

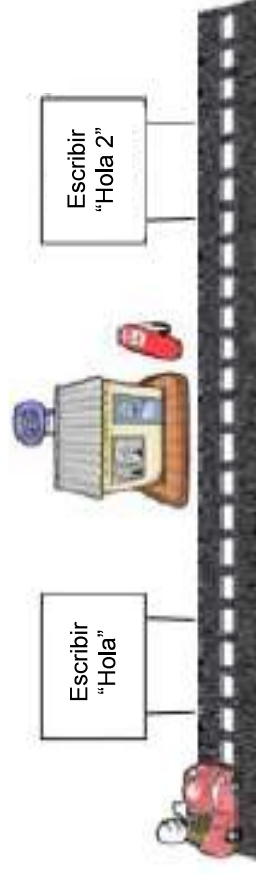
**Técnicas de Programación**

# **Carrera Programador full-stack**

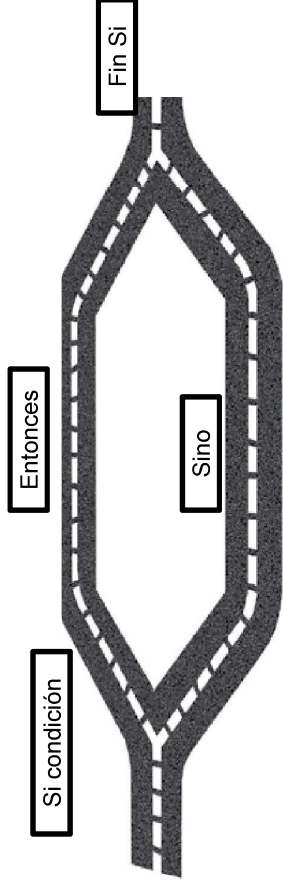
***Repetición (Repaso)***

# Estructuras de Control

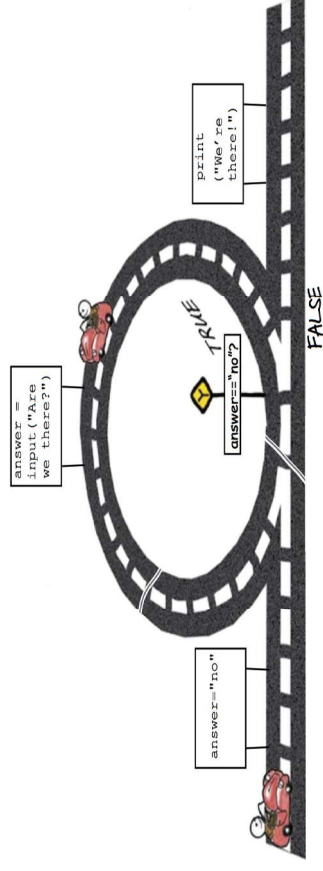
Secuenciales



Selectivas o De  
Decisión



Repetitivas



# Estructuras de Control

## *Instrucción While*

- La instrucción **while** ejecuta una secuencia de instrucciones mientras una condición sea verdadera



- También llamados iteraciones o “loops” en Inglés
- Sirven para ejecutar código varias veces
- La condición se verifica al principio
- La cantidad de veces ejecutado depende de una condición (puede que no se ejecute ninguna vez)

# Estructuras de Control

## *Instrucción For*



- La instrucción **for** ejecuta una secuencia de instrucciones utilizando contadores con principio, incrementos y final
  - Los bucles For son útiles cuando hay que hacer un conteo (fijo, en términos de “n” veces)
  - El valor inicial del conteo, la condición de corte y el incremento del contador se definen en una sola instrucción.
  - La declaración de la variable debe realizarse antes

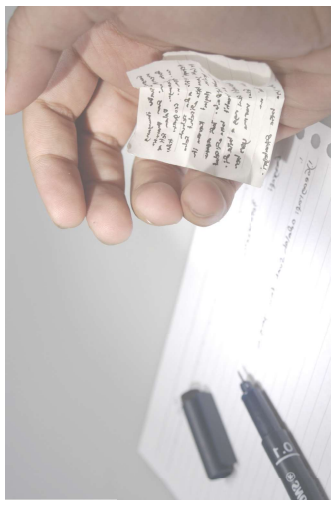
# Estructuras de Control

## *Guía Memoria*

```
while while (<condición>) {  
    <instrucciones>  
}
```

---

```
for (for <var> = <inicial>; <var> <= <final>; <var>++) {  
    <instrucciones>  
}
```



**Técnicas de Programación**

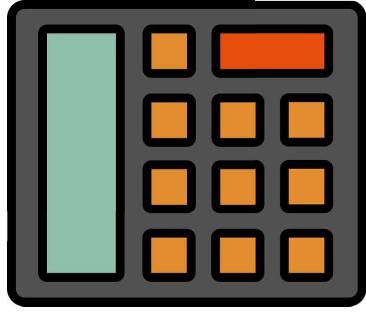
# **Carrera Programador full-stack**

***Modularización y Métodos (Conceptos)***

# Modularización

## *Implementar una Calculadora*

- Realice una calculadora que sume o reste según el pedido del usuario.
- El usuario deberá ingresar 2 números por teclado
- Luego ingresará una opción:
  - Si ingresa 1 los números se sumarán
  - Si ingresa 2 los números se restarán
  - Si ingresa cualquier otra tecla termina el programa
  - Para informar el resultado de la operación debe usar el siguiente formato (*40 guiones '-'*):



-----  
El resultado de la operación es: X  
-----

# Modularización

## *Implementar una Calculadora*

- Definimos las variables y leemos desde teclado

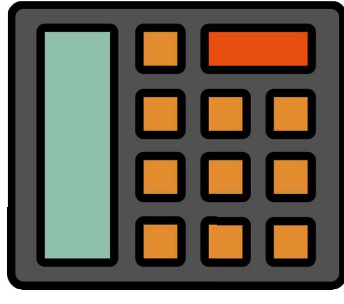
```
import * as rls from 'readline-sync';
```

```
let i : number, linea : string;
```

```
let numero1 : number = rls.questionInt("Ingrese un número: ");
```

```
let numero2 : number = rls.questionInt("Ingrese un número: ");
```

```
let opcionMenu : number = rls.questionInt("Ingrese 1 para sumar, 2 para restar, cualquier  
otra tecla para salir: ");
```





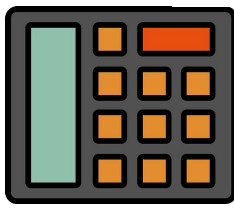
# Modularización

## *Implementar una Calculadora*

- Realizamos la operación según la opción

```
if (opcionMenu == 1) {  
  linea = "-";  
  for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
  console.log("el resultado es: ", numero1 + numero2);  
  linea = "-";  
  for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
}
```

```
else if (opcionMenu == 2) {  
  linea = "-";  
  for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
  console.log("el resultado es: ", numero1 - numero2);  
  linea = "-";  
  for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
}
```



# Modularización

## *Implementar una Calculadora*

- Hay código repetido!

```
if (opcionMenu == 1) {  
  linea = "";  
  for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
}
```

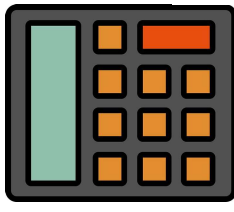
```
console.log("el resultado es: ", numero1 + numero2);
```

```
linea = "";  
for (i = 0; i <= 40; i++) {  
  linea = linea + "-";  
};  
console.log(linea);  
}
```

```
else if (opcionMenu == 2) {  
  linea = "";  
  for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
}
```

```
console.log("el resultado es: ", numero1 - numero2);
```

```
linea = "";  
for (i = 0; i <= 40; i++) {  
  linea = linea + "-";  
};  
console.log(linea);  
}
```



# Código Repetido

- **No debemos** duplicar el código
- Cuando tenemos la **misma funcionalidad** en distintas partes del programa debemos **reutilizar**
- ¿Cuál es la diferencia entre copiar y reutilizar?
  - Una sola copia que puede ser llamada donde lo necesitemos
  - Programas más cortos
  - Cambios acotados a un solo lugar



# Código Repetido

- Cambio acotado en un solo lugar (una opción)

```
linea = "";  
for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
};  
if (opcionMenu == 1) {
```

```
    console.log(linea);
```

```
    console.log("el resultado es: ", numero1 + numero2);
```

```
    console.log(linea);
```

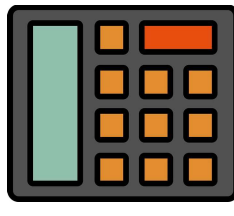
```
} else if (opcionMenu == 2) {
```

```
    console.log(linea);
```

```
    console.log("el resultado es: ", numero1 - numero2);
```

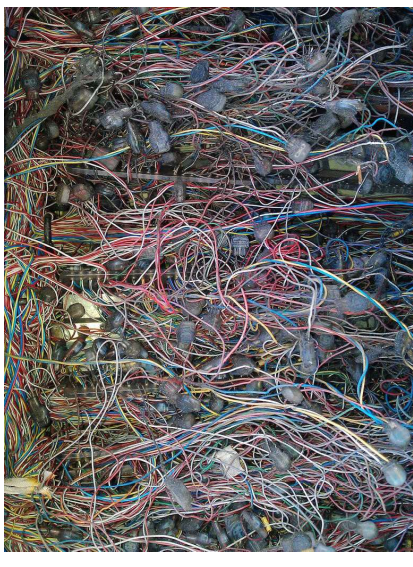
```
    console.log(linea);
```

```
}
```

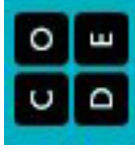


# Funciones o Métodos

- Por ahora **función** y **método** es lo mismo.
- Cuando los programas crecen se vuelven más complejos:
  - Necesitamos manejar la complejidad
- Debemos **agrupar** las sentencias que tienen **cohesión**
  - Mayor legibilidad
  - Mayor mantenibilidad



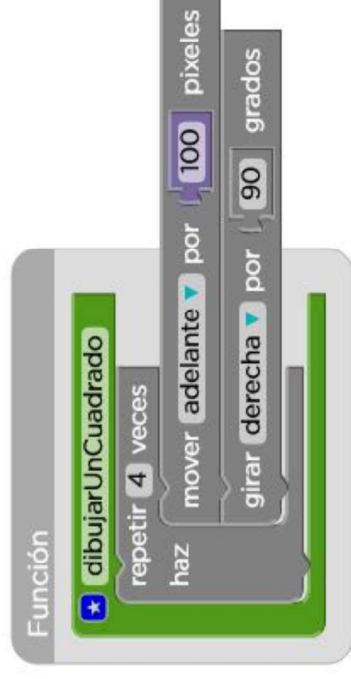
# Comparativa code.org



En este ejemplo de programa de bloques de code.org, la función se llama “dibujarUnCuadrado” y dibuja un cuadrado de tamaño 100.

```
function dibujarUnCuadrado() {  
  //Dibuja un cuadrado de tamaño fijo en 100  
  for (let count : number = 0; count < 4; count++) {  
    moverAdelante(100);  
    girarADerecha(90);  
  }  
}  
  
dibujarUnCuadrado();
```

cuando se ejecuta  
dibujarUnCuadrado



# Métodos

## *Características*

- Los **métodos**:
  - Poseen un conjunto de sentencias de código
  - Tienen un nombre (suelen ser verbos)
  - Pueden ser invocados
  - Pueden devolver un valor

```
function dibujarGuiones() {  
  let i : number;  
  let linea : string = "";  
  for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
}
```

# Funciones

## Sintaxis

- Las Funciones se pueden declarar de 3 maneras

<i>// como sentencia</i> <b>function</b> otraFuncion () { <b>console.log</b> ("Hola!"); } ... otraFuncion();	<i>// como valor de una variable</i> <b>let</b> miFuncion = <b>function</b> () { <b>console.log</b> ("Hola!"); } ... miFuncion();	<i>// como arrow function</i> <b>let</b> arrowFunction = () => { <b>console.log</b> ("Hola!"); } ... arrowFunction();
---	--	--

función  
*function*

expresión de función  
*function expression*

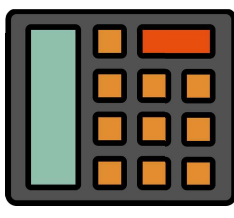
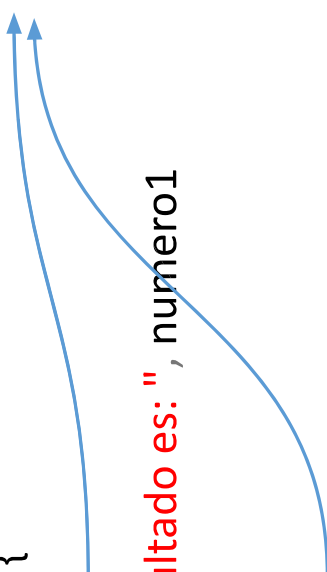
función flecha  
*arrow function*



# Métodos

- Cada vez que se encuentra una llamada a un **método**:
  - El programa ejecuta el código del método hasta que termina
  - Vuelve a la siguiente línea del lugar donde partió

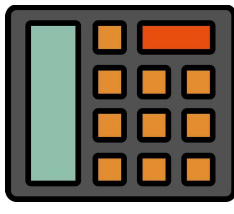
```
if (opcionMenu == 1) {  
  dibujarGuiones();  
  console.log("el resultado es: ", numero1  
+ numero2);  
  dibujarGuiones();  
}  
  
function dibujarGuiones () {  
  let i : number;  
  let linea : string = "";  
  for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
}
```



# Modularización

*Implementar una Calculadora*

- Implemente un método llamado dibujarGuiones para evitar el código repetido

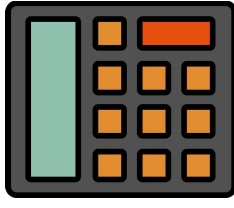


# Modularización

## *Implementar una Calculadora*

- Implemente un método llamado dibujarGuiones para evitar el código repetido

```
if (opcionMenu == 1) {  
  dibujarGuiones();  
  console.log("el resultado es: ", numero1 + numero2);  
  dibujarGuiones();  
} else if (opcionMenu == 2) {  
  dibujarGuiones();  
  console.log("el resultado es: ", numero1 - numero2);  
  dibujarGuiones();  
}  
  
function dibujarGuiones () {  
  let i : number;  
  let linea : string = "";  
  for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
}
```



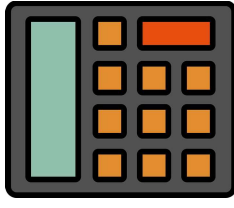
# Modularización

## *Implementar una Calculadora*

- Implemente un método llamado dibujarGuiones para evitar el código repetido

```
if (opcionMenu == 1) {  
  dibujarGuiones();  
  console.log("el resultado es: ", numero1 + numero2);  
  dibujarGuiones();  
} else if (opcionMenu == 2) {  
  dibujarGuiones();  
  console.log("el resultado es: ", numero1 - numero2);  
  dibujarGuiones();  
}  
  
function dibujarGuiones () {  
  let i : number;  
  let linea : string = "";  
  for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
}
```

¿Y si queremos que en un caso se dibujen 40 guiones y en otro 30?



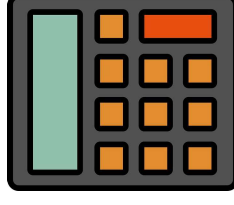
# Modularización

## *Implementar una Calculadora*

```
function dibujarGuiones40 () {  
  let i : number;  
  let linea : string = "";  
  for (i = 0; i <= 40; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
}
```

```
function dibujarGuiones30 () {  
  let i : number;  
  let linea : string = "";  
  for (i = 0; i <= 30; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
}
```

**Necesitamos un mecanismo que nos permita seleccionar la cantidad de guiones**



# Métodos

## Parámetros

- Son valores que enviamos a los métodos
- Se inicializa fuera del método
- Tienen un tipo
- Dentro del método se comporta como una variable
- Nos ayudan a evitar métodos duplicados

```
function dibujar30Guiones() {
  let x:number, linea:string = "";
  for (x=1; x<=30; x++) {
    linea += "-";
  }
  console.log(linea);
}
```

```
dibujar30Guiones();
dibujar40Guiones();
```

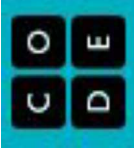
```
function dibujar40Guiones() {
  let x:number, linea:string = "";
  for (x=1; x<=40; x++) {
    linea += "-";
  }
  console.log(linea);
}
```

```
function dibujarGuiones(cantidad:number) {
  let x:number, linea:string = "";
  for (x=1; x<=cantidad; x++) {
    linea += "-";
  }
  console.log(linea);
}
```

```
dibujarGuiones(30);
dibujarGuiones(40);
dibujarGuiones(20);
```

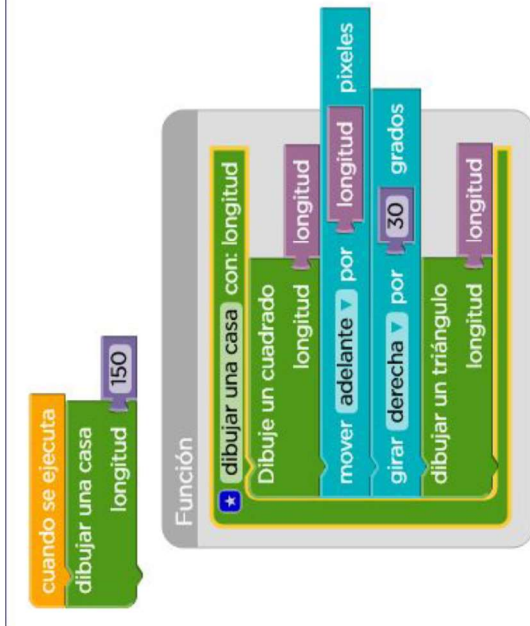
cantidad es un parámetro y nos permite indicar cuantos guiones queremos dibujar

# Comparativa code.org



En este ejemplo de programa de bloques de code.org, la función se llama “dibujarUnaCasa” y recibe un parámetro “longitud” que indica el tamaño de la casa que se debe dibujar.

```
function dibujarUnaCasa(longitud: number) {  
  //Dibuja un casa de tamaño longitud  
  
  dibujarUnCuadrado(longitud);  
  moverAdelante(longitud);  
  girarADerecha(90);  
  dibujarUnTriangulo(longitud);  
}  
  
dibujarUnaCasa(150);
```

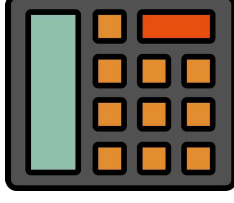


# Modularización

## *Implementar una Calculadora*

- Indique por parámetros la cantidad de guiones a dibujar en el método dibujarGuionesN

```
if (opcionMenu == 1) {  
  dibujarGuionesN (40);  
  console.log("el resultado es: ", numero1 + numero2);  
  dibujarGuionesN (40);  
} else if (opcionMenu == 2) {  
  dibujarGuionesN (30);  
  console.log("el resultado es: ", numero1 - numero2);  
  dibujarGuionesN (30);  
}  
  
function dibujarGuionesN (n : number) {  
  let i : number;  
  let linea : string = "";  
  for (i = 0; i <= n; i++) {  
    linea = linea + "-";  
  };  
  console.log(linea);  
}
```





# Métodos

## *Parámetros*

- ¿Puedo omitir un parámetro?
- ¿Cuántos parámetros puede tener un método?
- ¿Los parámetros pueden ser de diferentes tipos?
- ¿Es importante el orden de los parámetros?



# Modularización

## *Implementar una Calculadora*

```
let resultado = 0;
if (opcionMenu == 1) {
  resultado = numero1 + numero2;
} else if (opcionMenu == 2) {
  resultado = numero1 - numero2;
}
```

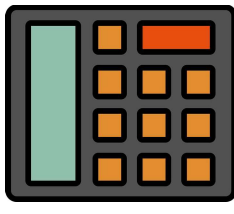
```
dibujarGuionesN (50);
```

```
console.log("el resultado es: ", resultado);
```

```
dibujarGuionesN (50);
```

```
function dibujarGuionesN (n : number) {
  let i : number;
  let linea : string = "";
  for (i = 0; i <= n; i++) {
    linea = linea + "-";
  };
  console.log(linea);
}
```

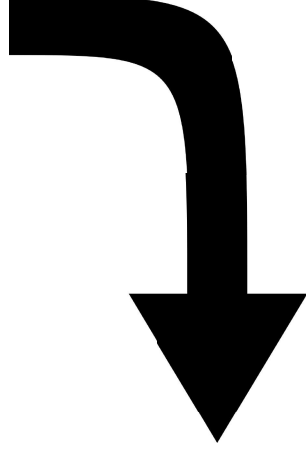
¿Y si quiero agrupar el cálculo del resultado en un método?



# Métodos

## *Retornos*

- Los métodos pueden retornar un valor al finalizar su ejecución
- El retorno es el “resultado” del método
- El retorno de un método puede ser de diferentes tipos:
  - Texto
  - Numérico
  - Lógico
  - Etc.



# Métodos con Parámetros

## Ejemplo

- El retorno nos permite devolver un valor al terminar de ejecutarse la función. Este valor puede ser cualquier tipo de dato de los muchos que tenemos en TypeScript.

```
console.log(dibujarGuiones(30));
```

```
function dibujarGuiones(cantidad:number) : string {  
  let x:number, linea:string = "";  
  for (x=1; x<=cantidad; x++) {  
    linea += "-";  
  }  
  return linea;  
}
```

el programa llamador, puede enviar el resultado a consola directamente

```
let guiones = dibujarGuiones(40);  
console.log(guiones);
```

la funcion ya no envia a consola los guiones sino que los retorna como texto al programa llamador.

o bien guardar el resultado en una variable y luego enviarla a consola

# Modularización

## *Implementar una Calculadora*

- Implemente un método llamado `calcularResultado` que reciba por parámetros los dos números y la opción y retorne el resultado de la operación

# Modularización

## *Implementar una Calculadora*

- Implemente un método llamado `calcularResultado` que reciba por parámetros los dos números y la opción y retorne el resultado de la operación

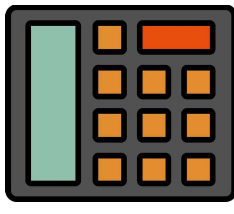
```
let resultado = calcularResultado (numero1, numero2, opcionMenu);
```

```
dibujarGuionesN (50);
```

```
console.log("el resultado es: ", resultado);
```

```
dibujarGuionesN (50);
```

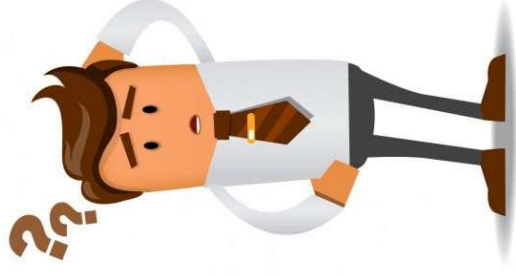
```
function calcularResultado (numero1:number, numero2:number,  
opcionMenu:number):number {  
  let resultado:number;  
  if (opcionMenu == 1) {  
    resultado = numero1 + numero2;  
  } else if (opcionMenu == 2) {  
    resultado = numero1 - numero2;  
  }  
  return resultado;  
}
```



# Métodos

## Más Preguntas

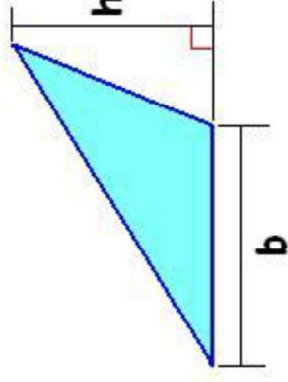
- ¿**return** es lo mismo que **console.log**?
- ¿Qué pasa si no guardo nada en el retorno?
- ¿El retorno va al final del método?
- ¿Se puede retornar más de un valor?



# Métodos

## *Ejercicio Triángulos*

- Realice un programa que devuelva el área del triángulo usando los siguientes pares de base-altura:
  - (1,2) (2,4) (3,6) (4,8) (5, 10) (6,12) (7,14)
- Implemente un método llamado `calcularAreaTriangulo` que reciba dos números por parámetro (uno llamado base y otro altura)



$$\text{Area Triángulo} = \frac{b \cdot h}{2}$$



# Métodos

## Ejercicio Triángulos

```
function calculandoTriangulos () {
  let resultado:number=calcularAreaTriangulo(1, 2);
  console.log("El area es ", resultado);
  resultado=calcularAreaTriangulo(2, 4);
  console.log("El area es ", resultado);
  resultado=calcularAreaTriangulo(3, 6);
  console.log("El area es ", resultado);
  resultado=calcularAreaTriangulo(4, 8);
  console.log("El area es ", resultado);
  resultado=calcularAreaTriangulo(5, 10);
  console.log("El area es ", resultado);
  resultado=calcularAreaTriangulo(6, 12);
  console.log("El area es ", resultado);
  resultado=calcularAreaTriangulo(7, 14);
  console.log("El area es ", resultado);
}
```

**function** calcularAreaTriangulo (base:number, altura:number):number {  
     **return** (base\*altura)/2;  
};

**¿Y si seguimos la serie numérica hasta 100?**

**Técnicas de Programación**

# **Carrera Programador full-stack**

***Modularización y Métodos (Ejercicios)***

# Métodos

## *Ejercicio: Potencias*

- Realice un programa que devuelva la potencia de un número.
- La base y el exponente deben ser ingresados por teclado.
- Sólo deben admitirse exponentes mayores o iguales a cero.
- Recuerde que el resultado de un numero elevado a 0 es 1.
- Separe la lógica de calcular la potencia utilizando métodos.

The diagram illustrates the components of the equation  $3^4 = 3 \times 3 \times 3 \times 3 = 81$ . It features three colored arrows pointing to specific parts of the equation: a blue arrow labeled 'Base' points to the number 3; an orange arrow labeled 'Exponente' points to the superscript 4; and a purple arrow labeled 'Resultado' points to the final value 81.

Exponente

Base

Resultado

$$3^4 = 3 \times 3 \times 3 \times 3 = 81$$

# Métodos

## *Ejercicio: Múltiplos*

- Cree un método `esMultiplo` con 2 parámetros que devuelva el valor lógico **verdadero** o **falso** según si el primer número que se indique como parámetro es múltiplo del segundo
- Recuerde que un número es múltiplo de otro si al dividirlo su resto es cero
- Recuerde que la operación **mod** permite saber si el resto de una división es cero

$$\begin{array}{r} \text{dividendo} \quad 40 \quad | \quad \text{divisor} \quad 8 \\ \hline \text{resto} \quad 0 \quad \text{cociente} \quad 5 \end{array}$$

# Métodos

## *Ejercicio: Divisores*

- Implemente un método llamado `cantidadDeDivisores` que reciba un número entero y devuelva la cantidad de divisores
- Por ejemplo, para el número 16, sus divisores son 1, 2, 4, 8, 16, por lo que la respuesta debería ser 5
- **Re-utilice** el método `esMultiplo` implementado para el ejercicio anterior

