El futuro digital es de todos

MinTIC

Universidad Industrial de Santander

MisiónTIC2022

FUNCIONES

INTRODUCCIÓN

En matemáticas es normal el uso de funciones como son los casos de las funciones polinomios y las funciones trigonométricas. Estas funciones poseen una variable dependiente que evalúa su resultado mediante una variable independiente. De igual manera que como se definen funciones en matemáticas, son definidas funciones en cada uno de los lenguajes de programación, algunas de estas funciones vienen predeterminadas y otras son construidas por cada programador de acuerdo al problema que quiera resolver.

Una función "*es un bloque de código que tiene asociado un nombre y que recibe o no una serie de argumentos como entrada, para luego seguir una serie de instrucciones que tienen como finalidad llevar a cabo una tarea específica y retornar un valor*". John Snowden (2020).

Want to Learn Python The Right and Simple Way. Independently published). Como ejemplos de funciones tenemos realizar la suma de dos números, contar las palabras de una cadena de caracteres, etc. Las funciones permiten dividir un código u aplicación en pequeñas tareas que hace parte de la solución a un problema más grande. El hecho de construir funciones tiene varias ventajas, las cuales son:

- Te permite asignar un nombre a un conjunto de órdenes o instrucciones, lo que hace que el programa sea más fácil de leer, entender y depurar.

•- Hacen el programa más pequeño ya que se elimina código repetido.

- Dividir por funciones un programa ayuda en la depuración de programas largos, ya que se puede depurar por funciones.

•- Una función puede ser útil para el desarrollo de otros programas.

Como podemos ver dentro del contexto de programación, las funciones cumplen un papel muy importante, facilitándonos el desarrollo de programas o aplicaciones. Es por eso, que en este curso aprenderemos cómo desarrollarlas y cómo usarlas.

El futuro digital es de todos

MinTIC

COMPOSICIÓN DE UNA FUNCIÓN (NOMBRE, ARGUMENTOS, SECUENCIA DE INSTRUCCIONES Y RETORNO)

Universidad Industrial de Santander

MisionTIC2022

En cualquier tipo de lenguaje de programación se pueden definir funciones de varios tipos. Estas funciones son definidas usando una estructura básica, la cual se encuentra compuesta por los siguientes elementos:

COMPOSICIÓN DE UNA FUNCIÓN			
Nombre: Es un identificador que permite identificar la función. Este nombre puede estar asociado a la tarea que realiza la función; como por ejemplo, si la función es una función que realiza la suma de dos números, entonces lo más lógico es que esta se llame suma, de esta manera quien lea el código sabrá cual es la tarea de dicha función.	•**Argumentos**: son parámetros que recibe la función y son necesarios para llevar a cabo la tarea para la cual fue creada. Un ejemplo puede ser también la función que lleva a cabo la suma de dos números. Para realizar tal operación se requiere como parámetros dos números, los cuales se van a sumar.	**Secuencia de instrucciones**: Además del nombre que identifica la función y sus argumentos, se requiere construir un algoritmo o secuencia de instrucciones que permita resolver la tarea que se requiere que lleve a cabo la función.	•**El retorno**: El retorno de la función son los datos que se generan después de procesar los datos de entrada de la función. Puede que una función no reciba datos de entrada o argumentos y todas las variables necesarias para resolver una tarea sean definidas dentro de la función, en este caso dicha función realiza un proceso interno y puede también retornar determinados datos de salida.

Un ejemplo de los elementos de una función se puede ver a continuación:

```
define <nombre de función>(<argumento 1>,
<argumento 2>, …) {
    <secuencia de instrucciones>
    retorna <resultado>
}
```

En este ejemplo, que es una estructura general, se puede observar que la función se crea con un nombre dado por la persona que programa la función, definida como `<nombre de función>`. Esta asignación de nombre va seguida del paso de todos los argumentos que se requieren dentro de la función, también llamados argumentos de entrada, `<argumento 1>, <argumento 2>, ….` Ya dentro de la función se colocan todas las instrucciones necesarias para el procesamiento de todos los argumentos de entrada, en el proque identificado como `<secuencia de instrucciones>`, para finalmente retornar un resultado en <resultado>. Esta función de ejemplo recibe parámetros de entrada y retorna unos datos, pero una función también puede no recibir parámetros de entrada o retornar datos. Un ejemplo de una función que no retorne un valor sería una función que reciba datos de entrada y simplemente imprima unos datos de salida.

2.1. Ciclo de vida de una función

Las funciones son secuencias de instrucciones que como vimos anteriormente poseen una estructura definida, ellas son definidas y son implementadas en donde se requiera a lo largo del desarrollo de un código o aplicación. Para ser usadas, ellas deben recibir o no un conjunto de parámetros de entrada, los cuales son procesados internamente y como resultado es retornado o no un resultado para el cual fue creada.

Una función tiene un ciclo de vida bien definido, el cual empieza cuando es declarada y continúa con su posterior uso para finalmente morir. Luego de que una función muere, esta puede ser reutilizada las veces que sea necesaria.

El futuro digital es de todos

MinTIC

FUNCIONES Y MÓDULOS PREDEFINIDOS

Universidad Industrial de Santander

MisiónTIC2022

3.1. Funciones predefinidas

Uno como usuario de un lenguaje de programación, puede definir cualquier función que se desee; pero, para facilitarnos un poco la vida existen funciones predefinidas, lo que quiere decir que ya vienen incorporadas con el lenguaje de programación; por ejemplo, en el caso del lenguaje de programación Python, existe lo que son funciones predefinidas y métodos. Los métodos también son funciones, pero difieren de las funciones predefinidas en que ellos no pueden existir por sí solos, ya que están asociados a determinado objeto o tipo de dato, de tal manera que operan sobre ellos. Tanto las funciones predefinidas como los métodos, facilitan el manejo y el procesamiento de algunos tipos de datos.

Como ejemplos, a continuación se muestran algunas funciones predefinidas en Python.

3.1.1. Función type()

La función type es una función que retorna el tipo de clase de un objeto que es pasado como argumento.

Sintaxis:

```
type(objeto)
```

Parámetros:

objeto: objeto al cual se le desee conocer el tipo de clase.

3.1.2. Función max()

La función max es una función que retorna el máximo valor de una serie de parámetros de igual tipo o de una lista.

Sintaxis:

```
max(a,b,c,d,…)
```

Parámetros:

a,b,c,…: datos del mismo tipo.

3.1.3. Función sorted()

La función sorted es una función que retorna una lista ordenada de datos a partir de una datos en una variable iterable.

Sintaxis:

```
sorted(objeto)
```

Parámetros:

objeto: datos de entrada de tipo variable iterable.

3.1.4. Función print()

La función print es una función que imprime el objeto dado en la salida estándar del dispositivo.

Sintaxis:

```
print(objeto)
```

Parámetros:

objeto: objeto para imprimir.

3.1.5. Función len()

La función len es una función que retorna el número de elementos en un objeto.

Sintaxis:

```
len(objeto)
```

Parámetros:

objeto: secuencia de datos (cadena de caracteres, tuplas, diccionarios o conjuntos).

3.1.6. Función input()

La función input es una función que permite obtener texto introducido usando el teclado.

Sintaxis:

```
variable = input(prompt)
```

Parámetros:

prompt: es el mensaje que será mostrado antes de que se ingrese alguna información por medio del teclado. Esa información es almacenada dentro de la variable luego de recibir una orden emitida al presionar la tecla Enter.

3.2. Módulos

En la medida que el desarrollo de una aplicación va creciendo, se hace más difícil poder realizar tareas de depuración del código. Por este motivo, es necesario que el desarrollo sea dividido en tareas más pequeñas que pueden estar estructuradas como funciones. Adicionalmente, se requiere que estas funciones puedan ser accedidas desde cualquier parte del desarrollo desde donde sea necesario su uso.

Para acceder a funciones desde cualquier parte de un código se crean ficheros a los cuales se les llaman módulos. Estos módulos son archivos de Python que permiten enlazar y hacer referencia. Estos archivos contienen funciones y estructuras más abstractas como son las clases, un tema que no será tocado en este curso ya que está enfocado a introducir al estudiante en temas de programación en Python. Una ventaja de la construcción de módulos dentro del desarrollo, es que al igual que las funciones, facilita la lectura, el entendimiento y la depuración. Para ser usado en un código, un módulo requiere de su importación, lo cual es logrado mediante la palabra reservada `import`.

El modo de uso de `import` es el siguiente:

```
>>> import <nombre del modulo>
```

donde `<nombre del módulo>` es el nombre que se le da al fichero. Para usar algún elemento dentro del módulo importado se hace mediante el operador punto de la siguiente manera:

```
>>> <nombre del modulo>.<elemento>
```

donde `<elemento>` es el elemento que queremos usar.

Otra manera de importar los módulos es asignándoles otro nombre diferente al nombre que se le da al fichero, esto se hace de la siguiente manera usando la palabra reservada as:

```
>>> import <nombre del modulo> as <otro nombre>
```

donde `<otro nombre>` es el nombre por medio del cual queremos usar el módulo. En este caso el uso de algún elemento dentro del módulo se haría de la siguiente manera, usando el operador punto:

```
>>> <otro nombre>.<elemento>
```

También es posible importar de manera específica un elemento que se encuentre dentro de un módulo, esto se hace de la siguiente manera usando la palabra reservada `from`:

```
>>> from <nombre del modulo> import <elemento>
```

donde `<elemento>` es un elemento dentro del módulo que puede ser, por ejemplo, una función.
A estos elementos importados desde dentro de un módulo también se les puede llamar con otro nombre que queramos, usando la palabra reservada as, esto de la siguiente manera:

```
>>> from <nombre del modulo> import <elemento> as <nuevo nombre>
```

donde `<nuevo nombre>` es el nombre con el cual será usado el elemento importado.
Además, se pueden importar varios elementos o todos los elementos dentro de un módulo, esto de la siguiente manera:

Para importar varios elementos se usa la instrucción:

```
>>> from <nombre del modulo> import <elemento 1>, <elemento 2>, …
```

donde `<elemento 1>`,`<elemento 2>`, etc, son los elementos que se desean importar.

Para importar todos los elementos se usa la instrucción:

```
>>> from <nombre del modulo> import *
```

No se recomienda esta manera para el importe de todos los elementos dentro de un módulo ya que puede suceder que se importe un módulo con un nombre de un elemento definido dentro del código que se está desarrollando.

3.2.1. Módulos predefinidos

La biblioteca estándar de Python viene con un conjunto de módulos, cada uno de ellos desarrollados para llevar a cabo diferentes tipos de tareas. A continuación se muestra una lista de alguno de estos módulos y sus funciones.

3.2.1.1. Módulo os

Este módulo provee funciones que permiten al desarrollador interactuar con el sistema operativo; como por ejemplo, este módulo tiene la función getcwd() que permite conocer la ruta del directorio en el cual te encuentras.

```
>>> import os
>>> os.getcwd()
>>> /ruta/del/directorio/donde/te/encuentras/
```

Este módulo posee también un submódulo, `path`, que permite acceder a funcionalidades como conocer si una ruta es un archivo, `os.path.isfile()`, o si la ruta es un directorio, `os.path.isdir()`.

Conocer más del módulo os.
Conocer más del submódulo path.

3.2.1.2. Módulo glob

Este módulo permite encontrar todos los nombres de rutas que coincidan a cierto patrón específico

3.2.1.3. Módulo sys

Este módulo permite tener acceso a variables y funciones relacionadas con el intérprete de Python; como por ejemplo, las variables relacionadas con los parámetros que se le pasan a un código en Python por línea de comando. Los códigos en Python son ejecutados por medio de la siguiente línea de comando:

```
>> python archivo.py
```

donde adicionalmente se le pueden pasar argumentos de la siguiente manera:

```
>>> python archivo.py arg1 arg2 arg3
```

Estos argumentos, `archivo.py`, `arg1`, `arg2` y `arg3`, son almacenados en una variable, la cual se puede acceder desde dentro del código como:

```
import sys

argumentos = sys.argv
print(argumentos)
```

Conocer más del módulo sys.

3.2.1.4. Módulo math

Este módulo permite el acceso a funciones matemáticas dentro de la biblioteca de C. Por ejemplo, se requiere el uso de las funciones trigonométricas. Estas se pueden acceder de la siguiente manera:

```
>>> from math import sin, cos, tan, pi
>>> sin(pi/3)
>>> 0.866025403784386
>>> cos(pi/3)
>>> 0.5000000000000001
>>> tan(pi/4)
>>> 0.9999999999999999
```

Se pueden emplear las funciones `sin, cos, tan`, entre otras y además se puede emplear el número `pi`. Existen muchos otros tipos de funciones dentro de este módulo que puedes explorar; como por ejemplo, la función raíz cuadrada (`sqrt`), exponenciación (`exp`), valor absoluto (`fabs`), entre otras.

Conocer más del módulo math.

3.2.1.5. Módulo random

Este módulo provee herramientas para la generación de elementos aleatorios. Para generar un número aleatorio se hace de la siguiente manera:

Si se desea obtener un valor aleatorio tipo entero entre dos valores a y b, se hace usando la orden `random.randint(a, b)`.
Si se desea obtener un valor flotante aleatorio entre dos valores a y b se puede hacer usando la orden `random.uniform(a, b)`.

Si se desea elegir de manera aleatoria un ítem de una lista de ítems, se puede hacer usando la orden `random.choice(lista)`.

Existen otras funciones que se pueden explorar y usar, y que se encuentran dentro de este módulo.

Conocer más del módulo random.

3.2.1.6. Módulo statistic

Este módulo contiene elementos que permiten calcular propiedades de estadística básica. Este módulo contiene funciones como mean() para determinar la media de un conjunto de datos, la función median() que permite determinar el valor medio de unos datos, la función mode() que permite determinar la moda dentro de un conjunto de datos, entre otras funciones.

Conocer más del módulo statistics.

3.2.1.7. Módulo datetime

Este módulo proporciona los elementos necesarios para la manipulación de fechas y horas. Si se requiere conocer cuál es la fecha actual se puede hacer de la siguiente manera:

```
>>> import datetime
>>> fecha = datetime.date.today()
>>> fecha
>>> datetime.date(2021, 4, 27)
```

Se puede acceder a el día, el mes y el año, de la siguiente manera:

```
>>> fecha.day
>>> 27
>>> fecha.month
>>> 4
>>> fecha.year
>>> 2021
```

Si se desea conocer cuál es la fecha y hora se puede hacer de la siguiente manera:

```
>>> import datetime
>>> fecha_hora = datetime.datetime.now()
>>> fecha_hora
>>> datetime.datetime(2021, 4, 27, 19, 19, 35, 472337)
```

Se puede acceder al día, el mes, el año, la hora, los minutos y los segundos, de la siguiente manera:

```
>>> fecha_hora.day
>>> 27
>>> fecha_hora.month
>>> 4
>>> fecha_hora.year
>>> 2021
>>> fecha_hora.hour
>>> 19
>>> fecha_hora.minute
>>> 23
>>> fecha_hora.second
>>> 41
```

Se pueden crear datos de tipo `datetime` mediante la función `date(year, month, day)`, esto de la siguiente manera:

```
>>> import datetime
>>> fecha = datetime.date(2021, 4, 27)
>>> fecha
>>> datetime.date(2021, 4, 27)
```

Adicional a estas funciones, existen otras funciones que se pueden explorar dentro de este módulo.

Conocer más del módulo datetime.

El futuro digital es de todos

MinTIC

FUNCIONES PARA CARACTERES Y CADENAS

Universidad Industrial de Santander

MisiónTIC2022

4.1. Métodos para cadena de caracteres

Los distintos tipos de datos definidos en Python poseen una serie de métodos que permiten manejar estos datos de manera más fácil. Por medio de un tipo de dato se puede crear un objeto, y este objeto por pertenecer a determinado tipo de datos posee métodos o funciones predeterminadas que facilitan su procesamiento; por ejemplo, se tienen los datos tipo cadena de caracteres, los cuales poseen un método que permite buscar un carácter en particular dentro de dicha cadena.
A continuación mostraremos algunos ejemplos de métodos para el manejo de cadena de caracteres.

4.1.1. Método capitalize()

Este método muestra la primera letra de la cadena de caracteres en mayúscula.

Ejemplo:

```
>>> caracteres = 'hola'
>>> caracteres.capitalize()
>>> 'Hola'
```

4.1.2. Método count()

Este método indica cuantas veces se repite un valor dentro de una cadena de caracteres.

Ejemplo:

```
>>> caracteres = 'asombroso'
>>> valor = 'o'
>>> caracteres.count(valor)
>>> 3
```

4.1.3. Método find()

Este método permite buscar un valor específico dentro de una cadena de caracteres y retorna la posición donde fue encontrado.

Ejemplo:

```
>>> caracteres = "Hola a todos, bienvenidos a este curso"
>>> valor = "este"
>>> caracteres.find(valor)
>>> 28
```

El valor retornado es 28 ya que si contamos desde el primer carácter de la línea de caracteres definida, vamos a encontrar que la palabra 'este' se encuentra a partir del carácter 28, donde el primer carácter es el cero.

4.1.4. Método split()

Este método permite dividir una cadena de caracteres dependiendo de un separador, el cual puede ser cualquier carácter o conjunto de caracteres. Si no se le especifica cual es el separador, este es tomado como el espacio en blanco entre conjunto de caracteres o palabras dentro de la cadena de caracteres. Este método retorna una lista de trozos de la cadena de caracteres tomados de acuerdo al separador.

Ejemplo:

```
>>> caracteres = "1 2 3 4 5 6"
>>> caracteres.split()
>>> ['1', '2', '3', '4', '5', '6']
```

Ejemplo:

```
>>> caracteres = "datos.txt"
>>> caracteres.split('.')
>>> ['datos', 'txt']
```

4.1.5. Método format()

Este método permite darle formato a valores específicos e insertarlos en un marcador de posición dentro de una cadena de caracteres. El marcador de posición es definido usando corchetes, `{}`. Este método retorna la cadena de caracteres formateada. En algunas ocasiones, este método es usado para generar salidas.

Ejemplo:

La manera más simple de utilizar este método es la siguiente:

```
>>> cadena = "Hola {}, ¡bienvenido a este curso de {}!"
>>> nombre = "Juan"
>>> curso = "Python"
>>> cadena.format(nombre, curso)
>>> Hola Juan, ¡bienvenido a este curso de Python!
```

Ejemplo:

Dentro de la cadena de caracteres también podemos identificar la posición de cada argumento que se pasa a formar, esto por medio de números, de la siguiente manera:

```
>>> cadena = “Hola {1}, ¡bienvenido a este curso de {0}!”
>>> cadena.format(curso, nombre)
>>> Hola Juan, ¡bienvenido a este curso de Python!
```

En este caso, el primer argumento que es curso debe ir ubicado en `{0}` y el segundo argumento que es nombre debe ir ubicado en `{1}`.

4.1.6. Método join()

Este método permite concatenar un objeto iterable, cuyos elementos son caracteres o cadena de caracteres, con un delimitador definido por el usuario. Este método toma cada uno de los elementos del objeto iterable y los concatena en una cadena de caracteres separando cada elemento con el delimitador definido.

Ejemplo:

Un ejemplo con una cadena de caracteres podría ser:

```
>>> letras = 'abcdefg'
>>> delimitador = ','
>>> delimitador.join(letras)
>>> 'a,b,c,d,e,f,g'
```

Las cadenas de caracteres son objetos iterables y el método `join()` toma cada uno de sus elementos y los concatena en una nueva cadena de caracteres separándolos por el delimitador que en este caso es una coma.

Ejemplo:

Un ejemplo usando una lista es:

```
>>> letras = [‘Hola’, ‘como’, ‘estas’]
>>> delimitador = ‘ ’
>>> delimitador.join(letras)
>>> ‘Hola como estas’
```

En este ejemplo el delimitador es un espacio en blanco. Se toma cada elemento de la lista y es concatenado en una cadena de caracteres separando cada uno de los elementos con un espacio en blanco. De esta misma manera se puede llevar a cabo la concatenación usando conjuntos o tuplas.

4.1.7. Método replace()

Este método permite crear una nueva cadena de caracteres mediante el reemplazo de alguna parte de otra cadena de caracteres.

Ejemplos:

Un ejemplo del uso de este método es el siguiente:

```
>>> frase = 'Los patos vuelan'
>>> frase.replace('vuelan', 'nadan')
>>> 'Los patos nadan'
```

En este caso se genera una nueva cadena de caracteres a partir del reemplazo o modificación de cierta parte de otra cadena de caracteres. En el ejemplo, la frase es 'Los patos vuelan' y se quiere crear otra cadena de caracteres que indique que los patos nadan, por lo que en la frase original se cambia la palabra 'vuelan' por la palabra 'nadan'. El primer parámetro de entrada del método es lo que se quiere modificar y el segundo por qué se va a modificar.

4.1.8. Método casefold()

Este método retorna una nueva cadena de caracteres eliminando todas las distinciones de casos que presente, de tal manera que es usado para hacer comparaciones de cadena de caracteres.

Ejemplo:

Un ejemplo para el uso de este método podría ser:

```
>>> palabra1 = 'Estado'
>>> palabra2 = 'estado'
>>> palabra1.casefold() == palabra2.casefold()
>>> True
```

En este ejemplo se comparan dos cadenas de caracteres. Estas cadenas de caracteres son palabras iguales pero una de ellas contiene una letra en mayúscula y la otra no. El método `casefold()` permite comparar estas dos palabras sin tener en cuenta el hecho de que una de ellas tiene una letra en mayúscula.
Como se puede observar en los ejemplos anteriores de cada uno de los métodos mostrados, estos son llamados haciendo uso de la variable definida en conjunto con el operador punto. Cada uno de estos métodos opera sobre los datos de la variable sin modificarla. Existen muchos otros métodos con diversas funcionalidades que permiten el manejo de cadenas de caracteres.

El futuro digital es de todos

MinTIC

FUNCIONES PSEUDOCÓDIGO

Universidad Industrial de Santander

MisiónTIC2022

Un pseudocódigo permite describir la manera en cómo opera un programa o algoritmo. Las funciones dentro de los algoritmos o códigos se pueden representar por medio de pseudocódigos, los cuales nos permiten describir de manera informal cuales son las funcionalidades que caracterizan a dicha función.

A continuación se muestra un ejemplo del pseudocódigo para una función que permite contar de cada elemento dentro de una cadena de caracteres cuantas veces aparece dicho elemento dentro de la cadena de caracteres, y retorna una lista con tales valores:

```
function count_each_character(input_string)
    string_length ← length of input_string
    counter_array ← empty array of length string_length
    for i ← 0 to i ← string_length - 1 do
        character ← input_string(i)
        counter ← 0
        for j ← 0 to j ← string_length - 1 do
            if input_string(i) == input_string(j) then
                counter ← counter + 1
            end if
        end for
        counter_array(i) ← counter
    end for
    return counter_array
end function
```

La función lleva como nombre `count_each_character` y toma como dato de entrada una cadena de caracteres almacenada en `input_string`. Se define dentro de la función dos variables, una de tipo entero `(string_length)` para almacenar el tamaño de la cadena de caracteres, y la otra es un arreglo `(counter_array)` de tamaño `input_string`. Se procede creando un doble bucle for, ambos iterando desde cero a el valor string_length-1. El primer bucle permite definir cada uno de los caracteres que componen el arreglo de caracteres mediante la variable character, y dentro de este se define un contador (`counter`) que se hace cero justo antes de empezar el segundo bucle for. El segundo bucle for permite nuevamente iterar sobre cada uno de los caracteres que componen la cadena de caracteres, los cuales son comparados con la variable character por medio de un condicional if. Si el condicional se cumple entonces el contador agrega una unidad a su valor, de tal manera que cuando se finaliza el segundo bucle for, el valor de counter es igual al número de veces que el carácter almacenado en character aparece dentro de la cadena de caracteres. Este valor es almacenado en el arreglo `counter_array` de acuerdo a cada posición de los caracteres en la cadena de caracteres.

Al finalizar el segundo ciclo `for` la función habrá hecho esta misma tarea sobre cada carácter de la cadena de caracteres para finalmente retornar el valor almacenado en `counter_array`.

Existen algunas cosas que no se han tenido en cuenta en este pseudocódigo de una función, como es el caso de no contar los caracteres repetidos dentro de la cadena de caracteres. Esto se ha hecho de manera intencional ya que se busca que la lógica sea más sencilla de tal manera que se entienda la idea de cada elemento dentro de la función.

El futuro digital es de todos

MinTIC

FUNCIONES Y ARGUMENTOS

Universidad Industrial de Santander

MisiónTIC2022

6.1. Definición de una función en Python

En Python existe una sintaxis para la definición de funciones. Hemos revisado algunas funciones predeterminadas que vienen dentro del lenguaje pero no hemos visto como estas vienen definidas dentro de las librerías que conforman el lenguaje.

Las funciones predeterminadas que vienen con Python son de bastante utilidad, pero son bastante genéricas. Esto nos indica que para poder realizar alguna tarea más particular; como por ejemplo, conocer la velocidad de un vehículo dada en unidades de **m/seg**. Esta es una tarea que no viene incorporada dentro de una función predeterminada de Python, por lo que es necesario definirla.

Si se genera un programa en donde se requiera realizar esta conversión de unidades de velocidad, y se colocara por ejemplo, la función hipotética kmh_to_ms() esta generaría un error porque no se encuentra definida. Debido a que no se encuentra definida se debe definir para ser implementada dentro del algoritmo o programa.

Una función en Python es definida mediante la palabra reservada def seguida del nombre de la función, para luego entre paréntesis colocar los argumentos que esta recibirá. Al finalizar cada línea de definición de una función se colocan : y en los próximos renglones las líneas de comando que la componen.

Ejemplo:

```
def suma(a,b):

def resta(a,b):
```

donde `suma` y `resta` son los nombres de las funciones definidas, `a` y `b` son sus parámetros de entrada o argumentos.

Una función en Python puede o no finalizar retornando uno o múltiples valores. El retorno de una función se hace usando la palabra reservada `return` seguido del valor a retornar, o los valores a retornar separados por comas.

Ejemplo:

```
return a
```

```
return a, b, c
```

donde a, b, c son los valores a retornar.

Pensemos en la función hipotética anteriormente mencionada, `kms_to_ms()`, que permite transformar unidades de velocidad en **km/h** a unidades en **m/seg**. La conversión es simple, se requiere tener en cuenta que **1 km** son **1000 m** y que **1 h** equivale a **3600 seg**. A partir de esto, mediante regla de tres, tenemos que la conversión es la siguiente:

```
velocidad en m/seg ← (velocidad en km/h) * (1000m / 1km) * (1h / 3600 seg)
```

Entonces definimos la función `kms_to_ms()` como:

```
def kms_to_ms(velocity_kms):
    kilometer_in_meter = 1000
    seconds_in_hour = 3600
    return velocity_kms * kilometer_in_meter / seconds_in_hour
```

Otro ejemplo de funciones puede ser una función que emita un mensaje de acuerdo a un nombre de un usuario y retorna la fecha y hora en la que ese mensaje fue emitido. Esta función recibe **2** parámetros de entrada, uno es una cadena de caracteres donde es almacenado un nombre y el otro es otra cadena de caracteres donde se almacena un mensaje. Dentro de la función se imprime el nombre seguido de dos puntos seguido de el mensaje que se quiere emitir. Esto lo podemos asemejar a un algoritmo para visualizar el mensaje de un usuario de una aplicación de chat. Esta función se puede definir de la siguiente manera:

```
from datetime import datetime


def chat(name, msg):
    date = datetime.now()
    print("{}: {}".format(name, msg))
    return date
```

La función `chat()` hace uso del módulo datetime para acceder a la información de la fecha y hora actual, esto mediante el método `now()`. La información de la fecha y hora es almacenada es la variable date. Luego se imprime el nombre del usuario junto con el mensaje emitido. Finalmente se retorna la información de la fecha y hora.

El llamado de la función `chat()` se hace de la siguiente manera:

```
>>> chat('Pedro', 'Hola, como estas?')
>>> Pedro: Hola, ¿cómo estás?
```

Si se ejecuta esta función solamente con el nombre del usuario esta retorna error. Se puede hacer que el segundo argumento tenga un valor predeterminado, esto de la siguiente manera:

se cambia en la función, esto

```
def chat(name, msg):
```

por esto

```
def chat(name, msg = 'Hola!'):
```

quedando la función definida como

```
from datetime import datetime
def chat(name, msg = 'Hola!'):
    date = datetime.now()
    print("{}: {}".format(name, msg))
    return date
```

En este caso `msg = 'Hola!'` significa que este será el mensaje predeterminado, de tal manera que si ejecutamos la función sin el segundo argumento, esta no arrojará un error, sino que visualizará el mensaje predeterminado, como se muestra a continuación:

```
>>> chat('Pedro')
>>> Pedro: Hola!
```

Otra manera de pasar argumentos es mediante el uso de las palabras clave usadas para nombrar los argumentos de la función; por ejemplo, en el caso de la función anterior se puede hacer uso de los nombres `name` y `msg` para pasar los argumentos de la función. De esta manera lo que se hace es evitar colocar los argumentos en el orden en que aparecen al inicio de la definición de la función. Un ejemplo de esto es:

```
>>> chat(msg = 'Hola, como estas?', name = 'Pedro')
>>> Pedro: Hola, ¿cómo estás?
```

6.1.1. Función recursiva

Una función recursiva es una función que se llama a sí misma al momento de llevar a cabo un cálculo, y en Python es posible definir una función de este tipo; por ejemplo, pensemos en la función factorial que permite determinar el factorial de determinado número, en donde el factorial de un número entero y positivo es el número multiplicado por los números que le preceden. Esto es el factorial de 1 es 1 el factorial de 2 es 2x1, el factorial de 3 es 3x2x1 y así sucesivamente. Una función recursiva que permite resolver este problema es la siguiente:

```
def factorial(n):
    if n == 1:
        return n
    elif n < 1:
        return ("No existe el factorial de números negativos!")
    else:
        return n*factorial(n-1)
```

Esta función toma como parámetro de entrada un número al cual se le evaluará el factorial de la siguiente manera. Si el número `n` es cualquier número entero positivo se buscará de manera recursiva el factorial de este número. La función accede a sí misma cuantas veces sea necesario, hasta que se cumpla la condición interna de la función. Si el número n es menor que 1 el factorial no existe y la función retorna un aviso.

Material de estudio complementario

Analogía de pseudocodigos a lenguaje Python
Funciones predefinidas en Python
Funciones en Python
Argumentos de una función
Módulos en Python

Referencias Bibliográficas

John Snowden (2020). Python For Beginners: A Practical Guide For The People Who Want to Learn Python The Right and Simple Way. Independently published.

Gries, P., Campbell, J., & Montojo, J. (2017). Practical programming: an introduction to computer science using Python 3.6. Pragmatic Bookshelf.

Zelle, J. M. (2004). Python programming: an introduction to computer science. Franklin, Beedle & Associates, Inc.