# I. Programmation typescript

## Table des matières

# II. Environnement

## 1. Installation

### *Installation*

Installation avec npm

```
>npm install -g typescript
```

version du compilateur ts

```
>tsc -version
Version 2.8.3
```

### *Utilisation console ts-node*

Installation de node-ts pour exécuter des script + console.log

```
>npm install -g ts-node
```

Déclaration d'une fonction

```
> function somme(i,j) {
... return i+j
... }
undefined
> somme(10,20)
30
```

Déclaration d'une constante

```
> const i : number = 10
undefined
> i
10
> i=11
Thrown: × Unable to compile TypeScript
[eval].ts (1,1): Cannot assign to 'i' because it is a constant or a read-only property. (2540)
```

## 2. Types de données

### *Enumeration*

```
enum semaine {
   lundi,
   mardi,
   mercredi,
   jeudi,
```

```
        vendredi,
        samedi,
        dimanche
    }

    var jour:semaine = semaine.lundi;
    console.log (semaine[0]);
    console.log (jour);
```

Affiche

```
    lundi
    0
```

### *Tableaux*

Créer

```
    var abool: boolean[];
    // initialiser
    abool = [true, false];
    console.log(abool);
    console.log(abool[0]);
```

Créer et initialiser

```
    var abool:boolean[] = [true, false];
    console.log(abool);
```

ajouter dans le tableau

```
    abool.push(true)
    console.log(abool);
```

Ajouter

```
    abool.unshift(true)
    console.log(abool);
```

retirer

```
    var d = abool.pop()
    console.log(d);
    console.log(abool);
```

retirer à gauche

```
    var d = abool.shift()
    console.log(d);
    console.log(abool);
```

parcourir

```
for (var i of abool)
{
    console.log('***' + i);
}
```

### *Types combinés*

```
var i : number|string
i=10
console.log(i)
i='10'
console.log(i)
```

```
var i : sring |{nom : string } ;
```

### *Tuple*

```
var i= [11,22,33]
console.log(i)
console.log(typeof i)
```

Affiche

```
[ 11, 22, 33 ]
object
```

### *Function*

```
var f : (i:number)=>number
f = (i:number) : number =>{
    return i * i
}
console.log (f(10))
```

Affiche

```
100
```

### *Types particuliers*

- any : quelconque
- null : non initialisé
- undefined : non défini
- void : pas attendu pour les retour de fonctions

```
var d:number;
d= 'TOTO';
```

Exception

```
TSError:Unable to compile TypeScript
```

```
exo_type.ts (2,1): Type '"TOTO"' is not assignable to type 'number'. (2322)
```

Any

```
var d:any;
console.log (typeof d)
d='123'
console.log (d)
console.log (typeof d)
d=123
console.log (d)
console.log (typeof d)
```

Affiche

```
undefined
123
string
123
number
```

null et undefined

```
var i:number = null
console.log(typeof i)
console.log(i)

var i:number = undefined
console.log(typeof i)
console.log(i)
```

Affiche

```
object
null
undefined
undefined
```

void

```
function somme(i,j) : void
{
    return i + j
}

var k:number = somme(10,20)

console.log(k)
console.log(typeof k)
```

Affiche

```
TSError: Unable to compile TypeScript
exo_type.ts (6,5): Type 'void' is not assignable to type 'number'. (2322)
```

## 3. Instructions

### *Boucle*

for

```
var max = 10;
for (var i = 0; i < max; i++) {
   console.log("Je compte " + i);
}
```

while

```
var ii : number = 0;
while(ii++<max)
{
    //console.log("Je compte " + ii)
}
```

```
var ii : number = 0;
do
{
    console.log("Je compte " + ii)
}
while(ii++<max)
```

### *tests*

```
const max : number = 10;
for(var i=0; i <max; i++)
{
    if (i<5) console.log("Je compte <5 :" + i)
    else console.log("Je compte >=5 :" + i)
}
```

```
const max : number = 10;
for(var i=0; i <max; i++)
{
    if (i<5) console.log("Je compte <5 :" + i)
    else break
}
```

```
const max : number = 10;
```

```
for(var i=0; i <max; i++)
{
    if (i<5) continue;
    console.log("Je compte <5 :" + i)
}
```

switch

```
const max : number = 10;
for(var i=0; i <max; i++)
{
    switch (i%2)
    {
            case 0 : console.log(i + "::pair"); break
            default : console.log(i + "::impair"); break
    }
}
```

## 4. Opérateurs

Opérateur ternaire

```
var i : number = 5;
console.log(i % 2 == 0 ? "pair":"impair")
i = 4;
console.log(i % 2 == 0 ? "pair":"impair")
```

## 5. Fonctions

Fonction avec argument et valeur de retour

```
function bonjour (aqui) : string {
   return "Bonjour, " + aqui;
}
let personne="à tous
console.log(bonjour(personne))
```

Argument facultatif

```
class Calcul
{
    somme (i:number,j:number,k?:number) :number
    {
            if (k == undefined)
            {
                    return i+j;
            }
            else
```

```
                {
                        return i+j+k;
                }
        }
    }

    let calcul =  new Calcul()
    console.log(calcul.somme(10,20))
    console.log(calcul.somme(10,20,30))
```

Autrement

```
    class Calcul
    {
        somme (i:number,j:number,k?:number) :number
        {
                if (k)
                {
                        return i+j+k;
                }
                else return i+j;
        }
    }
```

Nombre variable d'arguments : for i in

```
    function somme (...data:number[]) : number
    {
        let s=0;
        console.log(data);
        for (var i in data)
        {
                s+=data[i];
        }
        return s;
    }

    console.log (somme(10,20));
    console.log (somme(10,20,30));
```

parcours avec : for of

```
    function somme (...data:number[]) : number
    {
        let s=0;
        console.log(data);
```

```
        for (var i of data)
        {
                s+=i;
        }
        return s;
    }
```

curring

```
    let add = (x: number) => (y: number) => x + y;
    console.log(add(111)(222));
    let add111=add(111)
    console.log(add111(222));
```

Afficher

```
    333
    333
```

méthode static

```
    const {compter} = new class {
       count = 0;
       compter = () => {
          this.count++;
          return this.count;
       }
    };

    console.log(compter())
    console.log(compter())
```

Affiche

```
    1
    2
```

## 6.Modules

Les modules sont des fichiers .ts définissant des données, fonctions et classes (export)  pour être utilisée ailleurs après importation (import)

Module

```
    export var data = 123;

    export function somme(i,j) : number
    {
        return i + j
    }
```

Appel

```
import {data, somme} from "./monmodule"
console.log(data)
console.log(somme(10,20))
```

## 7. Programmation orientée objets

### *classes et instances*

constructeur et autoréférence this

```
class Formation {
	id : number;
	libelle : string;
	constructor (id : number, libelle : string)
	{
		this.id = id;
		this.libelle= libelle;
		console.log ("constructeur");
	}
	getAll () : string
	{
		return this.id + "::" + this.libelle
	}
}

let f = new Formation (10, 'Formation 10')
console.log (f.getAll ())
```

### *Public, private*

```
class Calcul
{
	private somme (i:number,j:number) : number
	{
		return i+j;
	}
}

let calcul =  new Calcul()
console.log(calcul.somme(10,20))

exo_type.ts (15,20): Property 'somme' is private and only accessible within class 'Calcul'. (2341)
```

### *héritage et dérivation*

Redéfinition et super

```
class Personne {
	nom : string;
	age : number;
	constructor (nom : string, age : number)
	{
		this.nom = nom;
		this.age= age;
		console.log ("constructeur");
	}
	getAll () : string
	{
		return this.nom + "::" + this.age
	}
}

class Salarie extends Personne
{
	anciennete : number;
	constructor (nom : string, age : number, anciennete: number)
	{
		super(nom, age);
		this.anciennete=anciennete;
		this.nom = nom;
	}
	getAll () : string
	{
		return super.getAll () + "::" + this.anciennete
	}
}

// Personne
var p = new Personne("TOTO1",1)
console.log(p.getAll())
// Salarie
var s = new Salarie("TOTO1",1,10)
console.log(s.getAll())
```

### *Classe template et généricité*

```
class Stocker<T> {
	data : T[] = [];
```

```
        push (o) : void
        {
                this.data.push(o)
        }
}

let s = new Stocker<string> ()
s.push("TOTO1")
s.push("TOTO2")
console.log (s.data)
```

### *Interface*

```
interface IFormation {
        id : number,
        libelle : string,
        getAll:()=>string
}

let uneformation:IFormation = {
        id : 11,
        libelle : 'formation 22',
        getAll : () : string => { return this.libelle}
}
console.log (uneformation)
console.log (uneformation.getAll())
```

Interface en tant que type

```
interface IFormation{
        id : number;
        libelle : string;
}

let uneformation = {} as IFormation
uneformation.id = 10;
uneformation.libelle = "formation 10";
console.log(uneformation)

let uneformation:IFormation = {
        id : 10,
        libelle : "formation 10"
}
console.log(uneformation)
```

```
let uneformation : IFormation
uneformation = {id : 11, libelle : 'Formation 11'}
console.log(uneformation)
```

### *Itérateur*

```
interface Iterator<T> {
   next(value?: any): IteratorResult<T>;
   return?(value?: any): IteratorResult<T>;
   throw?(e?: any): IteratorResult<T>;
}
```

```
interface IteratorResult<T> {
   done: boolean;
   value: T;
}

class Stock implements Iterator<String>
{
     data = ["AAA","BBB","CCC"];
     i:number =0;
     public next(): IteratorResult<String> {
               if (this.i < this.data.length) {
                 return {
                          done: false,
                          value: this.data[this.i++]
                 }
               }
               else
               {
                 return {
                          done: true,
                          value: null
                 }
               }
     }
}

var s:Stock = new Stock();
console.log(s.next())
console.log(s.next())
console.log(s.next())
console.log(s.next())
```

Affiche

```
{ done: false, value: 'AAA' }
{ done: false, value: 'BBB' }
{ done: false, value: 'CCC' }
{ done: true, value: null }
```

## 8. Exceptions

try/catch

```
try {
  throw new Error('un soucis');
}
catch(e) {
  console.log(e);
}
```