

Programación en Python

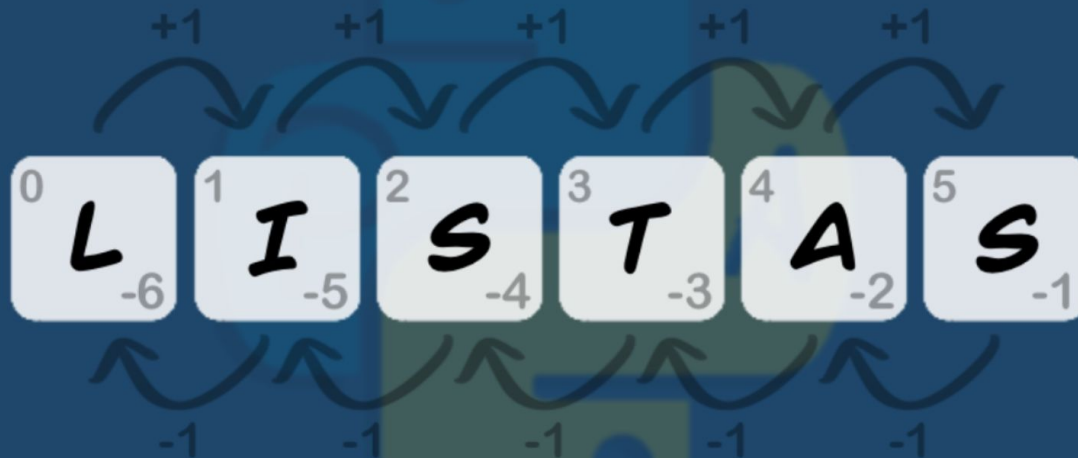
Profesora: Nancy Bernal Sánchez

TEMARIO

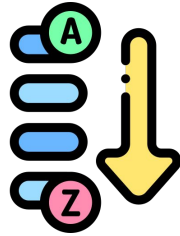
- ★ Listas
- ★ Diccionarios
- ★ Funciones

Listas

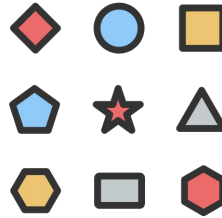
Son un tipo de dato que permite almacenar datos de cualquier tipo.



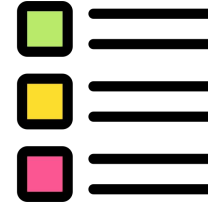
Listas: Características



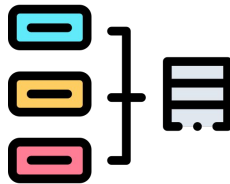
Son ordenadas, dado que mantienen el orden en el que han sido definidas.



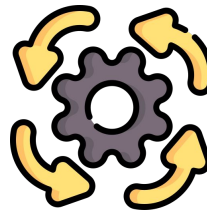
Pueden ser formadas por tipos arbitrarios.



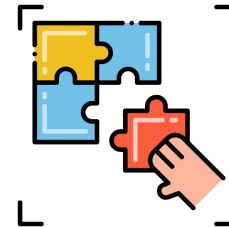
Pueden ser indexadas con [i]



Se pueden anidar, es decir, incluir una lista dentro de otra.



Son mutables, ya que sus elementos pueden ser modificados.



Son dinámicas, ya que se pueden añadir o eliminar elementos.



Referencia:
<https://ellibrodepython.com/listas-en-python>

Listas



En cada caja podemos guardar un valor, sea este una variable numérica, cadenas de caracteres, lógicos, incluso una lista.



```
lista=[7, 'Algoritmos', True, 13.141789, [1,2,3,4]]
```

Listas

Se asemejan bastante a los Arrays, si consideramos otros lenguajes de programación.

Si necesitamos incluir más de un valor dentro de la lista, sólo los separamos con coma.



```
▶ lista = [1, 2, 3, 4]
```

Listas: Creación



```
notas=[5.2 , 7, 6.9, 4.8]
```



Nombre de la
lista

Los paréntesis de
corchete indican que
es una lista



Listas: Elementos



```
miLista=[6, 'I Love', 8.61]
```

El ejemplo muestra que posee 3 elementos y que para acceder a ellos, sólo se utiliza un índice.



```
print(a[0])  
print(a[1])  
print(a[2])
```

- El elemento 1 está en la posición 0
- El elemento 2 está en la posición 1
- El elemento 3 está en la posición 2

Listas: Elementos

Para acceder al último elemento de la lista, sólo se utiliza el índice -1

```
miLista=[6, 'I Love', 8.61]  
print(miLista[-1])
```

8.61



Resultado

Si considero el índice -2, ¿cuál será el resultado?

Operaciones con Listas

Append(<obj>)

El método `append()`, permite agregar elementos al final de la lista.



```
miLista.append('amiguitos')  
print(miLista)
```

```
[6, 'I Love', 8.61, 'amiguitos']
```

Operaciones con Listas

Extend(<iterable>)

El método extend(), permite añadir una lista a una lista inicial.



```
lista = [1, 2]  
lista.extend([3, 4])  
print(lista) #[1, 2, 3, 4]
```

```
[1, 2, 3, 4]
```

Operaciones con Listas

Insert(<index>,<obj>)

El método insert(), permite añadir un elemento en una posición determinado.



```
lista = [1, 3]  
lista.insert(1, 2)  
print(lista) #[1, 2, 3]
```

```
[1, 2, 3]
```



Operaciones con Listas

Remove(<obj>)

El método remove(), permite recibir como argumento un objeto y lo borra de la lista.

```
[6, 'I Love', 8.61, 'amiguitos']
```



```
miLista.remove(8.61)  
print(miLista)
```

```
[6, 'I Love', 'amiguitos']
```

Operaciones con Listas

Pop()

El método `pop()`, elimina el último elemento de la lista.

```
▶ lista = [1, 2, 3]
  lista.pop()
  print(lista) #[1, 2]
```

[1, 2]

El método `pop(posición)`, elimina el elemento que se encuentra en la posición ingresada.

```
▶ lista = [1, 2, 3]
  lista.pop(1)
  print(lista) #[1, 3]
```

[1, 3]

Operaciones con Listas

Reverse()

El método reverse(), permite invertir el orden de la lista.



```
lista = [1, 2, 3]  
lista.reverse()  
print(lista) #[3, 2, 1]
```

```
[3, 2, 1]
```


Operaciones con Listas

Sort()

El método `sort()`, permite ordenar los elementos de menor a mayor.



```
lista = [3, 1, 2]  
lista.sort()  
print(lista) #[1, 2, 3]
```

 `[1, 2, 3]`



Operaciones con Listas

Sort()

El método `sort()`, también permite ordenar los elementos de mayor a menor.



```
lista = [3, 1, 2]  
lista.sort(reverse=True)  
print(lista) #[1, 2, 3]
```

[3, 2, 1]



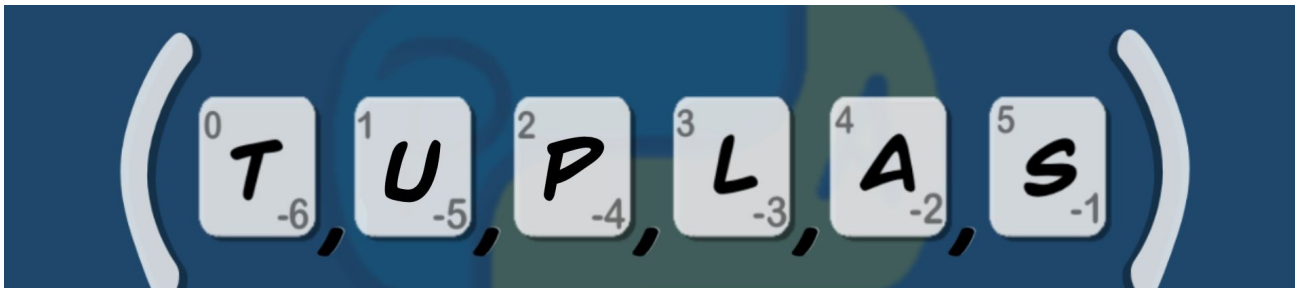
Tuplas

Las tuplas en Python o tuples son muy similares a las listas, pero con dos diferencias. Son **inmutables**, lo que significa que no pueden ser modificadas una vez declaradas, y **en vez de inicializarse con corchetes se hace con ()**. Dependiendo de lo que queramos hacer, las tuplas pueden ser más rápidas.

```
tupla = (1, 2, 3)
print(tupla) #(1, 2, 3)
```

```
tupla = (1, 2, 3)
#tupla[0] = 5 # Error! TypeError
```

inmutables



Métodos con Tuplas

Al igual que las listas, las tuplas también pueden ser anidadas.

```
tupla = 1, 2, ('a', 'b'), 3
print(tupla)          #(1, 2, ('a', 'b'), 3)
print(tupla[2][0])    #a
```

Y también es posible convertir una lista en tupla haciendo uso de la función tuple().

```
lista = [1, 2, 3]
tupla = tuple(lista)
print(type(tupla))    #<class 'tuple'>
print(tupla)          #(1, 2, 3)
```

Métodos con Tuplas

Se puede iterar una tupla de la misma forma que se hacía con las listas.

```
tupla = [1, 2, 3]
for t in tupla:
    print(t) #1, 2, 3
```

Y se puede también asignar el valor de una tupla con **n** elementos a **n** variables.

```
l = (1, 2, 3)
x, y, z = l
print(x, y, z) #1 2 3
```

count(<obj>)

El método count() cuenta el número de veces que el objeto pasado como parámetro se ha encontrado en la lista.

```
▶ tupla = (1,1,1,3,1,5)
print(tupla.count(1))
```

4

Métodos con Tuplas

index(<obj>[,index])

El método index() busca el objeto que se le pasa como parámetro y devuelve el índice en el que se ha encontrado.

```
[3] tupla = (9,1,10,7,1,5)
     print(tupla.index(7))
```

3

```
▶ tupla = (9,1,10,7,1,5)
   print(tupla.index(3))
```

```
↳ -----
ValueError                                Traceback (most recent call last)
<ipython-input-4-3b82e69f933d> in <cell line: 2>()
      1 tupla = (9,1,10,7,1,5)
----> 2 print(tupla.index(3))

ValueError: tuple.index(x): x not in tuple
```

Diccionarios



Diccionario

Un diccionario es una colección de elementos, donde cada uno tiene una llave **key** y un valor **value**. Los diccionarios se pueden crear con paréntesis **{ }** separando con una coma cada par **key: value**. En el siguiente ejemplo tenemos tres keys que son el nombre, la edad y el documento.

```
d1 = {  
    "Nombre": "Sara",  
    "Edad": 27,  
    "Documento": 1003882  
}  
print(d1)  
#{'Nombre': 'Sara', 'Edad': 27, 'Documento': 1003882}
```

```
d2 = dict([  
    ('Nombre', 'Sara'),  
    ('Edad', 27),  
    ('Documento', 1003882),  
)  
print(d2)  
#{'Nombre': 'Sara', 'Edad': '27', 'Documento': '1003882'}
```

Otra forma equivalente de crear un diccionario en Python es usando **dict()** e introduciendo los pares **key: value** entre paréntesis **()**.

Diccionario

Algunas propiedades de los diccionario en Python son las siguientes:

- **Dinámicos**: pueden crecer o decrecer, se pueden añadir o eliminar elementos.
- **Indexados**: los elementos del diccionario son accesibles a través del key.
- **Anidados**: un diccionario puede contener a otro diccionario en su campo value.

Acceder a los elementos:

Se puede acceder a sus elementos con `[]` o también con la función **get()**

```
print(d1['Nombre'])    #Sara  
print(d1.get('Nombre')) #Sara
```


Diccionario

Modificar los elementos:

Para modificar un elemento basta con usar `[]` con el nombre del key y asignar el valor que queremos.

```
d1['Nombre'] = "Laura"  
print(d1)  
#{'Nombre': Laura, 'Edad': 27, 'Documento': 1003882}
```

Si el key al que accedemos no existe, se añade automáticamente.

```
d1['Direccion'] = "Calle 123"  
print(d1)  
#{'Nombre': 'Laura', 'Edad': 27, 'Documento': 1003882, 'Direccion': 'Calle 123'}
```

Diccionario

Iterar:

Los diccionarios se pueden iterar de manera muy similar a las listas u otras estructuras de datos. Para imprimir los **key**.



```
for x in d1:  
    print(x)
```

```
Nombre  
Edad  
Documento  
Direccion
```

Se puede imprimir también solo el **value**. O si queremos imprimir el key y el value a la vez.



```
for x in d1:  
    print(d1[x])
```

```
Laura  
27  
1003882  
Calle 123
```



```
for x, y in d1.items():  
    print(x, y)
```



```
Nombre Laura  
Edad 27  
Documento 1003882  
Direccion Calle 123
```

Métodos con Diccionarios

`clear():`

El método **`clear()`** elimina todo el contenido del diccionario.

```
▶ d = {'a': 1, 'b': 2}
  d.clear()
  print(d)
```

```
☞ {}
```

`get(<key>[,<default>])`

El método **`get()`** nos permite consultar el value para un key determinado. El segundo parámetro es opcional, y en el caso de proporcionarse es el valor a devolver si no se encuentra la key.

```
▶ d = {'a': 1, 'b': 2}
  print(d.get('a'))
  print(d.get('z', 'No encontrado'))
```

```
1
No encontrado
```

Métodos con Diccionarios

items():

El método **items()** devuelve una lista con los keys y values del diccionario. Si se convierte en list se puede indexar como si de una lista normal se tratase, siendo los primeros elementos las key y los segundos los value.



```
d = {'a': 1, 'b': 2}
it = d.items()
print(it)
print(list(it))
print(list(it)[0][0])
print(list(it)[0][1])
print(list(it)[1][0])
print(list(it)[1][1])
```

```
dict_items([('a', 1), ('b', 2)])
[('a', 1), ('b', 2)]
a
1
b
2
```

Métodos con Diccionarios

keys():

El método keys() devuelve una lista con todas las keys del diccionario.

```
▶ d = {'a': 1, 'b': 2, 'c': 3}
  k = d.keys()
  print(k)
  print(list(k))
```

```
dict_keys(['a', 'b', 'c'])
['a', 'b', 'c']
```

values()

El método values() devuelve una lista con todos los values o valores del diccionario.

```
▶ d = {'a': 1, 'b': 2, 'c': 3}
  print(list(d.values()))
```

```
[1, 2, 3]
```

Métodos con Diccionarios

`pop(<key>[,<default>]):`

El método `pop()` busca y elimina la `key` que se pasa como parámetro y devuelve su valor asociado. Daría un error si se intenta eliminar una `key` que no existe.

```
d = {'a': 1, 'b': 2, 'c': 3}
d.pop('b')
print(d)
```

```
{'a': 1, 'c': 3}
```

También se puede pasar un segundo parámetro que es el valor a devolver si la `key` no se ha encontrado. En este caso si no se encuentra no habría error.

```
d = {'a': 1, 'b': 2, 'c': 3}
d.pop('x', -1)
print(d)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

Métodos con Diccionarios

popitem():

El método popitem() elimina el último par **clave: valor** del diccionario y lo devuelve. Si el diccionario está vacío se lanza la excepción KeyError (****en versiones anteriores a Python 3.7, el valor eliminado podía ser aleatorio**)

```
▶ d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
  d.popitem()

('d', 4)
```

update(<obj>):

El método update() se llama sobre un diccionario y tiene como entrada otro diccionario. Los value son actualizados y si alguna key del nuevo diccionario no esta, es añadida.

```
▶ d1 = {'a': 1, 'b': 2}
  d2 = {'a': 0, 'd': 400}
  d1.update(d2)
  print(d1)

{'a': 0, 'b': 2, 'd': 400}
```

Recorrer Diccionarios

```
▶ capitales = {  
    'Chile': 'Santiago',  
    'Peru': 'Lima',  
    'Ecuador': 'Quito',  
}  
for pais in capitales:  
    print('La capital de', pais, 'es', capitales[pais])
```

```
☞ La capital de Chile es Santiago  
La capital de Peru es Lima  
La capital de Ecuador es Quito
```


Recorrer Diccionarios

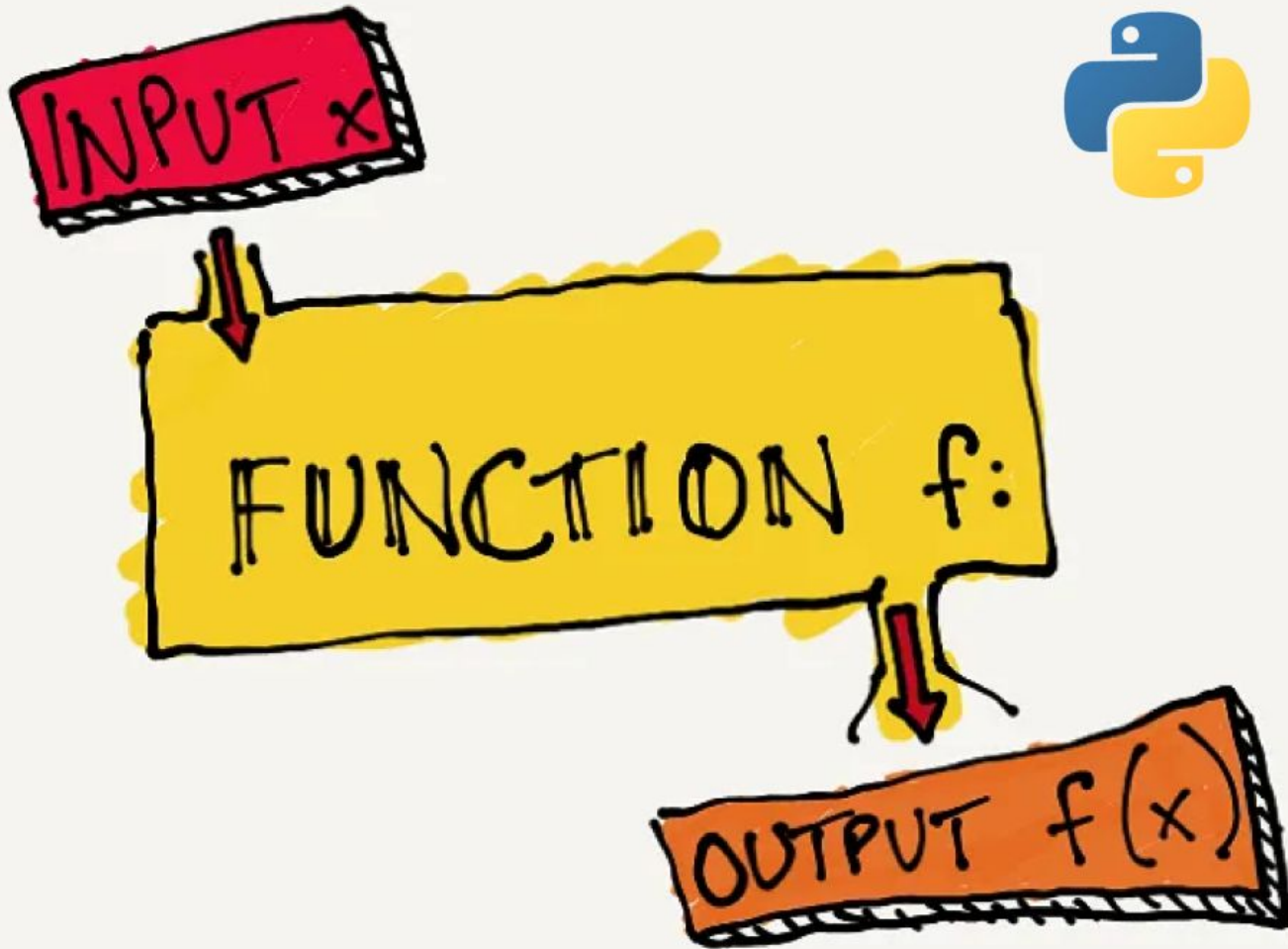
```
▶ for capital in capitales.values():  
    print(capital, 'es una linda ciudad')
```

```
↳ Santiago es una linda ciudad  
Lima es una linda ciudad  
Quito es una linda ciudad
```

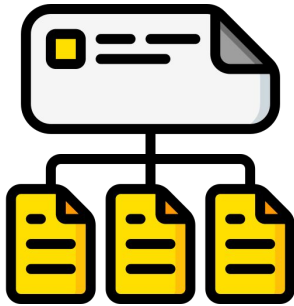
```
▶ for p, c in capitales.items():  
    print(c, 'es la capital de', p)
```

```
Santiago es la capital de Chile  
Lima es la capital de Peru  
Quito es la capital de Ecuador
```

Funciones



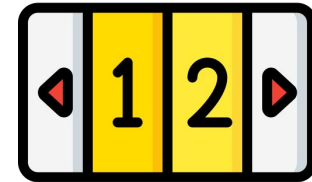
Funciones



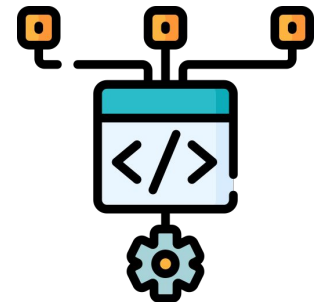
Son aquellas que dividen el programa en partes, considerando la parte principal y los diferentes métodos (tareas) que deben proporcionar resultados.



Se ejecutan sólo cuando son llamadas.



Pueden ser llamadas las veces que se requieran.



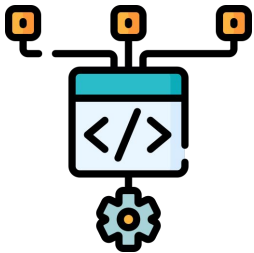
Sintaxis de las Funciones

```
def mi_funcion():  
    instrucciones
```

.....

La estructura de la función se compone de la siguiente forma:

1. Def palabra reservada (con minúscula)
2. Nombre descriptivo para la función
3. Paréntesis
4. Dos Puntos (:)
5. Las instrucciones, las cuales cumplen la misma norma que si fuera una sentencia de control.



Tipos de Funciones

1. Sin Argumentos y sin retorno

```
def saludo():  
    print("Saludando a mis estudiantes")
```

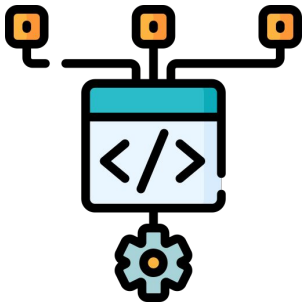
```
saludo()
```

```
Saludando a mis estudiantes
```

Función saludo(), sin argumentos y emite mensaje propio de la función

Llamada a función saludo(), no envía argumentos

Resultado de la llamada a la función saludo().



Tipos de Funciones

2. Sin Argumentos y con retorno



```
def sumar():  
    num1 = 3  
    num2 = 5  
    return(num1 + num2)
```

```
print("La suma es: ", sumar())
```

```
La suma es: 8
```



Función sumar() sin argumentos, posee dos variables con valor y retorna la suma de ellas.



Instrucción que invoca la función sumar() y espera el resultado para ser mostrado por pantalla.



Resultado del proceso de la función que proporciona un valor final.

Tipos de Funciones

3. Con Argumentos y sin retorno

```
def restar(valor1, valor2):  
    resultado=valor1-valor2  
    print(f"{valor1} - {valor2} = {resultado}")
```

Función restar() con argumentos, posee dos variables con valor y no retorna la resta de ellas, sino que muestra el resultado en la misma función.

```
while True:  
    try:  
        a=int(input("Ingrese primer valor: "))  
        b=int(input("Ingrese segundo valor: "))  
        break  
    except:  
        print("Ingresar solamente números!!")  
restar(a,b)
```

Solicitud de los argumentos numéricos.

Llamada a la Función sumar() con dos argumentos.

```
Ingrese primer valor: 800  
Ingrese segundo valor: 300  
800 - 300 = 500
```

Ingreso de los argumentos numéricos.

Muestra resultado del proceso realizado por la Función sumar().

Tipos de Funciones

4. Con Argumentos y con retorno

```
def multiplicar(x,z):  
    return x*z  
  
while True:  
    try:  
        x=int(input("Ingrese primer valor: "))  
        z=int(input("Ingrese segundo valor: "))  
        break  
    except:  
        print("Ingresar solamente números!!")  
  
print(multiplicar(x,z))
```

Función sumar() con argumentos, posee dos variables con valor y retorna la suma de ellas.

Solicitud de los valores a las variables.

En la impresión hace llamada a la Función multiplicar() con argumentos.

Ingrese primer valor: 8
Ingrese segundo valor: 9

Ingresa los valores a los argumentos.

72

Muestra el resultado final del proceso de la función multiplicar().

Otras Funciones



```
def varios_valores(*args):  
    for arg in args:  
        print(arg)
```

```
varios_valores(4.5, "Buen dia", [1, 2, 3, 4, 5])
```

4.5

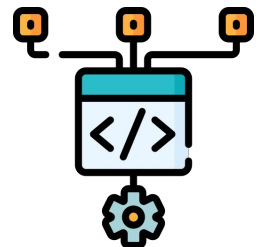
Buen dia

[1, 2, 3, 4, 5]

Función `varios_valores()` que recibe una lista dinámica de argumentos y por ello, se debe incluir un `*`. Esto se llama recibir parámetros indeterminados por posición.

Llamada a la Función `varios_valores()` que envía diferentes argumentos.

Resultados del proceso de la Función `varios_valores()`.



Otras Funciones

```
def mostrar_valores():  
    return("Buen dia", 15, [10,20,30])
```

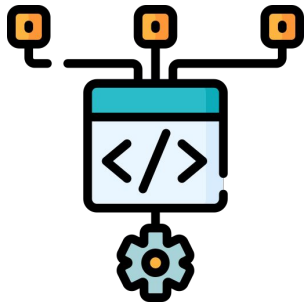
Función `mostrar_valores()` que no recibe parámetros, sin embargo retorna valores múltiples, esto se llama "Retorno múltiple".

```
mostrar_valores()
```

Llamada a la Función `mostrar_valores()` que no envía argumentos.

```
('Buen dia', 15, [10, 20, 30])
```

Resultados del proceso de la Función `mostrar_valores()`.



***Este retorno también puede asignarse a la misma cantidad de variables que retorna la función**

Actividad 1

Analiza con tus compañeros las siguientes funciones y comenta el resultado con tu docente.



1

.

```
def resta(a, b):  
    return a - b  
  
resta(30, 10)
```

Resultado

4

.

```
def resta(a=None, b=None):  
    if a == None or b == None:  
        print("Error, faltan parámetros a la función")  
        return  
    return a - b  
  
resta()
```

Resultado

2

.

```
def resta(a, b):  
    return a - b  
  
resta(b=30, a=10)
```

5

.

```
def calculo(precio, descuento):  
    return precio - (precio * descuento / 100)  
  
datos = [10000, 10]  
print("El monto final a pagar es: ", calculo(*datos))
```

3

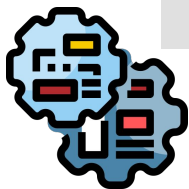
.

```
def funcion():  
    return "Bienvenidos a Python"  
  
frase = funcion()  
print(frase)
```

6

.

```
def saludo(nombre, mensaje='Python'):  
    print(mensaje, nombre)  
  
saludo(mensaje="Buen día", nombre="Pedro")
```



Actividad 2

Se pide escribir las instrucciones necesarias para crear un menú con las opciones de:

- Calcular_Iva
- Descuento
- Calcular_Imc

Las cuales deben ser desarrolladas en funciones (métodos).

Donde:

- Calcular_Iva: Es el precio del producto, multiplicado por el 19% (0.19)
- descuento: Es el precio del producto menos el descuento por aplicar. Mostrar el valor final del producto.
- Calcular_Imc: Índice de masa corporal. Su fórmula es: $\text{peso} / \text{estatura}^2$

Además se debe mostrar el estado de la persona de acuerdo a la siguiente tabla:

< 18.5	Bajo peso
18.5 - 24.9	Adecuado
25.0 - 29.9	Sobrepeso
30.0 - 34.9	Obesidad grado 1
35.0 - 39.9	Obesidad grado 2
>40	Obesidad grado 2