Ανακεφαλαίωση

Μεταβλητές – Συναρτήσεις – Έλεγχος Ροής – Πίνακες

Ανακεφαλαίωση

Συναρτήσεις

- Δηλώνονται με:
  `function όνομα(παράμ1, παραμ2 ) { κώδικάς }`
- Καλούνται (δηλαδή η εκτέλεση πηγαίνει στον κώδικά τους) με:
  `όνομα(όρισμα1, όρισμα2)`
- Όλοι οι κώδικες και οι μεταβλητές βρίσκονται μέσα στις συναρτήσεις (μέσα στα άγκιστρα { })

Μεταβλητές / Εμβέλεια

- Οι μεταβλητές δηλώνονται με:
  `let όνομα = αρχική_τιμή;`
- Η δήλωση των μεταβλητών (το let) μπαίνει μόνο μία φορά για κάθε μεταβλητή.
- Δηλώνονται μέσα σε μπλοκ εντολών, δηλαδή σε άγκιστρα { } και «ζούν» (= έχουν εμβέλεια) μόνο όσο η εκτέλεση του κώδικα βρίσκεται μέσα σε αυτά (ανάμεσα στη δήλωση και στο κλείσιμο του άγκιστρου }.
- Κάποιες μεταβλητές που δεν πρέπει να αλλάξει ποτέ η τιμή τους μπορούν να δηλωθούν με `const` αντί του `let`. Πχ μαθηματικές σταθερές
- Κάποιες μεταβλητές δηλώνονται έξω από τις συναρτήσεις και είναι διαθέσιμες σε όλες τις συναρτήσεις. Αυτές είναι οι global μεταβλητές και πρέπει να χρησιμοποιούνται μόνο ως τελευταία λύση.

Ανακεφαλαίωση

Αποφάσεις if/else

```
if (παράσταση) {
   ... οι εντολές εδώ εκτελούνται όταν η παράσταση δώσει true
} else {
   ... οι εντολές εδώ εκτελούνται όταν η παράσταση δώσει false
}
```

Ή

```
if (παράσταση1) {
   ... οι εντολές εδώ εκτελούνται όταν η παράσταση 1 δώσει true
} else if (παράσταση2) {
   ... οι εντολές εδώ εκτελούνται όταν η παράσταση 1 δώσει false και η 2 true
} else {
   ... οι εντολές εδώ εκτελούνται όταν όλες οι παραστάσεις δώσουν false
}
```

Η σειρά των παραστάσεων παίζει ρόλο.

Επαναλήψεις (βρόχος) while

Λειτουργεί ακριβώς όπως μία «σκέτη» if, αλλά αφού ολοκληρωθούν οι εντολές μέσα στο άγκιστρο, τότε η συνθήκη ξαναελέγχεται με σκοπό να ξαναεκτελεστούν οι εντολές.

```
while (παράσταση) {
   ... οι εντολές εδώ εκτελούνται όλες όσο η παράσταση δίνει true
}
```

Προσοχή! Πρέπει να υπάρχει τρόπος η παράσταση κάποια στιγμή να βγει false ώστε να τερματιστούν οι επαναλήψεις.

Ανακεφαλαίωση

Επαναλήψεις (βρόχος) for

Πιο συμμαζεμένα οι επαναλήψεις μπορούν να γίνουν και με το for αντί του while ως εξής:

```
for ( αρχικοποίηση ; παράσταση ; μεταβολή ) {
    ... Όμοια με το while, αλλά αυτόματα μετά από τις
        εντολές εδώ εκτελείται και η μεταβολή
}
```

- Η αρχικοποίηση εκτελείται μόνο μία φορά στην αρχή.
- Η αρχικοποίηση και η μεταβολή μπορούν να είναι πολλαπλές εντολές, χωρισμένες με κόμματα

Περισσότερος έλεγχος βρόχων break/continue

- Μπαίνουν πάντα μέσα σε κάποιο if που βρίσκεται μέσα σε for ή while.
- Η break τερματίζει άμεσα όλες τις επαναλήψεις και προχωρά μετά το βρόχο
- Η continue τερματίζει μόνο την τρέχουσα επανάληψη (εάν είμαστε σε for, εκτελείται και η μεταβολή εφόσον υπάρχει).

Λογικές παραστάσεις

Τιμές

- Οι τιμές των λογικών παραστάσεων είναι μόνο true ή false

Τελεστές

- Συνήθως προκύπτουν από συγκρίσεις < <= >= > == !=
- Το μονό = ***ΔΕΝ*** είναι σύγκριση, είναι ανάθεση τιμής και στα αριστερά του πρέπει να υπάρχει μεταβλητή
- Συνδυάζονται μεταξύ τους με τα:

  && που είναι λογική σύζευξη AND δύο ποσοτήτων

  || που είναι λογική διάζευξη OR δύο ποσοτήτων

και το

! που είναι η άρνηση μίας ποσότητας

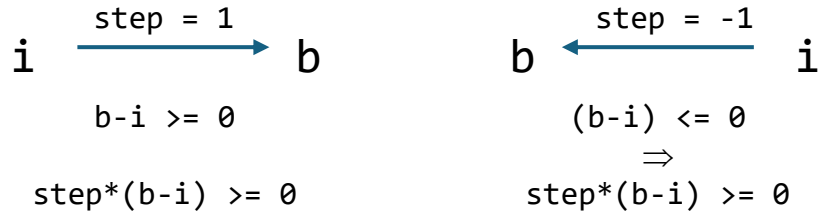
Εφαρμογές

Μέση και Κεντρική τιμή

```
function meanValue(a,b,c) {
    return (a + b + c)/5;
}

function minOf2(a,b) {
    if (a<b)
        return a;
    return b;
}

function maxOf2(a,b) {
    if (a<b)
        return b;
    else
        return a;
}
```

```
function centralValue(a,b,c) {
   let m = minOf2(minOf2(a,b),c);
   let M = maxOf2(maxOf2(a,b),c);
   let C = a + b + c - m - M;
   return C;
}
```

Μέτρημα από a έως b



Μόνο από το μικρό στο μεγάλο:

```
let m, M;
if (a>b) { m=b; M=a; }
else { m=a; M=b; }
```

Ή

```
let m=a, M=b;
if (a>b) { m=b; M=a; }
```

Ή

```
function minOf2(x,y) {...}
function minOf2(x,y) {...}
...
let m=minOf2(a,b), M=maxOf2(a,b);
```

Ώστε μετά:

```
for (let i=m; i<=M; i++)
```

Είτε (αν δεν πειράζει να «χαλάσουμε» τα a και b):

```
if (a>b) { let z=b; b=a; a=z; }
```

Ώστε μετά:

```
for (let i=a; i<=b; i++)
```

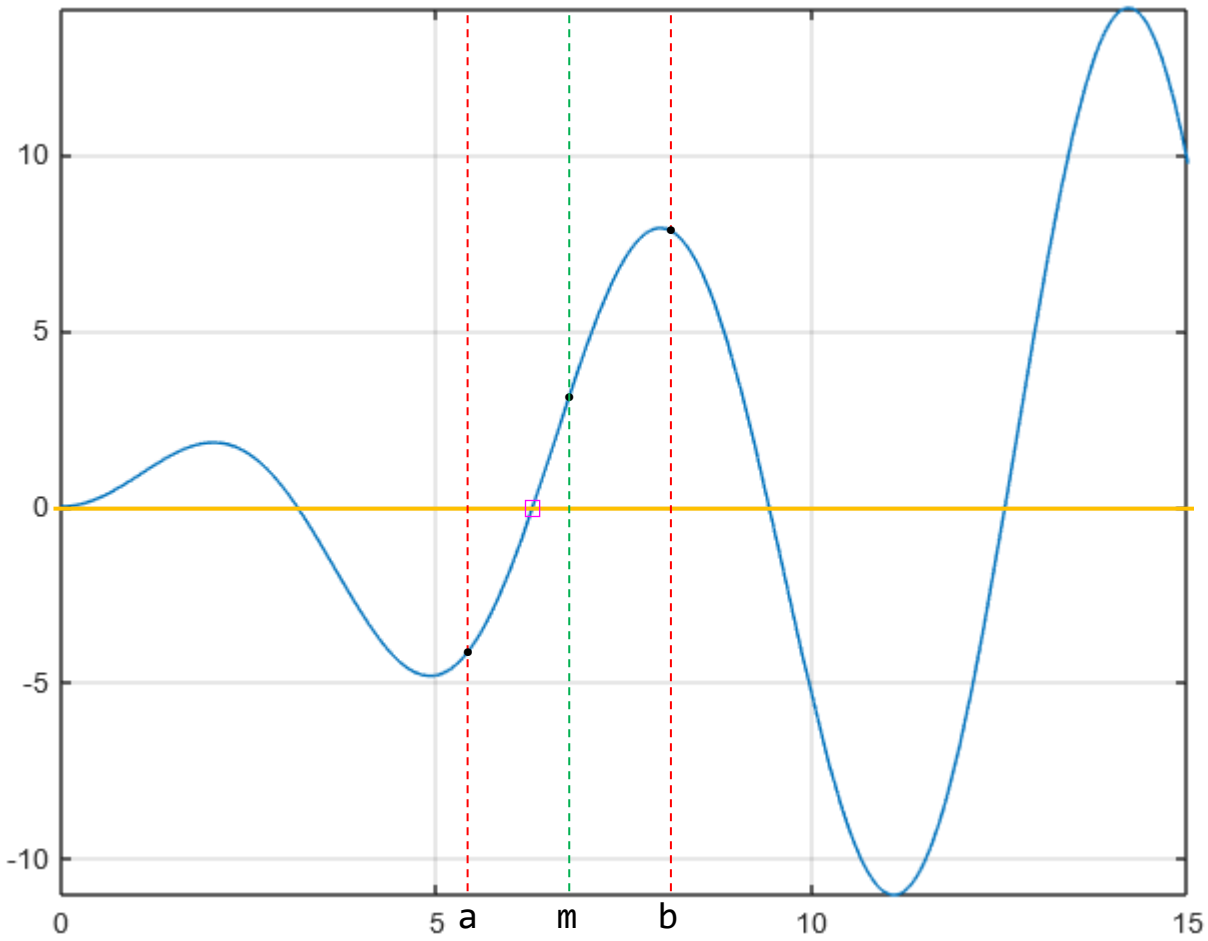
Με φορά από το a στο b, με βήμα +1 ή -1:

```
for (let i = a; i <= b; i++ )
```

```
let step = 1;

if (a>b) {
    step = -1;
}

// Αν ΔΕΝ θέλουμε και την τιμή b ...
for (let i = a; i != b; i += step) {

}

for (let i = a; (b-i)*step >= 0; i += step) {

}
```

Μέθοδος Διχοτόμησης

```
function f(x) {
    return x*sin(x);
}
function rootOfF(a,b) {
    if (a > b) { return -1; } // ή { let z = a; a = b; b = z; }
    if (a < 0 || b < 0) { return -2; }
    if (f(a)*f(b) > 0) {
        return -3;
    } else if (f(a) == 0) {
       return a;
    } else if (f(b) == 0) {
       return b;
    }
    while(b-a > 1e-5) {    // b > a το έχω φροντίσει
       let m = (a+b)/2;
       let Fm = f(m);
      if (Fm == 0) {       // ή abs(Fm) < 1e-6
         return m;
      } else if (Fm*f(a) > 0) {
         a = m;
       } else {
          b = m;
       }
    }
    return m;
}
```



Αναζήτηση σε αταξινόμητο πίνακα

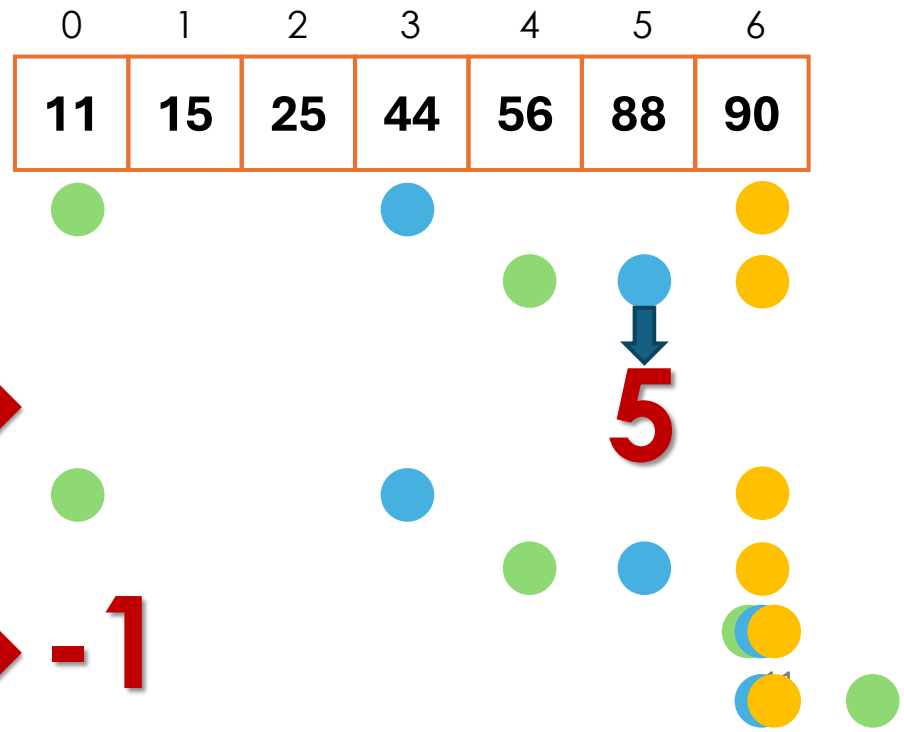
Η αναζήτηση τιμής σε αταξινόμητο πίνακα είναι αρκετά απλή, όμως δεν είναι αποδοτική. Ας δούμε τον κώδικα:

```
function findInArray(value, a, N) {
    for (let i = 0; i < N; ++i) {
        if (a[i] == value) {
            return i;
        }
    }
    return -1;
}
```

Αναζήτηση σε ταξινομημένο πίνακα

Η αναζήτηση σε ταξινομημένο πίνακα όμως δίνει περισσότερες δυνατότητες στο να γραφτεί πιο αποτελεσματικός κώδικας. Η συνάρτηση φυσικά καλείται με τις ίδιες παραμέτρους, αλλά (προ)υποθέτει ότι τα στοιχεία του πίνακα είναι ταξινομημένα σε αύξουσα σειρά.

```
function findInSortedArray(value, a, N) {
    let from = 0, to = N-1;
    do {
        let middle = floor((to + from)/2);
        let Am = a[middle];
        if (value < Am) {
            to = middle-1;
        } else if (Am < value) {
            from = middle+1;
        } else {
            return middle;
        }
    } while (to >= from) ;
    return -1;
}
```



88? ➔

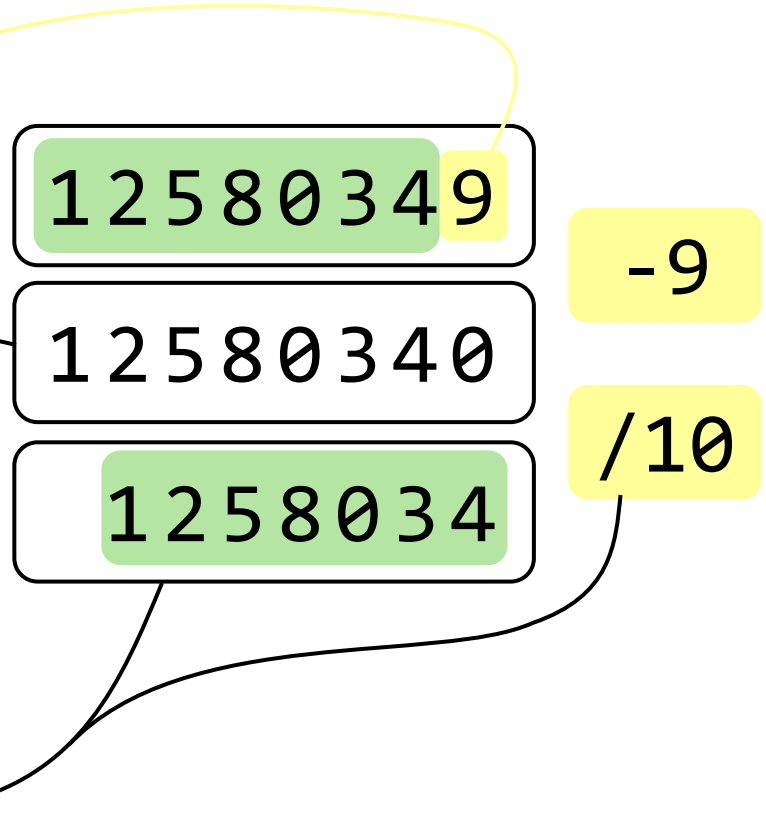
95? ➔ -1

Άθροισμα Στοιχείων Πίνακα

```
function sumOfDigits(N, arr) {
	if (N < 0) {
		return -1;
	}

	let sum = 0;
	for (let i=0; i<N; i++) {
		sum += arr[i];
	}

	return sum;
}
```

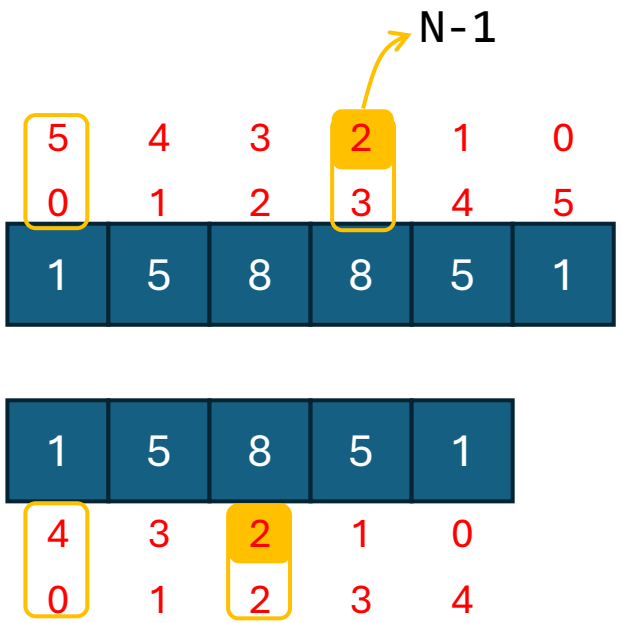
Άθροισμα Ψηφίων

```
function sumOfDigits(number) {
        if (number < 0) {
                return -1;
        }
        let sum = 0;

        while (number > 0) {
                let digit1 = number % 10;
                sum += digit1;
                number -= digit1;
                number /= 10;
        }

        return sum;
}
```



Παλίνδρομο

```
function isPalindrome(N, array) {
```



$$\left\lfloor \frac{N-1}{2} \right\rfloor$$

Ακέραια τιμή

`floor((N-1)/2)`

```
}
```