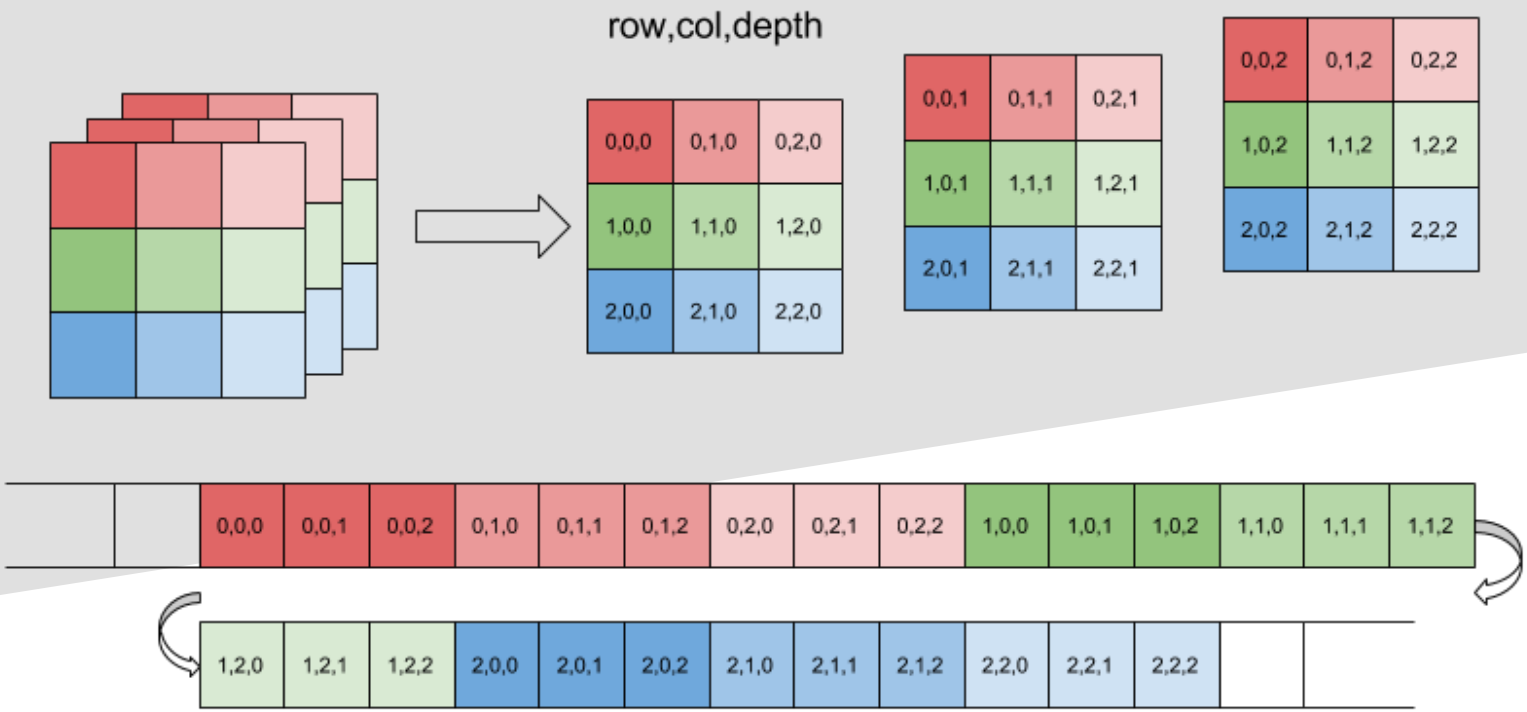
Tema 05. Arrays



Índice

1.- Arrays unidimensionales (vectores).

Son **estructuras de datos** para agrupar elementos de un mismo tipo dentro de un programa.

Ejemplo:

Si queremos almacenar las notas de un examen de los 30 alumnos de la clase, ¿ 30 variables ? ¿ nota1, nota2, nota3, ... nota30 ?

NO!

Crearemos un array o vector de notas donde se almacenan de forma organizada las 30 notas

índices	0	1	2	3	4	…	30
arrayNotas	6	4	8	9	6	…	3

1.- Arrays unidimensionales (vectores).

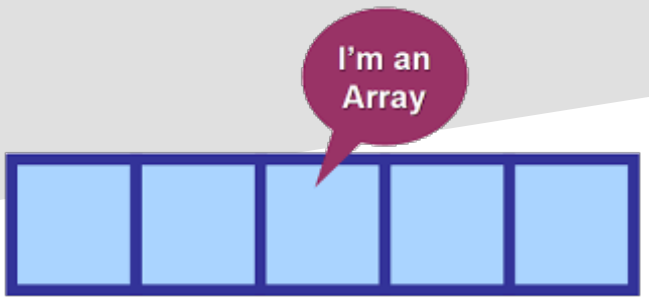
Un array (vector) es una colección ordenada de elementos del mismo tipo, donde cada elemento está asociado a un índice o posición que ocupa.

Los elementos de un array se almacenan en posiciones contiguas de memoria.

Un vector queda determinado por:

- Tipo de sus elementos.

- Número de elementos.



Para utilizar un array o vector hay que realizar tres operaciones:
declarar una referencia al vector, crear el vector, e inicializarlo.

1.- Arrays unidimensionales (vectores).

DECLARACIÓN

Como otras variables, antes de poder utilizar un vector, primero se debe declarar.

Se pueden declarar de formas distintas:

```
Tipo nombre[];
```

```
Tipo[] nombre;
```

Donde `Tipo`, indica el tipo de los elementos del vector, que pueden ser de cualquier tipo (`int, double, String, un objeto…`); y `nombre` es un identificador que nombra al array o vector.

De esta forma, `nombre` es una referencia a un vector, pero **el vector todavía no está creado.**

1.- Arrays unidimensionales (vectores).

DECLARACIÓN (EJEMPLOS)

```
Double[] vectorNotas;       //Vector de elementos tipo double
Cuenta[] vectorCuentas;     //Vector de objetos Cuenta
```

O también se podrían declarar así:

```
Double vectorNotas[];
Cuenta vectorCuentas[];
```

1.- Arrays unidimensionales (vectores).

CREACIÓN

Tras declarar el vector, el siguiente paso es crearlo o construirlo. Hay que indicar el número de elementos del vector para que se pueda reservar la cantidad de memoria para contener todos sus elementos.

La sintaxis es la siguiente:

```
nombre = new Tipo[numeroElementos];
```

Donde `nombre` es el nombre del vector (**previamente declarado**); `Tipo`, es el tipo de los elementos del vector; y `numeroElementos` especifica el número de elementos que tendrá.

1.- Arrays unidimensionales (vectores).

CREACIÓN (EJEMPLOS)

Crear un vector identificado por notas, con 30 elementos de tipo double:

```
double[] vectorNotas;
vectorNotas = new double[30];
```

Crear un vector de 10 objetos Cuenta:

```
Cuenta vectorCuenta[];
vectorCuenta = new Cuenta[10];
```

IMPORTANTE: cuando se crea un vector de objetos se ha hecho el `new` del vector, no de los objetos. Se reserva memoria para poder almacenar 10 objetos de tipo `Cuenta` pero todos están a `null`.

1.- Arrays unidimensionales (vectores).

CREACIÓN (EJEMPLOS)

Es muy común declarar y crear el vector en una sola línea de la siguiente forma:

```
Tipo[] nombre = new Tipo[tamano];

Tipo nombre[] = new Tipo [tamano];
```

Ejemplos:

```
double[] vectorNotas = new double[30];
Cuenta vectorCuentas[] = new Cuenta[10];
```

1.- Arrays unidimensionales (vectores).

INICIALIZACIÓN

Si no se especifica ningún valor, los elementos de un vector se inicializan automáticamente a unos **valores predeterminados** (variables numéricas a 0, objetos a `null` y booleanos a `false`).

Es posible inicializarlo con los valores que se deseen a la vez que se declaran. En este caso no hace falta hacer el `new` y el tamaño del vector se entiende por el número de elementos.

Ejemplo:

```
double[] vectorNotas = { 5.5, 6, 8, 9.1, 10, 7.2};
// Tamaño vector, 6
String[] diasLaborables = {"Lunes", "Martes", "Miercoles", "Jueves", "Viernes"};
//Tamaño vector 5
```

1.- Arrays unidimensionales (vectores).

ACCESO

Para acceder al valor de un elemento de un vector, se utiliza el nombre del vector seguido de un subíndice entre corchetes [x].

Los índices van **desde la posición 0** hasta el tamaño del vector-1.

Para el ejemplo de las notas:

índices	0	1	2	3	4	5
vectorNotas	5.5	6	8	9.1	10	7.2

Para acceder a la nota del alumno 4 → `vectorNotas[3];`

1.- Arrays unidimensionales (vectores).

ACCESO

Los elementos de un vector no son más que variables, por tanto, se pueden utilizar exactamente igual que cualquier otra variable y operar con ellos.

Por ejemplo, en las operaciones que se muestran a continuación intervienen elementos de un vector:

```
double[] vectorNotas = new double[30];
vectorNotas[5] = 9;
vectorNotas[8] = vectorNotas[2] + 5.5;
```

1.- Arrays unidimensionales (vectores).

ACCESO (Excepciones)

Si se intenta acceder a un elemento del vector, con un subíndice fuera de rango, Java lanzará una excepción del tipo:

`ArrayIndexOutOfBoundsException`

¿ Cómo podemos asegurarnos de no exceder el final de un vector ?

Verificando la longitud del mismo, mediante el atributo **`length`**

Ejemplo:

```
int numeroDeElementos = vector.length;
//Devuelve el número de elementos del vector
```

1.- Arrays unidimensionales (vectores).

RECORRIDO

Siempre que se necesite realizar un tratamiento sobre todos los elementos de un vector, habrá que realizar un recorrido de éste.

El programador debe preocuparse de que el índice no se pase de los límites del vector.

Ejemplo:

```
for (int i = 0; i < vector.length; i++) {
    //Realizar el tratamiento que se desee con el elemento vector[i]
}
```

1.- Arrays unidimensionales (vectores).

BÚSQUEDA

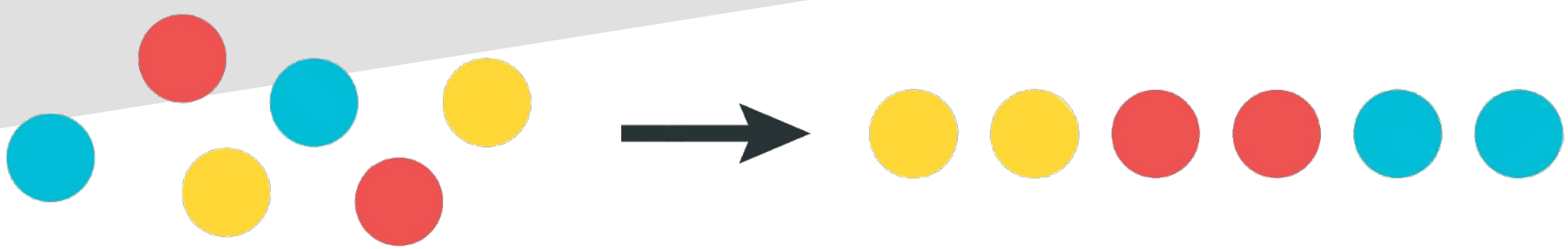
```java
/**
 * Método para buscar en elemento en un vector
 * @param vector
 * @param elementoBuscado
 * @return posición en la que se encuentra el elemento dentro del vector
 *          -1 si no encuentra el elemento en el vector.
 */
public static int buscarElementoEnVector(double[] vector, double elementoBuscado) {
    int posicion = -1;
    int i = 0;
    while (i < vector.length && posicion == -1) {
        if (vector[i] == elementoBuscado)
            posicion = i;
        else
            i++;
    }
    return posicion;
}
```

1.- Arrays unidimensionales (vectores).

ORDENACIÓN

La ordenación de un vector es un tema que se ha estudiado mucho. Hay muchos algoritmos, entre ellos el algoritmo de selección, la burbuja, y el algoritmo del quicksort que es el que implementa Java en su método `Arrays.sort`

`Arrays.sort` ordena por defecto de forma ascendente. Para ordenar con otro criterio también existe la forma que veremos cuando estudiemos las colecciones.



1.- Arrays unidimensionales (vectores).

ORDENACIÓN (tipos básicos y String)

```
int[] vectorEnteros = { 5, 22, 30, 5, 6, 8 };
String[] vectorNombres = { "LUCIA", "ANA MARIA", "LAURA", "ROSA", "JUAN" };

System.out.println("Antes: ");
System.out.println(Arrays.toString(vectorEnteros));
System.out.println(Arrays.toString(vectorNombres));

Arrays.sort(vectorEnteros);
Arrays.sort(vectorNombres);

System.out.println("Después: ");
System.out.println(Arrays.toString(vectorEnteros));
System.out.println(Arrays.toString(vectorNombres));
```

```
Antes:
[5, 22, 30, 5, 6, 8]
[LUCIA, ANA MARIA, LAURA, ROSA, JUAN]
Después:
[5, 5, 6, 8, 22, 30]
[ANA MARIA, JUAN, LAURA, LUCIA, ROSA]
```

1.- Arrays unidimensionales (vectores).

ORDENACIÓN (objetos)

Si necesitamos ordenar un array de objetos (por ejemplo `Cuenta`), la clase debe implementar la interfaz `Comparable`.

Esto significa que debe tener implementado el método `compareTo` que describe el criterio de ordenación (en nuestro ejemplo, cuando una cuenta es menor o mayor que otra).

```
public class Cuenta implements Comparable<Cuenta> {
    @Override
    public int compareTo(Pruebas.Cuenta o) {
        // TODO Auto-generated method stub
        return 0;
    }
}
```

1.- Arrays unidimensionales (vectores).

ORDENACIÓN (método `compareTo`)

El método `compareTo` debe tener el prototipo:

```
@Override
public int compareTo(Objeto otro) {
    // TODO Auto-generated method stub
    return 0;
}
```

Funciona de la siguiente forma:

- Devuelve 0 si el objeto `this` es igual al objeto `otro`.
- Devuelve un valor menor que 0 si el objeto `this` es menor que el objeto `otro`.
- Devuelve un valor mayor que 0 si el objeto `this` es mayor que el objeto `otro`.

1.- Arrays unidimensionales (vectores).

ORDENACIÓN (método compareTo) EJEMPLO

```
public int compareTo(Cuenta otra) {
    int resultado;
    if (this.getSaldo() == otra.getSaldo())
        resultado = 0;
    else if (this.getSaldo() < otra.getSaldo())
        resultado = 1;
    else
        resultado = -1;
    return resultado;
}
```

NOTA: suponiendo que dos cuenta son iguales si sus saldos son iguales.

1.- Arrays unidimensionales (vectores).

Java maneja los vectores como si fueran objetos, por lo tanto existen una serie de métodos heredados de la clase `Object` que están en el paquete `java.lang`. Entre ellos cabe destacar:

EQUALS

Devuelve `true` si los dos arrays comparados son el mismo, es decir **compara las referencias**. Se puede sobreescribir para que compare objetos según nuestras propias reglas.

¿Se puede reutilizar compareTo ?

CLONE

Devuelve un objeto nuevo (array) con los mismos datos que el array del que se clona.

¿Se clonan los objetos también?

1.- Arrays unidimensionales (vectores).

EQUALS Y CLONE (EJEMPLOS)

```
double[] notas1 = { 5.5, 6, 5, 8, 3 };
double[] notas2 = { 5.5, 6, 5, 8, 3 };
double[] notas1Clonado;
double[] notas3;
notas1Clonado = notas1.clone();
notas3 = notas1;
if (notas1.equals(notas2))
    System.out.println("Equals entre notas1 y notas2 dice IGUALES");
else
    System.out.println("Equals entre notas1 y notas2 dice DISTINTOS");
if (notas1.equals(notas1Clonado))
    System.out.println("Equals entre notas1 y notas1Clonado dice IGUALES");
else
    System.out.println("Equals entre notas1 y notas1Clonado dice DISTINTOS");
if (notas1.equals(notas3))
    System.out.println("Equals entre notas1 y notas3 dice IGUALES");
else
    System.out.println("Equals entre notas1 y notas3 dice DISTINTOS");
```

1.- Arrays unidimensionales (vectores).

USO COMO PARÁMETRO

Un vector puede pasarse como parámetro a un método. El paso siempre es por referencia, es decir, **cualquier cambio que se haga en un método sobre los elementos de un vector se conservará** al abandonar dicho método.

Al pasar un vector como parámetro solo se debe indicar el nombre del vector. En el prototipo del método si se indica que es un vector con los corchetes [].

1.- Arrays unidimensionales (vectores).

USO COMO PARÁMETRO (EJEMPLO)

```
public static void main(String[] args) {
    double[] vectorNotas = new double[TOTAL_ALUMNOS];
    insertarNotasEnVector(vectorNotas);
    mostrarVectorNotas(vectorNotas);
}

private static void insertarNotasEnVector(double[] vectorNotas) {
    for (int i = 0; i < vectorNotas.length; i++) {
        System.out.println("Introduce la nota del alumno " + (i + 1));
        vectorNotas[i] = Double.parseDouble(teclado.nextLine());
    }
}

private static void mostrarVectorNotas(double[] vectorNotas) {
    for (int i = 0; i < vectorNotas.length; i++) {
        System.out.println("Nota del alumno " + (i + 1) + ":" + vectorNotas[i]);
    }
}
```

2.- Arrays bidimensionales (matrices).

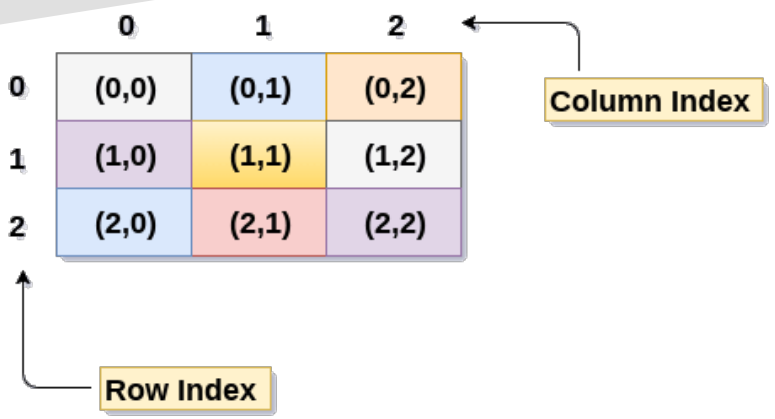
INTRODUCCIÓN

Una matriz es un array de dos dimensiones, es decir, un conjunto de elementos del mismo tipo que se distribuyen en forma de filas y columnas.

Las matrices suelen usarse para relacionar 2 magnitudes.

Ejemplo: Las notas de los 30 alumnos de la clase en los tres trimestres.

La representación de una matriz es:



2.- Arrays bidimensionales (matrices).

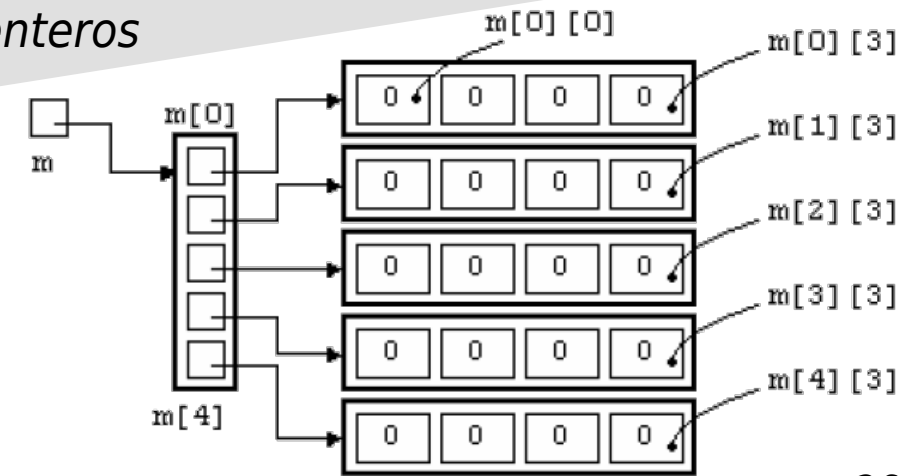
DECLARACIÓN Y CREACIÓN

```
Tipo nombre[][] = new Tipo[numFilas][numCol];
```

dónde “numFilas” es el número de filas y “numCol” el número de columnas.

Ejemplo: una matriz de tamaño 5 x 4 de enteros

```
int[][] m = new int[5][4];
```



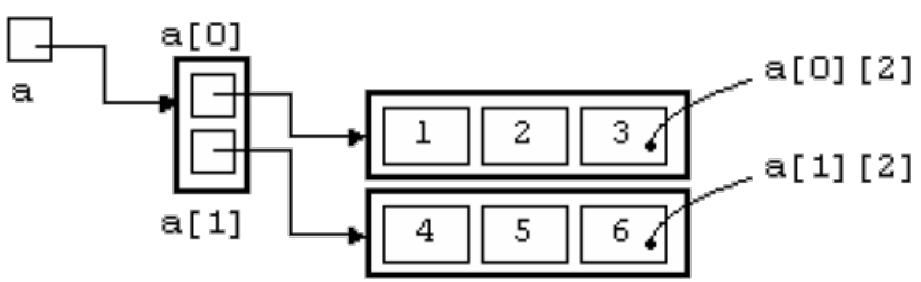
2.- Arrays bidimensionales (matrices).

INICIALIZACIÓN

Al igual que ocurría con los vectores, también podemos llevar a cabo la declaración de la referencia, creación de la matriz e inicialización de sus elementos en una sola línea.

Ejemplo: declaración, creación e inicialización de una matriz 2 x 3 de números enteros.

```
int a[][] = {{1,2,3}, {4,5,6}};
```



2.- Arrays bidimensionales (matrices).

ACCESO

```
// Crea una matriz para almacenar las notas de 5 alumnos
// agrupadas por trimestres (3)
double[][] notas = new double[3][5];

// Aumenta en 1 la nota del alumno 4 en el segundo trimestre:
notas[1][3] ++;
// ES UNA REFERENCIA NO HACE FALTA notas[1][3] = notas[1][3] ++;
```

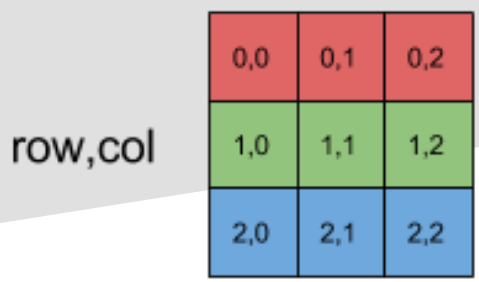
	0	1	2	3	4
0	6.5	6	3	2	5
1	6	8	4.5	**3**	4.4
2	9.2	9	2	1	6.3

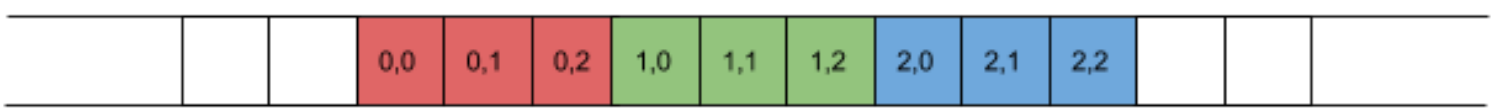
2.- Arrays bidimensionales (matrices).

OPERACIONES CON MATRICES

Recorrer una matriz es realizar un tratamiento a cada uno de sus elementos.

Una matriz puede recorrerse por filas o por columnas, según nos interese.





2.- Arrays bidimensionales (matrices).

RECORRIDO (Por filas)

```java
double[][] notas = new double[3][5];
// Ejemplo para una matriz de 3 filas y 5 columnas
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 5; j++) {
        notas[i][j] = 0;
    }
}
// Para cualquier matriz
for (int i = 0; i < notas.length; i++) {
    for (int j = 0; j < notas[i].length; j++) {
        notas[i][j] = 0;
    }
}
```

2.- Arrays bidimensionales (matrices).

RECORRIDO (Por columnas)

```java
double[][] notas = new double[3][5];
// Ejemplo para una matriz de 3 filas y 5 columnas
for (int j = 0; j < 5; j++) {
    for (int i = 0; i < 3; i++) {
        notas[i][j] = 0;
    }
}
// Para cualquier matriz
for (int j = 0; j < notas[0].length; j++) {
    for (int i = 0; i < notas.length; i++) {
        notas[i][j] = 0;
    }
}
```

Tema 05. Arrays