functions

Different ways in which you can write fn.

1. **Named functions:**

```
function add (a,b) {
    return a+b
}
```
↓ javascript

```
function add (n1: number, n2: number)
{
  return n1 +n2
}
```
↓ typescript

Always have a return type:

```
function add (n1: number, n2: number): number
{
   return n1+n2
}

add(2,3);
```

2. **Arrow function**

```
const sub = (num1: number, num2: number) : number =>
                                          num1-num2
```

3. **Function expression**

```
const mult = function (num1: number, num2: number): number {
                return num1 x num2
}
```

Parameters:

1. **Optional Parameters.**

```
function add (a:number, b:number, c?:number)
{
    return c? a+b+c : a+b
}
```

2. **Required parameter.**

```
function add( a:number, b:number, c: number=10)
{
   return a+b+c;
}
```

3. **Rest Parameters** (rest of the parameters)

```
function add (a:number, b:number, ...c:number[])
{
   return a+b+ c.reduce ((a,b)=> a+b, 0);
```

```
}

add(2,3,...[1,2,5,7])
let nums = [1,2,5,7]
add(2,3,...nums)
```

Generic functions - works with generic types.

```
function getItems<Type>(items: Type[]): Type[] {
    return new Array<Type>().concat(items);
}

let res = getItems([1,2,3,4])
let ---- = getItems<number>([1,2,3,4])

let res2 = getItems<string>(["a","b"])
```

functions

Different ways in which you can write fn.

1. **Named functions:**

```
function add (a,b) {
    return a+b
}
```
↓ javascript

```
function add (n1: number, n2: number)
{
    return n1 + n2
}
```
↓ typescript

Always have a return type:

```
function add (n1: number, n2: number): number
{
    return n1+n2
}

add(2,3);
```

2. **Arrow function**

```
const sub = (num1: number, num2: number) : number =>
                num1 - num2
```

3. **function expression**

```
const mult = function (num1: number, num2: number): number {
                return num1 x num2
}
```

Parameters:

1. **Optional Parameters.**

```
function add ( a: number, b: number, c?: number)
{
```

```
    return c? a+b+c : a+b
}
```

2. Required parameter.

```
function add(a:number, b:number, c:number=10)
{
    return a+b+c;
}
```

3. Rest Parameters (rest of the parameters)

```
function add(a:number, b:number, ...c:number[])
{
    return a+b+c.reduce((a,b)=> a+b, 0);
}

add(2,3,...[1,2,5,7])
let nums = [1,2,5,7]
add(2,3,...nums)
```

Generic functions - works with generic types.

```
function getItems<Type>(items:Type[]):Type[] {
    return new Array<Type>().concat(items);
}

let = getItems<number>([1,2,3,4])
```