Taller de preparación para la OCI - PUC

Funciones

1. Introducción

Las funciones son una herramienta muy poderosa en programación. Hasta ahora cada vez que queríamos replicar una acción una y otra vez debíamos repetir el código, lo que era una tarea bastante tediosa y ademas lo volvía menos legible y menos ordenado.

Las funciones son bloques de código que nos permiten realizar una tarea sistemáticamente de manera más simple y ordenada. En concreto, ahora tenemos el poder de reutilizar código dentro de nuestros programas invocando a las funciones que definamos, lo que nos puede ahorrar mucho trabajo.

Otra razón importante por la que nos es útil utilizar funciones es que habiendo definido una podemos utilizarla en otros programas que escribamos. Esto también implica que podemos **importar**[1] funciones que ya han sido creadas y utilizarlas sabiendo simplemente qué acepta como parámetro y qué arroja[2].

De hecho, sin darnos cuenta ya hemos usado funciones. *println()* presenta un tipo de función, que revise un dato y lo muestra en la consola. Estas funciones predefinidas por el lenguaje son denominadas **built-in**. El proceso de cómo realizan sus operaciones no nos importa por ahora, pero nos sirven para ejemplificar el concepto de que las funciones sirven como *cajas negras* recibiendo o no algún parámetro (**inputs**) y arrojando o no un valor de salida(**outputs**). Más adelante profundizaremos en por qué recibir y arrojar algo es opcional.

2. Estructura

2.1. Estructura básica

```
1 public static <return type> <nombre de la función>() {
2     <código>
3 }
```

Es importante diferenciar la definición de la función, que es cuando la creamos, de su llamado en el Main (programa principal), que es cuando efectivamente utilizamos la función. Es decir, que tener la función definida sin llamarla en el programa principal es equivalente a tener una carta en nuestra baraja sin nunca utilizarla en el juego.

Llamado a la función

```
1 funct();
```

Declaración de la función

```
1 public static void funct() {
2     System.out.println("Hello World");
3 }
```

2.2. Aceptando parámetros

A veces vamos a querer que nuestras funciones revisen algún dato en especifico para poder trabajar con él. Por ejemplo, si queremos crear una función que genere un saludo, este va a depender de distintos

[1] Importar es notificarle a nuestro programa de la existencia de un bloque de código externo para permitir su utilización
[2] Nótese que ni siquiera es necesario saber como lo hace

parámetros: uno podría ser la hora, de esto dependería si se dirá "Buenos días", "Buenas noches", o "Buenas tardes", y otro podría ser el nombre de la persona. Para poder hacer esto se necesita contar con una estructura básica:

```
public static <return type> <nombre de la función>(param1, param2) {
    // Código
}
```

En el ejemplo `param1` y `param2` separados por comas corresponden a los parámetros que recibe la función. Al momento de declararlos se debe explicitar el tipo de dato de estos ya sean String, double, int, etc.
Un ejemplo sencillo es definir una función que imprima el resultado de una suma:

```
// Primero se declara que el tipo del los inputs a y b será int
public static void sumar(int a, int b) {
    int suma = a + b;
    System.out.println("Resultado operación:" + suma);
    //
}
```

Un detalle que podemos destacar del ejemplo anterior es que la variable *suma* sólo existe dentro de la función, por lo que si la llamamos fuera de ella nos arrojará un error. Otro ejemplo más elaborado que serviría para modelar la situación descrita al inicio podría ser:

```
public static void saludar(int hora, String nombre) {
    // El parámetro hora: un int como 23
    // nombre: String

    String mensaje = "" ;

    // Condicional que selecciona tipo de saludo según la hora ingresada

    if (hora <12) {
      mensaje += "Buenos días ";
    }
    else if (hora<8) {
      mensaje += "Buenas tardes ";
    }
    else {
      mensaje += "Buenas noches ";
    }

    // Función que imprime el mensaje
    System.out.println(mensaje + nombre);
}
```

2.3. Return type

2.3.1. El concepto

Si entendemos las funciones como cajas negras, el return type declara el tipo de aquello que la función retornará. Hasta el momento hemos estado usando **void**. Este será el return type que usaremos cada vez que no se quiera retornar algo, si no más bien realizar una acción tal como imprimir.

Otros return types comunes son boolean, int, double, String y char. Al usar alguno de estos (que explicitarán que la función retorna un dato) se deberá escribir dentro de la función ***return*** *dato a retornar*. Esto permitirá reutilizar el output de la función dentro de nuestro código principal.

Volvamos por un minuto a nuestra función sumar, solo que esta vez digamos que queremos igualar una variable a la suma de dos números. Para hacerlo, el return type tendría que ser int y el código se vería así:

```
1 public class Main {
2     public static void main(String[] args) {
3         int new_number = sumar(4,3);
4     }
5     public static int sumar(int a, int b) {
6         suma=a+b;
7         return suma;
8     }
9 }
```

2.3.2. Recibiendo ejecuciones de funciones como input

Ahora que entendemos el concepto de return type, otra cosa que permiten las funciones es que se ingresen otras funciones como parámetros. Por ejemplo, digamos que definimos una función que imprime un String cualquiera para no tener que escribir System.out.println().

```
1 public static void imprimir(String mensaje) {
2     System.out.println(mensaje);
3 }
```

Teniendo otra función que *retorne* (es decir que su return type sea distinto de void) un String personalizado (como el ejemplo de la sección 2.2 del saludo) nos permitiría perfectamente ejecutar algo como

```
1 imprimir(saludo(11,"Mario"));
2 /* Esto generaría como salida
3 >>> Buenos días Mario
4 */
```

Notemos que esto funciona sólo porque el tipo de input que recibe `imprimir()` es un String lo que calza con el output de `saludo()`.

3. Ejemplos

Ejemplo 3.1. Defina una función que ingresando una medida en kg. retorne libras y viceversa.

Aquí para identificar si se está ingresando un valor en kilos o libras le pedimos al usuario que lo ingrese como String en el parámetro *modo*. Después se hacen las operaciones correspondientes sabiendo que 1 kg equivale a 2.20462 libras.

```
1 public class Main{
2
3     public static void main(String[] args) {
4         // Para probar ingresaremos un par de inputs
5         double libras = convertir("kilos", 51.1);
6         double kilos = convertir("libras", 32);
```

```
7           /* Aquí los imprimimos. \n representa un salto de linea
8           y String.format() le asigna a %s el valor de las variables
9           después de la expresión seguidas por comas en orden.*/
10          System.out.println(String.format("Libras: %s\nKilos: %s", Libras, kilos));
11      }
12
13      public static double convertir(String modo, double valor) {
14          if (modo == "libras") {
15              double kilos = valor * 0.453592;
16              return kilos;
17          } else {
18              double libras = valor * 2.20462;
19              return libras ;
20          }
21      }
22  }
```

Ejemplo 3.2. Escribir un programa que calcule el área y perímetro de un cuadrado de lado 4, de un rectángulo de lados 4 y 6, y de un triángulo rectángulo de base 5 y altura 7.

Si no utilizáramos funciones, el proceso de escribir el código independiente para calcular cada operación para cada figura se volvería absurdamente largo y tedioso. Afortunadamente, ahora podemos hacer funciones que calculen área y perímetro (si bien podemos hacer una función que haga ambas, siempre es útil y recomendable que una función se encargue de sólo una tarea).

```
1   public class Main {
2       public static void main(String[] args) {
3           // Programa principal
4           /* Ahora simplemente llamamos a nuestras funciones
5           con los valores del enunciado */
6
7           double area_cuadrado = area_cuadrilatero(4, 4) ;
8           double area_rect = area_cuadrilatero(4, 6) ;
9           double area_tri = area_triangulo(5, 7) ;
10          double perimetro_cuadrado = perimetro("cuadrado", 4, 4) ;
11          double perimetro_rect = perimetro("rectangulo", 4, 6) ;
12          double perimetro_tri = perimetro("triangulo", 5, 7) ;
13      }
14
15      public static double area_cuadrilatero(double lado_a, double lado_b) {
16          return lado_a * lado_b;
17      }
18
19      public static double area_triangulo(double base, double altura) {
20          return (base * altura) / 2;
21      }
22
23      public static double perimetro(String tipo, double lado_1, double lado_2) {
24          if (tipo == "triangulo") {
25              //Se calcula con el Teorema de Pitágoras el tercer lado
26            /* Se utilizan las FUNCIONES del modulo Math
27            para calcular la raiz y así poder elevar */
28              double lado_3 = Math.sqrt((Math.pow(lado_1,  2) + Math.pow(lado_2, 2)));
29              return lado_1 + lado_2 + lado_3;
30          }
```

```
31          else{
32              return lado_1 * 2 + lado_2 * 2;
33          }
34      }
35  }
```

Ejemplo 3.3. Escribir un programa en que se diga la cantidad de días en un año. Tome en consideración si el año es bisiesto.

Esta claro que se podría resolver el problema sin funciones. Sin embargo, para consolidar el concepto lo resolveremos definiendo dos funciones. Notar que en la lnea 13 se utilizan las condiciones para que un año sea bisiesto, es decir si es divisible entre 4, excepto aquellos divisibles entre 100 pero no entre 400.

```
1  public class Main {
2      public static void main (String[] args) {
3          // Se llama a la función num_dias para imprimir cantidad de días
4          // En el año 2008 (en este ejemplo)
5          System.out.println(num_dias(2008));
6      }
7
8      public static boolean bisiesto(int year) {
9          /*
10         Se declaran las condiciones para que
11         el año sea bisiesto
12          */
13         if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0){
14             // Si se cumplen se retorna true
15             return true;
16         } else {
17             // Si no se retorna false
18             return false;
19         }
20     }
21
22     public static int num_dias(int year){
23         // Aquí usamos la función bisiesto para el condicional
24         if (bisiesto(year)) {
25             // Si el año es bisiesto se retorna 366
26             return 366;
27         } else {
28             // Si el año no es bisiesto se retorna 365
29             return 365;
30         }
31     }
32 }
```

4. Ejercicios

Ejercicio 4.1. Escriba un programa en que se utilice una función *esPrimo* para chequear si el número es primo. Adicionalmente, se debe seguir pidiendo que el usuario ingrese inputs hasta que escriba "x".

Observación 4.1. La función debe retornar un boolean.

Ejercicio 4.2. Definir una función *cuadrática* que reciba tres enteros a, b y c correspondientes a los coeficientes de la ecuación

$$ax^2 + bx + c = 0 \tag{1}$$

y que imprima "no hay solución real"si las soluciones son imaginarias o imprima las soluciones en caso de que estas sean reales.