IF Anidados

Hay situaciones en la vida real en las que necesitamos tomar algunas decisiones y, en base a estas decisiones, decidimos qué debemos hacer a continuación. También surgen situaciones similares en programación donde necesitamos tomar algunas decisiones y en base a estas decisiones ejecutaremos el siguiente bloque de código. Esto se hace con la ayuda de declaraciones de toma de decisiones en Python.

Cuando se anidan varios niveles de enunciados **if/else**, puede ser difícil determinar cuáles expresiones lógicas deben ser verdaderas (o falsas) con la finalidad de ejecutar cada conjunto de enunciados. La función **elif** le permite comprobar criterios múltiples mientras se mantiene el código fácil de leer.

```
if Condición :
    #código
    #código
elif Condición :
    #código
    #código
else:
    #código
    #código
#código fuera del if/else
```

bucle while

La sentencia o bucle while en Python es una sentencia de control de flujo que se utiliza para ejecutar un bloque de instrucciones de forma continuada mientras se cumpla una condición determinada.

el uso principal de la sentencia *while* es ejecutar repetidamente un bloque de código mientras se cumpla una condición.

La estructura de esta sentencia *while* es la siguiente:

```
while condición:
    bloque de código
```

Es decir, mientras condición se evalúe a True, se ejecutarán las instrucciones y sentencias de bloque de código.

BUCLE FOR

El bucle for se utiliza para recorrer los elementos de un objeto *iterable* (lista, tupla, conjunto, diccionario, …) y ejecutar un bloque de código. En cada paso de la iteración se tiene en cuenta a un único elemento del objeto iterable, sobre el cuál se pueden aplicar una serie de operaciones.

Su sintaxis es la siguiente:

```
for <elem> in <iterable>:
    <Tu código>
```

Aquí, elem es la variable que toma el valor del elemento dentro del iterador en cada paso del bucle. Este finaliza su ejecución cuando se recorren todos los elementos.

Qué es un iterable

Un *iterable* es un objeto que se puede iterar sobre él, es decir, que permite recorrer sus elementos uno a uno.

Finalmente, un *iterador* es un objeto que define un mecanismo para recorrer los elementos del *iterable* asociado.

FOR ANIDADOS

Es posible **anidar** los for, es decir, **meter uno dentro de otro**. Esto puede ser muy útil si queremos iterar algún objeto que en cada elemento, tiene a su vez otra clase iterable. Podemos tener por ejemplo, una lista de listas, una especie de matriz.

FUNCIONES

Las funciones en Python, y en cualquier lenguaje de programación, son estructuras esenciales de código. Una función es un grupo de instrucciones que constituyen una unidad lógica del programa y resuelven un problema muy concreto.

las funciones en Python constituyen unidades lógicas de un programa y tienen un doble objetivo:

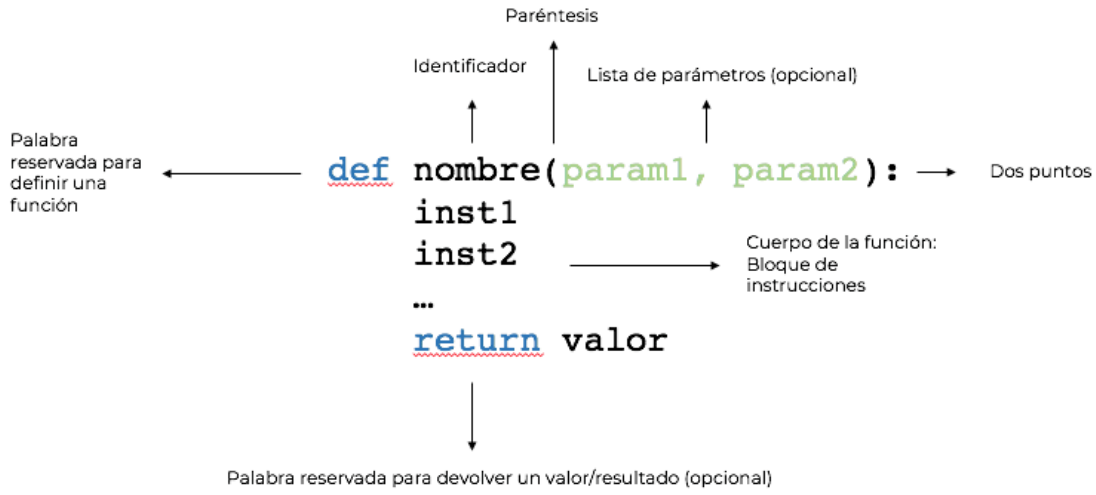
- Dividir y organizar el código en partes más sencillas.
- Encapsular el código que se repite a lo largo de un programa para ser reutilizado.

Python ya define de serie un conjunto de funciones que podemos utilizar directamente en nuestras aplicaciones. Algunas de ellas las has visto en tutoriales anteriores. Por ejemplo, la función len(), que obtiene el número de elementos de un objeto contenedor como una lista, una tupla, un diccionario o un conjunto. También hemos visto la función print(), que muestra por consola un texto.

Como programador llega el momento que defines tus propias funciones para estructurar el código de manera que sea más legible y para reutilizar aquellas partes que se repiten a lo largo de una aplicación. Esto es una tarea fundamental a medida que va creciendo el número de líneas de un programa.

Cómo definir una función en Python

La siguiente imagen muestra el esquema de una función en Python:



Para definir una función en Python se utiliza la palabra reservada def. A continuación, viene el nombre o identificador de la función que es el que se utiliza para invocarla. Después del nombre hay que incluir los paréntesis y una lista opcional de parámetros. Por último, la cabecera o definición de la función termina con dos puntos.

Tras los dos puntos se incluye el cuerpo de la función (con un sangrado mayor, generalmente cuatro espacios) que no es más que el conjunto de instrucciones que se encapsulan en dicha función y que le dan significado.

En último lugar y de manera opcional, se añade la instrucción con la palabra reservada return para devolver un resultado.

Cómo usar o llamar a una función

Para usar o invocar a una función, simplemente hay que escribir su nombre como si de una instrucción más se tratara. Eso sí, pasando los argumentos necesarios según los parámetros que defina la función.

Funciones return

cuando acaba la última instrucción de una función, el flujo del programa continúa por la instrucción que sigue a la llamada de dicha función. Hay una excepción: usar la sentencia return. return hace que termine la ejecución de la función cuando aparece y el programa continúa por su flujo normal.

Además, return se puede utilizar para devolver un valor.

La sentencia return es opcional, puede devolver, o no, un valor y es posible que aparezca más de una vez dentro de una misma función.

return que no devuelve ningún valor

```
def nombre(parametro):
    if sentencia:
        return
    else:
        sentiencia
```

Varios return en una misma función

```
def nombre(parametro):
    if sentencia:
        return True
    else:
        return False
```

Devolver más de un valor con return en Python

```
def nombre(parametro):
    return sentencia



def nombre(parametro):
      sentencia…
      sentencia…
      sentencia…
    return respuesta/resultado
```

En Python una función siempre devuelve un valor

Python, a diferencia de otros lenguajes de programación, no tiene procedimientos. Un procedimiento sería como una función pero que no devuelve ningún valor.

¿Por qué no tiene procedimientos si hemos vistos ejemplos de funciones que no retornan ningún valor? Porque Python, internamente, devuelve por defecto el valor `None` cuando en una función no aparece la sentencia `return` o esta no devuelve nada.

Argumentos y Parámetros

En la definición de una función los valores que se reciben se denominan parámetros, pero durante la llamada los valores que se envían se denominan argumentos.

Argumentos por posición

Cuando enviamos argumentos a una función, estos se reciben por orden en los parámetros definidos. Se dice por tanto que son argumentos por posición:

```
def resta(a, b):
    return a - b

resta(30, 10)  # argumento 30 => posición 0 => parámetro a
               # argumento 10 => posición 1 => parámetro b
```

Argumentos por nombre

Es posible evadir el orden de los parámetros si indicamos durante la llamada que valor tiene cada parámetro a partir de su nombre:

```
resta(b=30, a=10)
```

Funciones Recursivas

Se trata de funciones que se llaman a sí mismas durante su propia ejecución. Funcionan de forma similar a las iteraciones, pero debemos encargarnos de planificar el momento en que dejan de llamarse a sí mismas o tendremos una función rescursiva infinita.

Suele utilizarse para dividir una tarea en subtareas más simples de forma que sea más fácil abordar el problema y solucionarlo.

Funciones Integradas

Python tiene un total de 69 funciones integradas que podemos utilizar sin necesidad de importar ningún módulo.

Entrada/salida

print()

Esta es seguramente la función más conocida de todas. Lo que hace print() es imprimir por la salida estándar la representación en string de cualquier objeto. Además tiene varios parámetros de entrada opcionales que modifican su comportamiento.

input()

La función integrada input() toma datos de entrada por el teclado hasta que pulsamos intro. Normalmente se usa en la forma input(mensaje), donde mensaje es un string para indicar al usuario qué datos espera el programa. Dichos datos suelen almacenarse en una variable para su posterior procesado.

format()

La función format(valor, formato) formatea el valor numérico de acuerdo al formato que le especifiquemos. En concreto retorna un string que representa ese valor formateado. Esta función se utiliza típicamente para determinar el número de decimales con los que se muestra un valor numérico al usuario.

Funciones matemáticas

abs()

La función abs(numero) retorna el valor absoluto de numero, es decir su valor sin importar su signo. El parámetro numero puede ser un entero, un número de coma flotante o un objeto que implemente la función __abs__().

round()

La función round(numero) redondea numero a su entero más próximo. Pero los números decimales que terminan en 5 son un caso especial. Python sigue la estrategia de “redondear empates a números pares”. Esto significa que parar redondear un número de coma flotante terminado en 5, se mira el dígito que tiene

a su izquierda. Si ese dígito es par, el redondeo se hace hacia abajo. Es por eso que en Python tanto 1.5 como 2.5 redondean a 2.

pow()

La función pow(base, exponente) calcula la potenciación de base elevado a exponente. Se trata de una función equivalente a realizar el cálculo base ** exponente.

Estructuras de datos

list()

En realidad list() no es una función en si, sino que se trata de un constructor para listas. Cuando lo utilizamos sin argumentos crea una lista vacía. También le podemos pasar como parámetro una secuencia iterable, en cuyo caso la convierte a una lista. Normalmente esa secuencia iterable que le pasamos a list() no suele ser una lista ya que estaríamos generando código redundante.

dict()

dict() es el constructor de la clase diccionario. Usado sin argumentos retorna un diccionario vacío. Con dict() también podemos crear un diccionario que no sea vacío. Para ello tenemos que pasarle argumentos nombrados, donde los nombres de los argumentos se corresponden con las claves del diccionario y los valores con los valores para dicha clave.

tuple()

tuple() es el constructor de la clase tupla. Podemos usarlo sin argumentos para crear una tupla vacía, o pasarle un objeto iterable cuyos elementos serán los elementos de la tupla. En este sentido, podemos utilizar tuple() para convertir fácilmente un string o una lista a una tupla.

set()

set() es el constructor de la clase set. Podemos usarlo sin argumentos para crear un set vacío, o pasarle un objeto iterable cuyos elementos no repetidos serán los elementos del set. En este sentido, podemos utilizar set() para filtrar los elementos repetidos de un string o una lista.

Generación de secuencias iterables

range()

La función range(fin) genera una secuencia de números enteros que podemos utilizar para iterar en un bucle. La secuencia generada empieza en 0 y termina en el entero anterior a fin. Es decir, el valor fin no forma parte de la secuencia. Alternativamente, podemos usar range(inicio, fin, paso), donde inicio indica el primer número de la secuencia y paso cada cuantos números se toman en cuenta para la secuencia.

enumerate()

La función enumerate(iterable) toma una secuencia o un iterador y retorna objeto de tipo enumerate. Dicho objeto es una secuencia iterable de tuplas con sus respectivos índices. Cada una de estas tuplas tiene la forma (i, x) donde i es el índice del objeto x perteneciente a iterable, el parámetro de entrada de la función. Por defecto el primer índice es 0, aunque también podemos definir su valor usando la función enumerate(iterable, inicio).

Operaciones con objetos iterables

len()

La función len(objeto) retorna el número de elementos que contiene un objeto. Dicho objeto puede ser tanto una secuencia (un string, una lista, una tupla, etc.) como una colección (un diccionario).

sum()

La función sum(iterable) retorna el total de sumar los elementos de la secuencia iterable. Para poder realizar la suma, los elementos de iterable tienen que ser números. Por defecto, el resultado de la suma empieza a contar en 0, pero podemos hacer que empiece en otro valor si lo pasamos como segundo argumento a la función.

max()

La función max(iterable) retorna el elemento más grande del objeto iterable. También se pueden utilizar dos o más argumentos, en cuyo caso retorna el mayor de los argumentos.

min()

La función min(iterable) retorna el elemento más pequeño del objeto iterable. También se pueden utilizar dos o más argumentos, en cuyo caso retorna el menor de los argumentos.

Modificaciones de objetos iterables

sorted()

La función sorted(iterable) retorna una lista con los elementos de iterable ordenados de menor a mayor. También puede retornar los elementos ordenados de mayor a menor cuando se especifica el parámetro opcional reverse del siguiente modo: sorted(iterable, reverse=True).

reversed()

La función reversed(secuencia) retorna un iterador revertido de los valores de una secuencia como por ejemplo un string, una lista, una tupla, etc. Es decir, intercambia el primer elemento con el último, el segundo con el penúltimo, etc.

Funciones aplicadas a iterables

map()

La función map(funcion, iterable) aplica funcion a cada uno de los elementos del objeto iterable y retorna el resultado en un objeto map. Dicho objeto se puede convertir a tipo lista con list(), a tipo tupla con tuple(), etc.

zip()

La función zip() toma como argumentos un número arbitrario de iteradores, y los agrega en un objeto de tipo zip. Dicho objeto es un iterador de tuplas, donde la primera tupla está compuesta por los primeros elementos de cada uno de los iteradores, la segunda tupla por los segundos elementos, etc.

Funciones lambda

Si empiezo diciendo que las funciones o expresiones lambda sirven para crear funciones anónimas, pues estamos ante unas de las funcionalidades más potentes de Python a la vez que más confusas para los principiantes.

Una función anónima, como su nombre indica es una función sin nombre. ¿Es posible ejecutar una función sin referenciar un nombre? Pues sí, en Python podemos ejecutar una función sin definirla con def. De hecho son similares pero con una diferencia fundamental:

El contenido de una función lambda debe ser una única expresión en lugar de un bloque de acciones.

Y es que más allá del sentido de función que tenemos, con su nombre y sus acciones internas, una función en su sentido más trivial significa realizar algo sobre algo. Por tanto podríamos decir que, mientras las funciones anónimas **lambda** sirven para realizar funciones simples, las funciones definidas con **def** sirven para manejar tareas más extensas.

Si deconstruimos una función sencilla, podemos llegar a una función lambda.

FUNCIONES DE ORDEN SUPERIOR

Las funciones de orden superior, son simplemente funciones que reciben como argumento otras funciones. Suelen usarse por ejemplo en las listas, con métodos como Find, que reciben una función de búsqueda.

De esta manera la función opera sobre los elementos de un objeto o clase. Y la función de destino no debe preocuparse de implementar las diversas funcionalidades, solo inyecta los parámetros adecuados en la otra función.

Manejo de Archivos

Para el manejo de archivos en python usaremos el módulo estándar de python oi.

Objetivo Principal: persistencia de datos

Alternativas:

1. Manejo de archivos externos
2. Trabajo con BD

Manejo de Archivos Externos

Fases necesarias para guardar información en archivos externos

Manejo de Excepciones

Los errores de ejecución son llamados comúnmente excepciones, por ende, durante la ejecución de un programa, si dentro de una función surge una excepción y la función no la maneja, la excepción se propaga hacia la función que la invocó, si esta otra tampoco la maneja, la excepción continúa propagándose hasta llegar a la función inicial del programa y si esta tampoco la maneja se interrumpe la ejecución del programa.

Manejo de excepciones

Para el manejo de excepciones los lenguajes proveen ciertas palabras reservadas, que nos permiten manejar las excepciones que puedan surgir y tomar acciones de recuperación para evitar la interrupción del programa o, al menos, para realizar algunas acciones adicionales antes de interrumpir el programa.

En el caso de Python, el manejo de excepciones se hace mediante los bloques que utilizan las sentencias **try, except y finally.**

Dentro del bloque try se ubica todo el código que pueda llegar a levantar una excepción, se utiliza el término levantar para referirse a la acción de generar una excepción.

A continuación, se ubica el bloque except, que se encarga de capturar la excepción y nos da la oportunidad de procesarla mostrando por ejemplo un mensaje adecuado al usuario.