UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

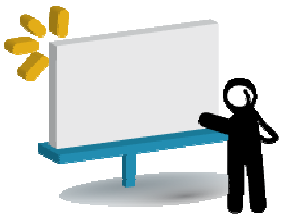
FUNDAMENTOS DE LA PROGRAMACIÓN

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

Variables 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.

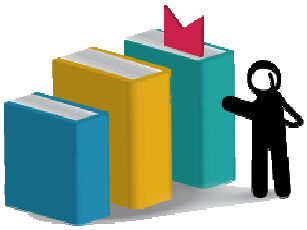
Todos estos temas son progresiones de los temas vistos anteriormente y representan, también, los contenidos sobre los cuales ser irán agregando más complejidad y nuevos temas complementarios.

Objetivos:

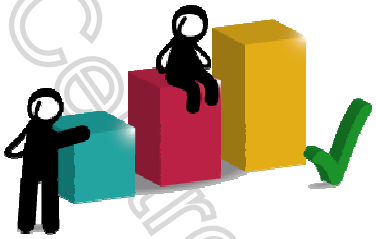
Que los participantes:

- Comprendan el uso y funcionalidad de las variables
- Comprendan el uso y funcionalidad de las constantes
- Incorporen el uso de tipos de datos para variables y constantes
- Comprendan los conceptos y uso de los vectores y matrices
- Incorporen el uso de funciones y conocer los tipos

Bloques temáticos:



1. Variables
    - 1.1 Definición
    - 1.2 Convenciones de Nomenclatura
    - 1.3 Constantes
    - 1.4 Ejercicio resuelto
2. Tipos de datos
    - 2.1 Definición
    - 2.2 Tipos
    - 2.3 Declaración de variables
3. Arreglos
    - 3.1 Definición
    - 3.2 Arreglos unidimensionales: vectores
    - 3.3 Arreglos bidimensionales: matrices
    - 3.4 Ejercicio resuelto
4. Funciones
    - 4.1 Definición
    - 4.2 Funciones con parámetros
    - 4.3 Devolución de valores



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:

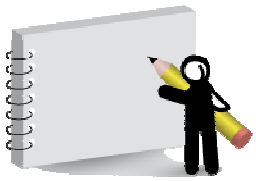
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

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.

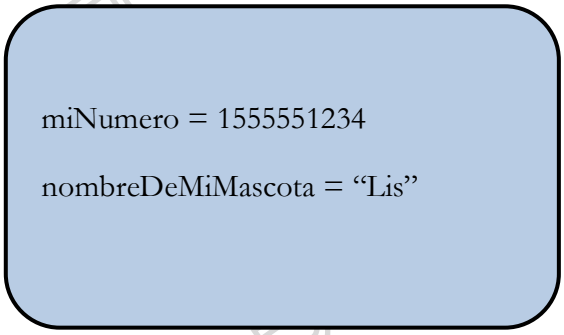
Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de 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. Variables

1.1 Definición

Si bien las venimos nombrando y utilizando informalmente, es necesario adentrarnos algo más en el concepto de “variable”, dado que su uso y convenciones así lo requieren.

Si metaforizáramos sobre el uso de las variables, diríamos que son distintos **recipientes donde colocamos distintos tipos de elementos**. Es eso lo que venimos haciendo hasta este momento cuando decimos que asignamos un valor a una variable, por ejemplo:



Pero no todos los recipientes son iguales, eso está claro. Imaginemos que tenemos diferente recipientes, por ejemplo:

- un tubo de ensayo
- una bolsa
- una canasta

Por otro lado, tenemos que llenarlos con estos contenidos:

- un puñado de clavos
- un líquido
- pan

Si bien físicamente sería posible colorar el pan en el tubo de ensayo, el líquido en la bolsa y los clavos en la canasta, nos deberíamos preguntar: ¿Es lo mismo cualquier elemento en cualquier recipiente? ¿Conceptualmente, tiene sentido?

Lo que está claro es que si existe un tipo definido de recipiente para un tipo definido de contenido, el sentido común nos lleva a vincular uno con otro, según corresponda. Lo mismo ocurre con las variables: sus "tipos" (que diferenciaremos más adelante en esta misma Unidad) son heterogéneos y su uso debe estar ligado a su razón de ser. **No es lo mismo (más allá que algún lenguaje lo permita, siendo esto conceptualmente dudoso) asignar un número a un tipo de variable preparado para contener caracteres.**

Para esto, vamos a realizar lo que se conoce como **declaraciones de variables**. Esto significa que explícitamente vamos a crear una variable diciendo qué tipo de datos va a poder contener. De esta forma, cuando programemos una rutina en un lenguaje en particular, le estarán diciendo al compilador (ver tema Compiladores del Módulo 1) para qué vamos a utilizar esa variable. Y el compilador será capaz de advertirnos si ese uso es correcto o no, ya que va a verificar si realmente le estamos asignando los valores del tipo que espera recibir.

1.2 Convenciones de Nomenclatura

Por regla general, todos los lenguajes de programación sugieren realizar las declaraciones y asignaciones en la parte superior e inicial del programa. Esto aporta limpieza y claridad al código, permitiendo ver de un vistazo cuáles son las variables que serán utilizadas en un programa, aunque posteriormente éstas serán manipuladas (modificadas, vaciadas, consultadas…) en otro punto del programa.

Sobre las variables anteriormente dijimos que no existe un estándar ni un único criterio unificado para nombrarlas. Aunque en este curso establecimos como regla general el uso del método llamado lowerCamelCase, el cual respetaremos de ahora en adelante.

Por otro lado, en los lenguajes de programación modernos encontramos (como regla genérica, pero no absoluta) que las asignaciones de valores numéricos se realizan igualando una variable con el valor (por eso **miNumero = 1555551234**), a diferencia de un valor alfanumérico (letras, números, caracteres especiales), cuya asignación se realiza igualando la variable al valor entre comillas (por eso **nombreDeMiMascota = "Lis"**).

Para el tratamiento de fechas, cada lenguaje suele implementar su forma propia de asignación, aunque a fines prácticos en este curso lo utilizaremos con la misma convención que usamos con los caracteres: el valor va a ir entre comillas dobles (""). Siempre evitaremos incluir en el nombre de las variables caracteres especiales (que no sean las letras del abecedario inglés). **Esto incluye las letras "ñ" y (especialmente) las tildes en las letras acentuadas.** Más allá de que algunos compiladores soporten estos caracteres, no resulta una buena práctica de programación. Es muy común encontrar variables como "año" o "número", por lo que deben ser cuidadosamente descartadas.

Finalmente, una recomendación importante: las variables deben tener nombres significativos. Es muy importante evitar el uso de variables del tipo "var1" o "pepe", por más extraño que parezca. Las variables siempre (SIEMPRE) deben tener un nombre que se corresponda con lo que vayan a contener. Si tenemos fechas y no sabemos a qué fechas nos estaremos refiriendo, simplemente tendremos una variable llamada "fecha". Si tenemos un rango de fechas, tendremos "fechaInicial" y "fechaFinal". Si tenemos una fecha de cumpleaños, será "fechaDeNacimiento", etc. En este caso, entra en juego una de las principales habilidades del programador: **el sentido común**.

1.3 Constantes

En contraposición (incluso de conceptos) con respecto a las variables, encontramos ciertos tipos llamado Constantes, dado que se les suele asignar un valor en forma inicial (cuando se declara o en forma posterior) y su valor no suele cambiar a lo largo del programa.

Este tipo de valores estáticos resultan de gran utilidad (por su simpleza y por la claridad que le aportan al código) a la hora de realizar comparaciones y/o ante la necesidad de tener listas de valores que sabemos que no van a cambiar. Por ejemplo: los días de la semana, los meses, marcas de autos, etc.

Como regla general, también se declaran y se les asignan valores al principio de un programa.

Para nombrarlas, siendo concordantes con la mayoría de los lenguajes de programación modernos, utilizaremos MAYUSCULAS, a diferencia del lowerCamelCase definido anteriormente para las variables.

Cabe destacar que en realidad sí es posible cambiar durante la ejecución de un programa (en algunos lenguajes) el valor de una constante, salvo que ésta haya sido declarada de una forma especial, evitando las posibilidades de cambios, aunque modificar el valor representaría una mala práctica de programación. En el caso en que sospechemos que una constante pueda tomar un valor distinto al declarado inicialmente, siempre deberemos declararla como variable y no como constante (regla general: ante la duda, siempre es variable).

1.4 Ejercicio resuelto

```
programa diaDeLaSemana
inicio
        const  LUNES = 1
        const MARTES = 2
        const MIERCOLES = 3
(Sigue...)
```

(continúa...)

```
const JUEVES = 4
const VIERNES = 5
const SABADO = 6
const DOMINGO = 7
const ERROR = "El valor ingresado es incorrecto"
mostrar: "Ingrese número de día de la semana: "
ingresa: dia
en caso de dia hacer
        caso LUNES
                mostrar: "Es LUNES"
        caso MARTES
                mostrar: "Es MARTES"
        caso MIERCOLES
                mostrar: "Es MIERCOLES"
```

(sigue...)

```
(continúa...)
                caso JUEVES
                        mostrar: "Es JUEVES"
                caso VIERNES
                        mostrar: "Es VIERNES"
                caso SABADO
                        mostrar: "Es SABADO"
                caso DOMINGO
                        mostrar: "Es DOMINGO"
                sino
                        mostrar: ERROR
        fin en caso de
fin
```

En este simple ejemplo, vemos cómo inicialmente se declaran 7 constantes, cada una llamada como un día de la semana. A éstos, se les asigna un valor del 1 al 7 para identificarlos. Adicionalmente se declara otra constante llamada ERROR, a la cual se le asigna un mensaje.

Posteriormente se declara una estructura condicional “en caso de”, como se ha visto en la Unidad anterior. Cada uno de los “caso” compara el valor ingresado y guardado en la variable “dia” (que se sigue utilizando como hasta ahora, sin declarar, cosa que veremos en los próximos puntos de la Unidad). En el caso de que el usuario haya ingresado un número entre 1 y 7, éste será detectado por la condición y se mostrará el mensaje que corresponde.

Hacer "caso LUNES", sería lo mismo que hacer "caso 1"; "caso MARTES" sería "caso 2" y así sucesivamente. Pero supongamos un extenso programa en el que cual se usan los días de la semana, digamos, 20 veces. Y un buen día nos llega el pedido para adaptarnos a una nueva regulación por la cual el día "1" de la semana pasa de Lunes a Domingo. Si usáramos los valores (1, 2, 3...) en forma directa, tendríamos que hacer el cambio en 20 lugares diferentes. Si en lugar de usar los valores, usamos CONSTANTES, sólo habría que cambiar la declaración de las constantes, sin tener que modificar el resto del programa.

Si el número ingresado no se encuentra en el rango antes mencionado, se muestra el mensaje de error guardado en la constante ERROR.

ACTIVIDAD 1:

¿Cuáles serían las reglas de nomenclado de variables y constantes de los lenguajes actuales? ¿Cuáles utilizan lowerCamelCase o CamelCase o alguna otra estandarización? ¿Hay lenguajes que no propongan o sugieran la aplicación de ninguna regla?

Investigue y comparta sus respuestas (fundamentadas) en el foro de actividades de la Unidad.

2. Tipos de datos

2.1 Definición

Los tipos de datos, más allá de las extrañas metáforas de clavos y tubos de ensayo, son uno de los componentes más importantes y sensibles de los lenguajes de programación.

Decimos que son importantes porque, a través de la combinación de pequeñas variables, cada una de su tipo, logramos representar en forma abstracta una realidad compleja.

Aunque también son sensibles, ya que es uno de los puntos más susceptibles a fallas y errores por parte del programador. Para explicar esto, antes daremos algunas definiciones breves sobre términos comúnmente usados:

- **Tiempo de programación**: se refiere al momento y las actividades realizadas por el programador durante la etapa de construcción de un programa.
- **Tiempo de compilación**: hace referencia a todas aquellas actividades y pasos que realiza internamente un compilador, cuando un programador ordena la compilación de un programa que se encuentra construyendo.
- **Tiempo de ejecución**: se refiere al uso que le da un usuario a un programa previamente desarrollado y compilado por un programador. Por "usuario" entendemos a cualquier persona que ejecute el programa, pudiendo ser el mismo programador, en caso de estar realizando pruebas, por ejemplo.

Volviendo a la "sensibilidad" de los tipos de datos de las variables, es extremadamente común que una variable que concebimos en tiempo de programación pensando que iba a recibir un número, en tiempo de ejecución finalmente puede terminar recibiendo una letra, con la consecuente falla en el programa, al tratarse de dos tipos de datos incompatibles.

¿Por qué ocurriría esto? En nuestro ejemplo anterior de los días de la semana: ¿Qué ocurriría si el usuario en lugar de ingresar un número del 1 al 7, ingresar los siguiente: "/(uy687f"? La respuesta es simple: al tratar de asignar ese valor a la variable "dia", se produciría una falla en el programa. Una falla que al no estar preparados para capturarla, ocasionaría una interrupción en la ejecución del programa.

Esto es con lo que tenemos que lidiar: el tiempo de programación es un escenario estático, en el que controlamos todo lo que ocurre. Por el contrario, el tiempo de ejecución es un escenario dinámico, donde las posibilidades de multiplican más allá de las variantes que podemos imaginar o prever.

La primera reacción podría ser: "¿Cómo es posible que un usuario sea tan ***** para ingresar "/(uy687f" cuando le estamos pidiendo explícitamente que "Ingrese número de día de la semana"? … La respuesta es que lamentablemente tendremos que lidiar con estos factores externos que condicionan nuestro trabajo de programación. Sean usuarios, sean tiempos o costos, sean máquinas de baja calidad, siempre deberemos tener en cuenta los aspectos contextuales a la hora de encarar un desarrollo de software.

Finalizando, para darle un cierre al marco teórico de los tipos de datos, podemos decir que se trata de definir qué clase de información puede contener una variable.

2.2 Tipos

Dependiendo de la implementación de cada lenguaje de programación en particular, podemos encontrar distintos tipos de datos en cuanto a nombres (tipos de datos "string" y "str", que tienen la misma función) y capacidades (cantidad de números detrás de la coma que soportan dos tipos de datos similares pero de diferentes lenguajes de programación).

La siguiente es una tabla comparativa de los tipos de datos más usados:

Tipo de dato	Tamaño	Descripción
Boolean	1 byte	Valores Sí/No o Verdadero/Falso.
Date	8 bytes	Una fecha entre los años 100 y 9999
Float	4 bytes	Un valor de coma flotante entre –3,402823E38 y –1,401298E-45 para valores negativos, y desde 1,401298E-45 a 3,402823E38 para valores positivos, y 0.
Double	8 bytes	Un valor de coma flotante entre –1,79769313486232E308 y –4,94065645841247E-324 para valores negativos, y desde 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos, y 0.
Integer	4 bytes	Un entero entre – 2.147.483.648 y 2.147.483.647.
String	2 bytes por carácter	Desde 0 a 255 caracteres.

ACTIVIDAD 2:

¿Cuáles son los tipos de datos de lenguajes como Java y C#? ¿Qué relación tiene esto con los lenguajes de programación fuerte o débilmente tipados?

- Evitar copypasteos extensos y sin analizar

Investigue y comparta sus respuestas (fundamentadas) en el foro de actividades de la Unidad.

A fines prácticos, definiremos las siguientes **palabras reservadas** como un subconjunto de tipos de datos válidos para este curso:

- **string** (“cadena” de “cadena de caracteres”): lo usaremos para identificar variables que contienen de 0 a 255 caracteres
- **integer** (“entero” de “número entero”): lo usaremos para representar números enteros (sin coma)
- **float** (“flotante” de “coma flotante”): lo usaremos para representar números con coma
- **date** (“fecha”): lo usaremos para representar fechas
- **boolean** (“booleano” o lógico): lo usaremos para almacenar valores binarios (verdadero o falso), como por ejemplo, el resultado de una condición

Serán una de las pocas palabras en idioma inglés que utilizaremos en nuestro seudocódigo, para evitar confusiones.

2.3 Declaración de variables

Como se dijo anteriormente, realizaremos la declaración de variables y su inicialización (primer valor que se asigna a una variable) al inicio del programa.

Un ejemplo podría ser:

```
programa declaracionDeVariables
inicio
        var date fechaDeNacimiento = "16/11/1978"
        var string nombre
        nombre = "Emilio"
        var integer edad = 33
        var flot altura = 1,77
        var boolean tieneHijos
        tieneHijos = verdadero
fin
```

Anteponemos al tipo de dato la **palabra reservada "var"** para diferenciarlas de las constantes, a las cuales identificaremos con la palabra reservada "**const**". Posteriormente definiremos el tipo de dato, utilizando alguno de los cinco tipos que precisamos hasta ahora en el presente curso: date, string, integer, float y boolean.

Vemos también cómo se puede realizar de dos formas distintas las inicializaciones de las variables:

- Junto con la declaración, como es el caso de fechaDeNacimiento, edad y altura
- Luego de la declaración, como en el caso de nombre o tieneHijos

Finalmente, prestemos atención a la forma en que se realizan las inicializaciones:

- Valores de date y string se van a asignar a las variables utilizando las comillas dobles (“”)
- Valores de float e integer se van a asignar a las variables igualándolas con los números dados, lo mismo que los valores posibles de las variables booleanas (verdadero y falso, sin comillas)

Resumiendo:

- **La variables sirven para almacenar datos en forma temporal**
- **Existen diferentes TIPOS de variables relacionadas con las características de los datos que van a contener**
- **Las constantes sirven para almacenar datos que no van a ser modificados**
- **Tanto variables como constantes deberían definirse al inicio del programa**
- **Tanto variables como constantes deben tener un tipo y los valores que se le asignen deben ser del mismo tipo que fuera declarada la variable o constante**

3. Arreglos

3.1 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.

3.2 Arreglos unidimensionales: vectores

El primer tipo de arreglo que analizaremos son los llamados vectores o arreglos unidimensionales, en cuanto se los representa como una fila casilleros, de 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.

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.

Es posible comparar a cada una de las posiciones 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 es una cadena de bytes, en la cual cada letra de esa cadena se encuentra en una posición 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

Vector de los hobbits de la Comunidad del Anillo:

Frodo	Merry	Pippin	Sam
1	2	3	4

3.3 Arreglos bidimensionales: matrices

Siguiendo con la tónica de agregar complejidad (y posibilidad de uso) a las estructuras de datos, nos encontramos con las matrices. Y poco tienen que ver con sagas de ciencia ficción. Todo lo contrario, 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 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.

3.4 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: diaDeLaSemana [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]
        mostrar: "Ingrese Oportunidades:"
        ingresar: matrizFoda [1] [2]
        mostrar: "Ingrese Debilidades:"
        ingresar: matrizFoda [2] [1]
        mostrar: "Ingrese Amenazas:"
        ingresar: 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.

4. Funciones

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

4.1 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 reuso 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()  //llamado a función
        fin funcion

        funcion otraFuncion ()
                // instrucciones de la funcion otraFuncion
        fin funcion
fin
```

Tendremos siempre una función llamada “principal” que será en todos los casos 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.

4.2 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.

```
programa ejemploDeFuncionConParametros

inicio

        var integer a

        var integer b

        var integer rdo

        (sigue...)
```

¡¡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!

```
(continúa...)
funcion principal ()
        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)
        rdo = primero + segundo
        mostrar: "El resultado es"
        mostrar:  rdo
fin funcion
fin
```

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 "()"

4.3 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:

```
programa ejemploDeFuncionCompleto
inicio
        var integer a
        var integer b
        var integer rdo
        (sigue...)
```

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

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

Tips sobre las funciones:

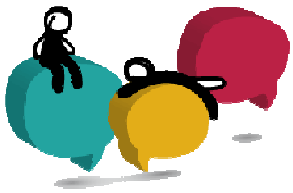
- **Si una función tiene un "retornar:" dentro SI o SI la función deberá declarar que devuelve un tipo de dato. Caso contrario, hay error.**
- **Una función que declara que devuelve un dato y no hace un "retornar:" en su interior representa un error.**

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.



Actividades de la Unidad 6 (optativa)

- Tomando como base los programas vistos en esta Unidad:
  - "diasDeLaSemana" (en ambas versiones: con y sin vectores)
  - "analisisFoda"
- Modificarlos para que incluyan funciones, separando la funcionalidad en bloques que tengan **coherencia lógica y utilidad real** de la o las funciones

Consignas:

- Utilizar el seucodcódigo presentado en el curso, con sus palabras reservas
- Utilizar los tipos de datos vistos en esta Unidad
- Cada alumno puede realizar y presentar COMO MÁXIMO la adaptación de un (1) programa.
- Las respuestas deberán ser posteadas en el foro "Foro de Actividad Optativa".
- El código deberá estar ***indentado***

El incumplimiento de alguna de las consignas invalida el trabajo presentado

Lo que vimos

- Uso y funcionalidad de las variables
- Uso y funcionalidad de las constantes
- Uso de tipos de datos para variables y constantes
- Conceptos y uso de los vectores y matrices
- Uso de funciones y conocer los tipos

Lo que viene:

- Integración de conceptos