Exercise : write a function `stringToInt` that takes `aString` parameter and returns a number associated to this string. It should be *tail-recursive*. Every non-numeric char found in the string makes the function return `undefined`. The function should return `0` on an empty string.

Information for better readability : the ASCII char code of '0' is 48. The comments have been removed from all codes but the reference.

For reference, this is the teacher's solution :

```
function stringToInt(aString) {
    const charCode0 = "0".charCodeAt(0); // Computed only once
    if (aString === "")
        return 0;
    else {
        function stringToIntRec(aCpt, aString) {
            if (aString === "")
                return aCpt;
            else {
                const aTail = aString.substring(1);
                const aHead = aString.charCodeAt(0) - charCode0;
                if ((aHead < 0) || (aHead > 9))
                    return undefined;
                else
                    return stringToIntRec(aCpt * 10 + aHead, aTail);
            }
        }
        return stringToIntRec(0, aString);
    }
}
```

In the following are given *all* the answers to that exercise, and a small text explaining whether the function is correct or not. No personal information has been give, this is not a matter of shaming anybody, this is a matter of demontrating bad code and good code.

Some remarks :

- Only 2 functions on the 14 are *functionally* correct, i.e they correspond to the specification. They may not be optimal in terms of complexity or clarity of the code, though.
- In all cases, sufficient testing would have detected the problems in the functions. Here is a possible set of test cases and their answers :

Argument	Expected result
`""`	`0`
`"1"`	`1`
`"1234567890"`	`1234567890`
`"1."`	`undefined`
`"One"`	`undefined`

In the following function, the variable `total` is never defined :

```
function stringToInt(aString) {
    function stringToIntRec(aString,index){
        if(aString.charCodeAt(0) < 48 || aString.charCodeAt(0) > 57 ){
            return undefined;
        };
        if(index === ""){
            return total;
        };
        total+= (aString.charCodeAt(0)-48)*Math.pow(10,index);
        stringToIntRec(aString.substring(index),index + 1);
    }
    stringToIntRec(aString,0);
    return total;
}
```

The following one has a strange idea of the acceptable ASCII chars (and accepts the nefast `'.'` character as valid). And it returns the number of recursive calls instead of the value requested.

```
function stringToInt(aString, res=0 ) {
    if (aString === "") {
        return res;
    }
    if (aString.charCodeAt(0) > 127 || aString.charCodeAt(0) < 65) {
        return undefined;
    }
    else {
        return stringToInt(aString.substring(1), res+1); // appels récursifs terminaux
    }
}
```

The following functions uses an array to return results instead of a number, and makes strange modular computations.

```
function stringToInt(aString) {
    let res = [];
    function stringToIntRec(aString, result) {
        if (aString === "") {
            return result.push("0".charCodeAt(0) % "0".charCodeAt(0));
        }
        else if (aString.charCodeAt(0) < "0".charCodeAt(0) ||
                 aString.charCodeAt(0) > "9".charCodeAt(0)) {
            return undefined;
        }
        else {
            result.push(aString.charCodeAt(0) % "0".charCodeAt(0));
            return stringToIntRec(aString.substring(1), result);
        }
    }
    return stringToIntRec(aString, res);
}
```

The following function bizarrely disregards digits over 6 :

```
function stringToInt(aString) {
    function stringToIntRec(aString, aInt){
        if (aString.length === 0)
            return aInt;
        else
            if (aString.charCodeAt(0) < 48 ||
                aString.charCodeAt(0) > 53)
                return "undefined";
            else
                return stringToIntRec(aString.substring(0),
                        aInt * 10 + (aString.charCodeAt(0) -48) );
    }
    return stringToIntRec(aString, 0);
}
```

The following function returns 0 or undefined (even if we add the missing return before the last call to `stringToIntCheck`) :

```
function stringToInt(aString) {
    function stringToIntCheck(aString)  {
        if (aString === "")  {
            return 0;
        }
        else if (aString.charCodeAt() < 48 || aString.charCodeAt() > 57)  {
            return undefined;
        }
        else  {
            stringToIntCheck(aString.substring(1));
        }
    }

    return stringToIntCheck(aString);
}
```

The following function is seemingly correct, but in fact uses a strange boolean test that does `a <= b <= c` which is understood as `(a <= b) <= c` where `true` equals `1`, meaning that any char code `>= 48` is accepted.

```
function stringToInt(aString) {
    function stringToInt_rec(aString, val) {
        if ( aString.length === 0 )
            return val;
        else if ( 48 <= aString.charCodeAt(0) <= 57 )
        {
            let l = aString.length;
            let n = aString.charCodeAt(0)-48;
            return stringToInt_rec(aString.substring(1),val + (n * (10 ** (l -1 ))));
        }
    }
    return stringToInt_rec(aString,0);
}
```

The following function also has terrible duplication and is not recursive, hence returns nonsense on strings of length > 1 :

```
function stringToInt(aString) {
    if (aString.length === 0)
        return 0;
    else
    {
        let c1 = aString[0];
        switch (c1) {
            case('0'):
                return aString.substring(1);
            case('1'):
                return 10**(aString.length -1) + aString.substring(1);
            case('2'):
                return 2*(10**(aString.length -1)) + aString.substring(1);
            case('3'):
                return 3*(10**(aString.length -1)) + aString.substring(1);
            case('4'):
                return 4*(10**(aString.length -1)) + aString.substring(1);
            case('5'):
                return 5*(10**(aString.length -1)) + aString.substring(1);
            case('6'):
                return 6*(10**(aString.length -1)) + aString.substring(1);
            case('7'):
                return 7*(10**(aString.length -1)) + aString.substring(1);
            case('8'):
                return 8*(10**(aString.length -1)) + aString.substring(1);
            case('9'):
                return 9*(10**(aString.length -1)) + aString.substring(1);
            default:
                return undefined;
        }
    }
}
```

The following function uses methods associated to no object in particular, like `charCodeAt` or `substring` (and so does not work on *any* example that is not the empty string) :

```
function stringToInt(aString) {
    function aux(aString, acc) {
        if (aString === ""){
            return acc;
        }
        if (aString[0] >= "0" & aString[0] <= "9"){
            let temp = charCodeAt(aString[0]);
            return aux(substring(aString), 10*acc + temp);
        }
        else {
            return undefined;
        }
    }
    return aux(aString, 0);
}
```

This function contains an unused subfunction, and always returns 0 :

```
function stringToInt(aString) {
    function isNumber(substring){
        if(substring.charCodeAt() >= 48 && substring.charCodeAt() <= 57){
            return substring;
        }
        return undefined;
    }
    if (aString === ""){
        return 0;
    }
    else {
        return stringToInt(aString.substring(1));
    }
}
```

This function is nearly correct, but has strangely decided to disregard all strings containing 0s, internal or not :

```
function stringToInt(str) {
    function rec(str,res,vide){
      if(str.length===0 && vide===0) return 0;
      if(str.length===0) return res;
      if(str[0].charCodeAt()>=49 && str[0].charCodeAt()<=57) {
        return rec(str.slice(1),
                   res*10+(str[0].charCodeAt()-49+1),
                   vide+1);
      }
      else return undefined;
    }
    return rec(str,0,0);
}
```

This one just does nothing really correct if the string is non-empty :

```
function stringToInt(aString) {
    if (aString===''){
        return 0;
    }
    else {
        console.log(aString.charAt(0));
        return ( aString.substring(1));
    }
}
```

This function accepts all possible strings, and also returns a string :

```
function stringToInt(aString) {
    function stringToIntRec(aString, acc) {
        if (aString !== '')
            {const aChar = (aString[0]).charCodeAt()
            return stringToIntRec(aString.substring(1), `${acc+aChar%48}`)}
        else
            return acc
    }
    return stringToIntRec(aString, 0);
}
```

The following function is functionally correct *but* has a horrible duplication of values, does subtraction on strings (and that works in Javascript, but is highly discouraged because addition of strings return strings whereas subtraction converts into numbers then returns numbers). Also, it does much more exponentiations than is necessary :

```
function stringToInt(aString) {
    function stringToIntrec(aString, num, length){
    if(aString === ""){
            return num;
    }
    if(aString[0] !== "0" &&
       aString[0] !== "1" &&
       aString[0] !== "2" &&
       aString[0] !== "3" &&
       aString[0] !== "4" &&
       aString[0] !== "5" &&
       aString[0] !== "6" &&
       aString[0] !== "7" &&
       aString[0] !== "8" &&
       aString[0] !== "9"){return undefined;}
    let Num = aString[0] - "0";
    let String = aString.substring(1);
    return stringToIntrec(String, num+Num*10**(length-1),length-1);
    }
    return stringToIntrec(aString, 0, aString.length);
}
```

This function is convoluted, the names are not very good, and it does too much multiplications by 10, but it is functionally correct :

```
function stringToInt(aString) {
  function stringToIntTer(aString, sum, i) {
    function convert(char) {
      const ascii = char.charCodeAt(0);
      return ascii -48;
    }
    if (aString.length===0) return sum;
    const ascii= convert(aString[0]);
    if (ascii < 0 || ascii > 9)
      return undefined;
    else
      return stringToIntTer(aString.slice(1), sum+10**i*ascii, i-1 );
  }
  return stringToIntTer(aString,0,aString.length - 1);
}
```

For reference, the following function was written in Haskell (and kindly provided by Mr Janin). It makes elegant use of the pattern-matching possibilities and of the option types. The `fromEnum` function is used here to convert a character into its integer ASCII code. Apart from that, the code is not very different from the Javascript reference.

```haskell
stringToInt :: String -> Maybe Int
stringToInt s = stringToIntRec (0,s)
  where
    stringToIntRec (aCpt, "")  = Just aCpt
    stringToIntRec (aCpt, c:s) =
        let d = fromEnum c - charCode0 in
        if (0 <= d) && (d <= 9)
        then stringToIntRec (aCpt*10 + d, s)
        else Nothing
    charCode0 = fromEnum '0'
```