Groupe 3

Membre du groupe :

**AKADE Sem**

**FOLLY-YISSOUH Kinvi**

**KOURAFEI Ruth**

**SONHOUIN Billali**

**Chargé du Cours : Mr APEKE**

**L’APPROCHE OBJET**

Table des matières

[Introduction 2](#_Toc93948665)

[1. Comprendre les principales notions de l’Approche Objet 3](#_Toc93948666)

[2. Rappel sur les Paradigmes de programmation [7] 3](#_Toc93948667)

[2.1 La programmation impérative [5] 4](#_Toc93948668)

[2.2 La programmation logique [7] 4](#_Toc93948669)

[2.3 La programmation orientée objet [6] 5](#_Toc93948670)

[3. Les objets : identité, état et comportement. Relations au monde réel et aux systèmes informatiques 6](#_Toc93948671)

[3.1 Caractéristiques d’un objet 6](#_Toc93948672)

[3.2 Relation entre le monde réel et les Systèmes Informatiques 6](#_Toc93948673)

[4. Avantages et popularité de l’Approche Objet 7](#_Toc93948674)

[4.1 Les avantages de l’approche Objet [1] 7](#_Toc93948675)

[4.2 Popularité de la Programmation Orientée Objet [3] 7](#_Toc93948676)

[5. Abstraction, Encapsulation, Classification, Classes et Instances, Classe abstraite, Héritage 8](#_Toc93948677)

[5.1 Abstraction 8](#_Toc93948678)

[5.2 Encapsulation 9](#_Toc93948679)

[5.3 Classification 9](#_Toc93948680)

[5.4 Classes 10](#_Toc93948681)

[5.5 Classes et instances 10](#_Toc93948682)

[5.6 Classe abstraite 10](#_Toc93948683)

[5.7 Héritage 10](#_Toc93948684)

[6. Méthodes et envois de messages entre objets : Polymorphisme, Surcharge et redéfinition 11](#_Toc93948685)

[6.1 Définition du Polymorphisme 11](#_Toc93948686)

[6.2 Polymorphisme ad hoc (Surcharge) : 11](#_Toc93948687)

[6.3 Polymorphisme paramétrique : 12](#_Toc93948688)

[6.4 Le polymorphisme d’héritage : 12](#_Toc93948689)

[7. Exercices 13](#_Toc93948690)

[Conclusion 14](#_Toc93948691)

[Références bibliographiques 15](#_Toc93948692)

# Introduction

L’approche objet introduite par la programmation orientée objet a été une méthode de programmation qui a révolutionné la manière de programmer. Ce paradigme reste largement utilisé de nos jours dans les différents ateliers de développement avec des innovations que ne cessent d’apporter les différents langages qui portent ce paradigme.

Cet exposé présente les notions de base de la conception et de la programmation  
orientée objet (POO). Il s’agit pour l’essentiel des notions d’objet, de classe, d’abstraction, de message, d’héritage et de polymorphisme.   
Afin de bien aborder ces différentes notions, nous allons dans un premier faire un rappel des différents paradigmes de programmation existants. Nous allons ensuite aborder les objets dans leur identité, état et comportement et leurs relations au monde réel. Nous allons ensuite enchainer avec les notions fondamentales de la POO que sont : L’abstraction, l’encapsulation, l’héritage et le polymorphisme. Nous allons terminer avec de petits exercices qui parlent de ces notions.

# Comprendre les principales notions de l’Approche Objet

La programmation orientée objet consiste à modéliser informatiquement un ensemble d'éléments d'une partie du monde réel (que l'on appelle domaine) en ensemble d'entités informatiques.

Ces entités informatiques sont appelées objets. Il s'agit de données informatiques regroupant les principales caractéristiques des éléments du monde réel (taille, couleur, ...).

Un **Objet** représente une entité du monde réel ou du monde virtuel (pour les objets immatériels) qui se caractérise par des attributs, des états significatifs et un comportement.

**Exemple** :

Soit la modélisation d'un objet « **Voiture** ».

Une voiture est caractérisée par :

* Liste de ses attributs (attributs) et leur état (valeur des attributs)
* Marque : Renault
* Couleur : grise
* Immatriculation : AF 1234
* Son comportement :
* Démarrer
* Changer de vitesse
* Arrêter

C’est le concept fondamental de la programmation orientée objet (POO).

# Rappel sur les Paradigmes de programmation [7]

Un paradigme est un style fondamental de programmation, définissant la manière dont les programmes doivent être formulés.

Chaque paradigme amène sa philosophie de programmation ; une fois qu'une solution a été imaginée par un programmeur selon un certain paradigme, un langage de programmation qui suit ce paradigme permettra de l'exprimer.

Il existe plusieurs paradigmes de programmation parmi lesquels, nous pouvons citer les principaux qui sont :

* Le paradigme impératif
* Le paradigme déclaratif
* Le paradigme orienté objet

La figure suivante représente les trois principaux paradigmes et leurs subdivisions

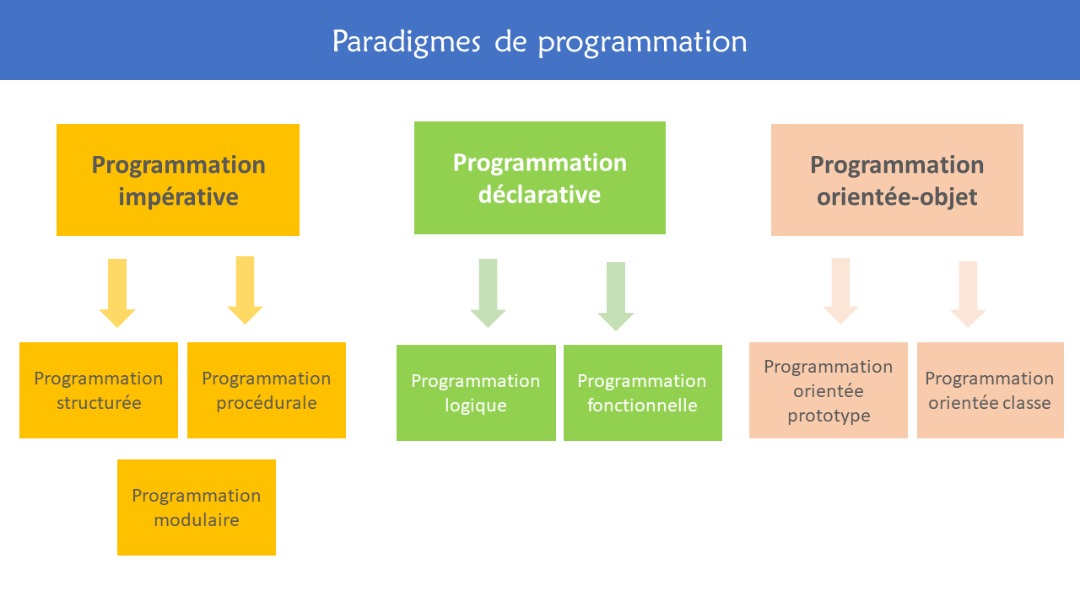


Figure 1 : Illustration des différents paradigmes de programmation

Voyons un peu en détail les principaux paradigmes que nous avons cités plus haut.

## La programmation impérative [5]

Le paradigme impératif est le plus répandu, les opérations sont une suite d’instructions exécutées par l'ordinateur pour modifier l'état du programme.

## La programmation logique [7]

La programmation logique consiste à exprimer les problèmes et les algorithmes sous forme de prédicats à l’aide d'une base de faits, d'une base de règles et d'un moteur d'inférence.

La programmation logique est une forme de [programmation](https://fr.wikipedia.org/wiki/Programmation) qui définit les [applications](https://fr.wikipedia.org/wiki/Application_(informatique)) à l'aide :

* D’une [base de faits](https://fr.wikipedia.org/wiki/Base_de_faits) : ensemble de faits élémentaires concernant le domaine visé par l'application,
* D’une [base de règles](https://fr.wikipedia.org/w/index.php?title=Base_de_règles&action=edit&redlink=1) : règles de [logique](https://fr.wikipedia.org/wiki/Logique) associant des conséquences plus ou moins directes à ces faits,
* D’un [moteur d'inférence](https://fr.wikipedia.org/wiki/Moteur_d%27inférence) (ou démonstrateur de théorème) : exploite ces faits et ces règles en réaction à une question ou requête.

Cette approche se révèle beaucoup plus souple que la définition d'une succession d'instructions que l'[ordinateur](https://fr.wikipedia.org/wiki/Ordinateur) exécuterait. La programmation logique est considérée comme une [programmation déclarative](https://fr.wikipedia.org/wiki/Programmation_déclarative) plutôt qu’impérative, car elle s'attache davantage au quoi qu'au comment, le moteur assumant une large part des enchaînements. Elle est particulièrement adaptée aux besoins de l’[intelligence artificielle](https://fr.wikipedia.org/wiki/Intelligence_artificielle), dont elle est un des principaux outils.

## La programmation orientée objet [6]

La programmation orientée objet (abrégé POO) consiste en la définition et l'interaction de briques logicielles appelées [**objets**](https://fr.wikipedia.org/wiki/Objet_(informatique)) ; ces objets représentent un concept, une idée. Chaque objet contient des *attributs* et des [*méthodes*](https://fr.wikipedia.org/wiki/Méthode_(informatique)) en rapport avec un sujet.

Comme type de programmation orientée objet, on peut distinguer la programmation orientée prototype et la programmation orientée classe.

* **La programmation orientée prototype**

Dans ce paradigme, chaque [objet](https://fr.wikipedia.org/wiki/Objet_(informatique)) est créé à partir d’un [prototype](https://fr.wikipedia.org/wiki/Prototype_(patron_de_conception)) qui est lui-même un objet. Le prototype a donc une existence physique en [mémoire](https://fr.wikipedia.org/wiki/Mémoire_vive) et est mutable contrairement aux classes.

Le [JavaScript](https://fr.wikipedia.org/wiki/JavaScript), le [Lua](https://fr.wikipedia.org/wiki/Lua) ou le [Self](https://fr.wikipedia.org/wiki/Self_(langage)) sont des exemples de langages utilisant ce paradigme.

* **La programmation orientée classe**

La programmation orientée classe est basée sur la notion de [*classes*](https://fr.wikipedia.org/wiki/Classe_(informatique)). Une classe est statique, c’est la représentation abstraite de l’[objet](https://fr.wikipedia.org/wiki/Objet_(informatique)), c’est à ce niveau que se passe l’[héritage](https://fr.wikipedia.org/wiki/Héritage_(informatique)). Tout objet est donc l’instance d’une classe.

Les langages à classes peuvent être sous forme [fonctionnelle](https://fr.wikipedia.org/wiki/Programmation_fonctionnelle) ([CLOS](https://fr.wikipedia.org/wiki/Common_Lisp_Object_System)) comme sous forme [impérative](https://fr.wikipedia.org/wiki/Programmation_impérative) ([C++](https://fr.wikipedia.org/wiki/C%2B%2B), [Java](https://fr.wikipedia.org/wiki/Java_(langage))), voir les deux ([Python](https://fr.wikipedia.org/wiki/Python_(langage)), [Ocalm](https://fr.wikipedia.org/wiki/OCaml)).

Dans la suite de ce document, nous allons nous intéresser uniquement à la programmation orientée objet étant donné qu’elle constitue le principal objet du document.

# Les objets : identité, état et comportement. Relations au monde réel et aux systèmes informatiques

## Caractéristiques d’un objet

Tout objet en POO est caractérisé par :

* **Une Identité ou OID**

L’identité d’un objet lui permet d’être référencé par d’autre de façon unique et constante pendant toute sa durée de vie donc d’être distingué avec un objet ayant les mêmes valeurs d’attributs.

* **Un état**

L’état d’un objet à un instant donné, correspond à une sélection de valeurs, parmi toutes les valeurs possibles des différents attributs.

* **Un comportement**

Regroupe toutes les compétences d’un objet et décrit les actions et les réactions de cet objet.

On peut donc poser l’égalité suivante :

**Objet = Identité + Etat + Comportement**

## Relation entre le monde réel et les Systèmes Informatiques

Les modèles de données à objets ont été créés pour modéliser directement les entités du monde réel avec un comportement et un état. Le concept essentiel est bien sûr celui d’objet. Dans un modèle objet, toute entité du monde réel est un objet, et réciproquement, tout objet représente une entité du monde réel.

# Avantages et popularité de l’Approche Objet

## Les avantages de l’approche Objet [1]

L’approche objet offre les avantages suivants :

* **Modularité** : les objets forment des modules compacts regroupant des données et un ensemble d'opérations.
* **Abstraction** : Les entités objets de la POO sont proches de celles du monde réel. Les concepts utilisés sont donc proches des abstractions familières que nous exploitons.
* **Productivité et réutilisabilité** : Plus l'application est complexe et plus l'approche POO est intéressante en termes de productivité. Le niveau de réutilisabilité est supérieur à la programmation impérative.
* **Sûreté** : L'encapsulation et le typage des classes offrent une certaine robustesse aux applications.

## Popularité de la Programmation Orientée Objet [3]

Selon le Popularity of Programming Language [Index (PYPL)](http://pypl.github.io/PYPL.html), dans son classement actualisé de ce mois de janvier 2022, les deux premiers langages de programmation les plus utilisés dans le monde sont basés sur la programmation orientée objet. Ce site analyse la fréquence avec laquelle les tutoriels sont recherchés pour apprendre un langage de programmation. Il suppose que plus il y a de recherches, plus le langage de programmation est populaire.

Selon l’index PYPL, les 10 langages de programmation les plus utilisés dans le monde sont les suivants.

1. Python
2. Java
3. JavaScript
4. C/C++
5. C#
6. PHP
7. R
8. Objective-C
9. Swift
10. TypeScript

Le célèbre site de développement : developpez.com, donne lui aussi le classement suivant actualisé, toujours pour le mois de janvier 2022. [4]

1. Python
2. C
3. Java
4. C++
5. C#
6. VB
7. JavaScript
8. Assembly Language
9. SQL
10. Swift
11. PHP

# Abstraction, Encapsulation, Classification, Classes et Instances, Classe abstraite, Héritage

## Abstraction

L’abstraction est le processus qui consiste à représenter des objets qui appartiennent au monde réel dans le monde du programme que l’on écrit. Il consiste essentiellement à extraire des variables pertinentes, attachées aux objets que l’on souhaite manipuler, et à les placer dans un modèle informatique convenable.

Les avantages de l’abstraction sont :

* Réduction de la complexité,
* Prévention de la duplication de code,
* Possibilité de réutilisation renforcée,
* Sécurité de l’application et du programme étant donné que seuls les détails importants sont fournis à l’utilisateur.

## Encapsulation

C’est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet, c'est-à-dire en empêchant l'accès aux données par un autre moyen que les services proposés. Ce mécanisme permet donc de garantir l'intégrité des données contenues dans l'objet.

L'encapsulation permet de définir le niveau de visibilité des éléments de la classe.

On définit les niveaux ou droits d'accès suivants :

* **Privé (-)** : attribut non vu de l’extérieur de l’objet ; Il s'agit du niveau de protection des données le plus élevé ;
* **Public (+)** : attribut visible pour toutes les classes ; Il s'agit du plus bas niveau de protection des données ;
* **Protégé (#)** : l'accès aux données est réservé aux fonctions des classes héritières, c’est-à-dire par les fonctions membres de la classe ainsi que des classes dérivées.

## Classification

La classification permet de gérer la complexité en ordonnant les classes au sein d’arborescence de classe d’abstraction croissante.

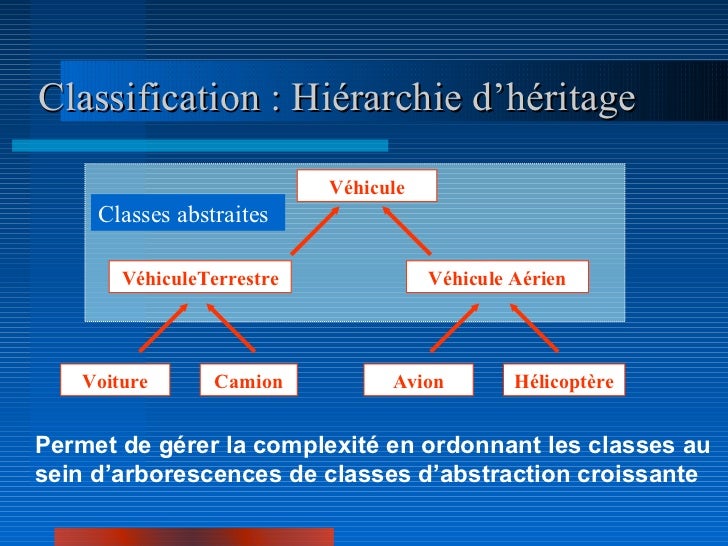


Figure 2 : Schéma montrant un exemple de classification

Pendant la classification, il faut éviter une trop grande profondeur de décomposition et veiller à ce que cette décomposition soit stable.

## Classes

Une classe est un ensemble d’objets qui ont les mêmes types de propriétés et qui appliquent les mêmes traitements à leurs données.

La définition d’une classe est :

* Le nom de cette classe,
* Les propriétés décrites par des attributs, leur type et éventuellement leur valeur initiale,
* Les traitements décrits par des méthodes, leur type de retour et leurs arguments.

## Classes et instances

Les objets associés à une classe se nomment des instances. Une instance est un objet, occurrence d'une classe, qui possède la structure définie par la classe et sur lequel les opérations définies dans la classe peuvent être appliquées.

## Classe abstraite

Une classe abstraite est classe qui ne donne pas directement des objets, et qui est donc non instanciable.

Une classe déclarée abstraite peut aussi définir des méthodes abstraites. Ces dernières devront obligatoirement être redéfinies dans les classes dérivées. C'est un moyen de s'assurer que la classe dérivée adoptera le comportement désiré.

Une méthode abstraite peut être vue comme une promesse de traitement qui existe dans les sous-classes concrètes, mais dont on ne peut pas encore écrire le code dans la classe mère.

## Héritage

L'héritage (en anglais inheritance) est un principe propre à la programmation orientée objet, permettant de créer une nouvelle classe à partir d'une classe existante ; de plus, elle peut étendre la nouvelle classe avec de nouvelles propriétés et méthodes. Cela a comme avantage majeur de ne pas avoir à repartir de zéro lorsque l'on veut spécialiser une classe existante.

# Méthodes et envois de messages entre objets : Polymorphisme, Surcharge et redéfinition

## Définition du Polymorphisme

Le nom de polymorphisme vient du grec et signifie qui peut prendre plusieurs formes.

Le polymorphisme signifie qu'une même opération peut se traduire différemment selon l'objet sur laquelle elle s'applique : C'est la capacité d’un objet à prendre plusieurs formes. C’est un des concepts essentiels de la programmation orientée objet. Alors que l'héritage concerne les classes (et leur hiérarchie), le polymorphisme est relatif aux méthodes des objets.

On distingue 3 types de polymorphisme :

* **Le polymorphisme ad hoc** (également surcharge ou en anglais overloading)
* **Le** [**polymorphisme paramétrique**](https://www.commentcamarche.net/contents/#parametrique) (également généricité ou en anglais template)
* **Le** [**polymorphisme d'héritage**](https://www.commentcamarche.net/contents/#heritage) (également redéfinition, spécialisation ou en anglais overriding)

La figure qui suit est une représentation schématique du polymorphisme et de ses différentes parties.

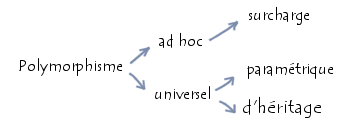


Figure 3 : Représentation du polymorphisme et ses différents embranchements

## Polymorphisme ad hoc (Surcharge) :

Le polymorphisme ad hoc permet d'avoir des fonctions de même nom, avec des fonctionnalités similaires, dans des classes sans aucun rapport entre elles (si ce n'est bien sûr d'être des filles de la classe objet).

Le polymorphisme ad hoc permet ainsi de définir des opérateurs dont l'utilisation sera différente selon le type des paramètres qui leur sont passés. Il est donc possible par exemple de surcharger l'opérateur et de lui faire réaliser des actions différentes selon qu'il s'agit d'une opération entre deux entiers (addition) ou entre deux chaînes de caractères (concaténation).

## Polymorphisme paramétrique :

Le polymorphisme paramétrique, appelé généricité, représente la possibilité de définir plusieurs fonctions de même nom mais possédant des paramètres différents (en nombre et/ou en type). Le polymorphisme *paramétrique* rend ainsi possible le choix automatique de la bonne méthode à adopter en fonction du type de donnée passée en paramètre.

Ainsi, on peut par exemple définir plusieurs méthodes homonymes addition () effectuant une somme de valeurs.

## Le polymorphisme d’héritage :

La redéfinition, aussi appelée « overriding », consiste à définir le comportement d’une méthode selon le type de l’objet qui l’invoque, elle consiste à donner une nouvelle implémentation à une méthode héritée sans changer sa signature.

La signature de méthode ou de constructeur est composée du :

* Nom de la méthode,
* Visibilité (modificateurs, ...)
* Nombre et du type de ses paramètres,
* Type qu'elle renvoie.

# Exercices

Exercice 1

Répondez aux questions suivantes :  
• Un même référent peut-il désigner plusieurs objets ?  
• Plusieurs référents peuvent-ils désigner un même objet ?  
• Un objet peut-il faire référence à un autre ? Si oui, comment ?  
• Pourquoi l’objet a-t-il besoin d’une classe pour exister ?  
• Un objet peut-il changer d’état ? Si oui, comment ?  
• Qu’appelle-t-on un envoi de message ?  
• Comment un premier objet peut-il conduire un deuxième objet à changer d’état ?

Exercice 2

Cet exercice a pour but de réfléchir sur la conception d’un programme, sa structuration en classes. Il ne s’agit pas pour le moment de réaliser ce programme, mais juste de concevoir son architecture.

On fait des cocktails avec différents liquides (alcools, sodas, jus de fruits). On a un bar avec des bouteilles qui peuvent être pleines ou à moitié vides. On a des shakers qui ont une contenance donnée. Il y a des recettes de cocktails qui indiquent seulement les proportions. Ces recettes peuvent s’appliquer à des quantités plus ou moins grandes selon les besoins du moment.

Les cocktails se font en déversant une partie du contenu des bouteilles dans des shakers. Après, il faut secouer. Les shakers sont ensuite vidés (dans les verres, mais on ne tiendra pas compte des verres dans cette application). Il faut les laver après usage.

Question : quelles classes faut-il créer ? Quelles informations faut-il dans chaque classe ? Quelles méthodes faut-il écrire, et dans quelle classe les mettre ?

Exercice 3

Placez dans un schéma en arbre, du plus général au plus spécifique, les concepts  
suivants :  
• humain, footballeur, avant-centre, sportif, skieur, spécialiste du slalom géant ;  
• guitare, instrument de musique, trompette, instrument à vent, instrument à corde,  
violon, saxophone, voix.

N.B. Une représentation UML peut être utilisée

# Conclusion

Dans cet exposé, nous avons abordé les bases de la programmation orientée objet (POO). Nous avons premièrement défini ce que c’est qu’un objet avant de parler de l’encapsulation, de l’héritage et du polymorphisme qui constituent les éléments fondamentaux du paradigme orienté objet.

Il serait difficile, voire même impossible d’aborder en quelques pages toutes les notions introduites par la POO. Nous nous sommes concentrés sur les éléments essentiels.

Nous espérons que les rappels faits dans cet exposé permettront de mieux cerner les rudiments qui seront évoqués dans le chapitre suivant qui concerne l’analyse et la conception objet ainsi que l’introduction à l'UML.

Personnellement, ce travail de recherche nous a permis de revisiter la POO, de découvrir de nouveaux concepts notamment ceux liés à la classification. Nous avons ainsi beaucoup appris en faisant ces recherches que ne saurait nous apporter un cours reçu. Nous espérons que les autres groupes auraient les mêmes bénéfices à la fin de leurs exposés.

Enfin, nous espérons également gagner autant en connaissances à la fin du projet de conception qui suivra cet exposé.

# Références bibliographiques

[1] OLIVIER CURE, « Cours Java 2011-12, 2011 » [En ligne].

Disponible : http://perso.univ-mlv.fr/ocure/imn/imnJava0809\_cm2.pdf. [Accès le 17 janvier 2022].

[2] LOUNIS HAKIM, « Concepts de base de l ‘orienté objet, 2008 » [En ligne].

Disponible : https://www.labunix.uqam.ca/~lounis\_h/dic9250-10-fall2008/generalites\_GL/Concepts-OO.pdf. [Accès le 15 janvier 2022].

[3] github.io, « PYPL PopularitY of Programming Language, janvier 2022 » [En ligne].

Disponible : https://pypl.github.io/PYPL.html. [Accès le 22 janvier 2022].

[4] BILL FASSINOU « Python est le langage de programmation le plus utilisé et il ne montre aucun signe de ralentissement, 5 janvier 2022 » [En ligne].

Disponible : https://programmation.developpez.com/actu/329957/Python-est-le-langage-de-programmation-le-plus-utilise-et-il-ne-montre-aucun-signe-de-ralentissement-selon-l-index-TIOBE-de-janvier-2022/ [Accès le 22 janvier 2022].

[5] Wikipédia, l’encyclopédie libre « Programmation impérative, 7 octobre 2021 » [En ligne].

Disponible : https://fr.wikipedia.org/wiki/Programmation\_imp%C3%A9rative. [Accès le 22 janvier 2022].

[6] Wikipédia, l’encyclopédie libre « La programmation orientée objet (POO), 3 décembre 2021 » [En ligne].

Disponible : https://fr.wikipedia.org/wiki/Programmation\_orient%C3%A9e\_objet [Accès le 22 janvier 2022].

[7] Wikipédia, l’encyclopédie libre « Paradigme (programmation), 16 novembre 2021 » [En ligne].

Disponible : https://fr.wikipedia.org/wiki/Paradigme\_(programmation) [Accès le 21 janvier 2022].