Facultad de Filología. Universidad Complutense de Madrid
Máster en Letras Digitales:
Estudios Avanzados en Textualidades Electrónicas

Prácticas en grupo de investigación FADoSS

Introducción a Python para lingüistas

Curso 2023-24

Alumna: Marián Moraga Galán
Tutor: Adrián Riesco Rodríguez

Índice

1. Sobre este manual

Este manual pretende ser una rápida introducción al lenguaje de programación Python. Está dirigido a aquellos sin conocimiento previo en programación y pensado para lingüistas, si bien puede servirle a cualquiera. Al acabar de revisar este manual, se recomienda continuar con el aprendizaje del uso de bibliotecas para procesamiento de lenguaje natural *spaCy* y *NLTK*, a través de los cursos oficiales indicados en los enlaces.

Como podrá verse a continuación, el manual contiene un primer apartado con un resumen de apuntes, para introducir al aprendiente a conceptos básicos y para proporcionar un listado de referencia de términos y funciones/métodos que irán apareciendo en este documento; después, se pasa al grueso del manual: ejemplos de código para que el aprendiente se familiarice gradualmente con el funcionamiento del lenguaje Python de manera inductiva; por último, se ofrece un apartado de práctica con ejercicios sobre funciones de usuario, pues se espera que, al finalizar el manual, el aprendiente sea capaz de crear funciones de usuario con diversos objetivos por sí mismo.

Podrá observarse que los ejemplos de código recogidos en el apartado de Casos de uso siguen el siguiente patrón:

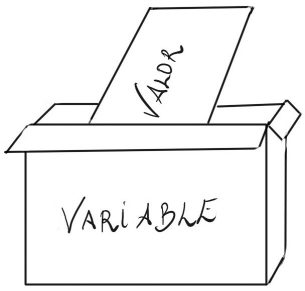
```
Líneas
de
código
```

```
Salida
```

Los casos de uso están pensados no solo para ser analizados, sino también para que se interactúe con ellos. A través del siguiente repositorio, se podrá acceder a cuadernos de Jupyter con todo el código presente en este manual.

2. Apuntes

variable = valor
print(*variable*)
.....
valor



nombredevariable = valor
nombre_de_variable = valor
NombreDeVariable = valor
variable2 = valor
VARIABLE = valor
_mi_variable = valor

**nombre de variable = valor*
**nombre-de-variable = valor*
**2variable = valor*

Tipos de datos:
texto: str
numérico: int, float
secuencia: list, tuple
mapeo: dict
booleano: bool

Operadores:
aritméticos: `+, -, +, /, //, %, **`
de comparación: `==, !=, >, <, >=, <=`
lógicos: and, or, not
de pertenencia: in, not in

Diccionarios:
variable = {clave:valor}

Funciones de usuario:
def *nombre_función(parámetro(s))*
  instrucción/es
  return *valor*

Funciones/métodos:
objeto.método(argumento)

	str	**list**	**tuple**	**dict**
len(a)	x	x	x	x
a.__contains__(“b”)	x	x	x	x
a.count(“b”)	x			
a[inicio:final]	x	x	x	
min(a)	x	x	x	
max(a)	x	x	x	
a.startswith(“b”)	x			
a.endswith(“b”)	x			
a.find(“b”)	x			
a.index(“b”)	x	x	x	
a.rfind(“b”)	x			
a.islower()	x			
a.isupper()	x			
a.lower()	x			
a.upper()	x			
a.capitalize()	x			
a.strip()	x			
a.lstrip()	x			
a.rstrip()	x			
a.replace(“b”, “c”)	x			
a.split(“a”)	x			
“b”.join(a)		x	x	
a.insert(nº, “c”)		x		
a.append(“b”)				
a.extend(b)			x	
a.count(“b”)		x	x	
a.sort()		x		
a.copy()		x	x	x
a.remove(“b”)		x		
a.pop(nº)		x		x
a.reverse()		x		
a.clear()		x		x
del a	x	x	x	x
a.get(“b”)				x
a.keys()				x
a.values()				x
a.items()				x
a.update({a:b})				x
a.popitem()				x

3. Casos de uso

3.1. Variables & str, int y float

```
string_var = "Esto es una cadena, siempre entre comillas simples o dobles"
integer_var = 1
floating_point_var = 1.0

print(string_var)
print(integer_var)
print(floating_point_var)
print("Las cadenas son secuencias de caracteres")
```

```
Esto es una cadena, siempre entre comillas simples o dobles
1
1.0
Las cadenas son secuencias de caracteres
```

```
#Esto es un comentario en una línea. La máquina lo ignorará.

"""
Esto es un comentario en bloque.
3 comillas dobles o simples antes y después.
La máquina lo ignorará.
"""

cadena_bloque = """Y esto es una string
en bloque, también con tres
comillas antes y después"""
print(cadena_bloque)

cadena2 = "Usa '\n' para hacer un salto de línea y '\t' para tabular"
print(cadena2)
```

```
Y esto es una string
en bloque, también con tres
comillas antes y después
Usa '
' para hacer un salto de línea y '      ' para tabular
```

```
x = str(7)
y = int("7")
z = float(7)

print(x)
print(y)
print(z)
```

```
7
7
7.0
```

```
a = "Saludos"
b = 7
c = 7.5

print(type(a))
print(type(b))
print(type(c))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
```

```
num1 = "7"
num2 = 7
num3 = 7.5

print(int(num1))
print(str(num2))
print(float(num2))
print(int(num3))
```

```
7
7
7.0
7
```

```
d, e, f = "Buenas", 7, 7.5

print(d)
print(e)
print(f)
```

```
Buenas
7
7.5
```

```
alpha = beta = "Hey"

print(alpha)
print(beta)
```

```
Hey
Hey
```

```
saludo = "Buenas"
print(saludo)

saludo = "Buenos-días"
print(saludo)
```

```
Buenas
Buenos días
```

```
saludo2 = "Buenas"
saludo2 = "Buenas-tardes"

print (saludo2)
print (saludo2)
```

```
Buenas tardes
Buenas tardes
```

3.2. Operadores aritméticos

```
suma = 10 + 3
resta = 10 - 3
multiplicación = 10 * 3
división = 10 / 3
división_entera = 10 // 3
resto_división = 10 % 3
exponente = 10 ** 3

print(suma)
print (resta)
print(multiplicación)
print(división)
print(división_entera)
print(resto_división)
print(exponente)
```

```
13
7
30
3.3333333333333335
3
1
1000
```

```
(7 - 2) * ((8 + 4) / (10 - 8))
```

```
30.0
```

```
horas = 24
minutos = horas * 60

print(minutos)
```

```
1440
```

```
ejercicios = 1
print(ejercicios)

ejercicios = ejercicios + 1
print (ejercicios)
```

```
1
2
```

3.3. Operadores de comparación y lógicos

```
"""
< menor
<= menor o igual
< mayor
<= mayor o igual
== igual
!= distinto
"""

print(7<8, 7!=7)
```

```
True False
```

```
"""
and [si se combinan dos expresiones
    y ambas se evalúan a True,
    devolverá True]

or [si se combinan dos expresiones
    y al menos una se evalúa a True,
    devolverá True]

not [se aplica a una sola expresión;
    si la expresión se evalúa a True,
    devuelve False y viceversa]
"""

print(7<8 and 7>8,  7<8 or 7>8, not (7>8))
```

```
False True True
```

```
print(7<8 or 7>8 and 7>8)       #"and" tiene preferencia sobre "or"
print((7<8 or 7>8) and 7>8)
```

```
True
False
```

3.4. Cadenas de texto

```
x = "Buenos"
y =  "días"

print(x, y)
```

```
Buenos días
```

```
nombre = "Python"

print(f"{nombre} es un lenguaje de programación")
```

```
Python es un lenguaje de programación
```

```
año = 1991

print(f"Guido van Rossum, su principal autor, publicó el código por primera
    vez en {año}")
```

```
Guido van Rossum, su principal autor, publicó el código por primera vez en
    1991
```

```
"limpia" + "para" + "brisas"              #*"limpia" + 4
```

```
'limpiaparabrisas'
```

```
a = "Buenas-"
b =  "tardes"

print(a + b)
```

```
Buenas tardes
```

```
"Buenas" * 4
```

```
'BuenasBuenasBuenasBuenas'
```

```
saludo = "Buenas"
saludo= saludo * 2

print(saludo)
```

```
BuenasBuenas
```

```
cadena = "Salamanca"

print(len(cadena))
```

```
9
```

```
print(cadena[0], cadena[3], cadena[-2])
```

```
S a c
```

```
print("s" in cadena)
print("S" in cadena)
```

```
False
True
```

```
cadena.__contains__("man")
```

```
True
```

```
cadena2 = "Era más de medianoche, antiguas historias cuentan"

cadena2.count("a")
```

```
6
```

```
print("cuando" not in cadena2)
```

```
True
```

```
#inicio:final [final no incluido]

print(cadena2[4:10])
```

```
más de
```

```
print(cadena2[:21])
```

```
Era más de medianoche
```

```
print(cadena2[23:])
```

```
antiguas historias cuentan
```

```
cadena3 = "Espronceda"

print(cadena3[-4:])
```

```
ceda
```

```
print(cadena3[-8:-4])
```

```
pron
```

```
print(min(cadena3))
print(max(cadena3))
```

```
E
s
```

```
cadena4 = "cuando en sueño y en silencio lóbrego envuelta la tierra"

cadena4.startswith("cuando")
```

```
True
```

```
cadena4.endswith("cuentan")
```

```
False
```

```
print(cadena4.find("sueño"))
print(cadena4.find("medianoche"))
```

```
10
-1
```

```
cadena4.index("z")   #da error si la substring no existe en la string
```

```
ValueError                                Traceback (most recent call last)
Cell In[21], line 1
----> 1 cadena4.index("z")

ValueError: substring not found
```

```
cadena4.rfind("a")    #última aparición
```

```
55
```

```
print(cadena4.islower(), cadena4.isupper())
```

```
True False
```

```
cadena4.upper()

cadena4
```

```
'cuando en sueño y en silencio lóbrego envuelta la tierra'
```

```
print(cadena4.upper())
```

```
CUANDO EN SUEÑO Y EN SILENCIO LÓBREGO ENVUELTA LA TIERRA
```

```
cadena2 = cadena2.upper()
print(cadena2)
```

```
ERA MÁS DE MEDIANOCHE, ANTIGUAS HISTORIAS CUENTAN
```

```
cadena2.lower()
```

```
'era más de medianoche, antiguas historias cuentan'
```

```
cadena5 = " Estudiante de Salamanca "
cadena5.strip()
```

```
cadena5 = " Estudiante de Salamanca "
cadena5.strip()
```

```
cadena5
```

```
'·Estudiante·de·Salamanca·'
```

```
print(cadena5)
```

```
 Estudiante de Salamanca
```

```
cadena5.lstrip()
```

```
'Estudiante·de·Salamanca·'
```

```
cadena5.rstrip()
```

```
'·Estudiante·de·Salamanca'
```

```
cadena6 = "los·vivos·muertos·parecen,·los·muertos·la·tumba·dejan"
cadena6.capitalize()
```

```
'Los·vivos·muertos·parecen,·los·muertos·la·tumba·dejan'
```

```
cadena6 = cadena6.replace("lo", "Lo")
cadena6 = cadena6.replace(",·", ",\n")

print(cadena6)
```

```
Los vivos muertos parecen,
Los muertos la tumba dejan
```

```
cadena6 = cadena6.replace("Los muertos la tumba dejan", "")

cadena6
```

```
'Los vivos muertos parecen,\n'
```

```
cadena6.split()
```

```
['Los', 'vivos', 'muertos', 'parecen,']
```

```
cadena7 = "Era más de medianoche, antiguas historias cuentan, cuando en sueño
    y en silencio lóbrego envuelta la tierra, los vivos muertos parecen, los
    muertos la tumba dejan"

cadena7.split(",")
```

```
['Era más de medianoche',
 ' antiguas historias cuentan',
 ' cuando en sueño y en silencio lóbrego envuelta la tierra',
 ' los vivos muertos parecen',
 ' los muertos la tumba dejan']
```

```
cadena7b = cadena7.split(", ")

"_".join(cadena7b)
```

```
'Era más de medianoche_antiguas historias cuentan_cuando en sueño y en
    silencio lóbrego envuelta la tierra_los vivos muertos parecen_los muertos
    la tumba dejan'
```

```
cadena7.replace(",", "")
.....

'Era-más-de-medianoche-antiguas-historias-cuentan-cuando-en-sueño-y-en-
    silencio-lóbrego-envuelta-la-tierra-los-vivos-muertos-parecen-los-muertos-
    la-tumba-dejan'
```

3.5. Listas & tuplas

```
lista = [1,2,3,4]                #colección ordenada y mutable (podemos
    modificar valores)
Lista = [1, 2, 3, 4]
LISTA = ["uno", "dos", "tres", "cuatro"]
mi_lista = [True, False, True]
OtraLista = [1, "dos", True, 1, "dos"]                    #se pueden duplicar
    valores

print(type(lista),type(Lista),type(LISTA),
      type(mi_lista),type(OtraLista))
```

```
<class 'list'> <class 'list'> <class 'list'> <class 'list'> <class 'list'>
```

```
tupla = (1,2,3,4)                #colección ordenada e inmutable (no podemos
    modificar valores)
Tupla = (1, 2, 3, 4)
TUPLA = ("uno", "dos", "tres", "cuatro")
mi_tupla = (True, False, True)
OtraTupla = (1, "dos", True, 1, "dos")                    #se pueden duplicar
    valores

print(type(tupla), type(Tupla), type(TUPLA),
      type(mi_tupla), type(OtraTupla))
```

```
<class 'tuple'> <class 'tuple'> <class 'tuple'> <class 'tuple'> <class 'tuple'
    >
```

```
milista = [5,6,7,8]
mitupla = (5,6,7,8)

print(list(mitupla))
print(tuple(milista))
```

```
[5, 6, 7, 8]
(5, 6, 7, 8)
```

```
Lfrutas = list(("plátano", "manzana", "uva"))

print(Lfrutas)
print(type(Lfrutas))
```

```
['plátano', 'manzana', 'uva']
<class 'list'>
```

```
Tfrutas = ("piña",)
print(type(Tfrutas))

Tfrutas = ("piña")
print(type(Tfrutas))
```

```
<class 'tuple'>
<class 'str'>
```

```
"-&-".join(Lfrutas)
```

```
'plátano-&-manzana-&-uva'
```

```
Lfrutas * 4
```

```
['plátano',
 'manzana',
 'uva',
 'plátano',
 'manzana',
 'uva',
 'plátano',
 'manzana',
 'uva',
 'plátano',
 'manzana',
 'uva']
```

```
Lciudades = ["Madrid", "Bilbao", "Granada"]
Tciudades = ("Madrid", "Bilbao", "Granada")

print(min(Lciudades))
print(max(Tciudades))
```

```
Bilbao
Madrid
```

```
print(Lciudades[0])
print(Tciudades[1])
```

```
Madrid
Bilbao
```

```
Lciudades[0] = "Valencia"

print(Lciudades)
```

```
['Valencia', 'Bilbao', 'Granada']
```

```
Lciudades2 = ["Bruselas", "Londres", "Milán"]
Lciudades = Lciudades2
Lciudades2[1] = "Viena"

print(Lciudades2)
print(Lciudades)
```

```
['Bruselas', 'Viena', 'Milán']
['Bruselas', 'Viena', 'Milán']
```

```
Lnúmeros = ["Uno", "Dos", "Tres", "Cuatro"]
Tnúmeros = ("Uno", "Dos", "Tres", "Cuatro")

print(Lnúmeros[-1])
print(Tnúmeros[-2])
```

```
Cuatro
Tres
```

```
Lanimales = ["caballo", "pato", "oveja", "cabra", "cerdo", "vaca"]
Tanimales = ("caballo", "pato", "oveja", "cabra", "cerdo", "vaca")

Lanimales.index("caballo")
```

```
0
```

```
Tanimales.index("oveja")
```

```
2
```

```
print(Lanimales[1:4])
print(Tanimales[0:3])
```

```
['pato', 'oveja', 'cabra']
('caballo', 'pato', 'oveja')
```

```
print(Lanimales[:3])
print(Tanimales[:2])
```

```
['caballo', 'pato', 'oveja']
('caballo', 'pato')
```

```
print(Lanimales[2:])
print(Tanimales[3:])
```

```
['oveja', 'cabra', 'cerdo', 'vaca']
('cabra', 'cerdo', 'vaca')
```

```
print(Lanimales[-3:-1])
print(Tanimales[-5:-2])
```

```
['cabra', 'cerdo']
('pato', 'oveja', 'cabra')
```

```
print(min(Lanimales), max(Lanimales))
print(min(Tanimales), max(Tanimales))
```

```
caballo vaca
caballo vaca
```

```
print("cabra" in Lanimales)
print("perro" in Tanimales)
```

```
True
False
```

```
Lanimales[0] = "perro"

print(Lanimales)
print("perro" in Lanimales)
```

```
['perro', 'pato', 'oveja', 'cabra', 'cerdo', 'vaca']
True
```

```
Lanimales[1:3] = ["gallina"]

print(Lanimales)
```

```
['perro', 'gallina', 'cabra', 'cerdo', 'vaca']
```

```
Lanimales.insert(3, "asno")

print(Lanimales)
```

```
['perro', 'gallina', 'cabra', 'asno', 'cerdo', 'vaca']
```

```
Tanimales = list(Tanimales)
Tanimales[0] = "perro"
Tanimales[1:3] = ["gallina"]
Tanimales.insert(3, "asno")
Tanimales = tuple(Tanimales)

print(Tanimales)
print("perro" in Tanimales)
```

```
('perro', 'gallina', 'cabra', 'asno', 'cerdo', 'vaca')
True
```

```
Lanimales.append("gato")
print(Lanimales)

animal = ("gato",)
Tanimales += animal              #igual que: Tanimales = Tanimales + animal
print(Tanimales)
```

```
['perro', 'gallina', 'cabra', 'asno', 'cerdo', 'vaca', 'gato']
('perro', 'gallina', 'cabra', 'asno', 'cerdo', 'vaca', 'gato')
```

```
animales = ["canario", "oveja"]
Lanimales = Lanimales + animales
print(Lanimales)
```

```
['perro', 'gallina', 'cabra', 'asno', 'cerdo', 'vaca', 'gato', 'canario', '
    oveja']
```

```
animales2 = ["hámster", "tortuga"]
Lanimales.extend(animales2)
print(Lanimales)

animales2 = tuple(animales2)
Tanimales += animales2
print(Tanimales)
```

```
['perro', 'gallina', 'cabra', 'asno', 'cerdo', 'vaca', 'gato', 'canario', '
    oveja', 'hámster', 'tortuga']
('perro', 'gallina', 'cabra', 'asno', 'cerdo', 'vaca', 'gato', 'hámster', '
    tortuga')
```

```
Lanimales.extend(Tanimales)

print(Lanimales)
```

```
['perro', 'gallina', 'cabra', 'asno', 'cerdo', 'vaca', 'gato', 'canario', '
    oveja', 'hámster', 'tortuga', 'perro', 'gallina', 'cabra', 'asno', 'cerdo'
    , 'vaca', 'gato', 'hámster', 'tortuga']
```

```
Lanimales.count("perro")
```

```
2
```

```
Tanimales.count("perro")
```

```
1
```

```
print(len(Lanimales))
print(len(Tanimales))
```

```
20
9
```

```
print(Lanimales)

Lanimales.sort()

print(Lanimales)
```

```
['perro', 'gallina', 'cabra', 'asno', 'cerdo', 'vaca', 'gato', 'canario', '
    oveja', 'hámster', 'tortuga', 'perro', 'gallina', 'cabra', 'asno', 'cerdo'
    , 'vaca', 'gato', 'hámster', 'tortuga']
['asno', 'asno', 'cabra', 'cabra', 'canario', 'cerdo', 'cerdo', 'gallina', '
    gallina', 'gato', 'gato', 'hámster', 'hámster', 'oveja', 'perro', 'perro',
     'tortuga', 'tortuga', 'vaca', 'vaca']
```

```
Lanimales.sort(reverse = True)

print(Lanimales)
```

```
['vaca', 'vaca', 'tortuga', 'tortuga', 'perro', 'perro', 'oveja', 'hámster', '
    hámster', 'gato', 'gato', 'gallina', 'gallina', 'cerdo', 'cerdo', 'canario
    ', 'cabra', 'cabra', 'asno', 'asno']
```

```
lista_desordenada = ["A", "c", "b", "C", "a", "B"]   #cuidado con las mayú
    sculas
lista_ordenada = lista_desordenada.copy()
lista_ordenada.sort()

print(lista_ordenada)
```

```
['A', 'B', 'C', 'a', 'b', 'c']
```

```
lista_compra = ["harina", "leche", "azúcar", "huevos", "leche", "canela", "
    vainilla"]
lista_compra.remove("leche")

print(lista_compra)
```

```
['harina', 'azúcar', 'huevos', 'leche', 'canela', 'vainilla']
```

```
lista_compra.pop()
```

```
'vainilla'
```

```
print(lista_compra)
```

```
['harina', 'azúcar', 'huevos', 'leche', 'canela']
```

```
lista_compra.pop(0)
```

```
'harina'
```

```
print(lista_compra)
```

```
['azúcar', 'huevos', 'leche', 'canela']
```

```
lista_compra.reverse()

print(lista_compra)
```

```
['canela', 'leche', 'huevos', 'azúcar']
```

```
lista_compra2 = lista_compra.copy()
lista_compra.clear()

print(lista_compra)
print(lista_compra2)
```

```
[]
['canela', 'leche', 'huevos', 'azúcar']
```

```
lista_inutil = [1,2,3]
tupla_inutil = (1,2,3)

del lista_inutil
del tupla_inutil

print(lista_inutil, tupla_inutil)              #Dará error porque estas lista
    y tupla ya no existen
```

```
NameError                                 Traceback (most recent call last)
Cell In[42], line 7
      4 del lista_inutil
      5 del tupla_inutil
----> 7 print(lista_inutil, tupla_inutil)

NameError: name 'lista_inutil' is not defined
```

```
tupla_despensa = ("brócoli", "trigo", "lenteja")
(verduras, cereales, legumbres) = tupla_despensa

print(verduras)
print(cereales)
print(legumbres)
```

```
brócoli
trigo
lenteja
```

```
tupla_despensa += ("garbanzo",)
(verduras, cereales, *legumbres) = tupla_despensa

print(verduras)
print(cereales)
print(legumbres)
```

```
brócoli
trigo
['lenteja', 'garbanzo']
```

```
Tautores = ("Shakespeare", "Cervantes", "Lope-de-Vega", "Dante", "Moliére")
(ingleses, *españoles, italianos, franceses) = Tautores

print(ingleses)
print(españoles)
print(italianos)
print(franceses)
```

```
Shakespeare
['Cervantes', 'Lope-de-Vega']
Dante
Moliére
```

```
lista_mercado = [["ternera", "pollo"], ["bacalao", "salmón"]]
lista_mercado[0].append("cordero")
lista_mercado[1].append("atún")

print(lista_mercado)
```

```
[['ternera', 'pollo', 'cordero'], ['bacalao', 'salmón', 'atún']]
```

```
(carnicería, pescadería) = lista_mercado

print(carnicería)
print(pescadería)
```

```
['ternera', 'pollo', 'cordero']
['bacalao', 'salmón', 'atún']
```

3.6. Diccionarios

```
diccionario = {"Autor": "Douglas Adams", "Novela": "Guía del autoestopista gal
    áctico", "Año": 1979}

print(diccionario["Novela"])
print(type(diccionario))
```

```
Guía del autoestopista galáctico
<class 'dict'>
```

```
diccionariobis = dict(Autor = "Douglas Adams", Novela = "Guía del
    autoestopista galáctico", Año = 1979)

print(diccionariobis)
print(type(diccionariobis))
```

```
{'Autor': 'Douglas Adams', 'Novela': 'Guía del autoestopista galáctico', 'Año'
    : 1979}
<class 'dict'>
```

```
dicc = dict()

dicc["Autor"] = "Douglas Adams"
dicc["Novela"] = "Guía del autoestopista galáctico"
dicc["Año"] = 1979

dicc
```

```
{'Autor': 'Douglas Adams',
 'Novela': 'Guía del autoestopista galáctico',
 'Año': 1979}
```

```
diccionario2 = {
    "Autor": "Douglas Adams",
    "Novela": "Guía del autoestopista galáctico",
    "Año": 1979,
    "Secuela": "El restaurante del fin del mundo",
    "Año": 1980
}

diccionario2["Año"]
```

```
1980
```

```
libro = diccionario2.get("Secuela")

libro
```

```
'El restaurante del fin del mundo'
```

```
len(diccionario2)
```

```
4
```

```
diccionario2.keys()
```

```
dict_keys(['Autor', 'Novela', 'Año', 'Secuela'])
```

```
diccionario2.values()
```

```
dict_values(['Douglas Adams', 'Guía del autoestopista galáctico', 1980, 'El
    restaurante del fin del mundo'])
```

```
diccionario2.items()
```

```
dict_items([('Autor', 'Douglas-Adams'), ('Novela', 'Guía-del-autoestopista-gal
    áctico'), ('Año', 1980), ('Secuela', 'El-restaurante-del-fin-del-mundo')])
```

```
print("Autor" in diccionario2)
```

```
True
```

```
diccionario3 = {
    "nombre": "Fulanito",
    "edad": 27,
    "ocupación": "profesor"
}

diccionario3["ocupación"] = "lingúista-computacional"
diccionario3["nacionalidad"] = "argentino"

print(diccionario3)
```

```
{'nombre': 'Fulanito', 'edad': 27, 'ocupación': 'lingúista-computacional', '
    nacionalidad': 'argentino'}
```

```
diccionario3.update({"edad": 30})

diccionario3
```

```
{'nombre': 'Fulanito',
 'edad': 30,
 'ocupación': 'lingúista-computacional',
 'nacionalidad': 'argentino'}
```

```
diccionario3.pop("edad")

diccionario3
```

```
{'nombre': 'Fulanito',
 'ocupación': 'lingúista-computacional',
 'nacionalidad': 'argentino'}
```

```
diccionario3.popitem()

diccionario3
```

```
{'nombre': 'Fulanito', 'ocupación': 'lingúista-computacional'}
```

```
del diccionario3["ocupación"]

diccionario3
```

```
{'nombre': 'Fulanito'}
```

```
diccionario3bis = diccionario3.copy()
diccionario3bis2 = dict(diccionario3)
diccionario3.clear()

print(diccionario3)
print(diccionario3bis)
print(diccionario3bis2)
```

```
{}
{'nombre': 'Fulanito'}
{'nombre': 'Fulanito'}
```

```
del diccionario3

diccionario3  #dará error porque el diccionario ya no existe
```

```
NameError                                 Traceback (most recent call last)
Cell In[16], line 3
      1 del diccionario3
----> 3 diccionario3

NameError: name 'diccionario3' is not defined
```

```
estudiantes = {
    "estudiante1": {
        "nombre": "Menganita",
        "facultad": "Derecho",
        "año": 2
    },
    "estudiante2": {
        "nombre": "Menganito",
        "facultad": "Informática",
        "año": 4
    }
}

estudiantes
```

```
{'estudiante1': {'nombre': 'Menganita', 'facultad': 'Derecho', 'año': 2},
 'estudiante2': {'nombre': 'Menganito', 'facultad': 'Informática', 'año': 4}}
```

```
curso1 = {
    "contenido": "Lingúística computacional",
    "horas": 300
}
curso2 = {
    "contenido": "Procesamiento del lenguaje natural",
    "horas": 20
}
curso3 = {
    "contenido": "Introducción al diseño de interfaces conversacionales",
    "horas": 4
}

formación_complementaria = {
    "curso1": curso1,
    "curso2": curso2,
    "curso3": curso3
}

formación_complementaria
```

```
{'curso1': {'contenido': 'Lingúística computacional', 'horas': 300},
 'curso2': {'contenido': 'Procesamiento del lenguaje natural', 'horas': 20},
 'curso3': {'contenido': 'Introducción al diseño de interfaces
     conversacionales',
  'horas': 4}}
```

```
print(formación_complementaria["curso1"]["contenido"])
```

```
Lingúística computacional
```

3.7. Condicionales

```
a = 7
b = 17

if a < b:                # *if a < b
    print(True)          # print(True)  [Recuerda tabular]
```

```
True
```

```
if a > b:
    print(True)
else:
    print(False)
```

```
False
```

```
print("Es mayor") if a > b else print ("No es mayor")
```

```
No es mayor
```

```
if not a > b:
    print("'a' no es mayor de 'b'")
```

```
'a' no es mayor de 'b'
```

```
if a > b and a == b:
    print("Ambas se cumplen")
else:
    print("Al menos una no se cumple")
```

```
Al menos una no se cumple
```

```
c = 70

if a > b or c > b:
    print("Al menos una condición se evalúa a True")
```

```
Al menos una condición se evalúa a True
```

```
y = z = 17

if y < z:
    print("'y' es menor que 'z'")
elif y > z:
    print("'y' es mayor que 'z'")
else:
    print("'y' es igual que 'z'")
```

```
'y' es igual que 'z'
```

```
microrrelato = "Cuando despertó, el dinosaurio todavía estaba allí."

if microrrelato.startswith("Cuando") == True:
    print("El microrrelato empieza con 'Cuando'")
elif microrrelato.lower() == microrrelato:
    print("El microrrelato está todo en minúsculas"),
elif microrrelato.capitalize() == microrrelato:
    print("El microrrelato empieza con una mayúscula"),
elif microrrelato.endswith(".") == True:
    print("El microrrelato termina con un punto")

print("Fin del análisis")
```

```
El microrrelato empieza con 'Cuando'
Fin del análisis
```

```
if microrrelato.startswith("cuando"):
    print("El microrrelato empieza con 'cuando'")
elif microrrelato.lower() == microrrelato:
    print("El microrrelato está todo en minúsculas"),
elif microrrelato.capitalize() == microrrelato:
    print("El microrrelato empieza con una mayúscula"),
elif microrrelato.endswith("."):
    print("El microrrelato termina con un punto")

print("Fin del análisis")
```

```
El microrrelato empieza con una mayúscula
Fin del análisis
```

```
if micrrrelato.startswith("cuando"):
    print("El microrrelato empieza con 'cuando'")
elif microrrelato.lower() == microrrelato:
    print("El microrrelato está todo en minúsculas"),
elif microrrelato.capitalize() == microrrelato:
    print("El microrrelato empieza con una mayúscula"),
    if microrrelato.endswith("."):
        print("El microrrelato termina además con un punto")

print("Fin del análisis")
```

```
El microrrelato empieza con una mayúscula
El microrrelato termina además con un punto
Fin del análisis
```

```
análisis = f"'{microrrelato}' contiene la palabra 'dinosaurio'" if
    microrrelato.__contains__("dinosaurio") else f"{microrrelato} no contiene
    la palabra 'dinosaurio'"
print(análisis)
print("Fin del análisis")
```

```
'Cuando despertó, el dinosaurio todavía estaba allí.' contiene la palabra '
    dinosaurio'
Fin del análisis
```

3.8. Bucle for y while

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

for número in lista:
    print(f"Esta es la vuelta número {número}")
```

```
Esta es la vuelta número 1
Esta es la vuelta número 2
Esta es la vuelta número 3
Esta es la vuelta número 4
```

```
for i in "Chomsky":       #iteración, repetición, bucle
    print(i)
```

```
C
h
o
m
s
k
y
```

```
for loquesea in ("a", "b", "c", "d"):
    print(f"Ahora 'loquesea' es igual a '{loquesea}'")
```

```
Ahora 'loquesea' es igual a 'a'
Ahora 'loquesea' es igual a 'b'
Ahora 'loquesea' es igual a 'c'
Ahora 'loquesea' es igual a 'd'
```

```
for x in range(5):
    print(x)
else:
    print("El bucle ha llegado a su fin")
```

```
0
1
2
3
4
El bucle ha llegado a su fin
```

```
for x in range(10):
    if x == 5: break
    print(x)
else:
    print("El bucle ha llegado a su fin")
```

```
0
1
2
3
4
```

```
nombres = ["Juan", "Carmen", "Luis", "Isabel"]
apellidos = ["Martínez", "López", "García", "Rodríguez"]

for nombre in nombres:
    for apellido in apellidos:
        print(nombre, apellido)
```

```
Juan Martínez
Juan López
Juan García
Juan Rodríguez
Carmen Martínez
Carmen López
Carmen García
Carmen Rodríguez
Luis Martínez
Luis López
Luis García
Luis Rodríguez
Isabel Martínez
```

```
Isabel López
Isabel García
Isabel Rodríguez
```

```
howmanyA = 0

for i in "La vaca es un animal todo forrado de cuero":
    if i == "a":
        howmanyA += 1

print(howmanyA)
```

```
6
```

```
oración ="Tiene las patas tan largas que le llegan hasta el suelo"

for elemento, posición in enumerate(oración.split()):
    print(f"{elemento}, {posición}")
```

```
0, Tiene
1, las
2, patas
3, tan
4, largas
5, que
6, le
7, llegan
8, hasta
9, el
10, suelo
```

```
#listas intensionales, añadir eltos a lista con for

palabra = [letra for letra in "vaca"]

print(palabra)
```

```
['v', 'a', 'c', 'a']
```

```
impares = [i for i in range(10) if i%2 != 0]

print(impares)
```

```
[1, 3, 5, 7, 9]
```

```
buitre_predilecta = "Más arriba, a la izquierda, tengo algo muy dulce para ti.
    (Ella se obstinó en el hígado y no supo el corazón de Prometeo.)"
lista_oración = buitre_predilecta.split()

contieneS = [palabra for palabra in lista_oración if palabra.__contains__("s")
    ]

print(contieneS)
```

```
['Más', 'se', 'obstinó', 'supo']
```

```
min_contieneS = [min(palabra) for palabra in lista_oración if palabra.
    __contains__("s")]

print(min_contieneS)
```

```
['M', 'e', 'b', 'o']
```

```
vuelta = 0

while vuelta < 5:
    print(vuelta)
    vuelta += 1      #sin esto, el bucle sería infinito
```

```
0
1
2
3
4
```

```
vuelta = 0

while vuelta < 5:
    print(vuelta)
    if vuelta == 2:
        break
    vuelta += 1
```

```
0
1
2
```

```
vuelta = 0

while vuelta < 5:
    vuelta += 1
    if vuelta == 2:
        continue
    print(vuelta)
```

```
1
3
4
5
```

```
vuelta = 0
palabras = ["Recuerda", "añadir", "siempre", "un", "contador", "de", "
    iteraciones"]

while vuelta < len(palabras):
    print(palabras[vuelta], end="-")
    vuelta += 1
```

```
Recuerda-añadir-siempre-un-contador-de-iteraciones-
```

3.9. Funciones de usuario

```
def empecemos():
    return " Al ataque!"

mensaje = empecemos()
print(mensaje)
```

```
 Al  ataque!
```

```
def saludo(persona):
    return f"Buenas, {persona}"

saludo("Don Quijote")
saludo("Sancho Panza")
```

```
'Buenas, Sancho Panza'
```

```
def presentación(nombre, profesión):
    return f"Me llamo {nombre} y soy {profesión}"

presentación("Don Quijote", "caballero")
presentación("Sancho Panza", "escudero")
```

```
'Me llamo Sancho Panza y soy escudero'
```

```
def hermanos(número, *mayor):
    return f"Tengo {número} hermanos y {mayor[0]} es el mayor"

hermanos(3, "Carlos", "Juan", "Marcos")
```

```
'Tengo 3 hermanos y Carlos es el mayor'
```

```
def presentación2(nombre, profesión="desconocida"):
    return f"Se llama {nombre} y su profesión es {profesión}"

print(presentación2("Claudia", "pintora"))   #por qué devuelve None?
print(presentación2("Claudio"))
```

```
Se llama Claudia y su profesión es pintora
Se llama Claudio y su profesión es desconocida
```

```
variable = "global"

def atención():
    variable = "local"
    return "Ahora se imprime el valor de la variable " + variable

print(atención())
print("Ahora se imprime el valor de la variable " + variable)
```

```
Ahora se imprime el valor de la variable local
Ahora se imprime el valor de la variable global
```

```
def palabra_larga(palabra1, palabra2):
    if len(palabra1) > len(palabra2):
        print(f"La palabra más larga es '{palabra1}'")
    elif len(palabra1) < len(palabra2):
        print(f"La palabra más larga es '{palabra2}'")
    else:
        return "Las dos palabras tienen el mismo número de letras"

palabra_larga("invierno", "verano")
palabra_larga("anís", "canela")
palabra_larga("abeja", "oveja")
```

```
La palabra más larga es 'invierno'
La palabra más larga es 'canela'
'Las dos palabras tienen el mismo número de letras'
```

```
def palabras_largas(lista_palabras):
    lista_largas = []
    for palabra in lista_palabras:
        if len(palabra) > 5:
            lista_largas.append(palabra)
    return lista_largas

milista = ["En", "un", "lugar", "de", "la", "Mancha", "de", "cuyo", "nombre",
    "no", "quiero", "acordarme"]

palabras_largas(milista)
```

```
['Mancha', 'nombre', 'quiero', 'acordarme']
```

4. Práctica

a) Escribe una función que devuelva True si hay un número mayor que 10 en una lista de números y False en caso contrario. Comprueba que funciona con todas las listas ofrecidas. ¿Se te ocurre una forma de conseguirlo sin usar condiciones?

```
lista1a = [1,2,3,4,5]
lista2a = [1,2,3,4,20]
lista3a = [1,2,10,4,5]
lista4a = [100,2,3,4,5]

#def
.....
```

---

b) Escribe una función que diga cuántas veces aparece el número 5 en una lista de números. Comprueba que funciona con todas las listas ofrecidas.

```
lista1b = [5,0,3,8,5,4]
lista2b = [5,5,5,5,5]
lista3b = [0,1,2,3,4]

#def
.....
```

---

c) Escribe una función que devuelva True si una cadena contiene la palabra "y". Comprueba que funciona con todas las cadenas ofrecidas.

```
cadena1c = "Hola y bienvenidos"
cadena2c = "Y qué tal"
cadena3c = "Muy bien"

#def
.....
```

---

d) Escribe una función que, dadas una cadena y una lista de palabras, devuelva True si la cadena contiene alguna de las palabras de la lista y False en caso contrario. Comprueba que funciona con los pares de cadena y lista ofrecidos.

```
cadena1d = "Buenas, qué tal"
cadena2d = "Hola, cómo vas"
lista1d = ["buenas", "hola", "tal"]
lista2d = ["adiós", "saludo"]

#def
.....
```

Puedes comparar tus respuestas con las soluciones propuestas en el repositorio de Github.