TypeScript

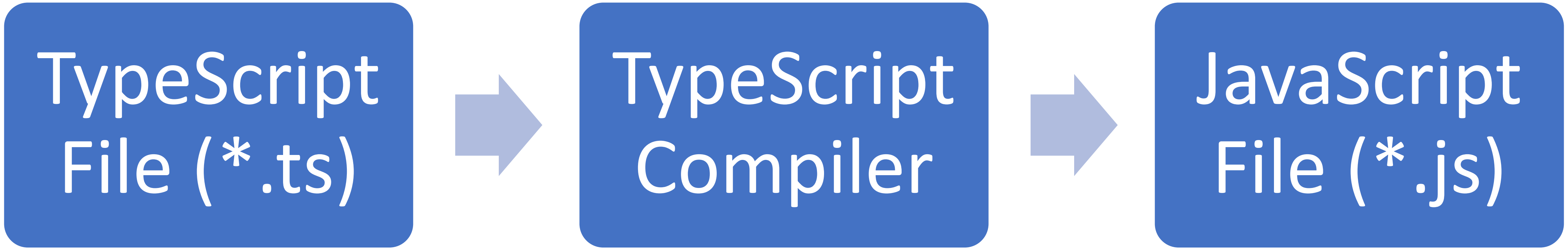
TypeScript Overview

- Free and Open Source
- High-level Programming Language
- Developed and Maintained by Microsoft
- Strict syntactical superset of JavaScript
- Origin: 2012

TypeScript References

- Programming Language
  - https://www.typescriptlang.org/
- Reference Handbook
  - https://www.typescriptlang.org/docs/handbook/intro.html
- TypeScript Tutorial
  - https://www.tutorialspoint.com/typescript/index.htm
- TypeScript Cheat Sheets
  - https://www.typescriptlang.org/cheatsheets

TypeScript Compile Process



Naming Conventions

TypeScript Naming Conventions

- https://makecode.com/extensions/naming-conventions
- https://google.github.io/styleguide/tsguide.html
- Namespaces, function, function parameters, methods, fields are camel cased. Single word names are all lowercase.
  - Style: aaaBbbCcc
  - Examples: myFunction(), myMethod(), myField
- Class, enums and enum members are capitalized.
  - Style: Name, TheName
  - Examples: MyClass, MyEnum, MyEnumMember
- Constants
  - Style: ABCDEF_GHIJ
  - Example: COUNTRY_CODE

TypeScript Naming Conventions

- Spell out words entirely instead of using acronyms
- Although the names are longer, this helps convey the meaning of your API.
  - Exceptions might be single letter identifiers, like the coordinate names x, y, z.

```
// long but self-explanatory
export function doSomethingAwesome() {
}

// not clear
export function dSA() {
}
```

Operators

Operators (1/2)

- **Arithmetic Operators**
  - Multiplication, Division
    - *, /
  - Exponential
    - **
  - Integer Division Remainder
    - %
  - Sum, Subtraction
    - +, -
  - Increment, Decrement
    - ++, --

- **Logical Operators**
  - Conjunction, Disjunction, Negation
    - &&, ||, !

- **Relational Operators**
  - Less, Than, Less of Equal Than
    - <, <=
  - Equal, not equal
    - ==, !=
  - Higher Than, Equal or Higher Than
    - >, >=

Operators (2/2)

- **Simple Assignment Operator**
  - =
- **Add and Assignment Operator**
  - +=;
    - c += a; is the same as: c = c + a;
- **Subtract and Assignment Operator**
  - -=;
    - c -= a; is the same as: c = c - a;
- **Same for Multiplicity and Division**
  - *=, /=

Statements

Statements

- Statements end with “;”
  - 4 + 5;

Variables

Variable Declaration

- Declaration
  - let <variable name> :<type> = <value>;

  - var <variable name> :<type> = <value>;
- Mandatory
  - <variable name>
- Optional
  - <type>
  - <value>
- Remarks
  - You should not use the var option

Types

Types

- Primitive Types
  - string
    - let city: string = “Kansas”;
  - number
    - let birthYear: number = 1950;
  - boolean
    - let thisHappened: boolean = true;
  - any
    - For anything
    - let surname: any = “Doe”

Constants

Constant

- Constants must be initialized
- Constant values do not change

- Constants do not require a type
  - `const PORTUGAL_COUNTRY_CODE = "PT";`

- But you can specify one
  - `const FRANCE_COUNTRY_CODE: string = "FR";`
  - `//FRANCE_COUNTRY_CODE = "PT"; //this is not possible`

Decision Control Structures

if statement

```
SE (condição) ENTÃO
        <bloco de instruções>
FIM SE

num <- = 5
SE (num > 0) ENTÃO
        ESCREVE “POSITIVE”
FIM SE
```

```
if (boolean_expression) {
        //statements
}

var num: number = 5;
if (num > 0) {
        console.log(”POSITIVE") ;
}
```

if…else statement

```
SE (condição) ENTÃO
        <bloco de instruções>
SENÃO
        <bloco de instruções>
FIM SE

num <- = 5
SE (num % 2 = 0) ENTÃO
        ESCREVE “EVEN”
SENAO
        ESCREVE “ODD”
FIM SE
```

```
if (boolean_expression) {
        //statements
} else {
        //statements
}

var num: number = 5;
if (num %2 == 0) {
        console.log(”EVEN");
} else
        console.log(”ODD");
}
```

else...if and nested if statements

```
SE (condição) ENTÃO
     <bloco de instruções>
SENÃO SE (condição) ENTÃO
          <bloco de instruções>
     SENÃO
          <bloco de instruções>
     FIM SE
FIM SE
```

```
if (boolean_expression) {
     //statements
} else if (boolean_expression) {
     //statements
} else {
     //statements
}
```

else...if and nested if statements

```
num <- = 5
SE (num > 0) ENTÃO
        ESCREVE “Positive”
SENAO SE (num < 0) ENTÃO
                ESCREVE “Negative”
        SENAO
                ESCREVE “Neither”
        FIMSE
FIM SE
```

```
var num: number = 5;
if (num > 0) {
        console.log(”Positive");
} else  if (num < 0) {
        console.log(”Negative");
} else  {
        console.log(”Neither");
}
```

switch statement

```
switch(variable_expression)
{           case constant_expr1: {
                        //statements;
                        break;
            } case constant_expr2: {
                        //statements;
                        break; }
            default: {
                        //statements;
                        break;
            }
}
```

```
var grade: string = "A";
switch(grade) {
            case "A": {
                console.log("Excellent");
                break;
            } case "B": {
                console.log("Good");
                break;
            } default: {
                console.log("Invalid");
                break;
            }
}
```

Repetition Control Structures

For Loop ...

```
REPETIR PARA <v> <- <vi> ATE <vf>
PASSO <p>
        <bloco de instruções>
FIMREPETIR
```

```
for (initial_count_value;
termination_condition; step){
        //statements
}
```

```
REPETIR PARA n <-1 ATE 100
PASSO 1
        ESCREVER (n)
FIMREPETIR
```

```
for(n = 1 ; n <= 100; n++) {
        console.log(n);
}
```

While Loop …

```
REPETIR ENQUANTO (<condição>)
        <bloco de instruções>
FIMREPETIR

n <- 1
REPETIR ENQUANTO (n <= 100)
        ESCREVER (n)
        n <- n +1
FIMREPETIR
```

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

let n = 1;
while(n <= 100) {
        console.log(n);
        n++;
}
```

Do ... While

```
REPETIR
        <bloco de instruções>
ENQUANTO (<condição>)

n <- 1
REPETIR
        ESCREVER (n)
        n <- n + 1
ENQUANTO (n <= 100)
```

```
do {
        //statements
} while(condition)

let n = 1;
do {
        console.log(n);
        n++;
} while(n <= 100)
```

Functions

Functions

- TypeScript allows you to specify the types of input values of functions
- When you declare a function, you can add type annotations after each parameter to declare what types of parameters the function accepts. Parameter type annotations go after the parameter name.

```
// Parameter type annotation
function greet(name: string) : string {
        console.log("Hello, " + name.toUpperCase() + "!!");
}
```

Functions

- TypeScript allows you to specify the types of output values of functions

```
function getFavoriteNumber(): number {
        return 26;
}
```

Comments

TypeScript Code Comments

- Comments should be avoided on production code

- It is possible to comment a single line of code with //

```
let anotherCityName: string = "Porto";  //specify the type, and initialize value
```

- It is possible to comment a block of code with /* */

```
/*
let anotherCityName: string = "Porto";
 */
```

References

- https://www.typescriptlang.org/
- https://www.typescriptlang.org/docs/handbook/intro.html
- https://www.tutorialspoint.com/typescript/index.htm
- https://www.typescriptlang.org/cheatsheets
- https://makecode.com/extensions/naming-conventions
- https://google.github.io/styleguide/tsguide.html
- https://makecode.com/extensions/naming-conventions