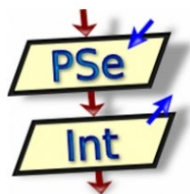


PROGRAMACIÓN DESDE CERO

# SUBPROGRAMAS EN PSEINT





## Objetivos de la Guía

En esta guía aprenderemos a:

- Separar el Algoritmo principal de las Funciones y SubProgramas.
- Diferenciar una función de un subprograma.
- Comprender qué debe ejecutarse en una función o subprograma.
- Lograr enviar información a las funciones o subprogramas a través de parámetros por valor o por referencia.
- Diferenciar pasaje por valor y por referencia.
- Llamar funciones o subprogramas desde el Algoritmo Principal.
- Definir variables de retorno y operar con ellas.
- Utilizar estructuras de control en Funciones y Subprogramas.

## SUBPROGAMAS

En esta guía aprenderemos el uso de los subprogramas. Hasta el momento, desarrollamos nuestro código en el Algoritmo Principal, con esto, si quisiéramos realizar varias sumas de variables, cada vez que quisiéramos adicionar su valor, deberíamos calcularlo, algo así:

```
Algoritmo sin_titulo
2
3   Definir num1, num2, num3, num4, num5 Como Entero
4   Definir resultado1, resultado2, resultado3, resultado4 Como Entero
5
6   resultado1 = num1 + num2
7   Escribir resultado1
8
9   resultado2 = num1 + num3
10  Escribir resultado2
11
12  resultado3 = num1 + num4
13  Escribir resultado3
14
15  resultado4 = num1 + num5
16  Escribir resultado4
17
18  FinAlgoritmo
```

Muchas veces nos ocurrirá en la programación que deberemos realizar la misma operación varias, incluso miles de veces. Para evitar que repitamos estas operaciones, surgen los subprogramas. Sería mucho más eficiente hacer lo siguiente:

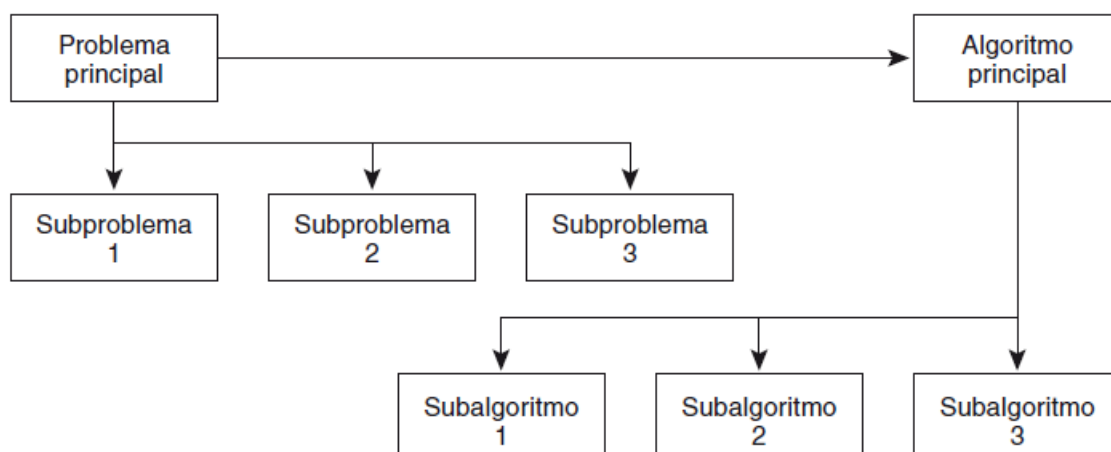
```
1  Algoritmo sin_titulo
2
3  Definir num1, num2, num3, num4, num5 Como Entero
4
5  sumar(num1, num2)
6  sumar(num1, num3)
7  sumar(num1, num4)
8  sumar(num1, num5)
9
10 FinAlgoritmo
11
12
13 SubProceso sumar(x, z)
14   Escribir x+z
15 FinSubProceso
```

Nótese aquí dos cosas importantes: la cantidad de líneas es menor y es mucho más legible y claro el código y su intención.

Un método muy útil para solucionar un problema complejo es dividirlo en **subproblemas** — problemas más sencillos— y a continuación dividir estos subproblemas en otros más simples, hasta que los problemas más pequeños sean fáciles de resolver. Esta técnica de dividir el problema principal en subproblemas se suele denominar “**divide y vencerás**”.

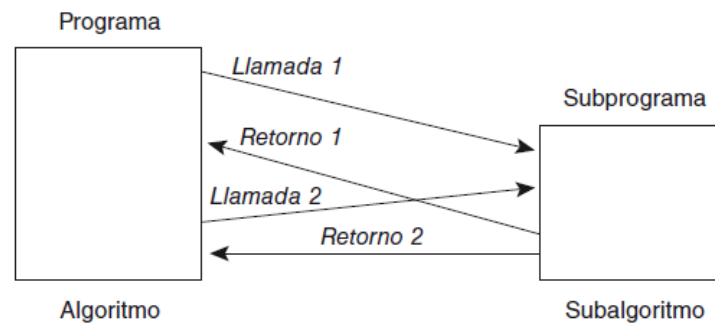
Este método de diseñar la solución de un problema principal obteniendo las soluciones de sus subproblemas se conoce como diseño descendente (top-down). Se denomina descendente, ya que se inicia en la parte superior con un problema general y se termina con varios subproblemas de ese problema general y las soluciones a esos subproblemas. Luego, las partes en que se divide un programa deben poder desarrollarse independientemente entre sí.

Las soluciones de un diseño descendente pueden implementarse fácilmente en lenguajes de programación y se los denomina subprogramas o sub-algoritmos si se emplean desde el concepto algorítmico.

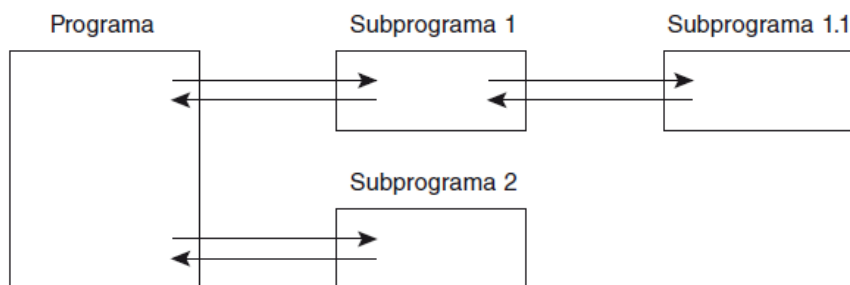


El problema principal se soluciona por el correspondiente programa o algoritmo principal, mientras que la solución de los subproblemas será a través de subprogramas, divididos en **procedimientos** y **funciones**. Un subprograma es un como un **mini algoritmo**, que recibe los *datos*, necesarios para realizar una tarea, desde el programa y devuelve *resultados* o realiza esa tarea.

Cada vez que el subprograma es invocado, el algoritmo va al subprograma invocado y se realiza el subprograma, y a su vez, un subprograma puede llamar a otros subprogramas.



*Un programa con un subprograma: función y procedimiento*



*Un programa con diferentes niveles de subprogramas*

## ¿QUÉ SON LAS FUNCIONES?

Matemáticamente una función es una operación que toma uno o más valores llamados argumentos y produce un resultado.

Cada función se evoca utilizando su nombre en una expresión con los argumentos encerrados entre paréntesis. A una función no se le llama explícitamente, sino que se le invoca o referencia mediante un nombre y una lista de parámetros.

## ¿CÓMO SE DECLARA UNA FUNCIÓN?

Cada función se crea fuera de nuestro algoritmo y requiere de una serie de pasos que la definen. Una función como tal subalgoritmo o subprograma tiene una constitución similar a los algoritmos. Esta comenzará con la palabra reservada **Función** y termina con la palabra **FinFunción** al igual que un Algoritmo. Al lado de nuestra palabra **Función**, comenzaremos la creación de nuestra función, esta constará de una cabecera que comenzará con un nombre para el **valor devuelto por la función**. El valor devuelto será una variable que definiremos dentro del cuerpo de nuestra función, ahí la daremos un tipo de dato. Este valor devuelto debe ser el resultado de la tarea que hemos dividido del problema general.

Después, va a ir el nombre de nuestra función y a continuación, entre paréntesis los parámetros de dicha función. Los parámetros van a ser los datos que necesitamos que nos envíe el algoritmo para realizar el subproblema en cuestión. Estos se escriben poniendo el nombre de la variable a recibir, sin su tipo de dato, y si quisiéramos pasar más de una variable, los separamos con comas. Los parámetros no son obligatorios a la hora de usar un subprograma, podemos tener una función sin parámetros, aunque es poco común.

Por último, irá el cuerpo de la función, que será una serie de acciones o instrucciones cuya ejecución hará que se asigne un valor al nombre de la función. Esto determina el valor particular del resultado que ha de devolverse al programa llamador.

El algoritmo o programa llama o invoca a la función con el nombre de esta última en una expresión seguida de una lista de argumentos que deben coincidir en cantidad, tipo y orden con los parámetros de la función. Se denominan argumentos a las variables o valores declarados en el algoritmo. Cuando se realiza una llamada a la función, los "valores" pasados o enviados a la función se denominan argumentos.

## SINTAXIS

```
Funcion variable_de_retorno <- Nombre (Parámetros)
```

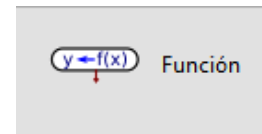
```
Definir variable_de_retorno como Tipo de Dato
```

```
<acciones> //cuerpo de la función
```

```
FinFuncion
```

¿Cómo se ve en PseInt?

```
1  Funcion variable_de_retorno <- Nombre ( Argumentos )
2
3  Fin Funcion
```



- **Parámetros:** uno o más parámetros de la siguiente forma: (parámetro 1 [Por Valor/Por Referencia], parámetro 2 [Por Valor/Por Referencia],...).
- **Nombre asociado con la función:** que será un nombre de identificador válido.
- **<acciones>:** instrucciones que constituyen la definición de la función y que debe contener alguna instrucción mediante la cual se asigne un valor a la variable\_de\_retorno.
- **Variable\_de\_retorno:** Esta variable la escribiremos en la definición de la función y debemos Definir su tipo dentro de la Función para poder usarla. En ella alojaremos el resultado final de la función. Cuando llamemos a la función, con los debidos argumentos, el algoritmo principal recibirá el valor de esta variable de retorno.



¿NECESITAS UN EJEMPLO?

```
1  Funcion retorno <- Sumar ( num1 Por Referencia, num2 Por Referencia )
2      Definir retorno Como Entero
3      retorno = num1 + num2
4  Fin Funcion
```



Para crear una función utiliza la plantilla que encuentras en el panel desplegable a la derecha de la pantalla.



### EJERCICIO COOPERAR

Realiza una función llamada Cooperar que reciba dos variables de tipo carácter, una variable debe contener el mensaje “Cooperando” y la otra “trabajamos mejor”. La función debe concatenar ambos textos.

### DETECCIÓN DE ERRORES

¿Puedes corregir esta función para que cumpla con su sintaxis?

```
Func retorno <- Paridad ( num
retorno : num MOD 2 == 0
Fin Funcion
```



#### Revisemos lo aprendido hasta aquí

- Seleccionar la plantilla de funciones desde el panel lateral.
- Nombrar funciones
- Asignar parámetros a las funciones.
- Identificar, definir y operar con el retorno.

### ¿CÓMO SE INVOCAN LAS FUNCIONES?

Una función puede ser llamada de la siguiente forma:

```
nombre_función(Argumentos)
```

- *nombre\_función*: función que va a llamar.
- *argumentos*: constantes, variables, expresiones.

Cada vez que se llama a una función desde el algoritmo principal se establece automáticamente una correspondencia entre los argumentos y los parámetros. Debe haber exactamente el mismo número de parámetros que de argumentos en la declaración de la función y se presupone una correspondencia uno a uno de izquierda a derecha entre los argumentos y los parámetros.

Además, cuando se llama a la función está va a devolver el resultado de las acciones realizadas en la función(**variable de retorno**) este resultado debe ser “atrapado” en el algoritmo. Ya sea para usarlo o solo para mostrarlo, por lo que al llamar una función debemos, o asignarle el resultado a una variable o concatenar el llamado de una función con un escribir

```
variable = nombre_funcion(argumentos)
```

```
Escribir nombre_funcion(argumentos)
```

Una llamada a la función implica los siguientes pasos:

1. A cada argumento se le asigna el valor real de su correspondiente parámetro.
2. Se ejecuta el cuerpo de acciones de la función.
3. Se devuelve el valor de la función y se retorna al punto de llamada.

Entonces para resumir se puede decir que la función tiene cinco componentes importantes:

- el **identificador**: va a ser el nombre de la función, mediante el cual la invocaremos.
- los **parámetros** son los valores que recibe la función para realizar una tarea.
- los **argumentos**, son los valores que envía el algoritmo a la función.
- las **acciones de la función**, son las operaciones que hace la función.
- **valor de retorno**(o el **resultado**) , es el valor final que entrega la función. La función devuelve un *único valor*.



¿NECESITAS UN EJEMPLO?

```
1  Funcion retorno <- Sumar ( num1 Por Referencia, num2 Por Referencia )
2      Definir retorno Como Entero
3      retorno = num1 + num2
4  Fin Funcion
5
6  Algoritmo sin_titulo
7      Definir num1, num2, num3, num4, num5, num6, resultado Como Entero
8      resultado = Sumar(num1, num2)
9      Escribir Sumar(num3, num4)
10  FinAlgoritmo
11
```



MANOS A LA OBRA!

## EJERCICIO COOPERAR – PARTE 2

¿Recuerdan la Función Cooperar? Hora de llamarla en el algoritmo principal y mostrar el mensaje por pantalla.



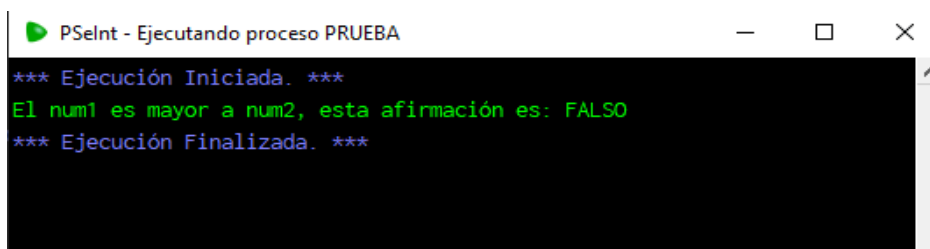
Una buena costumbre es alojar el retorno de las funciones en variables, ya que esto guardará el resultado por si debemos operar con él más adelante.

## DETECCIÓN DE ERRORES

¿Puedes corregir esta función para lograr el resultado esperado?

```
Funcion retorno <- Comparar ( num1 )
retorno = num1 num2
Algoritmo Prueba
Definir num1, num2 Como Entero
Definir resultado Como Logico
num1 = 3
num2 = 6
resultado = retorno(num1,num2)
Escribir "El num1 es mayor a num2, esta afirmación es: " resultado
FinAlgoritmo
```

¿Cuál es el resultado a lograr?



```
PSeInt - Ejecutando proceso PRUEBA
*** Ejecución Iniciada. ***
El num1 es mayor a num2, esta afirmación es: FALSO
*** Ejecución Finalizada. ***
```



Pueden encontrar ejemplos para descargar de Funciones en Aula Virtual.

## PROCEDIMIENTOS

Ahora que sabemos que las **funciones devuelven un valor** para que lo usemos a través de la variable de retorno, vamos a ver un caso distinto. Los **procedimientos** son similares, pero **no devuelven ningún valor**, sólo realizan una tarea.

### ¿QUÉ ES UN PROCEDIMIENTO?

Un **procedimiento** es un subprograma que ejecuta un proceso específico. En PseInt lo llamaremos **SubProceso**. Ningún valor está asociado con el nombre del procedimiento; por consiguiente, no puede ocurrir en una expresión. Un procedimiento se llama escribiendo su nombre. Cuando se invoca el procedimiento, los pasos que lo definen se ejecutan y a continuación se devuelve el control al programa que le llamó.



## Sintaxis

**SubProceso** Nombre (*parámetros*)

<acciones>

**FinSubProceso**

Los parámetros tienen el mismo significado que en las funciones.

## ¿CÓMO INVOCAMOS A UN SUBPROGRAMA?

Un procedimiento puede ser llamado de la siguiente forma:

*nombre(argumentos)*

- nombre\_procedimiento: procedimiento que se va a llamar.
- argumentos: constantes, variables, expresiones.



Pueden encontrar ejemplos para descargar de Procedimientos en Aula Virtual.



### Revisemos lo aprendido hasta aquí

- Declarar funciones y subprocesos
- Invocar a las funciones y subprocesos desde el algoritmo principal
- Enviar argumentos al invocar.
- Revisar la Función Cooperar de un compañero y compararla con la propia.

## ÁMBITO: VARIABLES LOCALES Y GLOBALES

Las variables utilizadas en los programas principales y subprogramas se clasifican en dos tipos: *variables locales* y *variables globales*.

Una **variable local** es aquella que está declarada y definida dentro de un subprograma, en el sentido de que está dentro de ese subprograma, es distinta de las variables con el mismo nombre declaradas en cualquier parte del programa principal y son variables a las que el algoritmo principal no puede acceder de manera directa.

El significado de una variable se confina al procedimiento en el que está declarada. Cuando otro subprograma utiliza el mismo nombre se refiere a una posición diferente en memoria. Se dice que tales variables son locales al subprograma en el que están declaradas.

Una **variable global** es aquella que está declarada en el programa o algoritmo principal, del que dependen todos los subprogramas y a las que pueden acceder los subprogramas, a través del paso de argumento. La parte del programa/algoritmo en que una variable se define se conoce como ámbito o alcance (scope, en inglés).

El uso de variables locales tiene muchas ventajas. En particular, hace a los subprogramas independientes, siendo solo la comunicación entre el programa principal y los subprogramas a través de la lista de parámetros.

Una variable local a un subprograma no tiene ningún significado en otros subprogramas. Si un subprograma asigna un valor a una de sus variables locales, este valor no es accesible a otros programas, es decir, no pueden utilizar este valor. A veces, también es necesario que una variable tenga el mismo nombre en diferentes subprogramas. Por el contrario, las variables globales tienen la ventaja de compartir información de diferentes subprogramas sin la necesidad de ser pasados como argumento

## COMUNICACIÓN CON SUBPROGRAMAS: PASO DE ARGUMENTOS

Cuando un programa llama a un subprograma, la información se comunica a través de la lista de parámetros y se establece una correspondencia automática entre los parámetros y los argumentos. Los parámetros son “sustituídos” o “utilizados” en lugar de los argumentos.

La declaración del subprograma se hace con:

```
Subproceso nombre (PA1, PA2, ..., PAn)
    <acciones>
```

```
FinSubproceso
```

y la llamada al subprograma con:

```
nombre (AR1, ARG2,..., ARGn)
```

donde **PA1, PA2, ..., PAn** son los parámetros y **ARG1, ARG2, ..., ARGn** son los argumentos.

Para dar un ejemplo de la vida real, tenemos que ver a los subprogramas como formularios vacíos que esperan ser llenados. Es decir, si yo tengo un formulario que espera que complete ‘nombre’ y ‘apellido’ esos serían los parámetros del subprograma. Al llenar con mis datos, estoy dando mi nombre y apellido como argumento. Esto permite que muchas personas puedan llenar el mismo formulario, pero cada uno con su información. Miremos el siguiente ejemplo donde la función tiene los parámetros genéricos nombre y apellido, pero al llamarlo desde el algoritmo le pasamos nombres concretos.

```
1  Algoritmo sin_titulo
2
3      saludar("María", "Martínez")
4
5      saludar("John", "Jones")
6
7  FinAlgoritmo
8
9
10 SubProceso saludar(nombre, apellido)
11     Escribir "Hola ", nombre, " ", apellido
12 FinSubProceso
13
```

PSelnt - Ejecutando proceso SIN\_TITULO

```
*** Ejecución Iniciada. ***
Hola María Martínez
Hola John Jones
*** Ejecución Finalizada. ***
```

Cuando nosotros decidimos los parámetros que va a necesitar nuestro subprograma, también podemos decidir cuál va a ser el comportamiento de los argumentos en nuestro subprograma cuando lo invoquemos y se los pasemos por paréntesis. Esto va a afectar directamente a los argumentos y no al resultado final del subprograma.

Para esto existen dos tipos más empleados para realizar el paso de argumentos, el **paso por valor** y el **paso por referencia**.

### Paso por Valor

Los argumentos se tratan como variables locales y los valores de dichos argumentos se proporcionan copiando los valores de los argumentos originales. Los parámetros (locales a la función o procedimiento) reciben como valores iniciales una copia de los valores de los argumentos y con ello se ejecutan las acciones descritas en el subprograma.

Aunque el paso por valor es sencillo, tiene una limitación acusada: no existe ninguna otra conexión con los parámetros, y entonces los cambios que se produzcan dentro del subprograma no producen cambios en los argumentos originales y, por consiguiente, no se pueden poner argumentos como valores de retorno. El argumento actual no puede modificarse por el subprograma.

En PSELnt todas las variables que pasemos como argumentos pasan por defecto **“Por Valor”** sino se especifica lo contrario explícitamente.

### Paso por Referencia

En numerosas ocasiones se requiere que ciertos argumentos sirvan como argumentos de salida, es decir, se devuelvan los resultados al programa que llama. Este método se denomina **paso por referencia** o también de llamada por dirección o variable. El programa que llama pasa al subprograma la dirección del argumento actual (que está en el programa que llama). Una referencia al correspondiente argumento se trata como una referencia a la posición de memoria, cuya dirección se ha pasado. Entonces una variable pasada como argumento real es compartida, es decir, se puede modificar directamente por el subprograma.

La característica de este método se debe a su simplicidad y su analogía directa con la idea de que las variables tienen una posición de memoria asignada desde la cual se pueden obtener o actualizar sus valores. El área de almacenamiento (direcciones de memoria) se utiliza para pasar información de entrada y/o salida; en ambas direcciones.

En este método los argumentos son de entrada/salida y los argumentos se denominan **argumentos variables**.

¡¡En la próxima pagina podremos encontrar un ejercicio!!



## MANOS A LA OBRA!

Copia, pega y ejecuta el código. Analiza qué está sucediendo

Algoritmo valorVSreferencia

```
    Definir num Como Entero
    num = 2
    Escribir num
    Escribir "Ahora enviamos el número a la función por valor y el
resultado es:"
    elevarAlCuadradoPorValor(num)
    Escribir num
    Escribir "*****"
    Escribir "Ahora enviamos el número a la función por referencia y el
resultado es:"
    elevarAlCuadradoPorReferencia(num)
    Escribir num
FinAlgoritmo
```

SubProceso elevarAlCuadradoPorValor(num Por Valor)

```
    num = num * num
```

FinSubProceso

SubProceso elevarAlCuadradoPorReferencia(num Por Referencia)

```
    num = num * num
```

FinSubProceso

## RECURSIÓN

Una función o procedimiento que se puede llamar a sí mismo se llama recursivo. La recursión (recursividad) es una herramienta muy potente en algunas aplicaciones, sobre todo de cálculo. La recursión puede ser utilizada como una alternativa a la repetición o estructura repetitiva. El uso de la recursión es particularmente idóneo para la solución de aquellos problemas que pueden definirse de modo natural en términos recursivos.

La escritura de un procedimiento o función recursiva es similar a sus homónimos no recursivos; sin embargo, para evitar que la recursión continúe indefinidamente es preciso incluir una condición de terminación, que suele hacerse con una estructura condicional.



### Revisemos lo aprendido hasta aquí

- Variables globales y locales
- Pasaje de datos por valor y por referencia
- Qué es la recursión y para qué sirve

## EJERCICIOS DE APRENDIZAJE

Para cada uno de los siguientes ejercicios realizar el análisis del problema e indicar cuáles son los datos de entrada y cuáles son los datos de salida. Escribir luego el programa en PSeInt utilizando funciones y/o procedimientos.



**VIDEOS:** Te sugerimos ver los videos relacionados con este tema, antes de empezar los ejercicios, los podrás encontrar en tu aula virtual o en nuestro canal de YouTube.

### Subprogramas: Funciones

1. Realizar una función que calcule la suma de dos números. En el algoritmo principal le pediremos al usuario los dos números para pasárselos a la función. Después la función calculará la suma y lo devolverá para imprimirlo en el algoritmo.
2. Realizar una función que valide si un número es impar o no. Si es impar la función debe devolver un verdadero, si no es impar debe devolver falso. **Nota:** la función no debe tener mensajes que digan si es par o no, eso debe pasar en el Algoritmo.
3. Crea una función *EsMultiplo* que reciba los dos números pasados por el usuario, validando que el primer número múltiplo del segundo y devuelva verdadero si el primer número es múltiplo del segundo, sino es múltiplo que devuelva falso.
4. Realizar un programa que pida al usuario una frase y una letra a buscar en esa frase. La función debe devolver la cantidad de veces que encontró la letra. **Nota:** recordar el uso de la función Subcadena().
5. Realizar una función que reciba un numero ingresado por el usuario y averigüe si el número es primo o no. Un número es primo cuando es divisible sólo por 1 y por sí mismo, por ejemplo: 2, 3, 5, 7, 11, 13, 17, etc. **Nota:** recordar el uso del MOD.

### Subprogramas: Procedimientos

6. Realizar un procedimiento que permita intercambiar el valor de dos variables de tipo entero. La variable **A**, debe terminar con el valor de la variable **B**.
7. Crear un procedimiento que calcule la temperatura media de un día a partir de la temperatura máxima y mínima. Crear un programa principal, que, utilizando un procedimiento, vaya pidiendo la temperatura máxima y mínima de n días y vaya mostrando la media de cada día. El programa pedirá el número de días que se van a introducir.
8. Realizar un procedimiento que permita realizar la división entre dos números y muestre el cociente y el resto utilizando el método de restas sucesivas.

El método de división por restas sucesivas consiste en restar el dividendo con el divisor hasta obtener un resultado menor que el divisor, este resultado es el residuo, y el número de restas realizadas es el cociente. Por ejemplo: 50 / 13:

50 – 13 = 37 una resta realizada

37 – 13 = 24 dos restas realizadas

24 – 13 = 11 tres restas realizadas

dado que 11 es menor que 13, entonces: el residuo es 11 y el cociente es 3.

9. Escribir un programa que procese una secuencia de caracteres ingresada por teclado y terminada en punto, y luego codifique la palabra o frase ingresada de la siguiente manera:

cada vocal se reemplaza por el carácter que se indica en la tabla y el resto de los caracteres (incluyendo a las vocales acentuadas) se mantienen sin cambios.

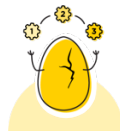
a	e	i	o	u
@	#	\$	%	*

Realice un subprograma que reciba una secuencia de caracteres y retorne la codificación correspondiente. Utilice la estructura “según” para la transformación.

Por ejemplo, si el usuario ingresa: Ayer, lunes, salimos a las once y 10.

La salida del programa debería ser: @y#r, l\*n#s, s@l\$m%s @ l@s %nc# y 10.

**NOTA:** investigue el uso de la función concatenar de PSeInt para armar la palabra/frase.



#### ¿Lograste los objetivos de la guía?

¡Terminamos los ejercicios fundamentales de esta guía! **Te invitamos a ir a tu aula virtual y contestar una breve encuesta para revisar los objetivos de la esta guía**, reflexiona sobre cada uno de ellos haciéndote la siguiente pregunta ¿Pude aplicar este objetivo al realizar los ejercicios? Si es así, ¡Excelente!

¿Terminaste con los ejercicios fundamentales? hay más ejercicios para que puedas seguir practicando.

#### ¿No pudiste cumplir los objetivos de la guía?

Si aún no has logrado tildar los objetivos propuestos, **no te preocupes**, a través de Slack contacta a un Coach para que te ayude a diseñar una estrategia personalizada para entender qué pasó y poder avanzar.

## EJERCICIOS DE APRENDIZAJE EXTRA

Si ya terminaste la guía y te queda tiempo libre en las mesas, puedes continuar con estos ejercicios extra, recordando siempre que no es necesario que los termines para continuar con el tema siguiente. Por último, recuerda que la prioridad es ayudar a los compañeros de la mesa y que cuando tengas que ayudar, lo más valioso es que puedas explicar el ejercicio con la intención de que tu compañero lo comprenda, y no sólo mostrarlo. ¡Muchas gracias!

### Subprogramas: Funciones

1. Realizar una función que calcule y retorne la suma de todos los divisores del número  $n$  distintos de  $n$ . El valor de  $n$  debe ser ingresado por el usuario.
2. Diseñar una función que reciba un número en forma de cadena y lo devuelva como número entero. El programa podrá recibir números de hasta 3 dígitos. Nota: no poner números con decimales ni letras. Ejemplo: ingresando "100"(carácter) debe convertirse en 100(entero).
3. Crear una función llamada "Login", que recibe un nombre de usuario y una contraseña y que devuelve Verdadero si el nombre de usuario es "usuario1" y si la contraseña es "asdasd". Además, la función calculara el número de intentos que se ha usado para loguearse, tenemos solo 3 intentos, si nos quedamos sin intentos la función devolverá Falso.
4. Los empleados de una fábrica trabajan en dos turnos: Diurno y Nocturno. Se desea calcular el jornal diario de acuerdo con las siguientes reglas:
  - a) La tarifa de las horas diurnas es de \$ 90
  - b) La tarifa de las horas nocturnas es de \$ 125
  - c) En caso de ser feriado, la tarifa se incrementa en un 10% si el turno es diurno y en un 15% si el turno es nocturno.

El programa debe solicitar la siguiente información al usuario: el nombre del trabajador, el día de la semana, el turno (diurno o nocturno) y la cantidad de horas trabajadas. Además, debemos preguntarle al usuario si el día de la semana (lunes, martes, miércoles, etc.) era festivo o no, para poder calcular el jornal diario. Utilice una función para realizar el cálculo.

5. Realizar una función que calcule la suma de los dígitos de un número.

**Ejemplo:**  $25 = 2 + 5 = 7$

**Nota:** Para obtener el último número de un dígito de 2 cifras o más debemos pensar en el resto de una división entre 10. Recordar el uso de la función Mod y Trunc.

6. Realizar una función que reciba un número ingresado por el usuario y averigüe si el número tiene todos sus dígitos impares (ejemplo: 333, 55, etc.). Para esto vamos a tener que separar el número en partes (si es un número de más de un dígito) y ver si cada número es par o impar. Nota: recordar el uso de la función Mod y Trunc(). No podemos pasar el número a cadena para realizar el ejercicio.
7. Realizar una función que permita obtener el término  $n$  de la sucesión de Fibonacci. La sucesión de Fibonacci es la sucesión de los siguientes números:

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Donde cada uno de los números se calcula sumando los dos anteriores a él. Por ejemplo:

La sucesión del número 2 se calcula sumando (1+1)

Análogamente, la sucesión del número 3 es (1+2),



Y la del 5 es (2+3),

Y así sucesivamente...

La sucesión de Fibonacci se puede formalizar de acuerdo a la siguiente fórmula:

Fibonacci (n) = Fibonacci (n-1) + Fibonacci (n-2) para todo  $n > 1$

Fibonacci (n) = 1 para todo  $n \leq 1$

Por lo tanto, si queremos calcular el término “n” debemos escribir una función que reciba como argumento el valor de “n” y que calcule la serie hasta llegar a ese valor.

Para conocer más acerca de la serie de Fibonacci consultar el siguiente link:

<https://quantdare.com/numeros-de-fibonacci/>

8. Realizar una función que reciba un numero ingresado por el usuario y averigüe si el número es capicúa o no (Por ejemplo: 12321). **Nota:** recordar el uso del MOD y el Trunc. No podemos transformar el numero a cadena para realizar el ejercicio.

### Subprogramas: Procedimientos

9. Crea un procedimiento “convertirEspaciado”, que reciba como argumento un texto y muestre una cadena con un espacio adicional tras cada letra.

Por ejemplo, “Hola, tú” devolverá “H o l a , t ú “. Crea un programa principal donde se use dicho procedimiento.

10. Realizar un subproceso que reciba una letra y muestre un mensaje si esa letra esta entre las letras “M” y “T”. Recordar que Pseint le da un valor numérico a cada letra a través del Código Ascii, lo que nos deja usar operadores relacionales con letras y cadenas.

11. Crear un programa que dibuje una escalera de números, donde cada línea de números comience en uno y termine en el número de la línea. Solicitar la altura de la escalera al usuario al comenzar. Ejemplo: si se ingresa el número 3:

```
1
12
123
```

12. Realice un algoritmo que solicite al usuario una fecha y muestre por pantalla la fecha anterior. Para ello se deberá utilizar un procedimiento llamado diaAnterior que reciba una fecha representada a través de tres enteros día, mes y año, y retorne la fecha anterior. Puede asumir que día, mes y año representan una fecha válida. Realice pruebas de escritorio para los valores día=5, mes=10, año=2012 y para día=1, mes=3, año=2004.

13. Diseñar un procedimiento que reciba una frase, y el programa remueva todas las vocales repetidas. Al final el procedimiento mostrará la frase final.

Por ejemplo:

Entrada: “Habia una vez un barco”

Salida: “Habi un vez n brco”

*Se marcan en rojo las repetidas sólo para explicar la consigna. Las vocales ‘e’, ‘i’ y ‘o’ quedan al no estar repetidas.*