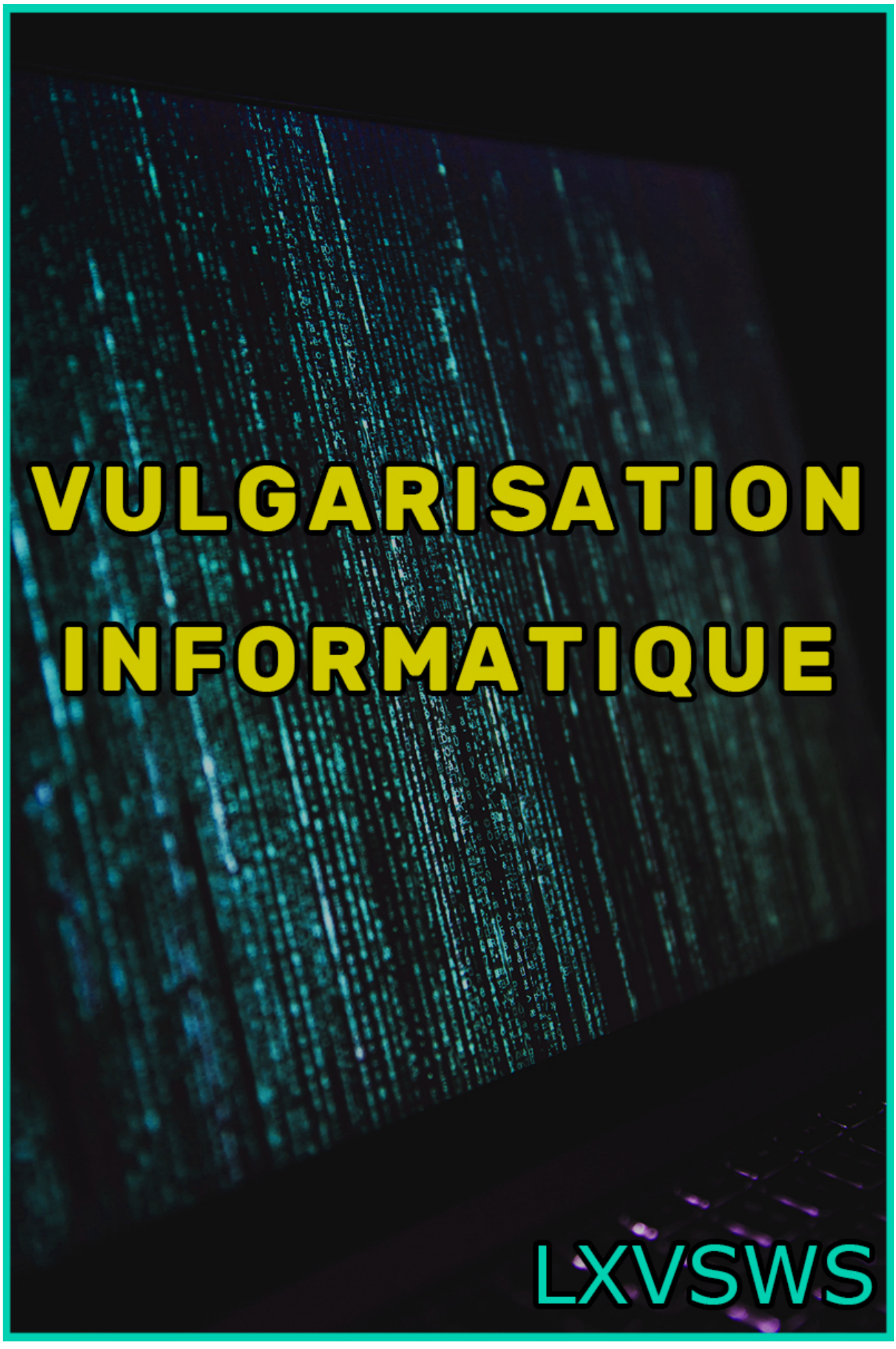

VULGARISATION
INFORMATIQUE
LXVSWS

Sommaire

5 – Programmation

5.1. Algorithmique

5.2. Complexité algorithmique

5.3. Application en informatique

5.4. Bugs

5.5. Données & Variables

5.6. Flux d’exécution

5.7. Construction

5.8. Programmation orientée objet

5.9. Langages

6 – Réseau

6.1. Adresse MAC

6.2. Adresse IP

6.3. Masques de sous-réseau

6.4. Protocoles de communication

6.5. Protocoles Ethernet

6.6. Ports

6.7. Sockets

6.8. Couches de protocoles

6.9. Modèle OSI

10 – Conclusion

L’information

Étant à la fois message et messager, l’information est ce qui lie notre expérience du monde avec le monde lui-même. Elle désigne à la fois le message à communiquer et le moyen utilisé pour le transmettre.

Lorsqu’on perçoit un ou plusieurs signes grâce à nos organes sensoriels, il y a information. Cette information est transmise via un **signal** que peuvent percevoir nos sens afin de nous mettre en contact avec l’élément d’où provient ce signal.

Les signaux se différencient par :

- L’amplitude (intensité du signal)
- Le genre (catégorie du signal)
- La nature (type physique du signal)

La somme de ces données représente un **code** qui doit être analysé et décodé pour avoir l’information. Le cerveau (ou la machine) interprète ce code en données adaptées.

La façon dont une information est attribuée à un nombre, chiffre ou symbole particulier est définie par le **codage**.

Codage de l’information

Pour transporter et stocker des données, il existe beaucoup de procédés, chacun pour un type d’informations (texte, image, son, etc.).
On peut cependant en distinguer deux principaux :

- L’**analogique** : Né avec le début de l’électricité.

Utilise des nombres *réels*. Consiste à reproduire fidèlement le signal sur un support magnétique. Multitude théorique d’amplitudes et de valeurs.

- Le **numérique** : Apparu à l’ère de l’informatique.

Utilise des nombres *entiers*. Transforme le signal en deux variations seulement, par intervalles.

Support de l’information

L’élément d’où provient l’information n’est pas déterminé. Dans le cas de l’ordinateur, c’est une tension électrique qui est le support.

La tension (*d’alimentation*) est reliée aux circuits, et alimente ces derniers en énergie.

Cette tension d’alimentation est manipulée par les circuits en tant que source de l’information, qui sera ensuite codée. Même le zéro a besoin d’une tension d’alimentation (appelée *la masse*).

Les tensions sont mesurées en **volts.**

L’informatique

L’informatique est la science de l’information automatisée. Elle transforme des informations à l’état brut en des données exploitables et structurées, ce qui permet d’améliorer considérablement les transmissions d’informations grâce à des machines mécaniques (ordinateurs).

L’ordinateur stocke les données via le codage **numérique**. Les données manipulées en informatique représentent l’information en entités binaire, ce qui veut dire qu’uniquement deux valeurs sont possibles. Pour les ordinateurs, ce sera soit 0, soit 1.

Cette entité est appelée un **bit**. C’est l’entité la plus petite d’information. Un bit représente donc soit 1 soit 0.

Le **protocole** définit la signification des bits échangés entre machines (cf. chapitre 6 « Réseau » à venir).

Binaire

Le codage numérique d’une tension électrique par les composants d’un ordinateur se base sur le codage *NRZ* pour coder un bit.

De 0 volt à un certain seuil, la tension vaudra un bit 0 (ou 1) .
De ce seuil à un seuil plus haut, il y a une *marge de sécurité*.
De ce seuil au maximum de la tension, la tension vaudra un bit 1 (ou 0).

L’octet

Un octet est une entité composée de 8 bits (256 valeurs possibles). C’est avec ce type d’entité que travaille un système informatique.

Base

Le nombre de chiffres utilisés détermine la base.

Dans la vie de tous les jours, nous utilisons la base 10 (0 à 9), donc 10 chiffres différents. Le nom de la base détermine la puissance à utiliser.

En informatique, nous utilisons principalement la base 2 (système binaire) et la base 16 (système hexadécimal). Quand les 10 chiffres ne suffisent plus, nous ajoutons les 6 premières lettres de l’alphabet latin.

Les circuits d’un ordinateur ne peuvent manipuler que des nombres entiers, de 4/8/12/16/32/48/64 bits maximum.
Les nombres plus grands ou plus petits que 64 bits devront être décomposés.

Matériel

L’appellation anglophone de « **matériel** » est « **hardware** ».

Regroupe tous les composants d’appareils électroniques.

Appareils électroniques

Un appareil électronique est composé de plusieurs composants soudés sur une carte (un support plat). Ces composants sont reliés par des fils conducteurs, ce qui leur permet de s’échanger des données. Tous les composants ne sont pas forcément tous reliés entre eux. L’association d’une carte et de composants reliés forme un circuit électronique.

Les appareils simples sont non-programmables, c’est-à-dire que les circuits électroniques de l’appareil sont conçus pour répondre à une tâche spécifique.

Les appareils programmables ne sont pas conçus pour une utilisation particulière. Il est possible de modifier leur fonction grâce aux logiciels. Ces derniers sont transmis aux circuits via les périphériques.

L’ordinateur est un appareil programmable contenant plusieurs composants séparés, aux fonctions distinctes.
L’**unité centrale** regroupe les composants indispensables (carte mère, mémoire, processeur, entrées-sorties…).
Les **périphériques** regroupent les composants facultatifs, qui se connectent aux entrées-sorties de l’unité centrale et permettent à l’utilisateur d’interagir avec l’ordinateur.

Types de périphériques

- *Périphérique d'entrée* → Numérise des informations physiques en représentations numériques (clavier, souris…).

- *Périphérique de sortie* → Transmet des informations sous forme analogique (écran…).

- *Périphérique de stockage* → Réunit plusieurs types de mémoire (disque dur, clé usb…). Permet à l'ordinateur de stocker des informations.

Les périphériques sont reliés à la carte mère par un connecteur. Le *système d'exploitation* (cf. chapitre 4 « Logiciel » à venir) doit disposer d'un logiciel **pilote** (ou *driver*) pour permettre au périphérique de fonctionner.

Par exemple, un pilote d'imprimante est un logiciel qui traduit les ordres de l'utilisateur dans le langage de l'imprimante.

Portes logiques

Les portes logiques sont des circuits électroniques possédant des entrées et sorties. Une tension est envoyée via un fil conducteur en entrée (donc un bit). Le circuit va réagir et renvoyer un bit en sortie.

- **porte *NON*** : un bit en entrée, renvoie son bit contraire.
- **porte *ET*** : plusieurs entrées, renvoie 1 si toutes ses entrées valent 1, sinon 0.
- **porte *NAND*** : plusieurs entrées, renvoie 0 si toutes ses entrées valent 1, sinon 1.
- **porte *OU*** : plusieurs entrées, renvoie 1 si au moins une entrée vaut 1, sinon 0.

- **porte *NOR*** : plusieurs entrées, renvoie 0 si au moins une entrée vaut 1, sinon 1.
- **porte *XOR*** : renvoie 1 si les deux bits en entrée sont différents, sinon 0.
- **porte *NXOR*** : renvoie 1 si les deux bits en entrée sont identiques, sinon 0.

En combinant ces portes logiques basiques, nous pouvons fabriquer des portes logiques complexes. On peut fabriquer tous les circuits possibles avec une seule ou plusieurs portes logiques. Les portes logiques sont fabriquées avec des composants électroniques reliés entre eux que l'on nomme **transistors**. Ces derniers, ainsi que le fil électrique conducteur, forment des circuits électroniques plus ou moins compliqués reproduisant les différentes portes logiques.

Matériel réseau

Le câble réseau des connexions Ethernet est le câble à paires torsadées (ou **RJ45**). Il comporte plusieurs catégories :

- **Cat 5** bas débit (< 100 MHz)
- **Cat 5e** ADSL + fibre (< 155 MHz)
- **Cat 6** fibre (< 250 MHz)
- **Cat 6a** fibre (< 500 MHz)
- **Cat 7** fibre (< 600 MHz)
- **Cat 7a** fibre (< 1000 MHz)

On distingue le matériel connecté (machine avec carte réseau possédant une prise RJ45 femelle) et le matériel de connexion (hubs & switchs) reliant les machines sur le réseau.

Hub topologie en bus (machines connectées entre elles)
Switch topologie en étoile (machines connectées à un switch)

Logiciel

L’appellation anglophone de « **logiciel** » est « **software** ».

Ce sont des outils virtuels utilisés pour manipuler des informations stockées dans la mémoire de l’ordinateur. Chacun d’eux est conçu pour répondre à une tâche spécifique, via des traitements appelés fonctionnalités.

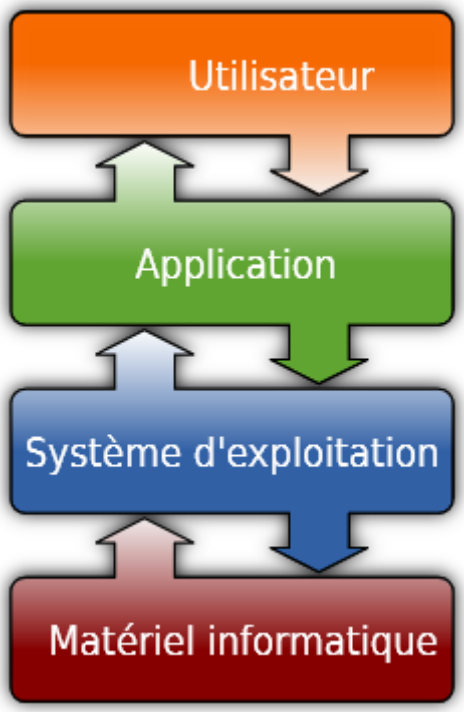
Sans logiciel, l’ordinateur ne fait rien parce qu’il n’a pas reçu les instructions lui indiquant ce qu’il doit faire. Un logiciel est un ensemble composé de plusieurs programmes ainsi que tout le nécessaire pour les rendre opérationnels.

Les deux principales catégories de logiciels sont les **applications** et les *programmes système.*

L’ensemble des *programmes système* se nomme **système d’exploitation**. Les applications lui délèguent la gestion de la mémoire et des périphériques.

Système d’exploitation et noyau

Le système d’exploitation (ou **OS** « *Operating System* ») s’occupe de la gestion du matériel. Il réalise le lien entre la machine et l’utilisateur. Ce travail est effectué par un programme central, le noyau (ou **kernel**) qui gère la mémoire, fournit aux applications des points d'entrée pour les périphériques et permet, grâce à une couche d’abstraction, l’exécution de programmes sur des ordinateurs avec des composants différents.



Les trois principaux systèmes d'exploitation sont **Linux**, **Mac OS** et **Windows**.

Un chapitre sera consacré à l'apprentissage de Linux, afin de vous permettre de faire vos premiers pas sur ce système d'exploitation très répandu parmi les professionnels de l'informatique.

Applications

Une application est un programme destiné à l'utilisateur et utilisé pour réaliser une tâche ou un ensemble de tâches.
Les applications s'exécutent en utilisant les services du système d'exploitation.

L'ensemble des activités qui permettent l'écriture des programmes se nomme la **programmation** (ou *codage*).

Programmation

La programmation consiste à écrire ce que l'on désire faire faire à la machine dans un langage formalisé nommé « langage de programmation ».

L'activité essentielle d'un programmeur consiste à résoudre des problèmes. Grâce à la logique, il décrit à la machine les actions à effectuer, ainsi que leur ordre dans des procédures nommées **algorithmes**.

Algorithmique

Un algorithme est une suite d'instructions/opérations permettant de résoudre un problème et/ou d'obtenir un résultat. Cette description précise, sous forme de concepts simples, permet de concevoir des méthodes de traitement d'information.
Ces méthodes sont inséparables des structures de données, qui décrivent une organisation.

Réduire la complexité de ces méthodes & choisir la bonne structure de données → augmenter la performance des algorithmes.

Complexité algorithmique

C'est la mesure formelle de la complexité d'un algorithme, exprimée en *formule mathématique*.

Avant tout, il faut vérifier que l'algorithme produise bien le résultat attendu. Même si l'algorithme est correct, des bugs peuvent survenir (dû au langage utilisé).
Une fois ces étapes vérifiées, il faut optimiser les performances du programme en réduisant la complexité de l'algorithme. On évalue la complexité algorithmique en fonction du nombre d'opérations effectuées (le temps) sur les conditions de départ (l'entrée, de taille changeante).

Application en informatique

Les ordinateurs sont efficaces pour appliquer concrètement des algorithmes, via un **langage de programmation**. Cela donne des **programmes**. Ils contiennent la transcription en langage informatique d'un ou plusieurs algorithmes.

Le degré de précision dépend de nombreux paramètres (langage utilisé, bibliothèques, etc.).
Le texte brut écrit sous la forme d'un langage informatique est appelé le **code source** du programme, et sera ensuite traduit en langage machine pour pouvoir s'exécuter.

Bugs

Si le programme n'a pas le comportement désiré, 3 types d'erreurs sont possibles :

- erreurs de syntaxe
- erreurs sémantiques (logique)
- erreurs à l'exécution (exceptions)

Données & variables

Un programme manipule essentiellement des données (en réalité, une suite finie de nombres binaires). Pour y accéder, il fait usage de **variables** de différents types (entier, réel, liste, etc.).

Pour la machine, ce sont des références désignant une adresse mémoire (un emplacement dans la mémoire vive) où est stockée la suite de nombres binaires (et donc la donnée).

Flux d'exécution

L'ordre des actions dans un programme est un *flux d'exécution.* Les structures de contrôle déterminent l'ordre des actions à exécuter :

- **séquence** : instructions s'exécutant les unes après les autres
- **sélection** : exécution conditionnelle (conditions)
- **répétition** : instructions répétitives (boucles)

Construction

Deux modèles sont couramment utilisés en programmation :

- la méthodologie **procédurale** → division des problèmes

- la méthodologie **objet** → classification des données en objets interagissant entre eux

Programmation orientée objet

Objet/classe = attributs/données + méthodes/fonctions

Les méthodes (ou fonctions) agissent sur les propriétés (données) de l'objet.

C'est la classe qui englobe les attributs & méthodes d'un type d'objet.

Les méthodes utilisables depuis l'extérieur constituent l'interface de l'objet.

Il est préférable d'utiliser uniquement les méthodes pour modifier un objet.

Bénéfices de la programmation orientée objet

- encapsulation
- dérivation (héritage)
- polymorphisme
- séparation du code
- ré-utilisabilité

Langages

Il existe une multitude de langages, chacun d'entre eux ayant ses propres spécificités. Voici une liste des plus connus :
Python, Java, C, C++, Javascript, C#, Ruby, PHP, etc.

Rendez-vous au chapitre 9 « Python » à venir pour une introduction à ce langage.

Réseau

Un réseau est un ensemble de machines reliées entre elles et qui échangent des informations.
Est appelé *matériel réseau* ce qui permet aux ordinateurs de communiquer entre eux.
Les machines présentes sur le réseau (clients & serveurs) communiquent via des liaisons décentralisées, connectées par des *câbles réseau*.

Adresse MAC

Elle identifie la carte réseau d'une machine. Toute carte réseau possède une adresse MAC. Elle est codée sur 6 octets (48 bits) en base 16 (hexadécimal).

ff:ff:ff:ff:ff:ff est l'adresse de *broadcast*. Elle représente et identifie n'importe quelle carte réseau du réseau et permet d'envoyer une requête à toutes les machines du réseau.

Adresse IP

Chaque machine possède une **adresse IP** (une suite de nombres séparée par des points) qui l'identifie sur le réseau. C'est l'adresse du réseau et de la machine.

Cette adresse IP est traduite en un *nom d'hôte* (une version textuelle) via les **DNS** (et inversement). Elles peuvent aller de *0.0.0.0* jusqu'à *255.255.255.255*.

Elle est codée sur 32 bits (4 octets), représentée en décimale pointée mais manipulée en binaire au niveau matériel et est inséparable du masque de sous-réseau.

Masques de sous-réseau

Le masque de sous-réseau indique quelle partie de l'adresse IP est la partie réseau et laquelle est la partie machine. Les bits à 1 (représentés par le nombre 255) indiquent la partie réseau. Les bits à 0 (représentés par le chiffre 0) indiquent la partie machine.
Écriture raccourcie : *IP/nombre de bits à 1.*

Une plage d'adresses est comprise dans la partie des bits à 0 du masque.
Début : tous les bits partie machine à 0.
Fin : tous les bits partie machine à 1.

La partie réseau est fixe, la partie machine, elle, varie.

1ère adresse d'une plage : le réseau lui-même
Dernière adresse : identifie toutes les machines du réseau

Protocoles de communication

Un protocole définit la signification des bits échangés entre machines. Les machines communiquent donc entre elles via des **protocoles**. Ces derniers indiquent comment les machines doivent communiquer.

Un protocole est une série d'étapes à suivre pour permettre une connexion/communication.

On distingue deux types de protocoles :

- protocoles **bas niveau** (TCP, UDP)
- protocoles **haut niveau**, surcouche de TCP ou UDP
Exemples : HTTP, HTTPS, FTP, SMTP…

Protocole Ethernet

Une trame Ethernet est un message envoyé sur le réseau. Elle possède une adresse MAC destinataire et une adresse MAC source (l'émetteur). Elle contient aussi le protocole de la couche adjacente, les données à transmettre et le *CRC* (gestion d'erreur).

Ports

Un **port** est représenté par un numéro spécifique et permet à une application de communiquer avec l'OS sans interférer avec les autres applications.

Sockets

Un **socket** représente une *interface de connexion* par laquelle une application peut envoyer et recevoir des données.

Cette interface permet à l'application de se brancher sur un réseau et communiquer avec d'autres applications qui y sont branchées. Les informations écrites sur une interface depuis une machine sont lues sur l'interface d'une autre machine, et inversement.

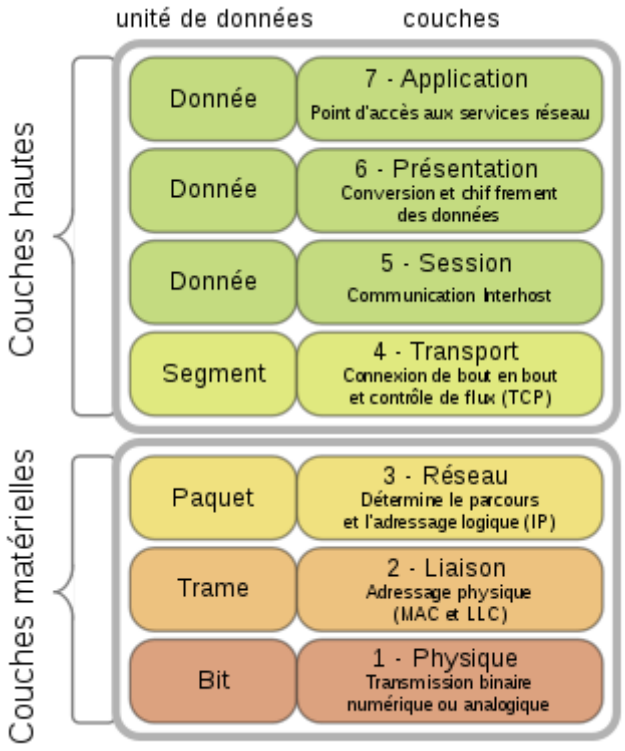
Couches de protocoles

Les protocoles sont hiérarchisés en couches pour décomposer et ordonner les différentes communications entre machines hétérogènes.

Un ensemble de couches est appelé un **modèle**. Il a pour objectif de constituer une référence théorique d'un réseau informatique. Un modèle est constitué de couches dont chacune correspond à une fonctionnalité particulière d'un réseau.

Modèle OSI

L'objectif théorique du modèle OSI est de normaliser les communications entre machines/ordinateurs.



Chaque couche :

- est indépendante, le protocole associé est donc interchangeable.
- ne peut communiquer qu'avec une couche adjacente
- a un rôle particulier à accomplir

Lors de l'envoi d'informations/données, on parcourt le modèle de haut en bas, traversant toutes les couches.

Modèle TCP/IP

Le modèle TCP/IP est une version simplifiée et plus réaliste du modèle OSI. Il se présente sous la forme suivante :

Application (HTTP, FTP, DNS…)
Transport (TCP, UDP)
Internet (IP)
Accès Réseau (Ethernet, Wifi, ATM, PPP…)

Comme pour le modèle OSI, lors de l'envoi d'informations (ou données), on parcourt le modèle de haut en bas, traversant toutes les couches.

Internet

Internet est le réseau informatique mondial accessible au public. Il englobe l'ensemble des protocoles du modèle TCP/IP, ce qui permet des applications variées (e-mails, messagerie instantanée, peer-to-peer, Web, etc.)

Web

Le web fait partie d'Internet. Le principe est de naviguer sur des pages reliées entre elles par des liens URL (*Uniform Resource Locator*) se présentant sous la forme :

http://www.site.com/ressource?
parametre=valeur&autreparametre=valeur

Il y a deux types de machines connectées au web :

- les **clients** : visualisent les sites web
- les **serveurs** : envoient les sites web aux clients

Une machine peut faire office de client et de serveur. Sur le web, deux types de langages informatiques sont utilisés :

- langages **frontend**
- langages **backend**

Les navigateurs traduisent les langages *frontend* en affichage, tandis que les serveurs traduisent les langages *backend* en comportements (la façon dont les pages sont produites).

Les données manipulées sur le web sont stockées dans des **bases de données**. Ces dernières communiquent avec les serveurs via le langage *SQL*.

Site web dynamique

Le serveur intègre des mécanismes automatiques de mises à jour des pages ; les données se modifient au fil du temps et sont produites à la volée. A la différence des sites web statiques, les pages sont générées par des programmes.

Le contenu final est le même (du HTML envoyé via HTTP). Ces programmes peuvent accéder à des bases de données pour gérer les données. Ils peuvent également recevoir des paramètres via l'URL pour agir différemment.

Protocole HTTP

HTTP (*Hypertext Transfert Protocol*) est un protocole de communication client-serveur pour le web. Il permet la communication entre clients et serveurs (par exemple un navigateur qui accède à un serveur contenant des données).

Il est basé sur le modèle **TCP/IP** (ou OSI). Son port est 80.
C'est un protocole de la couche *applicative*.

La version du protocole HTTP/1.1 permet plusieurs requêtes/réponses par connexion. Les données sont transmises en clair (non sécurisé).

HTTPS est une version sécurisée, qui utilise HTTP par dessus **SSL (TLS)** et permet donc un échange sécurisé d'informations. Son port est 443.

Les méthodes HTTP décrivent l'action à effectuer sur les données/ressources :

HTTP/1.0 : GET, POST, HEAD
HTTP/1.1 : OPTIONS, PUT, DELETE, TRACE, PATCH, CONNECT

Une méthode est une commande spécifiant une requête du serveur.

Requête HTTP

MÉTHODE URL HTTP/version de protocole
Nom d'en-tête : valeur

Corps de requête

Entre l'en-tête et le corps, il faut une ligne vide.

Réponse HTTP

HTTP/version code-réponse texte-réponse
En-tête : valeur
Corps de réponse (ex : HTML)

HTML & CSS

HTML est un langage de balisage conçu pour représenter l'architecture des pages web.

CSS est quant à lui un langage informatique qui décrit la présentation et le style des pages web.

Javascript

Javascript est un langage de programmation orienté objet à prototype principalement utilisé dans les pages web interactives ainsi que dans les applications web.

La majorité des navigateurs web disposent d'un moteur Javascript dédié pour l'interpréter.

DOM (Document Object Model)

Quand un fichier HTML est chargé dans le navigateur, il devient un objet Javascript **document** avec lequel on peut interagir.

Les propriétés et méthodes de l'objet document représentent les éléments et actions pouvant affecter le code HTML.

Linux

Parmi tous les systèmes d'exploitation disponibles, Linux est sans doute celui qui laisse l'utilisateur le plus libre de ses choix. Il est entièrement gratuit et libre d'utilisation. Le noyau est sous licence GNU GPL, le reste sous licences libres.

Il se base sur le système **UNIX**. Le nom original est GNU/Linux, Linux étant le noyau et GNU les logiciels l'accompagnant. Richard Stallman est à l'origine du projet GNU tandis que Linus Torvalds est à l'origine du noyau Linux.

Il n'est **pas** obligatoire que le noyau soit accompagné des logiciels GNU.

Distributions

Il existe plusieurs **distributions**, chacune ayant ses propres spécificités (généralement des logiciels différents entourant le noyau Linux). Voici quelques exemples de distributions :

Ubuntu, RedHat, Debian, CentOS, Android, KALI Linux, Archlinux, etc.

Pour utiliser un autre système d'exploitation que celui installé par défaut sur votre machine, plusieurs options s'offrent à vous :

- utiliser une **machine virtuelle** (un logiciel permettant de lancer un autre système d'exploitation au-dessus du vôtre sans changer quoi que ce soit à votre système)

- installer le système d'exploitation en **dualboot** (au démarrage de l'ordinateur, un choix vous sera proposé entre les deux OS)

- installer le nouveau système d'exploitation par-dessus l'ancien (perte de toutes les données car nécessite un formatage).

Quelle que soit la distribution choisie, des logiciels seront installés par défaut car ils sont indispensables à l'utilisation de base de l'OS. Pour exécuter ces logiciels, il vous faudra entrer des **commandes** dans la **console** (appelée aussi **terminal** ou **shell**).

Console

Une fois la console ouverte, celle-ci va vous afficher un message comme le suivant :

utilisateur@machine:~$

Ce message indique quel utilisateur exécutera la commande et sur quelle machine.

L’utilisateur peut être **root** et dans ce cas, le symbole à la fin sera un # et représentera le super utilisateur, celui ayant tous les droits sur la machine.

Répertoires

Sur Linux, tout est fichier. Cela signifie que tout est représenté comme étant des fichiers dans l’OS (même les périphériques).

Ces fichiers sont contenus dans des **répertoires** (ou *dossiers*)

permettant de les classer en arborescence afin de s’y retrouver.

L’adresse d’un répertoire se lit de gauche à droite et les répertoires séparés par des **/** (exemple : */mnt/usb/).*

Les principaux répertoires sont :

/	répertoire racine (contient tout le reste)
~	home (répertoire de l’utilisateur)
.	répertoire actuel
..	répertoire parent
bin	contient des programmes (exécutables)
boot	fichiers permettant le démarrage de Linux
dev	fichiers contenant les périphériques
/dev/null	répertoire effaçant tout ce qui se trouve dedans
etc	fichiers de configuration
home	répertoires personnels des utilisateurs

lib	bibliothèques partagées utilisées par les programmes
media	montage des périphériques amovibles
mnt	pareil que media mais pour un usage plus temporaire
opt	répertoire utilisé pour les add-ons de programmes
proc	contient des informations système
root	répertoire personnel de l'utilisateur "root"
sbin	programmes système importants
tmp	répertoire utilisé par les programmes pour stocker des fichiers temporaires
usr	contient la plupart des programmes installés par l'utilisateur
var	contient des données "variables" (logs, etc.)

Commandes

Voici une liste non exhaustive des principales commandes utilisées sous Linux :

sudo	obtenir droits root pour cette commande uniquement
su	obtenir droits root
pwd	afficher chemin du répertoire actuel
echo	afficher données ou variable
which	connaître l'emplacement d'une commande
history	affiche historique des commandes

exit	quitter root ou fermer shell
clear	nettoyer le shell
uptime	afficher la durée de fonctionnement de l'ordinateur
wh	afficher liste des connectés
ps	afficher liste statique des processus
top	afficher liste dynamique des processus
date	afficher date/heure/fuseau horaire
shutdown	éteindre système (requiert root)
reboot	redémarrer système
useradd	ajouter user
passwd	modifier mot de passe de l'utilisateur actuel
usermod	modifier user
userdel	supprimer user
ls	lister les fichiers et dossiers du répertoire actuel
cd /répertoire	changer de répertoire
cd ..	revenir au répertoire parent
cd	retourner au répertoire personnel
cat	afficher tout le contenu d'un fichier
less	afficher le contenu d'un fichier page par page
mv	déplacer fichier ou répertoire
grep	filtrer données
head	afficher premières lignes d'un fichier
tail	afficher dernières lignes d'un fichier
touch	créer fichier, apostrophes pour inclure espaces

mkdir	créer répertoire, apostrophes pour inclure espaces
cp	copier
rm	supprimer
ifconfig	interfaces réseau
netstat	statistiques réseau

Obtenir plus d'informations sur une commande

Parfois, écrire simplement une commande ne fonctionne pas, il faut y ajouter un ou des paramètres pour atteindre son plein potentiel. Pour connaître les paramètres à ajouter à une commande, consultez son manuel en la faisant précéder par *man* et naviguez dans la documentation :

Haut/Bas	déplacer ligne par ligne
Espace & PgUp/PgDown	déplacer page en page
Origine/Fin	aller au début/à la fin
/	faire une recherche
q	quitter manuel

Vous pouvez également consulter l'aide d'une commande en y ajoutant --*help.*

Utiliser la bonne commande en fonction de ce que vous voulez faire vous permettra de contrôler efficacement votre ordinateur.

Droits d’accès

Chaque fichier possède une configuration de droits d’accès qui lui est propre et affichée sous la forme suivante :

(-)(---)(---)(---)

1ère partie : type de dossier/lien/fichier

2ème partie : droits du propriétaire

3ème partie : droits des membres du groupe

4ème partie : droits des autres membres

Chaque tiret peut être remplacé par une lettre :

d l'élément est un dossier

l l'élément est un lien

r droit de lecture

w droit de modification

x exécutable si fichier ou accessible si répertoire

La commande *chmod* (*chmod -R* pour les répertoires) permet de modifier les droits d'accès (requiert d'être root).

Flux de redirection

Dans un système d’exploitation, les **entrées** sont les données envoyées par un périphérique à destination du processeur.

Les **sorties** sont les données émises par le processeur à destination d'un périphérique (disque, réseau, écran, etc.).

Il est possible de modifier ce comportement en utilisant certains sigles :

> rediriger sortie standard dans un fichier

>> rediriger sortie standard à la fin d'un fichier

2> rediriger sortie d'erreurs dans un fichier à part

2>> rediriger sortie d'erreurs à la fin d'un fichier

< entrer contenu d'un fichier dans une commande

<< entrer contenu du clavier dans une commande

commande1* | *commande2 entrer sortie commande1 dans entrée commande2

commande1* && *commande2 lance commande2 si commande1 s'est bien exécutée

Scripts

Une fois que vous serez familiarisé avec Linux, vous allez commencer à écrire vos propres scripts. Ceux-ci vous donneront un contrôle total sur votre système.

Pour cela, plusieurs langages sont disponibles et **Python** en fait partie. Dans le chapitre suivant, vous allez apprendre les bases de ce langage très populaire.

Python

Vous avez accumulé assez de connaissances pour commencer à programmer. Pour débuter, le langage Python est le candidat idéal. Sa syntaxe est intuitive et son apprentissage rapide.

Avec Python, vous pouvez écrire des scripts système, des applications et même des sites web.

Pré-requis

Pour commencer, il vous faudra un logiciel pouvant écrire du texte brut (sans mise en forme). Le bloc-notes par défaut de votre système d'exploitation vous permet cela mais personnellement je préfère l'utilisation du logiciel « *Visual Studio Code* ».

Le texte brut écrit en langage Python représentera le **code source** de votre programme, et devra être interprété par un interpréteur Python pour pouvoir s'exécuter.

Il devra être enregistré avec l'extension *.py*

Installation de l'interpréteur Python

L'interpréteur Python doit être installé sur votre machine pour pouvoir lancer les fichiers *.py*

Il est disponible en téléchargement à l'adresse :

https://www.python.org/downloads/

Une fois celui-ci installé sur votre machine, vous pourrez ouvrir les fichiers *.py* en tapant *python* suivi du nom de votre fichier dans la console Linux (ou l'invite de commandes Windows).

Bases

Python permet la manipulation de nombres réels et entiers ainsi que l'utilisation d'**opérateurs de calcul** :

Addition	:	+
Soustraction	:	-
Multiplication	:	*
Division réelle	:	/
Division entière	:	//
Reste de division (modulo)	:	%
Exponentiation (exposants)	:	**

Les nombres réels s'écrivent avec un point et non une virgule.

La première ligne d'un programme permet d'indiquer son encodage :

 # -- coding:encodage -*-*

Le texte après un # ne sera pas interprété (sauf la première ligne destinée à l'encodage), utile pour écrire des commentaires sur le code à destination des autres développeurs.

Une longue commande pourra être découpée avec \

Variables

L'utilité des variables a été décrite dans le chapitre « Programmation », voici comment les manipuler en Python :

Affectation	:	var1 = valeur
Affectation simultanée	:	var1 = var2 = valeur
Affectation parallèle	:	var1, var2 = valeur, valeur
Incrémentation	:	var1 += valeur
Permutation	:	var1, var2 = var2, var1
Association/Copie	:	var2 = var1 = var3

var1/2/3 sont à remplacer par les noms de vos variables.

Instructions

Le principe d’un langage de programmation est de donner des instructions à la machine dans un langage proche du nôtre.

Il existe des instructions **simples** (sur une ligne) et des instructions **composées** (en-tête + bloc d’instructions).

Python est un langage conçu pour être lisible et visuellement épuré. Les blocs d’instructions sont identifiés par **l’indentation**. Une indentation est représentée par une tabulation (un appui sur la touche TAB) ou quatre espaces.

```
en-tête:
    première instruction du bloc
    ...
    dernière instruction du bloc
```

Une instruction sans indentation s'exécute toujours. L'en-tête (qui définit si le bloc doit s'exécuter) finit par « : ». Il faut respecter l'indentation pour imbriquer plusieurs instructions composées et affecter les variables au préalable si besoin.

Conditions & Boucles

L’en-tête peut contenir une **condition** (*if* → *elif* → *else*)

Si celle-ci retourne le booléen *True* alors le bloc d’instructions sera exécuté.

Les **boucles** (*while / for)* permettent quant à elles de répéter le bloc d’instructions indéfiniment (ou un certain nombre

d’itérations) tant que l’en-tête retourne le booléen *True.*

Vous pouvez ajouter des **opérateurs de comparaison** ainsi que des **opérateurs logiques** à vos conditions et/ou boucles :

== Égal à
!= Différent de
< Strictement inférieur à
> Strictement supérieur à
<= Inférieur ou égal à
>= Supérieur ou égal à

Types de données

Chaque donnée manipulée possède un **type** dynamiquement déterminé par Python (vous n’avez pas à le déclarer).

Nombres entiers – *int()*
Nombres réels – *float()*
Booléens – *bool()*

Certains types de données sont des **séquences** (collections ordonnées d'éléments) :

Chaînes de caractères – *str()*
Listes – *list()*
Dictionnaires – *dict()*

Vous pouvez interagir avec chaque élément contenu dans une séquence grâce aux crochets :

variable[0]	accéder à un élément d'une séquence
variable[0:10]	accéder à plusieurs éléments d'une séquence

En programmation, on commence à compter à partir de **zéro**.

Fonctions

Une fonction apparaît sous la forme d'un nom quelconque associé à des **parenthèses**. Dans les parenthèses, on transmet à la fonction un argument (ou plusieurs, ou pas du tout).

La fonction renvoie une valeur en retour (ou rien dans le cadre d'une procédure).

Définir une fonction :

```
def nom_fonction(paramètre1, paramètre2=valeur):
    """documentation"""
    bloc d'instructions
    return résultat
```

Dans la définition d'une fonction, la variable particulière pour recevoir un argument transmis s'appelle un **paramètre**.

Le nom d'une variable que nous passons comme argument n'a rien à voir avec le nom du paramètre correspondant dans la fonction.

Lors de l’appel de la fonction, les arguments utilisés doivent être fournis dans le même ordre que celui des paramètres correspondants.

La définition des fonctions doit précéder leur utilisation.

Les paramètres sans valeur par défaut doivent précéder les autres.

Si tous les paramètres ont une valeur par défaut, l'ordre n'importe pas, à condition de désigner les paramètres correspondants.

Espaces de noms

Les variables définies à l’extérieur d’une fonction sont des variables **globales**.

Leur contenu est accessible à l’intérieur d’une fonction, mais la fonction ne peut pas les modifier.

Les variables définies à l’intérieur d’une fonction sont des variables **locales** à la fonction.

Leur contenu n'est accessible qu’à la fonction elle-même, elles n’interfèrent pas avec les variables globales.

Chaque classe possède son propre espace de noms, les variables qui en font partie sont appelées variables (ou attributs) de classe.

Chaque objet instance (créé à partir d’une classe) obtient son propre espace de noms, les variables qui en font partie sont appelées variables (ou attributs) **d’instance**.

Les variables peuvent être utilisées mais pas modifiées entre différents espaces de noms.

Classes / Attributs / Méthodes

Vous connaissez déjà le principe de la programmation orientée objet (se référer au chapitre « Programmation »).

Une **classe** englobe les **attributs** et **méthodes** d'un objet. Chaque objet est créé par instanciation d'une classe et stocké dans une variable.

Les classes auxquelles on fait appel dans une instruction doivent toujours être accompagnées de parenthèses.

Une **méthode** est une fonction contenue dans le corps de la classe.

Un **attribut** est une variable d'instance prédéfinie par un **constructeur**.

Un constructeur est une méthode exécutée automatiquement lorsque l'on instancie un nouvel objet à partir de la classe.

Définir une classe :

```
class NomClasse:
        """documentation"""
        def __init__(self, parametre1, parametre2):
                self.parametre1 = parametre1
                self.parametre2 = parametre2
                NomClasse.attribut = valeur
        def nom_methode(self):
```

Instancier une classe :

```
instance = NomClasse(attribut1, attribut2)
```

La définition d'une méthode est toujours placée à l'intérieur de la définition d'une classe. Elle doit toujours comporter au moins un paramètre, lequel doit être une **référence d'instance** (*self*).

self désigne l'instance à laquelle la méthode sera associée (agit comme instance de l'objet).

__init__ définit les paramètres pour créer une instance de classe et initialise certaines de ses variables d'instance au moment même de l'instanciation de l'objet.

Une classe peut hériter d'une autre grâce à **l'héritage** :

```
class NouvelleClasse(ClasseParente):
```

Lors de la création de la classe, indiquer la classe **parente** entre parenthèses. Elle héritera de tous ses attributs et toutes ses méthodes et pourra y ajouter les siennes.

L'héritage ne concerne que les classes, et non les instances de ces classes.

Modules

Ce sont des fichiers qui regroupent des ensembles de fonctions, des définitions de variables ainsi que des classes.

L'importation des modules se place au début du code source (après la première ligne destinée à l'encodage).

Importer toutes les fonctions d'un module :

```
from module import *
```

Importer une seule fonction d'un module :

```
from module import fonction
```

Exceptions

L'exécution de votre code peut générer des erreurs. Vous pouvez les prévoir grâce aux **exceptions** :

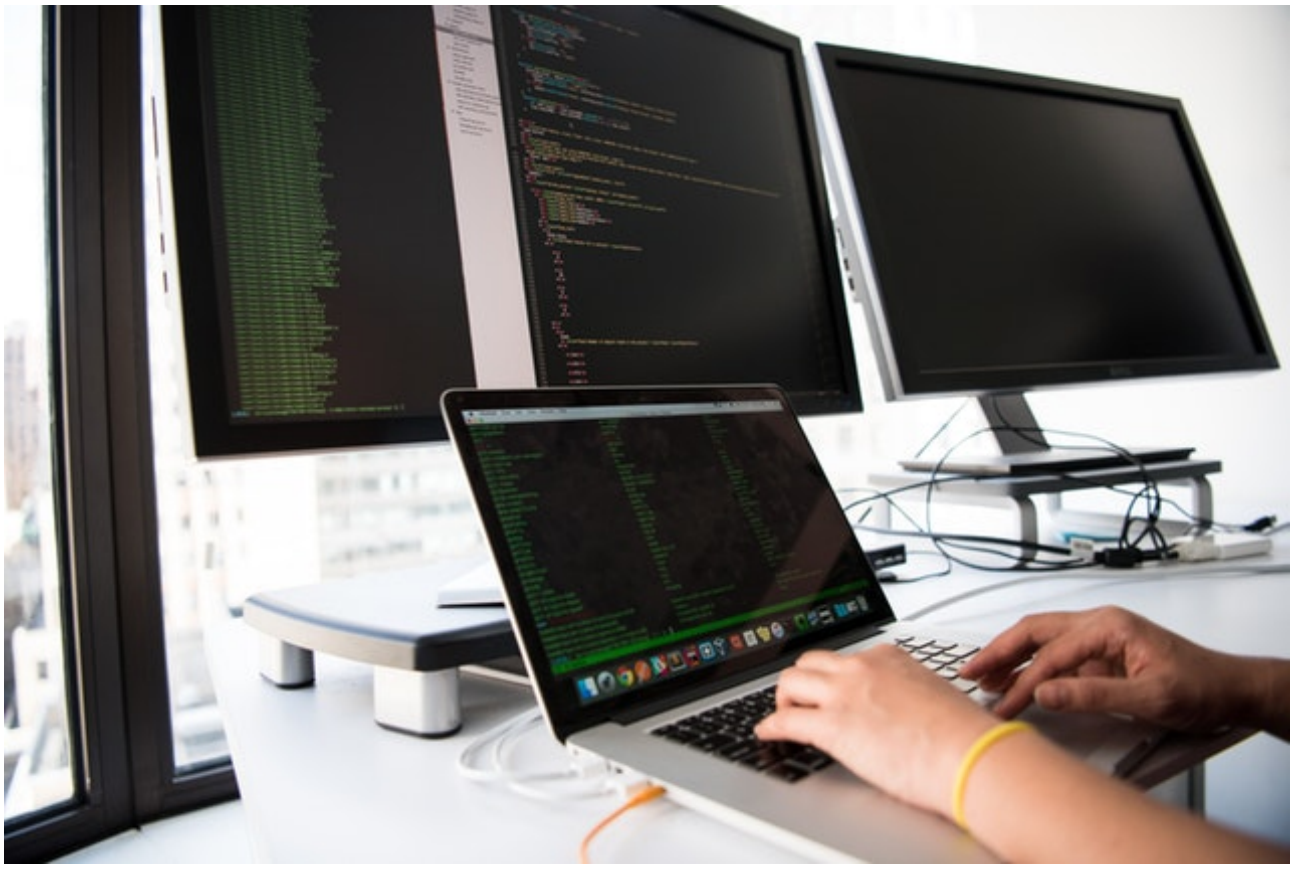
```
try:
    instruction à tester
    assert instruction (test d'une instruction, AssertionError si échec)
    raise TypeException (lever une exception)
except type_exception as variable_exception:
    instruction si exception
    pass (si rien ne doit se passer)
finally:
    instruction à exécuter dans tous les cas
```

Et maintenant ?

Il existe bien d'autres spécificités mais vous avez maintenant les cartes en main pour écrire vos propres programmes.

Google est votre ami si vous n'arrivez pas à comprendre un point précis. Faites une recherche par mot-clé et vous serez souvent surpris de voir une personne avec la même question que vous sur *Stack Overflow* (par exemple).

N'oubliez pas, c'est en faisant des erreurs que l'on apprend.



Conclusion

Merci d’avoir lu cet ebook. Malgré tout le sérieux investi dans l’écriture, il peut toutefois contenir des erreurs. Si tel est le cas, n’hésitez pas à le signaler à l’adresse :

contact@lxvsws.com

Pour permettre une compréhension plus facile et rapide, les informations présentées ne sont que partielles.

Pour approfondir votre apprentissage, faites des recherches plus ciblées grâce aux nouveaux concepts et termes que vous avez appris. L’informatique est un domaine vaste qui offre perpétuellement de nouvelles approches et découvertes.

L'auteur ne peut être tenu responsable de l'application ou non des informations présentées dans cet ebook. L’auteur ne peut être tenu responsable des erreurs et omissions, ni des conséquences liées à l’utilisation des informations présentées. Si des dommages sont causés par l'utilisation des informations présentées, l’auteur ne pourrait être tenu responsable.

Grâce à la vue d’ensemble apportée, vous êtes entrés dans la boîte noire et possédez maintenant les connaissances pour comprendre, et donc utiliser, un ordinateur.

Bonne continuation ! :)

Création & Distribution :

LXVSWS

www.lxvsws.com

2020