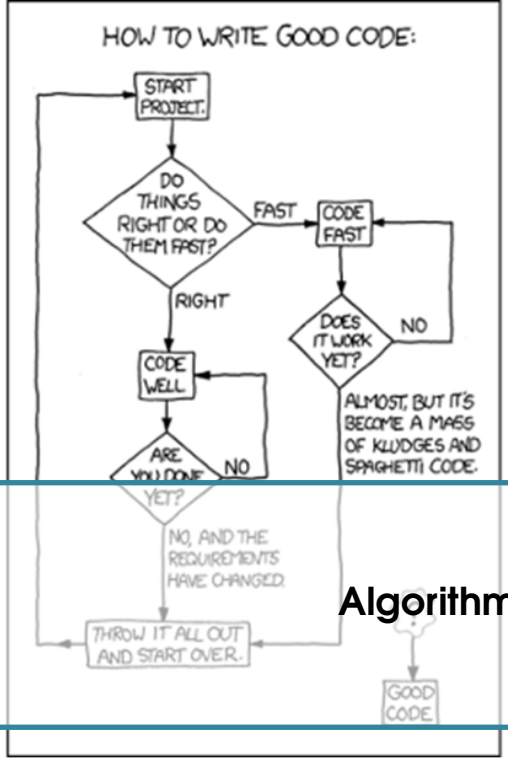
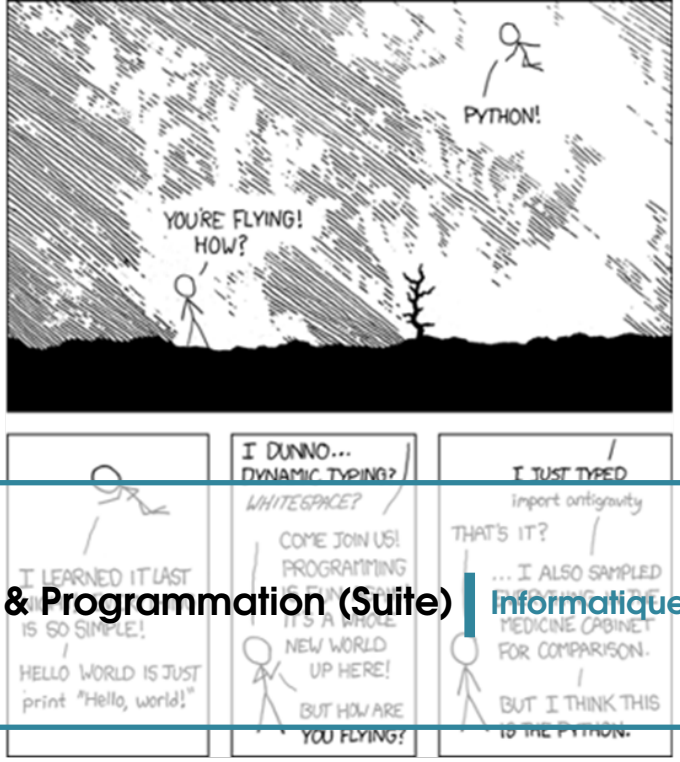


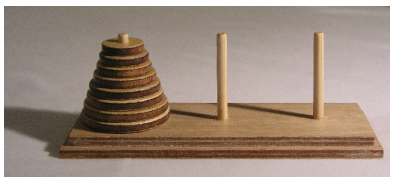


Cours

Chapitre 2
Piles et files

Savoirs et compétences :

- ❒ *Alg – C16 : Piles - Algorithmes de manipulation : fonctions «push» et «pop».*



Tour de Hanoi [2]

Xavier Pessoles
PT

1 Présentation

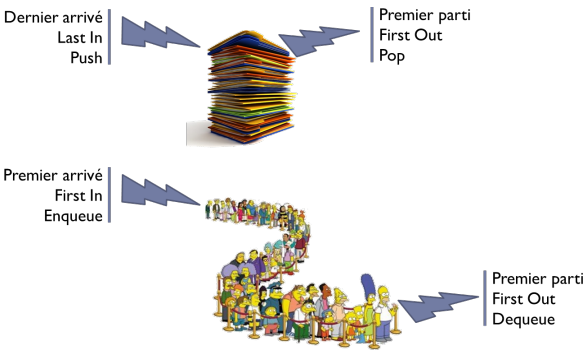
1.1 Qu'est ce qu'une pile ? une file ?

Les piles et les files sont des listes particulières : on accède aux éléments par les extrémités, c'est-à-dire au début ou à la fin. Elles sont utilisées par exemple pour des programmes qui doivent traiter des données qui arrivent au fur et à mesure. On distingue :

- les piles (« stacks ») : le premier empilé est le dernier à être dépilé, « LIFO » (Last In first Out) ;
- les files : le premier entré est le premier à sortir, « FIFO » (First In First Out).

Ainsi, ces structures de données nécessitent la réalisation de fonctions spécifiques :

- pour les piles :
  - créer une pile,
  - ajouter au sommet : empiler (« push »),
  - supprimer du sommet : dépiler (« pop »),
  - est_vide,est_pleine, taille, sommet ;
- pour les files :
  - ajouter en queue : enfiler (« enqueue ») ;
  - supprimer tête : défiler (« dequeue ») ;
  - est_vide, est_pleine.



1.2 Gestion des piles et des files en Python

L'implémentation des piles et des files peut se faire à l'aide d'une liste. Les opérations élémentaires permettant de manipuler une liste sont les suivantes :

- méthodes de base :
  - ajouter en élément `el` de n'importe quel type : `l.append(el)` ;
  - supprimer un élément à l'indice `i` et le retourner : `l.pop(i)` ;
  - insérer un élément `el` à l'indice `i` : `l.insert(i,el)` ;
  - supprimer la première occurrence de l'élément `el` : `l.remove(el)` ;
  - supprimer un élément sans le retourner : `del a[i]` ;
- modification d'une liste :
  - modifier un élément à l'indice `i` : `l[i]=el` ;
  - concaténation d'éléments : `l+[el]` ;
- accès au premier élément : `l[0]` ;
- accès au dernier élément : `l[-1]` ;
-

En Python, les listes sont des objets dynamiques mutables. Cela signifie que d'une part il n'y a pas besoin de connaître a priori la taille d'une liste en la déclarant : elle va être redimensionnée à mesure que des éléments sont ajoutés ou supprimés. D'autre part, on peut modifier un élément de la liste sans recréer cette liste.

▪ **Exemple** Les chaînes de caractères ou les tuples sont des objets non mutables. On ne peut pas les modifier sans créer de nouvelle instance de l'objet.

Modification d'un élément d'une liste :

```
>>> l = ["k","a","m","a","t","e"]
>>> l[0]="!"
```

Modification d'un caractère d'une chaîne de caractères :

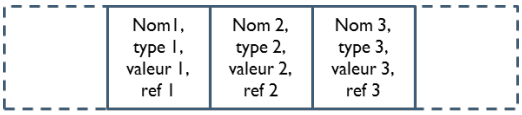
```
>>> ch = "kamate"
>>> ch = "!"+ch[1:len(ch)]
```

▪

1.3 Généralités sur les structures de données

Les données sont stockées de façon structurée dans la mémoire de l'ordinateur. Les objets de types «simples» (non mutables) sont stockés dans des variables ayant les caractéristiques suivantes : un identificateur (nom), un type, une valeur, une référence et des opérations associées. La référence constitue l'adresse de l'objet dans la mémoire de l'ordinateur.

Ainsi, si on représentait la mémoire de façon linéaire (ce qui est le cas sur un disque dur non SSD), on pourrait procéder ainsi :



Suivant la valeur, le cadre peut s'étendre sur un grand nombre de bits. Pour les types Python non mutables (int, flt, str, tuple) il est possible de considérer que les données sont juxtaposées car il est possible de connaître la taille de chaque variable lors de l'affectation.

Pour les types mutables (comme les listes), on ne sait pas à l'avance combien d'emplacements mémoire vont être nécessaires car on peut ajouter des données à une même liste de façon indéfinie.

Dans beaucoup de langages de programmation, les listes ont :

- un nombre de cases fixées à l'avance par l'utilisateur ;
- des cases qui sont toutes du même type ;
- un accès aux cases se faisant à coût constant ($\mathcal{O}(1)$).

En Python, ce n'est pas le cas ce qui rend très aisé la gestion des piles et des files.

2 Les piles

2.1 Quelques exemples d'applications

1. Empilement de dossiers : dans une pile de dossiers, le dernier arrivé est le premier traité.
2. Gestion de processus par un système d'exploitation : dans un ordinateur, lorsqu'un « processus 1 » fait appel à un « processus 2 » qui fait lui-même appel à un « processus 3 », l'ensemble est stocké dans une table des processus, propre au noyau du système d'exploitation. Lorsque le « processus 3 » se termine, le système sait qu'il doit revenir au « processus 2 », puis au « processus 1 ».
3. Dans un navigateur web, une pile sert à mémoriser les pages Web visitées. L'adresse de chaque nouvelle page visitée est empilée et l'utilisateur dépile l'adresse de la page précédente en cliquant le bouton « Afficher la page précédente ».

2.2 Implémentation d'une pile

Dans cette section, nous allons considérer simultanément qu'une pile est implémentée sous forme d'une liste. Une autre méthode (qui n'est pas au programme serait de créer une nouvelle classe d'objets Python).

Rappel Ainsi, pour déclarer une pile, on utilise la méthode usuelle pour d'initialisation d'une liste.

```
pile_liste = []
```

2.2.1 Ajouter un élément au sommet de la pile – empiler

Les spécifications de la fonction **empiler** sont les suivantes :

Algorithme : empiler
Ajouter un élément au sommet de la pile

Entrée :
- pile : une pile
- el : un élément à empiler

Sortie :
- pile : une pile composée de de la pile initiale et de l'élément à empiler.

Pseudo Code

empiler(pile,el) :
    pile ← pile+[el]
    retourner pile

Python

```
def empiler(pile,el):
    pile.append(el)
```

R En python le passage des arguments se faisant par référence, retourner la pile n'est pas indispensable car dans ce cas l'adresse de la liste n'a pas été changée. Le `append` est équivalent à l'opération de concaténation suivante :

```
pile = pile+[el]
```

2.2.2 Supprimer l'élément au sommet de la pile

Les spécifications de la fonction **empiler** sont les suivantes :

Algorithme : depiler
Supprimer le dernier élément de la pile et retourner cet élément

Entrée :
- pile : une pile

Sortie :
- el : élément qui était au sommet de la pile.

▪ Pseudo Code

```
depiler(pile) :
    n← Taille(pile)
    el← pile[n]
    pile_s ← []
    Pour i allant de 1 à n-1 faire
        pile_s[i]← pile[i]
    Fin Pour
    pile ← pile_s
    Retourner el
```

▪ Python

```
def depiler(pile):
    assert len(pile)>0
    return pile.pop()
```

R En Pyhton, la méthode pop retourne automatiquement le dernier élément d'une liste et le supprimer de celle-ci.

3 Les files

3.1 Quelques exemples d'application

1. File d'attente : dans une file d'attente, le premier arrivé est le premier servi.
2. Gestion de processus par un système d'exploitation : dans un ordinateur, lorsque des appels aux processus « 1 », « 2 », puis « 3 » se succèdent, l'ensemble est stocké dans une table des processus, propre au noyau du système d'exploitation. Lorsque le « processus 1 » se termine, le système sait qu'il doit passer au « processus 2 », puis au « processus 3 ».
3. Gestion des travaux d'impression.

R Les principales fonctions associées aux files sont :
- ajouter en queue : enfiler (« enqueue ») ;
- supprimer tête : défiler (« dequeue ») ;
- est_vide, est_pleine.

3.2 Implémentation d'une file

Dans cette section, nous allons considérer simultanément qu'une file est implémentée sous forme d'une liste.

3.2.1 Ajouter un élément en queue de file – enfiler

Les spécifications de la fonction **enfiler** sont les suivantes :

Algorithme : enfiler
Ajouter un élément en queue de file

Entrée :
- file : une file
- el : un élément à ajouter en queue de file

Sortie :
- file : une file composée de de la file initiale et de l'élément en queue.

▪ Pseudo Code

```
enfiler(file,el) :
    file ← file+[el]
    retourner file
```

▪ Python

```
def enfiler(file,el):
    file.append(el)
```

3.2.2 Supprimer l'élément en tête de la file

Les spécifications de la fonction **défiler** sont les suivantes :

Algorithme : defiler
Supprimer le premier élément de la file et retourner cet élément

Entrée :
- file : une file

Sortie :
- el : élément qui était en tête de la file.

Pseudo Code

```
defiler(file) :
    n← Taille(file)
    el← file[1]
    file_s ← []
    Pour i allant de 2 à n faire
        file_s[i-1]← file[i]
    Fin Pour
    file ← file_s
    Retourner el
```

Python

```
def defiler(file):
    assert len(file)>0
    el = file[0]
    del file[0]
    return el
```

R Pourquoi le code suivant ne fonctionne pas ?

```
def defiler(file):
    assert len(file)>0
    el = pile[0]
    pile=pile[1:len(file)]
    return el
```

Références

[1] Patrick Beynet, *Supports de cours de TSI 2*, Lycée Rouvière, Toulon.

[2] « Tower of Hanoi ». Sous licence CC BY-SA via Wikimedia Commons - https://fr.wikipedia.org/wiki/Tower_of_Hanoi.jpeg#/media/File:Tower_of_Hanoi.jpeg

[3] Christophe Lambert, *Supports de cours de PT – PT⋆*, Lycée La Martinière Monplaisir, Lyon.