Typescript Quick Reference

Types

```
let better than var more scope and call strict.
Use const for variables and readonly for properties
typeof: like javascript so: let x:number; typeof x == "number"
type alias: type Custom = string;
boolean: let isDone: boolean = false;
number:  let value: number = 6; (or 0xf00d, 0b1010, 0o744)
string:  let name: string = "Something\n" + fname + (age + 1);
array<>: let list: number[] = [1,2,3];
         let list2: Array<number> = [1, 2, 3];
tuple:   let x: [string, number]; x = ["hello", 10];
enum:    enum Color {Red, Green}; let c: Color = Color.Green;
         enum Test { V1=1,V2="123".length}; Test[Test.V1]==1;
any:     let n: any = 4; n = "str"; n = false; let an: any[];
void:    function test(): void {…}
special: undefined; null;
never:   function err(msg:string):never{throw new Error(msg);}
type assertions: let s:number=(<string>strval).length; //casts
to directly cast: something = other as type;
```

Destructuring Array/Object

```
swapping: [first, second] = [second, first];
for params:
    function f([first,second]:[number,number]) {…}
    let [first, ...rest] = [1, 2, 3];//first=1,rest=[2,3]
    let [, s, , f] = [1,2,3,4];//s=2,f=4, rest is omitted
Same for objects gives multiple useful features.
```

Interfaces

```
interface Example {
    label: string; // mandatory property
    color?: string; // optional property
    [propName: string]: any; // could have any number of props
    (par1: string, par2: string): boolean; //func signature
    [index: number]:string; // class can be indexed into
}
class Clock implements ClockInterface {…}
interface ExampleExtend extends Example, ExampleOther {…}
```

Classes

```
members are public by default. can be individually set to
private or protected. use readonly for constants.
class Example {
    prop1: string;
    static stprop: {x:0, y:0};
    constructor(msg: string) { this.prop1 = msg; }
    method() {…}
    get prop1_accessor(): string { return this.prop1; }
    set prop1_accessor(s:string) { this.prop1 = s; }
}
let exclass = new Example("Hello!");
class ExampleInherit extends Example {
    constructor(msg: string) { super(msg); }
    move(dist = 5) { super.move(dist); }
}
abstract class Test {
    abstract func1(): void;
    func2(): void {…}
} // Abstracts can be extended by classes of course
```

Functions

```
function add(x:number, y:number):number { return x + y }
let myAdd:(x:number,y:number)=>number = add; // function type
function example(defval:string="def", optionalval?:string){…}
function params(fn:string, ...rest:string[]) {…}
Arrow function captures this where function is created:
let something = { exampleFunc: function() {
    return () => {…} // stuff using `this` } };
```

Generics

```
function exFunc<T>(arg:T, aarg:T[], aaarg:Array<T>):T {…}
let myExFunc:<T>(arg:T, aarg:T[], aaarg:Array<T>)=>T = exFunc;
class GenericExample<T> { value: T; }
let c = new GenericExample<string>();
Setting up a generic constraint:
interface StuffWithLength { length: number; }
function exFunc2<T extends StuffWithLength>(arg:T):T {…}
For factory, necessary to refer to class type by constructor:
function create<T>(c: {new(): T;}):T { return new c(); }
```

Iterators

```
for(let i in list) { returns keys "0", "1", "2", .. }
for(let i of list) { returns values }
```

Modules and Namespaces

```
Each typescript runs in own scope. export vars, funcs, classes,
interfaces,.. and import them in another script to use.
export interface IExample {…}
export const someregex = /^[0-9]+$/;
export class CExample implements CParent {…} //module_name.ts
export { CExample as RenamedExportExample };
from other files, you can:
export {CExample as AReExport} from "./module_name"; //reexport
export * from "./module_name"; // exports class CExample
To import from another module:
import {CExample} from "./module_name"; let m = new CExample();
import {CExample as CMy} from "./module_name"; // rename
import * as EX from "./module_name"; let m = new EX.CExample();
A unique default exports file can be used: module.d.ts
declare let $: JQuery; export default $;
Then from another module to import that stuff:
import $ from "JQuery; $("something").html("something");
Classes and funcs can be authored directly as default exports:
export default class CExample {…}
From another module:
import Whatever from "./module_name"; // Whatever == CExample
For standard require functionality, to export then import:
export = CExample;
import ex = require("./module_name");
Namespaces are useful when working with modules:
namespace NExample { export interface IExample {…} … }
To use same namespace for N modules use special comment before:
/// <reference path="module_name.ts" />
Aliasing namespaces:
import ex = NExample.CExample;
```