****

Facultad de Ciencias Exactas y Naturales

Curso: Programación

Codigo: 0303158

Tema: Funciones

Semestre: 2020-1

Medellín, 2020

**LAS FUNCIONES EN PROGRAMACIÓN**

Una función permite asociar, a cada elemento de un conjunto, un único elemento de otro conjunto. La manera habitual de denotar la función *f* que asigna a cada elemento del conjunto A un elemento del conjunto B consta de dos partes:

|  |  |
| --- | --- |
| Perfil de la función | Regla de asignación |
|  | **,** |
| Contiene:   1. El nombre de la función: 2. El **dominio**, o conjunto de elementos a los que se aplica la función: ***A*** 3. El conjunto que contiene los elementos asociados a cada elemento del dominio, denominados **imágenes**: | denota la regla o [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) para obtener la [imagen](https://es.wikipedia.org/wiki/Imagen_(matem%C3%A1ticas))  del elemento  .  , la imagen de bajo la función , también se conoce como el valor de la función en .  también es conocido como el argumento de la función ,  se dice, también, que las funciones reciben argumentos y devuelven valores. |

Ejemplos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | |  | **,** | |
|  | |  |  | |
|  | |  |  | |
|  |  | | |  |

En los cuatro ejemplos anteriores, podríamos estar hablando de: el área de un círculo, la diagonal de un rectángulo, el factorial de un entero positivo y el m.c.d de dos enteros positivos, calculado por medio del algoritmo de Euclides, respectivamente.

**Sintaxis en Matlab u Octave**

Para definir una función **f** en Matlab u Octave, si se conoce la fórmula o el algoritmo para calcular el valor de la función, se debe crear un archivo **.m**, con el perfil de la función y las instrucciones para calcular el valor de la función a partir de los argumentos. El programa debe grabarse con el mismo nombre de la función.

|  |  |
| --- | --- |
| **function y = f(x)**  **<instrucciones>**  **endfunction** | **f** es el nombre de la función,  **y** es una variable en donde se devolverá el valor de la función,  **x** es la lista de argumentos, separados por comas,  **<instrucciones>** es la secuencia de instrucciones para calcular **y**. |
| **function y = f(x)**  **<instrucciones>** |

Ejemplo:

|  |
| --- |
| **function z = mcd(a,b)**  **r = mod(a,b);**  **while r > 0**  **a = b;**  **b = r;**  **r = mod(a,b);**  **end**  **z = b;** |

**Programación funcional**

Una función puede verse como un programa cuyos datos son los argumentos y cuyos resultados son los valores devueltos por la función.

En adelante, nuestros programas no serán **scripts** sino funciones. No se volverán a pedir datos con instrucciones de entrada, sino que ellos entrarán como argumentos, ni se volverán a mostrar resultados con instrucciones de salida, sino que ellos serán los valores devueltos por la función.

Ejemplos: Convertir scripts hechos anteriormente en funciones.

En algunos casos, las funciones imprimirán resultados, si ellos no son los valores devueltos por la función.

Ejemplo: Verificar si n es primo.

En otros casos las funciones no devuelven valores, pero imprimen resultados, caso en el cual se denominan Procedimientos.

La programación funcional permite delegar trabajos por medio de subprogramas que se encarguen de partes de procesos complejos o muy extensos.

**Funciones booleanas**

Una función que devuelve un valor de verdad se conoce como una función booleana. Puede decirse que se utilizan para verificar condiciones.

Ejemplo: Primos relativos

|  |
| --- |
| **function z = relativos(m,n)**  **if mcd(m,n) == 1**  **z = 1;**  **else**  **z = 0;**  **end** |

**Funciones recursivas**

Una función es recursiva si se utiliza a sí misma en su propia definición. Ello es posible si el llamado a sí misma depende de una condición que no siempre es verdadera durante la ejecución, es decir, si hay una o más condiciones de salida del proceso recursivo.

Ejemplos: Factorial, Fibonacci, mcd.

Todas estas fórmulas son recursivas, pues se utilizan en su propia definición. Pero permiten aplicarse a cualquier elemento de su dominio y encontrar el correspondiente valor de la función.

|  |  |
| --- | --- |
| Así: | Igualmente: |
| **fact(4) = 4\*fact(3)**  **fact(3) = 3\*fact(2)**  **fact(2) = 2\*fact(1)**  **fact(1) = 1**  **fact(4) = 4\*3\*2\*1** | **mcd(81,15) = mcd(15,6)**  **mcd(15,6) = mcd(6,3)**  **mcd(6,3) = 3** |
| Sustituyendo hacia atrás: | Es decir: |
| **fact(2) = 2**  **fact(3) = 6**  **fact(4) = 24** | **mcd(81,15) = 3** |

Los lenguajes de programación incluyen el mecanismo que permite esperar resultados pendientes y sustituirlos donde haga falta, una especie de 'memoria de recursión'. Esta memoria se puede ampliar, en caso de que haga falta.

**Funciones recursivas en Matlab**

|  |  |
| --- | --- |
| **function z = fact(n)**  **if n < 2**  **z = 1;**  **else**  **z = n\*fact(n-1);**  **end** | **function z = fibo(n)**  **if n < 3**  **z = 1;**  **else**  **z = fibo(n-1) + fibo(n-2);**  **end** |
| **function z = mcd(a,b)**  **r = mod(a,b);**  **if r == 0**  **z = b;**  **else**  **z = mcd(b,r);**  **end** |  |

Una función recursiva consta, entonces, de dos partes: una parte recursiva donde la función se llama a sí misma, pero para un argumento que tiende a una salida, y la 'salida' de la recursión, donde termina el proceso recursivo.

**De iterativo a recursivo**

Una recursión conlleva un proceso repetitivo. Cabe, entonces, la pregunta: ¿todo proceso iterativo se puede hacer recursivo?

Ejemplo:

|  |  |
| --- | --- |
| Iterativo: | Recursivo: |
| **function mitadear(n)**  **while n > 0**  **disp(n);**  **n = floor(n/2);**  **end;** | **function mitadear(n)**  **if n > 0**  **disp(n);**  **mitadear(floor(n/2));**  **end;** |
| Iterativo: | Recursivo: |
| **function S = cifras(n)**  **while n > 0**  **disp(mod(n,10));**  **n = floor(n/10);**  **end;** | **function S = cifras4(n)**  **if n > 0**  **disp(mod(n,10));**  **cifras4(floor(n/10));**  **end;** |
|  | **function S = cifras7(n)**  **if n > 0**  **cifras7(floor(n/10));**  **disp(mod(n,10));**  **end;** |
| **function S = sumacifras(n)**  **S = 0;**  **while n > 0**  **S = S + mod(n,10);**  **n = floor(n/10);**  **end;** | **function z = sumacifras(n,S)**  **if n > 0**  **u = mod(n,10);**  **r = floor(n/10);**  **z = sumacifras(r,S+u);**  **else**  **z = S;**  **end;** |

Pruebas de escritorio para funciones recursivas.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **mitadear(7)** | **disp(7)**  **mitadear(3)** | **disp(7)**  **disp(3)**  **mitadear(1)** | **disp(7)**  **disp(3)**  **disp(1)**  **mitadear(0)** | **disp(7)**  **disp(3)**  **disp(1)** |
| **cifras4(271)** | **disp(1)**  **cifras4(27)** | **disp(1)**  **disp(7)**  **cifras4(2)** | **disp(1)**  **disp(7)**  **disp(2)**  **cifras4(0)** | **disp(1)**  **disp(7)**  **disp(2)** |
| **cifras7(271)** | **cifras7(27)**  **disp(1)** | **cifras7(2)**  **disp(7)**  **disp(1)** | **cifras7(0)**  **disp(2)**  **disp(7)**  **disp(1)** | **disp(2)**  **disp(7)**  **disp(1)** |
| **sumacifras(271,0)** |  | … |  |  | |