# Εισαγωγή στην Πληροφορική & στον Προγραμματισμό

Αρχές Προγραμματισμού Η/Υ (με τη γλώσσα C)

Διάλεξη #5

Παρασκευή 19 Απριλίου 2024

Παναγιώτης Παύλου

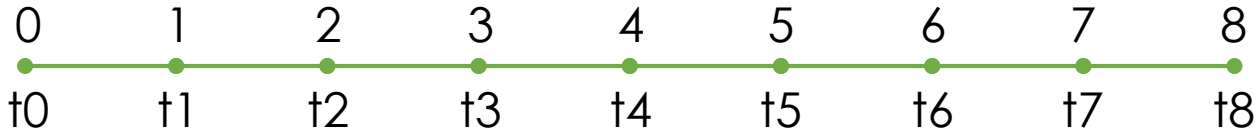
c-programming-24@allos.gr

# Πίνακες

Πίνακες δεδομένων μίας και περισσοτέρων διαστάσεων

# Μονοδιάστατοι πίνακες

Αν υποθέσουμε ότι θέλουμε – με όσα γνωρίζουμε μέχρι τώρα – να αποθηκεύσουμε τις θερμοκρασίες ενός ψυγείου μήκους 2μ ανά 25εκ.

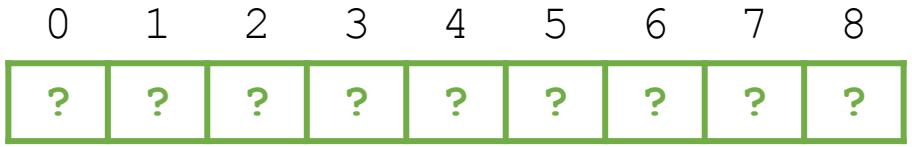



Θα χρειαστούν 9 μετρήσεις γι'αυτό θα πρέπει να χρησιμοποιήσουμε 9 μεταβλητές, τις t0, t1, t2, …, t7, t8 οπότε κάθε τι που θα γράφουμε θα πρέπει να τις αναφέρει ρητά π.χ.

```
let average = (t0 + t1 + t2 + t3 + t4 + t5 + t6 + t7 + t8)/9;
```

αφού δεν μπορούμε να δημιουργήσουμε παραμετρικές παραστάσεις. Λύση δίνουν οι πίνακες!

Οι **πίνακες (arrays)** είναι μία ομάδα μεταβλητών, κάτω από **κοινό όνομα** και κάθε μία από αυτές αναφέρεται μέσω ενός **δείκτη (index)**. Κάθε μία από τις μεταβλητές ονομάζεται **στοιχείο** του πίνακα και είναι προσδιορίζεται βάσει του δείκτη.




Η αρίθμηση **ξεκινά από το 0 μέχρι το Ν-1** (όπου Ν το πλήθος των στοιχείων του πίνακα).

# Μονοδιάστατοι πίνακες

Το Ν το αποκαλούμε και **μέγεθος** ή **μήκος** του πίνακα. Το μήκος του πίνακα είναι προκαθορισμένο και ορίζεται κατά τη δήλωση του πίνακα, ενώ στη συνέχεια δεν μπορεί να μεταβληθεί.

Η δήλωση ενός πίνακα με όνομα x με μήκος Ν γράφεται ως ακολούθως:

```
let x[N];
```

και το στοιχείο του πίνακα x με δείκτη i γράφεται ως:

```
x[i]
```

και αυτό λειτουργεί όπως οποιαδήποτε μεταβλητή. Αυτό σημαίνει ότι οι τιμές των στοιχείων είναι ακαθόριστες κατά τη δήλωσή του πίνακα και πρέπει να αρχικοποιηθούν.

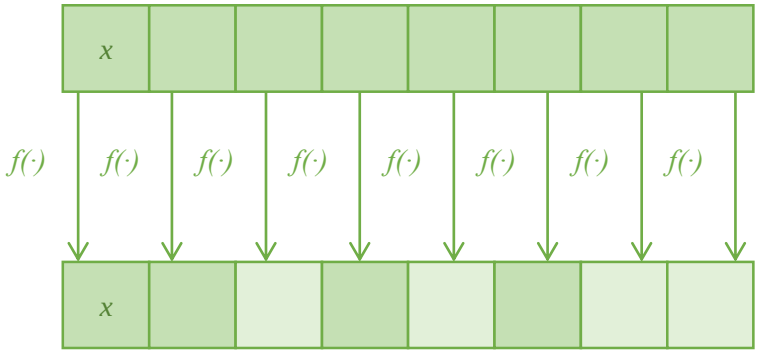
Η αρχικοποίηση του πίνακα μπορεί να γίνει κατά τη δήλωσή του με τον ακόλουθο τρόπο.

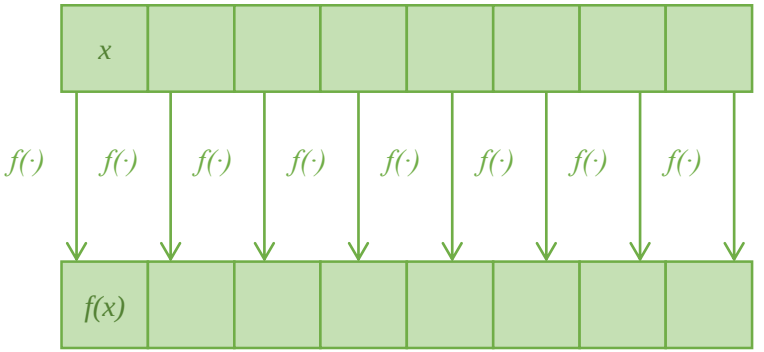
```
let x[N] = { 1,12,58,44,20,-3,0 };
```

Εάν τα στοιχεία είναι λιγότερα από το μέγεθος του πίνακα, τότε οι τιμές αποδίδονται στα πρώτα στοιχεία του και τα υπόλοιπα τα θεωρούμε μη αρχικοποιημένα. Εάν τα στοιχεία μέσα στα άγκιστρα είναι όλα τα στοιχεία του πίνακα, τότε το μέγεθος του πίνακα μπορεί να παραληφθεί.

# Μονοδιάστατοι πίνακες

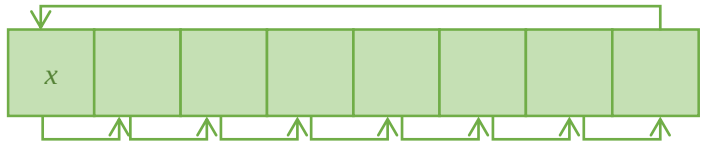
- **Ενδεικτικά** οι πίνακες μπορούν να χρησιμοποιηθούν για να παραστήσουν :
  - Διανύσματα
    - Συντεταγμένες
    - Κατάσταση ενός συστήματος
  - Ακολουθίες
    - χωρικές μεταβολές μεγεθών πχ θερμοκρασίας
    - χρονικές μεταβολές μεγεθών πχ παραμόρφωσης
    - γονιδίωμα
    - κυματομορφή ήχου
    - μετρήσεις αισθητήρων
    - κινήσεις παρτίδας παιχνιδιού πχ σκάκι, τρίλιζα
    - κείμενα (ως ακολουθίες γραμμάτων)
  - Σύνολα
    - θέση αντικειμένων σε αποθήκη
    - περιεχόμενα μιας κούτας
    - αριθμοί που έχουν κάποια ιδιότητα πχ πρώτοι αριθμοί, μέγιστες τιμές

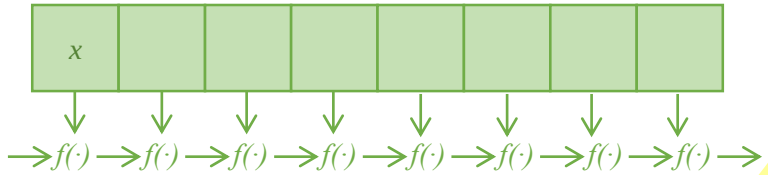



- Η χρήση των πινάκων είναι ευρύτατη και είναι η βασικότερη δομή δεδομένων πέρα από τις μεταβλητές, η οποία υποστηρίζεται και από τον ίδιο τον επεξεργαστή.

- Σχήματα επεξεργασίας των πινάκων :
  - Filter – Φιλτράρισμα
  - Map – Αντιστοίχιση
  - Reduce – Αναγωγή
  - Rotation – Περιστροφή

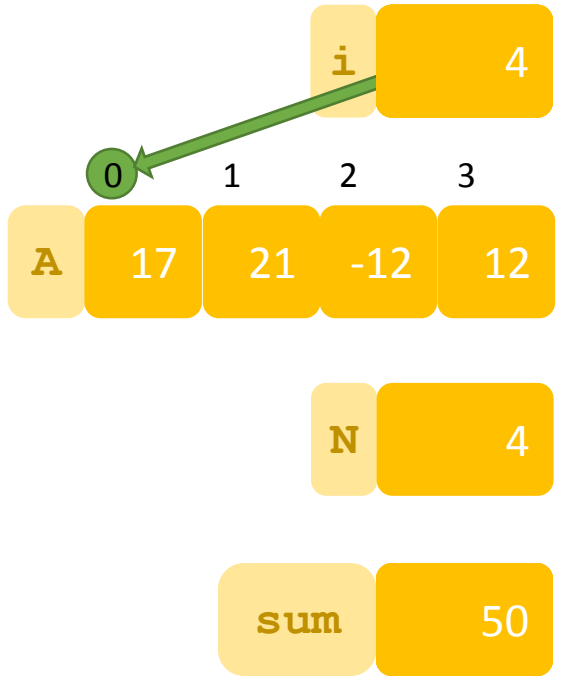


# Εφαρμογή – Άθροισμα θετικών

Ο παρακάτω κώδικας υπολογίζει το άθροισμα των θετικών στοιχείων του πίνακα A με μήκος N.

```
function sumPositives(A, N) {
    let sum = 0;
    for (let i=0; i<N; i++) {
        if (A[i] > 0) {
            sum += A[i];
        }
    }
    return sum;
}
```



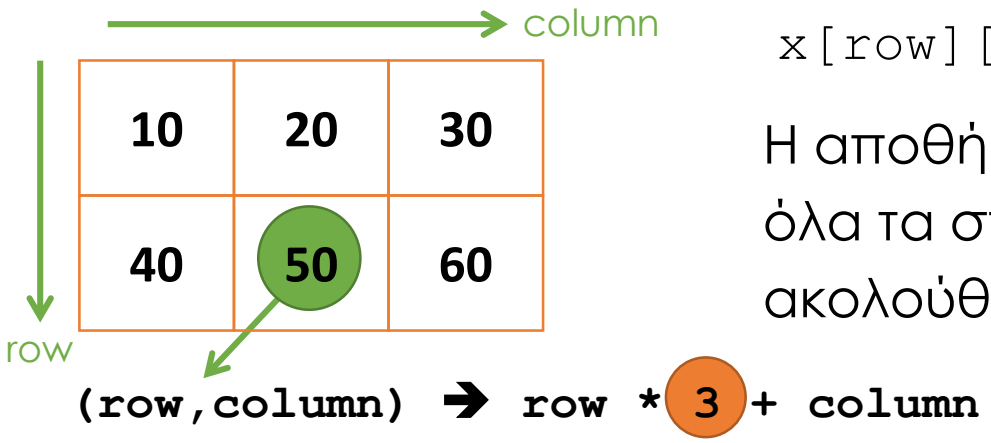
# Πολυδιάστατοι πίνακες

Αφού κάθε στοιχείο ενός πίνακα μπορεί να είναι οποιουδήποτε τύπου δεδομένων, τότε δεν μας απαγορεύει τίποτα να δημιουργήσουμε έναν πίνακα από πίνακες. Αυτό ισοδυναμεί με τη δημιουργία ενός δισδιάστατου πίνακα.

Η δήλωσή του γίνεται παρόμοια με τους απλούς μονοδιάστατους πίνακες ως εξής:

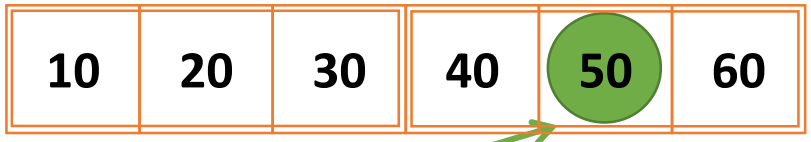
```
let x[2][3];
```

Που αντιστοιχεί σε έναν πίνακα 2 γραμμών και 3 στηλών, ενώ η αναφορά στο στοιχείο του στη γραμμή `row` και στη στήλη `column` γίνεται ως:




```
x[row][column]
```

Η αποθήκευσή του στη μνήμη είναι συνεχόμενη για όλα τα στοιχεία του και γίνεται όπως φαίνεται ακολούθως.



# Πολυδιάστατοι πίνακες

Τα στοιχεία του δισδιάστατου αρχικοποιουνται ως εξής:

```
let x[2][4] = { { 1,2,3,4 } , { 5,6,7,8 } };
```

Εδώ δεν μας επιτρέπεται να παραλείψουμε τις διαστάσεις του πίνακα όταν δίνονται όλα τα στοιχεία του.

Αντίστοιχα με την ίδια λογική μπορούμε να ορίσουμε και πίνακες περισσότερων διαστάσεων. Π.χ.

```
int y[10][10][10];
```

# Εφαρμογή printArray 2D

```
function print2DArray(R, C, A) {
    for (let r = 0; r < R; ++r) {
        for (let c = 0; c < C; ++c) {
            smPrint("% ", A[r][c]);
        }
        smPrint("\n");
    }
}
```

# Αρχικοποίηση 2D πίνακα

```
function SetToZeros(R, C, A) {
    for (let r = 0; r < R; ++r) {
        for (let c = 0; c < C; ++c) {
            A[r][c] = 0.0;
        }
    }
}
```

**Προσοχή** : Πως γίνεται να μεταβάλλονται τα περιεχόμενα του ορίσματος αφού το Call By Value το προστατεύει; Προσέξτε το αυτό, συγκρατήστε το στη μνήμη σας και κάντε υπομονή το για επόμενη διάλεξη. Είναι ένα εγγενές ζήτημα των υπολογιστών.

# Εφαρμογές πινάκων

Αναζήτηση και ταξινόμηση

# Αναζήτηση σε ταξινόμητο πίνακα

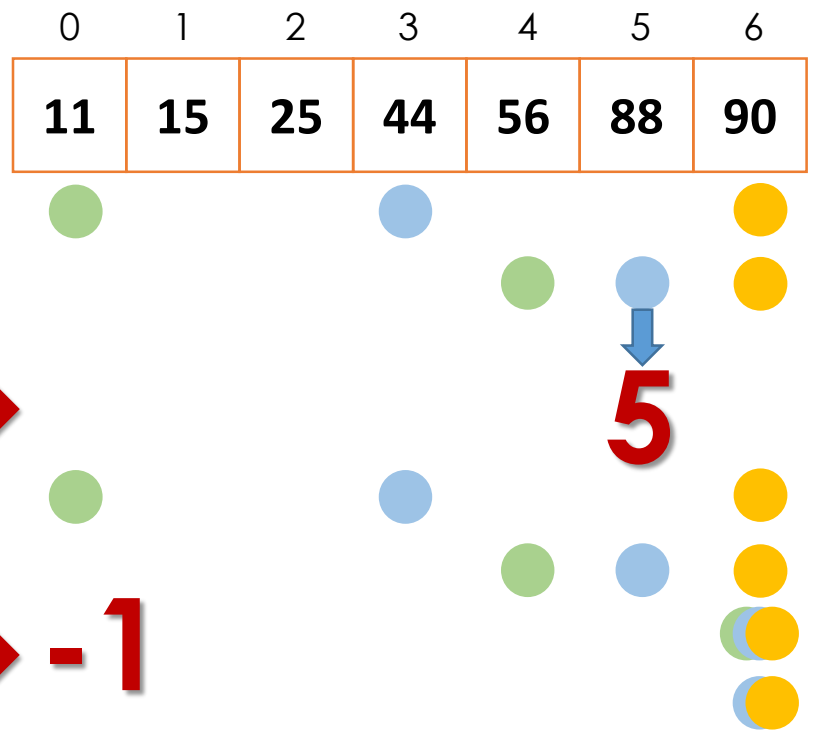
Η αναζήτηση τιμής σε ταξινόμητο πίνακα είναι αρκετά απλή, όμως δεν είναι αποδοτική. Ας δούμε τον κώδικα:

```
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? ➔

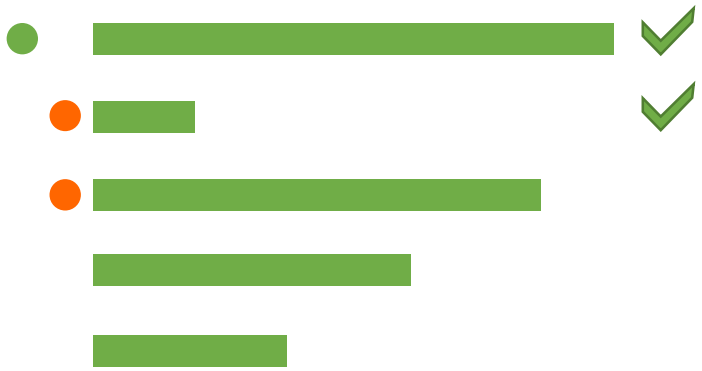
95? ➔ -1

# Ταξινόμηση μονοδιάστατου πίνακα

Υπάρχουν αρκετοί αλγόριθμοι ταξινόμησης και μπορείτε να δείτε μερικούς εν δράσει στη σελίδα αυτή.

Εδώ θα δούμε τον απλούστερο, που ονομάζεται selection sort.

```
function selectionSort(a, N) {
    for (let i = 0; i < N; ++i) {
        let indexOfMin = i;
        let minValue = a[i];
        for (let j = i+1; j < N; ++j) {
            if (a[j] < minValue) {
                indexOfMin = j;
                minValue = a[j];
            }
        }
        if (indexOfMin > i) {
            let Ai = a[i];
            a[i] = a[indexOfMin];
            a[indexOfMin] = Ai;
        }
    }
}
```



# Εφαρμογές

Ας εφαρμόσουμε επιτέλους τη θεωρία

# Πρακτική 1 – Πρώτοι αριθμοί

Υπολογισμός πρώτων αριθμών με το *Κόσκινο του Ερατοσθένη*

| | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

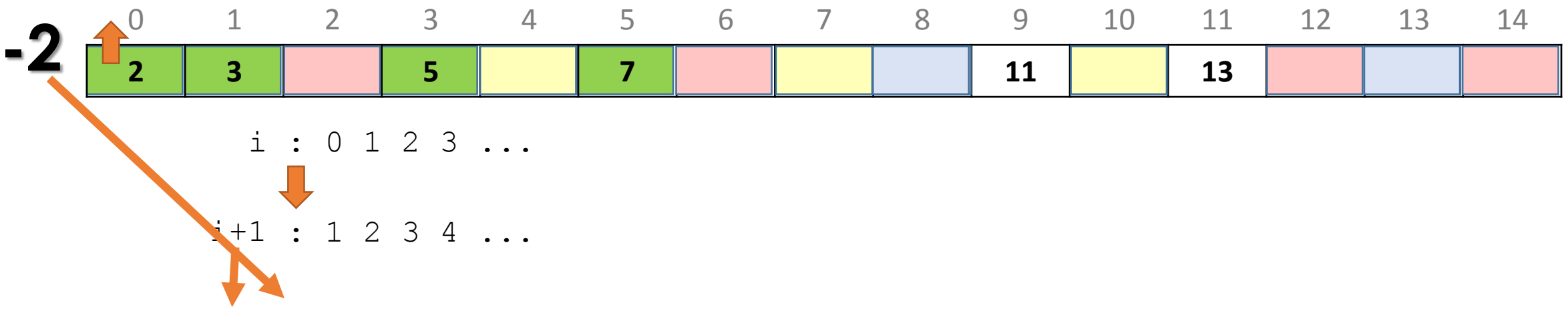
Η μεθοδολογία αυτή θα δούμε εδώ πως εφαρμόζεται για τον υπολογισμό των πρώτων αριθμών, όμως έχει αρκετά ευρείες εφαρμογές. Η λογική είναι από το σύνολο των πιθανών τιμών, με απλό τρόπο να απορρίψουμε όλες αυτές που δεν αποτελούν λύσεις (εδώ όσους δεν είναι πρώτοι). Έτσι όσες τιμές περισσέψουν είναι οι ζητούμενες λύσεις.

Αυτό στους πρώτους αριθμούς από το 1 μέχρι το 100 μεταφράζεται σε απόρριψη όλων των πολλαπλασίων του 2, του 3 και γενικά του επόμενου μη διαγραμμένου κοκ μέχρι εκεί που έχει νόημα ($=\sqrt{100}$).

**Προσοχή!** Δεν απορρίπτουμε τα 1x2 ή 1x3 αλλά μόνο τα πολλαπλάσιά τους.

Ας γράψουμε τη συνάρτηση `function printPrimesTo(N)` εκτυπώνει τους πρώτους αριθμούς ως το N

# Πρώτοι αριθμοί - παρατηρήσεις




2 : 2*(i+1) - 2 ➔ (2 4 6 8 10 12 ...) - 2 ➔ 0 2 4 6 8 10 ...

3 : 3*(i+1) - 2 ➔ (3 6 9 12 15 ...) - 2 ➔ 1 4 7 10 13 ...

5 : 5*(i+1) - 2 ➔ (5 10 15 ...) - 2 ➔ 3 8 13 ...

# Πρακτική 2 – Τρίλιζα

Εάν σε έναν πίνακα 3x3 θέσουμε τα στοιχεία του
ως κενά ή να περιέχουν τα σύμβολα X ή O τότε μπορούμε να παραστήσουμε μία τρίλιζα!

Στην τρίλιζα ξεκινά πάντα να παίζει πρώτος ο X (το θεωρούμε δεδομένο). Ας δούμε μερικές συναρτήσεις που έχουν ενδιαφέρον:

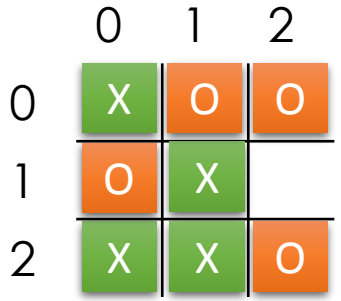
1. `function playersTurn(board)`
   Θα επιστρέφει τον παίκτη που έχει σειρά να παίξει ή -1 ως ένδειξη λάθους.

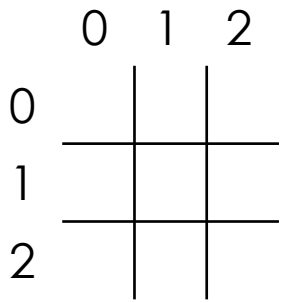
2. `function tripletExists(board, player)`
   Θα επιστρέφει αληθές εάν ο παίκτης player έχει κάνει τρίλιζα.

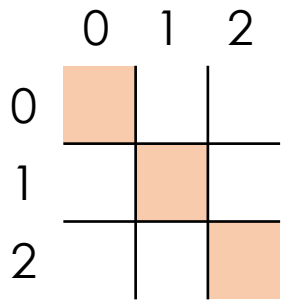
3. `function canPlay(board, row, col, player)`
   Θα επιστρέφει αληθές αν ο παίκτης player επιτρέπεται να παίξει στη θέση row,col

4. `function playAt(board, row, col)`
   παίζει τον παίκτη που είναι η σειρά του, στη θέση row,col , αλλιώς επιστρέφει false

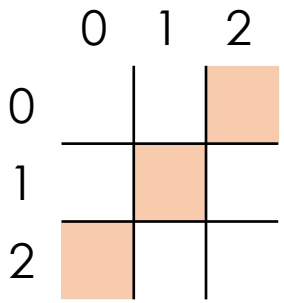
# Τρίλιζα παρατηρήσεις



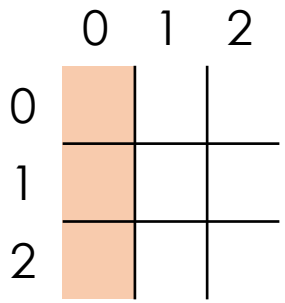



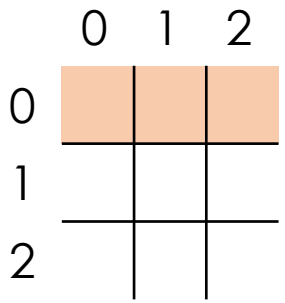

x 0 1 2
y 0 1 2




x 0 1 2
y 2 1 0




x 0 0 0
y 0 1 2




x 0 1 2
y 0 0 0