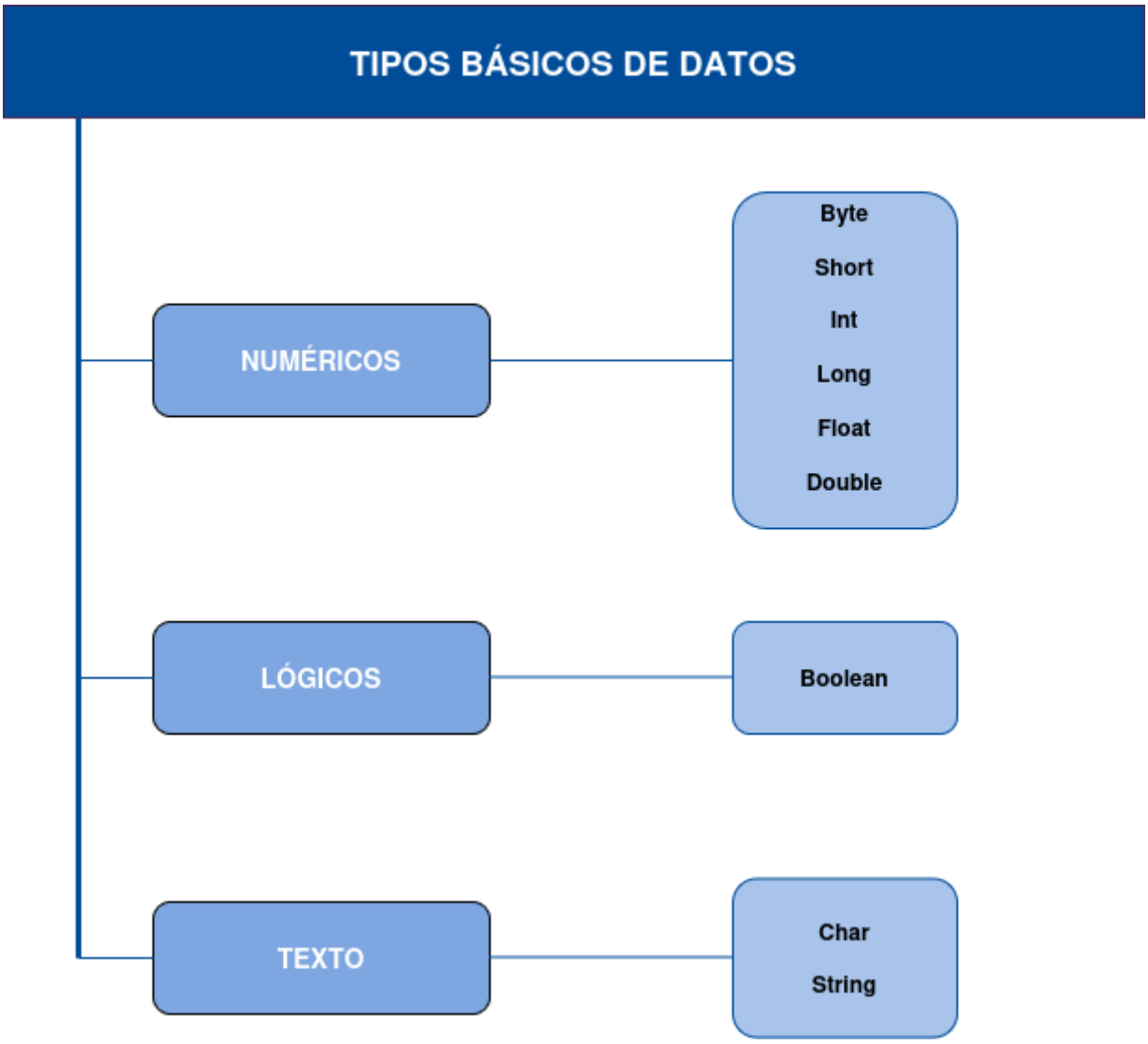
Sumario

1. Introducción a los tipos de datos y variables

En programación, un **tipo de datos** se refiere a una categoría o conjunto de valores que una variable o expresión puede tener. Los tipos de datos son fundamentales en cualquier lenguaje de programación, ya que permiten al programador especificar qué tipo de información se almacenará en una variable y cómo se manipulará.



Existen diversos tipos de datos comunes en la mayoría de los lenguajes de programación, como los siguientes:

1. **Números enteros (int)**: Representan números sin decimales. Pueden ser positivos o negativos.
2. **Números de punto flotante (float)**: Representan números con decimales. También pueden ser positivos o negativos.
3. **Caracteres (char)**: Representan un único carácter, como una letra o un símbolo.
4. **Cadenas de caracteres (Strings)**: Representan una secuencia de caracteres. Pueden contener letras, números, símbolos y espacios.
5. **Booleanos**: Representan un valor de verdad, que puede ser verdadero (true) o falso (false).
6. **Arreglos (arrays)**: Representan una colección ordenada de elementos del mismo tipo de datos.

Respecto a los arreglos o arrays, existen dos tipos:

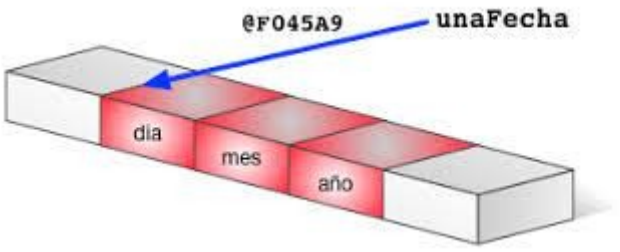
- **Arrays indexados**: En este tipo de array cada elemento allí contenido vendrá referenciado por un índice numérico que indicará exactamente la posición dentro del array en el que se encuentra.
- **Arrays asociativo**: En este caso el índice que señala la posición en la cual se encuentra el valor del array no tiene por que ser numérico pudiendo, por ejemplo, contener las siglas de lo que contiene: PR, BD, PW.

Array Indexado			
0	1	2	**POSICIÓN**
Programación	Base de Datos	Programación Web	**VALOR**

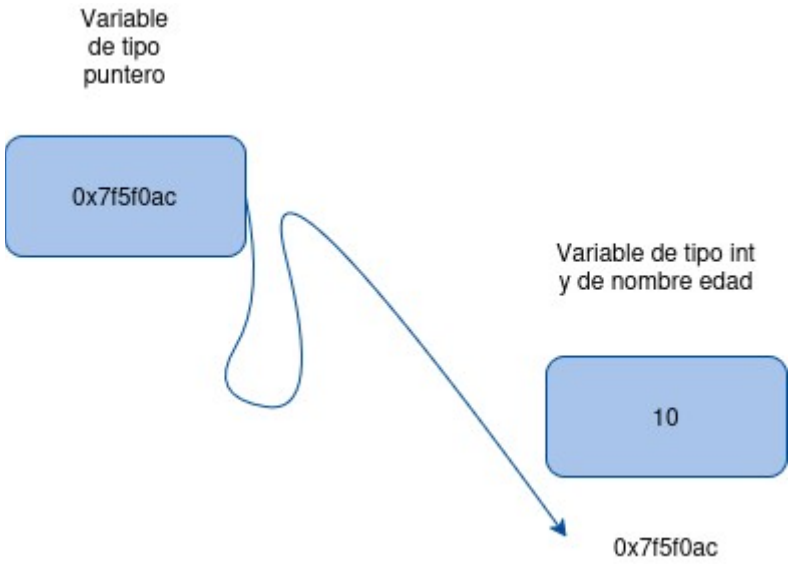
Array Asociativo			
PR	BD	PRW	**POSICIÓN**
Programación	Base de Datos	Programación Web	**VALOR**

7. Estructuras:

Permiten agrupar varios valores (del mismo o diferente tipo) relacionados en una sola entidad. En Java la declaración de estructuras se realiza mediante la creación de un objeto sin métodos instanciado a partir de una clase.



8. **Punteros (pointers):** Almacenan la dirección de memoria de otro objeto o variable.



Estos son solo algunos ejemplos de tipos de datos, y cada lenguaje de programación puede tener su propia variedad de tipos de datos incorporados.

1.1 ¿Qué es una variable?

Una **variable** es una dirección de memoria (RAM) dentro de la cual podrá ser guardado un valor de un determinado tipo de datos.

Para la creación de una nueva variable en un programa será necesario declarar dicha variable. Por ejemplo si el programa precisa una variable denominada *edad*, será necesario declararla así como el tipo de datos que va a contener. En java la declaración de la variable *edad* se realizaría de la forma siguiente:

int edad;

Variable edad		
Nombre	Dirección de memoria	Valor contenido en la variable
edad	0x715f0ac	

Otra posibilidad es declarar la variable edad y a la vez guardar en ella un determinado valor:

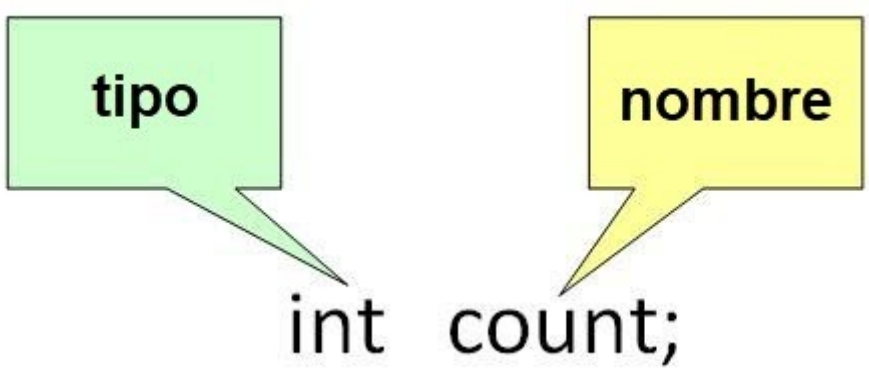
int edad=20;

Variable edad		
Nombre	Dirección de memoria	Valor contenido en la variable
edad	0x715f0ac	20

1.2 Declaración de una variable

Como se ha mencionado en el apartado anterior, para declarar una variable en Java, en primer lugar, en el lado izquierdo, se definirá su tipo de datos, y a continuación, a la derecha del tipo de datos se indica el nombre de la variable.

En al ilustración siguiente se define la variable de nombre *count* con el tipo de datos *int*.



Algunos ejemplos de declaración de variables en Java son los siguientes:

- Declaración de la variable denominada precio de tipo real (double):

```
double precio;
```

- Declaración e inicialización de la variable *precio*:

double precio=3,141516

- Declaración de la variable denominada *nombre* de tipo String (cadena de caracteres):

  String nombre;

- Declaración e inicialización de la variable *nombre*:

  String nombre=”Juan”;

- Declaración de la variable letra de tipo char:

  char letra;

- Declaración e inicialización de la variable *letra*:

  char letra=’a’;

- Declaración de la variable *casado* de tipo *boolean*:

boolean casado;

- Declaración e inicialización de la variable *casado*:

boolean casado=true;

- Declaración de la variable de tipo *float area*:

float area;

- Declaración e inicialización de la variable de tipo *float area*:

float area=34,67;

- Declaración e inicialización de la variable *edad* de tipo byte:

byte edad=4;

1.3 Convenciones de nomenclatura

En Java, se siguen algunas convenciones para la nomenclatura de variables. Aquí tienes algunas pautas comunes:

1. Los nombres de las variables deben comenzar con una letra minúscula. Si el nombre consta de varias palabras, se utiliza el estilo de camello (camelCase), en el cual la primera letra de cada palabra después de la primera se escribe en mayúscula, sin espacios ni guiones bajos. Ejemplo: `miVariable`, `nombreDeUsuario`.

2. No utilizar abreviaturas al denominar una variable.

3. No utilizar nombres letras para nombrar una variable a no ser que dichas variables sean utilizadas en un contador de un bucle: *int i; for(i=0;i<10;i++)*

4. Se recomienda nombrar las variables *booleanas* o lógicas (*true* o *false*) anteponiendo a su nombre el prefijo ***is***: *iscasado*, *iscorrecto*, etc.

5. Evita utilizar nombrEn Java, se siguen algunas convenciones para la nomenclatura de variables. Aquí tienes algunas pautas comunes:

6. Los nombres de las variables deben comenzar con una letra minúscula. Si el nombre consta de varias palabras, se utiliza el estilo de camello (camelCase), en el cual la

primera letra de cada palabra después de la primera se escribe en mayúscula, sin espacios ni guiones bajos. Ejemplo: `miVariable`, `nombreDeUsuario`.

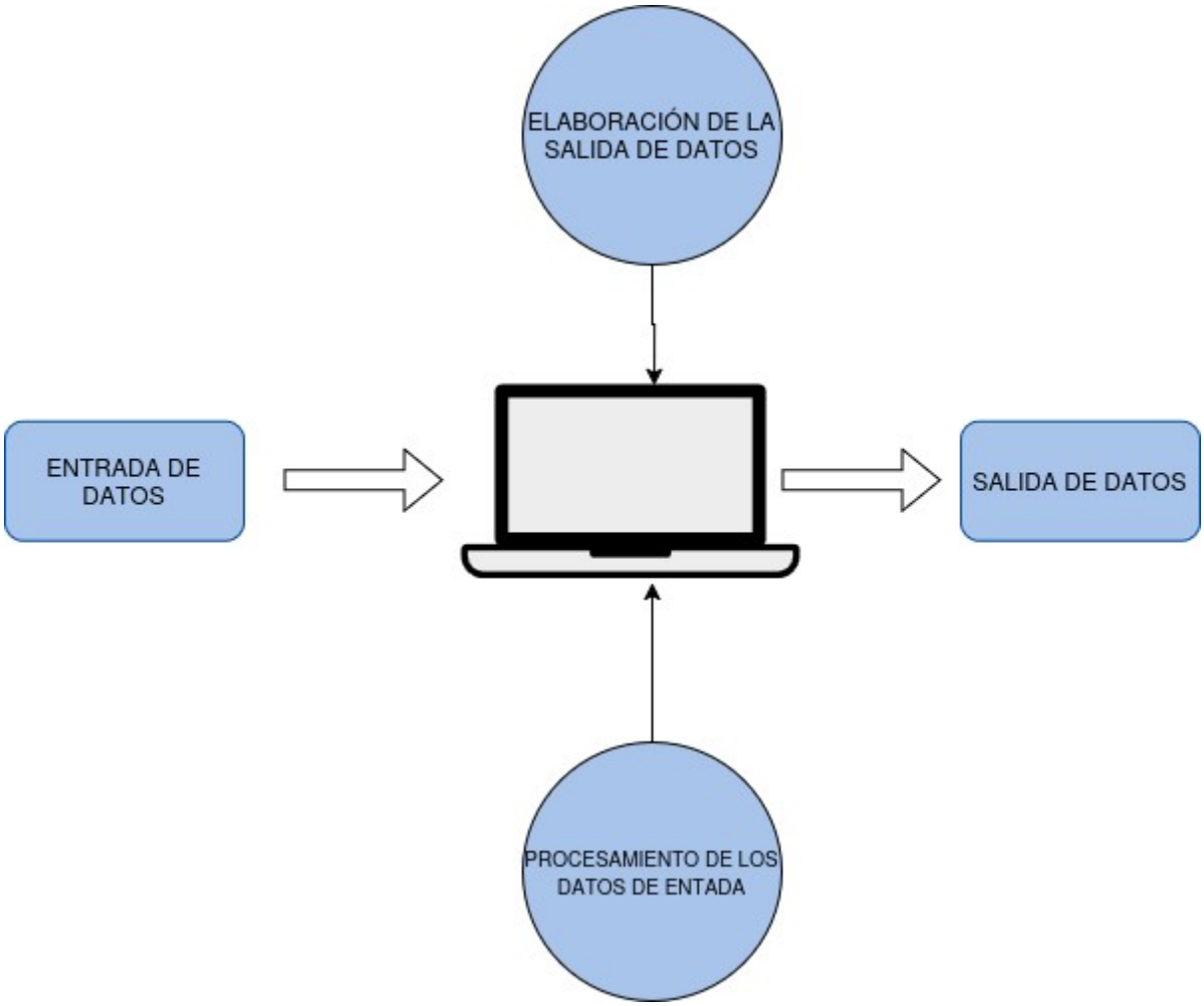
7. No utilizar abreviaturas al denominar una variable.
8. No utilizar nombres letras para nombrar una variable a no ser que dichas variables sean utilizadas en un contador de un bucle: *int i; for(i=0;i<10;i++)*
9. Se recomienda nombrar las variables *booleanas* o lógicas (*true* o *false*) anteponiendo a su nombre el prefijo ***is***: *iscasado*, *iscorrecto*, etc.
10. Evita utilizar nombres de una sola letra para variables, a menos que sea un caso comúnmente aceptado, como `i` para un índice de bucle.
11. Los nombres de las constantes se escriben en mayúsculas y se separan con guiones bajos. Ejemplo: `MI_CONSTANTE`, `VALOR_MAXIMO`.
12. Si una variable es miembro de una clase o una instancia, es común utilizar el prefijo `this` para distinguirlo. Ejemplo: `this.miVariable`, `this.nombreDeUsuario`.
13. Evita utilizar nombres genéricos como `objeto` o `dato`. Intenta darle un nombre más descriptivo que refleje el propósito o la función de la variable.
14. Si una variable es una constante y estática, es común utilizar el prefijo `final` para indicarlo. Ejemplo:

```
static final int MI_CONSTANTE = 10
```

Es importante tener en cuenta que estas son convenciones sugeridas y no reglas estrictas. La consistencia y la legibilidad son fundamentales, así que asegúrate de seguir un estilo de nomenclatura coherente en todo tu código.

2. Entrada y salida

En la mayoría de aplicaciones, se presenta la posibilidad al usuario de introducir datos al programa (entrada), el cual los procesará y empleará para realizar determinadas acciones que finalmente aportarán unos resultados los cuales serán mostrados (salida) al usuario de la aplicación.



La entrada de datos a un programa, se puede obtener de diferentes formas:

- A partir de los valores insertados desde la consola por parte del usuario del programa.
- A partir de los valores insertados desde un formulario por parte del usuario del programa.
- Desde un fichero.
- Desde una tabla de una base de datos.
- A partir de los datos obtenidos por un sensor (sensor de movimiento, infrarojos, ultrasonidos, humedad, etc.).

2.1 Entrada y salida en consola

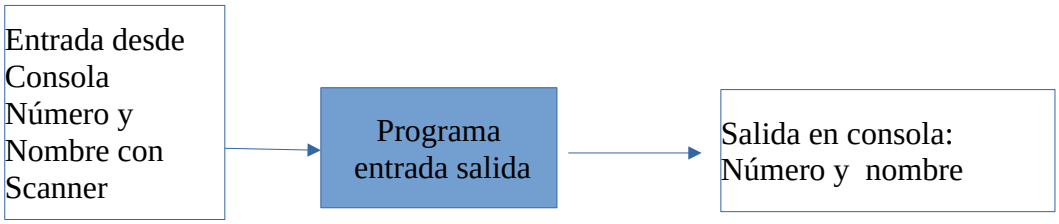
La salida en consola se realiza mediante la ejecución de la orden: system.out.println. Veamos el ejemplo de programa que vuestra en la salida de la consola los mensajes: “Hola Buenos días” y “Adios”:

```
public class Saludos {

    public static void main(String[] args) {
        System.out.println("Hola Buenos días");
        System.out.println("Adios");
    }
}
```

Lanzar el programa

Si lo que se necesita es guardar en una variable un valor introducido por el usuario conviene hacer uso del Scanner. Veamos el código de programa siguiente:

El programa siguiente realiza la entrada de datos solicitando un número entero y su nombre al usuario (mediante el empleo de la clase *Scanner*), finalmente el programa mostrará ambos datos al usuario (salida).




```
import java.util.Scanner;

public class EntradaSalidaConsola {

   public static void main(String[] args) {
      // Crear una instancia de Scanner para leer la entrada

    int numero;

  String nombre;
      Scanner sc = new Scanner(System.in);


      // Leer un entero
      System.out.print("Ingresa un número entero: ");
      numero = sc.nextInt();
 // Limpia buffer de teclado
```

```
        sc.nextLine();
        // Leer una cadena de caracteres
        System.out.print("Ingresa tu nombre: ");
        nombre = sc.nextLine();
        // Imprimir los valores leídos
        System.out.println("Número ingresado: " + numero);
        System.out.println("Nombre ingresado: " + nombre);
        // Cerrar el Scanner
        sc.close();
    }
}
```

Lanzar el programa

2.1.1 Entrada de datos en una variable de tipo int

El programa siguiente solicita un código de un producto al usuario y lo guarda en una variable de tipo int denominada *codigo*. Finalmente se mostrará dicho código en la consola.

```
import java.util.Scanner;

public class LeerCodigoProducto {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

         int codigo;

        // Solicitar al usuario que ingrese el código del producto

        System.out.print("Introduce el código del producto: ");

        codigo = sc.nextInt();  // Guardar el código en la variable 'codigo'



        // Mostrar el código ingresado

        System.out.println("El código del producto es: " + codigo);

    }

}
```

Lanzar el programa

2.1.2 Entrada de datos en una variable de tipo float

El programa siguiente debe de solicitar al usuario la altura en metros de una persona. Finalmente el programa visualizará dicha altura en metros por la consola, también deberá visualizar la misma medida en centímetros

```
import java.util.Scanner;
public class AlturaEnMetrosYCentimetros {
    public static void main(String[] args) {
        float alturaMetros;
        float alturaCentimetros;
        Scanner sc = new Scanner(System.in);
       // Solicitar al usuario que ingrese la altura en metros
       System.out.print("Introduce la altura en metros: ");
       alturaMetros = sc.nextFloat(); // Leer la altura en metros como un float
       // Convertir la altura a centímetros
       alturaCentimetros = alturaMetros * 100;
       // Mostrar la altura en metros y en centímetros
       System.out.println("La altura en metros es: " + alturaMetros + " m");
          System.out.println("La altura en centímetros es: " + alturaCentimetros + " cm");
    }
}
```

Lanzar el programa

2.1.3 Entrada de datos en una variable de tipo double

El siguiente programa es idéntico al anterior con la única diferencia que el tipo de datos a utilizar en lugar de ser float será *double*.

```
import java.util.Scanner;

public class AlturaEnMetrosYCentimetros {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

         double alturaCentimetros;

         double alturaMetros;

        // Solicitar al usuario que ingrese la altura en metros

        System.out.print("Introduce la altura en metros: ");

        alturaMetros = sc.nextDouble(); // Leer la altura en metros

        // Convertir la altura a centímetros

        alturaCentimetros = alturaMetros * 100;

       // Mostrar la altura en metros y en centímetros

        System.out.println("La altura en metros es: " + alturaMetros + " m");

        System.out.println("La altura en centímetros es: " + alturaCentimetros + " cm");

    }

}
```

Lanzar el programa

2.1.4 next(): Entrada de datos en una variable de tipo String

Mediante el método *next()* se guardará en una variable de tipo *String* una sola palabra introducida por el usuario. Si el usuario escribiera más de una palabra, una frase, únicamente se guardaría en la variable de tipo *String* la primera palabra.

```
import java.util.Scanner;

public class SolicitarNombre {

    public static void main(String[] args) {

        String nombre;

        Scanner sc = new Scanner(System.in);

        // Solicitar al usuario que ingrese su nombre

        System.out.print("Introduce tu nombre: ");

        nombre = sc.next(); // Leer el nombre utilizando next()

                // Mostrar el nombre ingresado

        System.out.println("Tu nombre es: " + nombre);

    }

}
```

Lanzar el programa

2.1.5 nextLine(): Entrada de datos en una variable de tipo String

A diferencia del método next(), con el método nextLine() se guardará en una variable de tipo String toda una frase (más de una palabra).

```
import java.util.Scanner;

public class SolicitarNombreYApellidos {

    public static void main(String[] args) {

        String nombreYApellidos

        Scanner sc = new Scanner(System.in);

        // Solicitar al usuario que ingrese su nombre y apellidos

        System.out.print("Introduce tu nombre y apellidos: ");

        nombreYApellidos = sc.nextLine(); // Leer la entrada completa

        // Mostrar el nombre y apellidos ingresados

        System.out.println("Tu nombre y apellidos son: " +
nombreYApellidos);

    }

}
```

Lanzar el programa

Nota:

Puede darse el caso que cuando se solicite al usuario la introducción de un texto, el programa no se espere a que el usuario pueda escribir lo que desee.

Para solucionar este problema se recomienda escribir antes la siguiente linea de código: ***sc.nextLine();***

```
import java.util.Scanner;

public class SolicitarNombreYApellidos {

    public static void main(String[] args) {

        String nombreYApellidos;

        Scanner sc = new Scanner(System.in);

        //Limpiar buffer del teclado

        sc.nextLine();

        // Solicitar al usuario que ingrese su nombre y apellidos

        System.out.print("Introduce tu nombre y apellidos: ");

        nombreYApellidos = sc.nextLine(); // Leer la entrada completa

        // Mostrar el nombre y apellidos ingresados

      System.out.println("Tu nombre y apellidos son: " + nombreYApellidos);

    }

}
```

Lanzar el programa

2.1.6 Entrada de datos en una variable de tipo char

Para leer en una variable de tipo char un carácter introducido por el usuario, se escribirá el comando siguiente:

```
categoria = sc.next().charAt(0);
```

El programa siguiente solicita al usuario el nombre, el precio y la categoría de un producto para visualizarlo todo en consola.

```
import java.util.Scanner;
public class Producto {
    public static void main(String[] args) {
        char categoria;
        String nombre;
        float precio;
      Scanner sc = new Scanner(System.in);
      // Solicitar el nombre del producto
      System.out.print("Introduce el nombre del producto: ");
      nombre = sc.nextLine(); // Leer el nombre del producto
      // Solicitar el precio del producto
      System.out.print("Introduce el precio del producto: ");
      precio = sc.nextFloat(); // Leer el precio del producto
      // Solicitar la categoría del producto
      System.out.print("Introduce la categoría del producto: ");
      categoria = sc.next().charAt(0); // Leer la categoría del producto (solo el primer carácter)
      // Mostrar los detalles del producto
      System.out.println("Nombre del producto: " + nombre + ", Precio del producto: " + precio + ", Categoría del producto: " + categoria);
    }
}
```

Lanzar el programa

2.1.7 Entrada de datos en una variable de tipo byte

El programa siguiente solicitará al usuario la introducción de una edad, la cual se guardará en una variable de tipo byte. Finalmente el programa mostrará el valor de dicha edad incrementado en una unidad junto con el mensaje: “¡Feliz cumpleaños!”

```
import java.util.Scanner;

public class FelizCumpleanyos {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // Solicitar la edad al usuario

        System.out.print("Introduce tu edad: ");

        byte edad = sc.nextByte(); // Leer la edad como byte

        // Incrementar la edad en una unidad

        edad++;

        // Mostrar el mensaje con la edad incrementada

        System.out.println("¡Feliz cumpleaños! Ahora tienes " + edad + "
años.");

    }

}
```

Lanzar el programa

2.1.8 Entrada de datos en una variable de tipo long

Para leer en una variable de tipo long los datos introducidos por el usuario se utiliza el método *nextLong()*;

El programa siguiente debe de solicitar al usuario el número de kilómetros existentes desde la tierra a una galaxia, también debe de solicitar al usuario el nombre de esa galaxia. Finalmente el programa debe de mostrar en consola el mensaje:

"La distancia en años a la galaxia 'la que sea' es de 'tanto'"

```
import java.util.Scanner;

public class DistanciaAGalaxia {

    public static void main(String[] args) {

        long kilometrosPorAnioLuz;

        long kilometros;

        double aniosLuz;

        Scanner sc = new Scanner(System.in);

        // Solicitar al usuario el nombre de la galaxia

        System.out.print("Introduce el nombre de la galaxia: ");

        String nombreGalaxia = sc.nextLine(); // Guardar el nombre de la galaxia

        // Solicitar al usuario el número de kilómetros hasta la galaxia

        System.out.print("Introduce el número de kilómetros hasta la galaxia: ");

        kilometros = sc.nextLong(); // Guardar el número de kilómetros en una variable de tipo long

        // Definir la cantidad de kilómetros en un año luz (1 año luz = 9.461e12 km)

        kilometrosPorAnioLuz = 9461000000000L;

         // Calcular el número de años luz

        aniosLuz = (double) kilometros / kilometrosPorAnioLuz;

         // Mostrar el resultado

        System.out.println("La distancia en años luz a la galaxia '" + nombreGalaxia + "' es de " + aniosLuz + " años luz.");    }}
```

Lanzar el programa

2.2 Entrada y salida con ventanas emergentes

Si en lugar de hacer uso de la consola de nuestro IDE se prefiere realizar la entrada de datos o la salida mediante ventanas puedes utilizar la clase **JOptionPane** de la biblioteca Swing. A continuación, te mostraré un ejemplo:

```
import javax.swing.JOptionPane;

public class EntradaSalidaVentanasEmergentes {
    public static void main(String[] args) {
        // Entrada de texto
        String nombre = JOptionPane.showInputDialog(null, "Ingrese su nombre:");

        // Mostrar el nombre ingresado
        JOptionPane.showMessageDialog(null, "Hola, " + nombre + "!");
    }
}
```

En este ejemplo, la ventana emergente **showInputDialog** se utiliza para solicitar al usuario que ingrese su nombre. El valor ingresado se guarda en la variable **nombre**. Luego, se muestra un mensaje emergente **showMessageDialog** que muestra un saludo con el nombre ingresado.

2.2.1 Actividades *(En color verde a realizar en casa)*

1. Calculadora básica:
    - Utiliza ventanas emergentes para solicitar al usuario dos números y una operación matemática.
    - Realiza el cálculo correspondiente y muestra el resultado en una ventana emergente.
2. Conversor de unidades:
    - Crea una ventana emergente que solicite al usuario una cantidad en una unidad de medida (por ejemplo, metros).
    - Convierte la cantidad a otra unidad de medida (por ejemplo, centímetros) y muestra el resultado en una ventana emergente.
3. Registro de estudiantes:
    - Utiliza ventanas emergentes para solicitar al usuario el nombre y la edad de un estudiante.
    - Muestra los datos ingresados en una ventana emergente como un registro de estudiante.

4. Validación de contraseña:

    - Crea una ventana emergente que solicite al usuario una contraseña.
    - Valida si la contraseña cumple con ciertos requisitos (por ejemplo, longitud mínima, presencia de letras mayúsculas y minúsculas, números, etc.).
    - Muestra un mensaje en una ventana emergente indicando si la contraseña es válida o no.

5. Juego de adivinanzas:

    - Genera un número aleatorio y utiliza una ventana emergente para solicitar al usuario que adivine el número.
    - Proporciona pistas al usuario (mayor o menor) hasta que adivine correctamente.
    - Muestra un mensaje en una ventana emergente cuando el usuario adivine el número.

3. Tipos de datos primitivos en Java

VARIABLES DE TIPOS PRIMITIVOS.					
Nombre	**Tipo**	**Tamaño**	**Valor por defecto**	**Forma de inicializar**	**Rango**
Boolean	Lógico	1 bit	False	Boolean a=true	True-false
Char	Carácter	16 bits	Null	Char a='Z'	Unicode
Byte	Numero entero	8 bits	0	Byte a =0	-128 a 127
Short	Numero entero	16 bits	0	Short a =12	-32.768 a 32.767
Int	Numero entero	32 bit	0	Int a= 1250	-2.147.483.648 a 2.147.483.649
Long	Numero entero	64 bits	0	Long a= 125000	-9*10^18 a 9*10^18
Float	Numero real	32 bits	0	Float a =3.1	-3,4*10^38 a 3,4*10^38
Double	Numero real	64 bits	0	Double a = 125.2333	-1,79*10^308 a 1,79*10^308

1. **`byte`**: Es un tipo de dato de 8 bits que puede almacenar valores enteros en el rango de -128 a 127.

2. **`short`**: Es un tipo de dato de 16 bits que puede almacenar valores enteros en el rango de -32,768 a 32,767.

3. **`int`**: Es un tipo de dato de 32 bits que puede almacenar valores enteros en el rango de -2,147,483,648 a 2,147,483,647.

4. **`long`**: Es un tipo de dato de 64 bits que puede almacenar valores enteros en un rango mucho mayor, de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.

5. **`float`**: Es un tipo de dato de 32 bits que puede almacenar números decimales de punto flotante con una precisión limitada.

6. **`double`**: Es un tipo de dato de 64 bits que puede almacenar números decimales de punto flotante con mayor precisión que `float`.

7. **`boolean`**: Es un tipo de dato que puede tener dos valores: `true` o `false`. Se utiliza para representar valores lógicos.

8. **`char`**: Es un tipo de dato de 16 bits que puede almacenar un único carácter Unicode.

   Estos tipos de datos primitivos se utilizan para declarar variables y almacenar valores de diferentes tipos en la memoria.

3.1 Tipos numéricos (int, double, etc.)

En Java, los tipos de datos numéricos se dividen en dos categorías: enteros y decimales o de punto flotante. Aquí puedes ver los tipos de datos numéricos en Java:

1. **Tipos de datos enteros**:
    - **`byte`**: 8 bits, rango de -128 a 127.
    - **`short`**: 16 bits, rango de -32,768 a 32,767.
    - **`int`**: 32 bits, rango de -2,147,483,648 a 2,147,483,647.
    - **`long`**: 64 bits, rango de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.
2. **Tipos de datos de punto o coma flotante**:
    - **`float`**: 32 bits, rango de números decimales con precisión limitada.
    - **`double`**: 64 bits, rango de números decimales con mayor precisión que `float`.

Es importante tener en cuenta que los tipos de datos de punto flotante pueden contener valores decimales y enteros, mientras que los tipos de datos enteros solo almacenan valores enteros sin fracciones.

Cuando se realiza la declaración de una variable en un programa, es conveniente **indicar el tipo de datos óptimo** necesario para los valores que deba contener la variable y hacer así un uso óptimo de la memoria RAM del ordenador en el que se pondrá en funcionamiento este programa. Por ejemplo, si se requiere guardar la edad de una persona, será suficiente declarar la variable edad de tipo *byte* (8 bits) y no de cualquier otro tipo de datos numérico: *short*, *int*, *long*, ya que de lo contrario se estará reservando una cantidad de memoria RAM excesiva: 16, 32 o 64 bits, para el uso de la variable edad.

Respecto a los tipos de datos con punto flotante (*float* y *double*), merece la pena matizar que si bien con ambos tipos de datos se representan números con valores decimales, el tipo de datos *double* ofrece la posibilidad de representar un mayor número de valores decimales que el tipo de datos *float*, con otras palabras, el tipo de datos *double* permite representar números decimales con mucha mayor precisión que el tipo de datos *float*.

El programa siguiente muestra en consola el mayor número *float* y el mayor número decimal *double* que se puede representar en Java:

```
float maxFloat = Float.MAX_VALUE;

double maxDouble = Double.MAX_VALUE;

System.out.println(maxFloat); // Imprime: 3.4028235E38

System.out.println(maxDouble); // Imprime: 1.7976931348623157E308
```

3.1.1 Actividades *(En color verde a realizar en casa)*

1. Realiza un programa que calcule la suma, resta, producto y división de los números: 5 y 3. En el caso de la división se deberán mostrar todos los números decimales correspondientes a su cociente.

2. Realiza un programa en el que se declaren , se inicialicen y se muestren por consola las variables correspondientes a los siguientes puntos:

- La edad de una persona es 20 años.
- Variable que permita almacenar cualquier edad de un persona.
- El código correspondiente a un producto siempre será un carácter.
- El código correspondiente al alumno de nombre “Juan Perez” es el ‘D’.
- El nombre del alumno es: “Juan Perez”.
- Se necesita guardar el área de una circunferencia con un único dígito decimal.
- Se necesita guardar el valor del número PI con 20 dígitos decimales.
- El precio de un producto.
- La distancia en metros del planeta Tierra a la galaxia Andromeda.

- Se debe guardar el valor 15.000 en una variable.
- Se debe guardar el valor -12.000 en una variable.
- Se debe guardar el valor 100.000 en una variable.
- Se debe guardar el valor 40.000 en una variable.
- Se debe guardar el valor -134.000 en una variable.
- Se debe de guardar el valor 2.000.000.000 en una variable.
- Se debe de guardar el valor 7.000.000.000 en una variable.
- Se debe de guardar el valor -8.000.000.000 en una variable.
- Se debe de guardar la suma de 29.000 y 30.000 en una variable.
- Se debe de guardar la suma de 2.000.000.000 y 3.000.000.000 en una variable.

3.2 Tipo booleano (boolean)

El tipo de dato *boolean* o lógico en Java se utiliza para representar valores lógicos. Puede tener solo dos posibles valores: `true` o `false`. El tipo *boolean* se utiliza comúnmente en expresiones condicionales y en la toma de decisiones dentro de programas.

Aquí tienes un ejemplo de declaración y asignación de una variable de tipo *boolean* en Java:

```
boolean esMayorDeEdad = true;
```

En este ejemplo, se declara una variable llamada *esMayorDeEdad* de tipo *boolean* y se le asigna el valor *true*, lo que indica que la persona es mayor de edad.

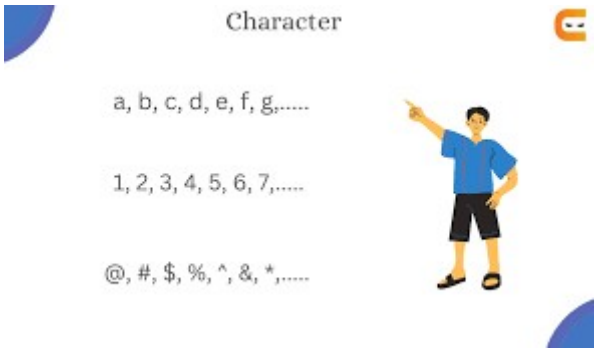
También puedes utilizar operadores lógicos, como *&&* (AND), `||` (OR) y *!* (NOT), para combinar o negar valores boolean. Aquí tienes un ejemplo de uso de operadores lógicos:

```
boolean tieneLicencia = true;
boolean tieneCoche = false;
boolean puedeConducir = tieneLicencia && tieneCoche;
System.out.println(puedeConducir); // Imprime: false
```

En este caso, se combinan las variables *tieneLicencia* y *tieneCoche* utilizando el operador *&&*. La variable `puedeConducir` se establece en `true` solo si ambas variables son `true`. En este caso, como `tieneCoche` es `false`, el resultado es `false`.

El tipo `boolean` también se utiliza en estructuras de control como `if`, `while`, `for` y `switch`, para evaluar condiciones y controlar el flujo del programa en función de los valores `true` o `false`.

3.3 Tipo carácter (char)



En Java, el tipo de datos *char* se utiliza para representar un único carácter Unicode. Es un tipo primitivo que ocupa 16 bits de memoria y se puede utilizar para almacenar cualquier carácter Unicode, incluyendo letras, números, símbolos y caracteres especiales.

Para declarar una variable de tipo char en Java, se utiliza la siguiente sintaxis:

char letra;

También se puede asignar un valor inicial a la variable en el momento de la declaración:

char letra='B';

El valor asignado debe estar encerrado entre comillas simples (') para representar un carácter específico.

Es importante tener en cuenta que el tipo de datos *char* almacena el valor del carácter en su forma numérica, según la tabla de códigos Unicode. Por lo tanto, se pueden realizar operaciones numéricas con variables de tipo `char`. Por ejemplo:

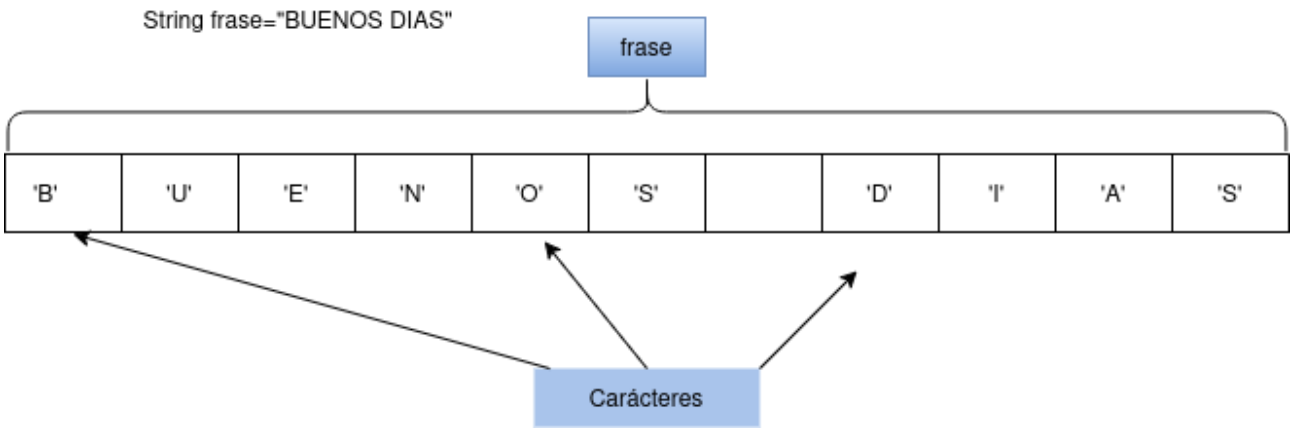
char miChar = 'A';

```
int valorNumerico = miChar; // Se asigna el valor numérico del carácter 'A' (65) a la variable
```

Si deseas obtener el carácter correspondiente a un valor numérico, puedes hacerlo utilizando una conversión (casting) explícita de tipo:

```
int valorNumerico = 65;

char miChar = (char) valorNumerico; // Se asigna el carácter correspondiente al valor numérico 65 ('A') a la variable
```

Recuerda que el tipo de datos *char* representa un solo carácter, mientras que una cadena de caracteres (texto) se representa con el tipo *String* en Java que puede contener un conjunto de caracteres.



3.3.1 Caracteres Unicode

Un carácter Unicode es un estándar de codificación que **asigna un número único a cada carácter** utilizado en la mayoría de los sistemas de escritura del mundo, incluyendo letras, números, símbolos y caracteres especiales. Unicode busca ser una solución universal para representar y procesar todos los caracteres de diferentes lenguajes y sistemas de escritura.

El estándar Unicode utiliza un espacio de código de 21 bits, lo que permite representar hasta 2^21 caracteres distintos. En la actualidad, Unicode ha asignado números a más de 143,000 caracteres de distintas escrituras, incluyendo el alfabeto latino, caracteres cirílicos, caracteres chinos, caracteres árabes, entre otros.

Cada carácter Unicode se identifica mediante un código numérico llamado "punto de código" (code point). Por ejemplo, el punto de código para la letra "A" mayúscula es U+0041. La notación U+ seguida de cuatro o más dígitos hexadecimales representa un punto de código Unicode.

La ventaja de utilizar Unicode es que proporciona una forma consistente de representar y manejar texto en diferentes idiomas y sistemas de escritura. Esto facilita la interoperabilidad entre sistemas informáticos y permite que las aplicaciones sean más accesibles y globales.

En Java, el tipo de datos ***char*** utiliza la codificación Unicode, lo que significa que se pueden representar caracteres de cualquier sistema de escritura utilizando este tipo de dato.

3.3.2 Actividades *(En color verde a realizar en casa)*

1. Realiza un programa que guarde en las correspondientes variables los valores siguientes: ‘d’, ‘H’, ‘D’, ‘A’, ‘a’, “Hola”, “Joan Manuel”. Finalmente el programa visualizará en consola el contenido de cada una de las anteriores variables.

2. Realiza un programa que guarde en las correspondientes variables los valores siguientes: ‘a’, ‘b’, ‘c’, ‘A’, ‘C’, “Adios”, “Pedro Jesús”. Finalmente el programa visualizará en consola el contenido de cada una de las anteriores variables.

3.3.3 Tabla ASCII versus Unicode

Unicode no es lo mismo que la tabla ASCII. La **tabla ASCII** (American Standard Code for Information Interchange) es un estándar más antiguo que define un conjunto de 128 caracteres (7 bits) que incluye letras mayúsculas y minúsculas, dígitos numéricos, caracteres de puntuación y algunos caracteres especiales.

El programa siguiente muestre la tabla ASCII de los 128 primeros caracteres:

```
public class tasccii {

	public static void main(String[] args) {

		char letra;

for (int i='A';i<='z';i++)

{

		System.out.println(i+" "+"="+(char)i);

}

}

}
```

Lanzar el programa

Por otro lado, Unicode es un estándar más amplio y moderno que busca abarcar todos los caracteres utilizados en los diferentes sistemas de escritura del mundo. Unicode utiliza un espacio de código de 21 bits, lo que permite representar más de 143,000 caracteres diferentes.

Unicode incluye una parte llamada el "Plano básico multilingüe" (BMP, Basic Multilingual Plane), que es similar a la tabla ASCII y contiene los primeros 65,536 caracteres (puntos de código U+0000 a U+FFFF). **Los primeros 128 caracteres de Unicode en el BMP coinciden con los de la tabla ASCII, lo que hace que la codificación ASCII sea una subconjunto de Unicode.**

Sin embargo, Unicode va más allá de los 128 caracteres ASCII y proporciona representaciones para una amplia gama de idiomas y sistemas de escritura, incluyendo caracteres no latinos como chino, árabe, griego, cirílico, entre otros.

En resumen, Unicode es un estándar más amplio y moderno que abarca muchos más caracteres que la tabla ASCII, permitiendo la representación de una amplia variedad de sistemas de escritura y caracteres especiales.

3.3.4 Actividades *(En color verde a realizar en casa)*

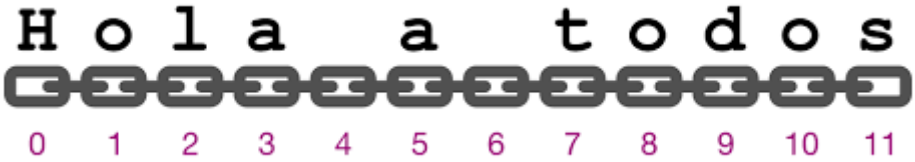
1. Realiza un programa que muestre el código ASCII correspondiente a las letras: a, b, d, h, m, v.

2. Realiza un programa que muestre el carácter ASCII correspondiente a los códigos: 65, 69, 70, 92, 100

3. Modifica el ejemplo de la actividad 1, de forma que el código ASCII sea mostrado en orden inverso al que aparecía.

4. Realiza un programa que solicite al usuario un total de 4 letras, por cada letra introducida se mostrará su correspondiente código ASCII.

4. Tipo de datos compuesto: Cadena de caracteres, String



H	o	l	a		a		t	o	d	o	s
0	1	2	3	4	5	6	7	8	9	10	11

Nota

Hay que tener en cuenta que el primer caracter en una variable de tipo String se encuentra en la posición (índice) 0 y no en la 1.

En el apartado 2.3 se habla del tipo de datos *char* utilizado en variables que guardan en ellas únicamente un carácter. El tipo de datos *String* estará compuesto por un conjunto de caracteres, o como una agrupación de datos de tipo *char*.

En Java, una cadena de caracteres se representa utilizando el tipo de datos `String`. Una cadena de caracteres es una secuencia de caracteres alfanuméricos y se utiliza para almacenar y manipular texto en Java.

Aquí tienes algunos ejemplos de cómo trabajar con cadenas de caracteres en Java:

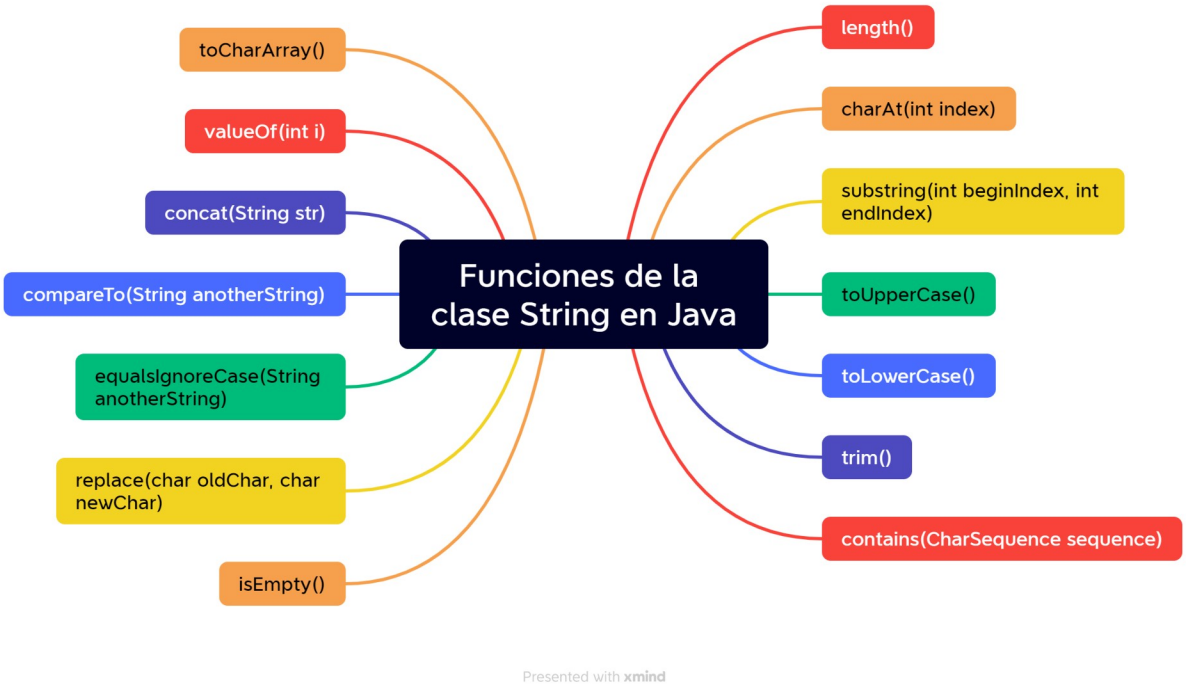
Declaración de una cadena de caracteres:

```
String miCadena = "Hola, mundo!";
```

Concatenación de cadenas:

```
String cadena1 = "Hola";

String cadena2 = "mundo";

String concatenada = cadena1 + ", " + cadena2; // Resultado: "Hola, mundo"
```

4.1 Funciones para el tipo String



La clase *String* en Java proporciona una amplia gama de métodos y funciones para trabajar con cadenas. Aquí hay algunos ejemplos de las funciones más comunes que puedes utilizar:

length(): Devuelve la longitud de la cadena.

```
String miCadena = "Hola, mundo!";
int longitud = miCadena.length(); // Resultado: 13
```

charAt(index): Devuelve el carácter en el índice especificado.

```
String miCadena = "Hola, mundo!";
char primerCaracter = miCadena.charAt(0); // Resultado: 'H'
char tercerCaracter = miCadena.charAt(2); // Resultado: 'l'
```

substring(beginIndex, endIndex): Devuelve una subcadena que abarca desde beginIndex (inclusive) hasta endIndex (sin incluir).

```
String miCadena = "Hola, mundo!";
String subcadena = miCadena.substring(0, 4); // Resultado: "Hola"
```

toLowerCase(): Devuelve una nueva cadena con todos los caracteres convertidos a minúsculas.

```
String miCadena = "Hola, mundo!";
String enMinusculas = miCadena.toLowerCase(); // Resultado: "hola, mundo!"
```

equals(anotherString): Compara la cadena actual con `anotherString` y devuelve `true` si son iguales.

```
String cadena1 = "Hola";
String cadena2 = "Hola";
boolean sonIguales = cadena1.equals(cadena2); // Resultado: true
```

4.1.1 Actividades *(En color verde a realizar en casa)*

1. Contar letras: Crea un programa que solicite al usuario que ingrese una palabra o frase y luego cuente la cantidad de letras en ella utilizando el método `length()`.

2. Realiza un programa que solicite al usuario la introducción de una palabra (sc.next()) y muestre los caracteres de la misma que se encuentran en la posición (índice) 0 y en la última posición.

3. Realiza un programa que solicite al usuario la introducción de una palabra (sc.next()) y muestre el caracter ubicado en la posición central de la palabra.

4. Conversión de mayúsculas y minúsculas: Escribe un programa que le pida al usuario que ingrese una palabra y luego la imprima en mayúsculas y minúsculas utilizando los métodos *`toUpperCase()`* y *`toLowerCase()`*.

5. Eliminar espacios en blanco: Crea un programa que solicite al usuario que ingrese una frase con espacios en blanco al principio y al final de la misma, luego imprima la misma frase sin los espacios en blanco existentes en los extremos `trim()`.

6. Reemplazar caracteres: Escribe un programa que tome una oración ingresada por el usuario y reemplace todas las letras 'a' por la letra 'e' utilizando el método `replace()`.

7. El programa siguiente elimina todos los espacios en blanco existentes en la frase, incluidos los que se encuentran entre palabra y palabra, para ello se hará uso del método replaceAll:

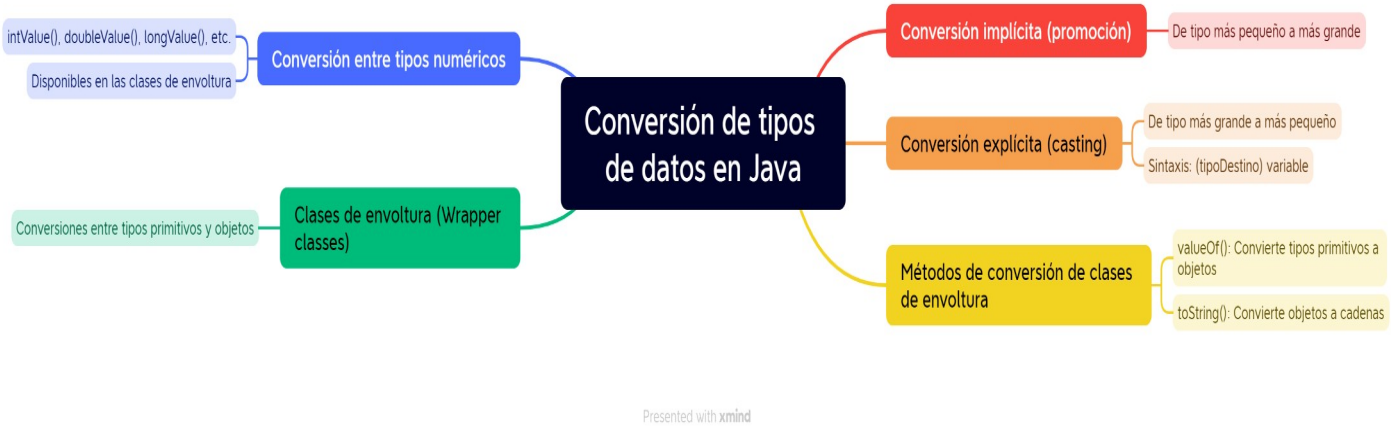
```
import java.util.Scanner;

public class EliminarEspacios {
    public static void main(String[] args) {
        // Crear un objeto Scanner para leer la entrada del usuario
        Scanner scanner = new Scanner(System.in);

        // Solicitar al usuario que ingrese una frase
        System.out.println("Por favor, ingresa una frase con espacios:");
        String frase = scanner.nextLine();

        // Eliminar todos los espacios en blanco de la frase
        String fraseSinEspacios = frase.replaceAll("\\s+", "");

        // Imprimir la frase sin los espacios
        System.out.println("Frase sin espacios: " + fraseSinEspacios);

        // Cerrar el scanner
        sc.close();
    }
}
```

Lanzar el programa

8. Dividir y unir palabras: Crea un programa que solicite al usuario que ingrese una frase y luego la divida en palabras individuales utilizando el método `split()`

9. Realiza un programa que solicite al usuario la introducción de dos nombres de personas. Finalmente el programa deberá indicar si los dos nombres insertados son iguales o diferentes.

5. Conversión de tipos de datos



En Java, puedes realizar conversiones entre diferentes tipos de datos utilizando los siguientes métodos y operadores:

1. Conversión implícita (promoción): Java realiza conversiones implícitas cuando se asigna un valor de un tipo de dato más pequeño a un tipo de dato más grande. Por ejemplo:

```
int numeroEntero = 5;
double numeroDecimal = numeroEntero; // Conversión implícita de int a double
```

2. Conversión explícita (casting): Si deseas convertir un tipo de dato más grande a uno más pequeño, o cuando no hay una conversión implícita disponible, puedes usar el casting para realizar la conversión explícitamente. Por ejemplo:

```
double numeroDecimal = 3.14;
int numeroEntero = (int) numeroDecimal; // Conversión explícita de double a int
```

3. Clases de envoltura (Wrapper classes): Las clases de envoltura en Java proporcionan métodos para convertir entre tipos primitivos y objetos. Por ejemplo:

```
String numeroTexto = "123";
int numero = Integer.parseInt(numeroTexto); // Convierte una cadena a int
```

4. Métodos de conversión de clases de envoltura: Las clases de envoltura también proporcionan métodos como **valueOf()** para convertir tipos primitivos a objetos y **toString()** para convertir objetos a cadenas. Por ejemplo:

```
int numero = 42;
String numeroTexto = Integer.toString(numero); // Convierte int a String
```

5. Conversión entre tipos numéricos: Puedes realizar conversiones entre diferentes tipos numéricos utilizando métodos como **intValue()**, **doubleValue()**, **longValue()**, etc., disponibles en las clases de envoltura. Por ejemplo:

```
double numeroDouble = 3.14;
int numeroEntero = Double.valueOf(numeroDouble).intValue(); // Convierte double a int
```

Recuerda que algunas conversiones pueden implicar pérdida de precisión o información. Es importante tener en cuenta los límites y restricciones de cada tipo de dato al realizar conversiones.

5.1 Actividades *(En color verde a realizar en casa)*

1. Solicita al usuario que ingrese un número entero y conviértelo a double. Luego, imprime el resultado.

2. Pide al usuario que ingrese un número decimal y conviértelo a int. Después, muestra el número entero resultante.

3. Declara una variable float e inicialízala con el valor 10,87. Luego, conviértela a un tipo int y asigna el resultado a otra variable de tipo Int. Imprime el contenido de ambas variables.

4. Crea una variable double con valor 3.7. Conviértela a int utilizando casting y guarda el resultado en otra variable. Imprime el valor entero resultante.

5. Declara una variable int con valor 65. Conviértela a char utilizando casting y almacena el resultado en otra variable. Imprime el carácter resultante.

6. Solicita al usuario que ingrese un número entero representado como una cadena. Convierte la cadena a int utilizando el método parseInt() y almacena el resultado en una variable. Realiza algún cálculo con el número y muestra el resultado.

7. Pide al usuario que ingrese un número decimal en formato de cadena. Convierte la cadena a double utilizando el método parseDouble() y realiza alguna operación matemática utilizando el número. Imprime el resultado.

6. Operador de incremento/decremento

Los **operadores de incremento y decremento** son operadores unarios en Java (y en muchos otros lenguajes de programación) que se utilizan para aumentar o disminuir el valor de una variable numérica en una unidad.

Operadores de incremento:

- **++**: Incrementa el valor de la variable en 1.
    - **Postfijo (n++)**: Incrementa el valor de la variable, pero devuelve el valor antes de incrementarlo.
    - **Prefijo (++n)**: Incrementa el valor de la variable y devuelve el valor después de incrementarlo.

Ejemplo:

```
int n = 5;

n++;   // n es ahora 6 (incremento postfijo)

++n;   // n es ahora 7 (incremento prefijo)
```

Operadores de decremento:

- **- -**: Decrementa el valor de la variable en 1.
    - **Postfijo (n- -)**: Decrementa el valor de la variable, pero devuelve el valor antes de decrementarlo.
    - **Prefijo (- -n)**: Decrementa el valor de la variable y devuelve el valor después de decrementarlo.

Ejemplo:

```
int n = 5;

n--;   // n es ahora 4 (decremento postfijo)

--n;   // n es ahora 3 (decremento prefijo)
```

6.1 Asignación con incremento/decremento postfijo

La **asignación con incremento postfijo** es una operación en programación que combina dos acciones: la asignación de un valor a una variable y el incremento del valor de otra variable utilizando el operador de incremento en su forma postfija.

Características:

1. **Operador Postfijo**: El operador ++ se coloca **después** de la variable que se está incrementando. Esto indica que el valor actual de la variable se utiliza primero antes de que se aplique el incremento.

2. **Proceso**:

    - El valor de la variable es **evaluado y utilizado** en la expresión antes de ser incrementado.

    - Luego, la variable se **incrementa en 1**.

Ejemplo:

```
int n = 5;

int total;

total = n++; // total se convierte en 5 (valor actual de n) y n se incrementa a 6
```

Este código es equivalente al siguiente código:

```
int n = 5;

int total;

total=n;

n=n+1;
```

La **asignación con decremento postfijo** es una operación en programación que combina dos acciones: la asignación de un valor a una variable y el decremento del valor de otra variable utilizando el operador de decremento en su forma postfija.

Características:

1. **Operador Postfijo**: El operador `--` se coloca **después** de la variable que se está decrementando. Esto indica que el valor actual de la variable se utiliza primero antes de que se aplique el decremento.
2. **Proceso**:
    - El valor de la variable es **evaluado y utilizado** en la expresión antes de ser decrementado.
    - Luego, la variable se **decrementa en 1**.

Ejemplo:

```
int n = 5;

int total;

total = n--; // total se convierte en 5 (valor actual de n) y n se decrementa a 4
```

Este código es equivalente al siguiente código:

```
int n = 5;

int total;

total=n;

n=n-1;
```

6.2 Asignación con incremento/decremento prefijo

La **asignación con incremento prefijo** es una operación en programación que combina dos acciones: la asignación de un valor a una variable y el incremento del valor de otra variable utilizando el operador de incremento en su forma prefija.

Características:

1. **Operador Prefijo**: El operador `++` se coloca **antes** de la variable que se está incrementando. Esto indica que el incremento se realiza antes de que se utilice el valor en la expresión.
2. **Proceso**:
    - La variable se **incrementa en 1**.
    - Luego, el nuevo valor incrementado se **asigna** a la otra variable.

Ejemplo:

```
int n = 5;

int total;

total = ++n; // n se incrementa a 6 y total también se convierte en 6
```

Este código es equivalente al siguiente código:

```
int n = 5;

int total;

n=n+1;

total=n;
```

La **asignación con decremento prefijo** es una operación en programación que combina dos acciones: la asignación de un valor a una variable y el decremento del valor de otra variable utilizando el operador de decremento en su forma postfija.

Características:

3. **Operador Prefijo**: El operador `--` se coloca **antes** de la variable que se está decrementando. Esto indica que primero se decrementa en una unidad el valor de lavariable y después, dicho valor decrementado se asigna a la segunda variable.
4. **Proceso**:
    - La variable se **decrementa en 1**.
    - La segunda variable recibe el valor de la primera variable una vez esta se decrementó en una unidad.

Ejemplo:

```
int n = 5;

int total;

total = --n; // n se decrementa a 4 y total se convierte en 4 (valor actual de n después de restarle una unidad)
```

Este código es equivalente al siguiente código:

```
int n = 5;

int total;

n=n-1;

total=n;
```

6.2.1 Actividades

1. Incremento simple

Declarar una variable entera `x = 5`. Usar `x++` y mostrar el valor antes y después del incremento.

2. Decremento simple

Declarar una variable entera `y = 10`. Usar `y--` y mostrar el valor antes y después del decremento.

3. Pre-incremento

Declarar una variable `a = 7`. Usar `++a` y mostrar el resultado. Comparar con el valor inicial.

4. Pre-decremento

Declarar una variable `b = 3`. Usar `--b` y mostrar el resultado. Comparar con el valor inicial.

5. Post-incremento en asignación

Declarar dos variables `x = 4` y `z`. Asignar `z = x++` y mostrar los valores finales de `x` y `z`.

6. Pre-incremento en asignación

Declarar dos variables `m = 6` y `n`. Asignar `n = ++m` y mostrar los valores finales de `m` y `n`.

7. Expresión combinada

Declarar `p = 5`. Calcular `int r = ++p + p++` y mostrar los valores de `p` y `r`.

8. Serie de incrementos y decrementos

Declarar `k = 10`. Aplicar en secuencia: `k++`, `++k`, `k--`, `--k` y mostrar el valor de `k` tras cada operación.

9. Operadores en una expresión aritmética

Declarar `x = 2`, `y = 3`. Calcular `int res = x++ * --y;` y mostrar los valores finales de `x`, `y` y `res`.

10. Múltiples incrementos en una sola línea

Declarar `a = 1`. Calcular `int total = a++ + ++a + a++;` y mostrar el valor de `a` y `total`.

7. Operadores de asignación compuesta: suma, resta, multiplicación, división

Un operador de asignación compuesta es un tipo de operador en programación que combina una operación aritmética (como suma, resta, multiplicación, división, etc.) con la asignación de un valor a una variable. Estos operadores permiten realizar una operación y asignar el resultado a la misma variable en una sola expresión, haciendo el código más conciso y legible.

Características:

1. **Combinación de Operaciones**: Realizan una operación y al mismo tiempo asignan el resultado a la variable. Por ejemplo, en n += 2, se suma 2 al valor actual de n y se asigna el resultado nuevamente a n. Equivale a escribir: n=n+2

2. **Sintaxis Simplificada**: Ofrecen una forma más compacta de escribir operaciones aritméticas que incluyen asignaciones.

3. **Usabilidad**: Se utilizan comúnmente en bucles, cálculos acumulativos y actualizaciones de estado para hacer el código más eficiente y menos propenso a errores.

Ejemplos de Operadores de Asignación Compuesta:

- **Suma**: n += 2; (equivalente a n = n + 2;)

- **Resta**: n -= 2; (equivalente a n = n - 2;)

- **Multiplicación**: n *= 2; (equivalente a n = n * 2;)

- **División**: n /= 2; (equivalente a n = n / 2;)

- **Módulo**: n %= 2; (equivalente a n = n % 2;)

Resumen:

> Los operadores de asignación compuesta son una herramienta poderosa en programación que simplifican la manipulación de variables al permitir la combinación de operaciones y asignaciones en una sola línea, lo que mejora la legibilidad y reduce la posibilidad de errores.

8. Generación de un número aleatorio

La **aleatoriedad en Java** se maneja principalmente a través de la clase `Random`, que forma parte del paquete `java.util`. Esta clase permite generar números aleatorios de varios tipos, como enteros, decimales y booleanos.

Ejemplos:

- **Generar un número entero aleatorio:**

  ```
  Random random = new Random();

  int numero = random.nextInt(10); // Entre 0 y 9
  ```

- **Generar un número decimal con `Math.random()`:**

  ```
  double numero = Math.random(); // Entre 0.0 y 1.0
  ```

La aleatoriedad en Java se basa en algoritmos pseudoaleatorios, lo que significa que los números generados parecen aleatorios, pero en realidad son el resultado de cálculos deterministas basados en una semilla inicial (`seed`).

- El programa siguiente genera un número aleatorio entre 10 y 20:

```
import java.util.Random;

public class NumeroAleatorio {

    public static void main(String[] args) {

        Random random = new Random();

            // Genera un número aleatorio entre 10 y 20 inclusivos

        int numeroAleatorio = random.nextInt(11) + 10;

// nextInt(11) genera de 0 a 10, +10 lo ajusta a 10-20

        System.out.println("Número aleatorio entre 10 y 20: " + numeroAleatorio);

    }

}
```

Lanzar el programa

- **Generación de un número aleatorio entre dos números (mínimo y máximo)**

```
import java.util.Scanner;

import java.util.Random;

public class GeneradorNumeroAleatorio {

public static void main(String[] args) {

// Crear un objeto Scanner para obtener la entrada del usuario

Scanner sc = new Scanner(System.in);

// Solicitar el valor mínimo

System.out.print("Ingresa el valor mínimo: ");

int minimo = sc.nextInt();
// Solicitar el valor máximo
System.out.print("Ingresa el valor máximo: ");
int maximo = sc.nextInt();
// Crear un objeto Random para generar números aleatorios
Random random = new Random();
// Generar el número aleatorio entre el mínimo y el máximo (incluyendo ambos)
int numeroAleatorio = random.nextInt(maximo - minimo + 1) + minimo;
// Mostrar el número aleatorio generado
System.out.println("El número aleatorio generado es: " + numeroAleatorio);
// Cerrar el scanner
sc.close();}}
```

Lanzar el programa

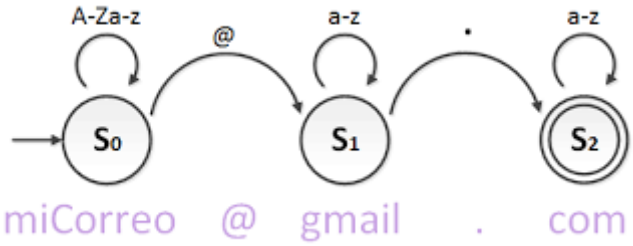
8.1 Actividades

1. Realiza un programa que genere dos números aleatorios entre un valor máximo y mínimo indicado por el usuario. A continuación, el programa deberá mostrar los dos números generados aleatoriamente y calcular y mostrar la suma, la resta, la multiplicación y la división de ambos números.

2. Realiza un programa que genere un número aleatorio entre el 3 y el 6. A continuación el programa deberá mostrar el número generado aleatoriamente incrementado en una unidad.

3. Realiza un programa que genere y visualice en consola un número aleatorio entre 0 y 1 sin incluir el 1.

9. Expresión regular



Una expresión regular, también conocida como regex o regexp (abreviatura de regular expression en inglés), es una secuencia de caracteres que define un patrón de búsqueda en un texto. Se utiliza principalmente para realizar operaciones de búsqueda, manipulación y validación de cadenas de texto.

Las expresiones regulares se componen de caracteres literales y metacaracteres que tienen un significado especial. Los caracteres literales representan símbolos específicos que se deben buscar en un texto, como letras, números o caracteres especiales. Los metacaracteres, por otro lado, proporcionan funcionalidades más avanzadas, como la búsqueda de patrones específicos de caracteres, la coincidencia de repeticiones, la búsqueda de rangos de caracteres y más.

Por ejemplo, la expresión regular "\d{3}" representa un patrón que busca tres dígitos consecutivos en un texto. Esto coincidiría con cualquier secuencia de tres dígitos, como "123", "456" o "789". Las expresiones regulares pueden volverse mucho más complejas, permitiendo realizar búsquedas y manipulaciones sofisticadas en los textos.

Las expresiones regulares se utilizan en una amplia variedad de lenguajes de programación, herramientas de manipulación de texto y aplicaciones informáticas en general. Son una herramienta poderosa y flexible para trabajar con cadenas de texto y proporcionan una forma concisa de especificar patrones de búsqueda y manipulación.

Veamos el ejemplo siguiente de expresión regualar en java:

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class ExpresionRegularJava {
    public static void main(String[] args) {
        // Definir el patrón de la expresión regular
        String patron = "\\b\\d{3}\\b";

        // Crear el objeto Pattern
        Pattern pattern = Pattern.compile(patron);

        // Definir la cadena de texto en la que buscar
        String texto = "123 456 789";

        // Crear el objeto Matcher
        Matcher matcher = pattern.matcher(texto);

        // Realizar la búsqueda y encontrar coincidencias
        while (matcher.find()) {
            String coincidencia = matcher.group();
            System.out.println("Coincidencia encontrada: " + coincidencia);
        }
    }
}
```

Lanzar el programa

En este ejemplo, la expresión regular "\b\d{3}\b" busca secuencias de tres dígitos completas en una cadena de texto. El metacaracter "\b" representa un límite de palabra, lo que asegura que solo se consideren secuencias de tres dígitos completas y no partes de secuencias más largas. El metacaracter "\d" representa cualquier dígito del 0 al 9, y "{3}" indica que debe haber exactamente tres dígitos en la secuencia.

La cadena de texto en este caso es "123 456 789". Al ejecutar el código, se encontrarán tres coincidencias de tres dígitos: "123", "456" y "789". Cada coincidencia se imprimirá en la salida.

9.1. Actividades *(En color verde a realizar en casa)*

1. Validar una dirección de correo electrónico:
    - Crea una expresión regular que verifique si una cadena de texto coincide con el formato de una dirección de correo electrónico válida.
    - Por ejemplo, verifica si "ejemplo@dominio.com" es una dirección de correo electrónico válida.
2. Extraer números de teléfono:
    - Dada una cadena de texto que contiene información de contacto, crea una expresión regular para extraer todos los números de teléfono presentes.
    - Por ejemplo, extrae los números de teléfono de la cadena "Mi número de teléfono es 123-456-7890 y también tengo otro número 987-654-3210".
3. Validar una contraseña segura:
    - Crea una expresión regular que verifique si una cadena de texto cumple con los requisitos de una contraseña segura.
    - Por ejemplo, verifica si "Abc123!$" es una contraseña segura que contenga al menos una letra mayúscula, una letra minúscula, un número y un carácter especial.

4. Separar palabras en una oración:

    - Dada una oración en forma de cadena de texto, crea una expresión regular para dividir la oración en palabras individuales.
    - Por ejemplo, separa la oración "Hola, cómo estás?" en palabras individuales: "Hola", "cómo" y "estás".

5. Validar un formato de fecha:

    - Crea una expresión regular que verifique si una cadena de texto cumple con un formato de fecha específico, como "dd/mm/aaaa".
    - Por ejemplo, verifica si "25/12/2022" es una fecha válida en el formato especificado.

Listado de programas