

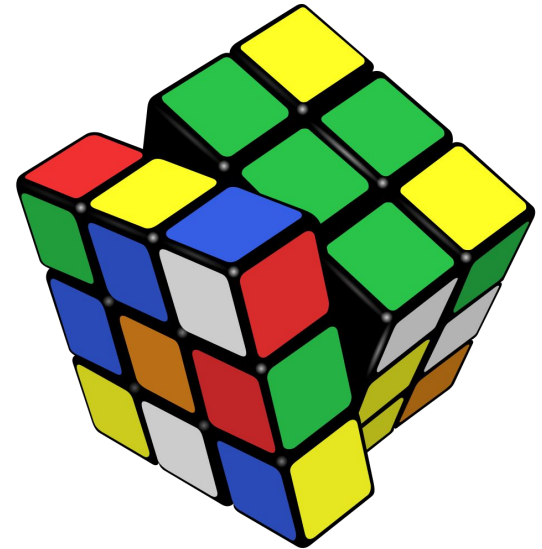
Técnicas de Programación

Programador full-stack

Algoritmos Secuenciales

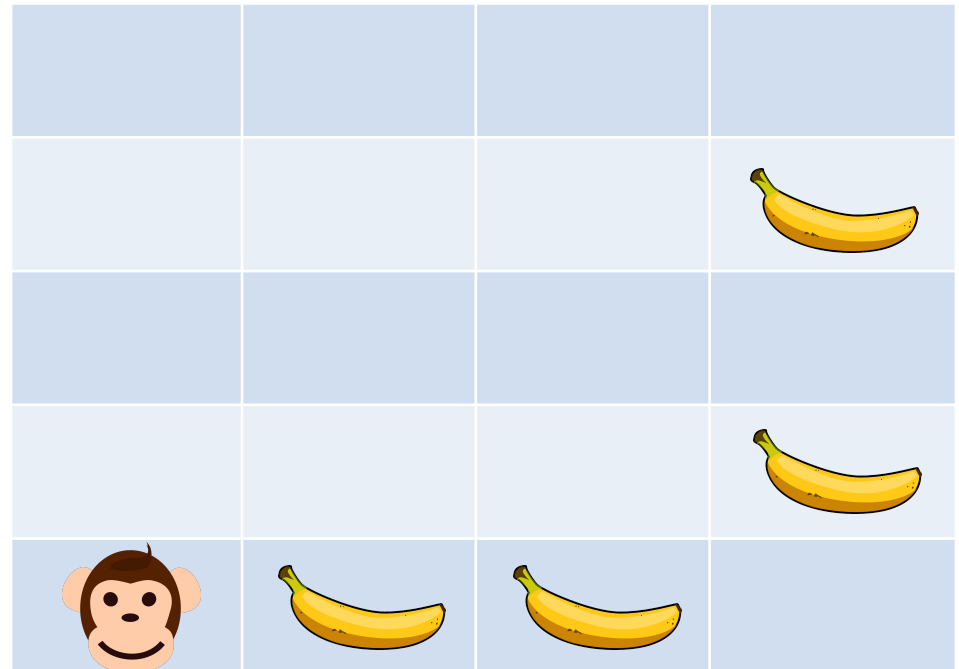
Algoritmos

- Nos ayudan a resolver un problema
- Consisten de pasos lógicamente ordenados
- Dado un conjunto de datos de entrada da un resultado (solución al problema)



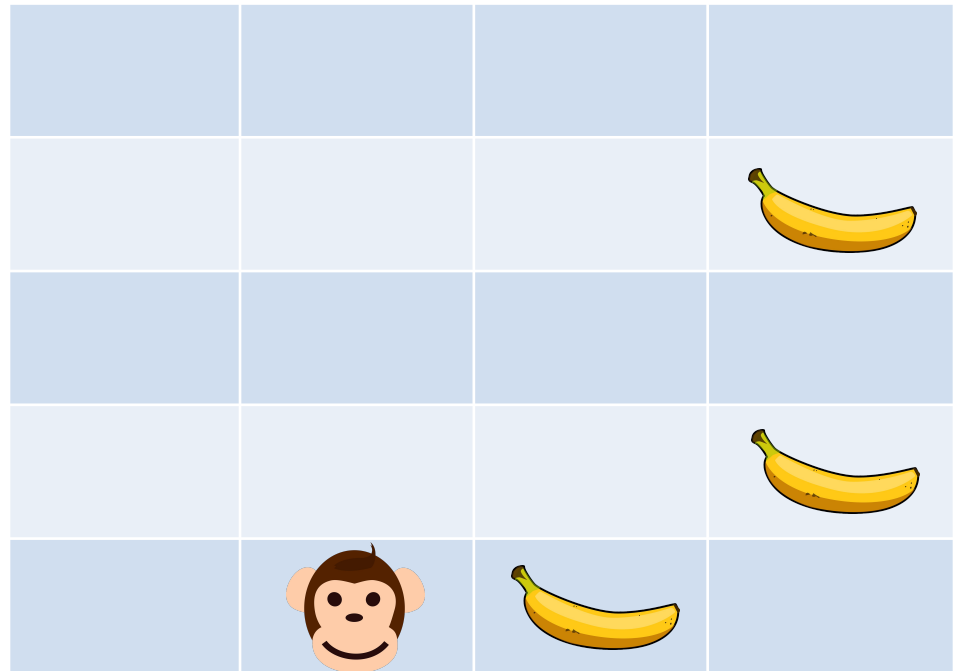
Algoritmos

- Instrucciones
 - Mover arriba
 - Mover abajo
 - Mover derecha
 - Mover izquierda
 - Comer banana



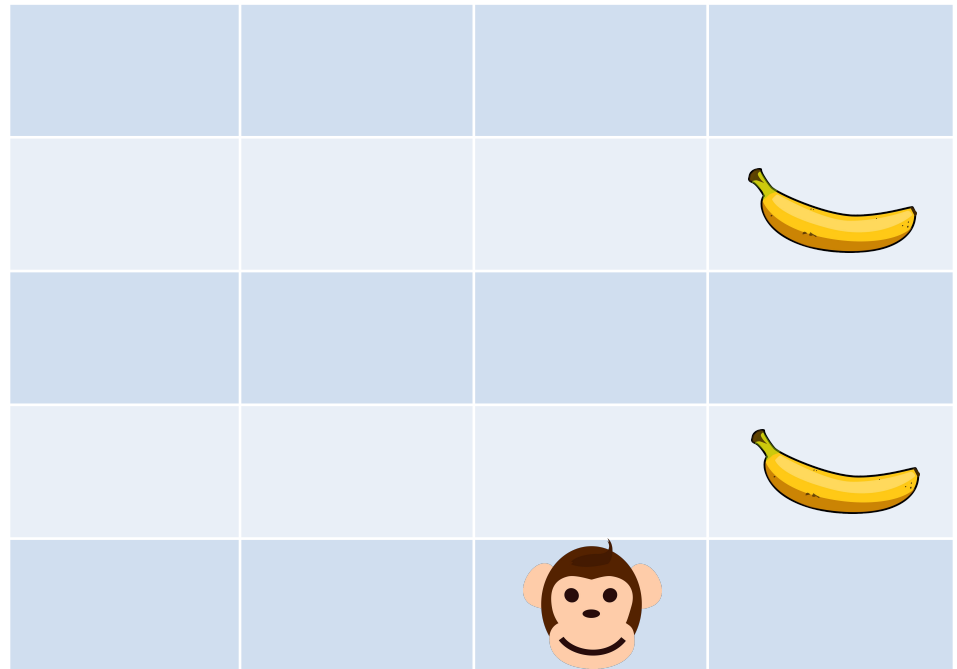
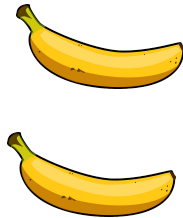
Algoritmos

- Algoritmo
 - Mover derecha
 - Comer banana



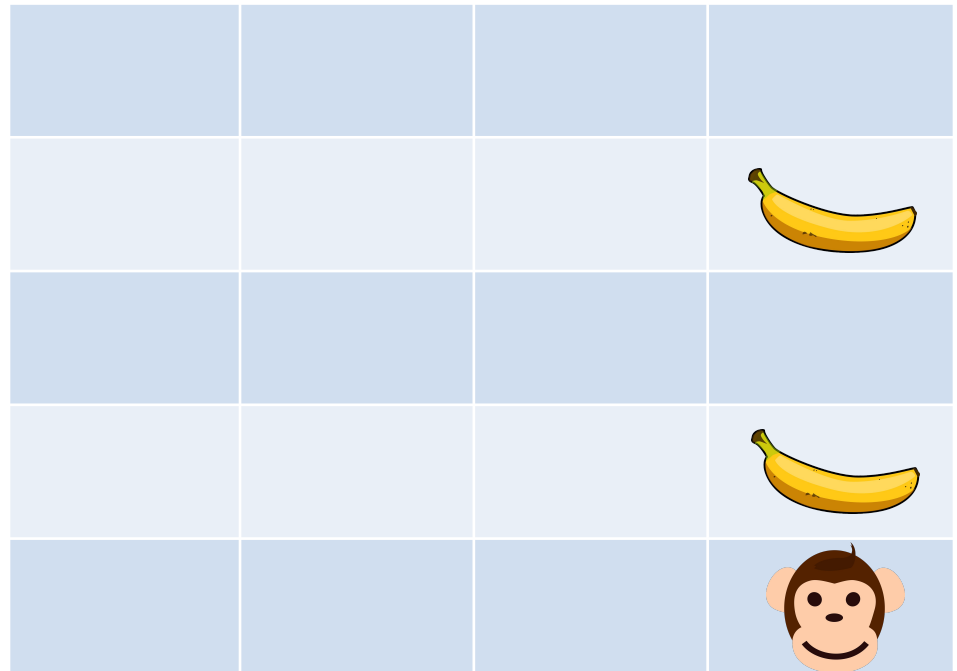
Algoritmos

- Algoritmo
 - Mover derecha
 - Comer banana
 - Mover derecha
 - Comer banana



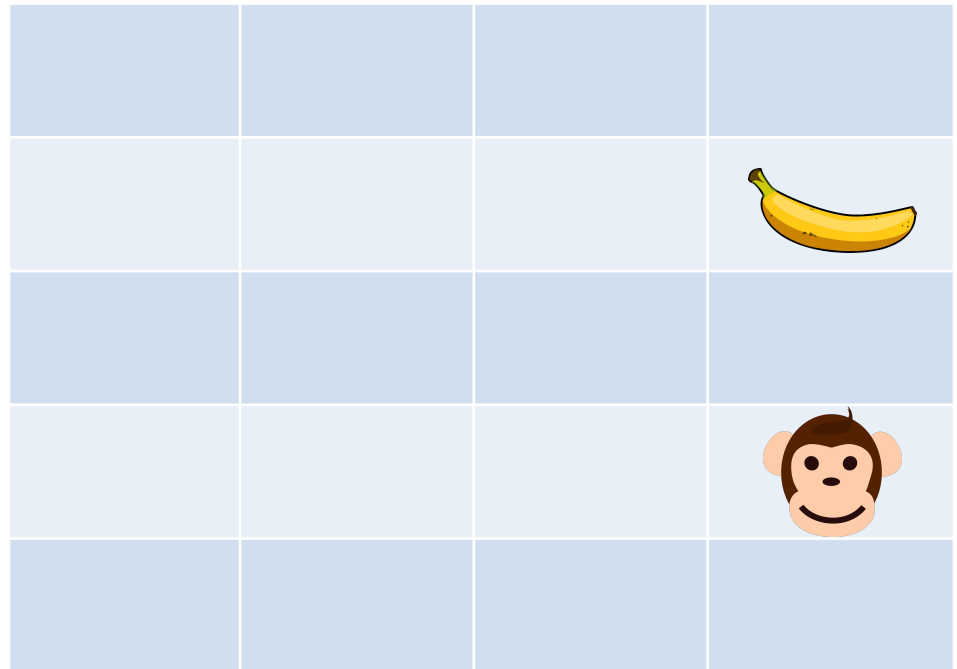
Algoritmos

- Algoritmo
 - Mover derecha
 - Comer banana
 - Mover derecha
 - Comer banana
 - Mover derecha



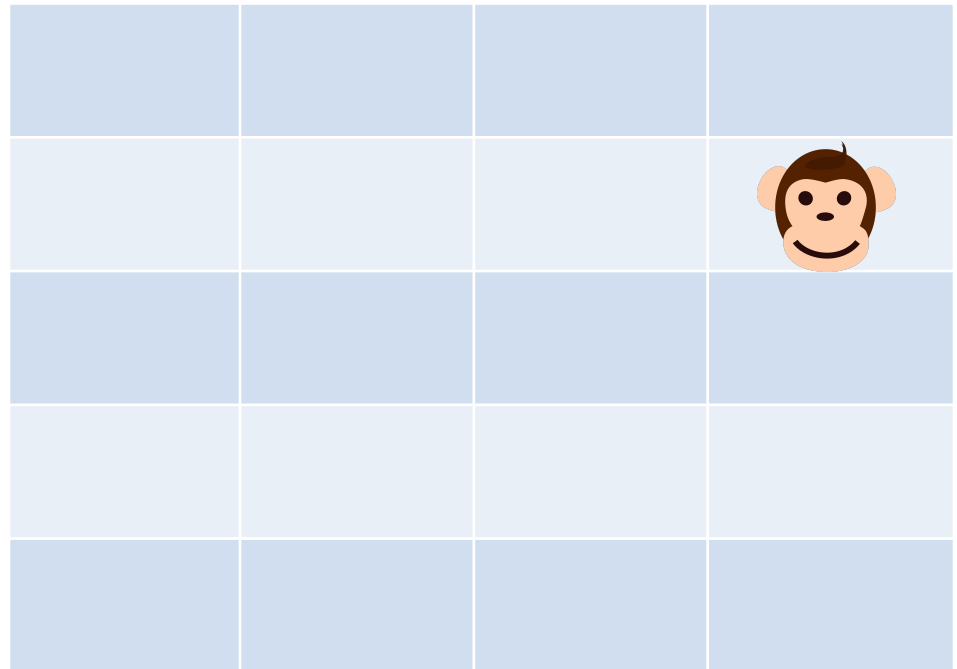
Algoritmos

- Algoritmo
 - Mover derecha
 - Comer banana
 - Mover derecha
 - Comer banana
 - Mover derecha
 - Mover arriba
 - Comer banana



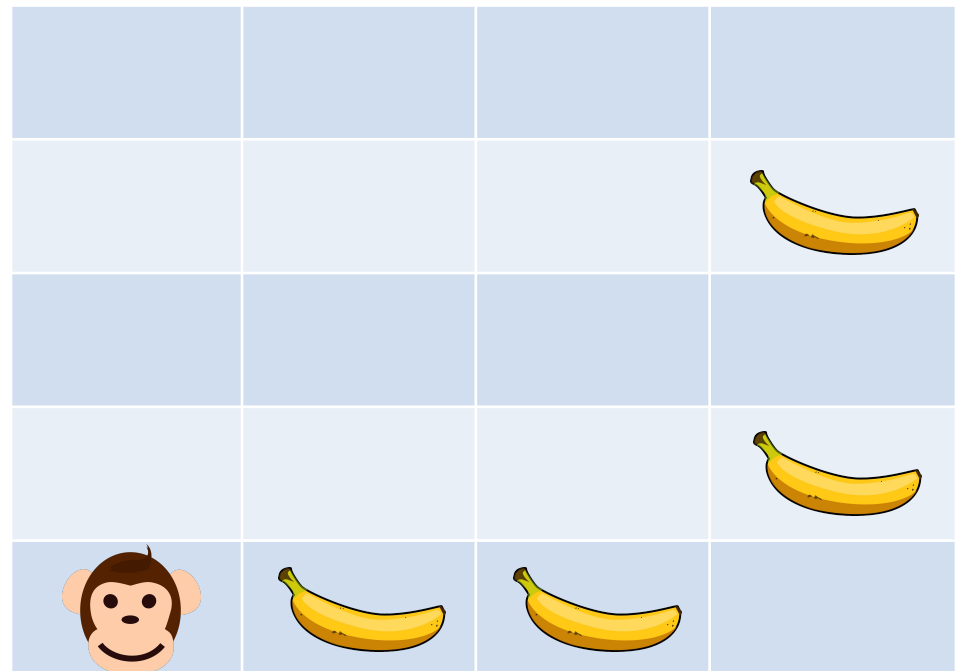
Algoritmos

- Algoritmo
 - Mover derecha
 - Comer banana
 - Mover derecha
 - Comer banana
 - Mover derecha
 - Mover arriba
 - Comer banana
 - Mover arriba
 - Mover arriba
 - Comer banana



Algoritmos

- Algoritmo
 - Mover derecha
 - Comer banana
 - Mover derecha
 - Comer banana
 - Mover derecha
 - Mover arriba
 - Comer banana
 - Mover arriba
 - Mover arriba
 - Comer banana



Algoritmos

- Debe ser **preciso**
- Debe estar **específicamente definido**
- Debe ser **finito**
- Debe ser **correcto**
- Debe ser **independiente** del lenguaje



Elementos de un Algoritmo

- Cálculo del área de un rectángulo

- **Entrada:**

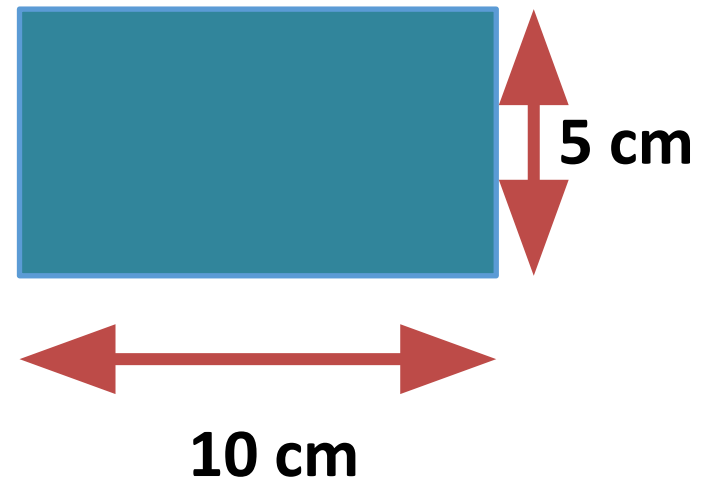
- Dato 1: altura 5 cm
- Dato 2: base 10 cm

- **Proceso:**

- $\text{Área} = \text{base} * \text{altura}$

- **Salida:**

- $5 * 10 = 50$



Prog. Orientada a Objetos

Programador full-stack

TypeScript

TypeScript - Glosario básico

- **Script:** cada uno de los programas, aplicaciones o trozos de código creados con el lenguaje de programación TypeScript. Unas pocas líneas de código forman un script y un archivo de miles de líneas de TypeScript también se considera un script.
- **Sentencia:** cada una de las instrucciones que forman un script.
- **Palabras reservadas:** son las palabras (en inglés) que se utilizan para construir las sentencias de TypeScript y que por tanto no pueden ser utilizadas libremente. Las palabras actualmente reservadas por TypeScript son: **any, as, boolean, break, case, catch, class, const, constructor, continue, debugger, declare, default, delete, do, else, enum, export, extends, false, finally, for, from, function, get, if, implements, import, in, instanceof, interface, let, module, new, null, number, of, package, private, protected, public, require, return, set, static, string, super, switch, symbol, this, throw, true, try, type, typeof, var, void, while, with, yield.**

Características de TypeScript

- Tipos de datos
 - Característica más importante (de ahí el nombre)
 - Código más legible/entendible
 - Más chequeos al momento de desarrollar → mayor seguridad
- Soporte de clases
 - Programación orientada a objetos (en el próximo cuatrimestre)

TypeScript - Sintaxis básica

Las normas básicas que definen la sintaxis de TypeScript son las siguientes:

- **No se tienen en cuenta los espacios en blanco y las nuevas líneas**
- **Se distinguen las mayúsculas y minúsculas:** La palabra “console” no es lo mismo que “CONSOLE” ni que “ConSolE”
- **No es necesario terminar cada sentencia con el carácter de punto y coma (;).** Aunque es conveniente hacerlo para usar todos el mismo estilo de código.

Técnicas de Programación

Programador full-stack

Algoritmos simples en TypeScript (TS)

Implementando Algoritmos

Recuerden que vamos a tener una carpeta por modulo y dentro ademas una carpeta por clase. Entonces dentro de nuestra **PFS2022** vamos a crear la carpeta **1-PB**

```
C:\Cursos\CFL>mkdir 1-PB
```

Y dentro de **1-PB** creamos la carpeta **Clase_1**

```
C:\Cursos\CFL>cd 1-PB
```

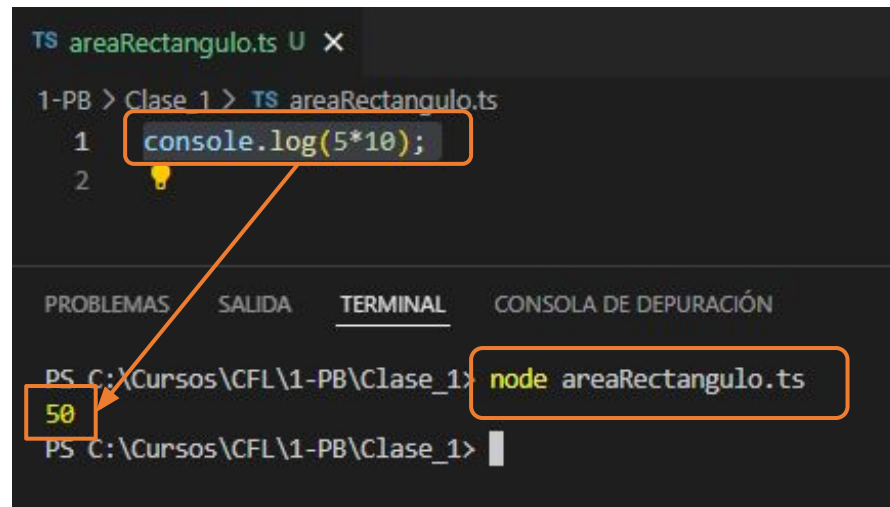
```
C:\Cursos\CFL\1-PB>mkdir Clase_1
```

Implementando Algoritmos

Ejercicio: Área del Rectángulo 5x10

Creamos “areaRectangulo.ts”, agregando esta sentencia:

```
console.log(5*10);
```



```
TS areaRectangulo.ts U X
1-PB > Clase_1 > TS areaRectangulo.ts
1 console.log(5*10);
2
PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN
PS C:\Cursos\CFL\1-PB\Clase_1> node areaRectangulo.ts
50
PS C:\Cursos\CFL\1-PB\Clase_1>
```

Implementando Algoritmos

Ejercicio: Área del Rectángulo 5x10



Una letra mal escrita puede hacer que la computadora no entienda el programa!

Debo aprender su idioma e incluso ser cuidadoso de escribirlo bien

Estructuras de Control

Secuenciales

Selectivas o De
Decisión

Repetitivas

Secuencia

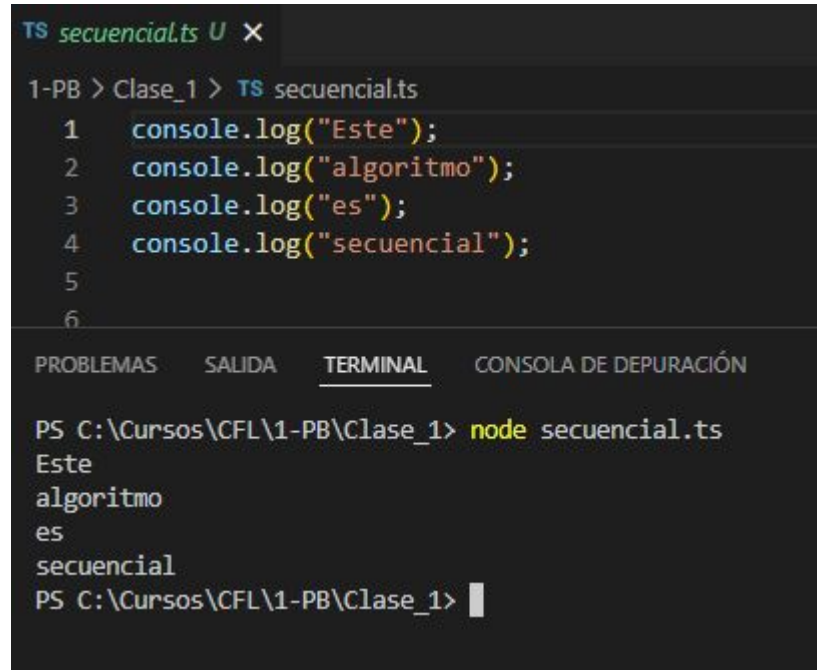


- Un algoritmo es una serie de pasos para resolver un programa
- Los algoritmos más simples son una lista de acciones que se ejecutan en orden

Secuencia

Ejemplo instrucción console.log()

```
console.log ("Este");  
console.log ("algoritmo");  
console.log ("es");  
console.log ("secuencial");
```



The screenshot shows a Visual Studio Code editor window with a file named 'secuencial.ts'. The code in the file is:

```
1 console.log("Este");  
2 console.log("algoritmo");  
3 console.log("es");  
4 console.log("secuencial");  
5  
6
```

Below the editor, the 'TERMINAL' tab is active, showing the command 'node secuencial.ts' being executed in a PowerShell prompt. The output of the command is:

```
PS C:\Cursos\CFL\1-PB\Clase_1> node secuencial.ts  
Este  
algoritmo  
es  
secuencial  
PS C:\Cursos\CFL\1-PB\Clase_1>
```

Comparativa code.org



Es lo que hacemos en code.org al poner un bloque debajo de otro

//se ejecutan cuando refrescamos la página

```
console.log ( "Paso 1" ) ;
```

```
console.log ( "Paso 2" ) ;
```



Comentarios

En TS **se pueden incluir comentarios**

Los comentarios se utilizan para añadir información en el código fuente del programa, son aclaraciones para otros programadores (o vos mismo)

```
/* este programa imprime 4 líneas por la consola,  
Además tiene comentarios */  
console.log ( "Este" ) ;  
//esto es un comentario  
console.log ( "algoritmo" ) ;  
//esto es otro comentario  
console.log ( "es" ) ;  
//esto es un cuarto comentario  
console.log ( "secuencial" ) ;
```


Secuencia

Ejercicios

Crear un nuevo script ***“laGranjera.ts”*** e imprimir por consola los pasos (bloques) para completar las tareas de la granjera.

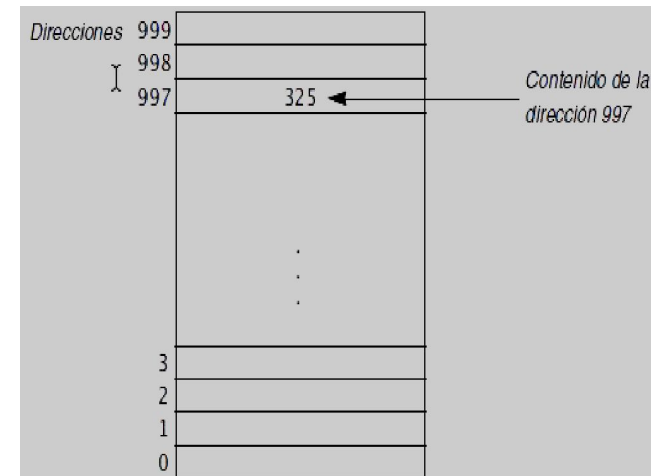


Variables

- Las variables en programación siguen una lógica similar a las variables usadas en otros ámbitos como las matemáticas.
- Una variable es un elemento que se emplea para almacenar y hacer referencia a un valor.
- Gracias a las variables es posible crear "programas genéricos", es decir, programas que funcionan siempre igual independientemente de los valores concretos utilizados.

Variables

- Guarda información (números, letras, etc.)
- Tiene una dirección de memoria
- Tiene un nombre
- Tiene un **tipo**
- Su contenido puede **variar** durante la ejecución del programa



Tipos básicos en TypeScript

Tipo	Significado
number	Cualquier tipo de número
boolean	Verdadero/falso
string	Texto
null	Cuando un elemento no tiene valores
undefined	Cuando una variable no está inicializada
any	Cualquier tipo
void	Cuando las funciones no retornan nada

Variables

Ejemplo:

```
resultado = 3 + 1
```

En TypeScript:

```
let numero_1 : number = 3;
```

```
let numero_2 : number = 1;
```


```
let resultado : number = numero_1 + numero_2;
```

Declarar variable: La palabra reservada **let** y el tipo solamente deben ser indicados al definir por primera vez la variable. Se recomienda declarar todas las variables que se vayan a utilizar.

Si cuando se declara una variable se le asigna un valor, se dice que la variable ha sido **inicializada**. Se pueden declarar por una parte y asignarles un valor posteriormente.

Restricción de Tipos en Variables

```
let numero: number;  
numero = "8";
```



Nos va a dar error porque los tipos no son compatibles

Recomendaciones Generales

- Usar “camelCase” para definir variables
- Usar nombres descriptivos para las variables
- Tener en cuenta que el código que hagamos lo van a leer otras personas, o nosotros mismos dentro de varios meses o incluso años
 - Lo mejor es que el código sea fácil de leer
 - En caso de que igualmente sea complicado, usar comentarios para facilitar la lectura

Comparativa code.org



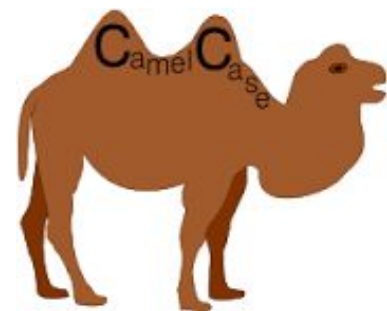
En este ejemplo de programa de bloques de code.org, la variable es **longitud** y se utiliza, como recordarán, para dibujar casas de distintos tamaño.



Variables

Buenas Prácticas

- Los nombres de las variables deben ser representativos
 - La falta de buenos nombres hace a nuestro programa muy difícil de entender y leer por nosotros y otros desarrolladores
- El nombre de una variable siempre comienza con una letra minúscula
- Si son varias palabras, se escribe en mayúsculas la primera letra de cada palabra (excepto la primera palabra) esta manera de nombrar se denomina **Camel Case**
 - Ejemplos: primerNumero, resultadoDeLaSuma



Variables vs. Constantes

Declarar una variable

```
let nombre: string;  
nombre = "Pepe";  
console.log(nombre);  
nombre = "Maria";  
console.log(nombre);
```

Declarar una constante

```
const nombre = "Pepe";  
console.log(nombre);  
nombre = "Maria"; //ERROR
```

Ejercicio

Modificar el ejemplo de secuencia:

- Qué cada mensaje se almacene en una variable a mostrar por consola y que el funcionamiento del script sea el mismo

Modificar el ejemplo de base por altura

- Almacenar la base y la altura en variables y el resultado y que el funcionamiento del script sea el mismo

Ingreso de datos

¿Cómo le decimos a un programa que datos de entrada deseamos que utilice?

Recordamos:

NodeJS

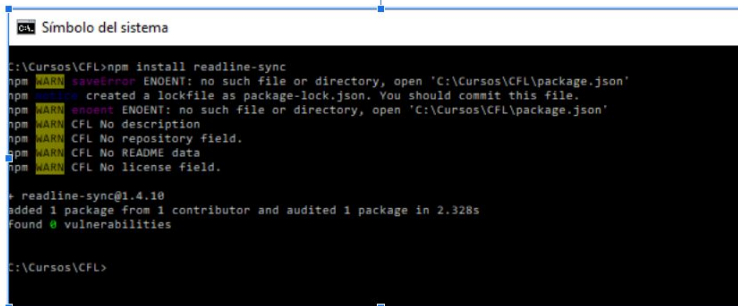
Instalación de paquete ***“readline-sync”*** usando el comando ***“npm”***

En el Command Prompt ejecutar:

- **npm install readline-sync**

Este paquete **“readline-sync”** permite ejecutar de forma interactiva una conversación con el usuario a través de una consola

De esta manera se puede ingresar datos a nuestros scripts



```
Símbolo del sistema

C:\Cursos\CFL>npm install readline-sync
npm WARN saveError ENOENT: no such file or directory, open 'C:\Cursos\CFL\package.json'
npm WARN createLockfileError ENOENT: no such file or directory, open 'C:\Cursos\CFL\package.json'
npm WARN notsup NO SUPPORTED: no such file or directory, open 'C:\Cursos\CFL\package.json'
npm WARN CFL No description
npm WARN CFL No repository field.
npm WARN CFL No README data
npm WARN CFL No license field.

+ readline-sync@1.4.10
added 1 package from 1 contributor and audited 1 package in 2.328s
found 0 vulnerabilities

C:\Cursos\CFL>
```

Ingreso de datos

¿Cómo le decimos a un programa que datos de entrada deseamos que utilice?

Creamos “alturaPersona.ts”, agregando estas sentencias:

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

En esta línea asociamos el nombre readlineSync al lector

```
let alturaPersona = readlineSync.question();
```

En esta línea leemos el valor que nos ingresa el usuario y lo almacenamos en la variable **alturaPersona**

```
console.log(alturaPersona);
```

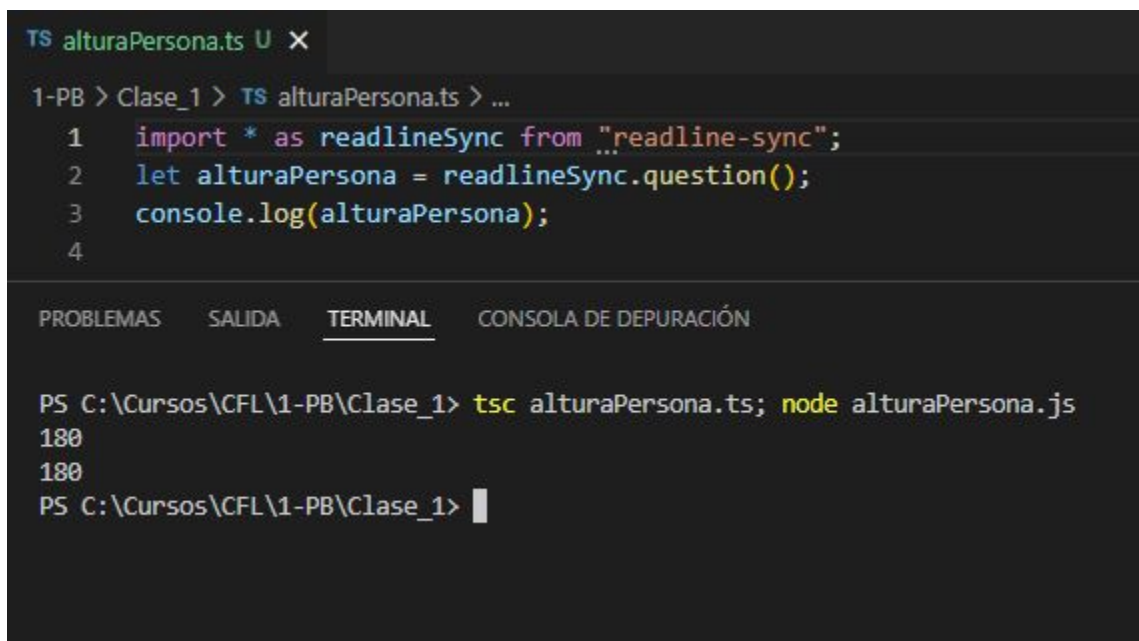
En esta línea enviamos el valor a la consola

Ingreso de datos

¿Cómo le decimos a un programa que datos de entrada deseamos que utilice?

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

```
let alturaPersona = readlineSync.question();  
console.log(alturaPersona);
```



The screenshot shows a code editor with a file named `alturaPersona.ts`. The code inside is:

```
1 import * as readlineSync from 'readline-sync';  
2 let alturaPersona = readlineSync.question();  
3 console.log(alturaPersona);  
4
```

Below the code editor is a terminal window with tabs for `PROBLEMAS`, `SALIDA`, `TERMINAL` (selected), and `CONSOLA DE DEPURACIÓN`. The terminal shows the command `tsc alturaPersona.ts; node alturaPersona.js` being executed, followed by two lines of output, both `180`, and a prompt `PS C:\Cursos\CFL\1-PB\Clase_1>`.

Ingreso de datos

Y ahora, ¿Cómo le decimos a un usuario que datos de entrada esperamos que ingrese?

Modificamos “alturaPersona.ts”, para que queden estas sentencias:

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

Esta línea no cambia.

```
let alturaPersona = readlineSync.question("Indique la altura de la persona: ");
```

En esta línea agregamos un texto para que el usuario sepa que esperamos de él.

```
console.log("La altura de la persona es: ", alturaPersona);
```

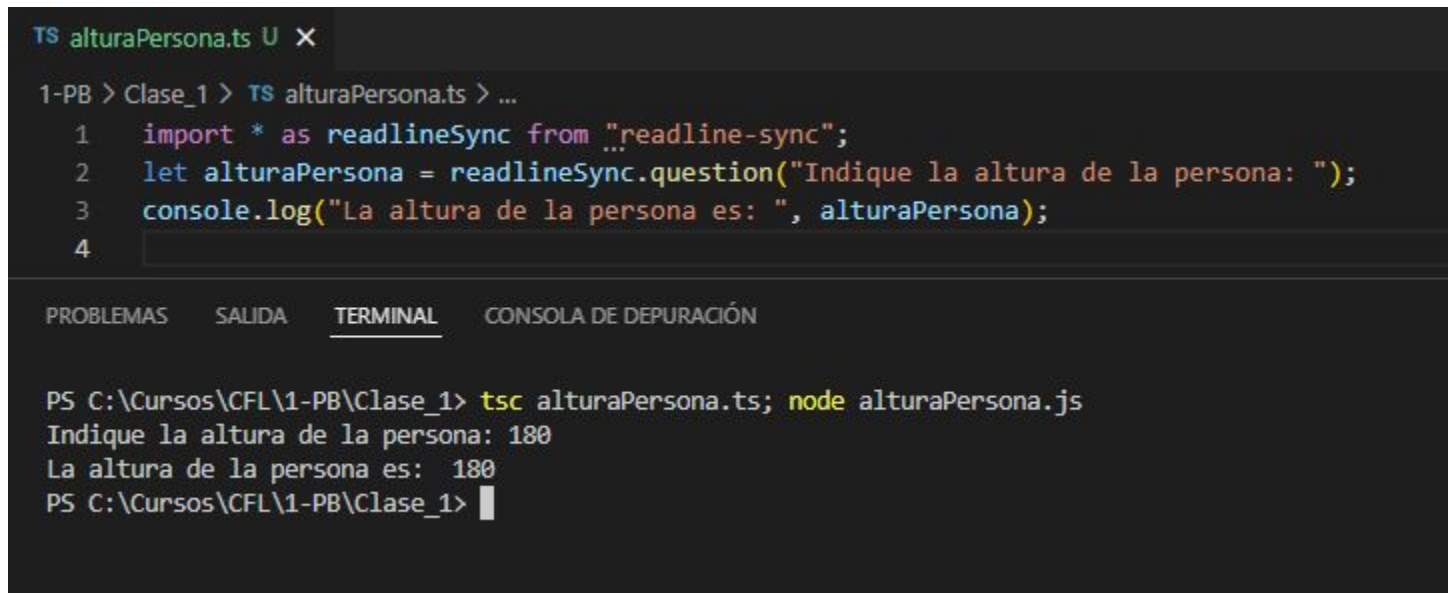
En esta línea agregamos un texto explicativo

Ingreso de datos

Y ahora, ¿Cómo le decimos a un usuario que datos de entrada esperamos que ingrese?

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

```
let alturaPersona = readlineSync.question("Indique la altura de la persona: ");  
console.log("La altura de la persona es: ", alturaPersona);
```



```
TS alturaPersona.ts U X  
1-PB > Clase_1 > TS alturaPersona.ts > ...  
1  import * as readlineSync from 'readline-sync';  
2  let alturaPersona = readlineSync.question("Indique la altura de la persona: ");  
3  console.log("La altura de la persona es: ", alturaPersona);  
4  
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN  
  
PS C:\Cursos\CFL\1-PB\Clase_1> tsc alturaPersona.ts; node alturaPersona.js  
Indique la altura de la persona: 180  
La altura de la persona es:  180  
PS C:\Cursos\CFL\1-PB\Clase_1> 
```


Tipado de Variables - Tipos

- El **tipado estático** nos obliga a definir desde el principio el tipo de una variable. Lenguajes con tipado estático son C++, Java, C#, TypeScript, etc.
- El **tipado dinámico** nos da la facilidad de no definir los tipos al declarar una variable, algunos ejemplos son PHP, JavaScript, Groovy, Python, entre otros.

Variables

Tipos de Datos Básicos

- **Numérico (number):**

- Números tanto enteros (integer) como decimales (float)
- Para separar decimales se utiliza el punto
- Ejemplos: 12, 0, -2.3, 3.14

```
let iva : number = 16;    // variable tipo entero  
let total : number = 234.65; // variable tipo decimal
```



Variables

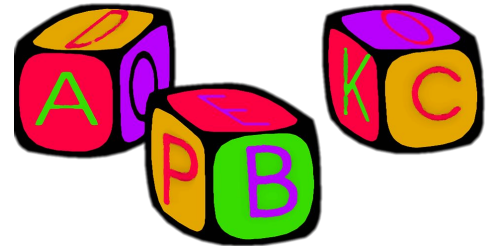
Tipos de Datos Básicos

- **Lógico (boolean):**

- Sólo puede tomar dos valores: **true** o **false**
- No se puede utilizar para almacenar números y tampoco permite guardar cadenas de texto.

```
let clienteRegistrado : boolean = false;
```

```
let ivaIncluido : boolean = true;
```



Variables

Tipos de Datos Básicos

- **Texto (string):**

- Caracteres o cadenas de caracteres encerrados entre comillas (dobles o simples)

```
let mensaje : string = "Bienvenido a nuestro sitio web";
```

```
let nombreProducto : string = 'Producto ABC';
```

```
let letraSeleccionada : string = 'c';
```

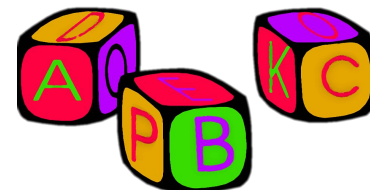
- Si el propio texto contiene comillas simples o dobles, la estrategia que se sigue es la de encerrar el texto con las comillas (simples o dobles) que no utilice el texto

// El contenido de texto1 tiene comillas simples, por lo qué se encierra con comillas dobles

```
let texto1 : string = "Una frase con 'comillas simples' dentro";
```

// El contenido de texto2 tiene comillas dobles, por lo qué se encierra con comillas simples

```
let texto2 : string = 'Una frase con "comillas dobles" dentro';
```



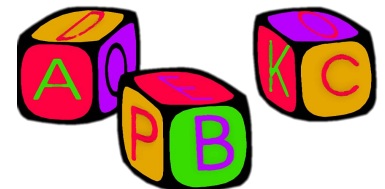
Variables

Tipos de Datos Básicos

- **Texto (string):**

- A veces las cadenas de texto contienen tanto comillas simples como dobles. Además existen otros caracteres (tabulador, ENTER, etc.) especiales. TypeScript permite caracteres especiales dentro de una cadena de texto con el "mecanismo de escape" de los caracteres problemáticos.

```
let texto1 : string = 'Una frase con \'comillas simples\' dentro';  
let texto2 : string = "Una frase con \"comillas dobles\" dentro";  
let texto3 : string = "Una frase con \n Una nueva línea dentro";  
let texto4 : string = "Una frase con \t Un tabulador dentro";  
let texto5 : string = "Una frase con \\ Una barra inclinada dentro";
```



Variables

Tipos de Datos Básicos

- **NULO (null):**

- El valor null es un literal de Typescript que representa un valor nulo o "vacío".

- **Indefinido (undefined):**

- Una variable a la cual no se le haya asignado valor tiene entonces el valor undefined.

- **Determinación del tipo usando el operador typeof**

```
console.log(typeof 42);
```

```
// salida esperada: "number"
```

```
console.log(typeof 'blubber');
```

```
// salida esperada: "string"
```

```
console.log(typeof true);
```

```
// salida esperada: "boolean"
```

```
console.log(typeof declaredButUndefinedVariable);
```

```
// salida esperada: "undefined";
```

Tipos y ReadlineSync

Así como hablamos de tipos en las variables también es posible determinar que tipo de datos le pedimos que ingrese el usuario. Para esto readline-sync provee diferentes métodos:

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

```
readlineSync.question([mensaje]);
```

ESTA LINEA NOS VA A LEER UN TEXTO

```
readlineSync.questionInt([mensaje]);
```

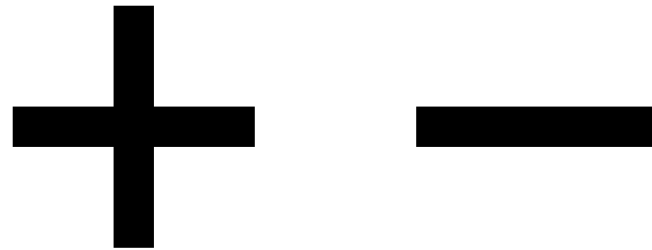
ESTA LINEA NOS VA A LEER UN NUMERO ENTERO

```
readlineSync.questionFloat([mensaje]);
```

ESTA LINEA NOS VA A LEER UN NUMERO DECIMAL

Operadores

- Son símbolos especiales que sirven para ejecutar una determinada operación, devolviendo el resultado de la misma



Operadores

Operador	Significado	Ejemplo
=	Asignación	nombre = "Juan"
+	Suma	total = cant1 + cant2
-	Resta	stock = disp - venta
*	Multiplicación	area = base * altura
/	División	porc = 100 * parte / total
**	Potenciación	sup = 3.41 * radio ** 2
% ó MOD	Resto de la división entera	resto = num MOD div

Secuencia

Prueba de Escritorio

- Una prueba de escritorio consiste en analizar (antes de hacer el algoritmo) cuál debe ser el resultado dada la entrada del algoritmo

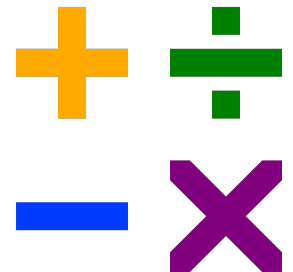
N° Prueba	Entrada		Salida	
	1er Num Ingresado	2do Num Ingresado	Suma	Mensaje
1	20	30	$20+30=50$	El resultado de la suma es: 50
2	15	150	$15+150=165$	El resultado de la suma es: 165
3	130	300	$130+300=430$	El resultado de la suma es: 430

Secuencia

Ejercicio: Suma de Dos Números

- Leemos los números desde el teclado y los guardamos en las variables

```
import * as readlineSync from 'readline-sync';  
  
let primerNumero : number = readlineSync.questionInt( "Ingrese el primer número: ");  
console.log("el primer número es ", primerNumero);  
let segundoNumero : number = readlineSync.questionInt( "Ingrese el segundo número: ");  
console.log("el segundo número es ", segundoNumero);
```

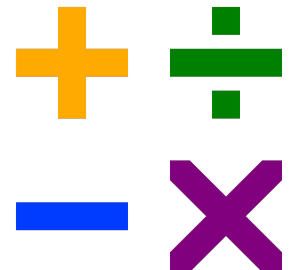


Secuencia

Ejercicio: Suma de Dos Números

- Realizamos la operación y mostramos el resultado

```
let resultado : number = primerNumero + segundoNumero;  
console.log("El resultado de la suma es:", resultado);
```



Secuencia

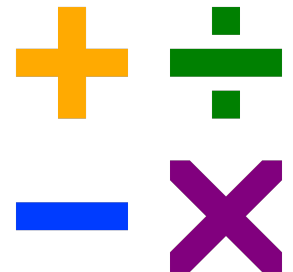
Ejercicio: Suma de Dos Números

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

let primerNumero : number = readlineSync.questionInt( "Ingrese el primer número: ");
console.log("el primer número es ", primerNumero);

let segundoNumero : number = readlineSync.questionInt( "Ingrese el segundo número: ");
console.log("el segundo número es ", segundoNumero);

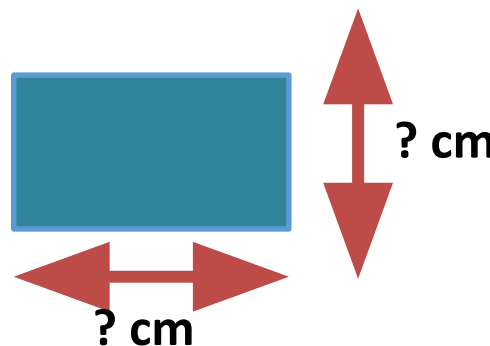
let resultado : number = primerNumero + segundoNumero;
console.log("El resultado de la suma es ", resultado);
```



Secuencia

Ejercicio: Área del Rectángulo

- Volvamos a implementar el proceso que calcula el área de un rectángulo pero **para cualquier base o altura**
 - El usuario debe ingresar la base y altura por teclado
 - El área debe guardarse en una variable
 - El resultado debe ser mostrado por pantalla



Secuencia

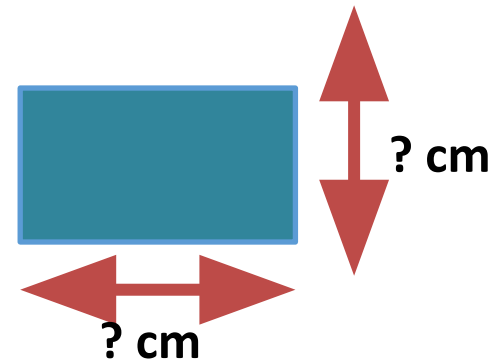
Ejercicio: Área del Rectángulo

- Leemos la base y la altura desde el teclado y las guardamos en las variables

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

```
let base : number = readlineSync.questionInt( "Ingrese la base: ");
```

```
let altura : number = readlineSync.questionInt( "Ingrese la altura: ");
```

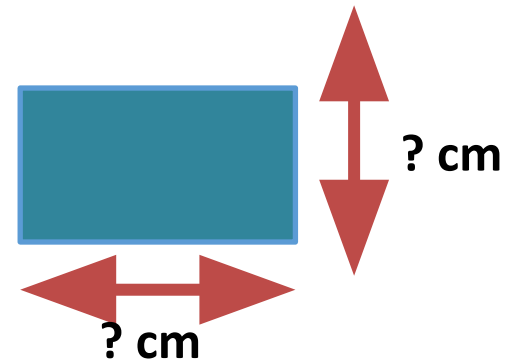


Secuencia

Ejercicio: Área del Rectángulo

- Calculamos el área y mostramos el resultado

```
let area : number = base * altura;  
console.log("El área es: ", area);
```



Secuencia

Ejercicio: Área del Rectángulo

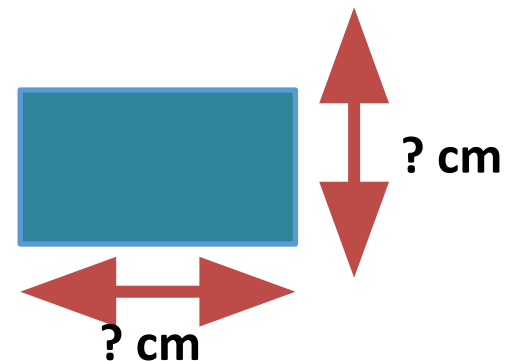
```
import * as readlineSync from 'readline-sync';
```

```
let base : number = readlineSync.questionInt( "Ingrese la base: ");
```

```
let altura : number = readlineSync.questionInt( "Ingrese la altura: ");
```

```
let area : number = base * altura;
```

```
console.log("El área es: ", area);
```



Técnicas de Programación

Programador full-stack

Ejercicios

Secuencia

Ejercicio: Cálculo de Descuento

- Implemente un algoritmo que calcule y muestre por pantalla el precio final de un producto después de aplicarle un descuento
 - El precio inicial del producto es \$450,50
 - El descuento a aplicar es del 10%. Recuerde que puede obtener el 10% de un valor multiplicado por 0,1
 - El precio y el descuento deben ser guardados en variables (no ingresados por teclado)

10% OFF

Estructuras de Control

Problema: Autos de Carrera

- En una prueba, un piloto tiene que completar 4 vueltas
- Se necesita un programa que permita ingresar por teclado el tiempo de cada vuelta
- El programa debe retornar el tiempo total y el promedio de vuelta

