

# Técnicas de Programación

## CFP Programador full-stack

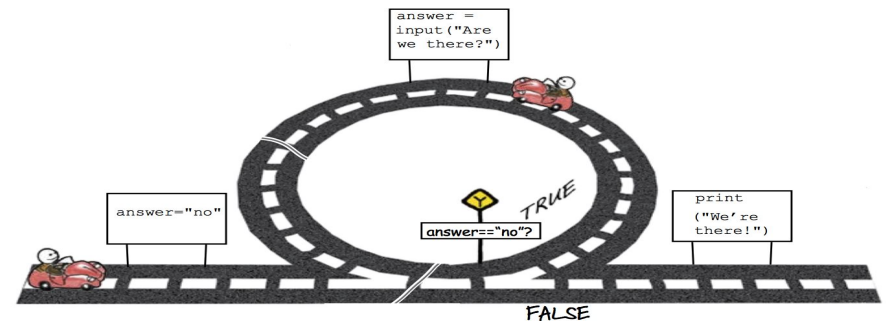
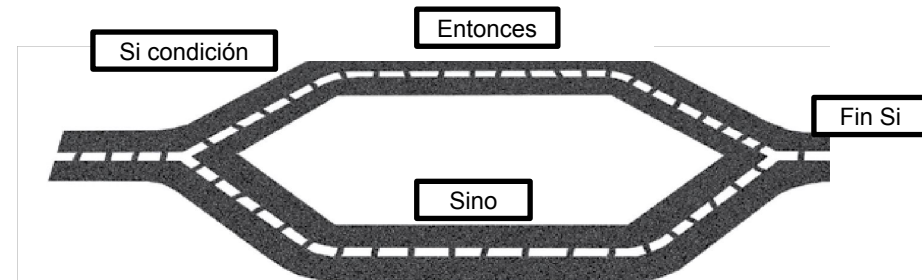
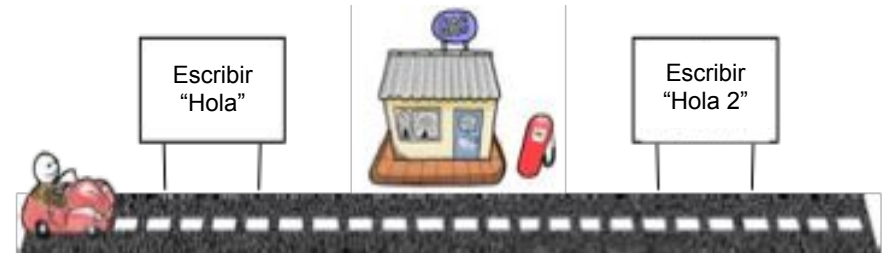
*Repetición (Repaso)*

# 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
- Son útiles cuando hay que hacer un conteo (fijo)
- El valor inicial del conteo, el valor final de corte y los 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

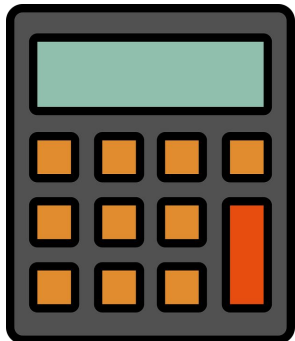
## CFP Programador full-stack

*Modularización y Métodos (Conceptos)*

# Modularización

## *Implementar una Calculadora*

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



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

# Modularización

## *Implementar una Calculadora*

- Definimos las variables y leemos desde teclado

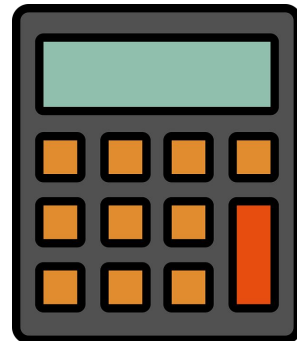
```
let readlineSync = require('readline-sync');
```

```
let i, linea;
```

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

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

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





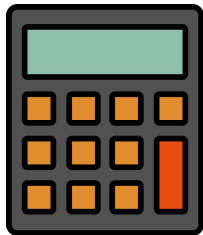
# Modularización

## *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);
```

```
}
```

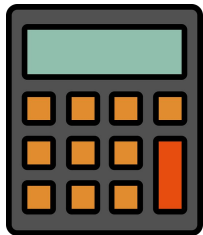
```
else if (opcionMenu == 2) {
```

```
  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

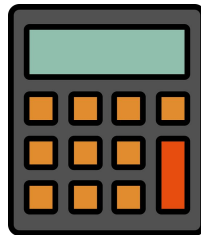
- **No debemos** duplicar el código
- Cuando tenemos la **misma funcionalidad** en distintas partes del programa debemos **reusar**
- ¿Cuál es la diferencia entre copiar y reusar?
  - Una sola copia que puede ser llamada donde lo necesitemos
  - Programas mas cortos
  - Cambios acotados a un solo lugar



# Código Repetido

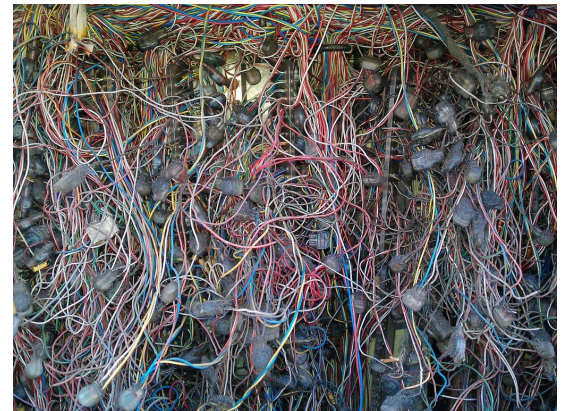
- Cambio acotado en un solo lugar (una opción)

```
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);  
}
```



# Métodos

- Cuando los programas crecen se vuelven mas complejos:
  - Necesitamos manejar la complejidad
- Debemos **agrupar** las sentencias que tienen **cohesión**
  - Mayor legibilidad
  - Mayor mantenibilidad



# Métodos

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

# Métodos

## Sintaxis

- Las Funciones se pueden declarar de 3 maneras

*/\* como sentencia \*/*

```
function otraFuncion () {  
  console.log("Hola!");  
}  
otraFuncion();
```

*/\* como valor de una variable \*/*

```
let miFuncion = function () {  
  console.log("Hola!");  
}  
miFuncion();
```

*/\* como arrow function \*/*

```
let arrowFunction = () => {  
  console.log("Hola!");  
}  
arrowFunction();
```

*/\* como método de un objeto \*/*

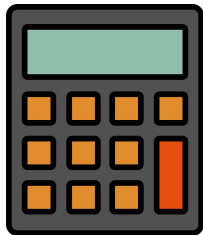

```
let miObjeto = {  
  propiedad: "Soy una propiedad";  
  metodo: function () {  
    console.log("Hola!");  
  };  
}  
miObjeto.metodo();
```

Esta opción la veremos más adelante

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

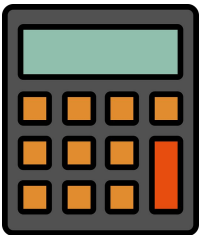




# Modularización

## *Implementar una Calculadora*

- Implemente un método llamado `dibujarGuiones` para evitar el código repetido



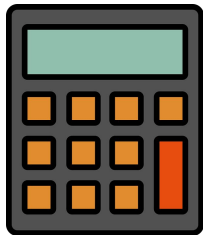
# Modularización

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



# Modularización

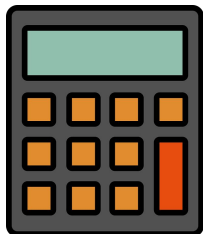
## *Implementar una Calculadora*

- 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;  
    let linea = "-";  
    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?



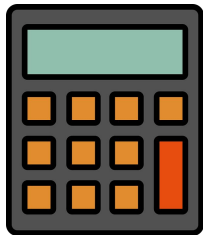
# Modularización

## *Implementar una Calculadora*

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

```
function dibujarGuiones30 () {  
  let i;  
  let linea = "-";  
  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

```
let readlineSync = require('readline-sync');
```

```
felicitarGente("Ana");  
felicitarGente("Pablo");  
felicitarGente("María");
```

```
let nombreIngresado = readlineSync.question("Ingrese a quien felicitar: ");
```

```
felicitarGente(nombreIngresado);
```

```
function felicitarGente( nombre ) {  
  console.log("Felicitaciones ", nombre);  
};
```

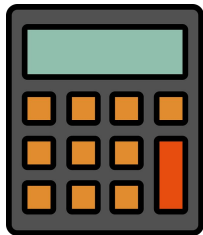
# Modularización

## *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) {  
    let i;  
    let linea = "-";  
    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

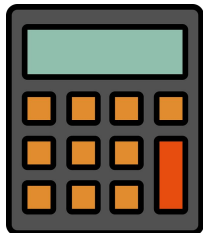
## *Implementar una Calculadora*

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

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

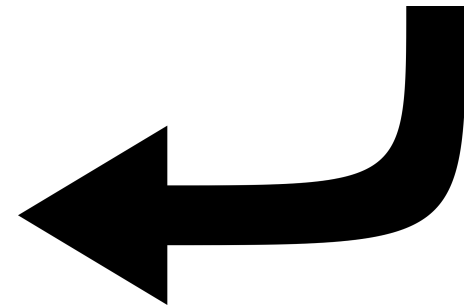




# Métodos

## *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 JavaScript.

```
function leerNombreDesdeTeclado (textoAMostrar) {  
  let readlineSync = require('readline-sync');  
  return readlineSync.question(textoAMostrar);  
}
```

```
let nombre = leerNombreDesdeTeclado ("Ingrese Nombre");  
let apellido = leerNombreDesdeTeclado ("Ingrese Apellido");  
console.log("Su nombre completo es ", nombre, " ", apellido);
```

# 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

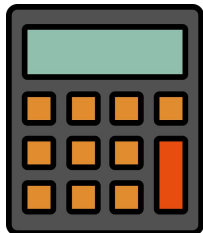
# Modularización

## *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, numero2, opcionMenu) {  
  let resultado;  
  if (opcionMenu == 1) {  
    resultado = numero1 + numero2;  
  } else if (opcionMenu == 2) {  
    resultado = numero1 - numero2;  
  }  
  return resultado;  
}
```



# Métodos

## *Más Preguntas*

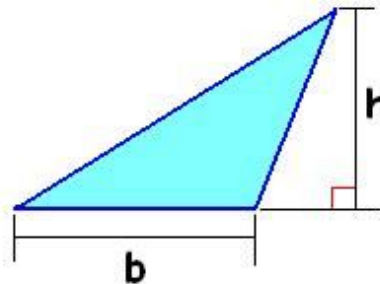
- ¿Retorno es lo mismo que Escribir?
- ¿Por qué no copiar y pegar? Es más fácil...
- ¿Qué pasa si no guardo nada en el retorno?
- ¿El retorno va al final del método?
- ¿Se puede retornar más de un valor?



# 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=calcularAreaTriangulo(1 , 2);  
  console.log("El area es = ", resultado);  
  resultado=calcularAreaTriangulo( 2, 4);  
  console.log("El area es = ", resultado);  
  resultado=calcularAreaTriangulo( 2, 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, altura) {  
  return (base*altura)/2;  
};
```

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

# Métodos

## *Ejercicio Triángulos*

```
let i;  
for (i = 1; i <= 100; i++) {  
    console.log("El area es = ", calcularAreaTriangulo (i, i*2));  
}
```

```
function calcularAreaTriangulo (base, altura) {  
    return (base*altura)/2;  
};
```



# Técnicas de Programación

## CFP 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 calculation of a power. It features the equation  $3^4 = 3 \times 3 \times 3 \times 3 = 81$ . Three colored arrows point to specific parts of the equation: a blue arrow points to the base '3', an orange arrow points to the exponent '4', and a purple arrow points to the result '81'. The labels 'Base', 'Exponente', and 'Resultado' are placed near their respective arrows.

Base

Exponente

Resultado

# 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 numero 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

dividendo		divisor
40		8
resto - 0		5
		cociente

# 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
- **Reutilice** el método `esMultiplo` implementado para el ejercicio anterior

