



**UTN.BA**  
UNIVERSIDAD TECNOLÓGICA NACIONAL  
FACULTAD REGIONAL BUENOS AIRES

**Centro de  
e-Learning**

# **FUNDAMENTOS DE LA PROGRAMACIÓN**

Centro de e-Learning - FRBA - UTN

**Centro de e-Learning SCEU UTN - BA.**

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

**[www.sceu.frba.utn.edu.ar/e-learning](http://www.sceu.frba.utn.edu.ar/e-learning)**



**UTN.BA**  
UNIVERSIDAD TECNOLÓGICA NACIONAL  
FACULTAD REGIONAL BUENOS AIRES

**Centro de  
e-Learning**

p. 2

## **MÓDULO 2 - UNIDAD 6**

# **Arreglos y Funciones**

Centro de e-Learning - FRBA - UTN

**Centro de e-Learning SCEU UTN - BA.**

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

**[www.sceu.frba.utn.edu.ar/e-learning](http://www.sceu.frba.utn.edu.ar/e-learning)**



## Presentación:

En esta Unidad nos enfocaremos en varios temas importantes que se encuentran interrelacionados: las variables y constantes y los tipos de datos a los cuales pueden pertenecer.

Luego, nos enfocaremos en una tipo complejo de variables: los arreglos, donde analizaremos los de tipo unidimensional (o vectores) y los bidimensionales (o matrices).

Finalizamos los contenidos con un nuevo tema: las funciones, donde analizaremos su uso, tipos y posibilidades.

Todos estos temas son progresiones de los temas vistos anteriormente y representan, también, los contenidos sobre los cuales se irán agregando más complejidad y nuevos temas complementarios.



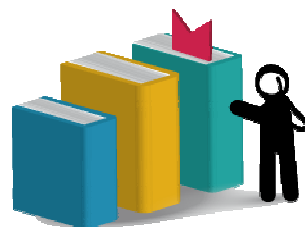
## Objetivos:

### Que los participantes:

- Comprendan los conceptos y uso de los vectores y matrices
- Incorporen el uso de funciones y conocer los tipos



## Bloques temáticos:



### 1. Arreglos

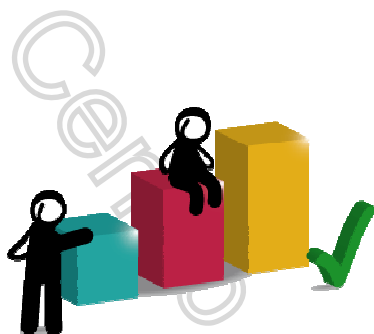
- 1.1 Definición
- 1.2 Arreglos unidimensionales: vectores
- 1.3 Arreglos bidimensionales: matrices
- 1.4 Ejercicio resuelto

### 2. Funciones

- 2.1 Definición
- 2.2 Cómo seguir el flujo de ejecución
- 2.3 Variables locales y globales
- 2.4 Funciones con parámetros
- 2.5 Devolución de valores

### 3. Tips para el uso de funciones y variables

- 3.1 Ejemplos resueltos de funciones y variables
- 3.2 Ejemplos resueltos funciones y arreglos



## Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC\*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

\* El MEC es el modelo de E-learning colaborativo de nuestro Centro.

**Centro de e-Learning SCEU UTN - BA.**

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

**[www.sceu.frba.utn.edu.ar/e-learning](http://www.sceu.frba.utn.edu.ar/e-learning)**



## Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.



## **1. Arreglos**

### **1.1 Definición**

Los arreglos son la primera estructura de datos no simple que analizaremos.

Cabe diferenciar los conceptos de “estructura de control” por sobre “estructura de datos”. La primera se refiere a los diferentes tipos de instrucciones interrelacionadas entre sí y que cumplen un objetivo en común. En cuanto a la segunda, se trata de distintas composiciones de datos simples, con el fin de crear datos complejos, interrelacionados entre sí a nivel de estructura.

Como vimos en el primer Módulo, la unidad de información mínima es el bit, que tiene un valor representativo de 1 o 0. Una composición mayor sería el byte, un conjunto de 8 bits, equivalente a un carácter. Podemos decir, entonces, que una variable que contiene una palabra, por ejemplo, “Lis” es una composición de 3 bytes, o un “string”. Este es un ejemplo de cómo se pueden componer distintas estructuras, combinándolas, para obtener estructuras de cada vez mayor complejidad, según sea necesario.

### **1.2 Arreglos unidimensionales: vectores**

El primer tipo de arreglo que analizaremos son los llamados vectores o arreglos unidimensionales, que se los puede representar figurativamente como una fila de casilleros, que posee un número finito de posiciones y donde cada casillero tiene un elemento anterior y otro posterior (salvo el primero y el último, claro está). Así mismo, cada posición cuenta con un identificador llamado “índice”, que representa la posición absoluta en la cual se encuentra un casillero con respecto a los demás.

Dependiendo de la implementación (de cada lenguaje de programación), los índices se empiezan a contar a partir del cero o del uno. A modo de convención, en este curso nombraremos a los índices a partir del número uno (1).





Es posible comparar a cada una de las posiciones de un vector (cada “casillero”) como una variable en sí misma, que se encuentra conectada con otras, aunque cabe aclarar que los arreglos sólo soportan un único tipo de dato para toda la estructura. Por lo que podemos agregar a lo dicho anteriormente, que es una “cadena” de **variables del mismo tipo**, interconectadas entre sí.

Algunos lenguajes de programación (como C) tienen un tipo de datos llamados “char”. Este tipo de datos se declara como un arreglo, ya que lo que contiene el vector es una cadena de bytes (si es que colocamos una letra en cada posición del vector), en la cual cada una de esas letras de esa cadena se encuentra en una posición unívocamente identificable del vector.

Para ejemplificar esto último y lo dicho anteriormente, veremos algunos ejemplos gráficos de representaciones conceptuales de vectores:

<b>L</b>	<b>i</b>	<b>s</b>
1	2	3

Aquí tenemos un arreglo unidimensional, o sea, un vector de tres posiciones, donde en la posición 1 tenemos el valor “L”, en la posición 2 tenemos el valor “i” y en la posición 3 tenemos el valor “s”. En su conjunto, se puede interpretar que esta cadena de caracteres forma la palabra “Lis”.

Vector de los hobbits de la Comunidad del Anillo:

<b>Frodo</b>	<b>Merry</b>	<b>Pippin</b>	<b>Sam</b>
1	2	3	4

En este caso tenemos un vector de 4 posiciones, con los siguientes valores:



- Posición 1: cuyo valor es “Frodo”
- Posición 2: cuyo valor es “Merry”
- Posición 3: cuyo valor es “Pippin”
- Posición 4: cuyo valor es “Sam”

## 1.3 Arreglos bidimensionales: matrices

Siguiendo con la tónica de agregar complejidad (y posibilidad de uso) a las estructuras de datos, nos encontramos con las matrices. Esta es una estructura comúnmente utilizada para guardar datos complejos que se encuentran doblemente relacionados entre sí.

La metáfora de la matriz podría ser un tablero de ajedrez (o la tabla de posiciones del Calcio, recuerden los ejemplos de la **Unidad 2**), donde tenemos una estructura de casilleros que responden a dos coordenadas de un eje cartesiano: x e y. De aquí se deriva el término bidimensional.

Un ejemplo gráfico podría ser la representación de un edificio de 3 pisos (eje vertical) y de 5 departamentos por piso (eje horizontal), en el cual identificamos según las coordenadas los apellidos de los habitantes del edificio:

<b>López</b>	<b>Pérez</b>	<b>Gómez</b>	<b>Fargo</b>	<b>Canal</b>
1,1	1,2	1,3	1,4	1,5
<b>Bayton</b>	<b>Ford</b>	<b>Santo</b>	<b>Trapiche</b>	<b>Stark</b>
2,1	2,2	2,3	2,4	2,5
<b>Casero</b>	<b>Black</b>	<b>Tina</b>	<b>Casio</b>	<b>King</b>
3,1	3,2	3,3	3,4	3,5



Por otro lado, podemos pensar que cada una de las filas (eje horizontal) puede ser un conjunto de datos relacionados entre sí, diferenciado de otro grupo de datos, que sería la siguiente fila. Adicionalmente, cada columna (eje vertical) podría contener datos que tengan sentido para todas las filas.

Para clarificar este punto, les propongo el siguiente ejemplo, que podríamos denominar “catálogo de películas”:

<b>1</b>	<b>Terminator</b>	<b>Acción</b>	<b>Disponible</b>
1,1	1,2	1,3	1,4
<b>2</b>	<b>Tron</b>	<b>Ciencia Ficción</b>	<b>Alquilada</b>
2,1	2,2	2,3	2,4
<b>3</b>	<b>Click</b>	<b>Comedia</b>	<b>Alquilada</b>
3,1	3,2	3,3	3,4
<b>4</b>	<b>Pelotón</b>	<b>Bélica</b>	<b>Disponible</b>
4,1	4,2	4,3	4,4

Este ejemplo, que podría representar el estado de una listado de películas en un video club, es una matriz en la cual cada una de la filas representa una película. A su vez, si observamos cada una de las columnas, podemos ver que la primera se trata de un código identificador, la segunda del nombre de la película, la tercera del género y la cuarta del estado de disponibilidad. Así es como obtenemos un conjunto de datos relacionados entre sí (fila) y otro conjunto de datos que tienen sentido entre sí (columna).

Cuando creamos este tipo de estructuras, es necesario mantener la coherencia de las posiciones seleccionadas. Por ejemplo, piensen en lo complejo que sería manejar una matriz en la que en una fila el nombre de película se encuentra en la posición 2, pero en la siguiente fila está ubicada en la posición 4.

Por otro lado, sería viable determinar que las columnas son las que tienen los datos de una película. Es posible y ningún compilador fallaría, aunque con respecto a esto



debemos decir que por convención ampliamente respetada, los datos relacionados entre sí (en este caso películas) se colocan en las filas y no en las columnas. Mantendremos esto también como estándar y buena práctica para el resto del curso.

## 1.4 Ejercicio resuelto

```
programa DiasDeLaSemana
inicio
    var integer dia
    var string diasDeLaSemana [7]
    diasDeLaSemana [1] = "Es lunes"
    diasDeLaSemana [2] = "Es martes"
    diasDeLaSemana [3] = "Es miércoles"
    diasDeLaSemana [4] = "Es jueves"
    diasDeLaSemana [5] = "Es viernes"
    diasDeLaSemana [6] = "Es sábado"
    diasDeLaSemana [7] = "Es domingo"

    mostrar: "Ingrese día de la semana"
    ingresar: dia

    si dia > 0 y dia < 8 entonces
        mostrar: diasDeLaSemana [dia]
    sino
        mostrar: "Día incorrecto"
    fin si
fin
```



En este sencillo ejemplo, siguiendo con el tema de los días de la semana, podemos ver cómo se declara un vector, con la instrucción “var string diasDeLaSemana [7]”, la cual analizaremos por partes:

- “var” indica que es variable
- “string” indica que contendrá caracteres
- “diasDeLaSemana” es el nombre de la variable
- “[7]” indica que es un vector y que contendrá 7 posiciones

Este programa nuevamente toma un número ingresado por el usuario y lo utiliza como índice del vector para mostrar el contenido de esa posición del arreglo. Se agrega una pequeña validación tendiente a comprobar que se haya ingresado un número que se encuentre en el rango esperado (entre 1 y 7, expresado como “mayor a cero” y “menor a ocho”). En esta estructura condicional podemos ver cómo se puede agregar más de una condición a una “si entonces” con el conector “y”.

Ejemplo de uso de matriz:

```
programa AnalisisFoda
inicio
    var string matrizFoda [2] [2]

    mostrar: "Ingreso Fortalezas:"
    ingresar: matrizFoda [1] [1]           //ejemplo de ingreso de datos
    mostrar: "Ingreso Oportunidades:"
    matrizFoda [1] [2] = "No disponible"   //ejemplo de asignación
    mostrar: "Ingreso Debilidades:"
    ingresar: matrizFoda [2] [1]
    mostrar: "Ingreso Amenazas:"
    ingresar: matrizFoda [2] [2]

    mostrar: matrizFoda [1] [1]
    mostrar: matrizFoda [1] [2]
    mostrar: matrizFoda [2] [1]
    mostrar: matrizFoda [2] [2]
fin
```



En este ejemplo vemos cómo se carga una matriz, puntualmente una FODA (esta es una herramienta clásica de gestión que sirve para analizar un negocio o situación en particular a través de los 4 ejes que la forman y le dan nombre: Fortalezas, Oportunidades, Debilidades y Amenazas).

Vemos cómo se realiza una declaración de la matriz en forma similar a la del vector, aunque se agregan un par de corchetes ([ ][ ]) que son los que determinan que se trata de una matriz, al crearse la estructura con dos dimensiones. Se trata de una matriz de 2 x 2 posiciones.

#### **IMPORTANTE:**



Los tamaños (cantidad de filas y columnas en las matrices y la cantidad de posiciones de los vectores) son **ESTÁTICOS**, por lo que en la declaración deben ser especificados explícitamente colocando **NÚMEROS** entre los corchetes. **NO** se permite colocar variables, ni dejar los corchetes vacíos en la declaración. Tampoco se puede cambiar estas cantidades durante la ejecución del programa agregando o eliminando filas, columnas y posiciones. Todo esto, aplica al pseudocódigo.

#### **TIP!**



Si vemos corchetes en el código, sabremos que se está haciendo referencia a arreglos. Si hay un par de corchetes, es un vector. Si hay dos pares de corchetes, es una matriz.



## 2. Funciones

Saliendo del tema que ocupa principalmente esta Unidad, incorporaremos un último concepto de cara a escribir un mejor código.

### 2.1 Definición

Uno de los mejores y más importantes aportes del paradigma estructurado es la posibilidad de subdividir los programas en forma lógica y funcional. Hasta ahora nos encontramos con ejemplos de programas relativamente cortos y simples.

¿Cómo podríamos concebir entonces cómo están escritos los programas de la NASA o, sin ir más lejos, los de un banco? ¿Como una lista inmensa de instrucciones, una debajo de otra, probablemente con cientos de millones de líneas? Es posible, claro está, pero si apelamos nuevamente a nuestro sentido común, nos daremos cuenta de lo inmanejable que podría resultar un programa así.

Entonces, en esta clase de contextos, donde no tenemos una simple rutina que muestra números o días de la semana, sino que tenemos que representar el funcionamiento de un negocio, debemos apelar a aquellos elementos que nos pueden facilitar el hecho de escribir código simple y legible.

Podemos decir que una función es una porción de código fuente que se agrupa con algún motivo (normalmente de índole lógica-funcional) y que permite su invocación y reutilización una cantidad ilimitada de veces. **Para esto definiremos una nueva palabra reservada. En este caso será “funcion”** (el tilde sobre la “o” se encuentra omitido).

La nomenclatura a seguir será:



```
programa EjemploDeFuncion
inicio
    // declaración de variables
    funcion principal ()
        // instrucciones de la función principal
        otraFuncion() //invocación a función
    fin funcion

    funcion otraFuncion () //encabezado de la función principal
        // instrucciones de la funcion otraFuncion
    fin funcion
fin
```

Entonces, tendremos **siempre** una función llamada “principal” que será siempre el punto de entrada en la ejecución del programa.

La lógica del funcionamiento de las funciones es que, cuando se vienen ejecutando las instrucciones en forma secuencial (como lo sostiene el paradigma estructurado) y el programa se encuentra con una “invocación” a otra función, el programa da “un salto” a la función invocada, completa todas las instrucciones de la misma y retorna a la función invocadora, donde sigue ejecutando las siguientes instrucciones.

## 2.2 Cómo seguir el flujo de ejecución

Veremos en este ejemplo cómo se lee el código con la incorporación de las funciones. Para eso, veamos el siguiente ejemplo:





```
programa EjemploParaLecturaDeProgramaConFunciones
inicio
    funcion principal ()                                //1
        var string seguirONo = "n"                    //2

        mostrar: "Funcion 'calculo()'"                //3
        calculo()                                     //4
        |      |      |      |      |      |      |      |      //14
        mostrar: "Desea seguir? Ingrese s/n"          //15
        ingresar: seguirONo                          //16

        si seguirONo = "s" entonces                   //17
            principal()                                //18
        sino
            mostrar: "Gracias y vuelva pronto"        //19
        fin si

    fin funcion                                        //20

    funcion calculo ()                                //5
        var integer uno                                //6
        var integer dos                                //6
        var integer rdo                                //6

        mostrar: "Ingresar el primer número"          //7
        ingresar: uno                                  //8
        mostrar: "Ingresar el segundo número"         //9
        ingresar: dos                                  //10

        rdo = uno + dos                                //11

        mostrar: "El resultado es " + rdo             //12
    fin funcion                                        //13
fin
```



#### Secuencia de ejecución:

1. Se debe identificar la función "principal": es una función obligatoria, y el "punto de entrada" de todo programa. Todo programa debe tener su función "principal".
2. Se declara e inicializa la variable "seguirONo"
3. Se muestra un mensaje
4. Se invoca la función "calculo()". El flujo de ejecución de la función "principal" se interrumpe y se produce un "salto" a la función invocada.
5. Se debe identificar la función "calculo". El flujo sigue en forma secuencial dentro de esta función.
6. Se declaran las variables "uno", "dos" y "rdo".
7. Se muestra un mensaje
8. Se ingresar un dato
9. Se muestra otro mensaje
10. Se ingresa otro dato
11. Se hace la suma de las variables "uno" y "dos" y se guarda su valor en "rdo"
12. Se muestra un mensaje al que se le concatena el valor de la variable "rdo"
13. Finaliza la función "calculo" y el flujo retorna al punto desde dónde se hizo la invocación a esta función.
14. El flujo retornar a la función "principal".
15. Se muestra un mensaje
16. Se pide el ingreso de un dato
17. Se hace el condicional
18. Si la condición es verdadera, se invoca a la función "principal", por lo que el programa se "reinicia", volviendo al punto 1.
19. Si la condición es falsa, se muestra un mensaje
20. Finaliza la función "principal" y con esto finaliza la ejecución de todo el programa



## 2.3 Variables locales y globales

Algo importante que hay que tener en cuenta: dónde se declaran las variables.

1) Si declaramos una variable DENTRO de una función, sea la "principal" o cualquier otra, se dice que esa variable va a ser "local" de función. Esto significa que puede ser utilizada dentro de la función, pero una vez que esta finaliza, la variable se DESTRUYE y ya no es posible volver a utilizarla

2) Si declaramos una variable FUERA de cualquier función, se dice que esa variable será "global" del programa. Esto significa que su valor podrá ser leído y/o modificado en cualquier función del programa.

La diferencia entre ambas está dada por la eficiencia de la solución: es mucho más "performante" utilizar variables locales, dado que eso implica que la memoria de la máquina se usa y libera constantemente. Pero si sabemos que vamos a tener que usar una variable en todas o la gran mayoría de las funciones, la podemos declarar global, para evitar tener que estar enviándola como parámetro a todas las funciones que la requieren, pero sabiendo que la memoria que ocupa esa variable global va a quedar ocupada hasta que el programa finalice. Si esa variable empieza a aumentar en tamaño, puede llegar a ser problemático de cara a los recursos disponibles.

## 2.4 Funciones con parámetros

Los parámetros son valores que reciben las funciones y permiten ser manipulados dentro de las mismas.

Estos serán declarados a continuación del nombre de la función entre paréntesis, debiendo indicarse de qué tipo de datos van a ser los parámetros recibidos, ya que tienen el mismo tratamiento que las variables locales.



```
programa EjemploDeFuncionConParametros
inicio
    funcion principal () //encabezado con la declaración de la función principal
        var integer a
        var integer b

        mostrar: "Ingrese valores:"
        ingresar: a
        ingresar: b
        suma(a, b) // se invoca a la función con parámetros
        mostrar: "Fin del programa"
    fin funcion

    funcion suma(integer primero, integer segundo)
        var integer rdo

        rdo = primero + segundo
        mostrar: "El resultado es "
        mostrar: rdo
    fin funcion
fin
```



### ¡¡IMPORTANTE!!:

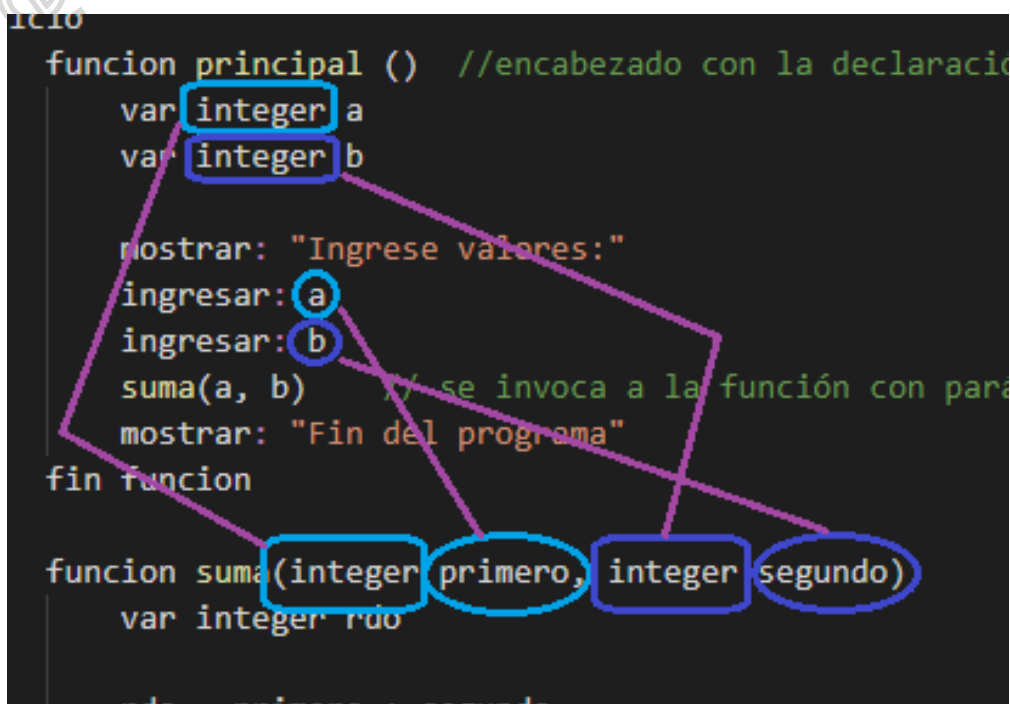
Las funciones son un tema extremadamente importante que estaremos trabajando, retomando y agregando complejidad hasta el fin del curso. ¡ES IMPORTANTE QUE EL CONCEPTO LES QUEDE MUY CLARO!

En este ejemplo podemos ver cómo la función principal “delega” la realización de la suma en una función específica para este fin.

La invocación de una función se diferencia del uso de una variable por los "()":

- sin "()" estaremos haciendo referencia a una variable
- con "()" estaremos invocando una función. Tenga o no parámetros, siempre usamos "()"

Los parámetros o datos que se envían en la invocación deben estar en el mismo orden secuencial en que están declarados en el encabezado de la función. También se debe respetar el tipo de dato de cada parámetro, como se puede ver a continuación:



```
1010
funcion principal () //encabezado con la declaraci
    var integer a
    var integer b

    mostrar: "Ingrese valores:"
    ingresar: a
    ingresar: b
    suma(a, b) // se invoca a la función con para
    mostrar: "Fin del programa"
fin funcion

funcion suma(integer primero, integer segundo)
    var integer rdo
    rdo = primero + segundo
```

The diagram illustrates the relationship between function declarations and calls. In the 'funcion principal' block, variables 'a' and 'b' are declared as 'integer'. These are then used in the 'ingresar' statements and passed to the 'suma' function. In the 'funcion suma' block, parameters 'primero' and 'segundo' are declared as 'integer'. The call 'suma(a, b)' in the first block matches the parameter types and order in the second block. Blue boxes and circles highlight these correspondences: 'integer' in the first declaration, 'a' and 'b' in the inputs, and 'integer primero' and 'integer segundo' in the second declaration.

La "declaración" o "encabezado" de la función es parte del código donde inicia la función. La fórmula a seguir es (hasta ahora):

- palabra reservada "funcion"
- nombre de la función
- (si tiene) el listado de los argumentos o parámetros: entre "()" y siempre con el tipo de dato y el nombre del parámetro.

En el ejemplo de arriba hay 2 declaraciones de funciones:

- "funcion principal()"
- "funcion suma(integer primero, integer segundo)"



Veremos a continuación que esto puede llegar a cambiar.

## 2.5 Devolución de valores

Uno de los sentidos principales de las funciones es la posibilidad de devolver valores a quien la invocó. De esta manera podemos devolver el resultado de un cálculo, un mensaje, un resultado de una condición, etc. al programa, según el motivo y actividad de la función.

**Lo que vaya a devolver puede ser almacenado en una variable, ya que todas las funciones siempre devuelven uno y sólo un valor.**

Dicho valor, como venimos viendo en esta Unidad, va a ser un tipo de dato. Por eso es que debemos declarar en el encabezado de la función cuál será este tipo devuelto. Lo podemos ver el ejemplo anterior, modificado para que funcione con valores por devolución:

Ahora, la "declaración" o "encabezado" de una función sigue esta nueva fórmula:

- palabra reservada "funcion"
- (si tiene) tipo de dato del valor que devuelve ("retorna")
- nombre de la función
- (si tiene) el listado de los argumentos o parámetros: entre "()" y siempre con el tipo de dato y el nombre del parámetro.



```
programa EjemploDeFuncionCompleto
inicio

    funcion principal ()
        var integer a
        var integer b

        mostrar: "Ingrese valores: "
        ingresar: a
        ingresar: b
        mostrar: "El resultado es "
        mostrar: suma(a, b)
    fin funcion

    funcion integer suma(integer primero, integer segundo)
        var integer rdo
        rdo = primero + segundo
        retornar: rdo
    fin funcion
fin
```

### IMPORTANTE:



- Si una función tiene un "retornar:" dentro SI o SI la función deberá declarar que devuelve un tipo de dato. Caso contrario, hay error.
- Una función que declara que devuelve un dato y no hace un "retornar:" en su interior representa un error.



### Las cosas por su nombre

Les dejo un extracto del libro "Código Limpio" de Robert C. Martin que hace referencia a la importancia de establecer nombres descriptivos, significativos y claros.

Si bien habla de funciones, esto es aplicable, también, a los nombres de los programas, las variables y las constantes.

**<< La selección de nombres correctos para una función mejora la explicación de su cometido, así como el orden y la utilidad de los parámetros. En formato monádico(\*), la función y el argumento deben formar un par de verbo y sustantivo. Por ejemplo, `write(name)` resulta muy claro. Sea lo que sea "name", sin duda se escribe ("write", en inglés).**

**Un nombre más acertado podría ser `writeField(name)`, que nos dice que name es un campo ("field", en inglés) [Nota EER: por ejemplo el "campo" de un formulario de una página web]. (...) Con este formato definimos los nombres de los parámetros en el nombre de la función. Por ejemplo, `assertEquals` se podría haber escrito como `assertExpectedEqualsActual(expected, actual)`, lo que mitiga el problema de tener que recordar el orden de los argumentos. >>**

(\*) Monádico: este término se refiere a las funciones que reciben un sólo parámetro.

Como verán, más corto no es necesariamente mejor. Tal vez sea mejor usar nombres largos, pero descriptivos, para evitar interpretaciones o estructuras crípticas o difíciles de entender a primera vista.





### **3. Tips para el uso de funciones y variables**

En el ejemplo integrador resuelto presentado a continuación vamos a encontrar ejemplos de uso de variables locales y globales además de funciones.

Localicen en el ejemplo cada uno de los conceptos que se listan abajo:

- (1) Variables globales, que pueden ser usadas en cualquier función
- (2) Variables locales, que solo pueden ser usadas dentro de la función dónde se declaran
- (3) Funciones propias
- (4) Invocación a funciones propias
- (5) Funciones propias que reciben parámetros
- (6) Funciones propias que devuelven valores
- (7) Funciones propias que reciben parámetros y devuelven valores



```
programa Calculo
inicio
    var integer nro1    //(1)

    funcion principal()
        var integer rdo = 0
        nro1 = 0
        reciboSumoYMuestro(23, 25) //(4)
        rdo = sumoDiezYDevuelvo() //(4) + guardo en una variable "rdo" lo
        //que devuelve la función, luego lo muestro
        mostrar: rdo
        rdo = reciboSumoDiezYDevuelvo(100, 200) //(4) + guardo en una variable "rdo"
        //lo que devuelve la función, luego lo muestro
        mostrar: rdo
    fin funcion

    //(3)
    //(5)
    funcion reciboSumoYMuestro(integer a, integer b)
        var integer rdo = 0    //(2)
        rdo = a + b
        mostrar: rdo
    fin funcion

    //(3)
    //(6)
    funcion integer sumoDiezYDevuelvo()
        var integer rno    //(2)
        nro1 = nro1 + 10    //uso la variable global
        rno = nro1
        retornar: rno //(6)
    fin funcion

    //(3)
    //(7)
    funcion integer reciboSumoDiezYDevuelvo(integer a, integer b)
        var integer rdo    //(2)
        rdo = a + b + 10
        retornar: rdo
    fin funcion
fin
```



Algunas reglas:

- Nunca se pone la palabra reservada "funcion". Esa palabra se usa ÚNICAMENTE en el encabezado de la función. NO para invocarla

```
//do = sumoDiezYDevuelvo() // (4) + guardo en una
//que devuelve la
mostrar: rdo
rdo = reciboSumoDiezYDevuelvo(100, 200) // (4) + gu
//lo q
mostrar: rdo
fin funcion

// (3)
// (5)
funcion reciboSumoYMuestro(int a, int b)
var int rdo = 0 // (2)
rdo = a + b
mostrar: rdo
fin funcion

// (3)
// (5)
```

Invocación

Declaración

- Nunca se envían por parámetros variables globales (no es un error de código, pero sí un error conceptual, porque las variables globales se puede usar desde cualquier función del programa).
- Nunca va a haber código (asignaciones, "mostrar:", "ingresar:", etc.) fuera de las funciones, salvo la declaración de variables globales.
- Si tengo un "retornar" en mi función, la función debe declarar que devuelve un valor (esto se hace colocando el tipo de dato de la variables o valores que se va a devolver entre la palabra "funcion" y el nombre de la función).

Ejemplo:

```
funcion boolean validacion(String u, String p)
```



```
funcion integer suma(integer primero, integer segundo)
```

```
funcion integer sumoDiezYDevuelvo()
```

```
funcion integer reciboSumoDiezYDevuelvo(integer a, integer b)
```

```
funcion boolean validacion()
```

- Si declaro que mi función devuelve un valor, SI o SI debo tener algún "retornar" adentro
- No puedo tener variables locales que se llamen igual que una variable global. Si hay una variable local o parámetro de una función que se llama igual que una variable global, la estaría duplicando, lo cual es un error
- Sí se puede tener variables locales en una función que se llaman igual que otras variables locales de otra función
- Es una mala idea tener funciones y variables con el mismo nombre (se presta a confusión)
- No importa el orden de las funciones: no es necesario que "principal" sea la primera, ni se requiere que se siga ningún orden en las funciones propias
- Cuando se llega al "fin funcion" de la función "principal", el programa finaliza.
- Cuando se llega al "fin funcion" de una función propia, el flujo vuelve al punto donde se hizo la invocación a esa función.
- Cualquier función puede invocar a cualquier función, incluso puede invocarse a sí misma (esto se llama "recursividad")



- La función "principal" nunca devuelve valores ni recibe parámetros
- Si un parámetro (por ejemplo, una variable local) es enviado a una función, se supone que dicha función va a necesitar de su valor, por lo que no tiene sentido modificar (asignar) o ingresar un nuevo valor en ese parámetro. Si bien no sería un error de código, sí es un error conceptual.

Por ejemplo:

```
funcion yVaElTercero(String nombre)
|   nombre = "Pity" //no hay error, pero no tiene sentido:
|   |   |   |   |   // para qué se usa el valor enviado si se va pisar?
|   mostrar: "El nombre es " + nombre
fin funcion

funcion yVaElTercero(String nombre)
|   mostrar: "Ingresar el nombre "
|   ingresar: nombre //no hay error, pero no tiene sentido:
|   |   |   |   |   // para qué se usa el valor enviado si se va pisar?
|   mostrar: "El nombre es " + nombre
fin funcion

funcion yVaElTercero(String nombre)
|   mostrar: "El nombre es " + nombre //forma correcta de usar el parámetro
fin funcion
```

- Vale aclarar que no puede haber más de una función con un mismo nombre, más allá de este ejemplo fuera del contexto de un programa.
- El "retornar" interrumpe la ejecución del flujo de instrucciones de toda función, por lo que no puede haber código luego de un "retornar":.

Ejemplo:



```
funcion boolean validacion(string nombre)
  si nombre = "Enzo" entonces
    retornar: verdadero      //sale por acá
  sino
    retornar: falso          //o sale por acá
  fin si
  mostrar: "Gracias"         //esta instrucción nunca se ejecuta
fin funcion

funcion string validacion()
  var string nombre
  ingresar: nombre
  retornar: nombre           //vuelve a la función que hizo la invocación

  mostrar: "El nombre es " nombre //esta instrucción nunca se ejecuta
fin funcion
```

- Se podría combinar el "mostrar:" con la invocación a una función que retornar algún valor. De esta forma, en una sola línea, se puede mostrar el resultado o valor que devuelve una función, y nos evitamos tener que declarar una variable nueva (\*).

Ejemplo:



```
//Correcto! La función devuelve un valor
// y lo puedo mostrar!
mostrar: autorDeLaLlamada()

funcion String autorDeLaLlamada()
    retornar: "Homero"
fin funcion

//ERROR!! La función no devuelve ningún valor
// NO HAY NADA PARA "mostrar"

mostrar: plagiadorDeLaLlamada()

funcion plagiadorDeLaLlamada()
    mostrar: "Moe"
fin funcion
```

- Tener en cuenta que esto se puede hacer únicamente cuando la función invocada retorna un valor.
- No se puede combinar el "ingresar:" con una invocación, solo el "mostrar:"

(\*) La alternativa sería usar una variable más.

```
//Expresión 1: usando una variable adicional
var string nombre
nombre = autorDeLaLlamada()
mostrar: nombre

//Expresión 2: sin usar una variable adicional
mostrar: autorDeLaLlamada()
```

Ambas expresiones son equivalentes.



**TIP EXTRA:**

No sería una mala idea que se preparen un "checklist" con lo que sabemos que puede y no puede tener una función, para ir verificando en las primeras funciones que desarrollen. De esta forma, se evitarán omitir alguna de las reglas, logrando incorporar mejor y más rápido los nuevos conceptos.

### 3.1 Ejemplos resueltos de funciones y variables

Se presentan a continuación 4 formas distintas de resolver un programa que valida usuario y contraseña, utilizando:

- A) Variables *locales*, función **con** parámetros, función que no devuelve valores
- B) Variables *locales*, función **con** parámetros, función que devuelve valores
- C) Variables *globales*, función **sin** parámetros, función que no devuelve valores
- D) Variables *globales*, función **sin** parámetros, función que devuelve valores





**A)**

```
programa ConFunciones
inicio
    funcion principal()
        //variables locales
        var string usuario = ""
        var string pass = ""

        ingresar: usuario
        ingresar: pass
        //caso 1: invocacion con parametros pero sin valor devuelto
        validacion(usuario, pass)
    fin funcion

    funcion validacion(String u, String p)
        si u = "emilio" Y p = "EnMendozaYEnMadrid" entonces
            mostrar: "acceso dado"
        sino
            mostrar: "usuario o password incorrecta"
        fin si
    fin funcion
fin
```



B)

```
programa ConFunciones
inicio
  funcion principal()
    //variables locales
    var string usuario = ""
    var string pass = ""

    ingresar: usuario
    ingresar: pass

    //caso 2: con parametros y valor devuelto
    var boolean rsta
    rsta = validacion(usuario, pass)
    si rsta = verdadero entonces
      mostrar: "acceso dado"
    sino
      mostrar: "usuario o password incorrecta"
    fin si
  fin funcion

  funcion boolean validacion(String u, String p)
    si u = "emilio" Y p = "EnMendozaYEnMadrid" entonces
      retornar: verdadero
    sino
      retornar: falso
    fin si
  fin funcion
fin
```



**C)**

```
programa ConFunciones
inicio

    //variables globales
    var string usuario = ""
    var string pass = ""

    funcion principal()
        ingresar: usuario
        ingresar: pass

        //caso 3: sin parametros y sin valor devuelto
        validacion()
    fin funcion

    funcion validacion()
        si usuario = "emilio" Y pass = "EnMendozaYEnMadrid" entonces
            mostrar: "acceso dado"
        sino
            mostrar: "usuario o password incorrecta"
        fin si
    fin funcion
fin
```



D)

```
programa ConFunciones
inicio
    //variables globales
    var string usuario = ""
    var string pass = ""

    funcion principal()
        ingresar: usuario
        ingresar: pass

        //caso 4: sin parámetros y con valor devuelto
        var boolean rsta
        rsta = validacion()
        si rsta = verdadero entonces
            mostrar: "acceso dado"
        sino
            mostrar: "usuario o password incorrecta"
        fin si
    fin funcion

    funcion boolean validacion()
        si usuario = "emilio" Y pass = "EnMendozaYEnMadrid" entonces
            retornar: verdadero
        sino
            retornar: falso
        fin si
    fin funcion
fin
```



A tener en cuenta:

Una variable local puede ser enviada a una función como parámetro. Si ese parámetro en la función es modificado, el valor de la variable en la función original no cambia.

```
programa Variables
inicio

    var string miVarGlobal = "LP"

    funcion principal()
        var string miVarLocal = "JFQ"

        mostrar: miVarLocal      // muestra JFQ
        secundaria(miVarLocal)
        mostrar: miVarLocal      // muestra JFQ

        mostrar: miVarGlobal     //muestra LP
        terciaria()
        mostrar: miVarGlobal     //muestra Te puedo felicitar
    fin funcion

    funcion secundaria(string miParapam)
        mostrar: miParapam      //muestra JFQ
        miParapam = "GM"
        mostrar: miParapam      //muestra GM
    fin funcion

    funcion terciaria()
        miVarGlobal = "Te puedo felicitar"
    fin funcion
fin
```



## 3.2 Ejemplos resueltos funciones y arreglos

Lo más importante a tener en cuenta cuando se envían arreglos como parámetros o cuando una función devuelve un arreglo, es NO incluir el tamaño del arreglo, colocando los corchetes vacíos: "[ ]".

De otra forma, haciendo "miArr[3]" estaría haciendo referencia a la posición 3 del arreglo, en lugar del arreglo completo.

```
programa EjemploFuncionesConVectores
inicio
  funcion principal()
    var integer miArr[3]
    var integer miOtroArr[3]

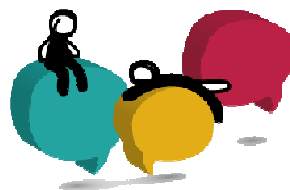
    miArr[1] = 10
    miArr[2] = 20
    miArr[3] = 30

    miOtroArr[] = miFunc(miArr[])
    miSegundaFunc(miArr[2])      //muestra 20
    miSegundaFunc(miOtroArr[2])  //muestra 21
  fin funcion

  funcion integer[] miFunc(integer arr[])
    arr[1] = arr[1] + 1
    arr[2] = arr[2] + 1
    arr[3] = arr[3] + 1

    retornar: arr[]
  fin funcion

  funcion miSegundaFunc(integer valor)
    mostrar: valor
  fin funcion
fin
```



## Actividades de la Unidad 6

### Ejercicio 1:

- Tomando el programa "DiasDeLaSemana"
- Modificarlo para que incluya funciones, separando la funcionalidad en bloques que tengan **coherencia lógica y utilidad real** en la o las funciones
  - Si inicializan un vector o matriz, recuerden hacer las asignaciones DENTRO de alguna de la funciones
  - Hacer un programa con la función "principal" y todo el código adentro no es válido.
  - Hacer un programa que en la función "principal" invoque a otra función y que todo el resto del código esté en esa función tampoco es válido.

### Ejercicio 2:

- Escribir un programa con las funciones "principal" y otra más definida por Ustedes, que va a recibir 4 parámetros de distintos tipos de datos.
- Desde la función "principal" van a hacer 1 invocación a esa función.
- Dicha invocación tienen que contar con:
  - 2 valores fijos (entre "" si son string, números si es un tipo de dato numérico, etc.)
  - 1 constante LOCAL
  - 1 variable LOCAL



- La declaración de la función propia tiene que coincidir con las invocaciones. O, al revés, las invocaciones tienen que ser compatibles con la declaración de la función.

Consignas:

- Utilizar el pseudocódigo presentado en el curso, con sus palabras reservas
- Utilizar los tipos de datos vistos presentados en el curso
- El código deberá estar **indentado**

*El incumplimiento de alguna de las consignas invalida el trabajo presentado*





## Lo que vimos

- Conceptos y uso de los vectores y matrices
- Uso de funciones y conocer los tipos



---

## Lo que viene:

- Integración de conceptos

