Introducción a la Programación

Práctica 4 – Funciones

Versión del 31 de agosto de 2020

1. Funciones

Una función es un conjunto encapsulado de sentencias que se agrupan bajo un nombre determinado. Una función permite usar una misma sección de código repetidas veces sólo invocando su nombre. Sin embargo, las funciones tienen un propósito aún más importante. En el desarrollo de un gran proyecto de software las funciones permiten a los programadores dividir el código en pequeñas unidades para poder trabajar independientemente. Crear una nueva función brinda una oportunidad de dar un nombre a un grupo de sentencias. Las funciones pueden simplificar un programa al esconder un cómputo complejo detrás de un comando simple, y usando una frase en castellano en lugar de código complicado. Crear una nueva función puede hacer un programa más corto al eliminar código repetitivo.
Podemos inventar (casi) cualquier nombre que queramos para nuestra función, al igual que lo hacemos con las variables. La lista de parámetros especifica que información hay que proveer, si es que la hay, para poder usar (o **llamar**) la nueva función.

Las escribimos de esta forma:

```
def NOMBRE( LISTA DE PARAMETROS ):
    SENTENCIAS
```

Se puede incluir cualquier número de sentencias dentro de la función, pero todas deben escribirse con sangría a partir del margen izquierdo. Al igual que un *for* o un *while*.

Las primeras dos funciones que vamos a escribir no tienen parámetros, por lo que la sintaxis se ve así:

```
def mostrarGuion():
    print("-", end="")
```

Algunas de las funciones preincorporadas que hemos usado tienen **parámetros**, que son valores que se le proveen para que puedan hacer su trabajo. Por ejemplo, si queremos encontrar el seno de un número, tenemos que indicar de qué número. Por ello, `sin` toma un valor como parámetro. Para imprimir una cadena, hay que proveer la cadena, y es por eso que `print` toma una cadena como parámetro. Algunas funciones toman más de un parámetro, como `math.pow`, la cuál toma dos números, la base y el exponente, y devuelve el resultado de elevar la base a la potencia indicada por el exponente.

Cuando definamos nuestras propias funciones, la lista de parámetros indica cuántos parámetros utiliza. Por ejemplo:

```
def imprimirDosVeces(unaCadena):
    print(unaCadena)
    print(unaCadena)
```

Esta función toma un sólo parámetro, llamado `unaCadena`. Cualquiera sea ese parámetro (y en este punto no tenemos idea cuál es), es impreso en pantalla dos veces.

Para llamar esta función, tenemos que proveer una cadena. Por ejemplo, podríamos tener un programa como este:

```
imprimirDosVeces("No me hagas decirlo dos veces!")
```

La cadena que proporcionamos se denomina **argumento**, y decimos que el argumento es **pasado** a la función.

Alternativamente, si tuviéramos una cadena almacenada en una variable, podríamos usarla como un argumento en vez de lo anterior:

```
argumento = "Nunca digas nunca."
imprimirDosVeces (argumento)
```

> **El nombre de la variable que pasamos como argumento no tiene nada que ver con el nombre del parámetro.**

Pueden ser el mismo o pueden ser diferentes, pero es importante darse cuenta que no son la misma cosa, simplemente sucede que tienen el mismo valor.
Una variable local a en una función no existe en las otras funciones del programa, y no está relacionada con las variables del mismo nombre declaradas en otras funciones. Las variables locales son creadas cuando se invoca la función y desaparecen cuando la función termina. La funciones están aisladas del resto del programa: ninguna variable local sobrevive cuando la función termina. Debido a este hecho se dice que las funciones son segmentos aislados y encapsulados de código.

La forma mas directa para obtener valores de una función es usar la instrucción *return*. El valor que la función devolverá se especifica con la instrucción: *return expresion*, donde *expresion* puede ser cualquier expresión válida en Python, por ejemplo, $y - (x + 7)/3$ o $x == 2 * y$. La expresión también puede ser simplemente una variable o un valor constante. La instrucción *return* se puede escribir en cualquier parte dentro de la función, no solamente al final de la misma. Sin embargo, esta instrucción siempre indica la terminación de la función y regresa el control al proceso invocante. Una función devuelve un valor al ser llamada, como si el nombre de la función fuera una variable que contiene un valor.

Una función en computación se considera *pura* si **siempre** que se la llame con parámetros adecuados retorna algún valor y nunca tiene efectos colaterales (como imprimir cosas en la pantalla). En Introducción a la Programación nos interesa crear principalmente funciones puras. En computación (y nosotros también de ahora en adelante) diremos **llamar** una función cada vez que usamos una función.

En Python, si queremos definir nuestra propia función, lo hacemos de esta forma:

```
def NOMBRE( LISTA DE PARAMETROS ):
    SENTENCIAS
```

Notas preliminares

- Los ejercicios marcados con el símbolo ★ constituyen un subconjunto mínimo de ejercitación. Sin embargo, aconsejamos fuertemente hacer todos los ejercicios. Aclaración: Cuando un ejercicio dice "hacer una función que tome ciertos parámetros y devuelva cierto valor", quiere decir que hay que definir una función. Cuando un ejercicio dice "Hacer un programa que pida al usuario tales valores y muestre en pantalla un resultado", quiere decir que hay que hacer un programa como los que estamos acostumbrados, solo que ahora podemos utilizar funciones existenes o definir las nuestras si resulta conveniente.

Ejercicio 1 ★

Hacer un programa para cada inciso que pida al usuario un número decimal x y muestre por pantalla el resultado de evaluar las siguientes fórmulas:

a) $\sqrt{x}$
b) $|x|$
c) $|x-3|$
d) $\sqrt{|x-5|}$

Ejercicio 2 ★

Teniendo estas definiciones de funciones:

```
def cuak():
    chan()
    print("pienso que ", end="")
    chan()

def chan():
    print ("yo", end="")
    plin ()

def plin():
    print(".")
```

Indicar qué se imprime en pantalla luego de ejecutar este programa:

```
print ("No, yo ", end="")
cuak()
print ("Yo ", end="")
chan()
```

AYUDA: Empezar describiendo con palabras qué hacen `chan` y `cuak` cuando se las llama.
REGLA: No vale correrlo en la computadora.

Ejercicio 3 ★

a) Escribir una función que reciba como parámetro una cadena y la muestre en pantalla 3 veces.
b) Guardar esta definición de función en un archivo.
c) Hacer un programa que le pida al usuario una cadena y que la muestre en pantalla 3 veces utilizando la función definida anteriormente.

Ejercicio 4 ★

a) Escribir una función que reciba dos números reales como parámetros y retorne su promedio.
b) Hacer un programa que pida al usuario dos números reales y muestre por pantalla el resultado de llamar a la función del primer inciso.
c) Idem de los dos anteriores pero con tres números. Escribir la función en el mismo archivo donde se escribió la del item a.

Ejercicio 5 ★

Definir una función que devuelva el valor absoluto de un número. (Hacerlo sin utilizar la función *abs*)

Ejercicio 6 ★

a) Escribir una función con el siguiente encabezado: `def exclamar(unaCadena):` que retorne la misma cadena entre símbolos de exclamación (`¡!`)
b) Escribir una función con el siguiente encabezado: `def gritar(unaCadena):` que retorne la misma cadena entre 3 símbolos de exclamación (`¡¡¡!!!`)
c) De no haberlo hecho en el punto anterior, escribir de nuevo la función `gritar` utilizando solo la función `exclamar`.

Nota: `gritar("Ouch")` deberá devolver la cadena `"¡¡¡Ouch!!!"`
Ayuda: Recordar que + utilizado entre cadenas las concatena.

Ejercicio 7 ★

a) Escribir una función que se llame `elevarAlCubo` que tome un número y retorne el valor de ese número al cubo.
b) Guardar el ejercicio anterior en un archivo llamado `funcionCubo.py`
c) Correr el siguiente código en un archivo nuevo y chequear que los resultados sean correctos:

```
print(0, “al cubo:”, elevarAlCubo(0))
print(1, “al cubo:”, elevarAlCubo(1))
print(2, “al cubo:”, elevarAlCubo(2))
print(3, “al cubo:”, elevarAlCubo(3))
print(4, “al cubo:”, elevarAlCubo(4))
print(5, “al cubo:”, elevarAlCubo(5))
print(6, “al cubo:”, elevarAlCubo(6))
print(-1, “al cubo:”, elevarAlCubo(-1))
print(-2, “al cubo:”, elevarAlCubo(-2))
print(-3, “al cubo:”, elevarAlCubo(-3))
print(-4, “al cubo:”, elevarAlCubo(-4))
print(-5, “al cubo:”, elevarAlCubo(-5))
```

Ejercicio 8 ★

a) Escribir una función que tome un parámetro de tipo entero y devuelva la cantidad de divisores positivos de ese número.
b) Escribir una función que tome un parámetro de tipo entero y devuelva el valor `True` si se la llama con un número primo y `False` en caso contrario.
c) ¿Cuál es el número primo más grande que encontraste?
d) Hacer una función (no pura) que reciba un entero e imprima sus factores primos. Por ejemplo para $a = 20$ la función debe mostrar 2 y 5.

   Nota: Los números primos son aquellos cuyos únicos divisores positivos son ellos mismos y el 1.

Ejercicio 9

a) Hacer una función que reciba dos enteros y retorne el mayor.
b) Hacer una función que reciba tres enteros y retorne el mayor.

Ejercicio 10

Hacer una función *potencia*, que reciba dos enteros a y b y retorne a^b.

Ejercicio 11

a) Hacer una función que sume los divisores propios de un número.
b) Hacer una función que indique si un número es *perfecto*. Número perfecto: a es perfecto si la suma de sus divisores propios es igual a a.
c) Hacer una función que determine si un número ingresado por el usuario es un *número abundante*. Número abundante: todo número natural que cumple que la suma de sus divisores propios es mayor que el propio número. Por ejemplo, 12 es abundante ya que sus divisores son 1, 2, 3, 4 y 6 y se cumple que 1+2+3+4+6=16, que es mayor al propio 12.

Ejercicio 12

Hacer una función que indique si un número es Poderoso: Número poderoso: es un número natural n que cumple que todos sus divisores primos al cuadrado también son divisores, es decir, si p es un divisor primo entonces p^2 también lo es. Por ejemplo, el número 36 es un número poderoso ya que los únicos primos que son divisores suyos son 2 y 3 y se cumple que 4 y 9 también son divisores de 36.

Ejercicio 13

Hacer una función que indique si un número es Libre de Cuadrados: Número libre de cuadrados: todo número natural que cumple que en su descomposición en factores primos no aparece ningún factor repetido. Por ejemplo, el número 30 es un número libre de cuadrados.

Ejercicio 14

Hacer un programa que solicite al usuario un número entero positivo e indique cuál es el número primo mayor más cercano. Usar funciones. Por ejemplo, si el usuario ingresa 24, el programa devolverá 29 (29 es el número primo más cercano mayor que 24). Si el usuario ingresa 5 el programa devolverá 7.

Ejercicio 15

Hacer una función (no pura) que reciba una palabra y la imprima recuadrada por asteriscos. Por ejemplo si la cadena fuera sobrevivir, la función debería imprimir

```
**************
* sobrevivir *
**************
```

Ejercicio 16

El propósito de este ejercicio es tomar el código de un ejercicio anterior y encapsular una parte de él en un función que toma parámetros. Partiendo de la resolución del ejercicio 25 de la práctica de ciclos y cadenas,

a) escribir una función que tome como parámetros una cadena y una letra, y retorne la cantidad de veces que aparece esa letra en esa cadena.
b) probarla para asegurarse que funciona bien.
c) transformar el código del ejercicio 25 para que utilice la nueva función.

Ejercicio 17 ★

Escribir una función (y probarla en un programa) para cada una de las siguientes descripciones:

a) una función que se llame `tieneRepetidas` que tome una cadena como parámetro y devuelva `True` si esa cadena tiene alguna letra que aparece más de una vez y `False` en caso contrario.
b) una función que se llame `aparece` que tome dos parámetros, una letra y una cadena, y devuelva `True` si la letra aparece en la cadena y `False` en caso contrario.
c) una función que se llame `dameRepetida` que tome una cadena como parámetro y retorne la primer letra que aparece repetida en la cadena
d) una función que se llame `quitarRepeticiones` que tome dos parámetros, una cadena y una letra, y devuelva otra cadena igual a la anterior pero sin las repeticiones de esa letra. Por ejemplo, un programa que llame a la función así: `quitarRepeticiones("mate cocido", "c")`, deverá retornar la cadena `"mate coido"`.

Ejercicio 18 ★

Se desea automatizar el cálculo de la tarifa telefónica para una lista de clientes. La empresa informa que cada llamado tiene un costo por conexión más un costo por el tiempo consumido en segundos. Se cuenta con las siguientes funciones ya implementadas.
`ObtenerCantidadLlamados(nroCliente)`: retorna la cantidad de llamados para un cliente.
`ObtenerTiempoPorLlamada(nroCliente, nroLlamada)`: retorna la cantidad de segundos de un llamado de un cliente.
`ObtenerCostoPorLlamada()`: retorna el costo fijo por cada llamada.
`ObtenerCostoPorTiempo(seg)`: retorna el costo de una llamada que dura `seg` segundos.
Realizar un programa que indique el monto de la factura para cada cliente. Además, el sistema registra los clientes en una base para promociones, con el siguiente comportamiento. Los clientes que ya estaban en la base, se los da de baja para evitar volver a ofrecer dos veces seguidas la promoción. Si el tiempo total de comunicaciones para un cliente supera las dos horas, se lo agrega a la base para ofrecerle la promoción, salvo los dados de baja previamente. Para ello se cuenta con las siguientes funciones ya implementadas.
`Esta(nroCliente)`: indica si el cliente está en la base de promociones.
`Alta(nroCliente)`: agrega el cliente a la base.
`Baja(nroCliente)`: da de baja al cliente en el registro de promociones.