### Q1.1 Type Intersection (6 points)

What is the intersection of the types denoted by the expressions T1 and T2? Describe values in the type (give examples, characterize them) and write a TypeScript type expression that denotes this intersection.

1.1.1 T1 = {a:number[]; } T2 = {b:string};

Example:

let a : {a: number[]} & {b: string} = {a : [5,3], b:"hello"}

1.1.2 T1 = {a: {b: number}} T2 = {a: {c: string}};

Example:

let a : {a: {b: number}} & {a: {c: string}} =   
{a :{b: 5, c: 'hi"'}}

1.1.3 T1 = {a: number[]; } T2 = {a: number};

T1 and T2 are disjoint sets. The only values that can   
satisfy both conditions are undefined and null.

let a: = {a: undefined}

### Q1.2 Type Inclusion (6 points)

Are the following types T1 and T2 subsets one of another? Answer T1 < T2, or T2 < T1 or none. Justify your answer.

1.2.1

type T1 = {a:number, b:{}}[]

type T2 = {a:number}[]

T1 is a subset of T2. Every element in T1 has a property  
 a:number therefore it is an element of T2.

[{a=5}]is an element of T2 but as we can see doesn't have the property b:{} and is not an element of T2.

T1<T2

1.2.2

type T1 = {a: {c: any}, b:any}

type T2 = {a: {c: number}, b: number}

T2 is a sub set of T1.  
any is a super type that includes all kind of types.   
therefore, number is also an any. However, not all any is a number. For example: any that is a string is clearly not a number.

T2<T1

1.2.3

type T1 = {a: number, b: undefined}

type T2 = {a: number, b: any}

any can be undefined but undefined can only be an   
undefined.

5 is any (number) but is not undefined.

T1<T2

### Q1.3 Type Inference (8 points)

Write the most specific TypeScript type expression associated to each of the following expressions:

1.3.1 let v1 = { name:"peter", age:20 };

V1: {name: string, age: number}

1.3.2 v2 = {children: [ {name: "john"}, {age:12}]};

V2: {children: [{name: string},{age: number}]}

1.3.3 v3 = (x) => x + 2;

V3: {(x: number) => number} = (x)=>x+2

The arithmetic operation + can also be applied on strings, but we assumed that by adding the number 2 the reference is only for numbers.

1.3.4 v4 = (f, l) => map((x)=>f(f(x)), l);

The type is:

(f:(x: T) => T, l:T[])=> T[])

Ex:

let a :<T> (f:(x: T) => T, l:T[]) => T[] = (f, l) => map((x)=>f(f(x)))

### Q1.4 Type Definitions (4 points) //to do

1.4.1 Is it possible to define a type in TypeScript for the set of all strings with length larger than 2 using the type constructors defined in class?

There isn't an option in to define a type in TypeScript for the set of all strings with length larger than 2, according class material.

We saw 2 ways to define types in TypeScript:

1) **By using existing types, primitive or compound** – as seen in class a type that satisfies the 2 conditions does not exist. Even the primitive type 'string' is too inclusive.  
2) **Using literals (exact values) of all strings with length larger than 2** -The set of all strings (specifically with length larger than 2) is infinite, therefore the solution is impossible, we can't write infinite number of literals.

1.4.2 Is it possible to define a type for the set of all numbers larger than 0?

As previously mentioned.  
1)The primitive type 'number' is too inclusive.  
2)The set of all numbers larger than 0 is infinite.