Cursus SALESFORCE

M2I - Formation - 2024

Donjon Audrey

Algorithmes et programmation structurée

Donjon Audrey
Formatrice Frontend

Contact :

www.linkedin.com/in/audrey-djn
audrey._
audrey-donjon

Sommaire

Bases de l’algorithmie

Qu’est ce qu’un algorithme?

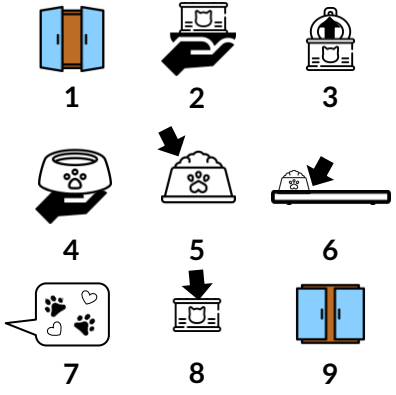
Un **algorithme** est un **ensemble d'instructions** qui remplit un **objectif donné**.

Ces **instructions** sont effectué **étape par étape et dans l’ordre,** elles sont écrites dans un **langage spécifique** puisqu’un ordinateur est incapable de comprendre le langage humain directement .

Le **langage** va servir de **passerelle** entre le langage humain et le langage que la machine peut comprendre.

Exemple simple :

Objectif : Préparer le repas du chat



1. **Ouvrir** le placard où est rangée la nourriture pour chat.
2. **Prendre** une boîte ou un sachet de nourriture pour chat.
3. **Ouvrir** la boîte ou le sachet.
4. **Prendre** la gamelle du chat.
5. **Remplir** la gamelle du chat avec la nourriture .
6. **Placer** la gamelle de nourriture où le chat mange.
7. **Appeler** le chat pour qu'il vienne manger.
8. **Fermer** la boîte ou le sachet de nourriture et le **ranger** dans le placard.
9. **Fermer** le placard.

Chaque étape est une instruction claire et l'ensemble des étapes résout le problème de préparer le repas du chat

Qu’est ce qu’un algorithme?

Exemple en programmation :

Objectif : Calculer la Somme des Nombres de 1 à N (On définit N à 10 pour notre cas)

input

```
N = 10
somme = 0
i = 1
tant que i <= N
     somme = somme + 1
     i = i + 1
Afficher somme
```

1. **Initialiser** une variable **somme** à **0**.
2. **Initialiser** une variable **i** à **1**.
3. **Répéter** les étapes suivantes **tant que i** est **inférieur ou égal** à **N**.
    - **Ajouter i** à **somme.**
    - **Incrémenter i** de **1.**
4. **Afficher** la valeur de **somme.**

output

```
1
2
3
4
5
6
7
8
9
10
```

Importance des algorithmes :

Les algorithmes sont essentiels pour **résoudre des problèmes** de manière **systématique** et **efficace**. Ils permettent **d'automatiser** des **tâches répétitives** et **complexes**. Lorsque l’on cherche à **optimisé son algo** il peut exécuter des tâches plus **rapidement** et avec **moins de ressources**.

Séquence, sélection et itération

En algorithmie nous avons **3 concepts de base** qui nous permettent de **structurer** et **organiser** les instructions pour résoudre un problème ou remplir un objectif.

1. Séquence

La séquence signifie que les **instructions** sont exécutées **les unes après les autres**, dans l'**ordre où elles apparaissent**. C'est la **forme la plus simple** d'un algorithme.

Exemple : Préparer un café

- Faire bouillir de l'eau.
- Mettre du café moulu dans un filtre.
- Verser l'eau chaude sur le café.
- Remuer.
- Servir le café.

Dans cet exemple, chaque étape suit la précédente de manière linéaire et sans conditions particulières.

Séquence, sélection et itération

2. Sélection

La sélection signifie que certaines instructions ne sont **exécutées que si une condition spécifique est remplie**. C'est comme prendre une décision dans l'algorithme.

Exemple : Vérifier si un nombre est pair ou impair

- Prendre un nombre.
- Si le nombre modulo 2 (le reste de la division par 2) est 0, alors le nombre est pair.
- Sinon, le nombre est impair.

En pseudo-code, cela ressemblerait à ceci :

```
si (nombre % 2 == 0) alors
     afficher "le nombre est pair"
sinon
     afficher "le nombre est impair"
```

Séquence, sélection et itération

3. Itération ou répétition

L'itération signifie que **certaines instructions sont répétées plusieurs fois**, généralement jusqu'à ce qu'une **condition soit remplie**. C'est comme **une boucle dans l'algorithme**.

Exemple : Calculer la somme des nombres de 1 à 5

- Initialiser une variable somme à 0.
- Initialiser une variable i à 1.
- Tant que i est inférieur ou égal à 5, ajouter i à somme et incrémenter i de 1.
- Afficher la valeur de somme.

En pseudo-code, cela ressemblerait à ceci :

```
somme = 0
i = 1
tant que (i<=5) faire
     somme = somme + i
     i = i + 1
afficher somme
```

Exercice 01 :

Comprendre les concepts

Écrivez un algorithme/les étapes pour chaque cas suivant en choisis le bon concept pour chacun :

1 – Écrivez un algorithme qui affiche les nombres de 1 à 10

2 – Écrivez les étapes pour préparer un bol de céréales

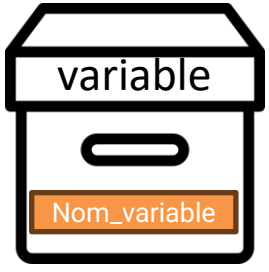
3 – Écrivez un algorithme qui vérifie si une personne est majeure (18 ans ou plus)

Structures de base de la programmation

Variables et types de données

1. Variables

Une **variable** est un conteneur qui permet de **stocker** des **valeurs**. Pensez à une variable, les variables ont des **noms** et ces noms sont utilisés pour accéder aux valeurs stockées.



2. Types de données

Les **types de données** indiquent au programme le **genre de données** qu'une variable peut **contenir**. Quelques exemples de **type de données courant** :

- **Int** (entier) : Nombre entier sans décimales
(ex: 1,2,-5)
- **Float** (nombre à virgule flottante) : Nombre décimal
(ex : 1.5, -0.002)
- **String** (chaîne de caractères) : Séquence de caractères
(ex : "Hello world" , "chat")
- **Bool** (booléen) : Valeur de vérité
(True ou False)

```
# Déclaration de variables
age = 25                # int
taille = 1.75           # float
nom = "Alice"           # string
is_student = True       # bool
```

Opérateurs arithmétiques et logiques

1. Opérateurs arithmétiques

Ces **opérateurs** sont utilisés pour effectuer des **opérations mathématiques** sur les **variables** et les **valeurs**.

Opérateurs arithmétiques :

- **+** : Addition
- **-** : Soustraction
- ***** : Multiplication
- **/** : Division
- **%** : Modulo (reste de division)
- ****** : Exponentiation

```
# Exemple
a = 10
b = 2

Addition = a + b    # 13
```

2. Opérateurs logiques

Ces **opérateurs** sont utilisés pour effectuer des opérations logiques, souvent dans des structures conditionnelles (comme des instructions if).

Opérateurs logiques :

- **AND** : ET logique
- **OR** : OU logique
- **NOT** : NON logique

```
# Exemple
x = True
y = false

Et_logique = x AND y
```

Écriture et exécution d'un premier programme simple

1. Écriture et exécution d'un premier programme simple

On souhaite écrire un programme simple qui demande à l'utilisateur son nom et son âge, puis affiche un message de salutation.

Exemple de code avec Python :

```
# Demande le nom de l'utilisateur
nom = input("Entrez votre nom: ")

# Demande l'âge de l'utilisateur
age = int(input("Entrez votre âge: "))

# Affiche un message de salutation
print(f"Bonjour {nom}, vous avez {age} ans.")
```

Ici **input()** = Fonction qui lit l'entrée de l'utilisateur sous forme de chaîne de caractères

int() : Fonction qui convertit une chaîne de caractères en entier.

print() : Fonction qui affiche un message à l'écran. Le **f** avant la chaîne permet d'insérer des variables directement dans la chaîne.

Vous pouvez tester rapidement ce code via le lien qui suit en créant un compte au préalable : https://trinket.io/login

Exercice 02 : Écrire un programme

1 - Écrivez un programme pour additionner deux nombres et testez le sur : https://trinket.io/login

2 - Écrivez un programme pour calculer la moyenne de la classe contenant 5 notes et testez le sur : https://trinket.io/login

Rappel : pour calculer une moyenne on additionne les notes entre elles et on les divises par le nombre de note (ici 5).

Structures de contrôle

Structures conditionnelles (if, else)

1. Structures conditionnelles (if, else)

Les structures conditionnelles permettent de **prendre des décisions** dans un programme en exécutant différentes instructions **en fonction des conditions spécifiées.**

```
# Exemple en pseudo-code

si (condition) alors
     exécuter ce bloc de code
sinon
     exécuter ce bloc de code
```

```
# Exemple en python

Temperature = 20
if (temperature > 30):
     print("il fait chaud")
else:
     print("il fait frais")
```

if : Permet d'exécuter un bloc de code si une condition est vraie.

else : Permet d'exécuter un autre bloc de code si la condition est fausse.

else+if = **elif en python** pour mettre une **condition intermédiaire sinon si** (on peut en mettre autant qu'on veut entre un if et un else.

Boucles for et while

1. Boucles

Les boucles permettent de répéter une série d'instructions plusieurs fois.

Pseudo code :

```
# Boucle for : surtout utilisé lorsque le nombre de répétition est définit

pour (initialisation; condition; incrémentation) faire
     exécuter ce bloc de code
```

Pseudo code :

```
# Boucle while : surtout utilisé lorsque le nombre de répétition n'est pas connu à l'avance

tant que (condition) faire
     exécuter ce bloc de code
```

Exercice 03 :
Structure conditionnelle

1 - Écrivez un programme pour vérifier si un nombre rentré par l'utilisateur est positif, négatif ou égal à zéro et afficher un message correspondant : https://trinket.io/login

> Infos supplémentaire pour cet exercice :
>
> **float(input("texte à afficher")) :**
>
> - **float ("5")** : fonction qui permet de convertir une chaîne de texte rentré en nombre à virgule
> - **input("texte")** : fonction qui lit ce que l'utilisateur rentre sous forme de chaîne de caratères
>
> Dans un premier temps écrivez votre programme en version pseudo-code puis en second temps essayez en python avec les infos dont vous disposez sur le support de cours

Exercice 04 : Boucle while

1 - Écrivez un programme pour améliorer le précédent avec une boucle while qui nous permettra de demander un nombre à l'utilisateur tant qu'il le souhaite (avec les mêmes conditions que précédemment) mais si le mot 'stop' est écrit alors cela arrête la boucle : https://trinket.io/login

> Infos supplémentaire pour cet exercice :
>
> **while True:** = **tant que vrai** faire
>
> **.lower()** = **méthode** utilisée en **python** pour convertir tous les caractères d'une chaîne de caractère en minuscule.
>
> **break** = **stop** la boucle
>
> Dans un premier temps écrivez votre programme en version pseudo-code puis en second temps essayez en python avec les infos dont vous disposez sur le support de cours

Fonctions et procédures

Définition et appel de fonction

1. Définition et appel de fonction

Une fonction est un bloc réutilisable qui effectue une tâche spécifique. Elle permet de structurer le code en le divisant en modules plus petit et faciles à gérer.

2. Pourquoi utiliser des fonctions

- Une fonction peut être utilisée plusieurs fois dans un programme.
- Les fonctions rendent le code plus clair et plus facile à comprendre.
- Elles permettent de diviser un programme complexe en morceaux plus simples.

3. Création d'une fonction

Pour créer une fonction, il faut généralement spécifier :

- Le nom de la fonction.
- Les paramètres (ou arguments) que la fonction prend (le cas échéant).
- Le corps de la fonction, c'est-à-dire les instructions à exécuter.
- La valeur de retour (le cas échéant)

Pour utiliser une fonction, il suffit de l'appeler avec les arguments appropriés

```
# Exemple en pseudo-code de création de fonction :
Fonction additionner(a, b)
     retour a + b
```

```
# Exemple en pseudo-code d'appel de fonction :
resultat = additionner(3,5)
afficher resultat // Affiche 8
```

Paramètres et valeur de retour

1. Paramètres

Les paramètres sont des variables qui sont passées à la fonction pour qu'elle puisse les utiliser. Ils sont définis entre parenthèses après le nom de la fonction.

```
# Exemple en pseudo-code
Fonction saluer(nom)
      afficher "Bonjour" + nom
```

Lors de l'appel de la fonction, on doit fournir les arguments correspondants.

```
# Exemple en pseudo-code
saluer("Alice") // affiche "Bonjour Alice"
```

2. Valeur de retour

Une fonction peut renvoyer une valeur après avoir effectué ses opérations. Cette valeur est souvent le résultat d'un calcul ou d'une opération.

```
# Exemple en pseudo-code
Fonction multiplier(x, y)
      retour x * y
```

On peut capturer la valeur envoyée par la fonction et l'utiliser dans notre programme.

```
# Exemple en pseudo-code
produit = multiplier(4, 2)
afficher produit // Affiche 8
```

Exercice 05 : Définir et appeler une fonction simple

En pseudo code :

1 – Définir une fonction nommée "carre" qui prend un argument "x" et retourne le carré de "x« .

2 – Appeler la fonction avec plusieurs valeurs différentes et afficher le résultat.

Exercice 06 : Utiliser des paramètres et des valeurs de retour

En pseudo code :

1 – Définir une **fonction** nommée "**estPair**" qui prend un argument "**n**" et retourne **vrai** si "**n**" est pair, **faux** sinon .

2 – Appeler la fonction avec plusieurs valeurs différentes et afficher le résultat.

Quelques bonnes pratiques

1. Utiliser des noms de fonctions clairs et significatifs :

Les noms des fonctions doivent être descriptifs et indiquer clairement ce que la fonction fait. Cela facilite la compréhension du code par nous-même et par d'autres développeurs.

```
# pseudo-code :
fonction f(x, y)
```

```
# pseudo-code :
fonction calculerSurfaceRectangle(largeur, hauteur)
```

2. Garder les fonctions courtes et focalisées :

Chaque fonction doit avoir une seule responsabilité ou tâche. Si une fonction devient trop longue ou complexe, on peut envisager de la diviser en fonctions plus petites et plus spécialisé quand c'est possible.

```
# pseudo-code :
fonction gererCompteUtilisateur()
     // Gère l'inscription
     // Gère la connexion
     // Gère la mise à jour du profil
     // Gère la suppression du compte
```

```
# pseudo-code :
fonction inscrireUtilisateur()
     // …
fonction connecterUtilisateur()
     // …
fonction mettreAJourProfil()
     // …
fonction supprimerCompteUtilisateur()
     // …
```

Quelques bonnes pratiques

3. Utiliser des paramètres et des valeurs de retour de manière appropriée :

Les fonctions doivent recevoir toutes les données nécessaires via leurs paramètres et retourner les résultats via une valeur de retour. Évitez de passer des variables globales comme arguments.

```
# pseudo-code :
variableGlobal = 10
fonction ajouter(x)
      retour variableGlobal + x
```

```
# pseudo-code :
fonction ajouter(a, b)
    retour a + b
```

4. Mettre des commentaires

Ajoutez des commentaires pour expliquer ce que fait la fonction, ses paramètres et sa valeur de retour. Cela aide à comprendre le code et à l'utiliser correctement.

```
# pseudo-code :
fonction calculerSurfaceRectangle(largeur, hauteur)

// Calcule la surface d'un rectangle
// Paramètres :
//   largeur - La largeur du rectangle
//   hauteur - La hauteur du rectangle
// Retourne :
//   La surface du rectangle (largeur * hauteur)

      retour largeur * hauteur
```

Quelques bonnes pratiques

5. Éviter la duplication de code :

Si on trouve le même code utilisé plusieurs fois dans notre programme, on peut envisager de le regrouper dans une fonction. Cela réduit la duplication et facilite la maintenance.

```
# pseudo-code :
fonction calculerSurfaceRectangle1()
      largeur = 5
      hauteur = 10
      surface = largeur * hauteur
      retour surface

fonction calculerSurfaceRectangle2()
      largeur = 3
      hauteur = 8
      surface = largeur * hauteur
      retour surface
```

```
# pseudo-code :
fonction calculerSurfaceRectangle(largeur, hauteur)
    retour largeur * hauteur
```

En suivant **ces bonnes pratiques**, on peut écrire des fonctions qui sont **claires**, **réutilisables** et **faciles à maintenir**. Cela rendra notre **code plus structuré** et **compréhensible**, facilitant ainsi le développement et la collaboration.

Les tableaux

Introduction aux tableaux

1. Qu'est ce qu'un tableau

Un tableau est une **structure de données** qui permet de **stocker plusieurs valeurs** sous un **même nom**. Chaque valeur est accessible via un **index** (ou **indice**), qui représente la **position de l'élément** dans le tableau. Les tableaux sont très utiles pour **gérer des collections de données** telles que des listes de nombres, de noms, etc.

2. Déclaration d'un tableau

Pour déclarer un tableau, on doit spécifier le type des éléments qu'il contiendra et éventuellement sa taille initiale.

```
# Exemple en pseudo-code :
tableau_notes = [12, 15, 14, 9, 16]
```

3. Accès aux éléments d'un tableau

Les éléments d'un tableau sont accessibles en utilisant leur index. Les index commencent généralement à 0.

```
# Exemple en pseudo-code :
premiere_note = tableau_notes[0]  // Accède à la première note (12)
deuxieme_note = tableau_notes[1]  // Accède à la deuxième note (15)
```

Exercice 07 :

En pseudo code : Calculer la moyenne des notes d'un étudiant en écrivant un programme utilisant un tableau pour stocker les notes :

1 – Déclarer un tableaux pour stocker les notes

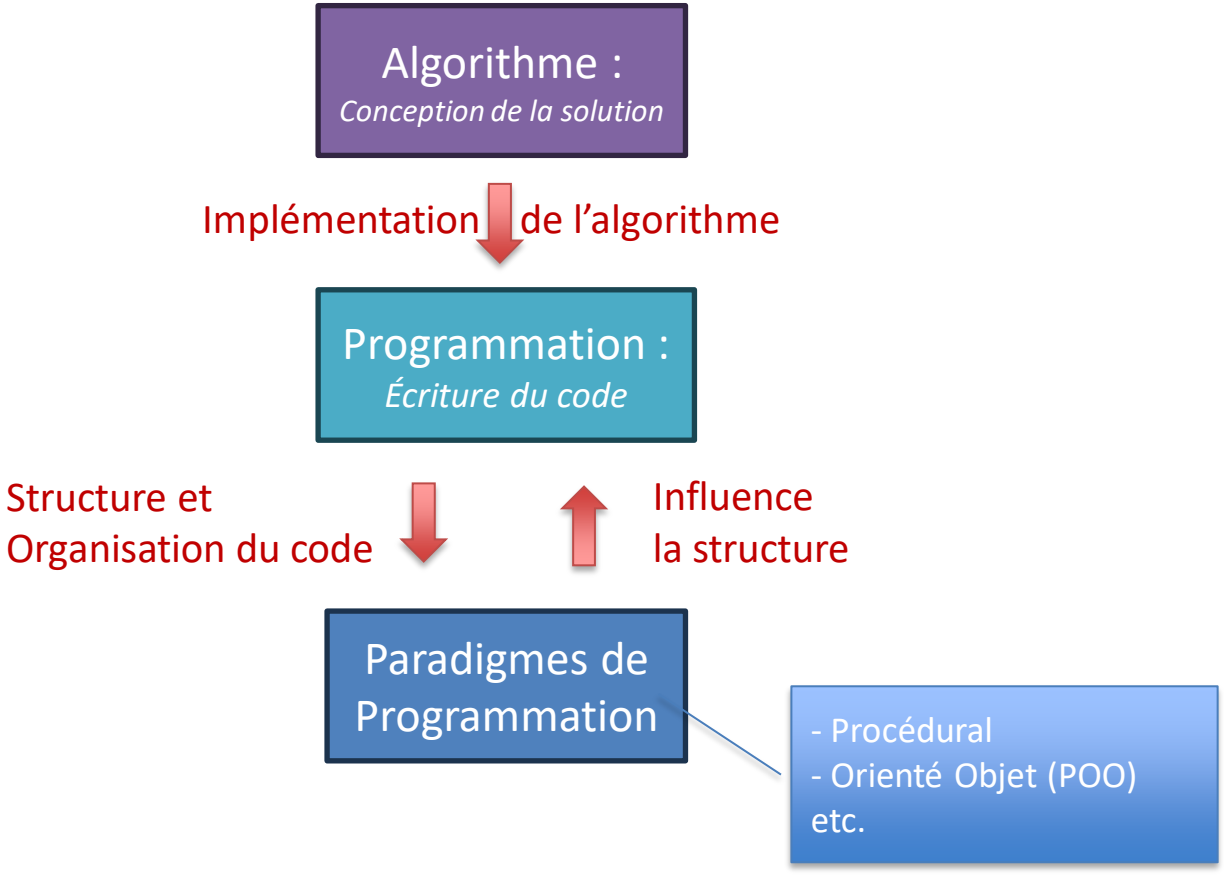
2 – Utiliser une boucle pour calculer la somme des notes

3 – Diviser la somme des notes par le nombre de notes pour obtenir la moyenne

4 – Afficher la moyenne des notes

Introduction aux paradigmes procédurale et Orientée Objet

Paradigme de programmation



Quelques définitions

1. **Qu'est-ce qu'un algorithme ?**

Un algorithme est une **série d'instructions claires** et **précises** pour **résoudre un problème** ou **accomplir une tâche**.

2. **Qu'est-ce que la programmation?**

La programmation est le **processus d'écriture de code** dans un **langage** de programmation pour **implémenter des algorithmes** et créer des logiciels.

3. **Qu'est-ce que les paradigmes de programmation ?**

Les **paradigmes de programmation** sont des **styles** ou des **approches** pour **structurer** et **organiser** le code, influençant la manière dont les solutions sont écrites et implémentées.

Concept de la programmation procédurale

1. Qu'est-ce que la programmation procédurale ?

La programmation procédurale est un **paradigme de programmation** basé sur le concept de la procédure ou fonction. C'est l'une des approches les plus simples et les plus directes pour structurer le code.

2. Caractéristiques principales :

- Le programme est composé de séquences d'instructions exécutées dans un ordre spécifique.
- Les tâches complexes sont décomposées en sous-programmes plus petits et plus gérables appelés fonctions ou procédures.
- Les variables peuvent être définies globalement ou localement à l'intérieur des fonctions.
- Utilise des structures de contrôle comme les boucles (for, while) et les conditionnelles (if, else).

2. Exemple : Imaginons que nous devons calculer la somme de deux nombres et afficher le résultat.

```
# Exemple en pseudo-code : Programme de départ

a = 3
b = 4
somme = a + b
afficher "Somme : " + somme
```

```
# Exemple en pseudo-code : Réorganisé en procédural

fonction calculerSomme(a, b)
      retour a + b

fonction principal()
      a = 3
      b = 4
      somme = calculerSomme(a, b)
      afficher "Somme : " + somme

principal()
```

Concept de la Programmation Orientée Objet (POO)

1. Qu'est-ce que la programmation orientée objet ?

La programmation orientée objet est un **paradigme de programmation** basé sur le concept des **objets**. Un objet est une **instance de classe** qui **encapsule** à la fois les **attributs** et les **méthodes** qui opèrent sur ces données.

2. Caractéristiques principales :

- **Regroupement des données** (attributs) et des comportements (méthodes) au sein des objets.
- **Une classe est un modèle** ou un plan pour créer des objets. Chaque objet est une instance de classe.
- **Les classes peuvent hériter** des attributs et des méthodes d'autres classes, favorisant la réutilisation du code.
- **Les objets** peuvent être **traités** de **manière interchangeable** s'ils partagent la même interface ou la même classe de base.

3. Exemple :

Nous devons modéliser un utilisateur avec son nom et son âge et lui écrire un message de bienvenu personnalisé et afficher des informations :

```
# Exemple en pseudo-code :

classe Personne
      attribut nom
      attribut âge

      méthode dire_bonjour()
            afficher "Bonjour, et bienvenue" + this.nom

      méthode afficher_info()
            afficher "Nom : " + this.nom
            afficher "Âge : " + this.âge + " ans"

fonction principal()
      // Créer un objet Personne
      personne1 = Personne()
      personne1.nom = "Alice"
      personne1.âge = 30

      // Utiliser les méthodes de l'objet
      personne1.dire_bonjour()
      personne1.afficher_info()

principal()
```

Conseils

Conseils de mise en pratique

1. Pratiquer avec des Environnements visuel comme Scratch

Scratch utilise des blocs de code visuels, ce qui rend la logique de programmation plus accessible et ludique.

Scratch offre de nombreux projets exemples que vous pouvez explorer, modifier et améliorer.

Il permet également d'apprendre à diviser un problème complexe en tâches plus petites et gérables.

Vous pourrez créer des jeux simples, des animations ou des histoires interactives.

2. Utiliser des Plateformes de Coding Challenges

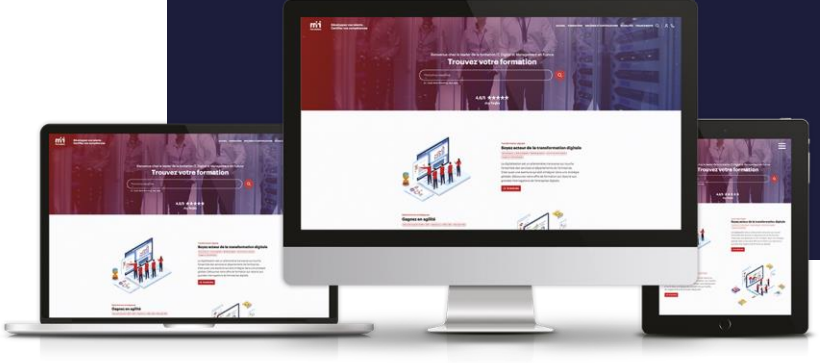
Il existe plusieurs plateformes pour s'entraîner à trouver des solutions pour divers problèmes donnés comme

-> HackerRank
-> CondinGame
-> Scratch

3. Cours en ligne

-> OpenClassroom
-> MaxiCours

Fin du module



m2iformation.fr