1 Python Básico - Resumen - Estructuras de datos

1.1 Listas

Las listas permiten almacenar objetos mediante un orden definido y con posibilidad de duplicados.

Las listas son estructuras de datos mutables, lo que significa que podemos añadir, eliminar o modificar sus elementos.

In [339]:
```
# Lista con datos del mismo tipo
languages = ['Python', 'Ruby', 'Javascript']
fibonacci = [0, 1, 1, 2, 3, 5, 8, 13]
```
executed in 12ms, finished 02:42:36 2023-01-10

In [340]:
```
#Listas con datos heterogéneos
data = ['Tenerife', {'cielo': 'limpio', 'temp': 24}, 3718, (28.2933947, -16.5226597)]
```
executed in 13ms, finished 02:42:36 2023-01-10

1.1.1 Funciones incorpodadas de python aplicables a listas

1.1.1.1 Crear una lista o transformar otro tipo de objeto a lista

In [341]:
```
#Crear una lista vacía
lista = list()
lista
```
executed in 14ms, finished 02:42:36 2023-01-10

Out[341]:
```
[]
```

In [342]:
```
lista = []
```

In [343]:
```
# Convertir cadena en lista
cadena = 'cadena de caracteres'
caracteres = list(cadena)
print(caracteres)
```
executed in 11ms, finished 02:42:36 2023-01-10

```
['c', 'a', 'd', 'e', 'n', 'a', ' ', 'd', 'e', ' ', 'c', 'a', 'r', 'a', 'c', 't', 'e', 'r', 'e', 's']
```

1.1.1.2 Cantidad de elementos de una lsita

In [344]:
```
len(languages)
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[344]:
```
3
```

1.1.1.3 Borrar elementos de una lista

In [345]:
```
del languages[1]
languages
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[345]:
```
['Python', 'Javascript']
```

1.1.1.4 Verificar si un elemento está en la lista

In [346]:
```
8 in fibonacci
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[346]:
```
True
```

1.1.1.5 Funciones matemáticas

In [347]:
```
max(fibonacci)
```
executed in 14ms, finished 02:42:36 2023-01-10

Out[347]:
```
13
```

In [348]:
```
max(languages)
```

Out[348]:
```
'Python'
```

In [349]:
```
min(fibonacci)
```
executed in 14ms, finished 02:42:36 2023-01-10

Out[349]:
```
0
```

In [350]:
```
1 sum(fibonacci)
```
executed in 11ms, finished 02:42:36 2023-01-10

Out[350]: 33

1.1.2 Operaciones con Listas

1.1.2.1 Obtener un elemento

In [351]:
```
1 shopping = ['Agua', 'Huevos', 'Aceite', 'Sal', 'Limón']
```
executed in 14ms, finished 02:42:36 2023-01-10

In [352]:
```
1 shopping [1]
```
executed in 14ms, finished 02:42:36 2023-01-10

Out[352]: 'Huevos'

In [353]:
```
1 shopping [-1]
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[353]: 'Limón'

1.1.2.2 Slicing

In [354]:
```
1 shopping[1:4]
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[354]: ['Huevos', 'Aceite', 'Sal']

In [355]:
```
1 shopping[-1: -4: -1]
```
executed in 12ms, finished 02:42:36 2023-01-10

Out[355]: ['Limón', 'Sal', 'Aceite']

In [356]:
```
1 #Invertir una lista
2 shopping[::-1]
```
executed in 12ms, finished 02:42:36 2023-01-10

Out[356]: ['Limón', 'Sal', 'Aceite', 'Huevos', 'Agua']

1.1.2.3 Cambiar un elemento

Modifica la lista original

In [357]:
```
1 shopping[3] = 'Pimienta'
2 shopping
```
executed in 10ms, finished 02:42:36 2023-01-10

Out[357]: ['Agua', 'Huevos', 'Aceite', 'Pimienta', 'Limón']

1.1.2.4 Cambiar una porción de una lista

Modifica la lista original

In [358]:
```
1 shopping[1:3] = ['Leche', 'Pan']
2 shopping
```
executed in 12ms, finished 02:42:36 2023-01-10

Out[358]: ['Agua', 'Leche', 'Pan', 'Pimienta', 'Limón']

1.1.2.5 Repetir una lista o parte de ella

In [359]:
```
1 shopping[:3] * 2
```
executed in 10ms, finished 02:42:36 2023-01-10

Out[359]: ['Agua', 'Leche', 'Pan', 'Agua', 'Leche', 'Pan']

1.1.2.6 Concatenar listas

In [360]:
```
1 lista1 = ['Elemto1', 'Elemento2', 'Elemento3']
2 lista2 = ['Cosa1', 'Cosa2', 'Cosa3']
3 lista1 + lista2
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[360]: ['Elemto1', 'Elemento2', 'Elemento3', 'Cosa1', 'Cosa2', 'Cosa3']

1.1.3 Métodos de las listas

Nota, la mayoria de los métodos modifican la lista original

1.1.3.1 Añadir un elemento al final

In [361]:
```
1 shopping.append('Azúcar')
2 shopping
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[361]: ['Agua', 'Leche', 'Pan', 'Pimienta', 'Limón', 'Azúcar']

1.1.3.2 Insertar un elemento según índice

In [362]:
```
1 shopping.insert(2, 'Arroz')
2 shopping
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[362]: ['Agua', 'Leche', 'Arroz', 'Pan', 'Pimienta', 'Limón', 'Azúcar']

1.1.3.3 Concatenar una lista con otra, modificando la primera

In [363]:
```
1 shopping2 = ['Naranja', 'Manzana']
```
executed in 13ms, finished 02:42:36 2023-01-10

In [364]:
```
1 shopping.extend(shopping2)
2 print(shopping)
```
executed in 13ms, finished 02:42:36 2023-01-10

```
['Agua', 'Leche', 'Arroz', 'Pan', 'Pimienta', 'Limón', 'Azúcar', 'Naranja', 'Manzana']
```

1.1.3.4 Borrar y extraer elemento según índice

In [365]:
```
1 shopping.pop(2)
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[365]: 'Arroz'

In [366]:
```
1 shopping
```
executed in 12ms, finished 02:42:36 2023-01-10

Out[366]: ['Agua', 'Leche', 'Pan', 'Pimienta', 'Limón', 'Azúcar', 'Naranja', 'Manzana']

1.1.3.5 Borrar elemento por su contenido

In [367]:
```
1 shopping.remove('Naranja')
2 shopping
```
executed in 12ms, finished 02:42:36 2023-01-10

Out[367]: ['Agua', 'Leche', 'Pan', 'Pimienta', 'Limón', 'Azúcar', 'Manzana']

1.1.3.6 Índice de un elemento

In [368]:
```
1 shopping.index('Pan')
```
executed in 18ms, finished 02:42:36 2023-01-10

Out[368]: 2

1.1.4 Cadenas y Listas

In [369]:
```
1 cadena = 'Esta es una cadena de texto'
2 separador = ' '
```
executed in 11ms, finished 02:42:36 2023-01-10

1.1.4.1 Convertir una cadena en una lista de acuerdo a un separador

In [370]:
```
1 lst_palabras = cadena.split(separador)
2 lst_palabras
```
executed in 12ms, finished 02:42:36 2023-01-10

Out[370]: ['Esta', 'es', 'una', 'cadena', 'de', 'texto']

1.1.4.2 Convertir una lista en una cadena

In [371]:
```
1 separador.join(lst_palabras)
```
executed in 12ms, finished 02:42:36 2023-01-10

Out[371]: 'Esta es una cadena de texto'

In [372]:
```
1 '_'.join(lst_palabras)
```

Out[372]: 'Esta_es_una_cadena_de_texto'

1.1.5 Copiar Listas

In [373]:
```
1 shopping
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[373]: ['Agua', 'Leche', 'Pan', 'Pimienta', 'Limón', 'Azúcar', 'Manzana']

1.1.5.1 Hacer otra referencia a la misma lista

In [374]:
```
1 shopping_ref = shopping
2 shopping_ref
```
executed in 12ms, finished 02:42:36 2023-01-10

Out[374]: ['Agua', 'Leche', 'Pan', 'Pimienta', 'Limón', 'Azúcar', 'Manzana']

In [375]:
```
1 #Y si cambio un elemento de shopping_ref, se cambiará en la lista original
2 shopping_ref[2] = 'Galletas'
3 shopping_ref
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[375]: ['Agua', 'Leche', 'Galletas', 'Pimienta', 'Limón', 'Azúcar', 'Manzana']

In [376]:
```
1 shopping
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[376]: ['Agua', 'Leche', 'Galletas', 'Pimienta', 'Limón', 'Azúcar', 'Manzana']

1.1.5.2 Copia de una lista de elementos inmutables

In [377]:
```
1 shopping_copy = shopping.copy()
2 shopping_copy
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[377]: ['Agua', 'Leche', 'Galletas', 'Pimienta', 'Limón', 'Azúcar', 'Manzana']

In [378]:
```
1 #Si cambio un elemento de la copia
2 shopping_copy[2] = 'Tostadas'
3 shopping_copy
```
executed in 10ms, finished 02:42:36 2023-01-10

Out[378]: ['Agua', 'Leche', 'Tostadas', 'Pimienta', 'Limón', 'Azúcar', 'Manzana']

In [379]:
```
1 # La lista original no se modifica
2 shopping
```
executed in 12ms, finished 02:42:36 2023-01-10

Out[379]: ['Agua', 'Leche', 'Galletas', 'Pimienta', 'Limón', 'Azúcar', 'Manzana']

1.1.5.3 Desempaquetado

In [380]:
```
1 writer1, writer2, writer3 = ['Borges', 'Cortázar', 'Piñeiro']
```
executed in 12ms, finished 02:42:36 2023-01-10

In [381]:
```
1 writer1
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[381]: 'Borges'

In [382]:
```
1 writer2
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[382]: 'Cortázar'

In [383]:
```
1 writer3
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[383]: 'Piñeiro'

1.2 Tuplas

El concepto de tupla es muy similar al de lista. Aunque hay algunas diferencias menores, lo fundamental es que, mientras una lista es mutable y se puede modificar, una tupla no admite cambios y por lo tanto, es inmutable

In [384]:
```
1 empty_tuple = ()
2 tenerife_geoloc = (28.46824, -16.25462)
3 three_wise_men = ('Melchor', 'Gaspar', 'Baltasar')
4 three_wise_men
```
executed in 13ms, finished 02:42:36 2023-01-10

Out[384]: ('Melchor', 'Gaspar', 'Baltasar')

In [385]:
```
empty_tuple = ()
```

In [386]:
```
type(empty_tuple)
```

Out[386]:
```
tuple
```

In [387]:
```
#Tupla de un elemento
one_item_tuple = ('Papá Noel',)
one_item_tuple
```

executed in 12ms, finished 02:42:36 2023-01-10

Out[387]:
```
('Papá Noel',)
```

In [388]:
```
# Tupla sin usar paréntesis
tupla_shopping = writer1, writer2, writer3
tupla_shopping
```

executed in 12ms, finished 02:42:36 2023-01-10

Out[388]:
```
('Borges', 'Cortázar', 'Piñeiro')
```

1.2.1 Funciones incorporadas de python aplicables a tuplas

Todas las funciones que vimos para listas aplican también a tuplas salvo las que intenten modificarla

1.2.1.1 Convertir un tipo de dato iterable a tupla

In [389]:
```
shopping = ['Agua', 'Aceite', 'Arroz']
```

executed in 11ms, finished 02:42:37 2023-01-10

In [390]:
```
# Lista a tupla
tuple_shopping = tuple(shopping)
tuple_shopping
```

executed in 13ms, finished 02:42:37 2023-01-10

Out[390]:
```
('Agua', 'Aceite', 'Arroz')
```

In [391]:
```
tuple_shopping
```

Out[391]:
```
('Agua', 'Aceite', 'Arroz')
```

In [392]:
```
tupla_shopping
```

Out[392]:
```
('Borges', 'Cortázar', 'Piñeiro')
```

In [393]:
```
tupla_lista = (tupla_shopping, 56, 'hola')
```

In [394]:
```
tupla_lista
```

Out[394]:
```
(('Borges', 'Cortázar', 'Piñeiro'), 56, 'hola')
```

In [395]:
```
tupla_lista[0].append('Bananas')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_28168\4239812302.py in <module>
----> 1 tupla_lista[0].append('Bananas')

AttributeError: 'tuple' object has no attribute 'append'
```

In [ ]:
```
tupla_lista
```

Out[271]:
```
(['Borges', 'Cortázar', 'Piñeiro', 'Yerba', 'Yerba', 'Yerba', 'Bananas'],)
```

1.2.1.2 Desempaquetado

In [ ]:
```
item1, item2, item3 = tuple_shopping
```

executed in 13ms, finished 02:42:37 2023-01-10

In [ ]:
```
item1
```

executed in 13ms, finished 02:42:37 2023-01-10

Out[207]:
```
'Agua'
```

In [ ]:
```
item2
```

executed in 12ms, finished 02:42:37 2023-01-10

Out[208]:
```
'Aceite'
```

In [ ]:
```
item3
```

executed in 10ms, finished 02:42:37 2023-01-10

Out[209]:
```
'Arroz'
```

1.2.2 Métodos de las tuplas

Algunos métodos son:

In [ ]:
```
dir(tuple)
```

Out[276]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'count',
 'index']
```

In [ ]:
```
tupla_swap = (21, 56)
```

In [ ]:
```
valor1, valor2 = tupla_swap
```

In [ ]:
```
id(tupla_swap)
```

Out[285]:
```
2527723343496
```

In [ ]:
```
tupla_swap = valor2, valor1
```

In [ ]:
```
tupla_swap
```

Out[294]:
```
(56, 21)
```

In [ ]:
```
id(tupla_swap)
```

Out[293]:
```
2527753433864
```

1.3 Diccionarios

Un diccionario es un objeto indexado por claves (como las palabras en un diccionario de lenguaje) que tienen asociados unos valores (los significados)

- Mantienen el orden en el que se insertan las claves
- Son mutables, con lo que admiten añadir, borrar y modificar sus elementos.
- Las claves deben ser únicas. A menudo se utilizan las cadenas de texto como claves, pero en realidad podría ser cualquier tipo de datos inmutable: enteros, flotantes, tuplas (entre otros).
- Tienen un acceso muy rápido a sus elementos, debido a la forma en la que están implementados internamente.

In [ ]:
```
empty_dict = {}
rae =  {'bifronte': 'De dos frentes o dos caras',
        'anarcoide': 'Que tiende al desorden',
        'montuvio': 'Campesino de la costa'
}

population_can = {
                    2015: 2_135_209,
                    2016: 2_154_924,
                    2017: 2_177_048,
                    2018: 2_206_901,
                    2019: 2_220_270
}
```

executed in 13ms, finished 02:42:37 2023-01-10

In [ ]:
```
1 rae
```
executed in 12ms, finished 02:42:37 2023-01-10

Out[211]:
```
{'bifronte': 'De dos frentes o dos caras',
 'anarcoide': 'Que tiende al desorden',
 'montuvio': 'Campesino de la costa'}
```

In [ ]:
```
1 population_can
```
executed in 13ms, finished 02:42:37 2023-01-10

Out[212]:
```
{2015: 2135209, 2016: 2154924, 2017: 2177048, 2018: 2206901, 2019: 2220270}
```

1.3.1 Funciones incorporadas de python aplicables a diccionarios

1.3.1.1 Crear un diccionario, o convertir otro objeto en diccionario

In [ ]:
```
1 # Convertir una lista de listas en diccionario
2 dict([('a', 1), ('b', 2)])
```
executed in 13ms, finished 02:42:37 2023-01-10

Out[296]:
```
{'a': 1, 'b': 2}
```

1.3.1.2 Cantidad de elemento

In [ ]:
```
1 len(rae)
```
executed in 12ms, finished 02:42:37 2023-01-10

Out[214]:
```
3
```

1.3.1.3 Borar un elemento

In [ ]:
```
1 del rae['montuvio']
2 rae
```
executed in 12ms, finished 02:42:37 2023-01-10

Out[215]:
```
{'bifronte': 'De dos frentes o dos caras',
 'anarcoide': 'Que tiende al desorden'}
```

1.3.1.4 Comprobar si una clave existe en el diccionario

In [ ]:
```
1 'bifronte' in rae
```
executed in 9ms, finished 02:42:37 2023-01-10

Out[216]:
```
True
```

1.3.2 Operaciones con diccionarios

1.3.2.1 Obtener un elemento

In [ ]:
```
1 rae['anarcoide']
```
executed in 13ms, finished 02:42:37 2023-01-10

Out[217]:
```
'Que tiende al desorden'
```

1.3.2.2 Añadir un elemento

- Si la clave ya existía en el diccionario, se reemplaza el valor existente por el nuevo.
- Si la clave es nueva, se añade al diccionario con su valor. No vamos a obtener un error a diferencia de las listas.

In [ ]:
```
1 #Añade un elemento nuevo
2 rae['enjuiciar'] = 'Someter una cuestión a examen, discusión y juicio'
```
executed in 11ms, finished 02:42:37 2023-01-10

In [ ]:
```
1 rae
```
executed in 14ms, finished 02:42:37 2023-01-10

Out[219]:
```
{'bifronte': 'De dos frentes o dos caras',
 'anarcoide': 'Que tiende al desorden',
 'enjuiciar': 'Someter una cuestión a examen, discusión y juicio'}
```

1.3.2.3 Modificar un elemento existente

In [ ]:
```
1 rae['enjuiciar'] = 'Instruir, juzgar o sentenciar una causa'
```
executed in 13ms, finished 02:42:37 2023-01-10

In [ ]:
```
rae
```
executed in 14ms, finished 02:42:37 2023-01-10

Out[221]:
```
{'bifronte': 'De dos frentes o dos caras',
 'anarcoide': 'Que tiende al desorden',
 'enjuiciar': 'Instruir, juzgar o sentenciar una causa'}
```

1.3.2.4 Fusionar diccionarios

- Si la clave no existe, se añade con su valor.
- Si la clave ya existe, se añade con el valor del «último» diccionario en la mezcla

In [ ]:
```
rae2 = {
    'verificar': 'Comprobar o examinar la verdad de algo',
    'montuvio': 'Campesino de la costa',
    'enjuiciar': 'Sujetar a alguien a juicio'
}
```
executed in 13ms, finished 02:42:37 2023-01-10

In [ ]:
```
{**rae, **rae2}
```
executed in 12ms, finished 02:42:37 2023-01-10

Out[223]:
```
{'bifronte': 'De dos frentes o dos caras',
 'anarcoide': 'Que tiende al desorden',
 'enjuiciar': 'Sujetar a alguien a juicio',
 'verificar': 'Comprobar o examinar la verdad de algo',
 'montuvio': 'Campesino de la costa'}
```

In [ ]:
```
print(dir(dict))
```

```
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

1.3.3 Métodos de los diccionarios

1.3.3.1 Obtener un elemento, si no está devuelve None

In [ ]:
```
print(rae.get('elasticidad'))
```
executed in 13ms, finished 02:42:37 2023-01-10

```
None
```

1.3.3.2 Obtener las claves

In [ ]:
```
rae.keys()
```
executed in 10ms, finished 02:42:37 2023-01-10

Out[310]:
```
dict_keys(['bifronte', 'anarcoide', 'enjuiciar', 'verificar', 'capturar'])
```

1.3.3.3 Obtener los valores

In [ ]:
```
rae.values()
```
executed in 14ms, finished 02:42:37 2023-01-10

Out[226]:
```
dict_values(['De dos frentes o dos caras', 'Que tiende al desorden', 'Instruir, juzgar o sentenciar una causa'])
```

1.3.3.4 Obtener los pares clave - valor

In [ ]:
```
rae.items()
```
executed in 14ms, finished 02:42:37 2023-01-10

Out[227]:
```
dict_items([('bifronte', 'De dos frentes o dos caras'), ('anarcoide', 'Que tiende al desorden'), ('enjuiciar', 'Instruir, juzgar o sentenciar una causa')])
```

1.3.3.5 Agregar claves nuevas desde otro diccionario, modificando el original

In [ ]:
```
rae3 = {
    'verificar': 'Salir cierto y verdadero lo que se dijo',
    'capturar': ' Aprehender, apoderarse de alguien o algo'
}
```
executed in 14ms, finished 02:42:37 2023-01-10

In [ ]:
```
rae.update(rae3)
```
executed in 13ms, finished 02:42:37 2023-01-10

In [ ]:
```
1 rae
```

executed in 12ms, finished 02:42:37 2023-01-10

Out[230]:
```
{'bifronte': 'De dos frentes o dos caras',
 'anarcoide': 'Que tiende al desorden',
 'enjuiciar': 'Instruir, juzgar o sentenciar una causa',
 'verificar': 'Salir cierto y verdadero lo que se dijo',
 'capturar': ' Aprehender, apoderarse de alguien o algo'}
```

1.3.4 Iterar elementos

In [ ]:
```
1 shopping.extend(shopping2)
```

In [ ]:
```
1 shopping
```

Out[315]:
```
['Agua', 'Aceite', 'Arroz', 'Naranja', 'Manzana']
```

In [ ]:
```
1 # Iterar una lista
2 for item in shopping:
3     print(item)
```

```
Agua
Aceite
Arroz
Naranja
Manzana
```

In [ ]:
```
1 # Iterar un diccionario, por sus claves
2 for elemento in rae.keys():
3     print(elemento)
```

```
bifronte
anarcoide
enjuiciar
verificar
capturar
```

In [ ]:
```
1 # Iterar un diccionario, por sus claves
2 for elemento in rae.values():
3     print(elemento)
```

```
De dos frentes o dos caras
Que tiende al desorden
Instruir, juzgar o sentenciar una causa
Salir cierto y verdadero lo que se dijo
 Aprehender, apoderarse de alguien o algo
```

In [ ]:
```
1 # Iterar un diccionario, por sus claves
2 for clave, valor in rae.items():
3     print(f'clave: {clave} -> {valor}')
```

```
clave: bifronte -> De dos frentes o dos caras
clave: anarcoide -> Que tiende al desorden
clave: enjuiciar -> Instruir, juzgar o sentenciar una causa
clave: verificar -> Salir cierto y verdadero lo que se dijo
clave: capturar ->  Aprehender, apoderarse de alguien o algo
```

In [ ]:
```
1 for clave in rae.keys():
2     print(f'clave: {clave} -> {rae[clave]}')
```

```
clave: bifronte -> De dos frentes o dos caras
clave: anarcoide -> Que tiende al desorden
clave: enjuiciar -> Instruir, juzgar o sentenciar una causa
clave: verificar -> Salir cierto y verdadero lo que se dijo
clave: capturar ->  Aprehender, apoderarse de alguien o algo
```

In [ ]:
```
1 # Cambiar un elemento que antes era un string por un diccionario
2 rae['verificar']={1: 'Salir cierto y verdadero lo que se dijo'}
```

In [ ]:
```
1 rae
```

Out[327]:
```
{'bifronte': 'De dos frentes o dos caras',
 'anarcoide': 'Que tiende al desorden',
 'enjuiciar': 'Instruir, juzgar o sentenciar una causa',
 'verificar': {1: 'Salir cierto y verdadero lo que se dijo'},
 'capturar': ' Aprehender, apoderarse de alguien o algo'}
```

In [ ]:
```
1 rae['verificar']
```

Out[328]:
```
{1: 'Salir cierto y verdadero lo que se dijo'}
```

In [ ]:
```
1 # Agregar un elemento al diccionario que creamos internamente
2 rae['verificar'][2] = 'Otro'
```

In [ ]:
```
1 rae['verificar'][3] = 'Otro2'
```

In [ ]:
```
rae
```

Out[332]:
```
{'bifronte': 'De dos frentes o dos caras',
 'anarcoide': 'Que tiende al desorden',
 'enjuiciar': 'Instruir, juzgar o sentenciar una causa',
 'verificar': {1: 'Salir cierto y verdadero lo que se dijo',
  2: 'Otro',
  3: 'Otro2'},
 'capturar': ' Aprehender, apoderarse de alguien o algo'}
```

In [ ]:
```
#Mostrar todos los elementos del diccionario principal y del interno
for palabra in rae.keys():
    if isinstance(rae[palabra], dict):
        for clave, valor in rae[palabra].items():
            print(f'clave: {palabra} significado {clave} = {valor}')
    else:
        print(f'clave: {palabra} -> {rae[palabra]}')
```

```
clave: bifronte -> De dos frentes o dos caras
clave: anarcoide -> Que tiende al desorden
clave: enjuiciar -> Instruir, juzgar o sentenciar una causa
clave: verificar significado 1 = Salir cierto y verdadero lo que se dijo
clave: verificar significado 2 = Otro
clave: verificar significado 3 = Otro2
clave: capturar ->  Aprehender, apoderarse de alguien o algo
```

In [ ]:
```

```