ANyP, Práctica 1d.

Elementos de programación Python: Programación modular.

Introducción a las funciones en Python.

Esta práctica tiene por finalidad familiarizarse con la programación modular, efectuada a través de funciones. En las prácticas aneriores hemos utilizado algunas funciones intrínsecas provistas por Python. En esta, usaremos funciones generadas por nosotros.

Diremos que una una función es un dispositivo que agrupa un conjunto de instrucciones para que puedan ejecutarse más de una vez en un programa. Las funciones también pueden calcular un valor de resultado y permitirnos especificar parámetros que sirven como entradas de la función, que pueden diferir cada vez que se ejecuta el código. Codificar una operación como una función la convierte en una herramienta generalmente útil, que podemos usar en una gran variedad de contextos.
Las funciones son la estructura de programa más básica que Python proporciona para maximizar la reutilización del código y minimizar su redundancia, permitiéndonos dividir programas complejos en partes más manejables.

Definiendo funciones.

En Python, se comienza definiendo una función usando la instrucción `def`. Luego especificamos el nombre de la función y la/s variable/s de entrada; si son más de una, las separamos con comas. Cerramos la línea de la definición con el operador de corte (`:`). A continuación, indentado, sigue el bloque de código con el proceso que deseamos realice la función. El resultado generalmente se da a conocer al programa que la invocó a través de la palabra clave `return`; aunque esto no es obligatorio. De hecho, esta instrucción puede no estar presente, o pueden definirse funciones que tengan más de una instrucción return. En el primer caso, el control es retornado al código invocante al finalizar la ejecución de todas las instrucciones del bloque. En el segundo caso, el control al código que invoca a la función retorna inmediatamente cuando la primera es alcanzada. Mencionamos finalmente que la instrucción return permite devolver múltiples objetos, característica muy útil del lenguaje.

La estructura general de una función es la siguiente:

```
def nombre de función( var_1, var_2, ... ):
  instrucción 1
  instrucción 2
  ...
  instrucción m
  return resultados
```

Como dijimos, *resultados* puede ser más de una variable. En ese caso, las variables a devolver simplemente se separan con comas (`,`) de modo tal que forman una tupla. En el módulo que invoca la función podemos acceder a los distintos elementos de esta tupla.

Ejercicio 1. Comencemos por crear una función muy sencilla que no requiera argumentos ni el uso de la sentencia `return`. El objetivo del ejercicio es practicar cómo se programa y cómo se invoca a una función. Cree una función llamada `saludar` que imprima "`Hola, Mundo`" en pantalla.

Ejercicio 2. Podemos agregar cierta generalidad a la función si la dotamos de argumentos. Cree una función `saludar_persona(nombre)` que reciba un nombre y lo salude.

Ejercicio 3. En los cálculos matemáticos es habitual que se requiera que una función devuelva uno o más valores al programa que la invocó, esto se consigue con la instrucción `return`. Cree una función `area_circulo(r)` que reciba el radio de un círculo y retorne su área.

Ejercicio 4. Una función en Python puede retornar más de un resultado, usando una tupla. Modifique la función anterior para que calcule, y retorne al programa que la invocó, el área y el perímetro de un círculo de radio r.

Ejercicio 5. En el plano, una rotación de ángulo θ alrededor del origen transforma las coordenadas (x, y) de un punto en nuevas coordenadas (x', y') dadas por,

$$\begin{cases} x' = x\cos\theta + y\text{sen}\,\theta, \\ y' = -x\text{sen}\,\theta + y\cos\theta. \end{cases}$$

Implementar una función que realice tal rotación.

Ejercicio 6. Implemente una función para convertir la temperatura de la escala Fahrenheit a la escala Celsius y viceversa.

Ejercicio 7. Los números de Fibonacci son una secuencia de números enteros en la que cada número es la suma de los dos anteriores, comenzando con 0 y 1. Cree una función `fibonacci(n)` que genere y muestre en pantalla los primeros n números de la secuencia. Pruébelo para $n = 20$.

Ejercicio 8. Implemente una función que calcule el factorial de un número entero ($n!$). Implemente la solución a través de un bucle que realice el cómputo

$$n \cdot (n-1) \cdot (n-2) ... \cdot 1$$

En caso de recibir un entero negativo como argumento, haga que la función retorne mostrando un mensaje de error.

Ejercicio 9. Implemente una función que calcule el factorial de un número entero ($n!$). En esta ocasión, implemente la solución en forma recursiva, teniendo en cuenta que:

$$n! = \begin{cases} 1 & \text{si } n = 0, \\ n(n-1)! & \text{si } n > 0. \end{cases}$$

En caso de recibir un entero negativo como argumento, haga que la función retorne mostrando un mensaje de error. Utilice la función para determinar el factorial de los 10 primeros enteros positivos.

Comparación de las versiones iterativa y recursiva.

- *Iterativa*: más eficiente en cuanto a uso de memoria, ya que no requiere mantener múltiples llamadas en la pila.
- *Recursiva*: más elegante y compacta, pero podría causar problemas de límite de recursión con números grandes.

Ejercicio 10. Dado dos enteros no negativos n y r con $n \geq r$ implementar una función para el cálculo del coeficiente binomial

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}.$$

Testee la función calculando $\binom{1}{0}$, $\binom{4}{2}$ y $\binom{25}{11}$.

Ejercicio 11. Escribir una función que devuelva un valor lógico (esto es, verdadero o falso) según un número entero positivo dado es primo o no.

Observación: Recordar que un entero positivo p es primo si es divisible sólo por 1, p, -1 y $-p$, es decir que posee *cuatro* divisores (1 no es primo ya que sólo posee *dos*).

El método más simple para determinar si p es primo consiste en verificar si no es divisible por todos los números sucesivos de 2 a $p-1$. Debido a que el único primo par es el 2, se puede mejorar el método separando la verificación de la divisibilidad por 2 y luego, si p no es divisible por 2, testear la divisibilidad por los enteros *impares* existentes entre 3 y $p-1$.

a) Utilice la función para determinar los N primeros números primos.

b) Utilice la función para imprimir los factores primos de un entero positivo dado.

c) Mejore la rapidez de la función notando que podemos terminar el testeo de la divisibilidad no en $p-1$ sino antes, en $[\sqrt{p}]$, ya que si hay un factor mayor que $[\sqrt{p}]$ entonces existe un factor menor que $[\sqrt{p}]$, el cual ya ha sido revisado. Esta modificación ¿afecta a la escritura de los programas anteriores? ¿Y a la ejecución de los mismos?

Ejercicio 12. Las funciones pueden recibir como argumento el nombre de otras funciones. Defina una función `calcular(f,x)` que reciba una función `f` y un valor `x`, y devuelva el resultado de aplicar `f(x)`.

Ejercicio 13. Python posee una alternativa a la definición de funciones convencionales, especialmente útil para funciones simples, llamada funciones lambda. Defina una función lambda para crear una función que calcule el doble de un número.

Ejercicio 14. Use una función lambda para crear una función cuadrática de la forma $f(x) = a\,x^2 + b\,x + c$. Permita al usuario especificar a, b y c, y evalúe la función en un valor dado de x.

Alcance de variables.

Los siguientes ejercicios permiten ejercitar el alcance de las variables (*global*, *local* y *no local*) en funciones creadas por el usuario en Python.

Ejercicio 15. Cree una función que declare una variable local y devuelva su valor. Muestre que dicha variable no es accesible fuera de la función.

Ejercicio 16. Use una variable global dentro de una función y modifique su valor. Muestre que el valor efectivamente cambió fuera de la función.

Ejercicio 17. Declare una variable local con el mismo nombre que una global y observe cómo interactúan.

Ejercicio 18. Declare una función dentro de otra y use `nonlocal` para modificar la variable de la función envolvente.

Ejercicio 19. Pase una variable global como argumento a una función, modifíquela dentro de la misma y observe cómo se comporta su valor cuando el control vuelve al programa que la invocó.

Ejercicio 20. a) Escriba una función para convertir coordenadas cartesianas a esféricas. Prestar atención a la correcta determinación del cuadrante.

b) Incorporarla en un código que lea (x, y, z) e imprima (ρ, θ, ϕ).