```
Functions
   Different ways in which you can write for.
 1. Named functions;
     function add (9,6) }
                               function add (ni: number, n2: number)
           return a+b
                                   return ni +n2
    Always have a return type:
     function add (ni: number, nz: number): number
        return n1+n2
    add(2,3);
  2 Amou function
    const sub = (numi: number, numz: number): number =>
                                         numi-num2
 3. Function expression
const mult = function (num 1: number, numz: number): number &
              return numixnum2
   Parameters:
  1. Optional Parameters.
     function add (ei:number, b:number, c?:number)
          return c? attote: att
      3
 2. Required parameter.
     function add (a: number, b: number, c: number=10)
         return attici,
      ζ
  3. Rest Parameters (rest of the parameters)
     function add (a: number, b: number, ... c: number ())
```

return a +6 + 'c reduce ((a,6) => a+6,0);

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

Queneric Functions - world with generic types.

function gettems = Type > (items: Type()): Type() }

return new Arroy = (): concat (items);

let res = gettems ((): 2,3,4))

-1: = gettems < number > ((): 2,3,4))

let res = gettems < string > (() "a", "b"))

## **Functions**

Different ways in which you can write for.

## · Named functions:

return a +b javascript

function add (q1b) } function add (n1: number, n2: number) return ni +n2

Always have a return type: function add (n: number, nz: number): number return n1+n2

add(2,3);

## 2 Amow function

const sub = (numi: number, numz: number): number => numi-numz

## 3. Function expression

const mult = function (num 1: number, numz: number): number & return numixnum2

Parameters:

1. Optional Parameters.

function add (ei:number, b:number, c?:number)

```
2. Required parameter.
   function add (a: number, b: number, c: number=10)
       return attic,
3. Rest Parameters (rest of the parameters) function add (a: number, b: number, ... c: number (3)
        return a +6 + 1c reduce ((a,6) => a+6,0);
    add ( 2,3, ... [1,2,5,7]
 let nums = [1,2,5,7)
    add (2,3, ... nums)
 aeneric functions - works with generic types.
  function getterne < Type > Citems: Type[] }
   return new Array (Type> (), concat (items);
            = gettems < number> ([1,2,3,4])
  <del>--</del>(1
```

return c? attote: att

3