);  
    modifieVariable1(tab);  
    System.out.println(tab.length+" "+tab[0]);  
    modifieVariable2(tab);  
    System.out.println(tab.length+" "+tab[0]);  
}
```



Paramètres de type composite

Exemple (Paramètres de type composite)

Résultat de l'exécution :

```
5 1  
5 42  
5 42
```



Sommaire

1 Fonction

Déclaration d'une fonction

Paramètres par défaut

Règles de portée

Notion de pile d'appel

2 Retour sur les tableaux

Recherche dichotomique

Diviser pour régner

3 Utilisation des classes de l'API Java

Quelques classes de l'API

Consultation de la documentation



- Variables locales de la méthode : portée limitée à la méthode
- Paramètres : portée limitée à la méthode

Exemple (Portée)

```
public void methodeTest(int parametre) {  
    int nombre = 10;  
}
```



Sommaire

1 Fonction

Déclaration d'une fonction

Paramètres par défaut

Règles de portée

Notion de pile d'appel

2 Retour sur les tableaux

Recherche dichotomique

Diviser pour régner

3 Utilisation des classes de l'API Java

Quelques classes de l'API

Consultation de la documentation



Pile d'appel

- Zone de mémoire



Pile d'appel

- Zone de mémoire
- Contient :



Pile d'appel

- Zone de mémoire
- Contient :
 - paramètres



Pile d'appel

- Zone de mémoire
- Contient :
 - paramètres
 - adresse de retour



Pile d'appel

- Zone de mémoire
- Contient :
 - paramètres
 - adresse de retour
 - variables locales



Pile d'appel

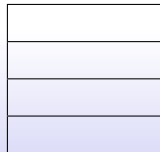
- Zone de mémoire
- Contient :
 - paramètres
 - adresse de retour
 - variables locales
- Fonctionnement lors de l'appel d'une méthode



Pile d'appel

- Zone de mémoire
- Contient :
 - paramètres
 - adresse de retour
 - variables locales
- Fonctionnement lors de l'appel d'une méthode

```
public void methode(int a, double b) {  
    int i;  
}  
methode(2, 1.5);
```





Pile d'appel

- Zone de mémoire
- Contient :
 - paramètres
 - adresse de retour
 - variables locales
- Fonctionnement lors de l'appel d'une méthode

```
public void methode(int a, double b) {  
    int i;  
}  
methode(2, 1.5);
```

2



Pile d'appel

- Zone de mémoire
- Contient :
 - paramètres
 - adresse de retour
 - variables locales
- Fonctionnement lors de l'appel d'une méthode

```
public void methode(int a, double b) {  
    int i;  
}  
methode(2, 1.5);
```

1.5
2



Pile d'appel

- Zone de mémoire
- Contient :
 - paramètres
 - **adresse de retour**
 - variables locales
- Fonctionnement lors de l'appel d'une méthode

```
public void methode(int a, double b) {  
    int i;  
}  
methode(2, 1.5);
```

0x1E3C85
1.5
2



Pile d'appel

- Zone de mémoire
- Contient :
 - paramètres
 - adresse de retour
 - **variables locales**
- Fonctionnement lors de l'appel d'une méthode

```
public void methode(int a, double b) {  
    int i;  
}  
methode(2, 1.5);
```

i
0x1E3C85
1.5
2



Sommaire

1 Fonction

Déclaration d'une fonction

Paramètres par défaut

Règles de portée

Notion de pile d'appel

2 Retour sur les tableaux

Recherche dichotomique

Diviser pour régner

3 Utilisation des classes de l'API Java

Quelques classes de l'API

Consultation de la documentation



Sommaire

1 Fonction

Déclaration d'une fonction

Paramètres par défaut

Règles de portée

Notion de pile d'appel

2 Retour sur les tableaux

Recherche dichotomique

Diviser pour régner

3 Utilisation des classes de l'API Java

Quelques classes de l'API

Consultation de la documentation



Recherche par dichotomie

- Exemple : recherche d'un élément dans un tableau trié

1	2	3	4	5
---	---	---	---	---



Recherche par dichotomie

- Exemple : recherche d'un élément dans un tableau trié
- Principe : comparer l'élément avec la valeur médiane

1	2	3	4	5
---	---	---	---	---



Recherche par dichotomie

- Exemple : recherche d'un élément dans un tableau trié
- Principe : comparer l'élément avec la valeur médiane
- puis recommencer dans le sous tableau contenant l'élément

1	2	3	4	5
---	---	---	---	---



Recherche par dichotomie

- Exemple : recherche d'un élément dans un tableau trié
- Principe : comparer l'élément avec la valeur médiane
- puis recommencer dans le sous tableau contenant l'élément

1	2	3	4	5
---	---	---	---	---



Recherche par dichotomie

- Exemple : recherche d'un élément dans un tableau trié
- Principe : comparer l'élément avec la valeur médiane
- puis recommencer dans le sous tableau contenant l'élément

1	2	3	4	5
---	---	---	---	---

Complexité :



Recherche par dichotomie

- Exemple : recherche d'un élément dans un tableau trié
- Principe : comparer l'élément avec la valeur médiane
- puis recommencer dans le sous tableau contenant l'élément

1	2	3	4	5
---	---	---	---	---

Complexité : $\Theta(\log n)$



Sommaire

1 Fonction

Déclaration d'une fonction

Paramètres par défaut

Règles de portée

Notion de pile d'appel

2 Retour sur les tableaux

Recherche dichotomique

Diviser pour régner

3 Utilisation des classes de l'API Java

Quelques classes de l'API

Consultation de la documentation

- Principe : décomposer le problème en sous problèmes (sous ensemble des données)

- Principe : décomposer le problème en sous problèmes (sous ensemble des données)
- résoudre les sous problèmes (éventuellement en redécoupant)

- Principe : décomposer le problème en sous problèmes (sous ensemble des données)
- résoudre les sous problèmes (éventuellement en redécoupant)
- puis fusionner les résultats

- Principe : décomposer le problème en sous problèmes (sous ensemble des données)
- résoudre les sous problèmes (éventuellement en redécoupant)
- puis fusionner les résultats
- Exemples :

- Principe : décomposer le problème en sous problèmes (sous ensemble des données)
- résoudre les sous problèmes (éventuellement en redécoupant)
- puis fusionner les résultats
- Exemples :
 - tri par partition/fusion

- Principe : décomposer le problème en sous problèmes (sous ensemble des données)
- résoudre les sous problèmes (éventuellement en redécoupant)
- puis fusionner les résultats
- Exemples :
 - tri par partition/fusion
 - tri rapide



Sommaire

1 Fonction

Déclaration d'une fonction

Paramètres par défaut

Règles de portée

Notion de pile d'appel

2 Retour sur les tableaux

Recherche dichotomique

Diviser pour régner

3 Utilisation des classes de l'API Java

Quelques classes de l'API

Consultation de la documentation



Sommaire

1 Fonction

Déclaration d'une fonction

Paramètres par défaut

Règles de portée

Notion de pile d'appel

2 Retour sur les tableaux

Recherche dichotomique

Diviser pour régner

3 Utilisation des classes de l'API Java

Quelques classes de l'API

Consultation de la documentation



Random

Classe **Random** : génération de nombres aléatoires.
Deux étapes :

Exemple

```
Random alea = new Random();  
int i = alea.nextInt(100);  
double x = alea.nextDouble();
```



Random

Classe **Random** : génération de nombres aléatoires.

Deux étapes :

- Créer une variable de type **Random**

Exemple

```
Random alea = new Random();  
int i = alea.nextInt(100);  
double x = alea.nextDouble();
```



Random

Classe **Random** : génération de nombres aléatoires.

Deux étapes :

- Créer une variable de type **Random**
- Utiliser la variable

Exemple

```
Random alea = new Random();  
int i = alea.nextInt(100);  
double x = alea.nextDouble();
```



Integer/Double/String

```
int i = Integer.MIN_VALUE;  
int j = Integer.parseInt("123");
```



Integer/Double/String

```
int i = Integer.MIN_VALUE;  
int j = Integer.parseInt(args[0]);
```



Integer/Double/String

```
int i = Integer.MIN_VALUE;  
int j = Integer.parseInt(args[0]);  
  
double x = Double.POSITIVE_INFINITY;  
double y = Double.parseDouble(args[1]);
```



Integer/Double/String

```
int i = Integer.MIN_VALUE;  
int j = Integer.parseInt(args[0]);  
  
double x = Double.POSITIVE_INFINITY;  
double y = Double.parseDouble(args[1]);  
  
String s = "abcde";  
boolean b = s.equals(args[2]);
```



Integer/Double/String

```
int i = Integer.MIN_VALUE;  
int j = Integer.parseInt(args[0]);  
  
double x = Double.POSITIVE_INFINITY;  
double y = Double.parseDouble(args[1]);  
  
String s = "abcde";  
boolean b = s.equals(args[2]);
```

Plus d'informations : consulter la doc de l'API Java.



LinkedList

- Objectif : manipuler des collections de données
- ≈ tableaux de taille variable



LinkedList

- Objectif : manipuler des collections de données

≈ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)

Exemple

```
LinkedList l = new LinkedList();
```



LinkedList

- Objectif : manipuler des collections de données

≈ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)
- Ajouter un élément

Exemple

```
l.add(42);
```



LinkedList

- Objectif : manipuler des collections de données

≈ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)
- Ajouter un élément
- Supprimer un élément

Exemple

```
l.remove(0);
```



LinkedList

- Objectif : manipuler des collections de données

≈ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)
- Ajouter un élément
- Supprimer un élément
- Rechercher un élément

Exemple

```
if(l.contains(42)) ...
```



LinkedList

- Objectif : manipuler des collections de données

≈ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)
- Ajouter un élément
- Supprimer un élément
- Rechercher un élément

Exemple

```
import java.util.LinkedList;
```



LinkedList

- Objectif : manipuler des collections de données
≈ tableaux de taille variable

Opérations de base :

- Créer une collection (ex. **LinkedList**)
- Ajouter un élément
- Supprimer un élément
- Rechercher un élément

Exemple

```
import java.util.*;
```



Stéréotypage des collections

Depuis Java 1.5 : possibilité de stéréotyper les collections :

- Permet de vérifier la cohérence des types
- Évite certains **cast**

Exemple

```
LinkedList<Integer> l = new LinkedList<Integer>();  
l.add(42);  
l.add(13.2);  
int i = l.get(0);  
Integer j = l.get(1);  
System.out.println(l);
```



Math

Classe particulière : ne pas créer d'objet de type **Math**.
Contenu :



Math

Classe particulière : ne pas créer d'objet de type **Math**.
Contenu :

- Constantes

Exemple

```
double pi = Math.PI;
```



Math

Classe particulière : ne pas créer d'objet de type **Math**.
Contenu :

- Constantes
- Valeur absolue

Exemple

```
double x = Math.abs(y);
```



Math

Classe particulière : ne pas créer d'objet de type **Math**.
Contenu :

- Constantes
- Valeur absolue
- Fonctions trigonométriques

Exemple

```
double x = Math.cos(y * Math.PI);
```



Math

Classe particulière : ne pas créer d'objet de type **Math**.
Contenu :

- Constantes
- Valeur absolue
- Fonctions trigonométriques
- Fonctions d'arrondis

Exemple

```
int i = (int)Math.floor(Math.E);
```



Sommaire

1 Fonction

Déclaration d'une fonction

Paramètres par défaut

Règles de portée

Notion de pile d'appel

2 Retour sur les tableaux

Recherche dichotomique

Diviser pour régner

3 Utilisation des classes de l'API Java

Quelques classes de l'API

Consultation de la documentation



API Java

- Toutes les classes de l'API sont documentées
- Apprendre à consulter la documentation
- Les paquetages les plus utiles :
 - java.lang
 - java.util

Accès depuis moodle : `https:`

`//moodle.ensieta.fr/file.php/1/docs/api/index.html`