

NIVEL 2

DICCIONARIOS: MUTABILIDAD, BORRADO
DE DATOS & PARÁMETROS POR
REFERENCIA



LOS DICCIONARIOS

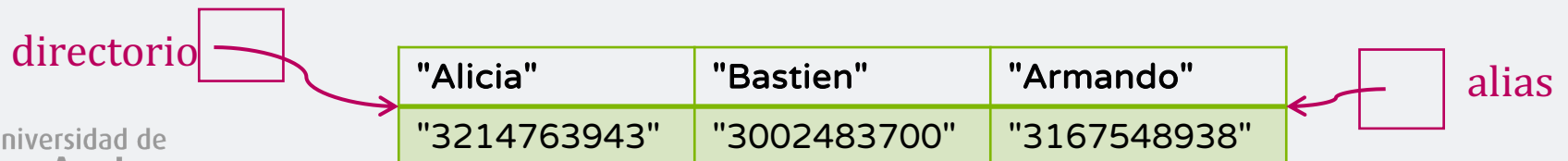


SON MUTABLES

- ✓ Cuando 2 variables referencian al mismo diccionario, los cambios en uno afectan al otro

```
Terminal 2/A  
In [129]: print(directorio)  
{'Alicia': '3231 476 39 04', 'Bastien': '300 248 37 00', 'Armando':  
'316 754 89 38'}  
  
In [130]: alias = directorio  
  
In [131]: print(alias)  
{'Alicia': '3231 476 39 04', 'Bastien': '300 248 37 00', 'Armando':  
'316 754 89 38'}
```

Las variables `alias` y `directorio` **referencian** al mismo diccionario. Esto decir que «**apuntan**» a la misma zona de memoria. No guardan el diccionario por dentro, sino una referencia a este. Por eso las pintamos como flechas:





LOS DICCIONARIOS SON MUTABLES

```
Terminal 2/A ✕  
  
In [129]: print(directorio)  
{'Alicia': '3231 476 39 04', 'Bastien': '300 248 37 00', 'Armando':  
'316 754 89 38'}  
  
In [130]: alias = directorio  
  
In [131]: print(alias)  
{'Alicia': '3231 476 39 04', 'Bastien': '300 248 37 00', 'Armando':  
'316 754 89 38'}  
  
In [132]: alias["Alicia"]="Borrado"  
  
In [133]: print(alias)  
{'Alicia': 'Borrado', 'Bastien': '300 248 37 00', 'Armando': '316 754  
89 38'}  
  
In [134]: print(directorio)  
{'Alicia': 'Borrado', 'Bastien': '300 248 37 00', 'Armando': '316 754  
89 38'}
```

Las variables alias y directorio referencian al mismo diccionario

Si modificamos alias, se modifica también directorio!

MÉTODO COPY



Dado que los diccionarios son mutables, si queremos conservar el diccionario original, tenemos que sacar una copia, con el método **copy**:

```
Terminal de IPython
Terminal 2/A x
In [136]: opuestos = {"arriba": "abajo", "derecha": "torcida", "si": "no",
"claro": "oscuro"}
In [137]: alias = opuestos
In [138]: print(alias)
{'arriba': 'abajo', 'derecha': 'torcida', 'si': 'no', 'claro': 'oscuro'}
In [139]: copia = opuestos.copy()
In [140]: print(copia)
{'arriba': 'abajo', 'derecha': 'torcida', 'si': 'no', 'claro': 'oscuro'}
```

Las variables
alias y opuestos
referencian al
mismo
diccionario

La variable copia referencia a
otro diccionario diferente

ANALICEMOS QUE PASA EN CADA CASO



Si modificamos alias...

```
Terminal 2/A ✕  
In [160]: alias["derecha"] = "izquierda"  
  
In [161]: print(opuestos)  
{'arriba': 'abajo', 'derecha': 'izquierda', 'si': 'no', 'claro': 'oscuro'}  
  
In [162]: print(alias)  
{'arriba': 'abajo', 'derecha': 'izquierda', 'si': 'no', 'claro': 'oscuro'}  
  
In [163]: print(copia)  
{'arriba': 'abajo', 'derecha': 'torcida', 'si': 'no', 'claro': 'oscuro'}
```

Modificamos también
opuestos

En cambio
copia no se
afecta

ANALICEMOS QUE PASA EN CADA CASO



Y si modificamos copia...

```
Terminal 2/A [X]

In [170]: copia["claro"] = "espeso"

In [171]: print(opuestos)
{'arriba': 'abajo', 'derecha': 'izquierda', 'si': 'no', 'claro': 'oscuro'}

In [172]: print(alias)
{'arriba': 'abajo', 'derecha': 'izquierda', 'si': 'no', 'claro': 'oscuro'}

In [173]: print(copia)
{'arriba': 'abajo', 'derecha': 'torcida', 'si': 'no', 'claro': 'espeso'}
```

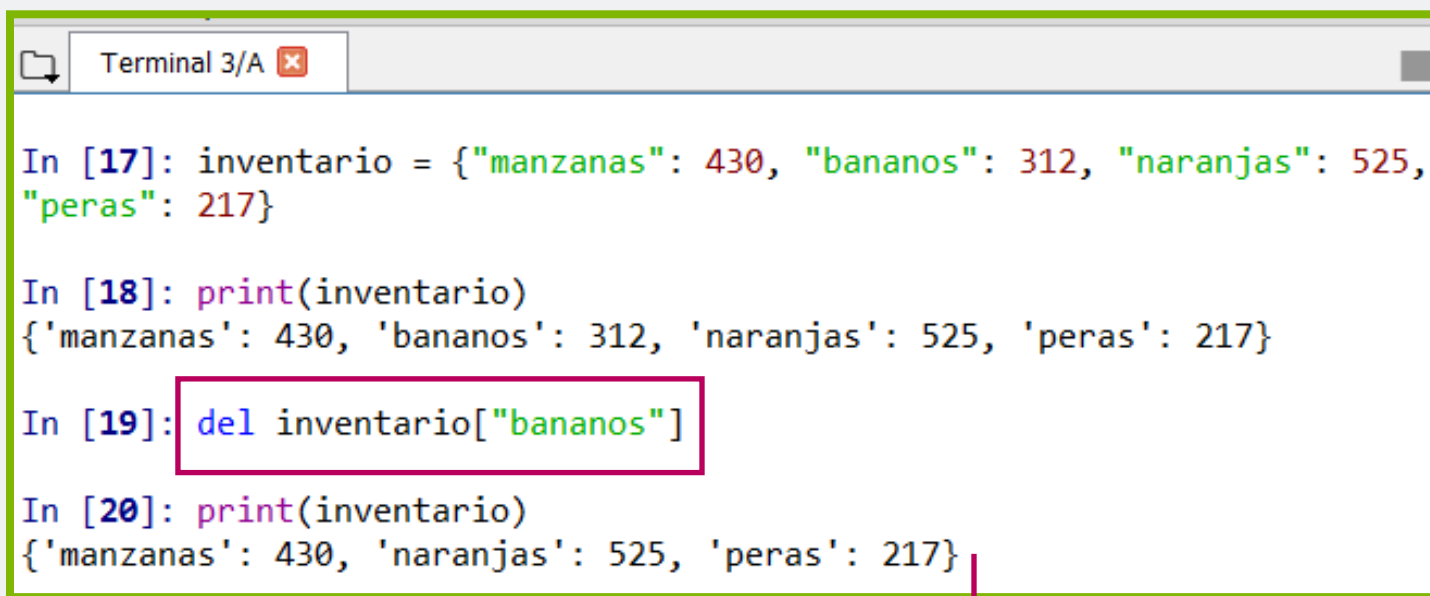
opuestos no se afecta

alias tampoco

El cambio solo
tiene efecto en
copia

BORRADO DE DATOS DE UN DICCIONARIO

La instrucción **del** suprime una pareja **clave:valor** de un diccionario



```
Terminal 3/A  
In [17]: inventario = {"manzanas": 430, "bananos": 312, "naranjas": 525,  
"peras": 217}  
  
In [18]: print(inventario)  
{'manzanas': 430, 'bananos': 312, 'naranjas': 525, 'peras': 217}  
  
In [19]: del inventario["bananos"]  
  
In [20]: print(inventario)  
{'manzanas': 430, 'naranjas': 525, 'peras': 217}
```

Note que inventario realmente se modifica!!

VEAMOS LA DIFERENCIA ENTRE UN PARÁMETRO POR VALOR Y UNO POR REFERENCIA



Parámetro por valor como de
tipos int, float, str, bool

Parámetro por referencia como
de tipo dict



PARÁMETRO DE TIPO ENTERO



```
EjemploFuncionConParametroDeTipoBasico.py ✕
1 def funcion_con_parametro_de_tipo_basico(a:int)->None:
2     print("\nBienvenid@ a la función que recibe un parámetro de tipo básico")
3     print("El parámetro a tiene el valor: ",a)
4     a += 1
5     print("Ahora a tiene el valor: ",a)
6
7
8 #PROGRAMA PRINCIPAL
9 var = int(input("Digite un número entero: "))
10 print("\nEl valor de var antes de llamar a la función es: ",var)
11 funcion_con_parametro_de_tipo_basico(var)
12 print("\nEl valor de var después de llamar a la función es: ",var)
```



¿Qué se imprime por pantalla?

LOS PARÁMETROS DE TIPO BÁSICO NO SE MODIFICAN

Aunque se modifique el valor de un parámetro ...

```
EjemploFuncionConParametroDeTipoBasico.py x
1 def funcion_con_parametro_de_tipo_basico(a:int)->None:
2     print("\nBienvenid@ a la función que recibe un parámetro de tipo básico")
3     print("El parámetro a tiene el valor: ",a)
4     a += 1
5     print("Ahora a tiene el valor: ",a)
6
7
8 #PROGRAMA PRINCIPAL
9 var = int(input("Digite un número entero: "))
10 print("\nEl valor de var antes de llamar a la función es: ",var)
11 funcion_con_parametro_de_tipo_basico(var)
12 print("\nEl valor de var después de llamar a la función es: ",var)
```

Resultado de la ejecución

```
Terminal 3/A x
Digite un número entero: 43
El valor de var antes de llamar a la función es: 43
Bienvenid@ a la función que recibe un parámetro de tipo básico
El parámetro a tiene el valor: 43
Ahora a tiene el valor: 44
El valor de var después de llamar a la función es: 43
```

Cuando la función termina, el parámetro conserva su valor original

PARÁMETRO DE TIPO DICT



```
EjemploFuncionConParametroDeTipoDict.py x
1 def funcion_con_parametro_de_tipo_dict(a: dict)->None:
2     print("\nBienvenid@ a la función que recibe un parámetro de tipo dict")
3     print("El diccionario original es:",a)
4     a["valor"] += 1
5     print("Ahora el diccionario es: ",a)
6
7
8 #PROGRAMA PRINCIPAL
9 var = int(input("Digite un número entero: "))
10 print("\nVoy a crear un diccionario con ese número como valor")
11 dicc = {"valor":var}
12 print("\nEl diccionario antes de llamar a la función es: ",dicc)
13 funcion_con_parametro_de_tipo_dict(dicc)
14 print("\nEl diccionario después de llamar a la función es: ",dicc)
```



¿Qué se imprime por pantalla?

LOS PARÁMETROS DE TIPO DICT SI SE MODIFICAN PORQUE SON REFERENCIAS

Si se modifica el contenido de un diccionario que entra como parámetro

```
EjemploFuncionConParametroDeTipoDict.py
1 def funcion_con_parametro_de_tipo_dict(a: dict)->None:
2     print("\nBienvenid@ a la función que recibe un parámetro de tipo dict")
3     print("El diccionario original es:",a)
4     a["valor"] += 1
5     print("Ahora el diccionario es: ",a)
6
7
8 #PROGRAMA PRINCIPAL
9 var = int(input("Digite un número entero: "))
10 print("\nVoy a crear un diccionario con ese número como valor")
11 dicc = {"valor":var}
12 print("\nEl diccionario antes de llamar a la función es: ",dicc)
13 funcion_con_parametro_de_tipo_dict(dicc)
14 print("\nEl diccionario después de llamar a la función es: ",dicc)
```

Resultado de la ejecución

Cuando la función termina, el diccionario ha sido modificado

```
Terminal 3/A
Digite un número entero: 34

Voy a crear un diccionario con ese número como valor

El diccionario antes de llamar a la función es: {'valor': 34}

Bienvenid@ a la función que recibe un parámetro de tipo dict
El diccionario original es: {'valor': 34}
Ahora el diccionario es: {'valor': 35}

El diccionario después de llamar a la función es: {'valor': 35}
```

A PROPÓSITO DEL NONE

- ✓ **None** significa en inglés «ninguno» y es un valor predefinido en **Python** que se usa para denotar «ausencia de valor»
- ✓ Es MUY usado en Python cuando una función no puede retornar un valor dado y así se precisa en el enunciado. Por ejemplo:

Diseñe una función que devuelva la solución de una ecuación lineal. Si la ecuación tiene infinitas soluciones o no tiene solución alguna, la función lo debe detectar y devolver el valor **None**



Más adelante trabajaremos con ejercicios como este, así que es muy importante recordar el **None**