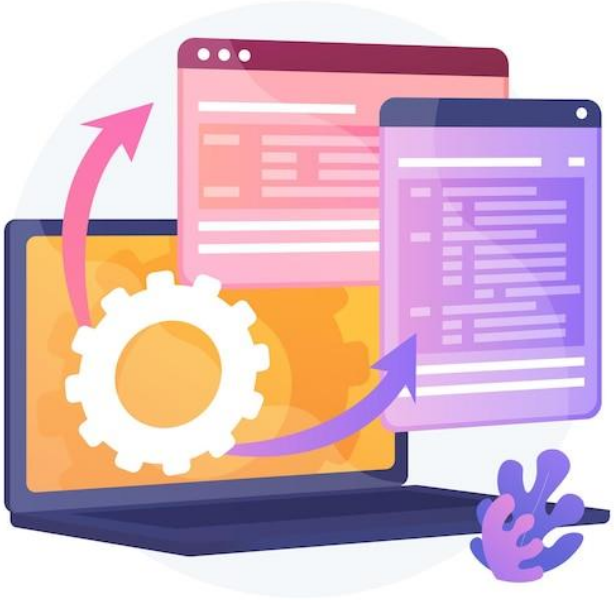


Syllabus d'informatique

YOANN AMAR ASSOULINE

Version : 1er mars 2023

Notice du document

Ce fascicule, rédigé par **Yoann AMAR ASSOULINE**, est un cours d'informatique organisé en fonction de différents sous-domaines comme la programmation ou l'infographie 3D. Pour chaque sous-domaine, on trouvera les sections suivantes :

- Différentes **parties de cours**, sous forme de fiches de synthèse.
- Des **exercices**, à effectuer en cours ou en devoir, pour le cours suivant. Chaque section d'exercice possède une notation en étoiles (⭐ ☆), correspondant aux niveau Licence 1 (bases fondamentales) jusqu'à Master 2 (niveau expert).
  - Des **fiches d'aide-mémoire** (nommées *Pocket Cards*) pour l'apprentissage de langages informatiques

Liens utiles

yoann.amar@outlook.com

pro@goldanniyatech.com

Lien Box pour la mise à jour du cours

Lien GitHub Yoann pour les *Pocket Cards*

Bases de l'informatique

Plan prévisionnel du cours

Maitrise des technologies et outils

- **Systèmes d'exploitation** : Windows

Maitrise des notions

- **Contexte** : historique et évolution dans la société
- **Architecture de l'ordinateur** : Architecture von Neumann
  - **Composants** : Processeur • Carte mère • Carte graphique • Mémoire vive • Disque dur •

Module 00 Histoire de l'informatique

Informatique est un mot-valise de « **information** » et « **automatique** », forgé par Philippe Dreyfus en 1962, et désigne l'automatisation du traitement de l'information par un système. Ce domaine est à la fois l'aboutissement de milliers d'années de progrès mais aussi une véritable révolution tel que la maîtrise du feu.

Un **ordinateur** est une machine qui peut acquérir, stocker et transformer de l'information pour la restituer sous une autre forme (les informations peuvent être de plusieurs types : textuelles, numériques, visuelles, auditives, etc.)

Histoire ancestrale

- Invention de **l'algorithme** par Abu Jaffar Al Khwarizmi (an 800)

Époque de systemes mecaniques

- Système de **calcul binaire** fortement développé par Gottfried Leibniz, mais inventé par Thomas Harriot et Juan Caramuel y Lobkowitz (Ares *et al.* 2018).

Systeme binaire

- **Bit** : un bit correspond à un 0 ou un 1. Correspond à un chiffre binaire, et contraction de l'anglais *binary digit*.
- **Byte** : composé de 6 à 9 bits, mais généralement de 8 bits, et peut représenter
- **Octet** : un byte de quantité standard, toujours composé de 8 bits.

Informatique classique

- **1938** : C. E. Shannon va définir le bit.
- **1945** : définition du modèle d'architecture von Neumann.
- **1947** : invention des transistors par John Bardeen et Walter Brattain.
- **1956** : premier ordinateur à transistors (TRADIC).
- **1959** : lancement du PDP-1, le premier ordinateur commercial qui est centré sur l'interaction avec l'utilisateur.

Informatique quantiques

Prochaine révolution, l'informatique quantique utilise la mécanique quantique pour résoudre des problème bien plus complexes que les ordinateurs classiques. On considère qu'il s'agira d'une évolution aussi importante que l'informatique classique depuis les années 1940, voire plus encore !

- Un **ordinateur quantique doit pouvoir résoudre un problème instantanément**, alors que celui-ci peut mettre des années, voire des milliers d'années pour être résolu sur un ordinateur classique.
- L'unité de base d'un ordinateur quantique est le ***qubits***, contrairement à l'ordinateur classique qui est le *bit*.
- Les **qubits sont extrêmement instables et peu fiables**, et a nécessité le développement d'une nouvelle branche de l'informatique quantique : la correction des erreurs quantiques.
- Les ordinateurs quantiques nécessitent des algorithmes quantiques qui exploitent au mieux la puissance des ordinateurs quantiques, mais qui sont adaptés aux processeurs quantiques.
- Il existe aujourd'hui des ordinateurs quantiques dans le cloud, disponibles pour tous :
  - **Azure Quantum**

- En 2019, le **processeur supraconducteur Sycamore de Google**, grâce à ses 54 qubits, a réussi à effectuer un calcul en 200 seconde qui prendrait 10 000 ans pour les plus grands super-ordinateurs actuels (de l'informatique classique).
- Les experts prévoient de premiers résultats significatifs de l'informatique quantique seulement après 2030.

Module 01 Architecture des ordinateurs

L'architecture des ordinateurs désigne la manière dont les différents sont assemblés et comment ceux-ci communiquent entre eux.

Architecture von Neumann

Modèle de conception d'ordinateurs, proposé par John von Neumann en 1945, et utilisé dans la majeure partie des ordinateurs modernes. Cette architecture segmente un ordinateur en quatre parties :

- **Processeur** (*Arithmetic Logic Unit*) : composant principal de l'ordinateur
- **Unité de contrôle** (Control Unit)
- **Mémoire** :
- **Input/ Output**

Architecture Harvard

Développé pour surpasser les problèmes rencontrés par l'architecture von Neumann.

Composants : descriptif primaire

- **Processeur** : aussi nommé CPU (Central Processing Unit), c'est l'unité qui traite toutes les instructions et données pour exécuter des programmes. Il est considéré comme le véritable cerveau de l'ordinateur.
- **Mémoire vive (RAM)**
- **Disques Durs**

Instructions

- **Code d'instruction** : groupe de bits (code binaire) qui va demander à l'ordinateur d'exécuter une séquence de micro-opérations. (divisé en deux parties : code d'opération et code d'adresse)

Module 02 Systeme d'exploitation Windows

Le système d'exploitation Windows est l'un des plus populaires au monde, utilisé dans environ 75% des ordinateurs de bureau en 2023 selon Statista.

1. Gestion de Projets informatiques

Plan prévisionnel du cours

Maitrise des technologies et outils

- **Communication** : Discord • Slack • Teams (MS)
- **Documentation/ *Management* de projets** : ClickUp • MS Office/ WPS Office • Notions
- ***IDE*** : IntelliJ IDEA • Visual Studio • Visual Studio Code
- **Langages** (théorie) : Niveaux (haut, bas) • Type (textuel, visuel) • Paradigmes • *Design Patterns*
- **Langages** (pratique) : C++ • C# • Java • Python
- **Moteurs 3D** : Unity • Unreal
- **Schématisation** : UML
- ***Versionning*** : Git • GitHub
- **Web** (optionnel) : HTML • CSS/ SASS • JavaScript & React • PHP & Symfony

Maitrise des methodologies de developpement

- **Modèles** (Référentiels : PMBOK® 7th Edition • SWEBOK)
  - *Software Development Life Cycle/* Cycle de vie : Waterfall • Iterative • Spiral • V-Model • Big Bang • Prototyping
  - AGILE : Kanban • Scrum…
  - DevOps
- **Performance :** Indicateurs de performance (KPI) • Évaluation de produits (MVP)…
- **Rédaction** : cahier des charges • cahier des spécifications techniques • doc technique • GDD (*Game Design Doc*)
- **Types de projets** : développement • intégration • déploiement • maintenance
- **Tests**
  - Formes de tests : automatisée • manuelle
  - Types de tests : fonctionnels • intégration • performance • unitaires

Évaluation et exercices

- **Exercices en classe** : exercice sur ordinateur via un *IDE*, un moteur 3D, un logiciel DCC, un logiciel de gestion de projets, MS Office/ WPS Office, etc.
- **Exercices de portfolio** : exercices à réaliser en solo, *from scratch*, et à déposer sur GitHub.
  - Création de projets complets avec documentation et outils de gestion de projets (cahier des charges, spécifications techniques, tâches à réaliser, planning, etc.).

1. 1. Contextualisation

Depuis des milliers d'annees

- Existence de la gestion de projet depuis (quasiment) toujours.
- Grands éléments architecturaux créées grâce à la **structuration** de projets : Pyramides d'Egypte, Sphinx, Grande Muraille de Chine, etc.
- Véritables débuts de gestion de projet lors de la **Renaissance** et l'arrivée des grands travaux de construction (dôme de Florence, etc.).

Depuis les annees 1950

- **Années 1950** : apparition des premiers langages de programmation (FORTRAN, LISP, etc.)
- **Années 1960** : Crise du logiciel et prise de conscience des difficultés de gestion d'un projet informatique.
  - Augmentation de la complexité et technicité d'un logiciel
  - Augmentation du nombre de lignes de code
  - **1962** : méthode WBS est présenté par la NASA via un article qui rédige un article sur la méthode
  - **Octobre 1968** : conférence de travail de l'OTAN sur les difficultés de production d'un logiciel (*Working Conference on Software Engineering*), popularisant les termes **Software Engineering** ou **Génie logiciel**
  - **1969 : naissance du PMI** (Project Management Institute)
- **Années 1980 :** apparition du **cycle en V** grâce à **Winston Royce** et importé du monde de l'industrie**.**
- **Années 1990 :** conception du processus SCRUM par Sutherland et Ken Schwaber au début des années 90.
  - **1995** : codification de SCRUM afin de le présenter à la conférence OOPSLA à Austin, au Texas (États-Unis). Publication de l'article "Processus de Développement Logiciel SCRUM".
- **Années 2000**
  - **2001** : création d'une nouvelle méthodologie de gestion de projet AGILE et rédaction du premier Manifeste AGILE. Opposition aux méthodes traditionnelles/ classiques.

1. 2. Definitions primitives

Definition generale

- **Création du génie logiciel** répondant à l'augmentation de la complexité et technicité des logiciels via des *Frameworks* de développement.
- Prendre en compte l'évolution des besoins et fonctionnalités des logiciels.
- Tenir compte de l'évolution constante des technologies et la diversification des architectures.
- Réduire le délai de réalisation des projets via des méthodes et outils efficaces.
- Diminuer substantiellement les coûts de développement en optimisant les ressources.
- Gérer l'augmentation des équipes de développement et des utilisateurs.

Methodologies

- Deux **grandes familles de méthodologies** de développement de logiciels informatiques :
  - **Séquentielles** (ou linéaires), représentée globalement par la méthode **Waterfall**
  - **Itératives** (ou continuelles), incarnée principalement par les méthodes **AGILE**

  ⚠ Attention, la méthodologie AGILE regroupe plusieurs formes, dont SCRUM. Mais SCRUM et AGILE ne sont pas des synonymes !

Transition d'une methode

Il est possible, en cours de projet, de faire la transition, par exemple d'une méthode **Waterfall** à une méthode **AGILE**, de manière graduelle.

Versioning

- **Statistiques** 2022 des Version Control (Stackoverflow) : https://survey.stackoverflow.co/2022/#technology-version-control
- **Git** est le seul système de version à utiliser. GitHub va abandonner le support pour Subversion (SVN) en janvier 2024 (source)

1. 3. Differentes formes de documentation

⚠ **Code** source d'un logiciel fait parfois office de documentation.

⚠ **Documentations** externes jamais à jour (coût de maintenance et mise à jour élevé)

Types de documentations

- **Cahier des charges :** permet de définir la finalité d'un projet et les étapes pour sa réalisation
- **Cahier des spécifications techniques :** documenter les méthodes, procédés et technologies à utiliser.
  - **Stack technique** : quels sont les outils que vous utilisez pour le projet. Justifier les choix des outils. Faire un comparatif/ étude du marché par rapport aux autres outils.
  - **Veille informationnelle**
  -
- **Documentation technique :** rédaction de documents destinés uniquement aux développeurs, internes ou externes.
- **GDD** : le *Game Design Document*, spécifique pour le développement de jeux vidéo
- **Manuel utilisateurs/ Notice :** document non technique, pour les utilisateurs finaux.

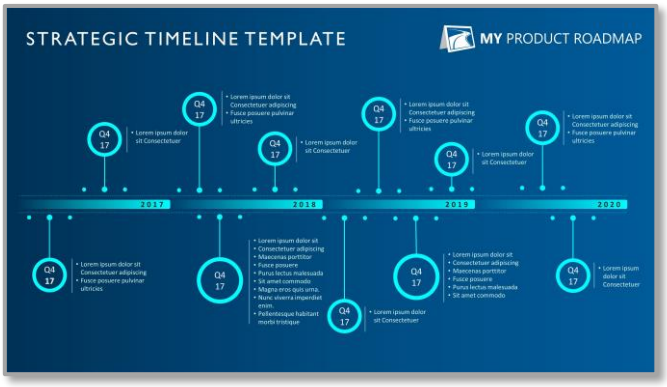
Utilite de documentation

- **Documentation technique automatique (interne ou externe)**
  - Documentation du code d'un logiciel, parfois générée automatiquement (cf. Python DocString). Explications parfois parcellaires et abstruses.
  - **Outils d'automatisation**
  - **Exemples : React** • Unreal Blueprints API • Unreal C++ API
- **Documentation de gestion de projet (interne)**
  - Documentation via différents outils (.) exclusivement destinés à un usage interne (gestion des bugs, compréhension du code, etc.).
  - **Exemples** : Asana • Clickup • Jira • Notions • Perforce
- **Manuels d'utilisation techniques (interne ou externe)**
  - Manuel ou Documentation publique, parfois presque sous forme de cours, avec des explications fournies et complètes (contrairement aux docs automatiques)
  - **Exemples :** Manuel Godot •Manuel Unity Enigne • Manuel Unreal

- **Veille informationnelle** : se renseigner sur l'évolution de la technologie et le marché actuel via différents outils :
  - **Réseaux sociaux** : Twitter • YouTube
  - **Sites de news** : Cnet • Developpez.com
  - **Flux RSS** : ancienne méthode

Roadmaps

- **Roadmap** : plan ou feuille de route permettant de détailler les étapes clés pour atteindre les objectifs d'un projet
  - **Utile pour prioriser les tâches**, maintenir un projet sur la bonne voie et estimer ressources et coûts pour chaque étape
  - **Présentation temporelle**, sous forme de diagramme (segmenté par étapes, mois, années, etc.)
  - **Exemples** (*Roadmap* publiques) **:** Autodesk 3DS Max • Unity



Preceptes d'une bonne documentation

- **Vue d'ensemble du projet**
- **Descriptif de la méthodologie utilisée**
- **Versioning** : chaque documentation doit correspondre à une version précise d'un logiciel (surtout pour un développement long terme).

1. 4. Methode AGILE

Frameworks

Il existe plusieurs Frameworks de la méthode agile :

- Scrum
- Safe
- Kanban
- Scrumban (alliage de Scrum et Kanban)
- Extreme Programming
- Adaptive Project Framework
- Nexus

1. 5. Évaluations

Dossier de Portfolio

- **Utilisation d'une plate-forme de gestion de projet** (au choix) et de rédaction d'une documentation (justification sur les choix des technologies, étude du marché, contexte, évaluation des risques, spécifications techniques, etc.). Dépôt d'un export du dossier de projet.
  - **ClickUp** : export des données via Excel pour uploader sur GitHub
  - **MS Office/ WPS Office** : Upload des fichiers source sur Github
  - **Notion** : Export d'un PDF de l'ensemble de la documentation
- **Création d'un projet complet en programmation** (au choix)
  - **Site internet**
  - **Prototype de logiciel 3D**
    - Concept au choix : jeu d'aventure, jeu de combat, jeu de course, ancien jeu d'arcade (Pacman, Galaga, etc.)…
    - Technologies (programmation) : Unity (C#) • Unreal Engine (BP, C++)
    - Technologie (infographie 3D) : Autodesk 3DS Max • Autodesk Maya • Blender
  - Logiciel classique (C# et .NET Core, etc.)

Examen Blanc

- Définir les différentes méthodologies de gestion de projets
- Définir en détail les idéologies des méthodes AGILE
  - Définir au moins deux Frameworks de la méthodologie AGILE

2. INFOGRAPHIE 3D

Plan prévisionnel du cours

MAITRISE DES TECHNOLOGIES ET OUTILS

- **Logiciels 2D** : Adobe Photoshop • Adobe Substance3D (optionnel) • Blender (optionnel)
- **Logiciels 3D** : Autodesk 3DS Max • Autodesk Maya (optionnel) • Blender (optionnel)
- **Moteur 3D** : Unity (optionnel) • Unreal (optionnel)

MAITRISE DE LA CREATION 3D

- **Logiciels 3D** (3DS Max • Maya • Blender)
  - Modélisation 3D : *polygone Modelling*
  - Matériaux
  - Lights
  - UV Unwarp
  - Animations : *skinning/ rigging • keyframes • bone-based • blend shapes/ morph targets*
  - Render : cameras • renderers • settings
- **Logiciels 2D**
  - Texture : traditionnelle (*Blinn*, *Phong*, *Lambert*) • Physique (*PBR*)
- **Moteur 3D**
  - Optimisation : optimisation des modèles pour la 3D temps réel (Unity, Unreal)

ÉVALUATION ET EXERCICES

- **Exercices en classe** : modélisation de bâtiments et de mobilier urbain.
  - Bâtiments : immeubles haussmanniens
  - Mobilier d'intérieur : armoires • canapés • chaises • lits • tables
  - Mobilier urbain : réverbère
  - Personnages : personnage stylisé • robot
  - Véhicule : voiture
- **Exercices de Portfolio** : création d'un portfolio complet
  - Publication du portfolio (au choix) : Site personnel/ pro • ArtStation • DeviantArt
  - ⚠ Publication des modèles 3D créés via 3DS Max ou Maya autorisée seulement si l'étudiant possède une licence indépendante (Indie) ou pro. Sinon, utiliser Blender.

2. 1. Bases de l'infographie 3D

L'infographie 3D est la création d'un **modèle 3D** via un logiciel, qu'on appelle communément un outil DCC (Digital Content Creation).

Modeles 3D

En simplifiant à l'extrême, on peut considérer qu'un modèle 3D (qui peut être un personnage, un véhicule, un bâtiment…) est composé de trois choses : le **maillage** (*the Mesh*), la texture et, parfois, des éléments pour les animer (l'ossature, les *Blend Shapes*, etc.).

Modéliser (créer) et/ ou animer n'importe quel modèle 3D peut se faire avec le logiciel gratuit et Open Source **Blender**, qui peut être installé sur n'importe quel ordinateur (pas nécessairement puissant). Il est quasiment aussi efficace que les logiciels utilisés dans le milieu professionnel, comme Autodesk 3DS Max, Autodesk Maya ou Maxon ZBrush. Il existe de nombreuses chaînes de tutoriaux pour Blender, principalement en anglais (Blender Guru, CBaileyFilm, CG Geek, Gleb Alexandrov, Grant Abbitt…).

Les textures sont à réaliser soit avec certains outils de Blender, très rudimentaire, soit avec le logiciel gratuit **Quixel Mixer** (pour des textures modernes utilisées dans les jeux PC et consoles, qu'on nomme « PBR ») ou **Krita** (pour des textures de jeux mobiles simples). Dans le milieu pro, on utilisera surtout Adobe Substance3D pour les textures modernes et Adobe Photoshop pour les textures des jeux mobile (non-PBR, qu'on peut aussi appeler *Blinn-Phong* si on simplifie).

⇨ **Blender** : https://www.blender.org/
⇨ **Epic Quixel Mixer** : https://quixel.com/mixer
⇨ **Krita :** https://krita.org/en/

Developpement de logiciels interactifs / jeux video

Pour créer un logiciel 3D interactif en temps réel, en 3D ou en 2D (ou un outil de visualisation, dans ce cas précis), on doit généralement utiliser un moteur de jeu, composé de plusieurs modules différents (graphique, physique, audio, etc.). Il faut ensuite importer ces mêmes modèles dans un moteur (on parle alors d'importer des *assets*) et les utiliser pour créer les différentes interactions, autrement dit le *gameplay* (conduire une voiture, contrôler un personnage, etc.) ; Généralement, il existe deux moteurs extrêmement populaires, aussi bien chez les débutants/ amateurs que chez les professionnels : Unreal et Unity.

Unreal Engine est principalement conçu pour développer des jeux AAA sur PC et consoles, aussi bien récents que ceux qui seront publiés prochainement (Fortnite, Gotham Knights, Tekken 8, Hogwarts Legacy, The Witcher 4, Tomb Raider Next, etc.). Il s'utilise via deux langages de programmation : le C++ et le langage visuel *Blueprints*.

La seconde option, **Unity**, s'utilise plutôt pour créer des jeux mobiles et/ ou PC d'entrée de gamme (*Cuphead*, *Hollow Knight*, *Fall Guys*, *Hearthstone*...). Celui-ci s'utilise avec le langage C# (à prononcer C Sharp) et est tout aussi pertinent pour des jeux en 3D que pour des jeux en 2D (chose plus difficile à réaliser avec Unreal).

Généralement, on dit que Unity s'adresse aux débutants et à ceux qui souhaitent créer des jeux facilement, alors que le moteur Unreal est plutôt destiné à un développeur (ou une équipe) ayant beaucoup plus de connaissances et un niveau avancé. Le langage C++ d' Unreal est aussi beaucoup plus complexe à comprendre que le C# de Unity, car il est bien plus technique (même si, pour Unreal, on peut aussi utiliser le langage visuel *Blueprints*, relativement simple dans son approche). Dans tous les cas, les deux moteurs sont des valeurs sûres, et sur le marché du travail, ils sont extrêmement présents (aussi bien pour des stages que pour des emploi CDD/ CDI voire lorsqu'on ouvre sa propre entreprise). On précise enfin qu'il est possible d'utiliser gratuitement les deux moteurs (il faut ensuite reverser de l'argent lorsqu'on publie son jeu, selon les modalités de chaque moteur, mais au départ, il n'y a rien à verser, surtout pour Unreal).

⇨ **Epic Unreal Engine 5** : https://www.unrealengine.com/
⇨ **Unity** : https://unity.com/

2. 2. Autodesk 3DS Max

2. 3. Adobe Photoshop

2. 4. Exercices en classe (modeles 3D)

Exercice A Prise en main des logiciels

Exercice B Mobilier d'interieur

Modélisation et textures de mobilier d'intérieur.

2. 5. Évaluation du Portfolio (Projet 3D)

- **Réalisation d'une scène intégrale**
  - **Rendu interactif de la scène** : (au choix) Godot • Unreal • Unity
- Modélisation des éléments suivants, dans une seule et même scène
  - Au moins **1 bâtiment**
  - Au moins un environnement intérieur (appartement, maison, etc.)

Parcours Programmation Generale (Python)

Programme

Plan prévisionnel du cours

Maitrise des technologies et outils

- ***IDE*** : IntelliJ IDEA • Visual Studio • Visual Studio Code
- **Langages** (théorie) : Niveaux (haut, bas) • Type (textuel, visuel) • Paradigmes • *Design Patterns*
- **Langages** (pratique) : C++ • C# • Java • Python

Maitrise generique des langages

Programme adaptable en fonction des caractéristiques des langages étudiés

- **Introduction** : Compilation/ interprétation (/ assemblage) et commentaire
- **Données** : types de données et variables (entiers, flottants, booléens, etc.)
- **Structure de contrôle** : *if*, *while*, etc.
- **Fonctions et procédures**
- **Paradigme orienté objet**
  - Bases : classes, objets, méthodes, propriétés, etc.
  - Concepts : héritage, polymorphisme, encapsulation, etc.
- **Optimisation de code**

Évaluation et exercices

- **Exercices en classe** : exercice sur ordinateur via un *IDE*
- **Exercices de Portfolio :** réalisation de projets en solo et *from scratch*, à déposer dans un dossier (*repository*) GitHub.

Stack technique

- Téléchargement de la dernière version de **Python** : https://www.python.org/downloads/
- Téléchargement de l'IDE **Visual Studio Code** : https://code.visualstudio.com/
  - Version Web de VS Code (https://vscode.dev/) a un support expérimental pour Python
- Installation des **extensions sur VS Code** : Python • Pylance • vscode-icons

Module 00 Introduction generale

Omniprésente dans notre quotidien et sur une échelle planétaire, la programmation informatique est une compétence essentielle pour aujourd'hui, et joue un rôle central dans la plupart des industries.

En substance, la **programmation** consiste à écrire des instructions sur un ordinateur, grâce à un **langage de programmation**, afin d'exécuter des tâches spécifiques. Grâce à la programmation, on peut créer des sites web, des jeux vidéo, des systèmes d'exploitation (Windows, Mac OS) ou un moteur de recherche comme Google. La plupart du temps, cela se fait via un **IDE** (*Integrated Development Environment*) comme Visual Studio ou IntelliJ IDEA.

Langue naturelle et langage de programmation

On différencie les **langue naturelle humaine** (français, anglais, espagnol…) et **langage de programmation informatique** (Python, C++, C#…) :

- Un **langage de programmation,** pour un programmeur expérimenté, peut s'apprendre en quelques semaines, en particulier s'il utilise les mêmes concepts ou le même paradigme. Généralement, c'est plutôt les outils autour du langage (Frameworks, API, etc.) qui sont chronophages ou qui prennent du temps pour l'apprentissage.
- Une **langue naturelle humaine** nécessite plusieurs années d'apprentissage pour en maîtriser les bases, surtout si celle-ci est dans une famille éloignée de notre ou nos langue(s) natale(s). Par exemple, le Japonais est l'une des langues les plus difficiles à apprendre pour un locuteur.

Traduction en binaire

Il existe principalement 3 moyens pour **traduire un script de programmation en langage binaire** (un ordinateur ne peut pas comprendre les langages de programmation) :

- **Assemblage** : traduction très proche entre le binaire et l'assembleur, très rarement utilisé aujourd'hui).
- **Compilation** : traduction en amont d'un ensemble de scripts pour créer un exécutable, un fichier qui sera lu par le système d'exploitation, utilisé par le C++ ou le C#
- **Interprétation** : traduction d'un langage pas à pas et en direct, utilisé par Python ou Java
  - On différencie parfois **l'interprétation pure** et les langages qui passent par une machine virtuelle (**interprétation hybride**)

Langage de programmation visuel et textuel

On différencie deux moyens de rédiger un langage de programmation : les **langages textuels** et les **langages visuels (Blueprints, Unity Visual Script…)**

- **Les langages de programmation textuels** sont rédigés via des lignes de code, de manière linéaire, et de gauche à droite. La totalité des mots clés (keywords) proviennent de l'anglais, et presque la totalité des langages informatiques sont en anglais.
  - C, C++, C#, Java, Python…
- **Les langages visuels** sont des langages utilisant un espace en 2D avec, la plupart du temps, des boîtes et des connecteurs (fils) afin de lier des séquences logiques.
  - Langage ***Blueprints*** du moteur 3D *Unreal Engine*.
  - Langage éducatif Scratch.

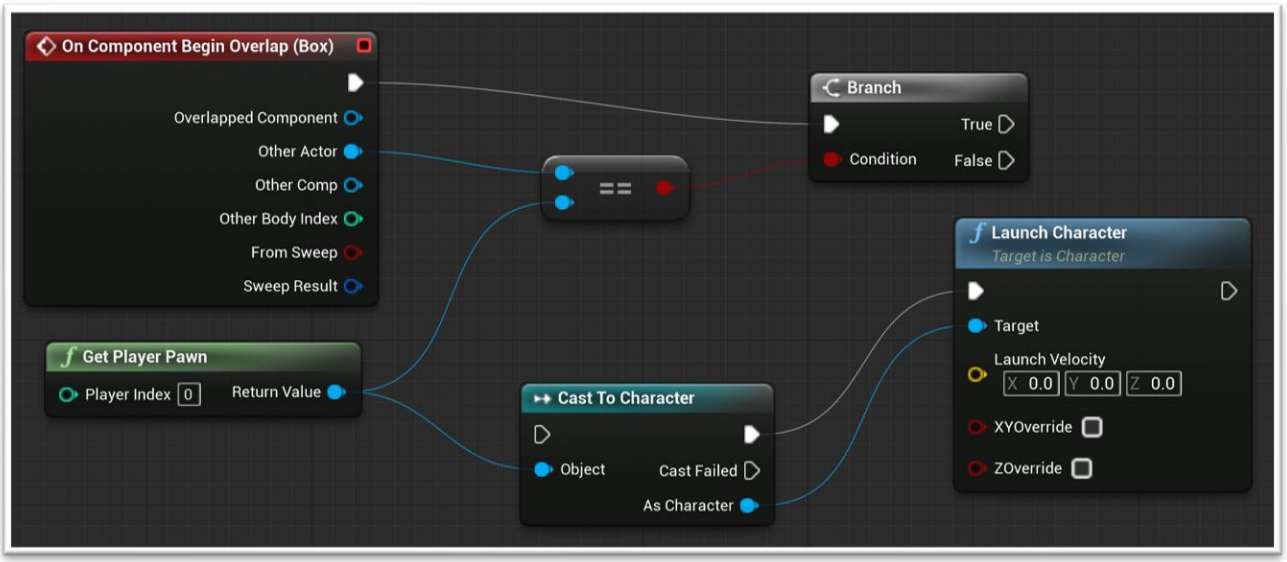


image 1 Langage visuel Blueprints du moteur Unreal Engine (UE4.27 Doc)

COMPLEXITE DES LANGAGES INFORMATIQUES

Généralement, on dit qu'il existe deux **niveaux de complexité** des langages informatiques : (haut niveau et bas niveau), mais il s'agit surtout d'une notion malléable et évolutive. Elle est conditionnée aussi par le contexet et les différents langages disponibles. Par exemple, le C++ était considéré comme un langage de haut niveau au début des années 2000, notamment par rapport à l'assembleur ou au C. Alors qu'aujourd'hui, il est plutôt considéré comme un langage bas niveau, par rapport à Python ou Java.

- **Langages de bas niveau** : très techniques et proches de la machine (binaire, assembleur).
  - **Lié au matériel** : pour écrire du code en assembleur, par exemple, il est indispensable de connaître l'architecture du processeur sur lequel le code sera exécuté, car il s'appuie sur les registres du processeur

- **Langages de haut niveau** : proches de l'anglais (Python) et plus simples à maîtriser.
  - **Indépendant du matériel** : généralement, un langage de haut niveau peut fonctionner sans faire des fonctions spécifiques pour la machine.

⚠ Attention, cette notion est toutefois malléable et évolutive. Auparavant, le C++ était considéré comme un langage de haut niveau, c'est-à-dire plus proche des langues humaines, au début des années 2000. Il s'opposait aux langages de bas niveau comme le C voire les différents langages assembleurs. Aujourd'hui, le C++ est décrit comme un langage bas niveau par rapport à JavaScript ou Python.

Multiplicite des langages dans un seul projet

Dans le milieu professionnel ou pour les projets de grande envergure, on constate qu'il y a souvent **plusieurs langages de programmation qui sont utilisés**. On peut illustrer cela par le jeu vidéo *Fortnite*, toujours en cours de d'évolution et conçu sur le moteur Unreal Engine, qui utilise et à utilisé au moins les langages suivants :

Image 2 Interface du moteur Unreal Engine, ici pour le développement du jeu Fortnite (Parrish 2018)

- **Blueprints :** un langage visuel afin d'écrire des formes simplifiées d'interactions (ou de *gameplay*)
- **C++** : un langage pour écrire des systèmes complexes d'interactions.
- **HLSL** : un langage de *shader*
- **UnrealScript** : un langage qui est l'ancêtre du Blueprints et qui était hypothétiquement utilisé dans la verison 3 du moteur Unreal Engine, lors du début du développement de *Fortnite*.

Facteurs decisionnels

Le **choix du langage** n'est pas toujours un critère pertinent lors de la phase de choix des technologies. Parfois, la décision repose plutôt sur le choix des *Frameworks*/ *Libraries*/ *Game Engines*, qui vont conditionner le langage. Par exemple, si un développeur souhaite réaliser un jeu mobile en 3D, alors il va choisir le moteur Unity, le plus populaire et le plus efficace, et va être obligé d'utiliser le langage C#. En revanche, s'il préfère créer un jeu sur console et exploiter la puissance maximale, alors son choix va se porter sur Unreal Engine, et obligatoirement sur le langage C++ (ainsi que les Blueprints).

Classement mondiaux

Il existe différents **classement mondiaux** de popularité annuelle des langages de programmation. Bien que peu objectifs, ces classements constituent *de facto* les seuls moyens de pouvoir mesurer la popularité d'un langage.

Classement	1	2	3	4	5
Githut 2.0	Python	Java	JavaScript	C++	Go
IEEE Spectrum	Python	C	C++	C#	Java
PYPL	Python	Java	JavaScript	C#	C/ C++
RedMonk rankings	JavaScript	Python	Java	PHP	C#
TIOBE index	Python	C	C++	Java	C#

Paradigmes de programmation

Les **paradigmes de programmation** sont des philosophies ou des manières d'écrire du code informatique (on pourrait aussi évoquer l'idée d'un style de programmation). Chaque paradigme possède des caractéristiques qui peuvent convenir à certains contextes, même si on considère aujourd'hui que le **paradigme orienté objet** est, de loin, le plus utilisé. Il existe plusieurs classifications, mais on peut distinguer deux grands groupes :

- La **programmation impérative** : le tout premier paradigme qui a existé. L'architecture des ordinateurs est bien de nature impérative. Celui-ci va définir chaque étape à exécuter pour l'ordinateur et le déroulement linéaire du code. Elle est nécessaire pour avoir un contrôle précis et détaillé.
  - **Programmation Procédurale** : elle va regrouper différentes instructions en procédures.
  - **Programmation orienté objet** : paradigme le plus utilisé, il développe des notions telles que la classe, l'héritage, etc.
- La **programmation déclarative** : plutôt centrée sur le résultat, et va décrire la logique du programme.
  - **Programmation logique**
  - **Programmation fonctionnelle** : écriture du code uniquement sous forme de fonctions, et basé sur la théorie mathématique du Lambda-calcul élaborée par Alonzo Church en 1930.

Paradigme orienté objets

Le paradigme orienté objet est le paradigme le plus utilisé actuellement. Il existe généralement **deux types de langages orientés objet** :

- Ceux qui utilisent des **classes** (C++, C#, Java…),

- Ceux qui utilisent des **prototypes** (JavaScript)**.**

Nous avons également **4 principes fondamentaux** du paradigme orienté objet :

- **Encapsulation** (encapsulation)
- **Abstraction** (abstraction)
- **Héritage** (*inheritance*)
- **Polymorphisme** (*polymorphism*)

PARADIGME FONCTIONNEL

Un programme écrit via le paradigme fonctionnel va être décomposé en un ensemble de fonctions. On observe surtout deux natures de fonctions :

- **Fonctions pures** : des fonctions qui produisent des résultats uniquement en fonction des arguments, et non pas en fonction de variables externes.
- **Fonctions impures**

CONCEPTION ET ARCHITECTURE DES LOGICIELS

Lors de la conception d'un logiciel, on distingue plusieurs niveaux de conception, qui interviennent également à différentes étapes :

- Les **architectures de logiciels** : élaboration des règles et conventions pour créer une application ou un logiciel (exemple : *Entity*, MVC, MVP…)
- Les **Design Patterns** : des moyens de concevoir, de manipuler ou de faire communiquer des objets (exemple : *Singleton*, *Factory*, etc.)
- Les **algorithmes** qui sont des solutions plus précises (pas à pas) pour résoudre un problème particulier.

Conseils professionnalisants

- **Créer un portfolio** (GitHub) **et le maintenir constamment à jour** permet d'améliorer son CV, en particulier pour ceux souhaitant être et rester sous le statut salarié. Privilégier les projets utiles, ou d'apparence utile (savoir se vendre et vendre ses propres projets).

- **Apprendre la dactylographie** (dans ce contexte, pouvoir taper à 10 doigts sur le clavier sans le regarder) est un avantage substantiel pour augmenter sa rapidité. Attention à apprendre sur le bon format de clavier (format francophone AZERTY ou anglophone QUERTY) !
  - **Site de dactylographie** : https://www.keybr.com/ • https://www.typing.com/

- **L'apprentissage de l'anglais** est obligatoire pour être un excellent développeur/ programmeur/ chef de projet.
  - **Applications utiles** : Drops (Application) • Duolingo

- **Les théories des intelligences multiples** peuvent être utilisées dans le cadre de la programmation informatique : cette activité mobilise à la fois l'intelligence linguistique et l'intelligence logico-mathématiques.
  - **L'apprentissage de langues** améliore les compétences en programmation
  - **Des exercices en mathématiques** via des langages de programmation (Python) permettent de développer l'intelligence logico-mathématique.

- Il existe des **sites interactifs soit pour apprendre différents langages de programmation** (avec un interpréteur en ligne), soit pour participer à des *challenges* en tout genre (Codewars, etc.). Ils peuvent être intéressants mais ne remplaceront jamais des projets réels. Perso, je conseille de travailler uniquement sur des projets de portfolio.
  - **Applications d'apprentissage** : Mimo
  - **Apprentissage en ligne :** Codecademy • Grasshopper • Sololearn
  - **Challenges** : Codewars • Leetcode

- **Lire le code source de logiciels** Open Source (ou Source-Available, qui signifie que le code source est disponible mais n'est pas librement modifiable) peut permettre de mieux appréhender un langage.

Module 01 Langage Python

Le **langage Python**, créé par Guido van ROSSUM en 1989 et publié en 1991, est aujourd'hui un des langages les plus populaires du monde.

Caracteristiques de Python

- **Langage de haut niveau :** très proche de l'anglais, il est conçu pour être facile à utiliser dans n'importe quel contexte.
- **Langage portable :** Python est par nature multiplateforme et peut donc fonctionner sur un grand nombre de systèmes d'exploitation (Mac, Windows, Linux, Android, iOS, etc.) et, théoriquement, sur n'importe quelle machine équipé de l'interpréteur Python.
- **Langage Interprété :** le langage est lu par la machine à chaque fois, contrairement au **langages compilés** (C++, C#...).
- **Multi-paradigme :** différents styles de programmation (déclarative, orientée objet, déclarative, etc.) peuvent être utilisés.
- **Bibliothèque Python Standard Library** : Python dispose d'une bibliothèque qui est disponible par défaut.
- Dans le milieu professionnel, **Microsoft Visual Studio Code** est l'IDE le plus utilisé avec Python. C'est le seul IDE qui sera utilisé pendant le cours. Néanmoins, d'autres **IDE** sont également assez populaires, notamment **PyCharm** ou **Sublime Text**.

Version	Date	Commentaires
Python 0.9.0	Février 1991	Première version publique
Python 1.0	Janvier 1994	
Python 2.0	Octobre 2000	
Python 3.0	Décembre 2008	
~~Python 2.7.17~~	Janvier 2020	Fin officielle du support de Python 2
Python 3.11.2	Février 2023	

Commentaires

Il existe 2 moyens de commenter en python :

- **Croisillon** : les commentaire via la touche croisillon (#) peuvent être en une seule ligne ou plusieurs lignes (il faudra alors refaire un croisillon à chaque nouvelle ligne)

- **Documentation Strings**

Variables basiques

- **Variable nulle : None**
- **Variables numériques** : **Int** (34) • **Float** (32.46) • **Complex** (1 + 3j)
  - **Opérations arithmétiques (addition, soustraction, etc.).**
  - **Règle des priorités PEMDAS** : Parentheses, Exponents, Multiplication & Division (de gauche à droite), Addition & Subtraction (de gauche à droite).
- **Variables booléennes** : True / False
- **Chaînes de caractères** : String ('Player Two')
- **Variable Byte et Bytearray**

Variables Strings : Operations

- **Slice**

Collection de variables

- **Dictionnaires :** collection non ordonnée d'objets.
- **Listes :** collection d'objets mutables.
- **Sets :** collection non ordonnée d'objets immutables
- ***Tuplets* :** collection ordonnée d'objets immutable (impossible à modifier)

Fonction

- **Bloc de code** qui est interprété uniquement lorsqu'il est appelé
- Une fonction peut avoir des **arguments**, optionnels ou non. Elle peut également **retourner une valeur**.
- **Deux types de fonctions**
  - Fonctions de la **Python Standard Library**, disponible dès l'installation de Python.
  - **Fonctions définies par l'utilisateur**, qu'on doit créer et qu'on peut ensuite utiliser.

Un langage Multi-paradigmes

Le paradigme orienté objet est le paradigme le plus utilisé.

Classes

- Une classe peut être abstraite (non utilisée) et concrète.

Exercice Python 01 Mini-jeu textuel

Cet exercice mobilise l'ensemble des éléments de base de Python pour réaliser un mini-jeu textuel, uniquement jouable via le clavier, et qui devra ensuite être déposé sur GitHub.

Le projet doit être organisé dans un dossier (portant le nom du mini-jeu), avec un fichier **__main__.py** qui doit gérer la totalité du jeu.

Consignes

- Créer de **nombreuses variables** (booléens, entiers, flottants, etc.) pour enregistrer différentes informations, comme le nom des personnages, le nombre d'ennemis, les points de vie ou de magie, etc.
  - Certaines variables doivent être entrées par l'utilisateur
- Créer des **fonctions** pour les différentes actions possibles du personnage jouable (aller quelque part, interagir avec un objet, etc.)
- Créer des **conditions** (if, elif, else) qui vont déterminer certains choix du joueur.
- Créer des **boucles** (**for** et **while** ainsi que des *Nested Loops*)
- Créer plusieurs **classe** pour les personnages, les véhicules ou différents objets (fruits, etc.)
  - Créer des classes abstraites et concrètes
  - Créer plusieurs propriétés et méthodes. Certaines méthodes doivent avoir des arguments
  - Créer plusieurs (au moins 5) objets de classes (ou instanciations)
- Créer des **conditions booléennes** pour gagner ou perdre une partie.

Exercice Python 02 Utilitaire Python

Les modules de la **Python Standard Library** se révèlent être extrêmement utile dans de nombreux cas. L'objectif est de créer un **utilitaire** graphique (via tkinter et CustomTkinter) qui va utiliser de nombreux modules de la PSL.

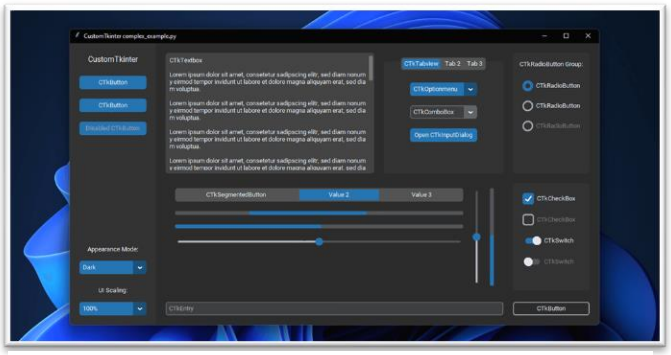


image 3 Interface CustomTkinter

CONSIGNES

- Dès le démarrage de l'outil, l'utilitaire affiche la date (module **datetime**)
- L'utilitaire doit permettre de **récupérer des informations du système**
  - Le **système d'exploitation** (module **os**)
- On doit avoir un **système très simpliste de gestion de fichiers**
  - Création et navigation dans différents dossiers (module **os** et **shutil**)
- Une calculatrice basique doit être implémenté
  - Opérations arithmétiques basiques.
  - Calcul du cosinus (module **math**)
  - Calcul d'un nombre aléatoire avec possibilité de le deviner (module **random**).

Évaluations

Évaluation du portfolio

Peu importe le projet réalisé, il faut au moins la présence d'un script Python réalisé en solo et *from scratch*.

Examen Blanc sur PC

Lors de l'examen final sur ordinateur, chaque étudiant va avoir une série aléatoire d'instructions pour réaliser un script Python.

Note : si l'étudiant n'a pas d'ordinateur ou si celui-ci ne fonctionne pas, alors il aura des questions qui sont plutôt présente dans la section **Examen Blanc sur table**

Examen Blanc sur table

3. Programmation Java

Plan prévisionnel du cours

Stack technique

- Téléchargement de l'IDE **IntelliJ IDEA**, en version Community 2022 (attention, la version normale est payante).

3. 1. Introduction

- Langage développé par **Sun Microsystems** et distribué en 1995.
- Il y a, aujourd'hui, plus de 3 milliards d'appareils qui utilisent Java.
- Le langage Java est proche du C++ ou du C# par sa syntaxe, et proche de Python par son fonctionnement (c'est également un langage interprété)

Caracteristiques de Java

- **Orienté Objet** : Java est un langage orienté objet, utilisant des classes, des méthodes et des objets.
- **Interprété** : le code source passe par une phase de **compilation à deux étapes**, c'est-à-dire qu'il est d'abord **compilé en *bytecode*** par le compilateur Java, et ce même *bytecode* est exécuté (interprété) par la JVM (*Java Virtual Machine*)
- **Portable** : le code Java est complétement indépendant du matériel, et le même code peut être exécuté sur l'ensemble des machines possédant une JVM.
- **Bibliothèque standard**

Utilisation de Java

- Développement **d'applications mobiles** (Android).
  - Utilisation de l'IDE **Android Studio** basé sur IntelliJ.
- **Développement web** (Backend).
- **Bases de données.**
- Java n'est pas utilisé dans le milieu du développement de jeux vidéo, contrairement à ce qui est affirmé dans de nombreux sites. L'exemple de Minecraft est très spécifique, et désuet à l'heure actuelle.

Bases

- On peut concevoir deux types de programme avec la version standard de Java
  - Une application fonctionnant sur un système d'exploitation (Mac OS, Windows).
  - Une applet qui fonctionne sur un navigateur (Chrome, Firefox).
- Tout code Java doit être **à l'intérieur d'une classe** et n'importe quel programme Java doit avoir la méthode **main()**. Le nom de la classe publique d'un script Java doit être le même que le nom du fichier.

- Attention, les applets n'ont pas de méthode main() et ne peuvent pas être testées via l'interpréteur, mais seulement via un applet viewer.

- Java, comme de nombreux langages, est **sensible à la casse**, cela signifie que les majuscules et minuscules sont prises en compte. Par exemple, la variable Ma_variable et ma_variable sont deux variables différentes.
- Un **package Java** est un fichier (ou une unité) qui regroupe une ou plusieurs classes. C'est comparable aux bibliothèques des langages C ou C++.

3. 2. Un langage Oriente Objet

Methodes

- La **méthode** d'une classe est un bloc de code qui va fonctionner seulement s'il est appelé par le script. Elle permet de créer du code réutilisable pour de nombreux cas de figure (plutôt que de réécrire le même code systématiquement).
- Une **méthode** qui ne va pas renvoyer de valeur doit commencer par **void**.
- Le mot clé **static** signifie qu'une méthode appartient à la classe elle-même et non pas à ses objets, et permet d'accéder à une méthode sans instancier un objet. Par exemple, si la classe **Car** possède la méthode static **Car_Type**, alors on peut accéder à cette méthode sans créer un objet **Car**.
- On peut utiliser des **paramètres** et **arguments** dans une méthode.
  - Un **paramètre** est spécifié juste après le nom de la méthode, entre parenthèses.
  - Un **argument** est spécifié lors de l'utilisation de la méthode

3. 3. Exercices : Terminal

Prerequis

- Installer tous les outils listés en première page du chapitre

Exercice A

- Création d'une première classe publique
  - Créer **20 variables**.
  - Afficher les informations de **5 variables.**

Exercice B

- Création d'un programme qui, à partir d'un nombre, va afficher une table de multiplications sur 10 lignes.

Exercice C

- Création de **deux classes** (dans le même fichier)
  - Chaque classe doit avoir au moins 3 méthodes.
  - Créer 5 objets différents pour chaque classe.

Exercice D

- Création d'une calculatrice textuelle.
  - Possibilité pour l'utilisateur d'entrer plusieurs nombres, et d'effectuer les opérations classiques (addition, division, etc.).

Exercice E

- Création d'un programme console pour deviner un nombre.
  - Le programme va démarrer et va générer un nombre directement.
  - L'utilisateur propose ensuite un premier nombre, et le programme indique simplement si c'est plus que le nombre aléatoire ou moins.

3. 4. Exercices : JavaFX

Exercice A : Gestion d'inventaire

- Créer une interface Java permettant de gérer un inventaire complet. Il faut pouvoir créer un nouvel item d'inventaire, afficher une liste de plusieurs éléments, donner la quantité, et supprimer un item de l'inventaire.

Plan prévisionnel du cours

Parcours Developpement Web Full-Stack

Programme

Plan prévisionnel du cours

Module 01 Introduction au developpement Web via HTML et CSS

- Bases du HTML (structure, balises, attributs, sémantique, accessibilité, compatibilité, etc.)
- Bases du CSS (reset, sélecteurs, propriétés, Box model, media queries, etc.) et SASS/ SCSS,
- Introduction aux CMS (WordPress, Joomla, etc.)
- Exercices de Portfolio (Réalisation de plusieurs maquettes ou *Templates statiques)*.

Module 02 Programmation Web Front-End via JavaScript (*Vanilla*)

- Introduction générale à la programmation (langages, paradigmes, algorithmes, Design Patterns, architecture logicielle MVC/ MVP…, etc.)
- Bases de JavaScript (caractéristiques, syntaxe, variable, types de données, conditions, fonctions, événements, objets, prototypes, classes, erreurs, etc.)
- Exercices de Portfolio (création de plusieurs pages Templates dynamiques et interactives)

Notice des exercices et de l'evaluation

- Exercices en classe : exercice sur les différents langages (HTML, CSS, Js, PHP) et *Frameworks* (React)
- Exercices de portfolio : tous les exercices sont à réaliser en solo, from scratch et à déposer sur son compte GitHub (ou alors à publier sur son propre site internet personnel/ professionnel)
- Évaluation finale sur ordinateur : réalisation d'une mini-site HTML, CSS avec quelques interactions en JavaScript

Module 00 Historique

Internet est probablement l'un des outils les plus importants qui existe aujourd'hui.

Developpement d'internet

- **1969** : développement d'un réseau de communication américain ARPANET
- **1989** : définition du couple de protocoles TCP/ IP par la RFC (et abandon progressif d'ARPANET).
- **1990** : dissolution du projet ARPANET
- **1995** : Invention du WWW (World Wide Web) par Tim Berners-Lee

Module 01 Architecture Web

Lorsque le navigateur va construire une page web, il va s'occuper de plusieurs éléments importants :

- **DOM** (Document Object Model)
- **CSSOM** (CSS Object Model)
- **AOM** (Accessibility Object Model)

Les langages **HTML** (**HyperText Markup Language**) et **CSS** (Cascading Style Sheets) sont deux langages standards utilisés pour le développement de pages web. Utilisés conjointemenet (avec d'autres langages qu'on verra par la suite), ils permettent de créer des sites attrayants, et sont deux langages incontournables du développement web dont il faut absolument maîtriser les fondamentaux.

Tout d'abord, le **HTML** est ce qu'on appelle un **langage de balisage**, qui sert à structure et présenter le contenu d'une page via différentes balises et attributs. Plus précisément, les **balises** permettent de créer des éléments qui structurent une page, comme des en-têtes (headings), des paragraphes, des images, des formulaires, etc. Alors que les **attributs** servent plutôt à préciser les caractéristiques de certaines balises, comme la taille, la source de l'image, et ainsi de suite.

image 4 Balise HTML (MDN)

Alors que le **CSS** est un **langage de feuille de style**, utilisé pour définir l'apparence visuelle d'une page web. On applique ainsi des **styles** aux éléments HTML pour modifier la couleur, les polices, les marges, les bordures ou d'autres propriétés. De manière plus technique, on utilise des **sélecteurs** pour cibler des éléments HTML précis, qu'on va modifier via différentes **déclarations**, dont chacune possède une propriété et une valeur.

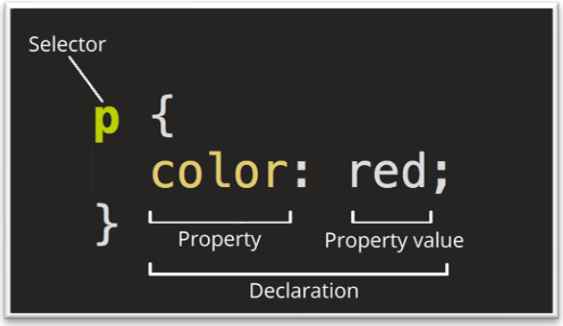


image 5 Sélecteur CSS (MDN)

Attention, ce ne sont pas des langages de programmation comme le JavaScript, le Python ou encore le C++. Les langages HTML et CSS ne permettent pas de gérer des données, de manipuler des variables (bien que la notion de variable existe dans le CSS) et encore moins de gérer des événements interactifs (sauf certaines exceptions) ! Néanmoins, on pourra étudier le langage de script SASS (SCSS) qui se rapproche grandement des langages de programmation, et qui permettra d'en introduire quelques principes.

Fondamentaux du HTML 5

- Documentation MDN sur le HTML
- Le HTML5 intègre la notion de **balises sémantiques** :
  - Sections de document : **<nav> <header> <main> <section> <aside> <footer>**
  - Parties de texte : ** <time>**
  - L'attribut global **<role>** permet de spécifier un élément sémantique à des balises qui ne sont pas sémantiques, comme ** ** ou ****. En réalité, <nav> est la même chose que **<nav role="navigation">**
- Prise en charge native des fichiers **audio** et **vidéo**

Fondamentaux du CSS 3

- Documentation MDN sur le CSS
- **Chaque navigateur** (Firefox, Chrome, Opera…) **possède un User Agent Stylesheet**, c'est-à-dire un fichier CSS qui va appliquer des règles de base pour afficher toute page HTML n'ayant pas spécifié de fichier CSS. Ces règles de base vont modifier la police d'écriture, sa taille, les marges, et autre.
  - Généralement, on réalise un Reset pour annuler les styles CSS des différents navigateurs, et que notre page affiche uniquement nos propres styles.
- Différents **sélecteurs CSS** : *class, id, universal, element…*

Fondamentaux du SASS (SCSS)

Le **SASS** est une version amélioré du CSS, avec des fonctionnalités supplémentaires.

Accessibilité (W3C)

- Utiliser les **balises sémantique**, autant que possible et lorsque le contexte le permet.
- Tous les **éléments non textuels** doivent voir apparaître un texte alternatif concis et descriptif, via l'attribut alt.
- Éviter d'utiliser uniquement la **couleur** comme moyen pour transmettre de l'information, et ajouter d'autres éléments pour augmenter la clarté (couleur + gras, par exemple).
- Toutes les **fonctions** de la page doivent être **utilisables via le clavier**, sans avoir besoin de la souris d'ordinateur.

- Le texte doit être compatible avec les technologies d'assistance, et doit posséder certaines caractéristiques (exemple : utiliser les *Headings* pour les différents titres, ne pas justifier l'alignement du texte, etc.).

Exercice : Static Web Templates

L'objectif de cet exercice est la réalisation d'une ou de plusieurs **Templates Web**, en HTML et CSS (il sera également possible d'ajouter, plus tard, du JavaScript, mais seulement pour les fonctionnalités impossible à réaliser en CSS, voire un langage Backend comme le PHP).

Le **choix thématique** des *Templates Web* est totalement libre, et doit correspondre soit à vos projets actuels, soit à vos aspirations. On peut trouver quelques idées sur EnvatoMarket ou TemplateMonster. Néanmoins, on peut également partir d'un mot clé, puis ajouter « template » sur Google images (exemple : « Restaurant Web Template »). On peut imaginer faire le site d'un magasin en ligne, d'un restaurant, d'un service de voyage (ou d'une agence), d'un centre sportif, etc.

Structure de base

Il faudra d'abord créer un dossier avec le nom du projet et, à l'intérieur, les dossiers suivants : **Assets** (pour les images, vidéos, fichiers audio) et **Core** (pour les fichiers comme CSS ou JavaScript). On pourra également rajouter, pour un site multilingue, des **dossiers par langue** (un dossier **en** pour l'anglais, un **fr** pour le français, etc.).

Il faut ensuite utiliser **Visual Studio Code** et ouvrir le dossier en question.

Consignes HTML

En fonction du thème (si c'est possible), il faudra réaliser plusieurs pages **HTML** (entre 2 et 6). Il est également possible de les réaliser en PHP, si vous maîtrisez déjà ce langage.

- Chaque page doit posséder une barre de navigation dans une , avec différents boutons qui permettent d'afficher les multiples pages.
- Il faut que chaque page puisse avoir une structure complète, avec les éléments suivants :
- Insérer un **contenu textuel**, notamment des paragraphes, des titres de plusieurs niveaux, une mise en forme de certains mots comme le gras, le soulignement et ainsi de suite,
- Utilisez des **structures sémantiques** pour découper votre contenu textuel en différentes parties (en-tête, sections, etc.)
- Intégrer différentes images, fichiers audios et vidéos, dont certaines doivent être responsives,
- Ajouter au moins deux listes, tableaux et formulaires
- Utiliser une balise **<iframe>** pour intégrer du contenu qui vient d'autres sites.

Consignes CSS

⚠ *Interdiction d'utiliser une bibliothèque CSS de type Bootstrap ou Tailwind*

L'objectif est de réaliser un **seul fichier CSS** pour la totalité du site, qui doit être importé dans les différentes pages HTML. Celui-ci peut prendre n'importe quel nom (style.css ou autre).

- Réaliser un Reset du CSS
- Utiliser différents sélecteurs pour cibler du contenu textuel (paragraphes, titres et sous-titres, balises sémantiques, etc.)
- Utiliser différentes propriétés pour modifier la manière dont les éléments HTML sont affichés
- Ajouter les **media queries** pour la gestion du Responsive
- Intégrer des animations et transitions

Le **langage JavaScript** est un langage de programmation extrêmement populaire pour le développement de sites, et s'occupe particulièrement de rendre les sites interactifs et dynamiques.

Caracteristiques

- JavaScript est un langage véritablement **orienté objet**, mais attention, il s'appuie sur des **prototypes** et non pas des classes (cf. les paradigmes de programmation),
- JavaScript est un langage interprété
- Il est ***Single Threaded***, cela signifie qu'il va exécuter une seule chose à la fois.

Exercice : Semi-Dynamic Web Templates

Cet exercice exige la réalisation d'un site web semi-dynamique, c'est-à-dire intégrant de l'interactivité et des fonctionnalités qui modifient la page web via JavaScript.

Évaluations

Évaluation du portfolio

Projet de Site web ou d'application complète

- **Utilisation d'une plate-forme de gestion de projet** (au choix) et de rédaction d'une documentation (justification sur les choix des technologies, étude du marché, contexte, évaluation des risques, spécifications techniques, etc.). Dépôt d'un export du dossier de projet.
  - **ClickUp** : export des données via Excel pour uploader sur GitHub
  - **MS Office/ WPS Office** : Upload des fichiers source sur Github
  - **Notions** : Export d'un PDF de l'ensemble de la documentation
- **Création d'un site intégral**
  - Architecture du site : *Landing Page* • SPA (Single Page Application) • *Static/ Dynamic Website*
- **Stack Technique (au choix)**
  - **Landing Page** : HTML • CSS/ SASS • JavaScript • PHP (facultatif)
  - **Static/ Dynamic Website**: HTML • CSS/ SASS • JavaScript • PHP (facultatif)
  - **Application Web 2D** : HTML • CSS/ SASS • JavaScript & React/ React Native
  - **Application Web 2D/ 3D** : Unity (C#) avec WebGL
- **Fonctionnalités de base**
  - Site Responsive
  - Menu de navigation pour Desktop et Mobile
  - Présentation de différents produits/ objets/ éléments
  - Utilisation de tableaux, listes (ordonnées ou pas), etc.

4. RESEAUX INFORMATIQUES

Plan prévisionnel du cours

MAITRISE DES NOTIONS

- **Topologies :**
- **Protocoles** : TCP/ IP • Wi-Fi
- **Sécurité** :
- *

ARCHITECTURE CLIENT – SERVEUR

4. 1. Introduction aux reseaux informatiques

Un **réseau informatique** permet de faire communiquer des équipements informatiques (nœuds) qu'on appelle des nœuds, via des liens de différents types (câbles, liaisons radio, fibre optique, etc.).

Chaque machine qui est connectée à un réseau s'appelle un **hôte**, avec lequel un numéro d'identification est attribué, **l'adresse IP**, qui permet d'identifier l'hôte et le réseau auquel il est connecté.

Généralement, les **deux architectures** principales sont les suivantes :

- L'architecture client serveur
- L'architecture Peer-to-Peer

Adresses

- **Adresse MAC** : adresse qui permet d'identifier n'importe quel matériel, avec une partie qui identifie le constructeur (3 octets) et une autre partie qui identifie l'appareil de manière unique (3 octets).
- **Adresse IP** : existe en deux versions (IPv4 et IPv6)

Protocoles

Deux types de protocoles en général : les **protocoles routables** (TCP/ IP, IPX/ SPX…) et les **protocoles non routables** (NetBIOS, NetBEUI, etc.).

Protocoles Routables : TCP/ IP

Le protocole TCP assure la fiabilité des communications sur le réseau. Il a été créé en s'inspirant du réseau français cyclades.

Topologies

On distingue deux types de topologies : les topologies physiques (configuration spatiale) et les topologies logiques ().

4. 2. Architecture client serveur

Architecture Client Serveur standard

- Front-End
- Serveur d'application (Application Server)
- Serveur de bases de données (Database Server)

Architecture N-tiers

Plan

A. Definition de l'architecture client-serveur

Structure qui va répartir les tâches entre fournisseurs (serveurs) et demandeurs du service (clients). Elle permet surtout un partage des ressources.

- Un serveur est un dispositif informatique qui offre des services à des clients
- Un client est un logiciel qui va faire des requêtes à un serveur.

Composants de l'architecture client-serveur (client, serveur, reseau)

- Rôles du client et du serveur dans l'architecture client-serveur
- Communication entre le client et le serveur (requêtes et réponses)
- Protocoles de communication client-serveur (HTTP, FTP, SMTP, etc.)
- Types de clients (navigateurs web, applications mobiles, etc.)
- Types de serveurs (web, fichiers, messagerie, etc.)
- Modèle de développement client-serveur (cycles de développement, tests, déploiement)
- Sécurité dans l'architecture client-serveur (authentification, autorisation, chiffrement)
- Avantages et inconvénients de l'architecture client-serveur
- Évolution de l'architecture client-serveur vers l'architecture microservices

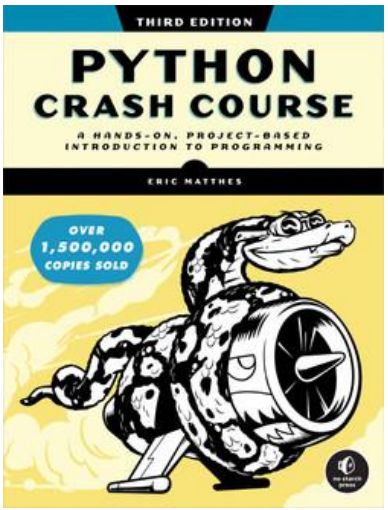
5. Ressources

Ressources generales en ligne

➡ Digital Ocean • Python Official Wiki • Real Python • W3Schools

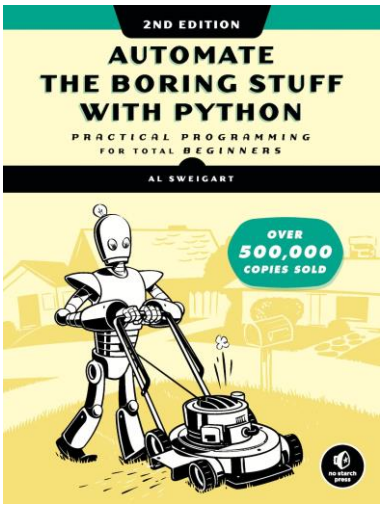
5. 1. Ressources livresques

Programmation : Python



Python Crash Course 3nd

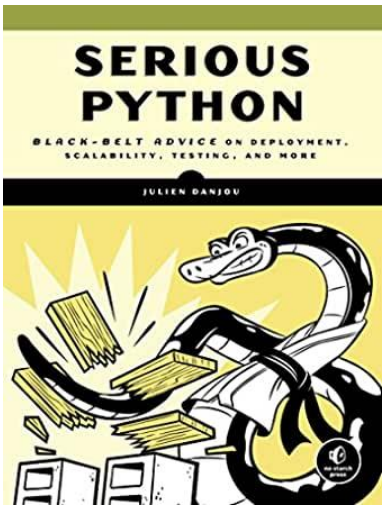
Eric Matthes (2023)



Automate the Boring Stuff 2nd

Al Sweigart (2020)

En ligne sous Creative Commons



Serious Python: Black-Belt

Julien Danjou (2018)

5. 2. Ressources en ligne

Gestion de projets informatiques

- **Sites**
  - Gestion de Projet (Manager Go). https://www.manager-go.com/gestion-de-projet/
  - https://bpesquet.developpez.com/tutoriels/introduction-genie-logiciel/#LI
  - https://www.zentao.pm/blog/different-methodologies-between-agile-and-waterfall-837.html
- **Vidéos (résumés)**
  - Scrum in under 5 minutes. https://www.youtube.com/watch?v=2Vt7Ik8Ublw
  - Scrum vs Kanban – What's the Difference? https://www.youtube.com/watch?v=rIaz-l1Kf8w

Programmation (generalites)

- **Invivoo** : paradigmes de programmation
- **Refactoring Guru** : Design Patterns in Python

Programmation : Java

- **Programiz** : Java Methods • Java Class and Objects

Programmation : Python

- 🇺🇸 **Programiz** : Python Object Oriented
- 🇺🇸 **Real Python** : Basic Data Types • Object-Oriented Programming