# Taller Integrador - Python.

### Clase 2

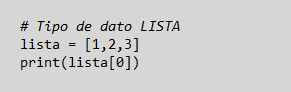
### Sábado 04/04/2020 (Chivilcoy)

### Martes 07/04/2020 CABA.

# Unidad 2. Listas, estructuras y operadores.

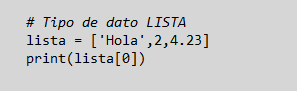
### Tipo de dato LISTA.

El tipo lista en Python me permite almacenar en su interior, diferentes tipos de datos, separados por coma. El primer elemento de la lista ocupa la posición 0 (cero).



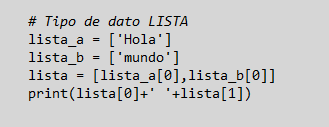
Si ejecutamos el código anterior, veremos por pantalla el elemento que ocupa la posición cero de la lista, en nuestro caso será el número 1.

Podemos cargar nuestra lista con elementos de varios tipos diferentes.



Veremos por pantalla, al ejecutarlo, la palabra HOLA.

Dijimos que las listas aceptaban cualquier tipo de dato, por lo tanto, podemos cargarle otras listas, por ejemplo:

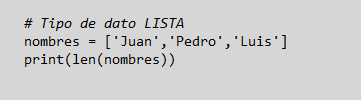


Veremos por pantalla, el famoso Hola mundo.

Podemos mostrar por pantalla todo el contenido de una lista sin necesidad de recorrerlo elemento por elemento, como vimos en el primer ejemplo o podemos manipular cada uno de los mismos, y recombinarlos en una nueva lista, utilizando algunos o todos los elementos de las listas originales.

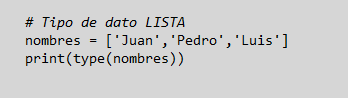
### LEN.

La función LEN nos permite conocer la longitud de una lista.



El resultado será el número 3, ya que la lista tiene tres elementos.

Al igual que con los tipos de datos que vimos, también podemos ver con TYPE que tipo de dato es:



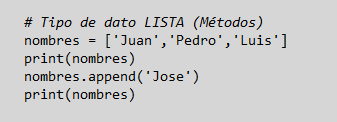
Esto veremos por pantalla:



### Métodos del tipo Lista.

#### APPEND

Este método agrega un elemento al final de la lista.



En el ejemplo anterior, creamos una lista con los nombres Juan, Pedro y Luis.

Luego, utilizando el método append() agregamos, al final de la misma, el nombre Luis.

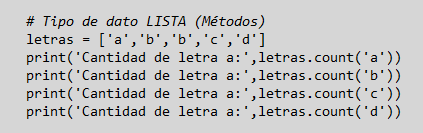
Si ejecutamos el código, esto es lo que veremos por pantalla:



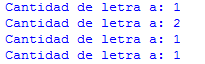
#### COUNT

Este método recibe un elemento como argumento, y nos devuelve la cantidad de veces que aparece dicho elemento dentro de la lista.

Por ejemplo, si creamos una lista con algunas letras del abecedario, podremos contar la cantidad de veces que aparece cada una dentro de la misma:



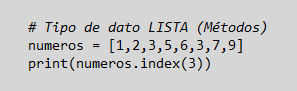
Y esto es lo que deberíamos ver en pantalla al ejecutar el código:



#### INDEX

Este método recibe un elemento como argumento, y devuelve el índice de su primera aparición en la lista.

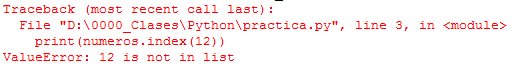
Por ejemplo, cargamos con números una lista de la siguiente manera, y mostramos el índice de la primera aparición de un elemento repetido:



Este es el resultado que veremos por pantalla:



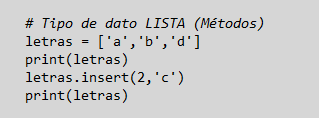
Si el elemento no se encuentra en la lista, este método devolverá un mensaje de error.



#### INSERT

Permite insertar un elemento en la lista, en el índice que necesitemos.

Por ejemplo, tenemos la lista de letras a, b y d pero queremos agregar la letra c inmediatamente después de la letra b.



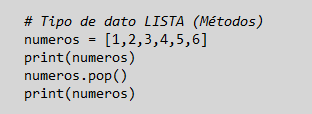
Al ejecutarlo, esto es lo que veremos por pantalla:



#### POP

Este método devuelve el último elemento de una lista, y lo borra.

Por ejemplo, en los envíos de datos en los que el último elemento es un código de verificación para comprobar que los datos han sido transferidos por completo y sin errores.



Esto es lo que veremos por pantalla ejecutando el ejemplo anterior:

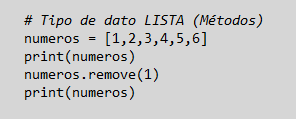


#### REMOVE

Este método recibe como argumento un elemento y borra su primera aparición en la lista.

En caso de no encontrar al elemento en la lista, devuelve un mensaje de error.

Veamos un ejemplo en el que cargamos una lista con los números del 1 al 6, los mostramos por pantalla y luego eliminamos el número 1.

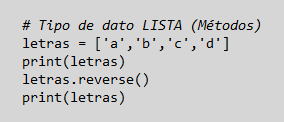


Así veremos el resultado de la ejecución por pantalla:



#### REVERSE

Invierte el orden de los elementos de una lista.

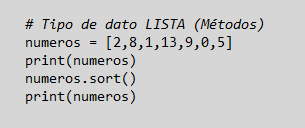


Mostramos por pantalla la lista original y la lista invertida:



#### SORT

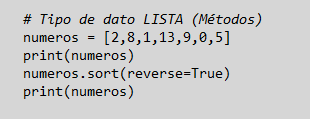
Ordena los elementos de una lista.



Este es el resultado visto por pantalla:



También tenemos la opción de ordenarlos en sentido inverso, de la siguiente manera:



Al poner el parámetro reverse en True, el ordenamiento se hace de mayor a menor:



# Estructuras.

### Condicional IF

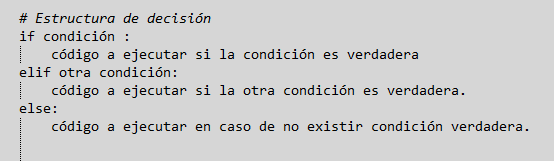
El condicional if se utiliza básicamente para tomar decisiones.

Esta decisión está basada en una operación lógica (verdadera o falsa).

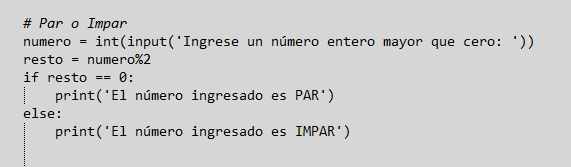
Veamos un ejemplo con ingreso de números por teclado, pero vamos a ver como convertimos los datos tipo texto que toma por defecto el input en un entero.

### 

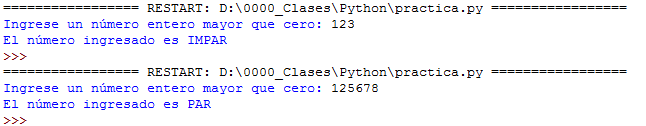
La sintaxis de la estructura de decisión es la siguiente:



Por ejemplo, le pedimos al usuario que ingrese un valor entero, mayor que cero y verificamos si el número es par o impar.

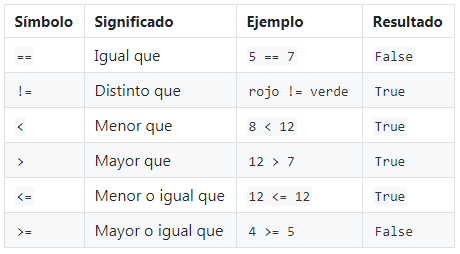


IMPORTANTE: hay que respetar el identado del código a ejecutar dentro del if, elif y else como así también los dos puntos que lleva cada uno.

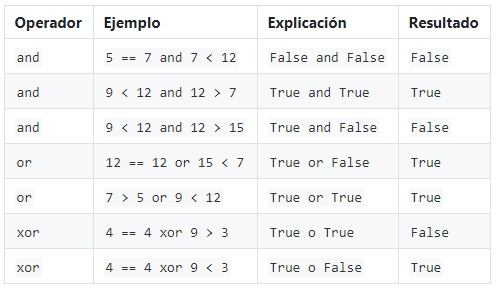


### Operadores relacionales.

Son aquellos que utilizamos para evaluar determinada condición.



### Operadores lógicos.

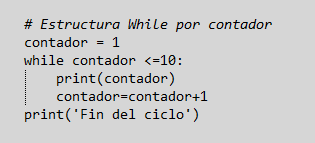


### Estructura While.

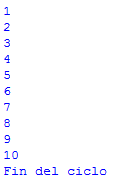
Es una estructura de repetición, que me permite ejecutar una porción de código la cantidad de veces que le indiquemos.

Estas repeticiones son controladas por el programador, por ejemplo, mediante el uso de un contador o el resultado de un evento.

El siguiente ejemplo, es una estructura While controlada por un contador:



Al ejecutarlo, este es el resultado que deberíamos ver por pantalla:

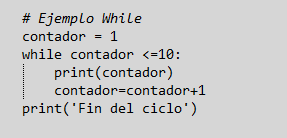


En otros lenguajes tenemos dos tipos de ciclos, el While y el Do While.

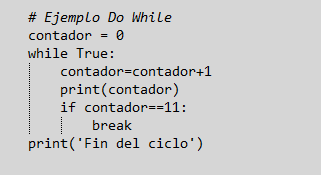
En Python (al menos en la versión 3) tenemos solo el ciclo While. Está en nosotros, como programadores, armar la estructura para que se comporte de una u otra manera.

Recordemos que la diferencia entre una y otra estructura es que el While evalúa la condición antes de empezar el ciclo y el Do While ejecuta al menos una vez el ciclo y recién después empieza a evaluar la condición.

Ejemplo While:



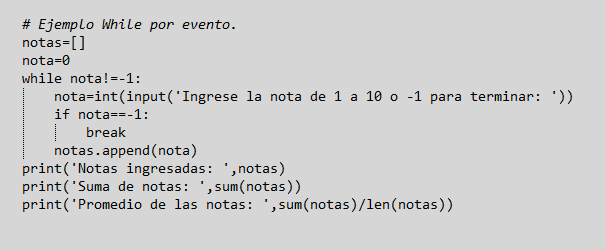
Ejemplo Do While:



Veamos ahora una estructura While controlada por un evento.

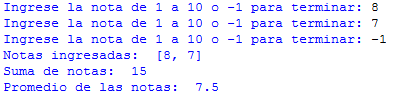
Le pedimos al usuario que ingrese una serie de notas del 1 al 10 y cuando no quiera ingresar más valores, que ingrese un -1.

Una vez ingresado el -1, dicho evento termina el ciclo y continúa con la ejecución del programa, mostrando los valores ingresados, la suma y el promedio.



Incorporamos el método sum que nos proporciona la suma de los elementos numéricos de una lista.

Esto veríamos por pantalla al ejecutar el código e ingresarlo, por ejemplo, los valores 8 y 7:

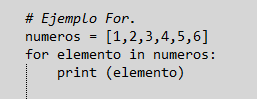


### Estructura FOR.

En Python, la sentencia for tiene un uso que difiere un poco del que se le da en otros lenguajes.

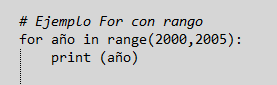
En nuestro caso la sentencia for itera sobre los ítems de cualquier secuencia, por ejemplo, una lista, en el orden en el que aparecen los elementos.

Por ejemplo:



El ciclo for recorre toda la lista.

También podemos usar esta estructura como generadora de elementos para utilizarlos en otra parte de nuestro código mediante el atributo range().



En este ejemplo le pedimos que muestre los años comprendidos entre el 2000 y el 2005.

Si prestamos atención a lo que se ve por pantalla luego de la ejecución, notaremos que la función, toma inicia en el primer valor del rango y termina uno antes del final.

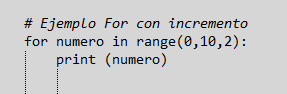


Este detalle es importante para no obtener un resultado inesperado.

Lo interesante de Python es que nos permite componer nuestro código combinando las diferentes sentencias, métodos y funciones.

Esto facilita muchísimo el desarrollo, acorta los tiempos y nos permite generar un código simple y fácil de mantener.

A la función range() le podemos agregar un tercer valor, para indicarle el incremento o decremento que queremos obtener.



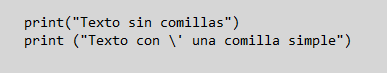
Este es el resultado visto por pantalla:



### Cadenas de caracteres.

Vimos que además de números, Python es capaz de manipular cadenas de caracteres.

Estas pueden estar encerradas entre comillas simples o dobles, y el carácter \ se utiliza para escapar a as comillas.



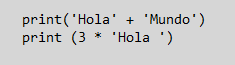


Si no queremos que los caracteres de escape se interpreten como tal, utilizamos la cadena cruda, anteponiendo una letra r antes de abrir las comillas.





Las cadenas de texto pueden concatenarse con el operador + y pueden repetirse con el operador \*.





También serán concatenadas dos o más cadenas que, sin estar ligadas con el operador +, estén juntas.





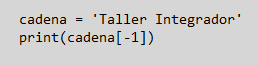
Las cadenas de texto se pueden indexar. El primer carácter de dicha cadena tendrá el índice 0 (cero).

No existe en Python un tipo especial para los caracteres, simplemente se los considera como una cadena de longitud 1.



Veremos por pantalla la letra T, que es el carácter que ocupa el índice cero dentro de la cadena de texto.

Si necesitamos empezar a utilizar los caracteres de una cadena de atrás hacia adelante, podemos averiguar su longitud con la función LEN o podemos utilizar argumentos negativos, siendo el -1 el último elemento de la cadena, -2 el penúltimo y así sucesivamente.



Al ejecutar el ejemplo anterior, veremos por pantalla la letra r.

### Sub cadenas.

En Python tenemos disponible el concepto de sub cadenas o rebanadas como figura en muchos documentos del lenguaje.

La notación es simple cadena [inicio: final]

Al igual que en la función RANGE, el inicio está incluido en la sub cadena, pero no el final.

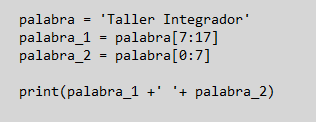
Por ejemplo, si queremos obtener, de nuestra cadena “Taller Integrador” solo la palabra “Integrador”, hacemos lo siguiente:



Esto es lo que veremos por pantalla:



Al obtener una sub cadena, no estamos modificando en nada a la cadena original, solo estamos mostrando una parte de esta o, en el caso de asignarla a otra variable, estaremos creando una nueva cadena de texto.



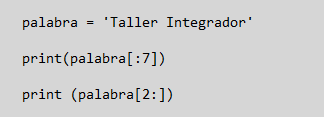
Esto vemos por pantalla:



Los parámetros de inicio y fin de las sub cadenas, tienen valores por defecto.

En el caso del inicio, si no lo aclaramos, tomara por defecto el primer elemento de la cadena de texto.

En el caso del final, si no se lo indicamos, tomará por defecto al último elemento de nuestra cadena.



En el primer PRINT lo que intentamos hacer es mostrar por pantalla, los elementos contenidos dentro de la variable “palabra” comprendidos entre el primero y el sexto elemento.

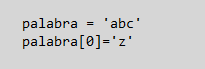
El cambio, en el segundo PRINT, mostramos desde el segundo elemento hasta el último.

Esto veremos por pantalla:

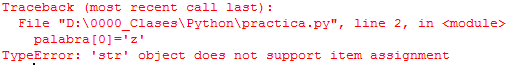


Las cadenas en Python son inmutables, no pueden ser modificadas.

Por ejemplo, si intentamos agregar un nuevo elemento a una cadena ya existente, el intérprete nos dará un mensaje de error.



Si ejecutamos lo anterior, veremos el siguiente mensaje de error:



### Listas (continuación).

Al igual que las cadenas de caracteres, las listas pueden ser rebanadas (sub listas).

Por ejemplo, podríamos armar una lista a partir de otra lista más grande, utilizando solamente los primeros tres elementos.

### 

Definimos una nueva lista a la que llamamos sub\_lista y le cargamos los primeros tres elementos de la lista original.

Así se ve el resultado por pantalla:

### 

Las listas también soportan operaciones de concatenación:

### 



Como ya habíamos visto, se puede agregar o quitar elementos de una lista, por lo que son del tipo mutable (es posible cambiar su contenido)

Otra operación interesante que podemos hacer con las sub listas, es reemplazar de una sola vez, el contenido de cierta parte de la lista original, sin necesidad de hacerlo de a un elemento.

Veamos un ejemplo, tengo una lista con cuatro letras “a” y quiero reemplazar la segunda y la tercera, por una letra “b”.

Hacemos lo siguiente:

### 

Recordar siempre que en este tipo de pase de argumentos donde pasamos el inicio y el fin, el inicio siempre está incluido, pero el final no. Es por eso que como solo quiero cambiar dos elementos, pongo el número 3.

### 

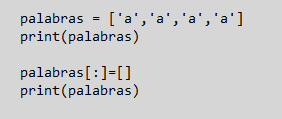
También puedo eliminar una porción de la lista, por ejemplo, las recién ingresadas letras “b”:

### 

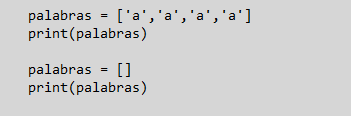
Al ejecutarlo, esto veremos por pantalla:

### 

Y con el mismo método, podemos también vaciar una lista entera:



Lo anterior es equivalente a hacer:

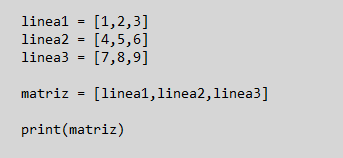


El resultado será el mismo, va a quedar la lista original, vacía:



### Listas anidadas.

Podemos anidar las listas, por ejemplo, para armar una matriz de datos.



En este caso, cuando la veamos por pantalla, veremos los elementos de una lista y a continuación los elementos de las demás listas:

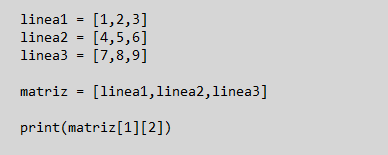


Si queremos ubicar un elemento determinado de nuestro arreglo, podemos indicarle dos índices a nuestra lista matriz.

El primer índice indica la fila y el segundo indica la posición dentro de dicha fila.

Por ejemplo, si queremos mostrar por pantalla el número 6 de nuestra matriz, deberemos utilizar sus “coordenadas”.

El primer argumento es el número de línea en donde se encuentra (contando a partir de cero) en nuestro caso será la línea 1, y a continuación, la posición dentro de dicha línea (también empieza a contar desde cero) en nuestro caso, el número 6 se encuentra en la posición 2.

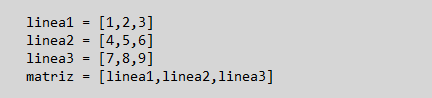


Lo que estamos haciendo con nuestro PRINT es mostrar por pantalla el elemento de la matriz que se encuentra en la fila 1, posición 2, es decir, el número 6.

### Recorriendo un arreglo.

Si bien Python proporciona herramientas para el uso de arreglos, podemos recorrer nuestra matriz que acabamos de crear de forma muy sencilla con una estructura FOR anidada:

Veamos un ejemplo en el que creamos tres listas, de tres elementos cada una y las anidamos creando una matriz de 3x3.

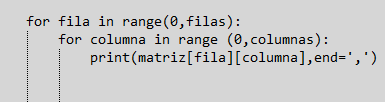


Para recorrerlo, lo primero que debeos averiguar, es cuantas filas tiene nuestra matriz.

También deberemos averiguar el número de columnas, lo que podemos obtener fácilmente contando los elementos de una de sus filas, por ejemplo, la primera:



Por último, anidamos dos estructuras FOR que recorran fila por fila, del número cero hasta la última, y por cada fila, que recorra todos sus elementos (columnas) del primero al último.

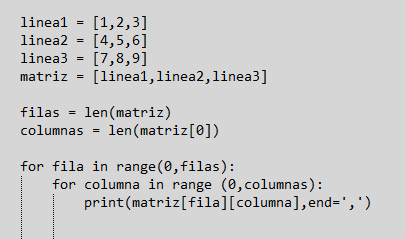


La sentencia END dentro del print es para evitar el salto de línea.

Esto veremos por pantalla:



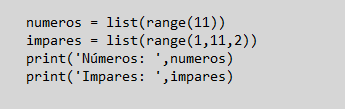
Y este es el código completo del ejercicio:



### Creación de listas con LIST.

Podemos combinar la función RANGE con LIST para crear listas, cuyos elementos estén acotados por esta última.

Por ejemplo, queremos crear dos listas, una con los números del 1 al 10 y otra con los impares entre 1 y 10.



Esta es la salida por pantalla:



### La sentencia PASS.

La sentencia PASS no hace nada, se utiliza en los casos en que el intérprete Python requiere una instrucción, pero no una acción, por ejemplo:



Este código no hace nada, solo se queda esperando que se presione una tecla.

También se utiliza (lo vamos a ver más adelante) en programación orientada a objetos, cuando declaramos una clase en su mínima expresión para luego desarrollar su contenido.

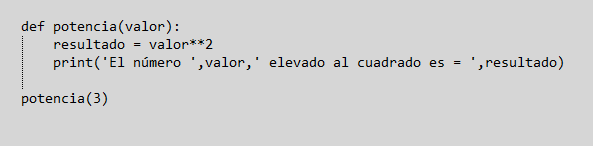
# Unidad 3. Funciones.

### Introducción.

Para definir funciones, utilizamos la palabra reservada DEF y a continuación, el nombre que le damos a la función y a lista de parámetros.

Las sentencias que forman parte del cuerpo de la función, comienzan a partir de la siguiente línea, y deben estar identadas (sangría).

Escribamos una función básica a la que le pasamos un entero por argumento, y nos devuelve dicho número, elevado al cuadrado.

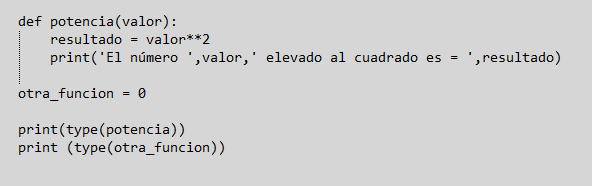


Al ejecutarlo, lo que hacemos es definir una función llamada potencia que recibe un valor por parámetro, lo guarda en la variable valor, lo eleva al cuadrado y muestra el resultado por pantalla.



Una función definida puede ser asignada a otra (sin definir) para luego utilizarla como si fuese la función original.

Veamos un ejemplo simple con nuestra función potencia definida anteriormente.

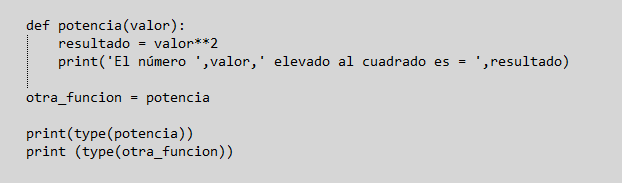


En nuestro código tenemos una función llamada potencia y una variable común llamada otra función.

Si ejecutamos el código anterior, veremos que Python nos informa el tipo de cada uno:



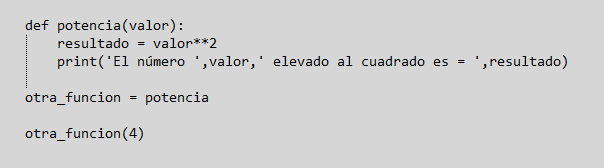
Probemos ahora de ver qué sucede si asignamos a la variable, nuestra función potencia:



Si lo ejecutamos, veremos por pantalla que ahora, nuestra variable es una función:



Y no solamente cambia el tipo de dato, ahora también podemos usar la nueva función de la misma forma en que utilizábamos la original:

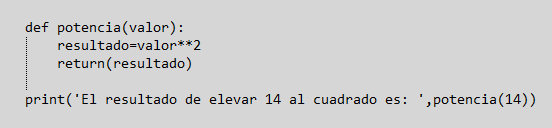


Este es el resultado visto por pantalla:



### Retorno de valores.

En nuestro ejemplo anterior, la función era la encargada de mostrar por pantalla, el resultado de la operación matemática. Pero podemos declararla de tal forma que retorne el resultado para poder utilizarlo en otra parte de nuestro código.



A diferencia del ejemplo anterior, en este caso la función calcula el cuadrado del número pasado por argumento y lo devuelve para ser utilizado en otra parte del programa.

### Argumentos y parámetros.

Cuando definimos una función que debe recibir valores, dichos valores son los parámetros de la función.

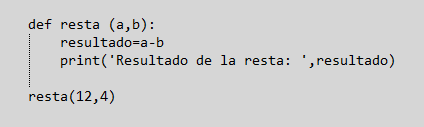
Cuando invocamos a la función y le pasamos los valores, esos valores se denominan argumento.

Veamos un par de ejemplos de cómo podemos invocar una función a la que debemos pasarle uno o más argumentos.

### Pasaje de argumento por posición.

Cuando le enviamos argumentos a una función, estos se reciben y se procesan en base al orden en que son recibidos.

Veamos un ejemplo:



En nuestro ejemplo, la función espera los valores para a y b, y los asignará en el orden en que se los pasemos.

Por ejemplo, si le pasamos el par de valores (12,4), al número 12 le restará el número 4.

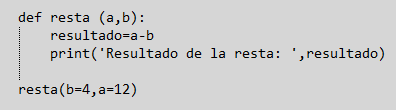


Probemos que sucede si los pasamos invirtiendo el orden:

### Pasaje de argumento por nombre.

En Python es posible desentendernos del orden en que pasemos los argumentos, si al invocar la función, especificamos a que parámetro de la misma corresponde cada valor que pasemos.

Ejemplo:



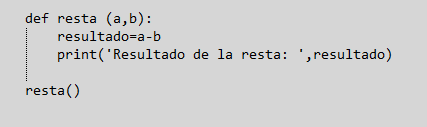
Si bien estamos pasando el par de valores en el mismo orden que nuestro ejemplo anterior (cuando la función hacía 4 – 12) en esta ocasión, le especificamos a que parámetro corresponde cada uno.

Este es el resultado ahora:

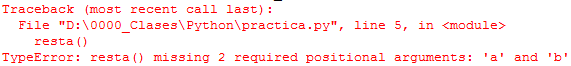


### Función invocada sin argumentos.

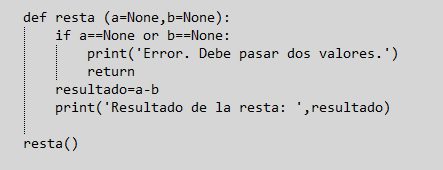
¿Qué pasaría si a nuestra función dedicada a restar dos valores que le pasamos por argumento, no le enviáramos uno de los argumentos o directamente ninguno?



Al ejecutar el código del ejemplo, veríamos por pantalla el siguiente mensaje de error:



Podemos evitar este tipo de error, asignando a nuestros parámetros, valores por defecto.



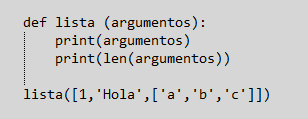
Al ejecutar el código, estamos invocando una función sin pasarle ningún argumento, pero en la definición de dicha función, dejamos especificado un valor por defecto para cada parámetro (None).

Al no recibir los argumentos, la función devuelve un mensaje de error:



### Argumentos indeterminados.

Podemos definir una función que reciba un número indeterminado de argumentos de la siguiente manera:



Definimos una función que espera un argumento, pero en realidad le pasamos una lista.

La función muestra por pantalla los argumentos recibidos y la longitud en caso de que sea una lista.



Si quisiéramos utilizar esta misma función para pasarle un solo argumento, lo aceptaría sin problemas, pero nos daría un mensaje de error por aplicarle la función LEN a un valor entero.

### Números aleatorios.

En nuestros primeros códigos de Python vimos cómo importar el módulo keyword.py que se encuentra en la instalación del intérprete en la ruta \lib

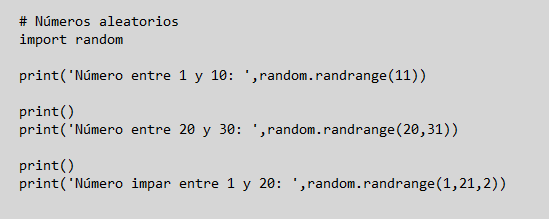
En esa misma carpeta tenemos el módulo random.py que vamos a importar de la misma manera:



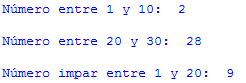
Para utilizarlo, nos valemos del método randrange(x,y,z) donde:

* x: inicio del rango de números.
* y: fin del rango de números (no está incluido)
* z: incremento

Solo es necesario el primer argumento, veamos algunos ejemplos de uso:

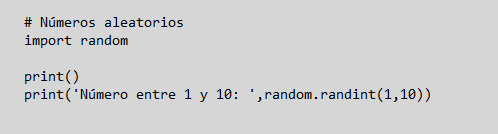


Este código nos dará la siguiente salida por pantalla:



### Método RANDINT.

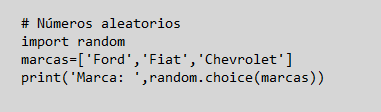
Este método a diferencia del anterior, si incluye en valor de fin de rango:



### Método CHOICE (SECUENCIA).

Devuelve un elemento al azar de una lista de elementos predefinida.

Por ejemplo, tenemos una lista con marcas de autos y queremos seleccionar una al azar:



### Método SHUFFLE ( ).

Mezcla al azar los elementos de una lista.

Supongamos que tenemos una lista con los números del 1 al 5 y reordenamos sus elementos antes de mostrarlos por pantalla:

