

Approche technique de l'espace numérique

Pablo Rauzy

pr@up8.edu

pablo.rauzy.name/teaching/aten



Institut Français de Géopolitique / M2 Cyberstratégie & Datascience

Séance 2

Quelques outils réseau

Introduction à la programmation (impérative)

Quelques outils réseau

- ▶ Utilisation : `ping host`.
- ▶ Envoyer des *echo requests* ICMP à un hôte.
- ▶ Typiquement pour voir si il est en ligne.
- ▶ Version IPv6 : `ping6`.

- ▶ Utilisation : `tracert host`.
- ▶ Déterminer le chemin parcouru par les paquets jusqu'à un hôte.
- ▶ Version IPv6 : `tracert6`.

- ▶ Utilisation : `nslookup name [server]`.
- ▶ Interroger les serveurs de noms.
- ▶ Par défaut on interroge le champs A, possibilité de spécifier un autre champs (AAAA, MX, TXT, ...) avec `-q=`.

- ▶ Utilisation : `whois name`.
- ▶ Interroger la base de données “whois” qui donne les informations de contact du propriétaire d’un nom de domaine.

- ▶ Utilisation : `netstat [options]`.
- ▶ Affiche les connexions réseau (et d'autres informations).
- ▶ Beaucoup d'options selon ce qu'on veut afficher :
 - sockets UNIX, TCP, UDP, raw ;
 - IPv4 / IPv6 ;
 - connexions sortantes, sockets passives ;
 - etc.

- ▶ top pour le trafic réseau.

- ▶ Utilisation nc [options] [host port].
- ▶ Connexions et écoutes arbitraires TCP et UDP.
- ▶ Véritable couteau suisse réseau.

- ▶ Utilisation : `openssl s_client host port`.
- ▶ Fonction client de netcat supportant TLS/SSL.

- ▶ Couteau suisse pour la connexion sécurisée :
 - session à distance,
 - création de tunnel,
 - redirection de port,
 - etc.
- ▶ Voir aussi :
 - `tsocks`.

- ▶ Capture et analyse du trafic réseau.

- ▶ Utilisation : `nmap [options] cibles`.
- ▶ Exploration réseau et scanneur de ports/sécurité.

- ▶ IP over DNS.
- ▶ Utile pour passer au travers des portails captifs.

Introduction à la programmation (impérative)

- ▶ Commençons par le commencement.

- ▶ Un *programme* est un ensemble d'opérations exécutables par un ordinateur.
- ▶ Quand on parle de programme, il peut s'agir de son *code source* ou d'un *binaire*.
- ▶ Le code source d'un programme peut être *interprété* ou *compilé*.

- ▶ L'interprétation d'un programme source est réalisée par un autre programme, qu'on appelle un *interpréteur*.
 - ▶ Pour chaque opération décrite dans le code source, l'interpréteur :
 - lit et analyse l'opération,
 - l'évalue ou l'exécute.
- + cycle de développement plus rapide, déboguage plus simple
- optimisation, vitesse d'exécution du programme

- ▶ La compilation d'un programme source est réalisée par un autre programme, qu'on appelle un *compilateur*.
 - ▶ Le compilateur traduit le programme *une fois pour toutes* vers un autre langage.
 - ▶ Généralement, vers un langage plus "proche" de l'ordinateur, pour que le code puisse être exécuté directement par le matériel.
- + optimisation, vitesse d'exécution du programme
- cycle de développement plus lent, déboguage plus complexe

- ▶ La compilation d'un programme source est réalisée par un autre programme, qu'on appelle un *compilateur*.
 - ▶ Le compilateur traduit le programme *une fois pour toutes* vers un autre langage.
 - ▶ Généralement, vers un langage plus "proche" de l'ordinateur, pour que le code puisse être exécuté directement par le matériel.
- + optimisation, vitesse d'exécution du programme
- cycle de développement plus lent, déboguage plus complexe
- ▶ Il existe des techniques intermédiaires (*bytecode, JIT*) qui tentent des compromis.

- ▶ Les codes sources sont écrits dans divers langages de programmation.
- ▶ Nous allons commencer par étudier des concepts génériques des langages impératifs.

- ▶ Dans un langage impératif, un programme est une suite d'*instructions*.
- ▶ Une instruction correspond à une “commande”, telle que :
 - la *déclaration* d'une *variable*,
 - l'*affectation* de la valeur d'une *expression* à une variable,
 - le test *conditionnel*,
 - l'*itération*.

- ▶ Une *variable* dans un programme a le même rôle qu'une variable en mathématique : elle symbolise (*nomme*) une valeur qui peut changer au cours du temps.
- ▶ Une variable a un *nom* et un *type*.
- ▶ Le type définit les valeurs possibles : un nombre, une chaîne de caractères, un booléen...
 - Dans certains langages le type est attaché à la variable et ne peut pas changer, on parle alors de langages *statiquement typés*.
 - Dans d'autres, le type est seulement attaché aux valeurs et donc le type d'une variable peut changer, on parle alors de langages *dynamiquement typés*.

- Une *expression* est une combinaison d'éléments du langage qui retourne une valeur quand elle est *évaluée* :
- `2 + 3`
 - `age > 20`
 - `taille * 100`
 - `prenom + " " + nom`

- ▶ Dans certains langages, on doit déclarer une variable avant de pouvoir l'utiliser :
 - **var** eleve_nom
 - **nombre** eleve_nom
 - **chaîne** eleve_nom
- ▶ Dans d'autres, il suffit d'affecter une valeur à la variable :
 - `eleve_nom ← "Sonia"`
 - `eleve_age ← 20`
- ! Attention, dans certains langages l'affectation se fait avec le symbole `=`, sans que celui-ci n'ait de rapport avec le `=` des mathématiques !

- ▶ Un programme doit pouvoir faire des choix dynamiquement, lors de son exécution.
- ▶ On utilise pour cela des *tests conditionnels*.
- ▶ Ils permettent de n'exécuter un *bloc* (sous partie) du programme que si une expression booléenne est vraie :
 - **si** `age ≥ 18` **alors**:
 `afficher("majeur")`
 - sinon**:
 `afficher("mineur")`

- ▶ Un programme doit pouvoir répéter certaines opérations plusieurs fois.
- ▶ On utilise pour cela des *boucles*.
- ▶ Elles permettent de d'exécuter un bloc du programme que tant qu'une expression booléenne est vraie :
 - $n \leftarrow 0$
tant que $n < 10$ **faire:**
 afficher_entier(n)
 $n \leftarrow n + 1$

- ▶ Une *fonction* est un “sous-programme”.
- ▶ Elles permettent de réutiliser plusieurs fois le même code à différents endroits.
- ▶ Une fonction reçoit un ou des *arguments* (ou paramètres) et renvoie un résultat.

- **fonction** calculer_age (annee_naissance):
 renvoyer annee_courante - annee_naissance

- **fonction** fact (n):
 resultat ← 1
 tant que n > 1 **faire**:
 resultat ← n * resultat
 n ← n - 1
 renvoyer resultat

! Attention, la notion de fonction en programmation n'est généralement pas équivalente à celle des mathématiques.

- ▶ Pour organiser les programmes, on utilise des *structures de données*.
- ▶ Une structure de données est un *type* qui regroupe plusieurs variables.
- ▶ Il y a différents types de structures de données.
- ▶ Selon les langages, certains types de structures de données existent nativement : les listes, les vecteurs, les dictionnaires...
- ▶ On peut aussi créer ses propres structures de données.
 - Par exemple, une structure représentant un élève comprendra son prénom, son nom, son numéro d'étudiant·e, son adresse email, son UFR, son année d'étude, ses options, ...
 - En créant ainsi un type "élève" on peut utiliser une seule variable de ce type là où on a besoin de toutes ces valeurs.