Introduction au Développement Logiciel

Exercices Dirigés

Gerson Sunyé
gerson.sunye@univ-nantes.fr

Gilles Ardourel
gilles.ardourel@univ-nantes.fr

Erwan Bousse
erwan.bousse@univ-nantes.fr

Nantes Université
22 mars 2024

Table des matières

1

Le langage TypeScript

Aide mémoire : syntaxe et fonctionnalités

Le langage TypeScript est un sur-ensemble du langage JavaScript découvert au premier semestre, ajoutant de nouvelles fonctionnalités et syntaxes. La documentation complète du langage est disponible sur son site dédié[1].

Les variables

Déclaration

```
let x : Type = Valeur;
/* Exemples */
let a : number = 32;
let b : number = -3.14159;
let c : boolean = false;
```

Les Types

```
number  //entiers et reels
string  //chaine de caractere
boolean //booleen
```

Les constantes

Déclaration

```
const x : Type = Valeur;
/* Exemples */
const a : number = 32;
const b : number = -3.14159;
const c : boolean = false;
```

[1] https://www.typescriptlang.org/docs/home.html

Les opérations

Affectation de variables

```
let x : number = 6;
let y : number = 5;
x = 24 / 12;
y = x + y; // y = 7
```

Opérateurs mathématiques et logiques

```
+ - * / // operateurs usuels
x % 3   // x modulo 3
x ** 3  // $x^3$
/* Raccourcis */
x += 2     // x = x + 2
x *= 3     // x = x * 3
x++        // x = x + 1

/* Operateurs de logique */
y == 5  // false (y!=5)
y >= 5   // true
true && (y<6) //false (et
    logique)
true || (y<6) //true (ou
    logique)
```

Les conditions

Bloc conditionnel simple

```
if(condition) {
    //operations
}
```

Conditions multiples

```
if(condition1){
    // cas 1
} else if(condition2) {
    // cas 2
} else {
   //cas par defaut
}
```

Les boucles

Pour x allant de 1 à 8

```
for(x=1; x<=8; x=x+1){
  //operations
}
```

Tant que ... faire

```
while(condition){
  //operations
}
```

Répéter ... tant que

```
do {
  //operations
} while(condition);
```

Les chaînes de caractères

```
let name : string = 'Dupont'; //delimitation par " ou '
let length : number = name.length;// longueur = 6
```

```
let c : string = name[2];// c == "p", on commence a 0
let fullName : string = firstName + ' Jean'; // concatenation
```

Les tableaux

Création et initialisation

```
let tab1 : Array<number> = [1,2,3];
let tab2 : number[]      = [1,2,3];
```

Opérations

```
tab1[1]; // 2
tab1.length; // 3
tab1.push(5); // tab1
    =[1,2,3,5]
```

Les enregistrements

Déclaration

```
type Point = {
    x : number
    y : number
}
```

Usage

```
let a : Point = {x : 1, y : 4};
a.x;  // 1
a.y;  // 4
```

Les fonctions

Déclaration d'une fonction de deux paramètres

```
function nomFonction(param1 : Type1, param2 : Type2) : TypeRetour {
    //operations
    return valeurRetour;
}
```

Utilisation

```
x = nomFonction(param1, param2);
```

Une procédure (sans valeur de retour) possèdera le TypeRetour égal à void

Commentaires

```
a=2; //commentaire de ligne

/* commentaire
   de
   bloc
*/
```

Affichage

```
1 console.log(expression);
```

Modules

Export de fonctionnalités (mesOutils.ts)

```
1 /* export de fonction */
2 export maFonction() : Type { ... };
3
4 /* export de variable */
5 export pi : number = 3.14159;
```

Import de fonctionnalités (depuis un autre fichier)

```
1 import {pi} from "mesOutils";
2 let x : number = pi/4;
```

1.1 De AlgoScript à TypeScript

L'objectif de ce chapitre est d'introduire la syntaxe du langage TypeScript à partir de celle du langage AlgoScript, présentée antérieurement. Pour cela, nous allons d'abord rappeler l'algorithme de détection d'une année bissextile. Rappelons d'abord qu'une année est bissextile si (et seulement si) elle est :

- soit divisible par 4 mais pas par 100 ;
- soit divisible par 400.

Exercice 1 *Algorithme*

Donner un algorithme qui affiche "Bissextile" ou "non Bissextile" en fonction d'une année saisie en entrée.

Exercice 2 *Types de variables*

Quels sont les variables utilisées dans cet algorithme ? Quels sont les types de données de ces variables ? Quels sont les mot-clés correspondant à ces types en TypeScript ?

Exercice 3 *Passage à TypeScript*

Écrivez le code TypeScript correspondant à cet algorithme.

1.2 Fonctions

Une fonction est un sous-programme qui effectue une tâche ou un calcul et qui permet de renvoyer un résultat. Dans la plupart des cas, elle contient des paramètres d'entrée, qui sont des informations qui seront récupérées par la fonction et qui serviront à l'exécution de la tâche ou du calcul.

Dans cette section, nous allons transformer l'algorithme de détection d'années bissextiles (développé dans l'exercice précédant) en une fonction, que nous allons ensuite écrire en TypeScript.

Exercice 4 *Paramètres de la fonction*

Quelles informations doivent être transmises à la fonction ? Quels sont leurs types ? Quel doit être le type du résultat de la fonction ?

Exercice 5 *Signature de la fonction*

Donnez la signature de la fonction isLeapYear()en langage TypeScript.

Exercice 6 *Mise en oeuvre de la fonction*

Basez-vous sur le code écrit précédemment pour écrire la fonction isLeapYear ().

1.3 Utilisation de fonctions

Exercice 7 *Utilisation de la fonction isLeapYear()*

Écrivez un exemple de code TypeScript appelant la fonction isLeapYear() pour une année donnée et affichant le résultat.

Exercice 8 *Signature de la fonction*

Proposez la signature d'une fonction capable de calculer le nombre de jours d'un mois à partir d'un mois et d'une année donnés.

Exercice 9 *La fonction daysInMonth()*

Utilisez le langage TypeScript pour écrire le code de la fonction daysInMonth ().

1.4 Des fonctions complémentaires

Maintenant, nous allons écrire une fonction capable de déterminer le jour de la semaine correspondant à une date donnée. Pour cela, nous allons utiliser l'algorithme de Delambre[2].

Cet algorithme a besoin d'au moins deux tableau contenant le nombre de jours de décalage de chaque mois, un pour les années bissextiles et un autre pour les années non-bissextiles. Voici le tableau des décalages pours les mois des années non-bissextiles :

Mois	1	2	3	4	5	6	7	8	9	10	11	12
Décalage	4	0	0	3	5	1	3	6	2	4	0	2

Voici maintenant le tableau des décalages pour les mois des années bissextiles :

Mois	1	2	3	4	5	6	7	8	9	10	11	12
Décalage	3	6	0	3	5	1	3	6	2	4	0	2

Enfin, pour simplifier la conversion du résultat, un chiffre entre 0 et 6, nous allons créer un tableau avec les jours de la semaine, où "Dimanche" est à l'index 0 et "Samedi" à l'index 6 :

[2] `https://fr.wikipedia.org/wiki/D%C3%A9termination_du_jour_de_la_semaine`

0	1	2	3	4	5	6
'Dimanche'	'Lundi'	'Mardi'	'Mercredi'	'Jeudi'	'Vendredi'	'Samedi'

Exercice 10 *Tableaux*

Créez en TypeScript les tableaux ci-dessus afin de stocker le nom des différents jours et la valeur de chaque mois. Faites attention à la numérotation des mois et des indices de tableau.

Exercice 11

Donnez les deux formules permettant à partir d'une année exprimée en entier (YYYY) de la décomposer sous la forme de deux entiers hundreds et centuryYear. Par exemple, l'année 1963 sera décomposée en hundreds = 19 et centuryYear = 63. On utilisera en TypeScript la fonction Math.trunc() pour récupérer la partie entière d'un nombre réel et l'opérateur % pour récupérer le reste d'une division entière.

Algorithme de Delambre

Pour déterminer le jour de la semaine à partir d'une date donnée, nous utilisons la formule suivante où AA est la centaine d'une date (hundreds), BB l'année dans le siècle (centuryYear), $E()$ est la fonction donnant la partie entière d'un nombre, J le jour du mois et $offset$ la valeur de décalage du mois correspondant aux tableaux définis précédemment :

$$\left[E\left(\frac{AA}{4}\right) + E\left(\frac{BB}{4}\right) + BB + 5AA + offset + J + 2\right] mod\, 7$$

Exercice 12 *Signature de la fonction*

Proposez la signature d'une fonction retournant le nom du jour de la semaine à partir d'une date.

Exercice 13 *La fonction dayOfWeek()*

Écrivez maintenant le code de la fonction dayOfWeek().

1.5 Enregistrements et tableaux

Les tableaux et les enregistrements sont des structures de données. Un tableau représente une séquence finie d'éléments de même type, auxquels on peut accéder efficacement par leur position. Un enregistrement, quant à lui, est composé d'un ensemble de valeurs potentiellement de types différents.

Le nombre de champs (ou d'éléments) d'un enregistrement est fixe, tandis que le nombre d'éléments d'un tableau peut varier de zéro à sa taille maximale. En outre, les champs d'un enregistrement sont dépendants les uns des autres, tandis que les éléments d'un tableau sont indépendants.

Exercice 14 *Un tableau de nombres*

Écrivez le code TypeScript quii :

1. déclare une variable de type tableau d'entiers ;
2. ajoute les valeurs 1, 2 et 3 à ce tableau ;
3. affiche le 2e élément.

Exercice 15 *La fonction average()*

Écrivez, en TypeScript, une fonction qui prend en paramètre un tableau de nombres et qui calcule et rend un nombre représentant la moyenne des valeurs de ce tableau.

Exercice 16 *L'enregistrement Date*

Écrivez un enregistrement capable de représenter des dates, c'est à dire, contenant trois composants : jour, mois et année.

Exercice 17 *Utilisation des enregistrements*

Réécrivez la fonction dayOfWeek(), pour la permettre d'accepter comme paramètre l'enregistrement Date.

1.6 Modules

Le modules de TypeScript permettent l'organisation des programmes trop longs. Leur objectif est d'organiser le code source dans différents fichiers, appelés "modules" et de les importer au besoin. En TypeScript, n'importe quel fichier source contenant les mots-clés import ou export est considéré comme un module.

Exercice 18

Placez le type Date et les fonctions en relation avec les dates dans un fichier appelé `date.ts`. Dans ce fichier, exportez seulement la fonction dayOfWeek. Créez ensuite un fichier appelé `main.ts` et importez et utilisez cette même fonction dayOfWeek dans ce nouveau fichier. Votre code compilera correctement ? Si non, expliquez pourquoi.

2

Qualité de code

2.1 Conventions de codage

Exercice 19 *Styles de nommage*

Complétez le tableau ci-dessous en appliquant les différents styles de nommage aux identifiants.

	name	first name	screaming lazy dog
Pascal			
Camel			
Snake			
Kebab			
Screaming Snake			

2.2 Simplification de code

Exercice 20

Simplifiez le plus possible le code `TypeScript` suivant :

```
1     if (a == true) {
2         return true;
3     } else {
4         return false;
5     }
```

Exercice 21

Simplifiez le code `TypeScript` suivant :

```
1 if(lastPlayer) {
2     lastPlayer = false;
```

```
} else {
    lastPlayer = true;
  }
```

Exercice 22

Simplifiez la fonction `TypeScript` suivant :

```
function isDivisibleBy(m: number, n: number): boolean {
    let result : boolean;

    if (m % n == 0) {
        result = true;
    } else {
        result = false;
    }
    return result;
}
```

Exercice 23

Considérez la fonction TypeScript suivante, qui calcule le produit entre deux nombres selon la technique dite russe.

```
function m(a: number, b: number): number {
    let result : number = 0;
    while(a > 0) {
        if (a % 2 == 1) {
            result += b;
        }
      a = Math.floor(a / 2);
      b = 2 * b;
    }
    return result;
}
```

1. Quels changements pourraient améliorer la lisibilité de la fonction `m()` ?
2. Quels commentaires pourraient améliorer la lisibilité de la fonction `m()` ?

Exercice 24

Simplifiez le plus possible le code `TypeScript` suivant :

```
    if (a > 12) {
      console.log('use this number in the following step: ' + b % 2)
          ;
    } else {
        if (a <= 12) {
```

```
5             console.log('use this number in the following step: ' + b
                  * 2);
6         }
7     }
```

Exercice 25

Considérez le code TypeScript suivant ://

```
1  /**
2   * Enregistrement decrivant un personnage.
3   */
4  type CHAR = {
5      Active  : Boolean
6      Strength  : number
7  }
8
9  /**
10  * La fonction F decide le vainqueur d'un combat entre deux
        Personnages.
11  * @param p1 Le defenseur.
12  * @param p2 L'attaquant.
13  * @param l Le lieu du combat
14  */
15 function F(p1 :CHAR, p2 :CHAR, l : Boolean) : CHAR {
16     let retVal  : CHAR
17     if (p1.Active == true && p2.Active == true) {
18         if (p1.Strength >= p2.Strength) {
19             retVal = p1;
20         } else {
21             retVal = p2;
22         }
23         return retVal;
24     } else if (p1.Active == false) {
25         return p2;
26     } else
27     return p1;
28 }
```

1. Réécrivez l'enregistrement (type) `CHAR`, en respectant les conventions de codage TypeScript et en améliorant la lisibilité du code selon les principes vus en cours.

2. Réécrivez ensuite la fonction `F()`, en respectant les conventions de codage TypeScript et en améliorant la lisibilité du code selon les principes vus en cours.

2.3 Nommage

Exercice 26

Supposez que vous avez besoin d'une variable de type entier qui représente le nombre de jours dans un mois. Proposez un identifiant adapté à TypeScript.

Exercice 27

Proposez les signatures des fonctions TypeScript dont les comportements sont les suivants :

- calcule la distance entre deux points ;
- convertit des mètres en pieds ;
- renvoie le plus grand nombre parmi ceux d'un tableau.

2.4 Commentaires

Exercice 28

Supprimez les commentaires inutiles du code suivant :

```
1  // Reverses a given string.
2  // For instance, the string "software" becomes "erawtfos".
3  function reverse(s: string): string {
4      // Variable declaration
5      let r = ""; // creates an empty string
6
7      for (let i = s.length - 1; i >= 0; i--) { // Loops from the end
           the string until the begining
8          r += s[i];
9      }
10
11     return r;
12 }
```

2.5 Lisibilité

Exercice 29

Considérez le code `TypeScript` ci-dessous. Quels sont les problèmes de lisibilité de la fonction fcn ?

```
1  function fcn(a :number, b :number) :number {
2  let x : number;
3  let i : number;
4  x = 0;
5  if (Math.cos(a) < Math.cos(b)) {
```

```
6         for(i = 0; i < 360; i++){
7                 x = x + Math.sqrt(i);
8         }
9   }
10  return x;
11  }
```

Exercice 30

Considérez le code suivant. Quel commentaire ajouteriez-vous à la ligne 2 ?

```
1  function m(array : Array<number>) :number {
2   let m: number = 0;
3   for(let each of array) {
4       m = m > each ? m : each;
5   }
6   return m;
7  }
```

2.6 Découpage de code

On considère le code du Listing 2.1 qui permet de comparer des listes de nombres. Ce code distingue trois cas : soit la première liste est une sous-liste de la seconde, et la fonction renvoie 1 ; soit les deux listes sont égales et la fonction renvoie 0 ; soit la fonction renvoie -1 dans les cas restants.

Une liste $l1$ est une sous liste de $l2$ si elle possède strictement moins d'éléments que $l2$, et si on peut trouver les mêmes éléments que $l1$ à l'intérieur de $l2$, dans le même ordre. Par exemple, `[7,8]` est sous liste de `[1,2,7,8,3,4]`.

On notera que l'opérateur `t.slice(x,y)` permet d'extraire une séquence d'éléments d'un tableau `t`. Cette séquence commence à l'index `x` et se termine à l'index `y` exclu. Par exemple, `[1,2,3,4].slice(1,3)` produit la séquence `[2,3]`.

Exercice 31

Renommez la fonction et les variables pour améliorer la lisibilité du code, et mettez les expressions pertinentes sous la forme de variables explicatives

Exercice 32

Identifiez une manière intéressante de découper ce code en trois fonctions distinctes—dont une isofonctionnelle avec la fonction `comp` donnée— de manière à améliorer la lisibilité du code et limiter les redondances.

Donner les signatures des deux nouvelles fonctions.

Exercice 33

Réécrivez ce code sous la forme de trois fonctions.

```
function comp(l1 : Array<number>, l2 : Array<number>) :number {

    if (l1.length != l2.length) {

        if (l1.length < l2.length) {

            for (let i = 0; i < l2.length; i += 1) {
                let sub: boolean = true
                if (l1.length != l2.length) {
                    for (let j = 0; j < l1.length && sub; j += 1) {
                        if (l1[j] != l2.slice(i, i + l1.length)[j])
                            {
                            sub = false
                        }
                    }
                }
                if (sub) {
                    return 1
                }
            }

        }
        return -1

    } else {
        for (let i = 0; i < l1.length; i += 1) {
            if (l1[i] != l2[i]) {
                return -1
            }
        }
        return 0
    }
}
```

Listing 2.1 : Fonction à découper.

3

Spécification, Vérification, Validation

3.1 Problématique

> Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots.
> So far, the Universe is winning.
>
> *Rick Cook, The Wizardry Compiled (1989)*

Exercice 34

Au-delà de son aspect humoristique, il y a un fond de vérité dans cette citation : quelles problématiques du développement logiciel sont évoquées ?

Exercice 35

Cependant, il ne faut bien sûr pas y voir une véritable définition : son manichéisme un peu facile peut occulter d'autres facettes et surtout d'autres responsabilités. Lesquelles ?

3.2 Dates, suite

Contexte : un formulaire demandant à un utilisateur sa date de naissance.

Exercice 36

Ecrire un enregistrement (type) capable de représenter des dates de naissance, déclarer une variable de ce type et lui affecter la date du 1 octobre 2002.

Exercice 37

Qu'est-ce qu'une date correcte ? En quoi le contexte spécifique d'une date de naissance d'utilisateur modifie ces contraintes ?

Exercice 38

Ecrire la ou les fonctions de validation pour le type MyDate

Exercice 39

Quel type utiliser pour représenter l'information en gras dans la phrase suivante ?
"Dans **2 ans 5 mois et 10 jours** j'aurai fini ma Licence."

Exercice 40

Dans le cadre d'un formulaire, serait-il préférable d'utiliser un widget Calendrier, des champs texte, ou les deux ? Argumenter.

3.3 What's in a name ?

On considère maintenant le champ Nom de l'utilisateur.

Exercice 41

Comment déterminer qu'une chaine de caractères est potentiellement un nom valide ?

Exercice 42

Donner la signature d'une fonction vérifiant la validité d'un nom.

Exercice 43

Quels sont les avantages et inconvénients d'effectuer la vérification à chaque nouveau caractère entré par l'utilisateur ?

Bien qu'elles soient hors du champs de ce module, il ne faut pas négliger les contraintes juridiques concernant les données enregistrées, ni les éventuels problèmes de sécurité (à titre d'exemple : `https://www.explainxkcd.com/wiki/index.php/327:_Exploits_of_a_Mom`).

4

Tests unitaires

4.1 Mémento sur Alsatian

Intitulé	Opération TypeScript	Alsatian
égalité	$a == b$	Expect(a).toBe(b)
inégalité	$a! = b$	Expect(a).not.toBe(b)
égalité en profondeur		Expect(a).toEqual(b)
inégalité en profondeur		Expect(a).not.toEqual(b)
Expression régulière		Expect(message).toMatch(/bar/)
null	$a == null$	Expect(a).toBeNull()
non null	$foo! = null$	Expect(foo).not.toBeNull()
Contient ou sous-chaîne		Expect(a).toContain("bar")
Inférieur	$e < pi$	Expect(e).toBeLessThan(pi)
Supérieur	$pi > e$	Expect(pi).toBeGreaterThan(e)

4.2 PGCD et PPCM

Considérez les deux signatures fonction suivantes :

```
1  /**
2   * Calculates the  Greatest Common Divisor (GCD) for two numbers.
3   * The GCD of two or more integers, which are not all zero, is the
4   * largest positive integer that divides each of the integers.
5   * For example, the gcd of 8 and 12 is 4.
6   *
7   * @returns the GCD for two positive integers, or -1 for
8   * invalid inputs.
9   *
10  **/
11 function gcd(a :number, b :number) : number;
12
13 /**
14  * Calculates the Least Common Multiple (LCM) for two numbers.
15  * In arithmetic and number theory, the least common multiple,
```

```
16  * lowest common multiple, or smallest common multiple of two
17  * integers a and b, usually denoted by LCM(a, b), is the smallest
18  * positive integer that is divisible by both a and b.
19  *
20  * Since division of integers by zero is undefined, this definition
21  * has meaning only if a and b are both different from zero.
22  *
23  * @returns the LCM for two positive non-zero integers, or -1 for
24  * invalid inputs.
25  *
26  **/
27 function lcm(a: number, b: number): number;
```

Exercice 44

Pour chacune des signatures de fonctions précédentes, proposez les données de test, ainsi que la réponse attendue, permettant de vérifier les cas suivants :

1. Valeur invalide du paramètre `a` ;
2. Valeur invalide du paramètre `b` ;
3. Deux valeurs paires ;
4. Deux valeurs impaires.

Exercice 45

Utilisez vos données de test pour écrire deux suites de test pour les fonctions `gcd` et `lcm`. Pour simplifier l'écriture des tests ainsi que leur exécution, utilisez Alsatian. Basez-vous sur la syntaxe de l'exemple ci-dessous :

```
1  import { Expect, Test } from "alsatian";
2  import { gcd } from "../src/gcd"
3
4  export class GCDTests {
5
6      @Test("Simple example") testGCD1() {
7          Expect(gcd(10, 10)).toBe(10);
8      }
9
10     @Test("Another example") testGCD2() {
11         Expect(gcd(1, 1)).toBe(1);
12     }
13 }
```

4.3 Décomposition d'un entier en nombres premiers

Exercice 46

La fonction factorize() met en oeuvre l'algorithme rho de Pollard, qui est un algorithme de décomposition en produit de facteurs premiers spécifique qui est seulement effectif pour factoriser les entiers naturels avec de petits facteurs. Il fut conçu par John M. Pollard (en) en 1975 :

```ts
import { gcd } from "./gcd"

export function factorize(value: number): number {
    // https://en.wikipedia.org/wiki/Pollard's_rho_algorithm
    if (value <= 0) { return -1; }
    if (value % 2 == 0) { return 2; }
    if (value % 3 == 0) { return 3; }
    if (value % 5 == 0) { return 5; }

    let x = 2;
    let y = 2;
    let factor = 1;

    while (factor == 1) {
        x = g(x, value);
        y = g(g(y, value), value);
        factor = gcd(Math.abs(x - y), value);
    }

    return factor;
}

function g(x: number, value: number): number {
    return (x * x + 1) % value;
}
```

factorize.ts

1. Proposez des données de test permettant de vérifier la fonction factorize();
2. Utilisez ces données pour écrire une suite de test unitaires capables de vérifier la fonction factorize().

Exercice 47

La fonction calculatePrimeFactors() décompose un nombre en plusieurs facteurs premiers. Elle doit retourner un tableau vide ([]) si le nombre est inférieur ou égal à 1. Elle utilise la fonction factorize() présentée ci-dessous.

```ts
import { factorize } from "./factorize"

/**
 * Decomposes an integer number in a sequence of prime numbers.
 *
 * @param value the number to be decomposed
 */
export function calculatePrimeFactors(value: number): number[] {
    // https://en.wikipedia.org/wiki/Prime_factor
    if (value == 1) { return []; }

    let factor = factorize(value);
    const tree = isPrime(factor, value) ? [factor] : [factor, value / factor];
    let index = 0;
    const result = [];

    while (index < tree.length) {
        const current = tree[index];
        factor = factorize(current);

        if (isPrime(current, factor)) {
            result.push(factor);
        } else {
            tree.push(factor, current / factor);
        }

        index++;
    }

    return result.sort((a, b) => a - b);
}

function isPrime(value: number, factor: number): boolean {
    return factor == value;
}
```

prime-factors.ts

1. Proposez des données de test permettant de vérifier la fonction calculatePrimeFactors();
2. Utilisez ces données pour écrire une suite de test unitaires capable de vérifier la fonction calculatePrimeFactors().

4.4 Dates

Exercice 48

La fonction age() calcule l'âge d'une personne (nombre d'années) cette année. La fonction renvoie -1 si la date est erronée.

```ts
export function age(day: number, month: number, year: number): number {
    if (year < 1 || month > 12 || day > 31) { return -1; }

    const today: Date = new Date();

    let age: number;

    if (year > today.getFullYear()) {
        return -1;
    }

    if (month > today.getMonth() || (month == today.getMonth() &&
        day > today.getDay())) {
        age = today.getFullYear() - year - 1;
    } else {
        age = today.getFullYear() - year;
    }
    return age;
}
```

age.ts

1. Proposez des données de test permettant de vérifier le bon fonctionnement de cette fonction.

2. Utilisez ces données pour écrire une suite de test unitaires capable de vérifier cette fonction.

Exercice 49

La fonction durationInMinutes() calculer) la durée écoulée en minutes entre deux horaires au cours de la même journée :

```ts
/**
 * Calculates the number of minutes between two Times.
 * @param begin The begining time
 * @param end The end time
 */
export function durationInMinutes(begin: Time, end: Time): number {

    return (end.hour - begin.hour) * 60 - begin.minute + end.minute;
}

export type Time = {
    hour: number;
    minute: number;
};
```

duration.ts

1. Proposez des données de test permettant de vérifier le bon fonctionnement de cette fonction.
2. Utilisez ces données pour écrire une suite de test unitaires capable de vérifier cette fonction.

5

Modules et Enregistrements

L'objectif de ce TD est de vous familiariser avec le travail en groupe sur un projet informatique. La réalisation du projet se déroulera en plusieurs étapes :

1. Modélisation et spécification du problème.
2. Programmation et test du projet.
3. Réflexion sur les pistes d'amélioration.

5.1 Jeu d'échecs

Le but est de créer un jeu d'échecs basique (cf. Fig 5.1). Il s'agit en fait d'un lecteur de parties d'échecs.

Un **pièce** d'échecs se définit par son type : Pion (Pa,...,Ph), Tour (Ta,Th), Cavalier (Cb,Cg), Fou (Fc,Ff), Dame (D) et Roi (R). Une pièce a également une couleur (Noir et Blanc) et un code html permet de l'afficher en Unicode.

Nous supposons que nous disposons d'une liste de **coups** joués par 2 joueurs. Une série coup s'écrit de la façon suivante :

FIG. 5.1 : Un échiquier

```
1 TaA4, RB1, CbH4, ...
```

cela veut dire que la Tour (initialement en colonne a) sera déplacée sur la case A4 et ensuite le roi passe à la case B1, etc.

Un **échiquier** consiste en une grille de pièces.

La **partie** commence par une grille initiale et consiste à jouer tous les coups dans l'ordre. L'échiquier doit être mis à jour à chaque coup.

5.2 Organisation

Placer les tables de façon à travailler par groupe de 4 étudiants. La partie modélisation devra être faite par concertation. Le travail devra ensuite être partagé par binôme.

Exercice 50 *Modélisation d'un jeu d'échecs*

1. Dessiner un diagramme qui modélise le jeu d'échecs décrit plus haut. Ce diagramme doit faire apparaître les différents modules (enregistrements et fonctions) et les interactions entre les fonctions de différents modules.
2. Créer les fichiers qui vont accueillir les différents modules.
3. Créer les enregistrements nécessaires au jeu.
4. Donner les signatures des fonctions du jeu.

Exercice 51 *Programmation d'un jeu d'échecs*

1. Écrire le code des fonctions du premier exercice.
2. Écrire quelques tests pour chaque fonction.

Exercice 52 *Pistes d'amélioration*

La solution à laquelle nous avons abouti ne prend pas en compte les obstacles durant les déplacement ni les prises de pièces. On peut pas non plus savoir quand une partie est finie ou bien quel joueur a gagné.

1. Proposer une modélisation (enregistrement et fonction) pour vérifier la validité d'un coup (absence d'obstacles) et la prise de pièces par un autre joueur.
2. Comment on pourra savoir si une partie est finie et qui a gagné ?

6

Développement Web

Avertissement

Ces 2 séances n'ont pas pour vocation de vous apprendre à écrire une page HTML dans le cas général, encore moins un site entier. Elles vous présenteront le minimum nécessaire pour comprendre le travail réalisé pendant les séances de travaux pratiques. Vous trouverez plus d'informations sur le standard HTML, le CSS et le Javascript sur le site du *World Wide Web Consortium*[1] ou celui de la *Fondation Mozilla*[2]. Le code présenté dans ce document respecte la norme HTML-5.

6.1 Introduction

Une page HTML est un fichier texte (d'extension `.html` ou `.htm`) comportant le contenu texte que le navigateur (ou toute autre application interprétant du HTML) doit afficher à l'utilisateur (nommé ici ensuite l'internaute), accompagné en particulier d'informations sur la mise en forme et la mise en page. Les informations au sujet du contenu sont distinguées du contenu par une syntaxe de balises :

- une balise est encadrée par les symboles "<" et ">" (inférieur et supérieur) ; par exemple <**html**> ;
- une balise peut être ouvrante (<**html**>), fermante (</**html**>) ou unique (c'est à dire auto-fermante comme <**html**/>) ;
- les balises non uniques vont par paires (une ouvrante — une fermante) comme des parenthèses.

[1] `https://www.w3schools.com`
[2] `https://developer.mozilla.org/fr/`

Une page html possède une structure (entête, tête et corps), le contenu visible de l'internaute étant encadré par <body> et </body> et décomposé en divisions et en paragraphes. Voici par exemple ci-dessous à gauche le code HTML d'une page et à droite la façon probable dont un navigateur la présente à l'internaute (2 images en fonction de la largeur de la fenêtre).

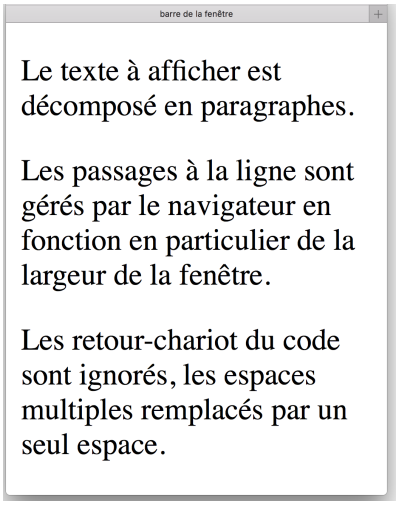
```
<!doctype html>
<html>        <!-- commentaire HTML -->
  <head>
    <title> barre de la fenêtre </title>
    <meta charset="UTF-8" />
    <!-- remplacer eventuellement UTF-8 par ISO-8859-1 -->
  </head>
  <body>
    <!-- c'est ici que le contenu visible est ajoute  -->
    <div> <!-- une division formee de paragraphes -->
      <p>   Le texte a afficher est <span> decompose </span>
            en paragraphes.
      </p>
      <p>  Les passages a la ligne
           sont geres par le navigateur
           en fonction en particulier
           de la largeur de la fenêtre.
       </p>
    </div>
    <div> <!-- une autre division -->
            Les retour-chariot du code
            sont ignores,
            les espaces multiples
            remplaces par un seul espace.
      </div>
   </body>
</html>
```

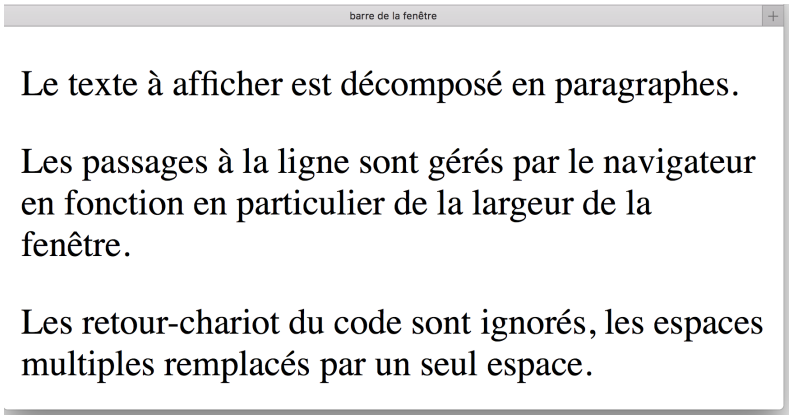
Listing 6.1 : Fichier document.html

6.2 Attributs

Une balise ouvrante ou unique peut posséder des attributs, c'est à dire des informations associées à la balise elle-même ou à son contenu compris entre l'ouvrante et la fermante. Chaque attribut possède une valeur ; les couples "(nom, valeur)" s'écrivent sous la forme nom="valeur", avec des apostrophes ou des guillemets. Par exemple :

- permet de faire afficher le fichier "chat.jpeg" dans une zone d'image.
- GPS transforme le texte "GPS" en un lien vers le site `http://perdu.com`.





- <**p style**=“**color** :red ;”>pompiers </**p**> permet de faire écrire le mot“pompiers” en rouge.

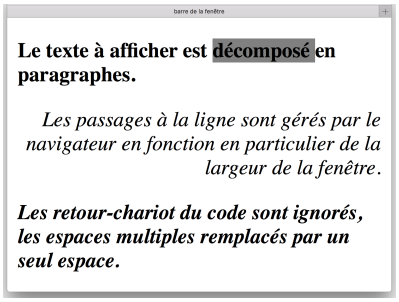
Une balise ne peut avoir qu'un seul exemplaire d'un attribut donné. Mais un attribut peut avoir un ensemble de valeurs. Deux attributs possèdent un intérêt particulier : class et id. Un identificateur (id) doit avoir une valeur unique dans la page. L'attribut class peut lui avoir plusieurs valeurs (séparateur : espace) : class=”auteurs nomPropre”.

Exercice 53

Que faut-il modifier dans le code fourni ci-dessus pour que le troisième paragraphe de texte (de *Les retour-chariot* à *espace.*) soit affiché en vert ?

6.3 Style

La façon dont les caractères du contenu sont affichés (mise en forme et mise en page) peut être modifiée par des règles dans une feuille de style (langage CSS). Un fichier texte (d'extension `.css`) peut être associé à la page HTML par la balise <**link rel**="stylesheet"**type**="text/css"**href**="mystyle.css "/> dans la partie <**head**>. Ce fichier précise comment les différentes zones (divisions, paragraphe, etc.) doivent s'afficher. Chaque règle de style comporte un sélecteur (qui désigne ce qui est concerné dans le HTML) et entre accolades une suite de couples (propriété, valeur) sous la forme propriété :valeur ; comme dans les exemples suivants. p text−align :right ; indique d'aligner le long du bord droit ce qui est entre un <**p**> et le </**p**> correspondant. img width :150px ; permet d'obliger chaque image à se restreindre à 150 pixels de large. Les principaux sélecteurs sont les noms de balises (`p` ou `div` par exemple), les classes (précédées d'un point) et les identificateurs (précédés d'un croisillon (#)).



Exercice 54

Écrire la règle CSS permettant de faire afficher en jaune le mot décomposé.

Exercice 55

Décrire les modifications à apporter au code HTML fourni pour que les règles CSS ci-dessous provoquent l'affichage ci-contre.

```
1 .feutre {font−weight :bold ;}
2 .stylo {font−style :italic ;}
3 p.stylo {text−align :right ;}
4 #dec {background−color :grey ;}
```

Listing 6.2 : Règles CSS

6.4 Javascript

Il est possible d'associer du Javascript à la page HTML, directement en l'encadrant par <**script**> et </**script**>, ainsi que dans un fichier séparé en utilisant <script src="myScript.js"></**script**> dans le <**head**>. Remarque : pour qu'une fonction Javascript init() soit exécutée juste après le chargement de la page, ajouter l'attribut onload="init();" dans la balise <**body**>.

Une fonction Javascript peut accéder aux éléments affichés en faisant référence aux balises HTML. Si le code HTML contient par exemple la balise <**img src**="chat.jpeg"**alt**="un joli chat"**id**="chat"/>, alors une fonction Javascript peut utiliser document.getElementById('chat').alt pour obtenir le texte alternatif, et utiliser document.getElementById('chat').src="garfield.jpg" pour modifier le fichier associé à l'image (ce qui a pour conséquence de changer l'image que voit l'internaute).

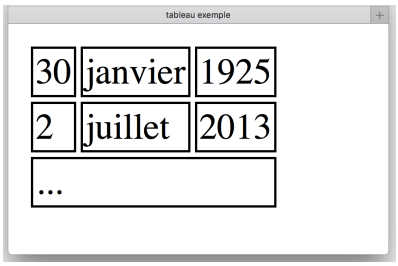
De la même façon document.getElementsByTagName('p')[1].innerHTML permet d'accéder au texte contenu entre la seconde balise <**p**> du document et la balise </**p**> correspondante (dans l'exemple, il s'agit de tout le texte allant de "Les passages" à "fenêtre".)

Exercice 56

Écrire le code Javascript d'une fonction affichant (en utilisant alert()) le nombre moyen de paragraphes (balise <**p**>) par division (balise <**div**>) dans le document HTML qui appelle la fonction. Dans l'exemple fourni, cette moyenne vaut 1 (une <**div**> de 2 <**p**>, une autre de 0 <**p**>).

6.5 Tableaux

Un tableau HTML est une succession de rangées (row) elles-mêmes constituées de cellules (data). Par exemple le code ci-dessous à gauche fournit l'affichage à droite.



Remarque : on peut accéder en Javascript à la 3ème cellule de la 5ème rangée en utilisant document.getElementsByTagName('tr')[4].getElementsByTagName('td')[2].

```
<!doctype html>
<html>
  <head>  <meta charset="UTF-8" />
          <title> tableau exemple </title>
         <style> td {border :solid ;} </style>
  </head>
  <body>
    <table>  <!-- le tableau commence ici -->
      <tr>   <!-- première rangee-->
         <td> 30 </td> <td> janvier </td>
         <td> 1925 </td>
      </tr>
      <tr>  <!-- seconde rangee-->
         <td> 2 </td> <td> juillet </td>
         <td> 2013 </td>
      </tr>
      <tr>  <!-- troisième rangee→-
            <!-- 3 cellules fusionnees-->
         <td colspan=3> ... </td>
      </tr>
    </table> <!-- le tableau finit ici -->
  </body>
</html>
```

Listing 6.3 : tableau.html

Exercice 57

Écrire le code Javascript permettant de mettre sur fond jaune les deux noms de mois.

Exercice 58

Écrire le code Javascript permettant d'ajouter dans la rangée en bas du tableau le prénom et le nom Douglas Engelbart.

Certains <td> peuvent être remplacés par <th> (table header) pour représenter les entêtes de ligne ou de colonne. La feuille de style peut ainsi leur attribuer un aspect différent. Une syntaxe particulière permet en CSS d'alterner les styles un élément sur deux, par exemple :

```
tr :nth-child(even) td :nth-child(odd) {color :red ;}
```

Listing 6.4 : Styles pair et impair

permet que soit écrit en rouge le texte des cellules de numéro impair dans les rangées de numéro pair.

Exercice 59

Écrire le code CSS permettant d'obtenir que toutes les cases sauf les noms de mois soient écrites en caractères gras.

Exercice 60

Écrire les trois fichiers (HTML, CSS et Javascript) permettant d'afficher le calendrier d'un mois comme dans l'exemple ci-contre. Le fichier HTML décrira un tableau vide et le contenu des cases du tableau sera rempli par une fonction Javascript prenant en paramètre le jour de la semaine, le mois et l'année de la date de départ.

7

TypeScript – Concepts avancés

Fonctions anonymes fléchées

Les fonctions anonymes fléchées sont des fonctions qui utilisent une syntaxe raccourcie. Elles omettent le mot-clé function et ajoutent une flèche => entre les arguments et le corps de la fonction. Dans l'exemple suivant, nous avons une fonction appelé hello qui ne prend aucun argument et ne retourne aucune valeur :

```
1 var hello = () => { console.log("Hello World");};
2
3 hello(); // appel
```

Tout comme les fonctions classiques, les fonctions fléchées peuvent avoir des paramètres et un type de retour. Voici la syntaxe d'une fonction fléchée ayant des paramètres et retournant une valeur :

```
1 var score = (username: string, points: number): string => {
2       return `${username} scored ${points} points!`;
3 };
4
5 console.log(score('Saad', 3)); // appel
```

Les fonctions anonymes fléchées sont particulièrement utiles pour manipuler des tableaux. Par exemple, la fonction forEach() applique une fonction à tous les éléments d'un tableau. Considérez l'exemple suivant :

```
1 let hello = (name:string) => console.log(`Hello ${name}`);
2 let names : Array<string> = ['Thomas', 'Saad', 'Belal'];
3
4 names.forEach(hello)
```

La fonction fléchée sera appliquée trois fois, c'est à dire, la chaine "Hello XXX" sera affichée trois fois sur la console. On peut aussi les utiliser directement, sans passer par une variable :

```
let names: Array<string> = ['Thomas', 'Saad', 'Belal'];
names.forEach((name: string) => {console.log(`Hello ${name}`);})
```

Exercice 61 *Fonctions fléchées*

Écrivez deux fonctions fléchées permettant de (i) additionner deux nombres et (ii) multiplier deux nombres :

Classes

Les classes sont un type très spécial d'enregistrement, qui combinent les champs (déjà vus avec les types) et les fonctions. Par exemple, supposez que vous souhaitez créer un enregistrement qui représente des coordonnées d'un point. Le code TypeScript correspondant est le suivant :

```
class Point {
    x: number;
    y: number;
}
```

Constructeurs

Contrairement aux Types, les classes peuvent avoir des *constructeurs*, qui sont un type particulier de fonction appelée lors de la création d'une instance (valeur) de la classe. En TypeScript, les constructeurs s'appellent toujours constructor. Les constructeurs simplifient la création de nouvelles instances. Voici un exemple d'un constructeur de la classe Point.

```
class Point {
    x: number;
    y: number;

    constructor(abscissa: number, ordinate: number) {
        this.x = abscissa;
        this.y = ordinate;
    }
}

let p1: Point = new Point(10, 15); // appel
```

Exercice 62 *Constructeurs*

Donnez le code Typescript d'une classe Date, contenant 3 propriétés (jour, mois et année), ainsi qu'un constructeur. Proposez un constructeur pour cette classe.

Méthodes

Les classes peuvent contenir des *méthodes*. Les méthodes sont un type particulier de fonction : elles sont contenues par une classe et possèdent une paramètre implicite appelé "this", dont le type est celui de sa classe. Par exemple, considérez la fonction toString() présentée ci-dessous :

```
function toString(this: Point): string {
    return `[${this.x}, ${this.y}]`;
}

let p1: Point = new Point(10, 15);
console.log(toString(p1));
```

On peut la transformer en méthode en la plaçant à l'intérieur de classe Point, comme suit :

```
class Point {
    x: number;
    y: number;

    toString(): string {
        return `[${this.x}, ${this.y}]`;
    }
}

let p1: Point = new Point(10, 15);
console.log(p1.toString());
```

L'appel des méthodes utilise la syntaxe *pointée*, c'est à dire, l'appel utilise le format "type.methode()".

Exercice 63 *Méthode*

Donnez le code TypeScript d'une méthode booléenne appelée isValid() permettant de valider votre classe Date. Basez vous sur les fonctions que vous avez déjà écrites.

8

Sujet d'entrainement

8.1 Jeu de la vie de Conway

Ce jeu se déroule sur une grille à deux dimensions, où les cases (aussi appelées *cellules*) composant cette grille peuvent avoir deux états : *vivante* ou *morte*. L'état de cette grille évolue par étapes en suivant les règles suivantes :

- Chaque cellule a 8 voisins, qui sont ses cellules adjacentes.
- Lors d'une étape, une cellule morte possédant exactement trois cellules voisines vivantes devient vivante.
- Lors d'une étape, une cellule vivante possédant deux ou trois cellules voisines vivantes le reste, sinon elle meurt.

Exercice 64

Définissez deux enregistrements : un premier pour représenter une cellule, un second pour représenter une grille.

Exercice 65

On considère le code de deux fonctions montrées en Fig 8.1. Deux développeurs Alice et Bob travaillent simultanément sur ce code, et se synchronisent à l'aide de *git* et d'un référentiel public distant. Les évènements suivants se produisent dans cet ordre :

1. Alice écrit les lignes 1-4 sur sa copie locale.
2. Bob écrit les lignes 6-21 sur sa copie locale.
3. Alice met ses changements sur la scène (`git stage`) et effectue un commit sur son référentiel local (`git commit`)

4. Bob met ses changements sur la scène (`git stage`) et effectue un commit sur son référentiel local (`git commit`)
5. Bob envoie ses changements (`git push`)
6. Alice récupère les changements distants (`git pull`)
7. Alice modifie les lignes 13-15 sur sa copie locale.
8. Alice met ses changements sur la scène (`git stage`) et effectue un commit sur son référentiel local (`git commit`)
9. Bob modifie la lignes 13 sur sa copie locale.
10. Bob met ses changements sur la scène (`git stage`) et effectue un commit sur son référentiel local (`git commit`)
11. Alice envoie ses changements (`git push`)
12. Bob récupère les changements distants (`git pull`)

Que se passe-t-il à chaque étape ? Si besoin, quelles actions doivent être réalisées pour résoudre la situation ?

Exercice 66

Améliorez le code des fonctions montrées en Fig 8.1.

Exercice 67

Définissez une fonction renvoyant une grille dont les dimensions sont passées en paramètres sous la forme de deux entiers. Les cellules de cette grille doivent être initialisées de façon à ce que chacune d'entre elle ait un certain pourcentage de chance d'être vivante ; ce pourcentage est passé en paramètre de la fonction. La fonction *Math.random()* renvoie un nombre entre 0 (inclusif) et 1 (exclusif).

Exercice 68

Définissez une fonction qui calcule si une cellule donnée doit être vivante ou morte suite à la prochaine étape.

Exercice 69

Définissez un ensemble de scénarios de test pour la fonction définie lors de la question précédente, et programmez ces tests avec Alsatian.

```
1  // true if cell is alive
2  function Alive(g: Grid, x: number, y: number): boolean {
3      return g.cells[x][y].isAlive;
4  }
5
6  // number of alive neighbours of cell
7  function Neighbours(g: Grid, x: number, y: number): number {
8      let res: number = 0;
9      for (let i: number = x-1; i<=x+1; i++) {
10         for (let j: number = y-1; j<=y+1; y++) {
11             if (i!=x || j!=y) {
12             if (i>=0 && j>=0 && i<=g.cells.length && j<=g.cells[0].
                   length) {
13             if (alive(g, i, j)) {
14                 res = res + 1;
15             }
16             }
17             }
18         }
19     }
20     return res;
21 }
```

FIG. 8.1 : Fonctions `alive` et `neighbours`

Exercice 70

Définissez une fonction prenant en paramètre une grille et renvoyant une nouvelle grille correspondant à l'étape suivante de celle passée en paramètre.

Exercice 71

Définissez une fonction permettant d'initialiser une grille de dimension passée en paramètre et de la faire évoluer sur un nombre d'étapes également passé en paramètre. On souhaite de plus afficher le résultat de chaque étape, on fait l'hypothèse pour cela avoir accès à une fonction `print(grid : Grid)`.

Exercice 72

Modifiez le code existant pour permettre de simuler la variante suivante : les bords de la grille sont connectés de telle manière que les cellules du haut sont voisines de celles du bas, et celles de droite voisines de celles de gauche.

Exercice 73

Modifiez le code existant pour permettre de simuler la variante suivante : les cellules vivantes possèdent également une couleur, rouge ou bleu. À l'initialisation, la couleur des cellules vivantes est choisie aléatoirement. Quand une

cellule passe de l'état mort à l'état vivant, sa couleur est la couleur majoritaire des cellules qui lui ont donné vie.