Maîtriser les bases de TypeScript

Support pour la session “Maîtriser les bases de Typescript” de Octobre 2026
Formateur : Dufrène Valérian
“Déclaration d’activité enregistrée sous le numéro 32800232680 auprès du préfet de région HAUTS-DE-FRANCE”

Durée totale : 2 jours

Public cible : Débutants

Objectif pédagogique : Permettre aux stagiaires de lire, comprendre et manipuler des scripts de test, des structures de données et des payloads d'API écrits en TypeScript, en abordant les premières bases.

Fondamentaux et Variables

Module 1 : Accueil et Introduction

- **Objectif :** Découvrir l'utilité du code et installer son espace de travail.
- **Explications détaillées :**
    - **Comprendre ce qu'est un programme :** Expliquer simplement qu'un programme est une suite d'instructions logiques données à une machine pour qu'elle exécute une tâche répétitive ou complexe.
    - **Pourquoi TypeScript ?** Montrer qu'il s'agit d'une version "sécurisée" de JavaScript. Contrairement à JavaScript qui pardonne tout (ce qui provoque des bugs cachés), TypeScript vérifie le code en amont et bloque l'exécution si un type de donnée est incorrect.
- **Travaux pratiques guidés :**
    - Installation de l'éditeur de code (Visual Studio Code) et du terminal.
    - Création d'un premier fichier de test nommé **mon-premier-test.ts**.
    - Écriture et exécution d'un message textuel dans la console pour valider que l'environnement fonctionne.

Module 2 : Variables et Constantes

- **Objectif :** Apprendre à stocker des informations de test en mémoire.
- **Explications détaillées :**
    - **La notion de variable :** Imaginer une variable comme une boîte étiquetée dans laquelle on range une information (un mot de passe, un score, un statut).
    - **Oublier var, utiliser let et const :** Expliquer pourquoi l'ancien mot-clé **var** est banni (problèmes de visibilité dans le code). Utiliser systématiquement **let** lorsque la valeur est appelée à changer au cours du test (ex: un compteur de tentatives), et **const** lorsque la valeur doit rester figée (ex: l'adresse URL de l'environnement de recette).
    - **Règles d'écriture :** Apprendre à nommer clairement ses variables (ex: **urlRecette**, **tentativesMax**) en évitant les espaces et les accents.
- **Travaux pratiques guidés :**
    - Écrire un script déclarant une constante pour fixer une URL cible et une variable **let** pour initialiser un statut de test à **true**.
    - Modifier la valeur de la variable **let** au milieu du script et afficher les résultats successifs dans la console.

Module 3 : Types de données et Opérateurs

- **Objectif :** Comprendre comment l'ordinateur classe les informations et réalise des comparaisons logiques.
- **Explications détaillées :**
    - Les types de base indispensables : **string** : du texte entre guillemets (ex: "Connexion réussie").
        - **number** : des chiffres, qu'ils soient entiers ou à virgule (ex: **200**, **3.14**).
        - **boolean** : une valeur binaire, soit vraie (**true**), soit fausse (**false**).
        - **null** et **undefined** : l'absence de valeur ou une valeur non définie (très utile pour tester les champs vides d'un formulaire).
    - **Inférence vs Typage explicite :** Montrer comment TypeScript devine le type tout seul, ou comment l'imposer explicitement (ex: **let codeErreur: number = 404;**).
    - **Les opérateurs de comparaison et logiques :** Comparer des valeurs avec l'égalité stricte (**===**) et la différence (**!==**). Insister sur le fait qu'en TypeScript, on utilise toujours **trois =** pour éviter les erreurs de conversion implicite.
        - Combiner des conditions avec **&&** (ET), **||** (OU), et **!** (NON).
- **Travaux pratiques guidés :**
    - Créer un mini-script de validation de panier d'achat combinant des nombres (montant du panier) et des booléens (client VIP ou non) pour évaluer si une condition de réduction est remplie.

Module 4 : Structures de contrôle conditionnelles

- **Objectif :** Apprendre à un script à réagir différemment selon le résultat d'un test.
- **Explications détaillées :**
    - **L'instruction if / else if / else (Si / Sinon si / Sinon) :** Traduire un arbre de décision métier en code. (Exemple : Si le code HTTP est 200, le test passe ; si c'est 401, l'accès est refusé ; sinon, afficher une erreur inconnue).
    - **Les ternaires :** Raccourcir une condition simple sur une seule ligne grâce au point d'interrogation (**condition ? actionSiVrai : actionSiFaux**).
    - **L'instruction switch :** Remplacer une longue suite de **if / else** par une structure propre lorsqu'on souhaite tester une variable face à de multiples valeurs fixes (ex: les différents états d'une commande de e-commerce).
- **Travaux pratiques guidés :**
    - Écrire un script qui analyse un code de statut de test aléatoire et affiche dynamiquement un message de succès ou d'alerte adapté en utilisant une structure conditionnelle.

Module 5 : Structures de contrôle itératives (Les Boucles)

- **Objectif :** Automatiser des tâches répétitives sans dupliquer le code source.
- **Explications détaillées :**
    - **La boucle for :** Idéale pour répéter une action un nombre précis de fois (ex: rejouer exactement 5 fois la même requête de test de charge). Expliquer l'initialisation, la condition d'arrêt et l'incrémentation.
    - **La boucle while (Tant que) :** Répéter une action en continu jusqu'à ce qu'une condition change (ex: attendre qu'un élément devienne visible à l'écran).
    - **Attention aux boucles infinies :** Mettre en garde les stagiaires contre le piège d'une boucle **while** dont la condition ne devient jamais fausse, ce qui bloque le programme.
- **Travaux pratiques guidés :**
    - Créer une boucle **for** qui simule l'envoi de 5 requêtes successives et affiche le numéro de la tentative en cours dans la console.

Types Avancés, Fonctions et Objets

Module 6 : Types personnalisés, Génériques et Opérateur typeof

- **Objectif :** Structurer proprement des données complexes de test et comprendre la flexibilité avancée de TypeScript.
- **Explications détaillées :**
    - **Les types custom (type et interface) :** Expliquer comment créer ses propres modèles de données sur-mesure. Une **interface** sert de contrat ou de plan pour décrire à quoi doit ressembler un objet (par exemple, un profil utilisateur de test avec un identifiant, un email et un statut).
    - **L'opérateur typeof :** Montrer comment extraire automatiquement le type d'une variable existante sans avoir à le réécrire manuellement, facilitant la maintenance du code de test.
    - **Les types génériques (<T>) :** Expliquer le concept du "moule universel" ou de la boîte intelligente. Un générique permet de créer une structure ou une fonction réutilisable qui s'adapte dynamiquement au type de donnée qu'on lui transmet, garantissant à la fois flexibilité et sécurité stricte.
- **Travaux pratiques guidés :**
    - Créer une interface **UtilisateurTest** décrivant un profil de test.
    - Utiliser **typeof** pour récupérer ce type sur un objet existant.
    - Manipuler une structure générique simple pour accepter différents types de données de test sans perdre le typage.

Module 7 : Fonctions et typage des fonctions

- **Objectif :** Organiser son code en blocs réutilisables et sécuriser les entrées/sorties.
- **Explications détaillées :**
    - **Qu'est-ce qu'une fonction ?** Une fonction est une mini-recette de cuisine : elle prend des ingrédients en entrée (les paramètres), effectue un traitement, et retourne un résultat (le **return**).
    - **Le typage strict des fonctions :** Montrer comment typer chaque paramètre d'entrée en utilisant les interfaces créées précédemment, et comment typer la valeur de sortie (ex: forcer la fonction à retourner obligatoirement un **boolean**).
    - **Les paramètres optionnels et par défaut :** Gérer les cas où un paramètre de test n'est pas toujours obligatoire.
- **Travaux pratiques guidés :**
    - Créer une fonction utilitaire de test appelée **verifierEligibilite** qui prend en paramètre un objet utilisateur typé et retourne **true** ou **false** selon des règles métier strictes.

Module 8 : Les tableaux de données (Arrays)

- **Objectif :** Manipuler des listes de données de test en masse.
- **Explications détaillées :**
    - **Qu'est-ce qu'un tableau typé ?** Une liste ordonnée d'éléments du même type (ex: **string[]** pour une liste d'emails, ou **UtilisateurTest[]** pour une liste de profils).
    - **Accéder et modifier les éléments :** Expliquer que l'on compte toujours à partir de **0** en programmation. Comment récupérer le premier élément d'une liste ou modifier un élément précis via son index.
    - **Parcourir un tableau :** Apprendre à itérer sur chaque élément de la liste pour appliquer un contrôle automatisé.
- **Travaux pratiques guidés :**
    - Déclarer un tableau typé d'identifiants de test et écrire un script qui parcourt ce tableau pour afficher et vérifier chaque élément un par un.

Module 9 : Introduction à la programmation Objet et au format JSON

- **Objectif :** Comprendre la structure des objets en mémoire et décoder les formats de données des API.
- **Explications détaillées :**
    - Qu'est-ce qu'un objet ? Une structure regroupant plusieurs informations sous forme de paires "clé-valeur" (ex: la clé **nom** a pour valeur **"Dupont"**). Comment accéder aux propriétés d'un objet en utilisant la notation par point (**objet.propriete**).
    - **Le format JSON (*JavaScript Object Notation*) :** Expliquer que le JSON est le standard universel du web. C'est exactement le format des réponses d'API que les testeurs reçoivent dans leurs outils (Postman, Cypress, Playwright).
- **Travaux pratiques guidés :**
    - Intégrer un faux payload de réponse d'API au format JSON dans un script TypeScript.
    - Typer ce payload à l'aide d'une interface sur-mesure.
    - Écrire une petite fonction de vérification qui lit les propriétés de cet objet JSON pour valider un scénario de test.