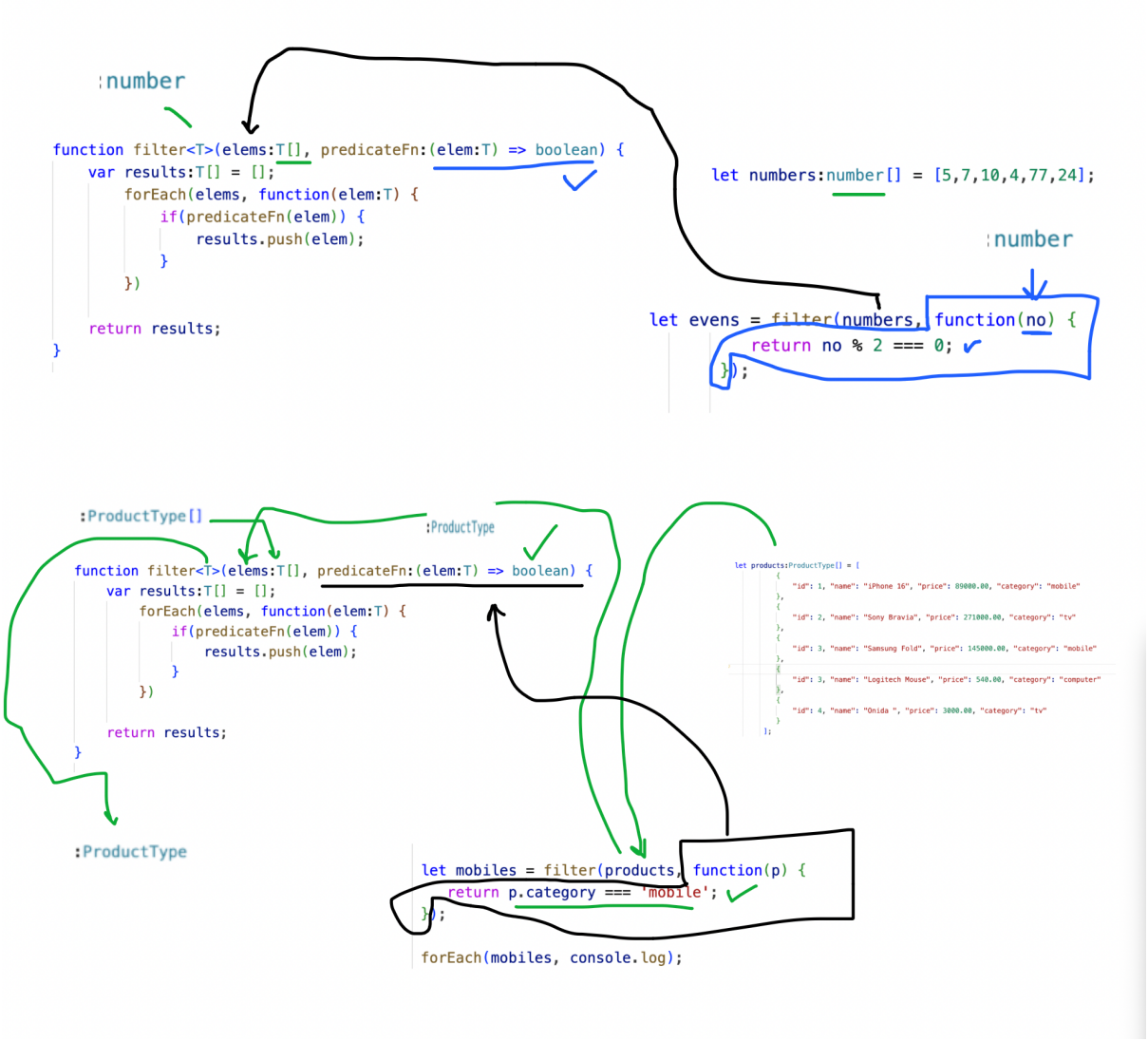
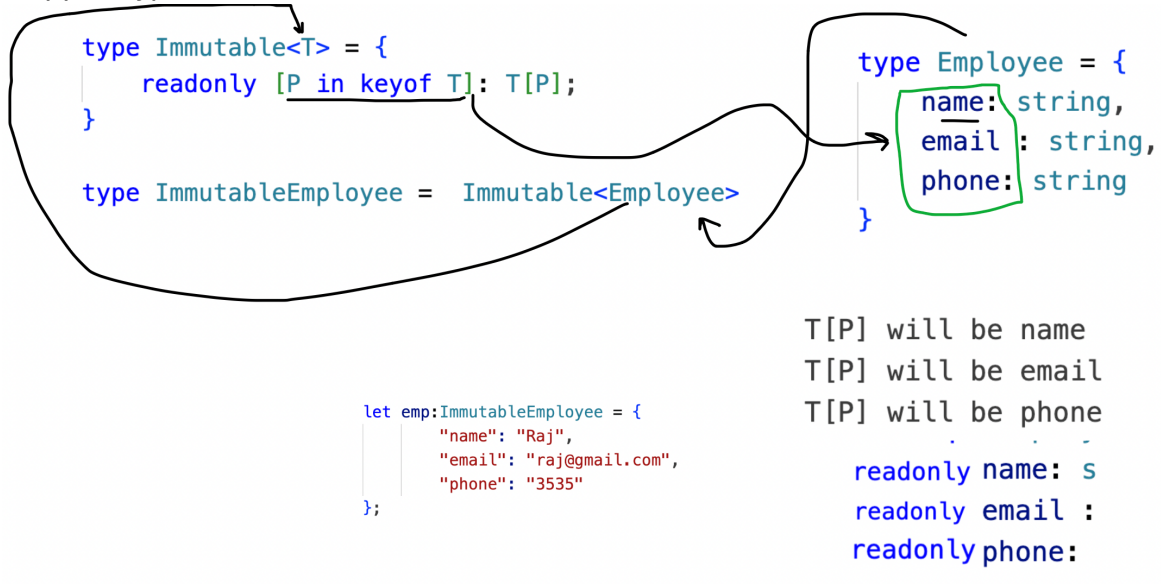
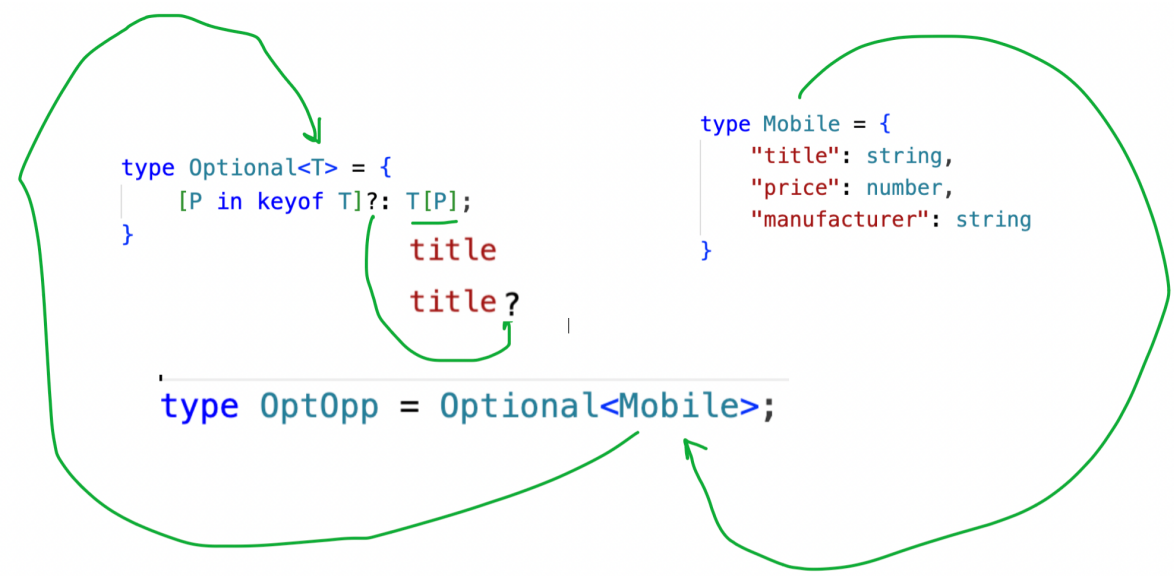
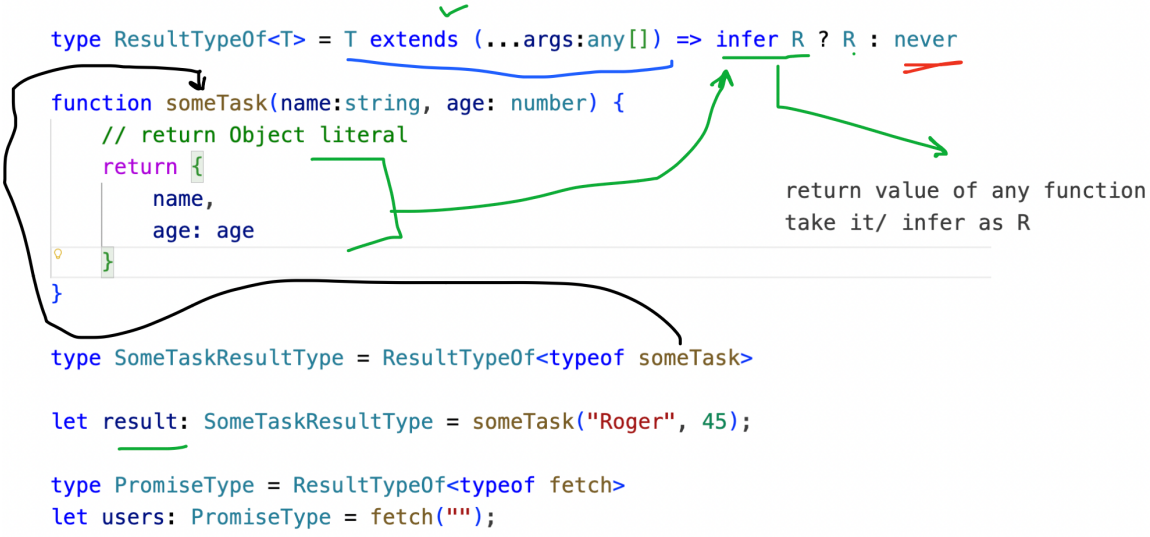

:number
function filter<T>(elems:T[], predicateFn:(elem:T) => boolean) {
var results:T[] = [];
forEach(elems, function(elem:T) {
if(predicateFn(elem)) {
results.push(elem);
}
})
return results;
}
let numbers:number[] = [5,7,10,4,77,24];
:number
let evens = filter(numbers, function(no) {
return no % 2 === 0;
});
:ProductType[]
:ProductType
function filter<T>(elems:T[], predicateFn:(elem:T) => boolean) {
var results:T[] = [];
forEach(elems, function(elem:T) {
if(predicateFn(elem)) {
results.push(elem);
}
})
return results;
}
:ProductType
let products:ProductType[] = [
{
"id": 1, "name": "iPhone 16", "price": 89000.00, "category": "mobile"
},
{
"id": 2, "name": "Sony Bravia", "price": 271000.00, "category": "tv"
},
{
"id": 3, "name": "Samsung Fold", "price": 145000.00, "category": "mobile"
},
{
"id": 3, "name": "Logitech Mouse", "price": 540.00, "category": "computer"
},
{
"id": 4, "name": "Onida ", "price": 3000.00, "category": "tv"
}
];
let mobiles = filter(products, function(p) {
return p.category === 'mobile';
});
forEach(mobiles, console.log);

Mapped Type


type Immutable<T> = {
readonly [P in keyof T]: T[P];
}
type ImmutableEmployee = Immutable<Employee>
type Employee = {
name: string,
email : string,
phone: string
}
let emp:ImmutableEmployee = {
"name": "Raj",
"email": "raj@gmail.com",
"phone": "3535"
};
T[P] will be name
T[P] will be email
T[P] will be phone
readonly name: s
readonly email :
readonly phone:


type Optional<T> = {
[P in keyof T]?: T[P];
}
title
title?
type Mobile = {
"title": string,
"price": number,
"manufacturer": string
}
type OptOpp = Optional<Mobile>;


type ResultTypeOf<T> = T extends (...args:any[]) => infer R ? R : never
function someTask(name:string, age: number) {
// return Object literal
return {
name,
age: age
}
}
return value of any function
take it/ infer as R
type SomeTaskResultType = ResultTypeOf<typeof someTask>
let result: SomeTaskResultType = someTask("Roger", 45);
type PromiseType = ResultTypeOf<typeof fetch>
let users: PromiseType = fetch("");