

Laboratorio de Datos



1er. Cuatrimestre
2024



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA



Laboratorio de Datos

1er Cuatrimestre 2024

Introducción a Python

Contenido

- + Python
- + Tipos de datos básicos - operaciones
- + Tipos de datos que contienen otros datos.
- + Condicional y ciclos
- + Correr archivos .py
- + Funciones
- + Copias
- + IDE - Spyder
- + Diccionarios

Python

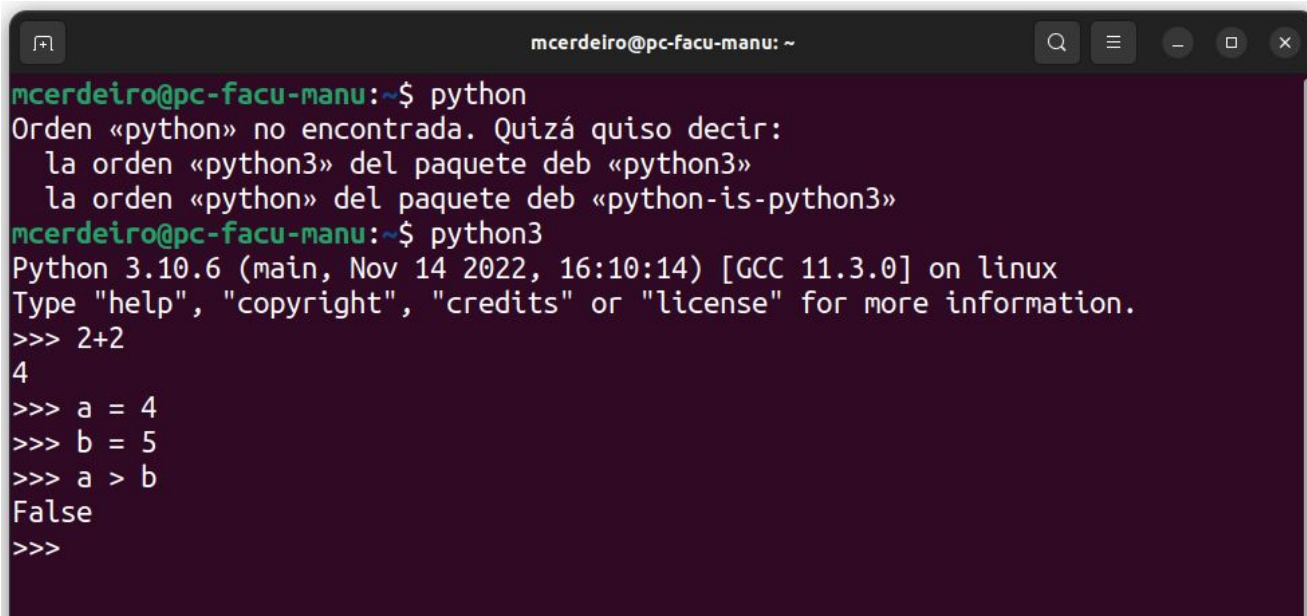


Instalación Python

- + código abierto (open source)
- + multiplataforma (sirve en linux, ios, windows...)
- + tiene muchas herramientas para ciencias de datos (y para muchas otras disciplinas)
- + alto nivel, sencillo de aprender
- + mucha mucha gente lo usa - facilita la consulta

Python desde la terminal

En una terminal, usamos "python" o "python3" para abrir un intérprete de python.

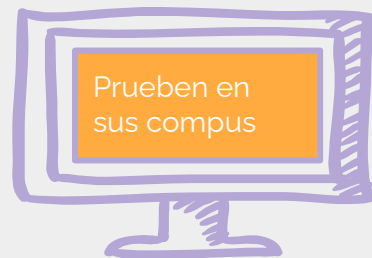


```
mcerdeiro@pc-facu-manu: ~  
mcerdeiro@pc-facu-manu:~$ python  
Orden «python» no encontrada. Quizá quiso decir:  
  la orden «python3» del paquete deb «python3»  
  la orden «python» del paquete deb «python-is-python3»  
mcerdeiro@pc-facu-manu:~$ python3  
Python 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 2+2  
4  
>>> a = 4  
>>> b = 5  
>>> a > b  
False  
>>>
```

Python desde la terminal

Al terminar, salimos con `exit()`

```
mcerdeiro@pc-facu-manu: ~  
mcerdeiro@pc-facu-manu:~$ python  
Orden «python» no encontrada. Quizá quiso decir:  
  la orden «python3» del paquete deb «python3»  
  la orden «python» del paquete deb «python-is-python3»  
mcerdeiro@pc-facu-manu:~$ python3  
Python 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 2+2  
4  
>>> a = 4  
>>> b = 5  
>>> a > b  
False  
>>> exit()  
mcerdeiro@pc-facu-manu:~$
```



Asignación de variables

```
a = 3
```

Esta instrucción crea una variable llamada **a** y le asigna el valor 3.

Si ya había una variable **a**, la pisa. Ahora la variable **a** vale 3.

```
a = 5 # ahora a vale 5
```

```
# el contenido posterior a un símbolo numeral (#)
```

```
# es un comentario
```

```
# python no lo ejecuta
```


Tipos de datos básicos

Tipos de datos básicos

- + int: representan números enteros, 1, 2, -5, 102978
- + float: representan números reales, 56.842
- + bool: representan valores booleanos, True/False (1/0).

Números enteros

280 + 16 # Este valor no se lo asigna a ninguna variable

base = 3

altura = 4

sup_rectangulo = base*altura

Calcula la superficie del rectángulo

20//3 # el símbolo // es para la división entera

15%4 # el símbolo % calcula el resto de la división entera

Variables float

base = 3

altura = 5

sup_trianguelo = (base*altura)/2

Variables booleanas

Operaciones con bool

- + and
- + or
- + not

Representan la conjunción,
disyunción y negación,
respectivamente.

`a = True`

`b = False`

`c = a and b`

`d = a or b`

`e = not a or b # $a \rightarrow b$`

Operaciones de int/float a bool

Al realizar comparaciones obtenemos valores de verdad (bool):

```
a = 8
```

```
b = 15
```

```
x = 2
```

```
a > b # False
```

```
(a > b) or (a > x) # True
```

Ejercicio. ¿Qué valor de verdad se obtiene?

```
(x < a) and (10*x >= b) and not (x == a - b)
```

Datos que contienen datos

- + Cadenas de caracteres
- + Listas
- + Tuplas
- + Conjuntos

Cadenas de caracteres - *Strings*

Son la forma de manipular texto.

Se construyen con comillas simples o dobles.

```
mcerdeiro@clemen-i7:~$ python3
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 'Hoy es lunes'
>>> b = 'empiezo labo de datos'
>>> c = a + b
>>> c
'Hoy es lunesempiezo labo de datos'
>>> c = a + ', ' + b
>>> c
'Hoy es lunes, empiezo labo de datos'
>>> 
```


Cadenas de caracteres - *Strings*

Posiciones y porciones

```
a = 'Hoy es lunes'
```

```
a[4] # lo que tiene a en la posición 4
```

```
a[4:8] # de 4 hasta pre-8
```

```
a[4:]
```

```
a[:5]
```

```
a[1] = 'x'
```



Cadenas de caracteres - *Strings*

La función **print** imprime en pantalla:

```
>>> print(a)
Hoy es lunes
>>> print('ahora imprimo otra cosa')
ahora imprimo otra cosa
>>> x = 5
>>> print(x)
5
>>> listita = [1,2,3,4]
>>> print(listita)
[1, 2, 3, 4]
>>>
```

Cadenas de caracteres - *Strings*

```
a = 'Hello' + 'World'    # concatenación: 'HelloWorld'
b = 'Say ' + a           # 'Say HelloWorld'
s = 'Hello '

len(s)                   # longitud (da 5)
'e' in s                 # test de pertenencia
'x' in s                 # False
'hi' not in s            # True
rep = s * 5              # 'Hello Hello Hello Hello Hello '
l = s.lower()            # 'hello '
u = s.upper()            # 'HELLO '
ss = s.strip()           # 'Hello'
```

Código de escape

Los códigos de escape (escape codes) son expresiones que comienzan con una barra invertida, \ y se usan para representar caracteres que no pueden ser fácilmente tipeados directamente con el teclado. Estos son algunos códigos de escape usuales:

<code>'\n'</code>	Avanzar una línea
<code>'\r'</code>	Retorno de carro
<code>'\t'</code>	Tabulador
<code>'\''</code>	Comilla literal
<code>'\"'</code>	Comilla doble literal
<code>'\\'</code>	Barra invertida literal

f- Strings (formatted string)

- Permite incluir variables de manera formateada.
- Se construyen agregando una f antes de la cadena.

Por ejemplo:

```
>>> lista = ['Ana', 'Bianca', 'Carolina']
>>> s = f'lista de alumnas:\n{lista}'
>>> print(s)
lista de alumnas:
['Ana', 'Bianca', 'Carolina']
>>>
```

```
>>> especie = 'Ombú'
>>> edad = 12
>>> altura = 6.4
>>> a = f'Especie: {especie:<10s} Edad: {edad:<10d} Altura: {altura:<10.2f}'
>>> print(a)
Especie: Ombú          Edad: 12          Altura: 6.40
>>>
```

Códigos de formato

d	Entero decimal
b	Entero binario
x	Entero hexadecimal
f	Flotante como [-]m.dddddd
e	Flotante como [-]m.dddddde+-xx
g	Flotante, pero con uso selectivo de la notación exponencial E.
s	Cadenas
c	Caracter (a partir de un entero, su código)

:>10d	Entero alineado a la derecha en un campo de 10 caracteres
:<10d	Entero alineado a la izquierda en un campo de 10 caracteres
:^10d	Entero centrado en un campo de 10 caracteres
:0.2f	Flotante con dos dígitos de precisión

Listas

Pueden contener todo tipo de variables.

Las listas se usan mucho. Se construyen con corchetes [].

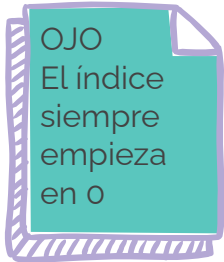
```
lista_num = [10, 43, 22, 5, 63, 101, -5, 3]
```

```
lista_nombres = ['Julia', 'Luciana', 'Manuel']
```

```
lista_num[0] # accedo al primer elemento de la lista
```

```
lista_nombres[1] # 'Luciana'
```

```
lista_vacia = []
```



OJO
El índice
siempre
empieza
en 0

Listas

Las listas se usan mucho. Se construyen con corchetes [].

```
lista_num = [10, 43, 22, 5, 63, 101, -5, 3]
```

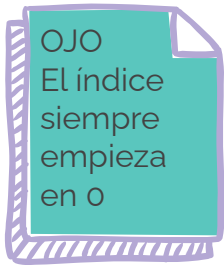
```
lista_num[0] # accedo al primer elemento de la lista
```

```
lista_num[3]
```

```
lista_num[7]
```

```
lista_num[1:5] # rebanada/slice
```

```
len(lista_num) # 8
```



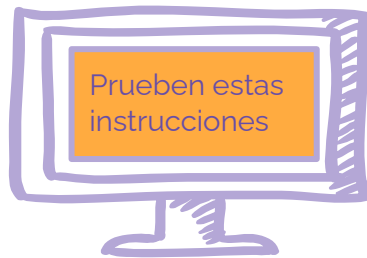
OJO
El índice
siempre
empieza
en 0

Listas

```
lista_num = [10, 43, 22, 5, 63, 101, -5, 3]
lista_nombres = ['Julia', 'Luciana', 'Manuel']
lista_num.append(28) # agrego un elemento al final
lista_nombres[1] = 'Lucía' # cambio el segundo elemento
22 in lista_num # me fijo si el elemento 22 está
'Manuel' in lista_nombres
25 not in lista_num # me fijo si NO está
nombres.remove('Julia')
lista_num.sort()
```

Listas

```
frutas = 'Frambuesa,Manzana,Naranja,Mandarina,Banana,Sandía,Pera'
lista_frutas = frutas.split(',') # separa en las comas
lista_frutas[0]    # Frambuesa
lista_frutas[1]
lista_frutas[-1]   # último elemento
lista_frutas[-2]   # Sandía
lista_frutas.append('Limón')
lista_frutas.insert(0, 'Limón') # insertar al principio
lista_frutas += ['Frutilla', 'Palta']
```



Listas

```
frutas = 'Frambuesa,Manzana,Naranja,Mandarina,Banana,Sandía,Pera'
lista_frutas = frutas.split(',') # separa en las comas
compras = ['café', lista_frutas, 'huevos'] # tiene una lista adentro!
compras[1] ?
compras[1][1] ?
compras[1][1][1] ???
```

compras

```
['café', lista_frutas, 'huevos']
```

compras[1]

```
['café', ['Frambuesa', 'Manzana', 'Naranja'], 'huevos']
```

compras[1][1]

```
['café', ['Frambuesa', 'Manzana', 'Naranja'], 'huevos']
```

compras[1][1][1]

```
mandados = [compras, 'cajero', 'pasear al perro']
```

```
len(mandados) ?
```

```
lista_frutas in mandados # ??
```

```
[compras, 'cajero', 'pasear al perro']
```

```
[['café', lista_frutas, 'huevos'], 'cajero', 'pasear al perro']
```

```
[['café', ['Frambuesa', 'Manzana', 'Naranja'], 'huevos'], 'cajero',  
'pasear al perro']
```

Matrices como listas de listas

Podemos utilizar listas de listas para definir matrices.
Cada lista representa una fila.

Ejemplos:

$$M = [[1, 2], [3, 4]]$$

1	2
3	4

$$N = [[1, 0, 3], [0, 5, 5], [8, -1, 1], [10, 3, 1]]$$

1	0	3
0	5	5
8	-1	1
10	3	1

Matrices como listas de listas

¿Cómo accedemos al elemento de la posición (i,j)?

`M[0] ?`

`M[0][1] ?`

1	2
3	4

`N[1][2] ?`

`N[?][?]` # quiero acceder al -1

1	0	3
0	5	5
8	-1	1
10	3	1

Tuplas

Son como vectores. Se arman con paréntesis ().

a = (2, 4)

b = (-1, 2)

c = a + b

d = (1, 5, 10, 3, 7)

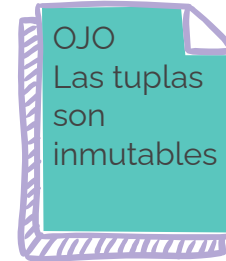
a[1]

d[9]

d[1:5]

d[2] = 11

d = (1, 5, 11, 3, 7)



Conjuntos

Se arman con llaves {}.

```
a = {2, 4}
```

```
citricos = {'Naranja', 'Limón', 'Mandarina'}
```

```
citricos = set(['Naranja', 'Limón', 'Mandarina'])
```

```
'Naranja' in citrics
```

```
citricos.add('Pomelo')
```

```
citricos.remove('Naranja')
```

```
len(a)
```

Ejercicio

Escribir un programa que decida si una palabra (de longitud 5) es palíndromo.

Ej. "NADAN" es palíndromo. "JUJUY" no lo es.

Condicional y ciclos

Condicional

Si queremos supeditar la instrucción a una determinada condición, utilizamos el condicional if.

```
if a > 0:
```

2 puntitos

nueva línea

```
    print('a es un número positivo!')
```

nueva línea

```
if a == 0:
```

```
    print('a es cero...')
```

```
if b > a:
```

```
    print('b le ganó a a')
```

Condicional

Ejercicio. Completar este código.

```
a = 2
```

```
b = 6
```

```
c = 4
```

```
if - - -:
```

```
    print('b está entre a y c')
```

```
if - - -:
```

```
    print('b es mayor a c')
```

Condicional

Else y elif.

```
if condicion1:
    instruccion1
    instruccion2
elif condicion2:
    instruccion3
else:
    instruccion4
```

Ejemplo

```
if a % 4 == 0:
    b = a//4
elif a % 4 == 2:
    c = a//2
else:
    print('a es impar')
```

Condicional

case en cpp

Else y elif.

```
if condicion1:
    instruccion1
    instruccion2
elif condicion2:
    instruccion3
else:
    instruccion4
```

condiciones: bool



Ejemplo

```
if a % 4 == 0:
    b = a//4
elif a % 4 == 2:
    c = a//2
else:
    print('a es impar')
```

Ciclo while

Para utilizar while vamos a considerar una condición que puede o no cumplirse (un bool True/False). Las instrucciones dentro del ciclo se ejecutarán mientras se satisface la condición.

```
i = 0
```

```
while i < 5:
```

```
    print(i)
```

```
    i += 1    # sumo de a 1
```


Ciclo while


Para utilizar while vamos a considerar una condición que puede o no cumplirse (un bool True/False). Las instrucciones dentro del ciclo se ejecutarán mientras se satisface la condición.

```
i = 0
```

```
while i < 5:
```

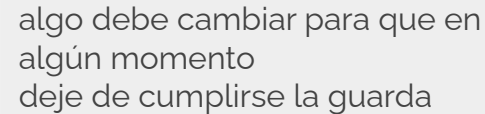
```
    print(i)
```

```
    i += 1    # sumo de a 1
```



guarda del while
donde figura la condición

A purple arrow points from this text box to the condition 'i < 5' in the 'while' statement of the code above.



algo debe cambiar para que en
algún momento
deje de cumplirse la guarda

A purple arrow points from this text box to the increment statement 'i += 1' in the code above.

Ejercicio

Escribir un programa que imprima en pantalla los números enteros entre 0 y 213 que sean divisibles por 13.

Ejercicio

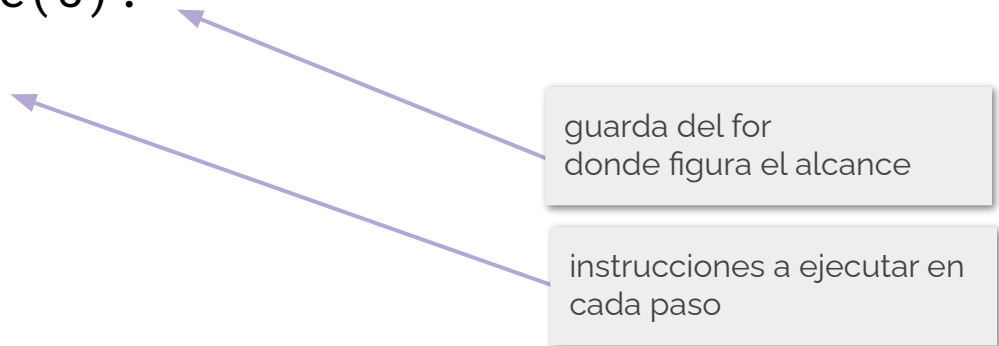
Una mañana ponés un billete en la vereda al lado del obelisco porteño. A partir de ahí, cada día vas y duplicás la cantidad de billetes, apilándolos prolijamente. ¿Cuánto tiempo pasa antes de que la pila de billetes sea más alta que el obelisco?

Datos: espesor del billete: 0.11 mm, altura obelisco: 67.5 m.

Ciclo for

Para utilizar el for vamos a considerar un iterador en la guarda. Las instrucciones dentro del ciclo for se ejecutarán en cada elemento generado por el iterador.

```
for i in range(5):  
    print(i)
```



guarda del for
donde figura el alcance

instrucciones a ejecutar en
cada paso

range()

```
for i in range(5):  
    print(i)
```

```
for i in range(0,5,1):  
    print(i)
```

```
for i in range(-5,-1):  
    print(i)
```

inicio, fin, paso

OJO
Fin significa
hasta antes
que ese
valor

valores por omisión:
inicio 0
paso 1

while vs. for

```
i = 0
```

```
while i < 5:
```

```
    print(i)
```

```
    i += 1
```

```
for i in range(0,5,1):
```

```
    print(i)
```

while vs. for

```
i = 0
```

```
while i < 5:
```

```
    print(i)
```

```
    i += 1
```

```
for i in range(0, 5, 1):
```

```
    print(i)
```

Ejercicio

Usá una iteración sobre el string cadena para agregar la sílaba 'pa', 'pe', 'pi', 'po', o 'pu' según corresponda luego de cada vocal.

```
cadena = 'Geringoso'
```

```
capadepenapa = ''
```

```
for c in cadena:
```

```
    COMPLETAR
```

```
print(capadepenapa)      # Geperipingoposopo
```

Luego hazelo con un while en vez del for.

Ejercicio

Una pelota de goma es arrojada desde una altura de 100 metros y cada vez que toca el piso salta $\frac{3}{5}$ de la altura desde la que cayó. Escribí un programa rebotes.py que imprima una tabla mostrando las alturas que alcanza en cada uno de sus primeros diez rebotes.

Tu programa debería generar una tabla que se parezca a esta:

```
1 60.0
2 36.0
3 21.6
4 12.96
5 7.776
6 4.6656
7 2.7994
8 1.6796
9 1.0078
10 0.6047
```

Archivos .py

Uso de archivos .py

Podemos guardar el código como archivo de texto con extensión .py y luego ejecutarlo desde la terminal con python. Por ejemplo:

Creamos un archivo de texto, con este contenido, y lo guardamos como holamundo.py

```
# holamundo.py
```

```
a = "Hola, mundo!"  
b = a[:4]  
c = a[6:]  
print(a)
```

Uso de archivos .py

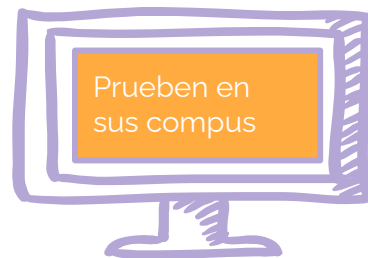
Ahora abrimos una terminal, nos situamos en el mismo directorio en el que está el archivo y ejecutamos "python3 holamundo.py"

```
mcerdeiro@clemen-i7:~/Downloads$ python3 holamundo.py  
Hola, mundo!
```

Uso de archivos .py

También podemos ejecutar con `-i` y quedarnos en el intérprete de python luego de la ejecución del programa. Las variables quedan en el estado final del programa.

```
mcerdeiro@clemen-i7:~/Downloads$ python3 holamundo.py
Hola, mundo!
mcerdeiro@clemen-i7:~/Downloads$ python3 -i holamundo.py
Hola, mundo!
>>> a
'Hola, mundo!'
>>> b
'Hola'
>>> c
'mundo!'
>>> b + c
'Holamundo!'
>>> exit()
mcerdeiro@clemen-i7:~/Downloads$
```



Funciones

Funciones

Las funciones son una herramienta para encapsular pedazos de código, facilitando su reutilización.

```
def sumar_le_uno(x):  
    res = x + 1  
    return res
```

```
def sumar(x,y):  
    res = x + y  
    return res
```

Funciones

Las funciones son una herramienta para encapsular pedazos de código, facilitando su reutilización.

```
def suma_gauss(n):  
    '''  
    Devuelve la suma de los primeros n enteros  
    '''  
    total = 0  
    while n > 0:  
        total += n  
        n -= 1  
    return total
```


Funciones

Definición de una función.

The diagram illustrates the syntax of a Python function definition with the following code and annotations:

```
def suma_gauss(n):  
    '''  
    Devuelve la suma de los primeros n enteros  
    '''  
    total = 0  
    while n > 0:  
        total += n  
        n -= 1  
    return total
```

Annotations and their corresponding code elements:

- nombre**: Points to the function name `suma_gauss`.
- parámetros / variables in**: Points to the parameter `n` in the function signature.
- 2 puntitos**: Points to the opening double quote of the docstring.
- 4 espacios**: Points to the four spaces of indentation for the first line of the function body.
- nueva línea**: Points to the start of a new line in the function body.
- out**: Points to the `return` statement, indicating the output of the function.

Funciones

Definición de una función.

```
def suma_gauss(n):  
    '''  
    Devuelve la suma de los primeros  
    n enteros  
    '''  
    total = 0  
    while n > 0:  
        total += n  
        n -= 1  
    return total
```

Llamada (uso) de una función.

```
suma_gauss(10)  
  
x = 25  
suma_gaus(x)
```

Ejercicios

- Definir una función **maximo(a,b)** que tome dos parámetros numéricos y devuelva el máximo.
- Definir una función **tachar_pares(lista)** que tome una lista de números y devuelva una similar pero donde los números pares estén reemplazados por 'x'.

Ejercicio

Queremos hacer un traductor que cambie las palabras masculinas de una frase por su versión neutra.

Como primera aproximación, intentaremos reemplazar todas las letras 'o' que figuren en el último o anteúltimo carácter de cada palabra por una 'e'.

Por ejemplo 'todos somos programadores' pasaría a ser 'todes somes programadores'.

```
>>> frase = 'todos somos  
programadores'  
>>> traductor_inclusivo(frase)  
'todes somes programadores'
```

Copias

Copias

Cuando creamos una lista igual a "a", al modificar una, se modifica la otra también.

```
a = [1,2,3]
b = a          # b = [1,2,3] igual que a
a.append(1)    # acá el 1 se agrega en ambas
b.append(2)    # acá el 2 se agrega en ambas
```

Copias

Las listas (y otros objetos) tienen un método para hacer copias. Cuando creamos una copia b de a, modificar una no tiene efecto sobre la otra.

```
a = [2,3,[100,101],4]
b = a.copy()
b == a # True
b is a # False
```

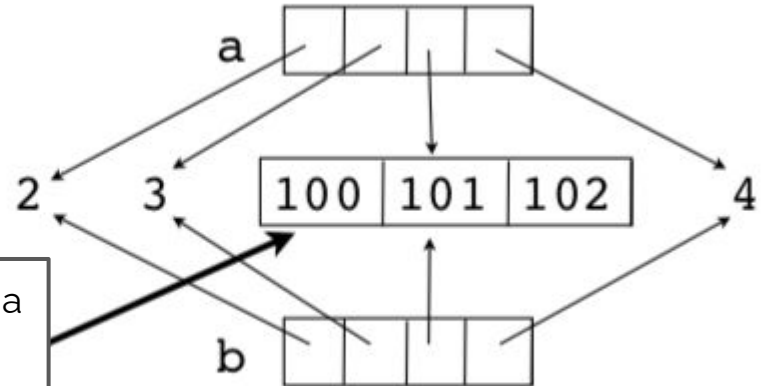
Ejemplo

```
a = [2,3,[100,101],4]  
b = a.copy()  
b == a # True  
b is a # False
```

```
a.append(5)  
print(b)
```

```
a[2].append(102)  
print(b)
```

Esta lista interna es la misma para a y b.



Python Tutor - <https://pythontutor.com/>

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6
[known limitations](#)

```
1 a = 3
→ 2 b = 10
→ 3 c = a + b
4 a = 5
```

[Edit this code](#)

→ line that just executed

→ next line to execute



<< First

< Prev

Next >

Last >>

Step 3 of 4

Frames

Objects

Global frame

a	3
b	10

Deepcopy

```
import copy
```

```
a = [2,3,[100,101],4]
```

```
b = copy.deepcopy(a)
```

```
a.append(5)
```

```
print(b)
```

```
a[2].append(102)
```

```
print(b)
```

Spyder



Entorno de desarrollo integrado (*IDE*) - Spyder

- + código abierto (open source)
- + multiplataforma (sirve en linux, ios, windows...)
- + es cómodo para escribir código, ejecutarlo, corregirlo, probarlo y utilizarlo, en el mismo entorno
- + el editor de texto remarca palabras clave del lenguaje
- + tiene atajos (shortcuts) útiles (y modificables a gusto de cada uno)

Entorno de desarrollo integrado (IDE) - Spyder

The screenshot displays the Spyder Python IDE interface. The main window is titled "Spyder (Python 3.8)". The menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar contains various icons for file operations, running, and debugging. The editor pane shows a file named "codigo_primeraclase.py" with the following code:

```
1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Created on Sun Feb 26 22:07:07 2023
5
6@author: mcerdeiro
7"""
8
9a = 3
10b = 10
11c = a + b
12a += 5
13
14
15
```

The Variable explorer pane on the right shows the following variables:

Name	Type	Size	Value
a	int	1	3
b	int	1	10

The IPython console pane at the bottom shows the following output:

```
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
Type "copyright", "credits" or "license" for more information.

IPython 7.13.0 -- An enhanced Interactive Python.

In [1]: a = 3
In [2]: b = 10
In [3]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: LF, Encoding: UTF-8, Line: 15, Column: 1, Memory: 41 %.

Entorno de desarrollo integrado (IDE) - Spyder

The image shows the Spyder Python IDE interface with several components and annotations:

- Editor:** The main window for editing code. It contains a file named `codigo_primeraclase.py`. The code is a Python script with a shebang, encoding declaration, docstring, and variable assignments.
- Variable explorer:** A panel on the right that displays the current state of variables in the program. It shows a table with columns for Name, Type, Size, and Value.
- IPython console:** A panel at the bottom right for running code interactively. It shows the output of the code executed in the editor.
- Annotations:** Four callout boxes with arrows pointing to specific parts of the interface: "nombre del programa" points to the file name in the editor; "directorio actual de trabajo" points to the current working directory in the top bar; "explorador de variables (muestra el estado del programa)" points to the Variable explorer; "intérprete de python" points to the IPython console.

Code in the editor:

```
1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Created on Sun Feb 26 22:07:07 2023
5
6@author: mcerdeiro
7"""
8
9a = 3
10b = 10
11c = a + b
12a += 5
13
14
15
```

Variable explorer table:

Name	Type	Size	Value
a	int	1	3
b	int	1	10

IPython console output:

```
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
Type "copyright", "credits" or "license" for more information.

IPython 7.13.0 -- An enhanced Interactive Python.

In [1]: a = 3
In [2]: b = 10
In [3]:
```

Status bar: Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 15 Column: 1 Memory: 41 %

Diccionarios

Diccionarios

Los diccionarios son útiles si vamos a querer buscar rápidamente (por claves).

- Se construyen con llaves
- Cada entrada tiene una clave y un valor, separados por dos puntitos :
- Las entradas se separan con comas

```
{clave1: valor1, clave2: valor2, ... }
```

- Se acceden con corchetes indicando una clave
- Tanto las claves como los valores pueden ser de distintos tipos de objetos
- Las claves deben ser de tipo inmutable

Ejemplo

```
dias_engl = {'lunes': 'monday', 'martes': 'tuesday', 'miércoles': 'wednesday', 'jueves':  
'thursday'}
```

```
>>> dias_engl['lunes']  
'monday'
```

```
>>> dias_engl['viernes']  
Traceback (most recent call last):
```

```
  File "<ipython-input-33-ee0fa133453b>", line 1, in <module>  
    dias_engl['viernes']
```

```
KeyError: 'viernes'
```

```
>>> dias_engl['viernes'] = 'friday'      # agrego la entrada  
>>> dias_engl['viernes']  
'friday'
```

Ejemplo

También se pueden armar y modificar agregando entradas:

```
feriados = {} # Empezamos con un diccionario vacío
```

```
# Agregamos elementos
```

```
feriados[(1, 1)] = 'Año nuevo'
```

```
feriados[(1, 5)] = 'Día del trabajador'
```

```
feriados[(13, 9)] = 'Día del programador'
```

```
# Modifico una entrada
```

```
feriados[(13, 9)] = 'Día de la programadora'
```

Diccionarios

También se pueden armar a partir de una lista de tuplas (clave, valor).

```
>>> cuadrados = dict([(1,1), (2,4), (3,9), (4,16)])  
>>> cuadrados[2]  
4
```

La función **zip** genera tuplas a partir de dos listas:

```
>>> for t in zip([1,2,3,4],[1,4,9,16]):  
    print(t)  
(1, 1)  
(2, 4)  
(3, 9)  
(4, 15)
```

Es decir que podemos construir el diccionario a partir de dos listas (claves y valores);

```
>>> cuadrados = dict(zip([1,2,3,4],[1,4,9,16]))
```

Cierre

- + Python
- + Tipos de datos - simples y compuestos
- + Condicional y ciclos - while y for
- + Correr archivos
- + Funciones
- + Copias
- + Spyder
- + Diccionarios