TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- ¿Qué es una función?
- ¿Para qué sirve una función?
- Sintaxis de funciones en Python
- Definir funciones
- Recepción de parámetros en una función
    - Orden de los parámetros
    - Listas como parámetros
    - Diccionarios como parámetros
- Retorno de una función
- Variables locales y variables globales
- Invocación de una función
- Alcance de las variables locales
- El problema de las variables globales

¿QUÉ ES UNA FUNCIÓN?

Es una porción de código que se puede utilizar una y otra vez. Las funciones permiten a los programadores dividir o descomponer un problema en trozos más pequeños, cada uno de los cuales realiza una tarea concreta.

Una vez que un programador define una función, puede llamarla siempre que la necesite, simplemente usando su nombre.

Estas pueden recibir entradas del programa. La entrada proporcionada a una función se llama argumentos de entrada, o simplemente argumentos. Los argumentos son el código que se pasa a una función como entrada.

Las funciones pueden producir una salida. Decimos que una función devuelve una salida al programa. La salida de una función puede ser asignada a una variable para su uso en un programa.

Algunas funciones ya están incorporadas a nuestro lenguaje de programación. En el caso de Python, ya hemos utilizado varias en los módulos anteriores, por ejemplo la función `print()`.

¿PARA QUÉ SIRVE UNA FUNCIÓN?

Las funciones pueden resolver cualquier tipo de problema que se requiera, usualmente como herramientas para resolver problemas más grandes dentro del sistema.

El uso de éstas nos brinda los siguientes beneficios:

- Permiten que el mismo trozo de código se ejecute varias veces.
- Dividen los programas largos en componentes más pequeños.
- Pueden ser compartidas y utilizadas por otros programadores.

SINTAXIS DE FUNCIONES EN PYTHON

```
1 def nombre_de_la_funcion(argumentos):
2     <cuerpo>
3     return respuesta
```

DEFINIR FUNCIONES

Una vez que conocemos la sintaxis de una función, desglosemos cada una de sus secciones para así tener claro como definirlas.

La primera línea de código de una función contiene 3 partes. La palabra clave `def` debe estar al principio de la línea que declara la función. `def` significa definición, e indica al intérprete de Python que seguirá una definición de función.

Cada función necesita un nombre. El `nombre_de_la_funcion` debe comenzar con una letra (o guion bajo), y suele ser todo en minúsculas. Los nombres de las funciones sólo pueden contener letras, números, y el carácter de subrayado también conocido como guion bajo.

Los nombres de las funciones van seguidos de un conjunto de paréntesis ( ). Muchas funciones tienen código, llamado argumentos, entre los paréntesis. El nombre utilizado para los argumentos de la función debe usarse en el cuerpo de la función, éstos son datos que pasamos a la función para realizar operaciones dentro de la misma. Cada parámetro está separado por comas (,), y hay funciones que no los requieren.

Después del nombre de la función, del paréntesis, y de los argumentos, vienen los dos puntos (:). En Python, se requieren dos puntos al final de la primera línea de todas las funciones.

El cuerpo de la función contiene el código que se ejecutará cuando se llame a la función. Cualquier variable declarada por los argumentos de la función puede ser utilizada en el cuerpo de ésta. Las variables utilizadas son variables locales, la cuales no pueden ser llamadas o accedidas por otros scripts, o utilizadas fuera del cuerpo de la función.

La palabra clave `return` suele ser la última línea de una función. Ésta indica que cualquier expresión que siga es la respuesta de la función, no es una función o un método, y no se utilizan paréntesis después de éste, sólo un espacio.

La respuesta es cualquier expresión que se incluya después de `return`. La expresión de salida después de `return` puede ser una sola variable, un valor, o ser una expresión compleja que incluya múltiples variables.

RECEPCIÓN DE PARÁMETROS EN UNA FUNCIÓN

Las funciones pueden ser escritas para aceptar múltiples argumentos de entrada. Cuando se especifican varios argumentos, éstos se enumeran entre paréntesis después del nombre de la función, y se separan con una coma.

Podemos definir valores preestablecidos para uno o varios parámetros de la función. Un parámetro con valor preestablecido significa que, si el usuario de la función no lo suministra automáticamente, se tomará el valor preestablecido. Para suministrar un valor prestablecido al declarar la función debemos escribir el nombre del parámetro, el signo igual, y luego el valor.

Como ejemplo, solo escribiremos la primera línea de la función, con un valor preestablecido para el parámetro mensaje.

```
1 def saludo(nombre, mensaje="Buenos días!"):
```

El **orden de los parámetros** es irrelevante, al menos que existan parámetros con valores preestablecidos. Cualquier número de argumentos de una función pueden tener un valor por defecto. Pero una vez que tenemos un argumento por defecto, todos los que están a su derecha también deben tener valores por defecto. Esto significa que los argumentos no predeterminados no pueden seguir a los argumentos predeterminados, o el intérprete nos dará un error.

Puedes enviar cualquier tipo de dato como argumento a una función (cadena, número, lista, diccionario, etc.), y será tratado como el mismo tipo de dato dentro de la función.

Podemos utilizar el operador * para enviar una **lista como parámetros**, éste es un método fácil y eficiente para enviar todos los valores de los parámetros de la función. El operador * desempaqueta los valores de la lista, y los convierte en los parámetros de la función.

De forma similar, también podemos emplear el operador ** para enviar un **diccionario como parámetro**, éste funciona de manera similar al anterior, pues al desempacar el diccionario lo que devolverá son los valores asociados a cada clave, y los asignará como parámetros de la función.

RETORNO DE UNA FUNCIÓN

Lo primero que debemos tener en cuenta es que una vez el intérprete encuentre el comando `return`, la ejecución de la función se detendrá al devolver el resultado.

El resultado puede ser un dato único (sin importar su tipo), o un grupo de éstos. En caso de ser un dato único, no hay que hacer ningún proceso en particular más que devolver la variable o el valor del resultado. Sin embargo, si necesitamos devolver un grupo de datos, la forma correcta es crear una tupla con todos los datos a devolver, y enviarla como resultado. La razón por la cual usamos una tupla es porque es una estructura inmutable.

VARIABLES LOCALES Y VARIABLES GLOBALES

En general, existen 2 tipos de variables como ya lo habíamos mencionado en otro módulo. Las variables locales y las variables globales.

Las variables globales se declaran en el cuerpo principal del código, y son llamadas así porque pueden ser utilizadas en cualquier momento y lugar del código.

Por su parte, las variables locales son aquellas que declaramos dentro de una función. Su **alcance** está restringido al cuerpo de la función, solo existen dentro del cuerpo de éste, y por lo tanto solo se pueden usar dentro del código de la función. Una vez que la función devuelve el resultado, las variables locales que se hayan creado dejarán de existir.

Existe un **problema de usar variables globales** en Python, y es que aun cuando pueden ser usadas dentro de una función, no pueden ser modificadas dentro de ella.

Cuando creamos funciones, y éstas tratan de modificar el valor de una variable global, internamente Python creará una variable local dentro de la función con el mismo nombre de la variable global, hará todas las instrucciones que indique el cuerpo, y al finalizar la función destruirá la variable local sin realmente hacer la modificación del valor de la variable global como originalmente se quería, pues todo el proceso lo hizo sobre una copia.

INVOCACIÓN DE UNA FUNCIÓN

Para hacer uso de una función dentro de código, debemos escribir su nombre seguido de paréntesis, y la lista de argumentos separados por comas dentro de los paréntesis.

```
print(“Este es el argumento”)

sumar(3, 6)

restar(*lista_de_datos)

multiplicar(**diccionario_de_datos)
```