UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

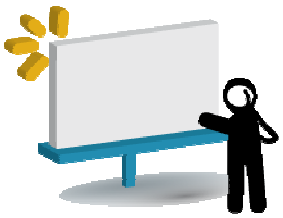
FUNDAMENTOS DE LA PROGRAMACIÓN

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

UTN.BA UNIVERSIDAD TECNOLÓGICA NACIONAL FACULTAD REGIONAL BUENOS AIRES | Centro de e-Learning

MÓDULO 2 - UNIDAD 6

Arreglos y Funciones



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.

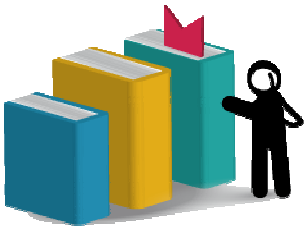
Todos estos temas son progresiones de los temas vistos anteriormente y representan, también, los contenidos sobre los cuales ser irán agregando más complejidad y nuevos temas complementarios.

Objetivos:

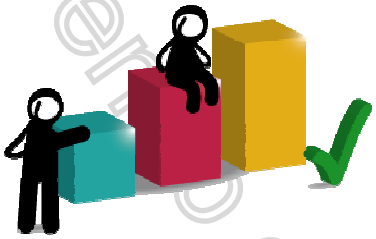
Que los participantes:

- 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:

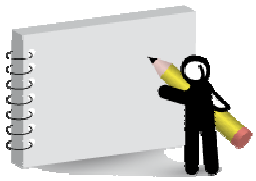
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

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.



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:

L	i	s
1	2	3

Aquí tenemos un arreglo unidimensional,o sea, un vector de tres posiciones, donde en la posicón 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:

Frodo	Merry	Pippin	Sam
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:

López	**Pérez**	**Gómez**	**Fargo**	**Canal**
1,1	**1,2**	**1,3**	**1,4**	**1,5**
Bayton	**Ford**	**Santo**	**Trapiche**	**Stark**
2,1	**2,2**	**2,3**	**2,4**	**2,5**
Casero	**Black**	**Tina**	**Casio**	**King**
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":

1	**Terminator**	**Acción**	**Disponible**
1,1	1,2	1,3	1,4
2	**Tron**	**Ciencia Ficción**	**Alquilada**
2,1	2,2	2,3	2,4
3	**Click**	**Comedia**	**Alquilada**
3,1	3,2	3,3	3,4
4	**Pelotón**	**Bélica**	**Disponible**
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 ingresa 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: "Ingrese Fortalezas:"
    ingresar: matrizFoda [1] [1]            //ejemplo de ingreso de datos
    mostrar: "Ingrese Oportunidades:"
    matrizFoda [1] [2]  = "No disponible"   //ejemplo de asignación
    mostrar: "Ingrese Debilidades:"
    ingresar: matrizFoda [2] [1]
    mostrar: "Ingrese 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 prontos"             //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 declamamos 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!!:

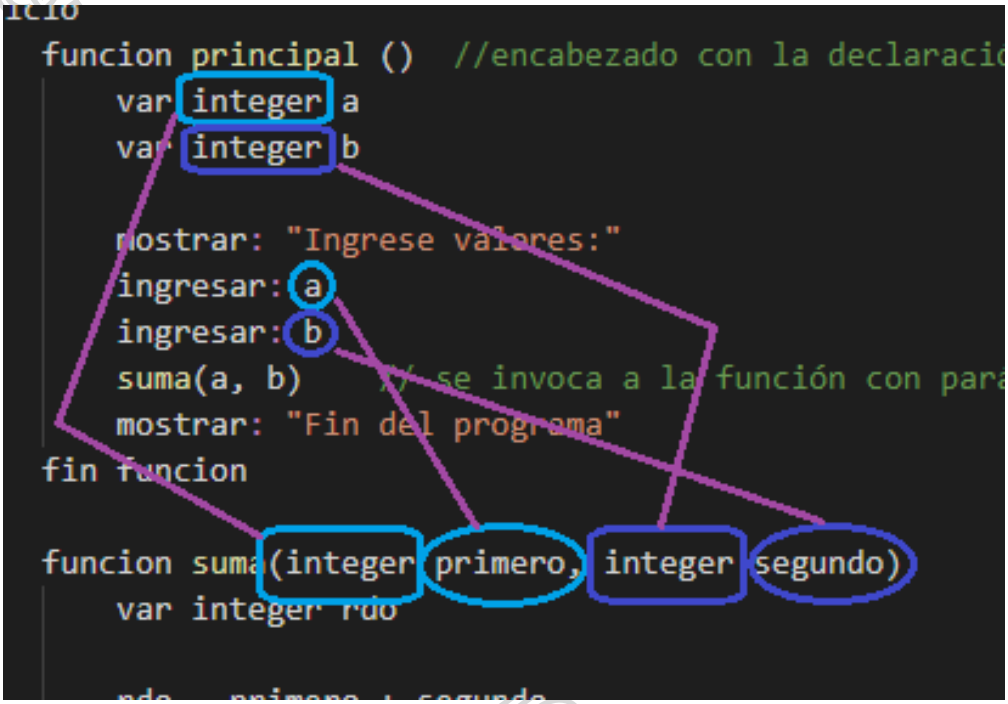
Las funciones son un tema extremadamente importante que estaremos trabajando, retomando y agregando complejidad hasta el fin del curso. ¡ES IMPORTANTE QUE EL CONCEPTO LES QUEDE MUY CLARO!

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 de declarados en el encabezado de la función. También se debe respetar el tipo de dato de cada parámetro, como se puede ver a continuación:



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 hacer 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 el 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

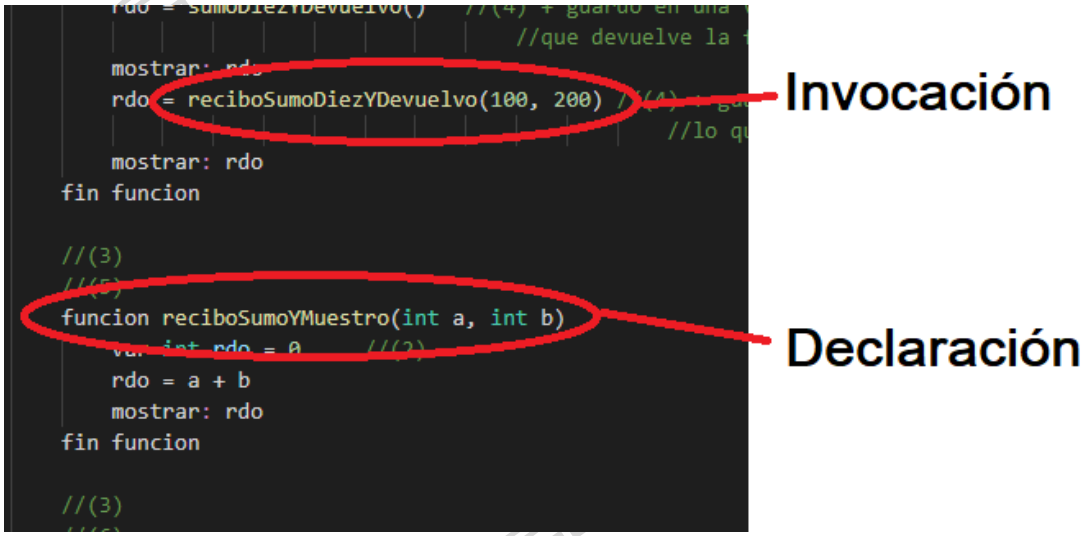
(6) Funciones propias que devuelven valores

(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



- 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 enviando 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: autorDeLaLlamarada()

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


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

mostrar: plagiadorDeLaLlamarada()

funcion plagiadorDeLaLlamarada()
    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 = autorDeLaLlamarada()
mostrar: nombre

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

Ambas expresiones son equivalentes.

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 devueto
        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)
        motrar: valor
    fun funcion
fin
```

ACTIVIDAD DE ANÁLISIS:

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
- Se puede incluir algún cambio de alcance agregando nuevas funcionalidades

Ejercicio 2:

- Escribir un programa con las funciones "principal" y otra más definida por Ustedes, que va a recibir 3 o 4 parámetros de distintos tipos de datos.
- Desde la función "principal" van a hacer 1 invocación a esa función.
- 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.

Podrán comparar sus respuestas con los ejemplos resueltos en el "ANEXO 1 - Respuesta Actividad de Análisis" al final de esta unidad.

Previamente pueden consultar el "ANEXO 2 - Checklist de código".

IMPORTANTE:

- **¡Traten de resolverlo cada uno por sus propios medios!**
- **Las respuestas (el programa) NO debe ser enviado a los foros del Campus, salvo dudas puntuales de las respuestas.**

ANEXO 1 - Respuesta Actividad de Análisis

¿Trataron de resolver los requerimientos?

En caso afirmativo, avancen 3 páginas.

En caso negativo, ¡no avancen y traten de resolverlos antes de ver las respuestas!

Página dejada intencionalmente en blanco

Página dejada intencionalmente en blanco

Página dejada intencionalmente en blanco

EJERCICIO 1

Ejemplo 1:

```
programa DiasYTiempoDeLaSemana

inicio
    funcion principal()
        var integer dia
        mostrar: "ingrese el dia a consultar (lunes = 1, Martes = 2 y asi sucesivamente)"
        ingresar: dia

        elTiempoEs(dia)
    fin funcion

    funcion elTiempoEs(integer consulta)
        var string diasYTiempoDeLaSemana [7]

        diasYTiempoDeLaSemana [1] = "Lunes, soleado"
        diasYTiempoDeLaSemana [2] = "Martes, soleado"
        diasYTiempoDeLaSemana [3] = "Miércoles, nublado"
        diasYTiempoDeLaSemana [4] = "Jueves, lluvioso"
        diasYTiempoDeLaSemana [5] = "Viernes, lluvioso"
        diasYTiempoDeLaSemana [6] = "Sábado, soleado"
        diasYTiempoDeLaSemana [7] = "Domingo, nublado"

        si consulta > 0 Y consulta < 8 entonces
            mostrar: diasYTiempoDeLaSemana[consulta]
        sino
            mostrar: "error"
        fin si
    fin funcion
fin
```

Ejemplo 2:

```
programa DiasDeLaSemana
inicio
    var integer dia
    var string diasDeLaSemana [7]

    funcion principal()
        asignaciones()
        ingresarNumeroDeDia()
        validacion()
    fin funcion

    funcion asignaciones()
        diasDeLaSemana [1] = "Ese día corresponde a lunes"
        diasDeLaSemana [2] = "Ese día corresponde a martes"
        diasDeLaSemana [3] = "Ese día corresponde a miércoles"
        diasDeLaSemana [4] = "Ese día corresponde a jueves"
        diasDeLaSemana [5] = "Ese día corresponde a viernes"
        diasDeLaSemana [6] = "Ese día corresponde a sábado"
        diasDeLaSemana [7] = "Ese día corresponde a domingo"
    fin funcion

    funcion ingresarNumeroDeDia()
        mostrar: "Ingrese día de la semana, usando el rango de 1 a 7"
        ingresar: dia
    fin funcion

    funcion validacion()
        si dia > 0 Y dia < 8 entonces
            mostrar: diasDeLaSemana [dia]
        sino
            mostrar: "Día incorrecto, por favor reintente"
            ingresarNumeroDeDia()
        fin si
    fin funcion
fin
```

Ejemplo 3:

```
programa DiasDeLaSemanaModificado
inicio
    funcion principal ()
        var integer dia
        var string diasDeLaSemana [7]

        diasDeLaSemana [1] = "Es Lunes"
        diasDeLaSemana [2] = "Es Martes"
        diasDeLaSemana [3] = "Es Miercoles"
        diasDeLaSemana [4] = "Es Jueves"
        diasDeLaSemana [5] = "Es Viernes"
        diasDeLaSemana [6] = "Es Sabado"
        diasDeLaSemana [7] = "Es Domingo"

        cartelDias()

        dia = ingresarDia()

        mostrarDia(dia, diasDeLaSemana[])

    fin funcion

    funcion cartelDias ()
        mostrar: "1 = Lunes"
        mostrar: "2 = Martes"
        mostrar: "3 = Miercoles"
        mostrar: "4 = Jueves"
        mostrar: "5 = Viernes"
        mostrar: "6 = Sabado"
        mostrar: "7 = Domingo"
    fin funcion

    funcion integer ingresarDia ()
        var integer diaElegido

        mostrar: "Ingrese el dia"
        ingresar: diaElegido

        retornar: diaElegido
    fin funcion

    funcion mostrarDia(integer diaRecibido, string diaMostrado[])
        si diaRecibido >= 1 Y diaRecibido <= 7 entonces
            mostrar: "Tu dia " + diaMostrado [diaRecibido]
        sino
            mostrar: "Tu opcion es incorrecta"
        fin si
    fin funcion
fin
```

Ejemplo 4:

```
programa DiasDeLaSemanaConFunciones
inicio
    funcion principal()
        cargaDeDatos()
    fin funcion

    funcion cargaDeDatos()
        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

        operaciones(dia, diasDeLaSemana[])

    fin funcion

    funcion operaciones(integer d, string ds[])
        si d > 0 Y d < 8 entonces
            mostrar: ds [d]
        sino
            mostrar: "Día ingresado incorrecto"
        fin si
    fin funcion
fin
```

Ejemplo 5:

```
programa DiasDeLaSemana
inicio
   funcion principal ()
        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

        en caso de dia hacer
            caso > 0 Y <= 5
                trabajaOchoHoras ()
            fin caso
            caso = 6
                trabajaSeisHoras ()
            fin caso
            caso = 7
                noTrabaja ()
            fin caso
            sino
                mostrar: "opción invalida, vuelva a ingresar el día"
                principal ()   //llamada recursiva, se reinicia el programa
        fin en caso de

    fin funcion

    funcion trabajaOchoHoras ()
        mostrar: "usted trabaja de 7 a 15 hs"
    fin funcion

   funcion trabajaSeisHoras ()
        mostrar: "usted trabaja de 7 a 13 hs"
    fin funcion

   funcion noTrabaja ()
        mostrar: "usted no trabaja, buen descanso"
    fin funcion
fin
```

Ejemplo 6:

```
programa Dias
inicio
    funcion principal()
        var string diasDeLaSemana [7]
        diasDeLaSemana [1]="Lunes"
        diasDeLaSemana [2]="Martes"
        diasDeLaSemana [3]="Miercoles"
        diasDeLaSemana [4]="Jueves"
        diasDeLaSemana [5]="Viernes"
        diasDeLaSemana [6]="Sabado"
        diasDeLaSemana [7]="Domingo"
        var integer dia

        dia = ingresarDia()

        mostrarDia(dia, diasDeLaSemana[])
    fin funcion


    funcion integer ingresarDia()
        var integer d=0

        mientras d <= 0 O d > 7 hacer
            mostrar: "Ingrese dia de lunes a domingo (Teniendo en cuenta que 1=lunes , 7=Domingo)"
            ingresar: d
        fin mientras

        retornar: d
    fin funcion

    funcion mostrarDia(integer dia, string diasDeLaSemana[])
        mostrar: "El dia ingresado es " + diasDeLaSemana[dia]
    fin funcion

fin
```

Ejemplo 7:

```
programa LaSemanaYSusDias
incio
    var integer dia

    funcion principal()
        diaDeLaSemana()
        validacion()
    fin funcion

    funcion diaDeLaSemana()
        mostrar: "Dígame un dia de la semana"
        ingresar: dia
    fin funcion

    funcion validacion ()
        var string diasDeLaSemana [7]
        diasDeLaSemana [1] = "Lunes"
        diasDeLaSemana [2] = "Martes"
        diasDeLaSemana [3] = "Miercoles"
        diasDeLaSemana [4] = "Jueves"
        diasDeLaSemana [5] = "Viernes"
        diasDeLaSemana [6] = "Sabado"
        diasDeLaSemana [7] = "Domingo"

        si dia > 0 Y dia <= 7 entonces
            mostrar: diasDeLaSemana [dia]
        sino
            mostrar: "El día ingresado no existe"
        fin si
    fin funcion
fin
```

Ejemplo 8:

```
programa DiaDeLaSemana
inicio
    var integer dia
    var string diasDeSemana [7]

    funcion principal ()

        diaDeSemana [1] = "es el dia Lunes"
        diaDeSemana [2] = "es el dia Martes"
        diaDeSemana [3] = "es el dia Miercoles"
        diaDeSemana [4] = "es el dia Jueves"
        diaDeSemana [5] = "es el dia Viernes"
        diaDeSemana [6] = "es el dia Sabado"
        diaDeSemana [7] = "es el dia Domingo"

        mostrar: "Bienvenido"

        diaPreferidoDeSemana()
        validacion()

        mostrar: "Su dia preferido " + diaDeSemana[dia]
    fin funcion

    funcion diaPreferidoDeSemana ()
        mostrar:"Ingrese su dia preferido de la semana"
        mostrar: "Valores validos del 1 al 7, empezando con 1 para el Lunes"
        ingresar: dia
    fin funcion

    funcion validacion ()
        si dia > 0 Y dia < 8 entonces
            mostrar: "el numero ingresado es correcto"
        sino
            mostrar: "el numero ingresado es incorrecto- Reintente otra vez"
            diaPreferidoDeSemana()
        fin si
    fin funcion
fin
```

Ejemplo 9:

```
programa DiasDeLaSemana
incio

    var integer dia

    funcion principal()
        elegirDiaDeLaSemana()
        validarDia()
    fin funcion

    funcion elegirDiaDeLaSemana()
        mostrar: "ingrese dia de la semana"
        ingresar: dia
    fin funcion

    funcion validarDia()
        var string diasDeLaSemana [7]
        diasDeLaSemana [1] = "Es lunes"
        diasDeLaSemana [2] = "Es martes"
        diasDeLaSemana [3] = "Es miercoles"
        diasDeLaSemana [4] = "Es jueves"
        diasDeLaSemana [5] = "Es viernes"
        diasDeLaSemana [6] = "Es sabado"
        diasDeLaSemana [7] = "Es domingo"

        si dia > 0 Y dia < 8 entonces
            mostrar: diasDeLaSemana [dia]
        sino
            mostrar: "Dia incorrecto"
        fin si

    fin funcion

fin
```

Ejemplo 10:

```
programa DiasSemana
inicio
    funcion principal ()
        var integer dia
        mostrar: "Bienvenido, ingrese día de la semana"
        ingresar: dia

        si dia >= 1 Y dia <= 7 entonces
            determinarDia (dia)
        sino
            validar ()
        fin si
    fin funcion

    funcion determinarDia (integer dia)
        var string diaDeLaSemana [7]
        diaDeLaSemana [1] = "Es lunes"
        diaDeLaSemana [2] = "Es martes"
        diaDeLaSemana [3] = "Es miércoles"
        diaDeLaSemana [4] = "Es jueves"
        diaDeLaSemana [5] = "Es viernes"
        diaDeLaSemana [6] = "Es sábado"
        diaDeLaSemana [7] = "Es domingo"

        mostrar: diaDeLaSemana[dia]
    fin funcion

    funcion validar ()
        mostrar: "Error!"
    fin funcion
fin
```

Ejemplo 11:

```
programa Semana
inicio
    funcion principal()
        var integer dia
        var string diasDeLaSemana[5]
        var string diasDeFinDeSemana [2]

        diasDeLaSemana [1] =  "Es Lunes"
        diasDeLaSemana [2] = "Es Martes"
        diasDeLaSemana [3] = "Es Miercoles"
        diasDeLaSemana [4] = "Es Jueves"
        diasDeLaSemana [5] = "Es Viernes"
        diasDeFinLaSemana [1] = "Es Sabado"
        diasDeFinLaSemana [2] = "Es Domingo"

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

        elegirDia(diasDeLaSemana[], diasDeFinDeSemana[], dia)

    fin funcion

    funcion elegirDia(string diasDeLaSemana[], string diasDeFinDeSemana[], integer diaElegido)
        si diaElegido > 0 Y diaElegido < 6 entonces
            mostrar: diasDeLaSemana [diaElegido]
        sino si diaElegido = 6 entonces
            mostrar: diasDeFinDeSemana [1]
        sino si diaElegido = 7 entonces
            mostrar: diasDeFinDeSemana [2]
        sino
            mostrar: "Error!"
        fin si
    fin funcion
fin
```

Ejemplo 12:

```
programa Dias
inicio
    var integer dia

    funcion principal()
        mostrar: "Ingrese número del dia de la semana"
        ingresar: dia
        diasSemana()
    fin funcion

    funcion diasSemana()
        var string diasDeLaSemana [7]
        diasDeLaSemana [1] = "Lunes"
        diasDeLaSemana [2] = "Martes"
        diasDeLaSemana [3] = "Miercoles"
        diasDeLaSemana [4] = "Jueves"
        diasDeLaSemana [5] = "Viernes"
        diasDeLaSemana [6] = "Sabado"
        diasDeLaSemana [7] = "Domingo"

        si dia < 1 O dia > 7 entonces
            mostrar: "Error, ingrese un valor correcto"
            principal()
        sino
            mostrar: "Hoy es " + diasDeLaSemana [dia]
        fin si
    fin funcion
fin
```

EJERCICIO 2

Ejemplo 1:

```
programa SueldoDiario
inicio

    var string nombreCompleto = ""

    funcion principal()

        var integer ventasDiaria = 0

        empleado()
        mostrar: nombreCompleto

        mostrar: "Por favor Ingrese ventas de hoy"
        ingresar: ventasDiaria

        //parámetros que se envían: variable, valor fijo, valor fijo, variable
        calculoSueldo("en el dia de hoy", 2000, ventasDiaria)
    fin funcion

    funcion empleado ()
        var string nombre = ""
        var string apellido = ""

        mostrar:"Ingrese su nombre por favor"
        ingresar: nombre
        mostrar: "Ingrese su apellido por favor"
        ingresar: apellido

        nombreCompleto = nombre + " " + apellido
    fin funcion

    funcion calculoSueldo(string dia, integer sueldoBasicoDiario, integer cantVendida)
        mostrar: "El sueldo para " + nombreCompleto + " " + dia + "es de " + sueldoBasicoDiario + "considerando " + cantVendida + "ventas"
    fin funcion
fin
```

Ejemplo 2:

```
programa AdivineElNumeroYColor
inicio
    const string colorCorrecto = "Verde"
    const integer numeroCorrecto = 6

    funcion principal()
        var integer numero = 0
        var string color = ""
        const string error = "Lo sentimos, seguí participando"

        mostrar: "Ingrese un número del 1 al 10"
        ingresar : numero
        mostrar: "Ingrese un color"
        ingresar: color

        //parámetros que se envían: valor fijo, valor fijo, variable, variable
        rdoAdivinanza ("Adivinaste! Felicitaciones:", error, numero, color)
    fin funcion

    funcion rdoAdivinanza (string mensajeOK, string mensajeNOK, integer numero, string color)
        si numero = numeroCorrecto Y color = colorCorrecto entonces
            mostrar: mensajeOK
        sino
            mostrar: mensajeNOK
        fin si
    fin funcion
fin
```

Ejemplo 3:

```
programa Calificaciones
inicio
    funcion principal ()
        var string alumno
        var integer dni
        var integer nota

        mostrar: "Ingrese el nombre del alumno"
        ingresar: alumno
        mostrar: "Ingrese el dni del alumno"
        ingresar : dni
        mostrar: "Ingrese nota"
        ingresar: nota

        //parámetros que se envían: variable, variable, variable, valor fijo
        aprobacion(alumno, dni, nota, " tiene una nota de ")

    fin funcion

    funcion aprobacion (string a, integer d, integer n, string m)
        const integer notaMinima = 6

        si n >= notaMinima entonces
            mostrar: a + " con dni nro " + d + m + n + " y esta aprobado"
            //ejemplo: "carlos con dni nro 21020202 tiene una nota de 8 y esta aprobado"
            //donde "carlos" es el valor del parámetro "a", "21020202" el param "d", etc
        sino
            mostrar: a + " con dni nro " + d + m + n + " y esta desaprobado"
        fin si
    fin funcion
fin
```

Ejemplo 4:

```
programa SorteoBandasFavoritas
inicio
    funcion principal()
        const integer premio = 10000
        var string nombre

        mostrar: "Bienvenido! ¿como te llamas?"
        ingresar: nombre

        //parámetros que se envían: variable, constante, valor fijo, valor fijo
        bandasFavoritas (nombre, premio, "Gracias por participar", "recordá que el sorteo es el 20/3/2020")
    fin funcion

    funcion bandasFavoritas(string participante, integer dinero, string despedida, string recuerda)
        mostrar: participante + "Estás participando por" + dinero + "pesos" + despedida
        mostrar: recuerda
    fin funcion
fin
```

Ejemplo 5:

```
programa MenuDelDia
inicio
    funcion principal ()
        var float cantidadMenu
        const string menuDelDia = " milanesa a la napolitana "
        const float precioMenu = 199,99

        mostrar: "Por favor, ingrese la cantidad de menús que desea agregar:"
        ingresar: cantidadMenu

        //parámetros que se envían: variable, valor fijo, variable, constante
        pedido (cantidadMenu, " menú(s) de ", menuDelDia, precioMenu)
    fin funcion

    funcion pedido (float cant, string msjMenu, string menuDia, float precio)
        mostrar: "Recibimos su pedido de " + cant + msjMenu + menuDia + ". El costo de su pedido es de " + (cant * precio)
    fin funcion
fin
```

Ejemplo 6:

```
programa DatosPropios
inicio

    funcion principal()
        const integer dni = 40685280
        const integer anios = 54

        //parámetros que se envían: valor fijo, valor fijo, constante, variable
        mostrarDatos("Ramon", "Diaz", dni, anios)
    fin funcion

    funcion mostrarDatos(string nombre, string apellido, integer dni, integer anios)
        mostrar: "Mi nombre es " + nombre
        mostrar: "Mi apellido es " + apellido
        mostrar: "Mi dni es " + dni
        mostrar: "Mi edad es " + anios + " años"
    fin funcion
fin
```

Ejemplo 7:

```
programa MixFrutosSecos
inicio
    funcion principal ()
        const string MIXFRUTOSECO = "running"
        var string nombreDeFrutoSecoUtilizado [5]
        nombreDeFrutoSecoUtilizado [1]= "chip de banana"
        nombreDeFrutoSecoUtilizado [2]= "mani sin sal"
        nombreDeFrutoSecoUtilizado [3]= "nueces"
        nombreDeFrutoSecoUtilizado [4]= "almendras"
        nombreDeFrutoSecoUtilizado [5]= "avellanas"

        //parámetros que se envían: valor fijo, variable (vector), constante, valor fijo
        armadoDeMix (100, nombreDeFrutoSecoUtilizado[], MIXFRUTOSECO, " mix ")
    fin funcion

    funcion armadoDeMix(integer cantidad, string nombreDeFrutoSecoUtilizado[], string mix, string tipo)
        var integer i = 1

        mostrar: "Este es un " + tipo + " llamado " + mix + " que tiene un peso de " + cantidad + " y contiene: "

        mientras i < 6 entonces
            mostrar: nombreDeFrutoSecoUtilizado[i]
            i= i + 1
        fin mientras
    fin funcion
fin
```

Ejemplo 8:

```
programa CargaDeNafta
inicio

    funcion principal ()
        var integer litros
        var string nafta

        mostrar: "Bienvenido a YPA, ¿Cuantos litros desea cargar?, "
        ingresar: litros
        mostrar: "¿Desea cargar super o premium?"
        ingresar: nafta

        //parámetros que se envían: valor fijo, variable, variable, valor fijo
        calculo("20/03/2020", litros, nafta, 79,44)

    fin funcion

    funcion calculo(date fecha, integer litros, string nafta, float dolar)
        var integer precio
        var integer precioDolar

        const integer premium = 71
        const integer super = 61

        si nafta = "premium" entonces
            mostrar: "El valor del dia (" + fecha + ") de la nafta premium es " + "$" + premium
            precio = premium * litros
            precioDolar = precio/dolar
            mostrar: "El valor a abonar es " + "$" + precio + " o " + "US$" + precioDolar
        sino si nafta = "super" entonces
            mostrar: "El valor del dia (" + fecha + ") de la nafta super es " + "$" + super
            precio = super * litros
            precioDolar = precio/dolar
            mostrar: "El valor a abonar es " + "$" + precio + " o " + "US$" + precioDolar
        sino
            mostrar: "Tipo de nafta invalido"
        fin si
     fin funcion
fin
```

ANEXO 2 - Checklist de código

Algunas consideraciones a la hora de revisar su propio código que se suman al checklist de la unidad anterior:

	Cumple	
Tema	**Si**	**No**
Las matrices declaradas tienen más de 1 columna		
En los arreglos se está guardando (asignando/ingresando) valores de un único tipo de dato que coincide con el tipo de dato de la declaración del arreglo		
Todos los índices de los arreglos son números		
Todos los arreglos se empiezan a recorrer/cargar desde la posición 1		
Cuando se envía un arreglo por parámetros se coloca el nombre y los corchetes vacíos		
En la declaración de una función, cuando se recibe un parámetros que es un arreglo, se el tipo de dato y el nombre y los corchetes vacíos		
Cuando se quiere agregar o mostrar el valor de un arreglo, se están colocando los índices correspondientes. Por ejemplo: mostrar: vector[i] Y no: mostrar: vector[ ]		
Las declaraciones de los vectores cuentan con el número de posiciones entre corchetes, que es un número entero (y no una variable)		
Las declaraciones de las matrices cuenta con el número de filas y columnas entre corchetes, y ambos son números enteros (y no variables)		
Todos los vectores van acompañados de un par de corchetes		
Todas las matrices van acompañadas de dos pares de corchetes		
Todas las declaraciones de arreglos incluyen la palabra reservada "var" o "const", el tipo de dato del arreglo, el nombre del arreglo y sus dimensiones (cantidad de posiciones si es un vector, cantidad de filas y columnas en caso de una matriz)		
Todos los programas tienen una función "principal"		
Todas las funciones llevan un par de paréntesis, incluso las invocaciones a las funciones no incluyen parámetros		
Todas las funciones llevan un par de paréntesis, incluso las declaraciones de las funciones no incluyen parámetros		
Ninguna variable global se está enviando como parámetro		
Ídem constantes		

No hay variables locales que se llamen igual que una variable global		
Ídem constantes		
No hay parámetros de funciones que se llamen igual que una variable global		
Ídem constantes		
No hay parámetros de funciones que se llamen igual que una variable local		
Ídem constantes		
No hay dos variables locales (en una misma función) con el mismo nombre		
No hay dos variables globales con el mismo nombre		
En la invocación a una función que declara que recibe parámetros, se agregaron variables, constantes y/o valores fijos para dichos parámetros		
La función "principal" no recibe parámetros		
La función "principal" invoca, al menos, una función más		
La función "principal" no devuelve valores		
Todas las funciones que declaran devolver un valor tiene un "retornar:" adentro		
Todas las funciones que tienen un "retornar:" declaran devolver un valor		
Se verifica que no hay código "ejecutable" en una función luego de un "retornar:"		
Todos los parámetros tiene su tipo de dato (en la declaración)		
Se verifica que en las invocaciones NO se colocaron los tipos de datos		
SE VERIFICA QUE LAS INVOCACION **NO** LLEVAN LA PALABRA RESERVADA "funcion"		
Todas la declaraciones de funciones están compuestas por: palabra reservada "funcion" (obligatorio), el tipo de dato del valor que devuelve (opcional), el nombre de la función (obligatorio), paréntesis (obligatorio), listado de parámetros compuesto por el tipo de datos y el nombre del parámetro (obligatorio)		
Se verifica que todas las variables y constantes usadas en una función fueron declaradas EN la función (locales) o FUERA de ella (global), pero sí o sí fueron declaradas		
Todas las funciones tienen un nombre significativo, claro, que indican qué acción realizan		
Se verifica que no hay variables locales de una función que se usen en otra función		
Se verifica que NO HAY CÓDIGO fuera de las funciones, salvo declaraciones de variables globales		
Toda función que recibe un parámetro USA su valor antes de modificarlo		

Si todas las respuestas son afirmativas, hay muy altas chances que el programa esté libre de errores de código.

ANEXO 3 - Preguntas frecuentes

1) P: ¿A qué se hace referencia en las actividades cuando se habla de "valores fijos"?

R: Los valores fijos son valores de variables o constantes que define el programador. Por ejemplo:

invocandoFuncion("hola", 123)

Donde:

- **"hola" es un valor fijo tipo string (por las comillas dobles)**
- **123 es un valor fijo numérico tipo integer (no tiene comillas ni coma)**

2) P: ¿Cómo diferenciar un valor fijo del nombre de una variable o arreglo o función?

R: Las invocaciones a funciones siempre llevan "()", de esta forma si encontramos un par de paréntesis, vamos a saber que se trata de una función.

De la misma manera, los arreglos siempre llevan "[ ]" (vector) o "[ ][ ]" (matriz).

Con respecto a confundir un valor fijo con una variable, podríamos pensar:

- **Si dice "verdadero" o "falso" no son variables, sino valores booleanos. Por ejemplo:**

  var boolean pasoPorAca = verdadero

  si esMayor = falso entonces...

- **si hay un número, no es una variable, sino un valor fijo, porque las variables no puede tener números por nombre ni empezar con números. Por ejemplo, nombres incorrecto de variables:**

1 = "uno" //1 es incorrecto como nombre de variable

1piso = "primer piso" //1piso es incorrecto como nombre de variable

- **Si hay una expresión alfanumérica entre comillas dobles, es un valor fijo tipo string. Recordar: los nombres de las variables, constantes, arreglos y funciones NUNCA van entre comillas.**
- **Si hay una expresión alfanumérica sin comillas, se trata del nombre de una variable**

3) P: ¿Una variable, si es declarada fuera de cualquier función pero iniciada dentro de alguna función, sigue siendo una variable global?

R: Sí, porque es global o local dependiendo del lugar dónde fue declarada, no del qué lugar es usada (leída o modificada).

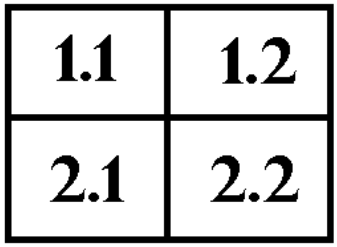
4) P: ¿Cuáles son las palabras reservadas vistas hasta ahora?

R: Repasemos:

- **Unidad 4: programa, inicio, fin, ingresar: , mostrar:**
- **Unidad 5 (se agregan): var, const, integer, string, float, double, boolean, si, entonces, sino, fin si, sino si, en caso de, caso, fin caso, fin en caso de, mientras, fin mientras, hacer, verdadero, falso**
- **Unidad 6 (se agregan): funcion, principal, fin funcion, retornar:**

5) P: ¿Cuándo se crea una matriz: cómo se le otorga un valor a cada "posición"?

R: Una matriz de forma de pensar figurativamente esta estructura sería a través de un cuadro doble entrada, donde cada posición tiene una "dirección" única dada por el número de fila y número de columna. Por ejemplo, una representación de una matriz de 2x2 podría verse como:



Donde el primer número podría indicar la fila y el segundo (luego del punto) podría indicar la columna. De esta forma, pudieron identificar unívocamente cada posición, podríamos realizar operaciones para agregar o modificar los datos que contienen. Por ejemplo:

Al hacer

ingresar: matrizFoda [1][1]

Se está agregando un valor en ese primer "casillero", identificado por la fila 1 y columna 1.

Para hacer una operaciones de asignación, se podría hacer:

matrizFoda[1][2] = "El valor que se le quiera dar"

//o un número u otro dato, dependiendo del tipo de la matriz

En este caso, se está asignando un string en la fila 1 y columna 2.

6) P: ¿Las funciones son términos equivalentes a las subrutinas en algunos lenguajes?

R: Subrutinas, procedimientos, funciones... diferentes nombres en diferentes lenguajes para la misma idea (agrupar código reutilizable), aunque hay que tener en cuenta que algunos lenguajes cuentan con estos conceptos y los diferencian entre sí dependiendo del tipo de su uso.

7) P: En el c aso de querer hacer una modificación de posiciones de una matriz ¿Es válido mover los datos que tengo a otra posición al modificar los índices?. Por ejemplo, para hacer un cambio de filas entre las posiciones [1][5] y [3][5].

R: No, siempre hay que "mover" el contenido. Los índice no se pueden modificar, son estáticos.

8) P: ¿Cuándo se muestra el ejemplo de que haciendo un ciclo repetitivo con condición "i<10" se dice que se recorren solo las filas. ¿Cómo sabe el programa que "i" es para las filas?

R: Esto aplica para las matrices: los índices son pares ordenados de "fila + columna", por lo tanto al ubicar a "i" en el primer par de corchetes, se está indicando que esa es la fila y no es necesario hacer más aclaraciones.

9) P: ¿A qué se refiere el concepto de "función propia"?

R: Es cualquier función definida por el programador, salvo la "principal" que se puede considerar una función del lenguaje (del pseudocódigo).

10) P: ¿Se puede tener una matriz que cada dato sea un arreglo? Por ejemplo una matriz de vectores.

R: Algunos lenguajes lo pueden permitir, pero no es buena gran idea, porque ese mismo lenguaje seguramente ofrezca un mecanismo alternativo un poco más sencillo.

En lo que hace a este curso, para ser coherentes con el pseudocódigo presentado, no lo vamos a permitir, porque un vector solo se puede asignar a otro vector, y en nuestro cada la cada posición de la matriz es un dato simple, no un vector. Por ejemplo:

```
var string listado[3]

listado[1] = ...        //permite guardar un string, no un vector
```

11) P: ¿Cuando se recibe un valor retornado de un función y dicho valor es un arreglo, como se puede mostrar en la función que recibe dicho arreglo?

```
funcion principal()
    var integer valorx
    valorRetornado()
    mostrar: valorRetornado() // A- puedo hacer esto ???
    mostrar: miArr[1]      //B- Y esto si se puede hacer??
    valor x = miArr[2]    // C- Es válido esto?
fin funcion
funcion integer[] valorRetornado()
    var integer miArr[3]
    miArr[1] = 100
    miArr[2] = 200
    miArr[3] = 300
    retornar: miArr[]
fin funcion
```

R: Respuesta a las 3 preguntas en los comentarios de código:

A)

mostrar: valorRetornado() // 1- puedo hacer esto ???

No, no es posible ya que para mostrar el contenido de un vector hay que hacer referencia a alguna de las posiciones.

B)

mostrar: miArr[1] //2- Y esto si se puede hacer??

No, no es posible ya que ese vector es local de la otra función

C)

valor x = miArr[2] // 3- Es válido esto?

No, no es posible ya que ese vector es local de la otra función

En cambio, el ejemplo corregido podría ser algo así:

```
funcion prinicipal()
        var integer vecLocal[3]

        vecLocal[] = valorRetornado()    //guardo vector retorno en vector local

        mostrar: vecLocal[1]      //muestra 100
        mostrar: vecLocal[2]      //muestra 200
        mostrar: vecLocal[3]      //muestra 300
fin funcion

funcion integer[] valorRetornado()
        var integer miArr[3]
        miArr[1] = 100
        miArr[2] = 200
        miArr[3] = 300
        retornar: miArr[]
fin funcion
```

12) P: ¿Una función que no es la principal: puede enviar o compartir datos con otras funciones que tampoco sean la principal?

R: Cualquier función puede enviar sus datos locales (variables, constantes, valores fijos) como parámetro a cualquier otra función, sea la "principal" o cualquier otra función propia.

13) P: ¿Dentro de una función se pueden usar los condicionales tipo "caso"?.

R: Si, todas las estructuras presentadas en la unidad 5 se puede pueden usar en cualquier función. En el particular del "caso", siempre deben estar dentro de una estructura "en caso de".

14) P: ¿Se puede usar como una condición el retorno de la función? Por ejemplo:

si funcionUno()=0 entonces

¿O, por el contrario, sí o sí hay q declarar una variable y después asignarle el valor de la función para después usarla en la condición?

R: Si, es perfectamente válido y no es obligatorio declarar la variable y hacer la asignación y luego usar esa variable en la condición, aunque también sería válido y es la opción más clara.

Ahora bien, esa expresión va ser válida siempre y cuando exista una función llamada "funcionUno" que no recibe parámetros y devuelve un integer.

15) P: ¿El usuario puede asignar valores a variables globales?

R: Si, puede. El uso de las variables es indistinto si son locales o globales.

16) P: ¿Sería correcto afirmar lo siguiente?

"Una función puede RECIBIR la cantidad de parámetros que necesite, pero... una función sólo puede DEVOLVER un dato."

R: 1) Si, es correcto. Por "puede" debe quedar claro que es posible no recibir ningún parámetro ni devolver ningún valor. Resumiendo:

- **Parámetros de una función: desde 0 a N**
- **Valore retornados de una función: 0 o 1**

17) P: Cuando se menciona que "Es mucho más "performante" utilizar variables locales, dado que eso implica que la memoria de la máquina se usa y libera constantemente", ¿a qué se entiende por "performante"?

P: Es una mala traducción o traducción literal de la palabra inglesa "performance". Se podría definir como el atributo que implica "usar menos recursos", que es "más eficiente", que "funciona más rápido". Equivalente a algo "óptimo" u "optimizado".

Lo que vimos

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

Lo que viene:

- Integración de conceptos