

# Curso Python 2024

## Teoría-3:

Argumentos, `*args`,  
`**kwargs`



# Hoy:

- ✓ Repaso: Unpacking
- ✓ Operador \*\*
- ✓ Positional args
- ✓ Keyword args
- ✓ \*args, \*\*kwargs



# Repaso: Unpacking

- Desempaquetar valores de un iterable en variables individuales
- Para secuencias
- Legibilidad

```
a, b, c, d = my_iterable
```

# Repaso: Operador \*

- Desempaquetar el resto de un iterable en una sola variable
- Desempaquetar un iterable en varios argumentos

`c = [1, 2, 3, 4]`  
`a, *b = c`  $\longrightarrow$  `a = 1`  
`b = [2, 3, 4]`

```
def fun(a, b, c, d):  
    print(a, b, c, d)
```

```
# Driver Code
```

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

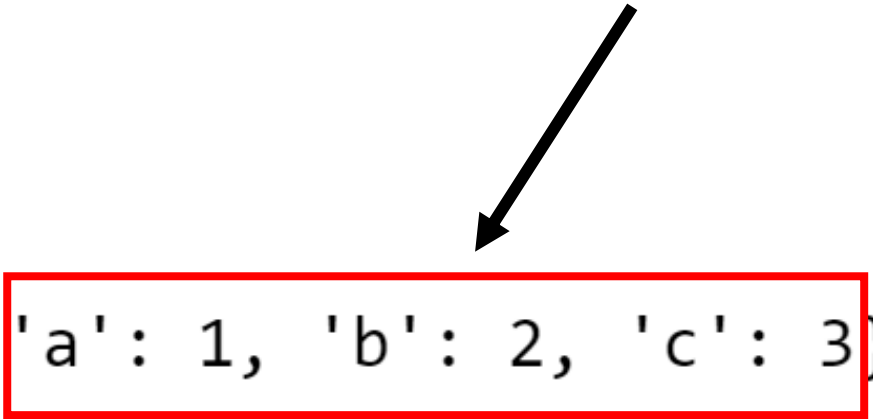
```
# Unpacking list into four arguments  
fun(*my_list)
```

# Operador \*\*

- Desempaquetar en diccionario
- Válido en argumentos de función (**clave-valor**) y dentro diccionarios

```
d1 = {'a': 1, 'b': 2, 'c': 3}
d2 = {'x': 10, 'y': 20, **d1}
```

`{'x': 10, 'y': 20, 'a': 1, 'b': 2, 'c': 3}`



# Operador \*\*

- Desempaquetar en argumento de función es equivalente a obtener argumentos opcionales **clave-valor**

```
def my_fun(a=5, b=4):  
    return a**2 + b**3  
  
d = dict(b=2)  
my_fun(**d, a=1)
```

# Argumentos vs Parámetros

- **Parámetros**

Nombres de variables locales

- **Argumentos**

Se pasan por referencia (memoria)

```
def my_fun(a, b):  
    # Some code  
    return
```

```
x1 = 100  
x2 = 'Hello'
```

```
my_fun(x1, x2)
```

# Argumentos posicionales

- Positional parameters
- Positional arguments

```
def my_fun(a, b):  
    return a**2 + b**3
```

```
my_fun(2, 3)
```



```
a = 2  
b = 3
```



# Parámetros por defecto

- Podemos dar valores por defecto a parámetros posicionales

```
def my_fun(a, b=10):  
    return a**2 + b**3
```

`my_fun(2, 3)`



`a = 2`  
`b = 3`

`my_fun(2)`



`a = 2`  
`b = 10`

# Parámetros por defecto: Cuidado

- Si un parámetro posicional tiene un valor por defecto los siguientes al mismo también deben tener valor por defecto

```
def my_fun(a, b=5, c):  
    return a**2 + b**3 + c**4
```



my\_fun(2, 3) → ???

↓

a

# Argumentos keyword (named)

- En ocasiones necesitamos keyword arguments
- Ejemplo: queremos especificar el 1º y 3º argumentos pero no el 2º (tomará el valor por defecto)

```
def my_fun(a, b=3, c=4):  
    return a**2 + b**3 + c**4
```

```
my_fun(2, c=10)
```



```
a = 2  
b = 3  
c = 10
```

# Argumentos keyword (named)

- De manera opcional podemos utilizar argumentos keyword para no respetar el orden posicional de argumentos posicionales

```
def my_fun(a, b, c):  
    return a**2 + b**3 + c **4  
  
my_fun(b=1, c=2, a=3)
```

# Argumentos keyword (named): Cuidado

- Al igual que ocurría con los parámetros una vez especificamos un argumento keyword los siguientes todos deben ser keyword.

```
def my_fun(a, b, c):  
    return a**2 + b**3 + c **4  
  
my_fun(b=1, 2, 3)
```

# \*args

- Similar a como ocurría en unpacking
- El operador \* nos permite guardar un número indefinido de valores posicionales en una variable (tupla)

```
def my_fun(a, b, *c):  
    return a**2 + b**3 + sum(c)**4  
  
my_fun(2, 2, 1, 1, 3, 2)
```

# \*args

- Diremos que hemos agotado los argumentos posicionales (exhausted)
- No podemos añadir más parámetros posicionales después de una variable con \*

OK pero habrá que usar keyword argument para d

```
def my_fun(a, b, *c, d):  
    return a**2 + b**3 + sum(c)**4 + d
```

```
my_fun(2, 2, 1, 1, 3, 2) ❌
```

???

# Resumen: positional + \* + keyword

- **a**: posicional obligado
- **b**: posicional opcional
- **args**: atrapa los posicionales restantes
- **\***: agota los parámetros posicionales
- **d**: keyword obligado
- **e**: keyword opcional

```
def my_fun(a, b=2, *args, d, e=10):  
    return
```



# \*\*kwargs

- Podemos desempaquetar diccionarios en argumentos keyword con \*\*
- Podemos guardar en variable un número indefinido de argumentos keyword (diccionario)

```
def my_fun(*args, **kwargs):  
    return args, kwargs
```

# \*\*kwargs

- \*\* Agotará los argumentos keyword
- Ejemplo:

```
def my_fun(*args, **kwargs):  
    return args, kwargs
```

```
my_fun(*(1, 2), **{'hola':1, 'mundo':2}, z=3)
```

Deben ser strings



# Problema venv: Windows

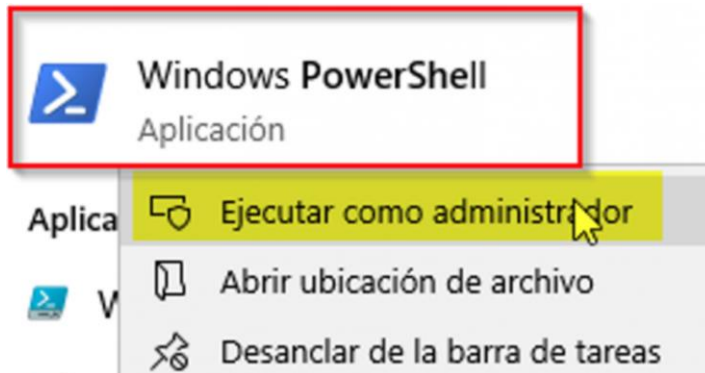
```
Windows PowerShell
PS C:\Users\Arturo\Documents> & '.\First Script.ps1'
& : No se puede cargar el archivo C:\Users\Arturo\Documents\First Script.ps1 porque la ejecución de scripts está
deshabilitada en este sistema. Para obtener más información, consulta el tema about_Execution_Policies en
https://go.microsoft.com/fwlink/?LinkID=135170.
En línea: 1 Carácter: 3
+ & '.\First Script.ps1'
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\Arturo\Documents>
```



Get-ExecutionPolicy -List

```
Windows PowerShell
PS C:\Users\Arturo\Documents> Get-ExecutionPolicy -List

Scope ExecutionPolicy
-----
MachinePolicy Undefined
UserPolicy Undefined
Process Undefined
CurrentUser Undefined
LocalMachine Undefined
```



Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

```
Windows PowerShell
PS C:\Users\Arturo\Documents> Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```