Bases de la programmation

été 2017/18

Dr Łukasz Starzak, MSc(Eng) PhD

Lodz University of Technology
Faculty of Electrical, Electronic, Computer and Control Engineering
Department of Microelectronics and Computer Science
ul. Wólczańska 221/223, bât. B18, bureau 51
http://neo.dmcs.p.lodz.pl/~starzak [pl, en]
http://neo.dmcs.p.lodz.pl/bdp [fr]

Objectif du cours

- Pourquoi programmer ?
  - L'ordinateur peux fournir la solution :
    - dans un temps plus court
    - avec plus de précision et vraisemblance
    - avec moins d'engagement de la part de l'humain
  - L'ordinateur est particulièrement utile en génie économique et gestion de même qu'en génie ingénieur
- Mais il faut savoir comment :
  - inventer ce que doit être fait et dans quel ordre
  - décrire les actions de façon compréhensible pour l'ordinateur
  - assurer que le résultat soit juste
  - trouver et éliminer des erreurs
  - améliorer son programme
  - coopérer avec d'autres codeurs et utilisateurs
- Programmation est une matière à apprendre

Plan du cours magistral

1. Notions de base de la programmation et de l'algorithmique
2. Éléments d'un langage de programmation ;
   introduction au langage Matlab et à l'environnement Octave
   a) Données et leurs types
   b) Affectation et opérateurs arithmétiques
   c) Obtention et affichage de données
3. Fonctions et portée de variables
4. Structures de contrôle
   a) Algorithmes simples du traitement de données
   b) Opérateurs de relation et logiques
   c) Alternatives et boucles
   d) Instructions de terminaison
5. Représentation et stockage de données
6. Problèmes avancés de l'algorithmique et de la programmation
7. Erreurs d'exécution et de programmation

Plan des travaux pratiques

- **1re partie : Programmation en bureautique**
  - Enseignant : Dr Małgorzata Napieralska
  - Durée : 15h = 5 semaines
  - Langage : HTML – développement de pages web
  - Langage : Visual Basic – automatisation des outils bureautique
- **2e partie : Algorithmique en commerce**
  - Enseignant : Dr Łukasz Starzak
  - Durée : 25h = 8⅓ semaines (2h dans les semaines 13 et 14)
  - Langage : Matlab – calcul numérique
- Travail personnel en plus des classes sera nécessaire
  - C'est prévu dans le programme du cours
  - C'est normal pour tous les cours en génie informatique
- Où travailler ?
  - On peut accéder aux laboratoires du Département
  - On va utiliser des environnements gratuits (à l'exception de Microsoft Office) qui peuvent alors être installés sur vos ordinateurs personnels

Évaluation

- Cours magistral
  - Interrogation écrite lors du dernier cours ou plus tard selon la décision du groupe
    - Plus tard est conseillé afin d'avoir la possibilité de pratiquer tous les sujets, ce qui permettra de les comprendre mieux
    - Voir la page web pour des repères (lorsque la date est fixée)
  - Deux interrogations de rattrapage
- Travaux pratiques – bureautique
  - Les règles seront communiquées par Dr Napieralska en TP
- Travaux pratiques – algorithmique
  - Travail pendant les cours
    - Montrez les résultats, sinon ils ne seront pas évalués
  - Compte-rendu après que chaque énoncé est accompli
    - Voir la page web pour des directives
- Note finale
  - $0{,}5 \cdot \mathrm{CM} + 0{,}5 \cdot (15/40 \cdot \mathrm{TP}_{\mathrm{bur}} + 25/40 \cdot \mathrm{TP}_{\mathrm{alg}})$

Présence aux cours

- Travaux pratiques
  - Présence obligatoire
  - Une absence sans excuse tolérable
- Conférence
  - Présence n'est pas obligatoire
  - Présence n'affecte pas votre note (directement)
- Mais
  - Je ne vais pas répéter en TP ce que j'aurai présenté et expliqué en conférence (ça ne veut pas dire qu'on ne peut pas me demander d'aide)
  - Ne pas venir en CM, c'est déclarer qu'on sait déjà ou bien librement choisir de ne pas savoir alors je ne vais rien expliquer aux personnes absentes à la conférence
  - C'est pourquoi :
    - la présence en CM est fortement conseillée
    - on fera la liste
    - il faut demander si on ne comprend pas

1. Notions de base de la programmation et de l'algorithmique

Programmation
Langages de programmation
Cycle de vie

Processus de la programmation

- Un **problème** est une question à résoudre par une solution informatique
- Une **instance** du problème est une entrée nécessaire pour calculer une solution du problème

- Un **programme** est une description d'actions qu'un ordinateur doit effectuer pour résoudre un problème donné
  - un ensemble d'actions
  - un ordre d'exécution
  - données [d'entrée] (entrées)
  - résultats (sorties)

... est le processus de développement (création) de programmes

Programmation : exemple

- Problème
  - Calculer le salaire d'un vendeur composé d'une partie fixe et d'une commission sur la valeur des ventes réalisées.
- Formule arithmétique
  - $s = f + 0{,}02 \cdot v$
    s – salaire
    f – partie fixe
    0,02 – taux de commission (2 %)
    v – montant des ventes
- Instance du problème
  - Calculer le salaire du vendeur dont la partie fixe est de 1100 € s'il a vendu des produits à 32 000 €.

- Programme (strictement : algorithme...)
  1. Obtenir les données d'entrée f et v
  2. Multiplier v par 0,02
  3. Additionner le résultat précédent à f
  4. Renvoyer le résultat s
- Dans la plupart des cas, c'est une réponse au problème et non pas à son instance !
- Le degré de généralisation sera différent selon le cas
  - Le taux de commission peut être un paramètre variant selon l'employé

Principes méthodologiques

1. Abstraire
   - Retarder le plus longtemps possible le développement du code destiné à l'ordinateur
2. Décomposer
   - Diviser le problème en parties plus facilement abordables
3. Combiner
   - Assembler les solutions partielles pour obtenir la solution globale

Spécification d'un problème

- Un **énoncé** est un texte définissant sans ambiguïté :
  - l'entrée (données d'entrée)
  - la sortie (résultats)
  - relations entre les données et les résultats
- Problèmes mal posés
  - Déterminer la position de l'élément le plus petit d'une suite.
  - Si l'entrée est { 2, 3, 1, 4, 1, 9 }, le résultat c'est... ?
  - Complément nécessaire : quel doit être le résultat quand il y a plusieurs éléments de la suite qui possèdent la valeur minimale
  - Il peut y avoir une solution univoque
  - Sinon il faut demander à l'utilisateur du programme
    - la position du premier d'entre ces éléments ?
    - la position du dernier d'entre ces éléments ?
    - une liste des postions de tous ces éléments ?
    - signaler une erreur ?

Précision sur la notion du programme : algorithme

- Un **algorithme** est une description d'un processus de résolution d'un problème défini
- Un **algorithme** est une succession d'actions qui, agissant sur un ensemble de données (entrée) fourniront la solution (sortie) à un problème défini
- Un **programme** est alors un algorithme destiné à l'ordinateur, écrit dans un **langage de programmation**
- La **codification** est le processus de conversion de l'algorithme en **code** (« texte » écrit dans un langage de programmation)
- Le programmeur ne doit pas laisser l'objectif final (le code) l'aveugler

Pseudo-code

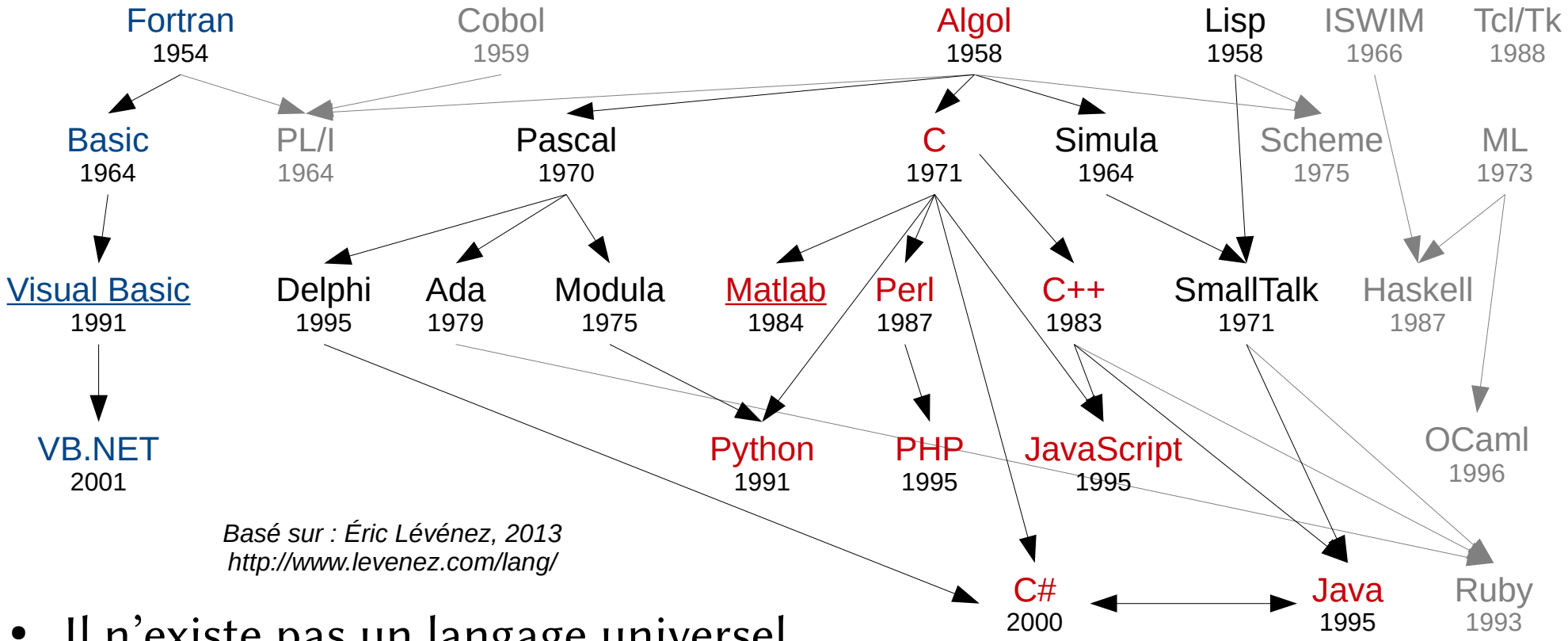
- Langage le plus proche possible du langage humain, néanmoins :
  - simplifié – on n'observe pas toutes les règles de grammaire
  - formalisé – pas de synonymes, interdit de changer de l'ordre de mots
- Sert à décrire des algorithmes, non pas des programmes
  - Peut être traduit en langage de programmation quelconque
  - Certains symboles pourront avoir une signification différente en pseudo-code et dans le langage utilisé
- Symboles généralement reconnus : + − * / = ≠ < > etc.
- Symbole de l'affectation ← :
  - on écrit : a ← 2
  - on lit : affecter la variable *a* avec la valeur 2
  - ça veut dire : faire que la valeur de la variable *a* soit 2
- Caractères à signification définie :
  - [ ] – élément d'un tableau (matrice, vecteur, chaîne de caractères...)
  - ( ) – arguments d'une fonction, procédure...
  - . – composant (champ, méthode...) d'un objet, enregistrement...

Langages de programmation

- Bas-niveau
  - **1ère génération** – codes machine
    2e génération – assembleurs
  - spécifiques pour chaque processeur
  - directement compréhensibles (codes machine) ou littéralement traduisibles (assembleurs) pour le processeur
  - alors directement exécutables par le processeur
  - très difficiles (plus faciles les 2G) à comprendre par l'humain
  - aujourd'hui rarement utilisés en génie informatique (plutôt en électronique)
  - mais toujours utilisés par les processeurs eux-mêmes
- Haut-niveau
  - plus proches du langage humain et du pseudo-code
  - alors plus abordables pour l'humain
  - incompréhensibles pour les processeurs
  - ne dépendent pas du processeur
  - il existe des traducteurs capables d'en produire un code machine
  - **3e génération** : plus formels et exigeants mais efficaces – exécution plus rapide et fiable

    4e génération : faciles et abordables même pour des non-professionnels – programmation plus rapide

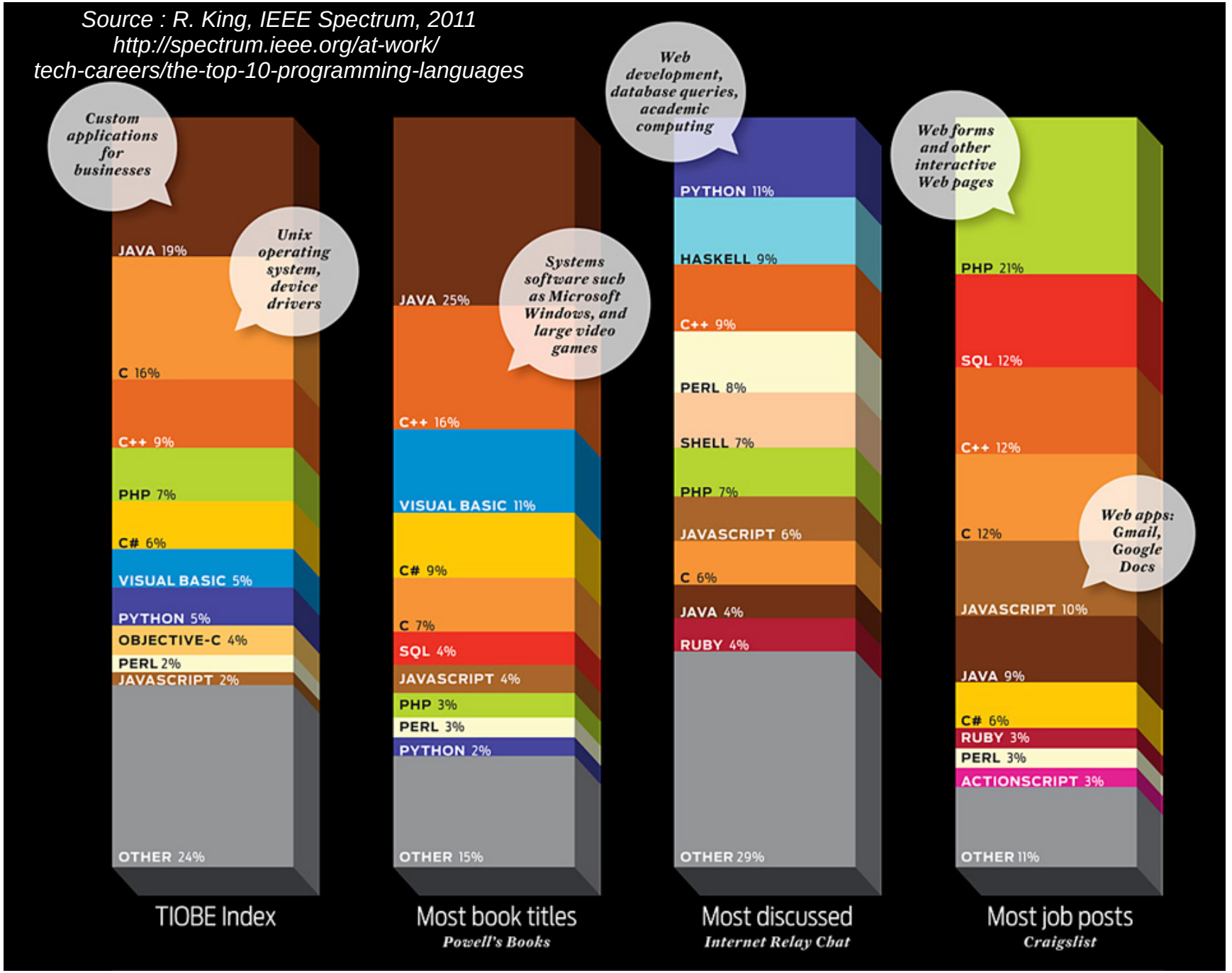
Généalogie simplifiée des langages haut-niveau 3G



Basé sur : Éric Lévénez, 2013
http://www.levenez.com/lang/

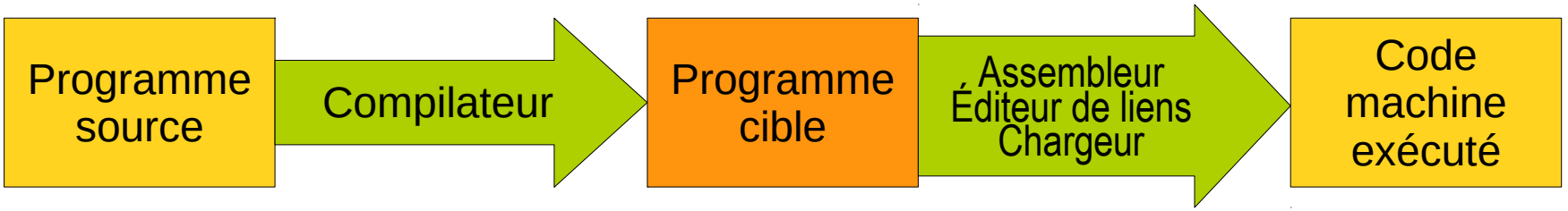
- Il n'existe pas un langage universel
  - Fortran : calcul numérique (mathématiques, contrôle)
  - Visual Basic : programmes simples « macros » p. ex. en bureautique
  - C, Python : programmes simples, fonctions de base, les systèmes Linux, MacOS
  - C++ : projets complexes, le système Windows
  - C#, Java : programmes modérément complexes avec interfaces graphiques
  - PHP, Java, JavaScript : programmation réseaux (internet, sites web...)
  - Matlab : créé strictement pour ce logiciel donc calcul numérique (4G)

Popularité des différents langages haut-niveau



Traduction et exécution de programmes

- Langages **compilés** (p. ex. Fortran, C)



- Langages **interprétés** (p. ex. Basic, Matlab)

  - Les commandes peuvent être entrées en ligne (mais ça dépend de l'environnement de programmation), au fur et à mesure de l'exécution
  - C'est plus simple du côté utilisateur
  - L'exécution paraît immédiate
  - En fait le code machine n'est pas optimal et l'interprétation peut être compliquée alors l'exécution prends plus de temps, de mémoire etc.