Livret de révision du programme de spécialité NSI de première

Types construits : les séquences

CE QU'IL FAUT SAVOIR

Il est possible de stocker plusieurs grandeurs dans une même structure, ce type de structure est appelé une **séquence.**

Les tuples

- Un **tuple** est une séquence (exemple de tuple : `tup = (1, 5, 3, 4)`)
- Chaque élément d'un tuple est identifié par un **indice** (premier élément d'indice `0`, deuxième élément d'indice `1`, etc.)
- Pour accéder à un élément d'indice `i` on utilise la notation entre crochets : `tup[i]` (où `tup` est un tuple)
- Une fois créé, **un tuple ne peut pas être modifié**

Les tableaux

- Un **tableau** (**liste** en Python) est une séquence (exemple de tableau : `tab = [1, 5, 3, 4]`)
- Chaque élément d'un tableau est identifié par un **indice** (premier élément d'indice `0`, deuxième élément d'indice `1`, etc.)
- Pour accéder à un élément d'indice `i` on utilise la notation entre crochets `tab[i]` (où `tab` est un tableau)
- **Il est possible de modifier un tableau**
- On peut créer des **tableaux de tableaux** (exemple `t = [[1, 2, 3], [7, 9, 15]]`)

La boucle for

On peut utiliser une boucle `for` pour parcourir :

- les indices d'un tableau (`for i in range(len(tab))`) ;
- les éléments d'un tableau (`for elt in tab`).

Créer un tableau par compréhension

Il est possible de créer un tableau en utilisant une boucle `for` et éventuellement une **condition** (`if`).

Exemple : `mon_tab = [elt ** 2 for elt in tab]` (où `tab = [1, 5, 3, 4]`)

CE QU'IL FAUT SAVOIR FAIRE

- Savoir utiliser les tuples (création, utilisation) et les tableaux (création, utilisation, modification, etc.)
- Savoir utiliser une boucle for pour parcourir une séquence (tuple et tableau)
- Savoir créer une tableau par compréhension (utilisation du for et du if)
- Savoir manipuler des tableaux de tableaux

Architecture de Von Neumann

CE QU'IL FAUT SAVOIR

Des transistors aux circuits complexes

- À la base de la plupart des composants d'un ordinateur, on retrouve le **transistor**
- Dans un ordinateur, les transistors sont regroupés au sein de ce que l'on appelle des **circuits intégrés.** Dans un circuit intégré, les transistors sont gravés sur des **plaques de silicium**
- Dans un ordinateur les transistors se comportent comme des **interrupteurs** : soit le transistor laisse passer le courant électrique (interrupteur fermé, état haut), soit il ne le laisse pas passer (interrupteur ouvert, état bas)
- On symbolise souvent l'état "**haut"** par le chiffre "1" et l'état "**bas"** par le chiffre "0"
- Un ordinateur ne connait que deux états (haut et bas ou 0 et 1), un ordinateur travaille donc en **binaire**
- Le transistor est l'élément de base des **circuits logiques.** Un circuit logique permet de réaliser une **opération booléenne**
- Il existe deux catégories de circuit logique :
  - les **circuits combinatoires** (les états en sortie dépendent uniquement des états en entrée) ;
  - les **circuits séquentiels** (les états en sortie dépendent des états en entrée ainsi que du temps et des états antérieurs).
- Les **portes logiques** sont des circuits combinatoires. Il existe quatre types de portes logiques : la porte NON, la porte **OU,** la porte **ET** et la porte **OU EXCLUSIF**
- À chaque type de porte logique on associe une **table de vérité** (vous devez connaître ces tables)
- En combinant les portes logiques, on obtient des circuits plus complexes (exemples : **mémoire, additionneur…)**

Les différents composants d'un ordinateur

- La **mémoire vive** ou **RAM** (Random Access Memory) : la mémoire vive permet de stocker des données et des programmes. La mémoire vive est une **mémoire volatile** : toutes les données présentes en mémoire sont perdues en cas de coupure de courant. Pour stocker les données plus longtemps on utilisera la **mémoire de masse** qui elle, n'est pas volatile (clé USB, disque dur, disque SSD…)
- Le **microprocesseur** ou **CPU** (Central Processing Unit) : Le microprocesseur est le coeur d'un ordinateur : les instructions sont exécutées au niveau du CPU. Il est schématiquement constitué de trois parties :
  - les **registres** permettent de mémoriser de l'information (donnée ou instruction) au sein même du CPU. Leur nombre et leur taille sont variables en fonction du type de microprocesseur ;
  - **l'unité arithmétique** et logique (**UAL** ou **ALU** en anglais) est chargée de l'exécution de tous les calculs que peut réaliser le microprocesseur. Nous allons retrouver dans cette UAL des circuits comme l'additionneur ;
  - **l'unité de commande** permet d'exécuter les instructions (les programmes).
- Le **bus** : les données doivent circuler entre les différentes parties d'un ordinateur, notamment entre la mémoire vive et le CPU. Le système permettant cette circulation est appelé bus. Il existe, sans entrer dans les détails, trois grands types de bus :
  - le **bus d'adresse** permet de faire circuler des adresses (par exemple l'adresse d'une donnée à aller chercher en mémoire) ;
  - le **bus de données** permet de faire circuler des données ;
  - le **bus de contrôle** permet de spécifier le type d'action (exemples : écriture d'une donnée en mémoire, lecture d'une donnée en mémoire).

Architecture de von Neumann

John von Neumann (mathématicien et physicien américano-hongrois 1903-1957) a eu l'idée en 1945 d'utiliser une structure de stockage unique pour les données et les instructions , on parle d'architecture de von Neumann. Voici un schéma qui représente ce modèle de von Neumann :

Sur ce schéma, nous avons :

- la mémoire qui correspond à la RAM vue ci-dessus ;
- l'unité arithmétique et logique qui correspond à l'UAL vu ci-dessus (l'accumulateur est un registre permettant de stocker les résultats intermédiaires lors d'un calcul) ;
- l'unité de commande qui gère l'exécution des instructions machines. À noter que cette unité de commande est aussi parfois appelée "unité de contrôle" ;
- le système "entrée-sortie" qui permet de communiquer avec "le monde extérieur" au système CPU+RAM (clavier, souris, écran, carte graphique, disque dur...).

Les ordinateurs actuels continuent de se baser sur l'architecture de von Neumann. Pendant très longtemps les constructeurs ont cherché à augmenter la fréquence d'horloge des CPU afin d'améliorer les performances, aujourd'hui la tendance est plutôt à l'augmentation du nombre de coeurs

CE QU'IL FAUT SAVOIR FAIRE

Être capable d'écrire et de lire (et comprendre) des programmes simples en assembleur (en ayant en votre possession l'aide mémoire sur l'assembleur)

Dictionnaires

CE QU'IL FAUT SAVOIR

- Un **dictionnaire** est un type construit de **données**
- Chaque élément d'un dictionnaire est composé de deux parties ; on parle de paire **`clé: valeur`**
- Pour afficher une **valeur particulière,** on utilise la notation **`mon_dico[nom_de_la_clé ]`**
- Il est possible de **parcourir l'ensemble des clés** d'un dictionnaire à l'aide d'une boucle for en utilisant la méthode **`.keys()`**
- Il est possible de **parcourir l'ensemble des valeurs** d'un dictionnaire à l'aide d'une boucle for en utilisant la méthode **`.values()`**
- Il est possible de **parcourir l'ensemble des clés et des valeurs** (en même temps) d'un dictionnaire à l'aide d'une boucle for en utilisant la méthode **`.items()`**

Implémentation des dictionnaires

Dans de nombreux langages de programmation, les dictionnaires sont implémentés à l'aide de **tables de hachage**.

Algorithme de recherche dans un dictionnaire

L'algorithme de recherche d'un élément dans un dictionnaire a une complexité $O(1)$. La durée de recherche ne dépend pas du nombre d'éléments présents dans le dictionnaire.

CE QU'IL FAUT SAVOIR FAIRE

- **Construire** un dictionnaire :
  - dictionnaire vide : **`d = dict()`** ou **`d =`** {} ;
  - **`mon_dico =`** {**`"nom": "Dupont", "prenom": "Jean", "date de naissance": "17/05/1971"`**}.
- **Utiliser la notation `mon_dico[nom_de_la_clé ]`** afin d'ajouter une nouvelle paire **`clé: valeur`**, d'utiliser une valeur déjà existante ou de modifier la valeur courante par une autre valeur (la clé restant identique)
- **Utiliser la méthode `get`** pour récupérer une valeur dans un dictionnaire : **`mon_dico.get("adresse", "Adresse introuvable")`**
- **Suprimer une clé** et la valeur qui lui est associée : **`del mon_dico["nom"]`**
- **Parcourir l'ensemble des clés**
- **Parcourir l'ensemble des valeurs**
- **Parcourir l'ensemble des clés et des valeurs** (en même temps)

Représentation des textes

CE QU'IL FAUT SAVOIR

Norme ASCII

La **norme ASCII** (American Standard Code for Information Interchange) permet de représenter les caractères. Chaque caractère est associé à un nombre codé sur 8 bits (7 bits pour coder le caractère + 1 bit dit de parité). Avec 7 bits il est donc possible de coder 128 caractères.

Norme ISO-8859-1

La norme ASCII convient bien à la langue anglaise, mais pose des problèmes dans d'autres langues, par exemple le français. En effet l'ASCII ne prévoit pas d'encoder les lettres accentuées. C'est pour répondre à ce problème qu'est née la **norme ISO-8859-1**. Cette norme reprend les mêmes principes que l'ASCII, mais les nombres binaires associés à chaque caractère sont codés sur 8 bits, ce qui permet d'encoder jusqu'à 256 caractères Cette norme va être principalement utilisée dans les pays européens puisqu'elle permet d'encoder les caractères utilisés dans les principales langues européennes.

Unicode

Le but de la norme **Unicode** est de pouvoir encoder tous les caractères que l'on peut rencontrer dans le monde. Unicode est uniquement une table qui regroupe tous les caractères existant au monde, il ne s'occupe pas de la façon dont les caractères sont codés dans la machine. Unicode accepte plusieurs systèmes de codage : UTF-8, UTF-16, UTF-32. Le plus utilisé, notamment sur le Web, est UTF-8. Pour encoder les caractères Unicode, UTF-8 utilise un nombre variable d'octets : les caractères usuels (les plus couramment utilisés) sont codés sur un octet, alors que des caractères moins courants sont codés sur un nombre d'octets plus important (jusqu'à 4 octets). Il y a compatibilité entre ASCII et UTF-8 puisque les caractères Unicode codés avec UTF-8 ont exactement le même code que les mêmes caractères en ASCII.

CE QU'IL FAUT SAVOIR FAIRE

Convertir un fichier texte dans différents formats d'encodage.

Représentation des entiers relatifs

CE QU'IL FAUT SAVOIR

- Connaître le principe du **complément à deux**
- Savoir que pour une représentation sur n bits, il est possible de coder des valeurs comprises entre -2^{n-1} et $2^{n-1}-1$

CE QU'IL FAUT SAVOIR FAIRE

Savoir convertir, dans les deux sens, un entier (positif ou négatif) en utilisant le complément à deux sur n bits

Représentation des réels

CE QU'IL FAUT SAVOIR

- La représentation en machine des **nombres réels** (on parle souvent en informatique de **nombres flottants)** diffère de la représentation en machine des entiers
- La **norme IEEE 754** est la norme la plus employée pour la représentation des nombres à virgule flottante dans le domaine informatique
- Il existe deux formats associés à la norme IEEE 754 : le **format simple précision** (le nombre est représenté sur 32 bits) et le **format double précision** (le nombre est représenté sur 64 bits)
- Que cela soit en simple précision ou en double précision, la norme IEEE754 utilise :
  - 1 bit de **signe** (1 si le nombre est négatif et 0 si le nombre est positif) ;
  - des bits consacrés à **l'exposant** (8 bits pour la simple précision et 11 bits pour la double précision) ;
  - des bits consacrés à la **mantisse** (23 bits pour la simple précision et 52 bits pour la double précision).
- À cause de la limitation de la taille de mantisse on peut dans certains cas avoir des erreurs d'arrondies, par exemple **`0.1 + 0,2`** n'est pas égal à **`0.3`**. On évitera ainsi de tester l'égalité entre deux flottants (par exemple **`0.1 + 0.2 == 0.3`** retourne Faux !)

CE QU'IL FAUT SAVOIR FAIRE

- Être capable de trouver la représentation en binaire d'un réel (par exemple **`0.1`**, **`0.25`** ou encore **`1/3`**)
- Connaitre les grands principes de la norme IEEE754 (bit de signe, mantisse, exposant)

Traitements de données en tables

CE QU'IL FAUT SAVOIR

Un fichier **CSV** (*comma separated values*) est un fichier texte. Chaque ligne du texte correspond à une ligne d'un tableau et les virgules correspondent aux séparations entre les colonnes.

Le contenu suivant...

```
Nom,Prénom,Date de naissance
Durand,Jean-Pierre,23/05/1985
Dupont,Christophe,15/12/1967
Terta,Henry,12/06/1978
```

... permet d'avoir l'équivalent de :

Nom	Prénom	Date de naissance
Durand	Jean-Pierre	23/05/1985
Dupont	Christophe	15/12/1967
Terta	Henry	12/06/1978

CE QU'IL FAUT SAVOIR FAIRE

Être capable, grâce au module **`csv`**, d'effectuer des requêtes sur des données au format **CSV**

Complexité des algorithmes

CE QU'IL FAUT SAVOIR

- Un **algorithme** est la spécification d'un schéma de calcul sous forme d'une suite finie d'opérations élémentaires obéissant à un enchaînement déterminé
- Quand on écrit un algorithme, on utilise un langage dit **langage naturel** ("tant que", "si"...), ce langage naturel permet de passer facilement à un **langage de programmation** (Python, Java...), on dit alors que l'on **implémente** l'algorithme
- La notion de **complexité d'un algorithme** va rendre compte de l'efficacité de cet algorithme. Il existe deux types de complexité : une **complexité en temps** et une **complexité en mémoire.**
- La complexité en temps est directement liée au nombre d'opérations élémentaires qui doivent être exécutées afin de résoudre un problème donné
- Nous nous intéresserons uniquement à la complexité en temps **dans le pire cas**
- Pour **comparer des algorithmes,** nous allons uniquement nous intéresser à ce que l'on appelle **l'ordre de grandeur asymptotique** (notation grand "O" : $\mathcal{O}$)

CE QU'IL FAUT SAVOIR FAIRE

- Être capable d'analyser le fonctionnement d'un algorithme (faire tourner un algorithme **à la main**)
- Être capable de déterminer la complexité en temps dans le pire des cas d'un algorithme
- Exemples de coût en temps :
  - $\mathcal{O}(1)$,
  - $\mathcal{O}(\log_2(n))$,
  - $\mathcal{O}(n)$,
  - $\mathcal{O}(n \cdot \log_2(n))$,
  - $\mathcal{O}(n^2)$,
  - $\mathcal{O}(n^k)$,
  - $\mathcal{O}(k^n)$,
  - $\mathcal{O}(n!)$).

Correction des algorithmes

CE QU'IL FAUT SAVOIR

- On appelle **invariant de boucle** une propriété qui est vraie avant et après chaque itération (boucle)
- Un invariant de boucle peut permettre de prouver la **correction d'un algorithme**
- Il est nécessaire de démontrer que l'on a bien un invariant de boucle, cette démonstration doit se faire en trois étapes :
  - **initialisation** : on doit montrer que l'invariant de boucle est vrai avant la première itération de la boucle ;
  - **conservation** : on doit montrer que si l'invariant de boucle est vrai avant une itération de la boucle, il le reste avant l'itération suivante ;
  - **terminaison** : une fois la boucle terminée, l'invariant fournit une propriété utile qui aide à montrer la correction de l'algorithme.

CE QU'IL FAUT SAVOIR FAIRE

- Être capable de trouver un invariant de boucle dans un algorithme
- Être capable de prouver que c'est bien un invariant de boucle
- Être capable d'utiliser cet invariant de boucle pour démontrer la correction d'un algorithme (notamment les algorithmes de tri par insertion et de tri par sélection)

Les réseaux

CE QU'IL FAUT SAVOIR

- Des ordinateurs sont dits **en réseau** quand ils sont capables de communiquer entre eux soit par l'intermédiaire d'un **câble** (ex : câble Ethernet avec prises RJ45) ou d'une **connexion radio** (ex : Wifi)
- Pour relier plus de deux ordinateurs, il est nécessaire d'utiliser un **commutateur** (*switch* en anglais)
- Pour identifier un ordinateur sur un réseau, on utilise une **adresse IP.** Les adresses IP sont de la forme : **a.b.c.d**, avec **a**, **b**, **c** et **d** des nombres entiers compris entre 0 et 255 (**a**, **b**, **c** et **d** sont codés sur un octet).
- Une partie de l'adresse IP permet d'identifier le réseau auquel appartient la machine et l'autre partie de l'adresse IP permet d'identifier la machine sur ce réseau (exemple : soit l'adresse IP d'un ordinateur : 192.168.3.1/24, 192.168.3 correspond à la partie réseau de l'adresse IP et 1 correspond à la partie machine de l'adresse IP)
- Toutes les machines appartenant au même réseau devront posséder la **même adresse réseau** (sinon elles ne pourront pas communiquer ensemble, même si elles sont bien physiquement reliées).
- les adresses IP (**a.b.c.d**) n'ont forcément pas les parties **a**, **b** et **c** consacrées à l'identification du réseau et la partie **d** consacrées à l'identification des machines sur le réseau : on ajoute à l'adresse IP un "/" suivi du nombre 8, 16 ou 24
  - si ce nombre est 8 (exemple : 192.168.2.1/8), cela signifie que pour une adresse a.b.c.d/8, la partie a est consacrée à l'adresse réseau, le reste (b, c, d) est consacré à la partie machine de l'adresse IP. On aura donc une adresse réseau de la forme a.0.0.0 ;
  - si ce nombre est 16 (exemple : 192.168.2.1/16), cela signifie que pour une adresse a.b.c.d/16, les parties a et b sont consacrées à l'adresse réseau, le reste (c, d) est consacré à la partie machine de l'adresse IP. On aura donc une adresse réseau de la forme a.b.0.0 ;
  - si ce nombre est 24 (exemple : 192.168.2.1/24), cela signifie que pour une adresse a.b.c.d/24, les parties a, b et c sont consacrées à l'adresse réseau, le reste (d) est consacré à la partie machine de l'adresse IP. On aura donc une adresse réseau de la forme a.b.c.0.

CE QU'IL FAUT SAVOIR FAIRE

Être capable d'attribuer des adresses IP à des ordinateurs

Le protocole TCP/IP

CE QU'IL FAUT SAVOIR

- Au début des années 70, le projet **ARPAnet** (ancêtre d'Internet) avait pour but de relier plusieurs réseaux informatiques entre eux
- Les **protocoles TCP** et **IP** s'imposent comme des standards au niveau du projet ARPAnet
- Pour transférer des données entre deux ordinateurs, **le protocole TCP encapsule ces données**. Les données issues de l'encapsulation de TCP sont elles-mêmes encapsulées par le protocole IP. On obtient alors un paquet IP qui pourra être transmis sur le réseau. **Une fois arrivées à destination, les données seront désencapsulées**
- **Le protocole IP s'occupe uniquement de faire arriver à destination les paquets en utilisant l'adresse IP de l'ordinateur de destination**. Les adresses IP de l'ordinateur de départ (ordinateur A) et de l'ordinateur destination (ordinateur B) sont ajoutées aux paquets de données.
- **Le protocole TCP permet de s'assurer qu'un paquet est bien arrivé à destination grâce à un système d'accusé de réception**. Si l'ordinateur A ne reçoit pas cet accusé de réception en provenance de B, après un temps prédéfini, l'ordinateur A renverra le paquet de données vers l'ordinateur B.
- Le **protocole UDP** ressemble beaucoup au protocole TCP (IP peut encapsuler UDP à la place de TCP), seule différence notable : **le protocole UDP ne propose pas de système d'accusé de réception**.
- **TCP/IP repose sur la notion de paquets de données** : un fichier n'est pas envoyé en une seule fois, les données (bits) sont **découpées en petits paquets**. Ces paquets de données n'empruntant pas forcément tous le même chemin sur le réseau, l'ordinateur devra les remettre dans l'ordre avant de pouvoir reconstituer le fichier d'origine. Si un paquet se perd, le fichier ne pourra pas être reconstitué, il devra donc être renvoyé

Systèmes d'exploitation

CE QU'IL FAUT SAVOIR

- Un **système d'exploitation** (**OS** : *Operating System)* est un un ensemble de programme permettant de faire fonctionner les logiciels sur un ordinateur donné. L'OS répartit les ressources d'un ordinateur (CPU, RAM...) entre les différents logiciels qui tournent sur cet ordinateur
- Il existe des **systèmes d'exploitation libres** (GNU/Linux) et des **systèmes d'exploitation propriétaires** (Unix, Windows, macOS - qui est un dérivé d'UNIX -, iOS, etc.)
- Un **logiciel libre** est un logiciel dont l'utilisation, l'étude, la modification et la duplication par autrui en vue de sa diffusion sont permises, techniquement et légalement, ceci afin de garantir certaines libertés induites, dont le contrôle du programme par l'utilisateur et la possibilité de partage entre individus (d'après Wikipédia)
- En 1991, un étudiant finlandais, **Linus Torvalds,** décide de créer un clone libre d'UNIX en ne partant de rien puisque le code source d'UNIX n'est pas public. Ce clone d'UNIX va s'appeler **Linux** (Linus+UNIX). Ce noyau d'OS (le noyau assure uniquement les fonctions de base de l'OS) sera complété par des logiciels libres issus du **projet GNU** (GNU's Not UNIX) fondé par **Richard Stallman,** pour donner **GNU/Linux**

Lignes de commande

CE QU'IL FAUT SAVOIR

- Linux (plus généralement les systèmes de type Unix) propose un système de fichier en arborescence.
- Pour indiquer la position d'un fichier (ou d'un répertoire) dans l'arborescence, il existe 2 méthodes : indiquer un chemin absolu ou indiquer un chemin relatif (vous devez être capable de donner un chemin relatif et un chemin absolu pour un fichier donné dans une arborescence donnée)
- Voici une liste de commandes Unix (donc utilisable sous Linux et macOS) que vous devez connaître :
  - **`cd`** permet de changer le répertoire courant
  - **`ls`** permet de lister le contenu du répertoire courant
  - **`pwd`** permet de connaitre le répertoire courant
  - **`mkdir`** permet de créer un répertoire dans le répertoire courant
  - **`rm`** permet de supprimer un fichier ou un répertoire
  - **`touch`** permet de créer un fichier vide
  - **`cat`** permet d'afficher dans la console le contenu d'un fichier
  - **`cp`** permet de copier un fichier
  - **`mv`** permet de déplacer un fichier
  - **`man`** permet d'obtenir la documentation d'une autre commande
- Les systèmes de type **UNIX** sont des **systèmes multi-utilisateurs** ; plusieurs utilisateurs peuvent donc partager un même ordinateur, chaque utilisateur possédant un environnement de travail qui lui est propre. Chaque utilisateur possède certains droits lui permettant d'effectuer certaines opérations et pas d'autres
- Un utilisateur un peu particulier est autorisé à modifier tous les droits : ce **super-utilisateur** est appelé **administrateur** ou **root.** L'administrateur pourra donc attribuer ou retirer des droits aux autres utilisateurs
- Au lieu de gérer les utilisateurs un par un, il est possible de créer des **groupes d'utilisateurs.** L'administrateur attribue des droits à un groupe au lieu d'attribuer des droits particuliers à chaque utilisateur
- Les fichiers et les répertoires possèdent trois types de droits :
  - les **droits en lecture** (symbolisés par la lettre **r**) : est-il possible de lire le contenu de ce fichier ?
  - les **droits en écriture** (symbolisés par la lettre **w**) : est-il possible de modifier le contenu de ce fichier ?
  - les **droits en exécution** (symbolisés par la lettre **x**) : est-il possible d'exécuter le contenu de ce fichier (quand le fichier du code exécutable) ?
- Il existe trois types d'utilisateurs pour un fichier ou un répertoire :
  - le **propriétaire** du fichier (par défaut c'est la personne qui a créé le fichier), il est symbolisé par la lettre **u** (de **user**)
  - un fichier est associé à un **groupe,** tous les utilisateurs appartenant à ce groupe possèdent des droits particuliers sur ce fichier. Le groupe est symbolisé par la lettre **g** (de **group)**
  - tous les autres utilisateurs (ceux qui ne sont pas le propriétaire du fichier et qui n'appartiennent pas au groupe associé au fichier). Ces utilisateurs sont symbolisés la lettre **o** (de **others)**
- Le propriétaire d'un fichier (ou root) peut modifier les **permissions** d'un fichier ou d'un répertoire à l'aide de la commande **`chmod`**

CE QU'IL FAUT SAVOIR FAIRE

Être capable d'utiliser toutes les commandes listées ci-dessus

Les tris simples

CE QU'IL FAUT SAVOIR

- Connaître l'algorithme du **tri par insertion**
- Connaître l'algorithme du **tri par sélection**
- Savoir que l'algorithme du tri par insertion et l'algorithme du tri par sélection ont tous deux une **complexité en temps dans le pire des cas en $O(n^2)$ (quadratique)**

CE QU'IL FAUT SAVOIR FAIRE

- Être capable d'analyser et d'expliquer (faire tourner **à la main**) les algorithmes de tri par insertion et par sélection sur un exemple donné
- Être capable **d'implémenter en Python** les algorithmes de tri par insertion et par sélection

Algorithmes gloutons

CE QU'IL FAUT SAVOIR

- Un algorithme est dit **glouton** s'il se base sur une méthode gloutonne pour résoudre un **problème d'optimisation**
- Dans une méthode gloutonne, on fait des **choix localement optimaux** dans l'espoir que ces choix mèneront à une solution globalement optimale. Ces choix ne seront jamais remis en cause au cours de la résolution du problème (pas de retour en arrière possible)
- **Une méthode gloutonne ne donne pas forcément une solution optimale**

CE QU'IL FAUT SAVOIR FAIRE

Écrire un programme Python permettant de résoudre un problème d'optimisation (sac à dos ou rendu de monnaie) à l'aide d'un algorithme glouton

Intelligence artificielle

APPRENTISSAGE AUTOMATIQUE

L'**apprentissage automatique** (*machine learning* en anglais) est un domaine de l'**intelligence artificielle** qui utilise des techniques pour donner aux systèmes informatiques la possibilité "d'apprendre" (par exemple, d'améliorer progressivement les performances d'une tâche spécifique) à partir de données, sans être explicitement programmés.

En d'autre termes, c'est la science qui consiste à amener les ordinateurs à apprendre et à agir comme les humains, à améliorer leur apprentissage de manière autonome en leur fournissant des données et des informations sous forme d'observations et d'interactions dans le monde réel. L'apprentissage automatique est défini par une large gamme d'**algorithmes d'apprentissage**. Les algorithmes ne sont pas tous destinés aux mêmes usages. On les classe généralement selon deux catégories :

- le **mode d'apprentissage** : on distingue les algorithmes **supervisés** des algorithmes *non supervisés* ;
- le **type de problème à traiter** : on distingue les algorithmes de **régression** de ceux de **classification.**

Pour différencier les algorithmes d'apprentissage supervisé des algorithmes d'apprentissage non supervisé, prenons un exemple. Imaginons que l'on fournisse à l'algorithme des images représentant des chats et des chiens. S'il s'agit d'un algorithme supervisé, c'est alors nous, humains, qui allons proposer les critères de classement ("chat" ou "chien"). L'algorithme non supervisé doit par contre trouver lui–même les critères pour les organiser en groupe : on parle alors de **clustering.**

Concernant le type de problèmes que les algorithmes d'apprentissage peuvent traiter, la distinction classification/régression n'existe que dans le cas de l'apprentissage supervisé. L'exemple précédent est évidemment un algorithme de classification : c'est un chien ou un chat. On appelle cela une valeur discrète. Il n'y a pas de possibilité intermédiaire. À l'inverse, la régression permet d'obtenir un nombre réel. À partir d'un échantillon contenant les données d'habitation (quartier, superficie, etc.), l'algorithme sera en mesure de proposer un prix pour une nouvelle habitation.

ALGORITHME DES K PLUS PROCHES VOISINS

L'**algorithme des k plus proches voisins** est un **algorithme d'apprentissage supervisé**. Il est souvent noté k-NN pour *k-nearest neighbors* en anglais. Pour cet algorithme, nous devons disposer d'une **base de données d'apprentissage** qui servira à **entraîner notre algorithme**, qui pourra alors être utilisé sur des nouvelles données. La qualité de l'algorithme dépend donc des données présentes dans cette base de données. Trop peu nombreuses ou représentatives, l'algortihme ne pourra pas être utilisé efficacement.

Lorsqu'on exécute l'algorithme sur une nouvelle donnée, la méthode consiste à prendre en compte les k échantillons d'apprentissage les plus proches pour calculer le résultat. Dans l'exemple précédent, si **`k = 1000`** et que, parmi ces mille images, 986 sont des représentations de chiens (donc 14 sont des chats) alorsl'algorithme k-NN va considérer que la nouvelle image est un chien. Dans ce calcul, la notion de **distance entre échantillons** est primordiale et doit être définie en fonction des données et du fonctionnement attendu.

L'algorithme des k plus proches voisins est communément utilisé pour faire de la **classification** ou de la **régression.** En classification, l'algorithme est utilisé pour déterminer à quelle classe appartient la donnée. La classe résultat est typiquement la **classe majoritairement observée dans les k plus proches voisins**. Dans l'exemple, il y avait deux classes : soit "chien", soit "chat". Si k vaut 1, alors la donnée sera de la même classe que son plus proche voisin. En régression, le résultat n'est pas une classe mais un résultat calculé typiquement à partir de la moyenne des valeurs des voisins. C'était le cas de l'évaluation du prix d'une maison.