**Intro**

**Primitive types**

* any – aby type (explicitly untyped)
* void – undefined or null, use for function returns only
* undefined
* null
* readonly
* string
* numer
* boolean
* object

**Type aliases**

*type int = numer*

*type float = number*

The JS does not have types Integer and Float. It is only the type aliases. So you can make your code more documented and give a hint for programmers.

**Type assertions**

*let len: number = (****input as string****).length*

*let len: number = (****<string> input****).length*

**var /let / const**

* **var** – define variable with function scope
* **let** – define variables with block scope
* **const** – define immutable variable

**Named types (interface, class, enum)**

* **interface**

***interface*** *Child extends Parent, SomeClass {*

*property: Type;*

*optionalProp?: Type;*

*optionalMethod?(arg1: Type): ReturnType;*

*}*

* **class**

*class* ***Child*** *extends Parent implements Child, OtherChild {*

*property: Type;*

*defaultProperty: Type = 'default value';*

*private \_privateProperty: Type;*

*static staticProperty: Type;*

*constructor(arg1: Type) {*

*super(arg1);*

*}*

*private \_privateMethod(): Type {}*

*methodProperty: (arg1: Type) => ReturnType;*

*overloadedMethod(arg1: Type): ReturnType;*

*overloadedMethod(arg1: OtherType): ReturnType;*

*overloadedMethod(arg1: CommonT): CommonReturnT {}*

*static staticMethod(): ReturnType {}*

*subclassedMethod(arg1: Type): ReturnType {*

*super.subclassedMethod(arg1);*

*}*

*}*

* **enum**

*enum* ***Options*** *{*

*FIRST,*

*EXPLICIT = 1,*

*BOOLEAN = Options.FIRST | Options.EXPLICIT*

*}*

*enum* ***Colors*** *{*

*Red = "#FF0000",*

*Green = "#00FF00",*

*Blue = "#0000FF"*

*}*

**Object type literals**

* **Object with implicit Any properties**

*{ foo; bar; }*

* **Object with optional property**

*{ required: Type; optional?: Type; }*

* **Hash map**

*{ [key: string]: Type; }*

* **Indexable types**

*{ [id :number] :Type; }*

**Union and intersection types**

* **Union type**

*let myUnionVariable: number | string;*

* **Intersection type**

*let myIntersectionType: Foo & Bar;*

**Arrays and tuples**

* **Array** (of strings)

*string[]*

*Array<string>*

* **Array** (of functions that return strings)

*{ (): string; }[]*

*Array<() => string>*

* **Tuples**

*let myTuple: [ string, number ];*

*myTuple = [ 'test', 42 ];*

**Functions**

* **Base function**

*function fn(arg1: Type): ReturnType {}*

* **Default argument**

*function fn(arg1: Type = 'default'): ReturnType {}*

* **Arrow function**

*(arg1: Type): ReturnType => {}*

*(arg1: Type): ReturnType => Expression*

**Rest parmeters**

*function iTakeItAll(first, second, ...****allOthers****) {*

*console.log(****allOthers****);*

*}*

*iTakeItAll('foo', 'bar'); // []*

*iTakeItAll('foo', 'bar', 'bas', 'qux'); // ['bas','qux']*

**Modules**

Starting with ECMAScript 2015, JavaScript has a concept of modules. TypeScript shares this concept.

Modules are executed within **their own scope**, not in the **global scope**; this means that variables, functions, classes, etc. declared in a module are not visible outside the module unless they are explicitly exported using one of the **export** forms. Conversely, to consume a variable, function, class, interface, etc. exported from a different module, it has to be imported using one of the **import** forms.

In TypeScript, just as in ECMAScript 2015, any file containing a top-level import or export is considered a module. Conversely, a file without any top-level import or export declarations is treated as a script whose contents are available in the global scope (and therefore to modules as well).