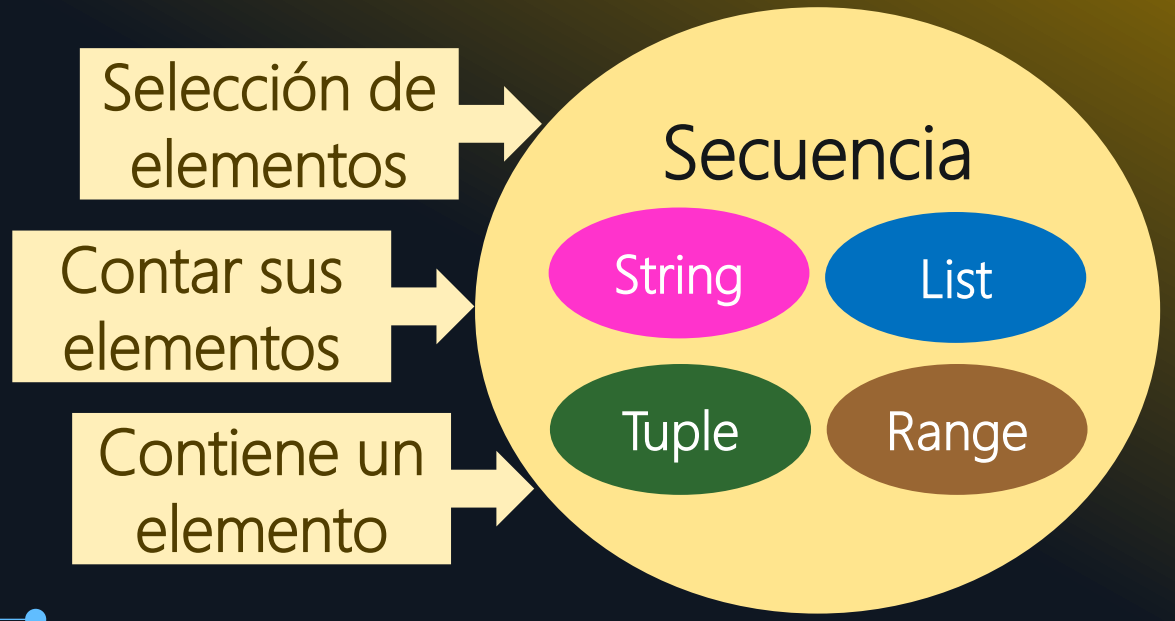
Curso de Python

3 – Colecciones

Ramón Invarato Menéndez

Ricardo Moya García

Secuencia



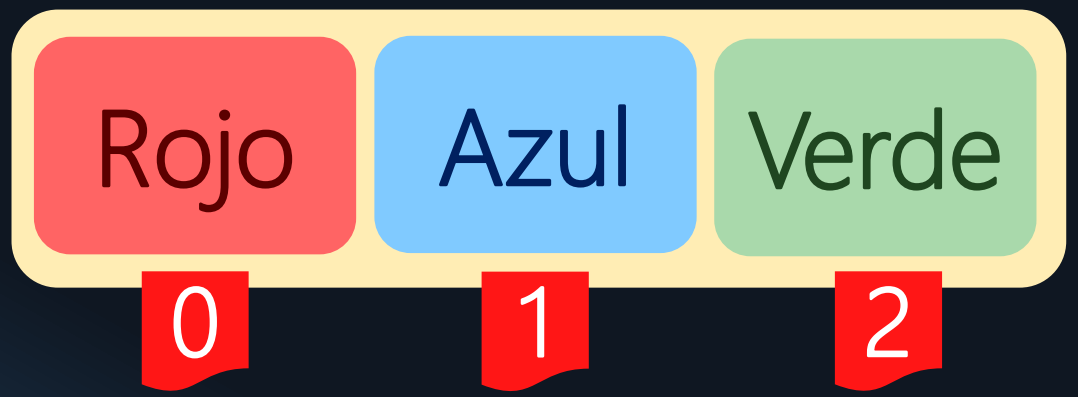
Con índice desde la posición 0 a <cantidad de elementos>-1

Código

```
string_color = "Azul"
```

Código

```
lista_colores = ["Rojo", "Azul", "Verde"]
```



Secuencia

String, List, Tuple, Range, Buffer, Bytes

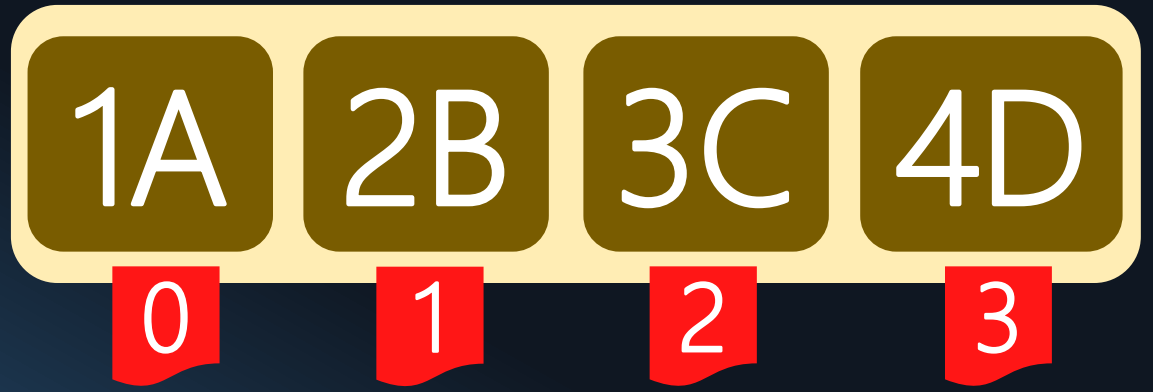
Código

```
lista_de_textos = ["1A", "2B", "3C", "4D"]
lista_de_textos
```

print

Consola

```
["1A", "2B", "3C", "4D"]
```



Seleccionar un elemento por índice

elemento = secuencia[posición]

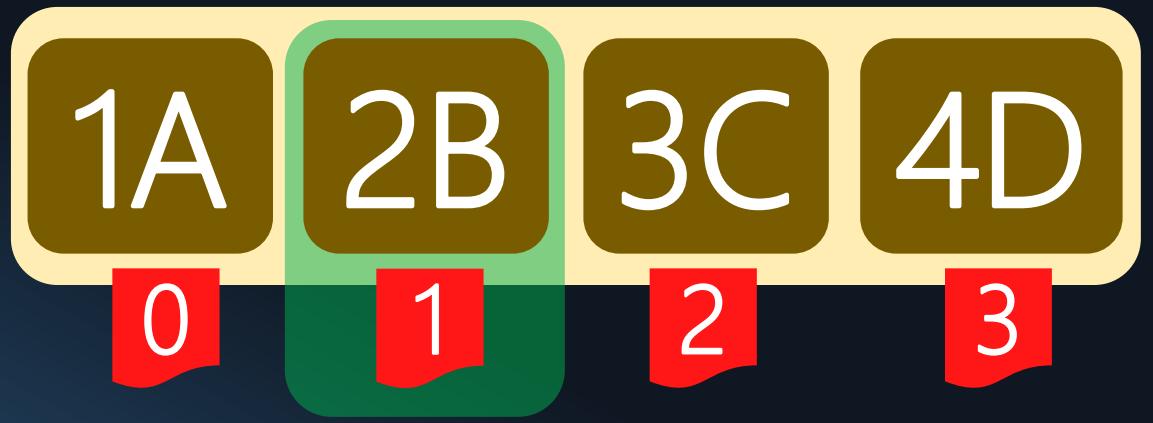
Código

```
mi_elemento = lista_de_textos[1]
mi_elemento
```

print

Consola

```
2B
```



Seleccionar un elemento por posición invertida

elemento = secuencia[-posicion]

Código

```
mi_elemento = lista_de_textos[-1]
mi_elemento
```

print

Consola

```
4D
```

Seleccionar varios elementos

elementos = secuencia[inicio:fin_excluido]

elementos = secuencia[inicio:]

elementos = secuencia[:fin_excluido]

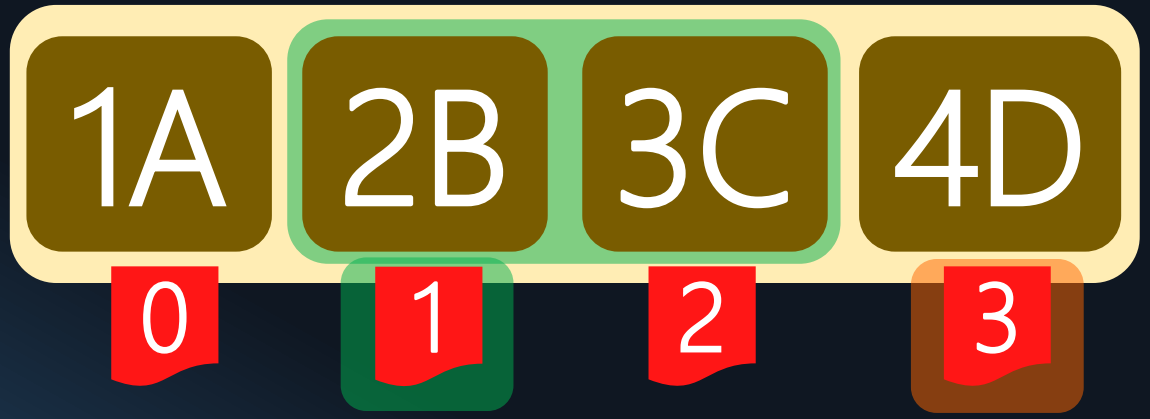
Código

```
elementos = lista_de_textos[1:3]
elementos
```

print

Consola

```
['2B', '3C']
```



Seleccionar varios elementos

```
elementos = secuencia[inicio:fin_excluido]
elementos = secuencia[inicio:]
elementos = secuencia[:fin_excluido]
```

Código

```
elementos = lista_de_textos[1:]
elementos
```

print

Consola

```
['2B', '3C', '4D']
```

Seleccionar varios elementos

```
elementos = secuencia[inicio:fin_excluido]
elementos = secuencia[inicio:]
elementos = secuencia[:fin_excluido]
```

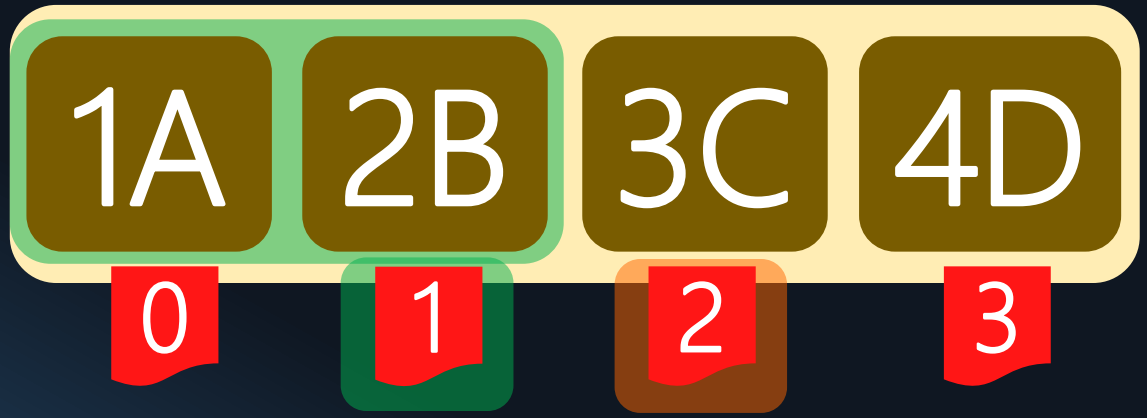
Código

```
elementos = lista_de_textos[:2]
elementos
```

print

Consola

```
['1A', '2B']
```



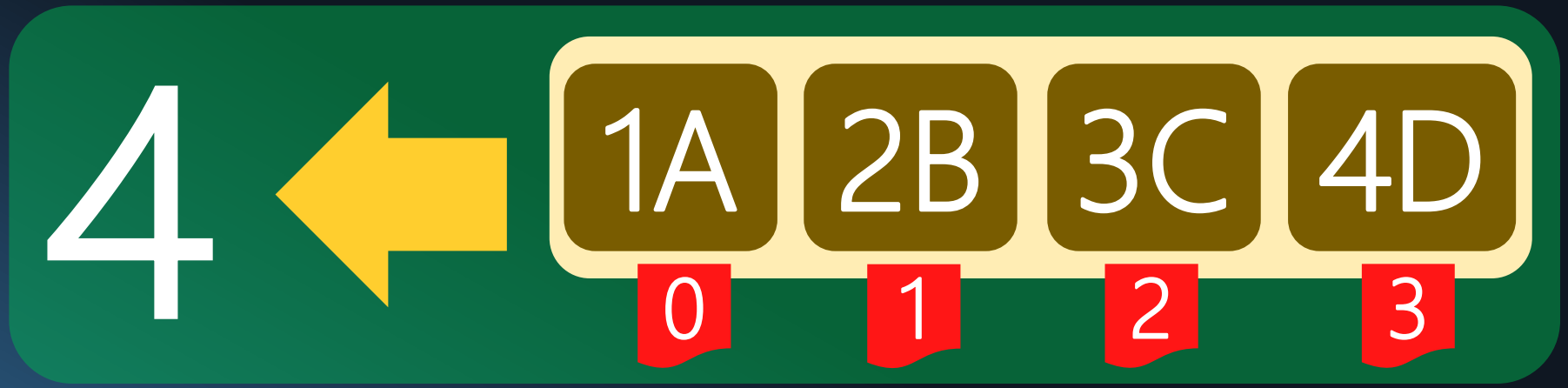
Contar elementos

```
cantidad_elementos = len(secuencia)
```

```
Código
cantidad_elementos = len(lista_de_textos)
cantidad_elementos
```

print

```
Consola
4
```



Contiene elemento

existe = elemento in secuencia

no_existe = elemento not in secuencia

Código

```
existe = "2B" in lista_de_textos
existe
```

print

Consola

True

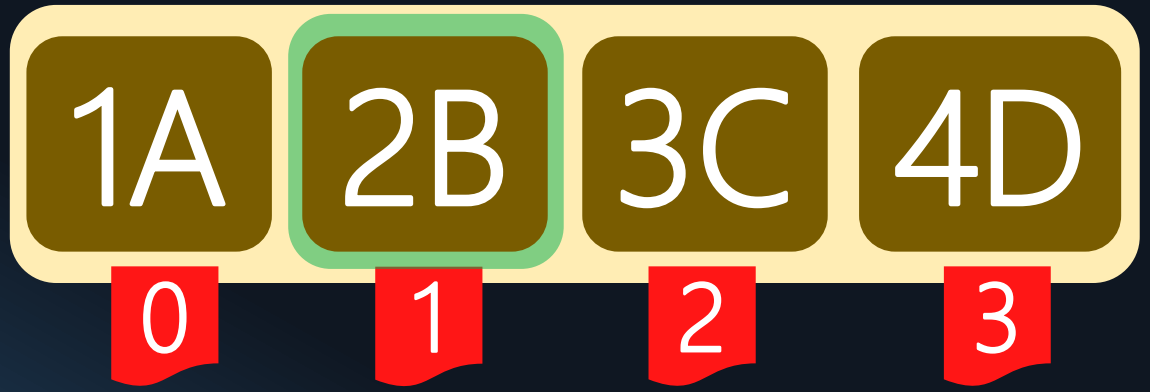
Código

```
no_existe = "0X" not in lista_de_textos
no_existe
```

print

Consola

True



Índex de elemento

```
posicion = secuencia.index(elemento)
```

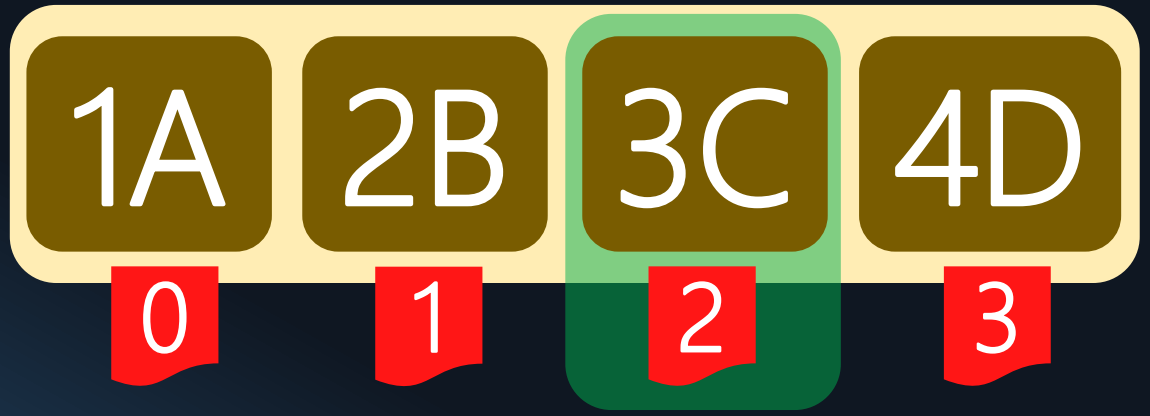
Código

```
posicion = lista_de_textos.index("3C")
posicion
```

print

Consola

```
2
```



Invertir elementos

secuencia.reverse()

Código

```
lista_de_textos.reverse()
lista_de_textos
```

print

Consola

```
['4D', '3C', '2B', '1A']
```

Contar elementos

```
veces = secuencia.count(elemento)
```

Código

```
texto = "canasta"
veces_a = texto.count("a")
veces_a
```

print

Consola

```
3
```

Concatenar secuencias

secuencia_concatenada = secuencia_1 + secuencia_2

Código

```
frutas = ["Manzana", "Platano", "Piña"]
verduras = ["Acelgas", "Judias"]

frutas_y_verduras = frutas + verduras
```

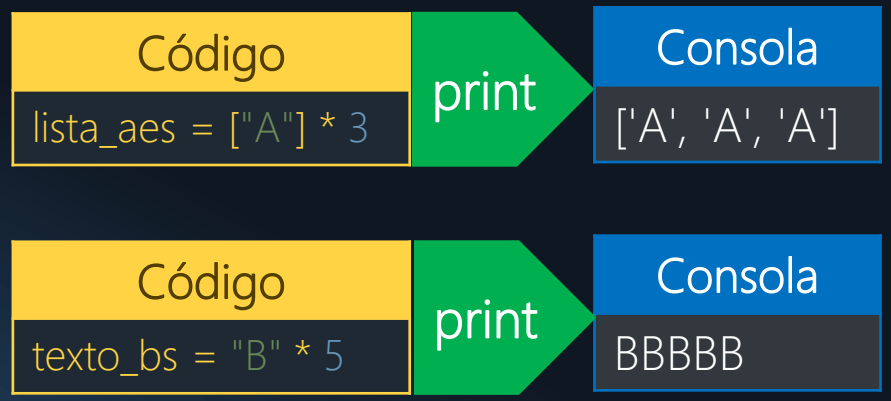
print

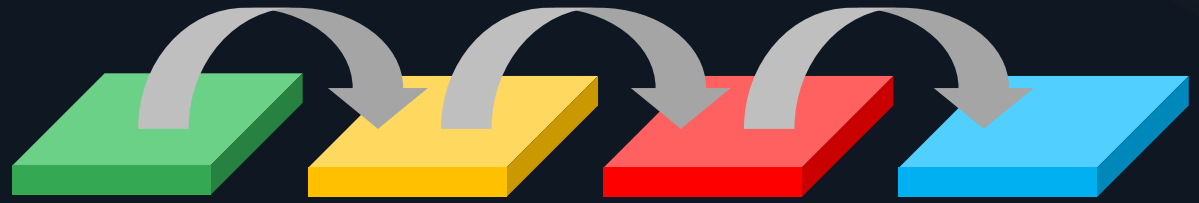
Consola

```
['Manzana', 'Platano', 'Piña', 'Acelgas', 'Judias']
```

Concatenar secuencias N veces

secuencia_n_veces_concatenada = secuencia * veces





Lista (List)

Secuencia de valores

- Se puede imprimir fácilmente la lista con "**print**"
- Se pueden **concatenar listas con** el operador **+**
- Para **copiar una lista** hay que pasársela por parámetro a **"list(lista)"**
- Como es una **Secuencia** se opera igual que String:
  - Se itera en "**for**"
  - Se seleccionan los elementos "**[inicio:fin]**"
  - Se puede ver su tamaño con "**len**"

Código

```
lista_de_caracteres = ["T", "e", "x", "t", "o"]

lista_de_textos = ["Texto1", "Texto2", "Texto3", "Texto4", "Texto5"]

lista_de_numeros = [1, 2, 3, 4, 5]

lista_de_listas = [["1A", "1B", "1C"],
                   ["2A", "2B", "2C"],
                   ["3A", "3B", "3C"]]
```

Código

```
print(lista_de_textos)

concatenadas = lista_de_textos + lista_de_caracteres

lista_copiada = list(lista_de_textos)

un_texto = lista_de_textos[2]
tres_ultimos_textos = lista_de_textos[2::]

longitud = len(lista_de_textos)

for texto in lista_de_textos:
    print(texto)
```

Ejemplos de listas

Vector (lista "simple")

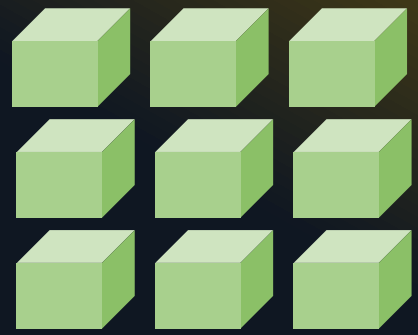
Código

```
lista = ["1A", "1B", "1C"]
```

Matriz

Código

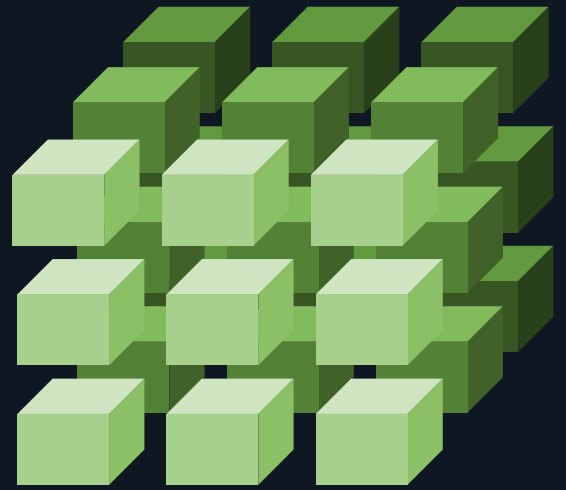
```
lista_de_listas = [ ["1A", "1B", "1C"],
                    ["2A", "2B", "2C"],
                    ["3A", "3B", "3C"] ]
```



Matriz tridimensional

Código

```
lista_de_listas_de_listas = [
                              [ ["1A", "1B", "1C"], ["2A", "2B", "2C"],  ["3A", "3B", "3C"] ],
                              [ ["4A", "4B", "4C"], ["5A", "5B", "5C"],  ["6A", "6B", "6C"] ],
                              [ ["7A", "7B", "7C"], ["8A", "8B", "8C"],  ["9A", "9B", "9C"] ]
                            ]
```

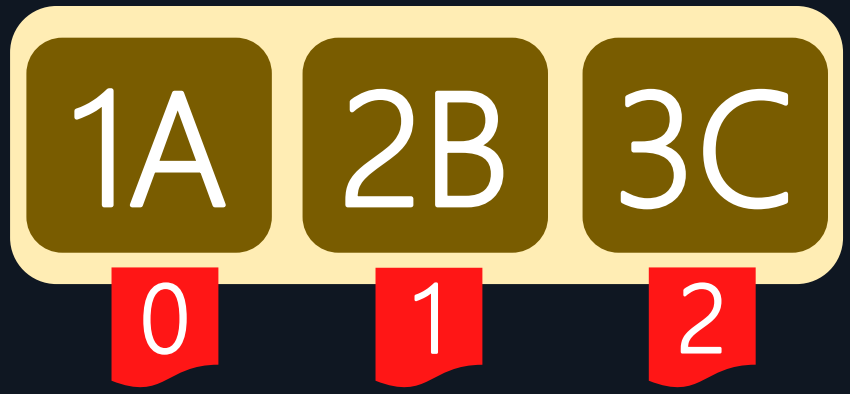


Inicializar una lista

variable = []

variable = list()

- La inicialización a una lista vacía puede ser con "[]" o con "list()"
- Una lista de objetos que pueden ser de cualquier tipo:
  - Strings
  - Lists
  - Ints



Adjuntar un elemento al final de la lista

lista.append(elemento)

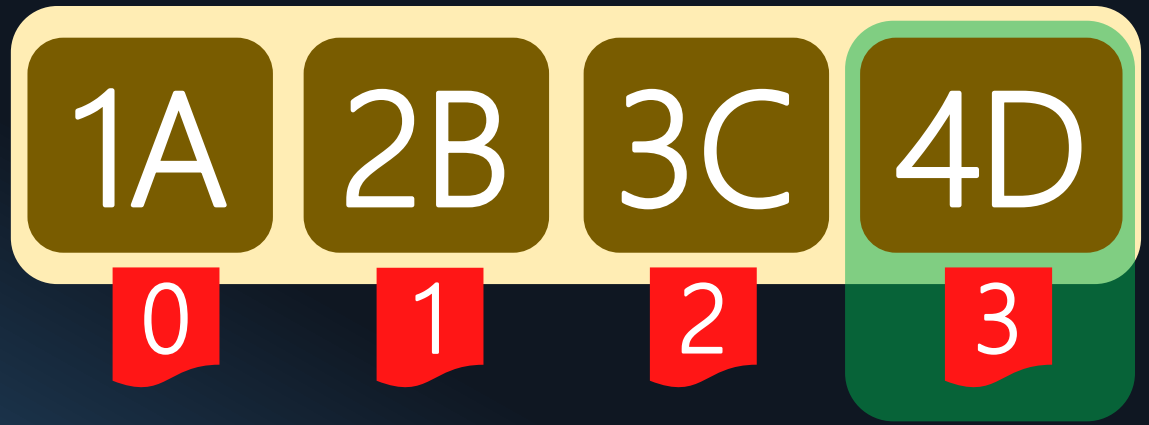
Código

```
lista_de_textos.append("4D")
lista_de_textos
```

print

Consola

```
["1A", "2B", "3C", "4D"]
```



Inserta un elemento en una posición

lista.insert(posicion, elemento)

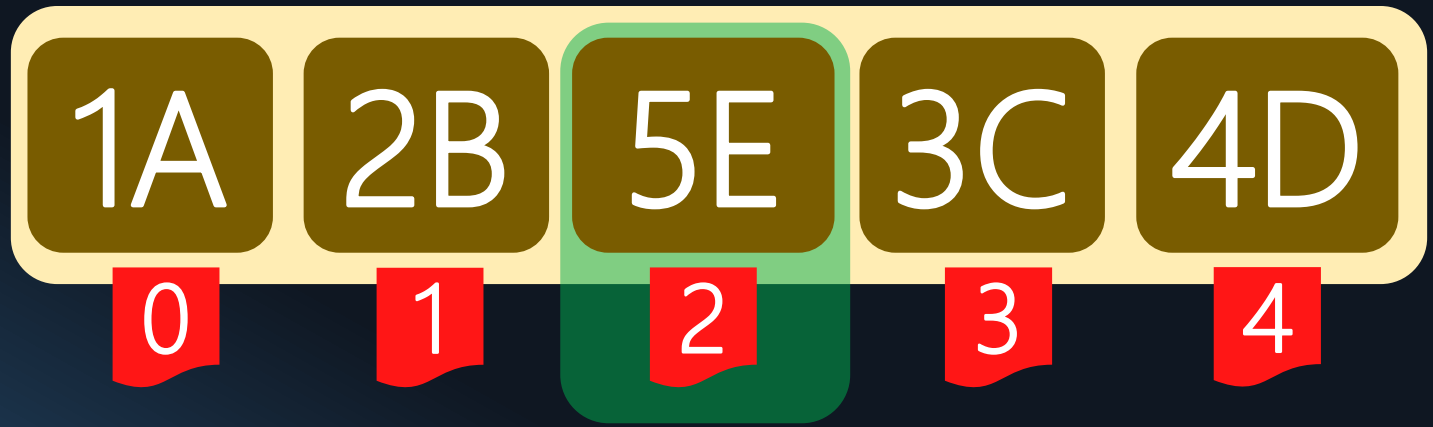
Código

```
lista_de_textos.insert(2, "5E")
lista_de_textos
```

print

Consola

```
["1A", "2B", "5E", "3C", "4D"]
```



Código

```
lista_de_textos[1] = "9Z"
lista_de_textos
```

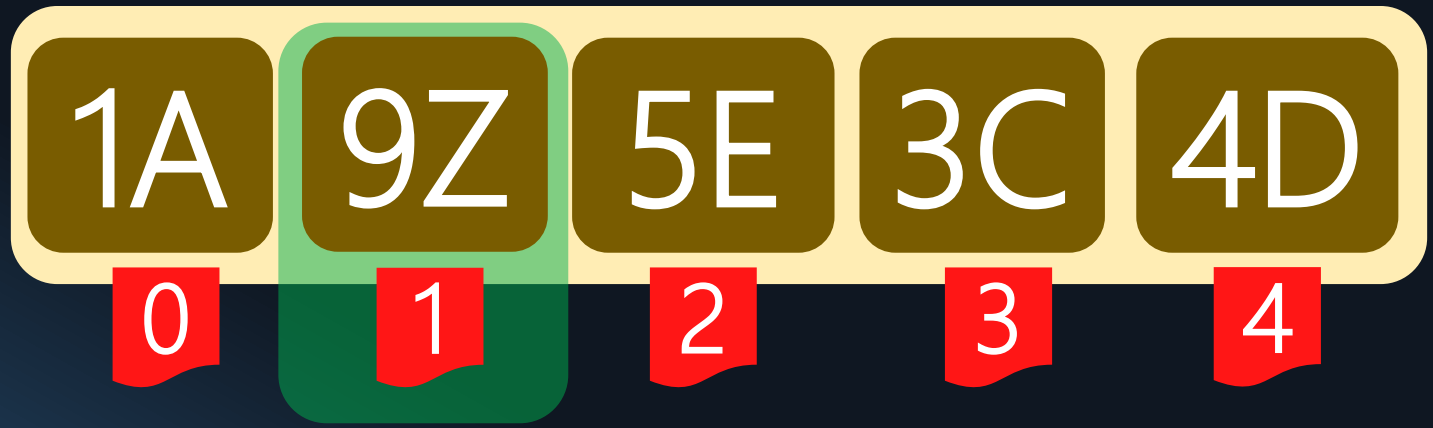
print

Consola

```
["1A", "9Z", "5E", "3C", "4D"]
```

Modificar un elemento

```
lista[posicion] = nuevo_valor
```



Extrae un elemento

elemento = lista.pop(posicion)

Código

```
valor = lista_de_textos.pop(3)
```

print

Consola

```
"3C"
```

Código

```
lista_de_textos
```

print

Consola

```
["1A", "9Z", "5E", "4D"]
```

Quitar un elemento

lista.remove(valor)

Código

```
lista_de_textos.remove("2B")
lista_de_textos
```

print

Consola

```
["1A", "5E", "4D"]
```

Eliminar elemento por selección

del lista[posicion]

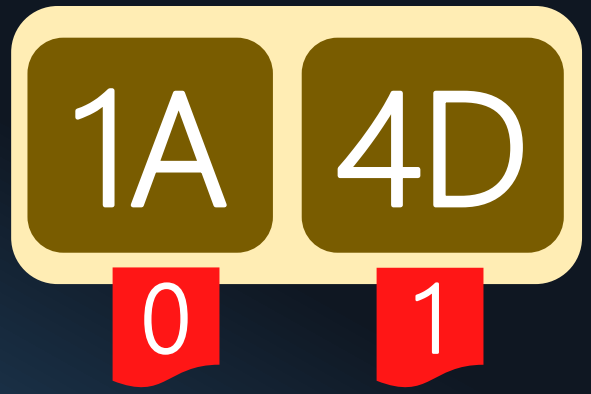
Código

```
del lista_de_textos[2]
lista_de_textos
```

print

Consola

```
['1A', '4D']
```



Resumen de métodos de las listas

Declaración de una lista:
lista = list()

Cuenta el número de elementos de la lista
len(lista)

Agrega un elemento (x) al final de la lista.
lista.append(x)

Inserta un elemento (x) en una posición determinada (pos)
lista.insert(pos, x)

Une dos listas (une la lista2 (la que se pasa como parámetro) a la lista)
lista.extend(lista2)

Borra el primer elemento de la lista cuyo valor sea x. Sino existe devuelve un error
lista.remove(x)

Borra el elemento en la posición dada de la lista, y lo devuelve.
lista.pop(pos)

Borra todos los elementos de la lista. Equivalente a lista.clear()
del lista[:]

Devuelve el índice en la lista del primer elemento cuyo valor sea x
lista.index(x)

Devuelve el número de veces que x aparece en la lista.
lista.count(x)

Invierte los elementos de la lista.
lista.reverse()

Devuelve una copia de la lista. Equivalente a lista.copy()
lista_copia = lista[:]

String a List

lista = texto.split(delimitador)

Código

```
frase = "un pastel de manzana"
palabras = frase.split(" ")
palabras
```

print

Consola

```
['un', 'pastel', 'de', 'manzana']
```

List a String

texto = delimitador.join(lista)

Código

```
ingredientes = ["huevos", "harina", "levadura"]
receta = ", ".join(ingredientes)
receta
```

print

Consola

```
huevos, harina, levadura
```

huevos, harina, levadura

Resumen de métodos para String

Para ETL (Extract, Transform and Load)

Devuelve una lista con todos elementos encontrados al dividir el string por el separador.
string.split('separador')

Sustituye en el string todos aquellos elementos que coinciden con la cadena 'string_a' por 'string_b'.
string.replace('string_a', 'string_b')

Elimina los espacios en blanco que hay al principio y al final del string.
string.strip()

Convierte todo el string a mayúsculas.
string.upper()

Convierte todo el string a minúsculas
string.lower()

Dada una lista que se pasa como parámetro construye un string con todos los elementos de la lista separados por el separador que se define como un string
'separador'.join(lista)

* Una lista también se puede extender, pero hay que ponerle asterisco como prefijo

*[elementos]

Tupla (Tuple)

Secuencia de valores finales

Código

```
tupla_de_un_elemento = ("1 elemento",)
tupla_de_un_elemento = "1 elemento",
```

Código

```
tupla = ("Paco", "Coche", "Manzana")
tupla = "Paco", "Coche", "Manzana"

persona = tupla[0]
transporte = tupla[1]
comida = tupla[2]

persona, transporte, comida = tupla
```

Código

```
lista_de_tuplas = [
    ("Paco", "Coche", 1),
    ("María", "Barco", 2),
    ("Juan", "Avión", 3),
]

for tupla in lista_de_tuplas:
    print(tupla)

for persona, transporte, cantidad in lista_de_tuplas:
    print(persona, transporte, cantidad)
```

Tupla (Tuple)

Secuencia de valores finales

- La tupla permite las **mismas operaciones que los listados** (permite objetos de cualquier tipo, for, [selección], longitud, concatenación +, print, etc.).

Código

```
mi_tupla = ("Manzana", 100, "Plátano", 99.5, "Sandía", 100, "Manzana")
```

Código

```
fruta = mi_tupla[2]

sub_tupla = mi_tupla[1:3]

existe_sandia = "Sandía" in mi_tupla

cantidad_elementos = len(mi_tupla)

tupla_ampliada = mi_tupla + ("Verdura", 123)

for elemento in mi_tupla:
    print(elemento)
```

Inicializar una tupla con elementos

variable = (elemento,... , elemento)

variable = elemento,... , elemento

variable = tuple([elemento,... , elemento])

- Los **elementos** de una tupla pueden ir **dentro de paréntesis** (opcional)
- Para **copiar una tupla** hay que pasársela por parámetro a **"tuple(tupla)"**.

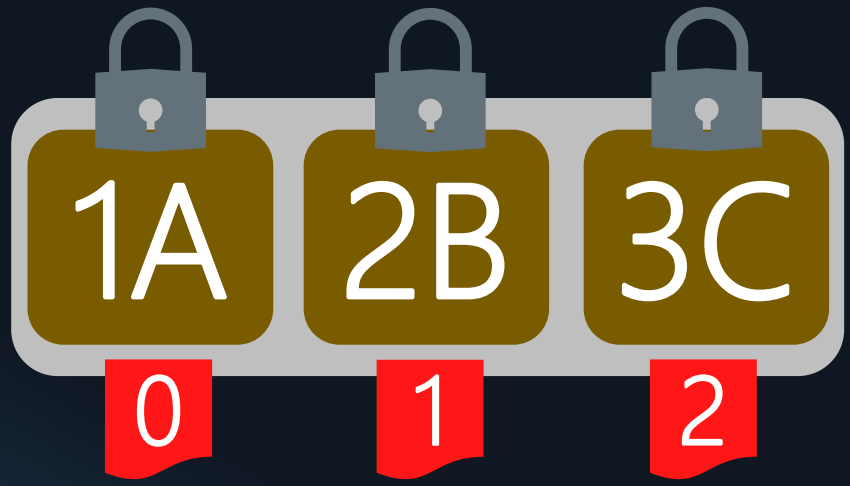
Código

```
tupla_de_textos = ("1A", "2B", "3C")

tupla_de_textos = "1A", "2B", "3C"

tupla_de_textos = tuple(["1A", "2B", "3C"])
```

print

Consola

("1A", "2B", "3C")



Inicializar una tupla con UN elemento

variable = (elemento,)

variable = elemento,

variable = tuple([elemento])

- Los **elementos de la tupla siempre van separados por comas**, aunque tenga uno (se pone una coma al final).

Código

```
tupla_de_textos = ("1A",)

tupla_de_textos = "1A",

tupla_de_textos = tuple(["1A"])
```

print

Consola

("1A",)

Código

```
un_string = ("1A")

un_sring = "1A"
```

print

Consola

"1A"

Empaquetar variables

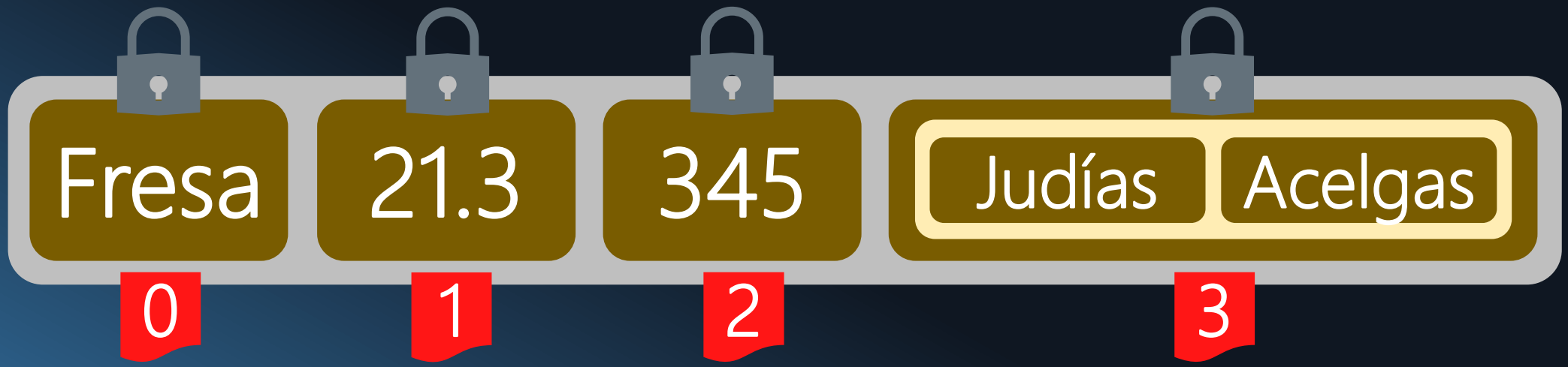
tupla = (variable1, ..., variableN)

tupla = variable1, ..., variableN

Código

```
fruta = "Fresa"
temperatura = 21.3
visitas = 345
verduras = ["Judías", "Acelgas"]

mi_tupla = (fruta, temperatura, visitas, verduras)
```

- Varios **valores se pueden empaquetar en una tupla** (una única variable con varios valores).



Código

```
mi_tupla2 = mi_tupla
```

Desempaquetar variables

(variable1, ..., variableN) = tupla

variable1, ..., variableN = tupla

Código

```
fruta = "Fresa"
temperatura = 21.3
visitas = 345
verduras = ["Judías", "Acelgas"]

mi_tupla = fruta, temperatura, visitas, verduras

# Más adelante, en el código ejecutado ...

fruta2, temperatura2, visitas2, verduras2 = mi_tupla
```

- Los elementos de una tupla se pueden **extender fácilmente en variables.**

fruta2 = "Fresa"

temperatura2 = 21.3

visitas = 345

Verduras = Judías Acelgas

Desempaquetar variables retornados por una función

variable1, ..., variableN = funcion(...)

(variable1, ..., variableN) = funcion(...)

Código

```
def mi_funcion():
    fruta = "Fresa"
    temperatura = 21.3
    visitas = 345
    verduras = ["Judías", "Acelgas"]

    return fruta, temperatura, visitas, verduras


fruta2, temperatura2, visitas2, verduras2 = mi_funcion()
```

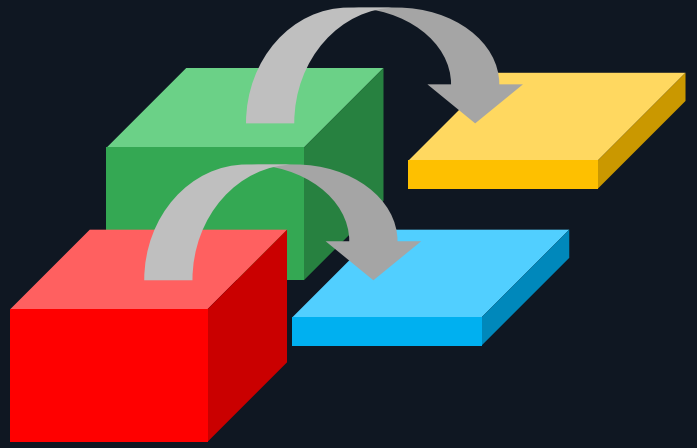
- Las tuplas se utilizan mucho **para pasar varios valores/variables de un sitio a otro** (el "return" en la declaración de las funciones, podrá devolver varios valores gracias a la tupla).

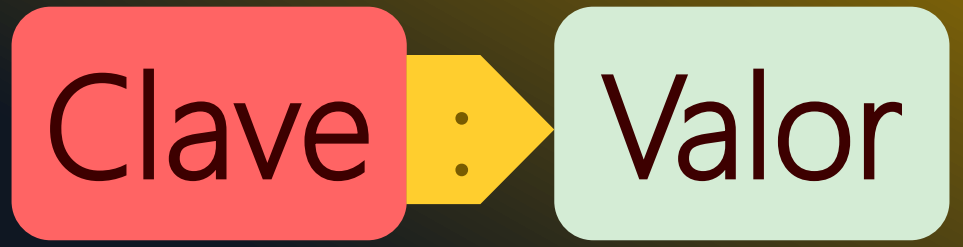
fruta2 = "Fresa"

temperatura2 = 21.3

visitas = 345

Verduras = Judías Acelgas





Diccionario (Dict)

Valores mapeados

- Un diccionario es **parecido a list**
- Un diccionario **NO es una secuencia** (es decir, no comparte la misma forma de operar que list, str o tupla)
- Un diccionario **es mutable** (las variables apuntan, no copian)
- Los **valores** de los diccionarios **son mutables,** por lo que puede cambiar el contenido de un valor en tiempo de ejecución
- Las **claves** de un diccionario deben ser **inmutables**
- Un diccionario **es una tabla hash**

Código

```
diccionario_persona = {
    "nombre": "Juan",
    "apellido": "López",
    "años": 25,
    "hijos": ["María", "Pedro", "Óscar"]
}
```

https://jarroba.com/tablas-hash-o-tabla-de-dispersion/

https://jarroba.com/resumibles-hashables-en-python/

Diccionario vs Listado

- Un **diccionario se diferencia del listado** en que:
  - **Almacena claves** que apuntan a valores
  - **Buscar** en su contenido es **inmediato** (las claves)

Listado

Código

```
lista_dnis = [
    7895478,
    5987561,
    6987453
]
```

Diccionario "actuando como un listado"

Código

```
diccionario_dnis = {
    7895478: None,
    5987561: None,
    6987453: None,
}
```

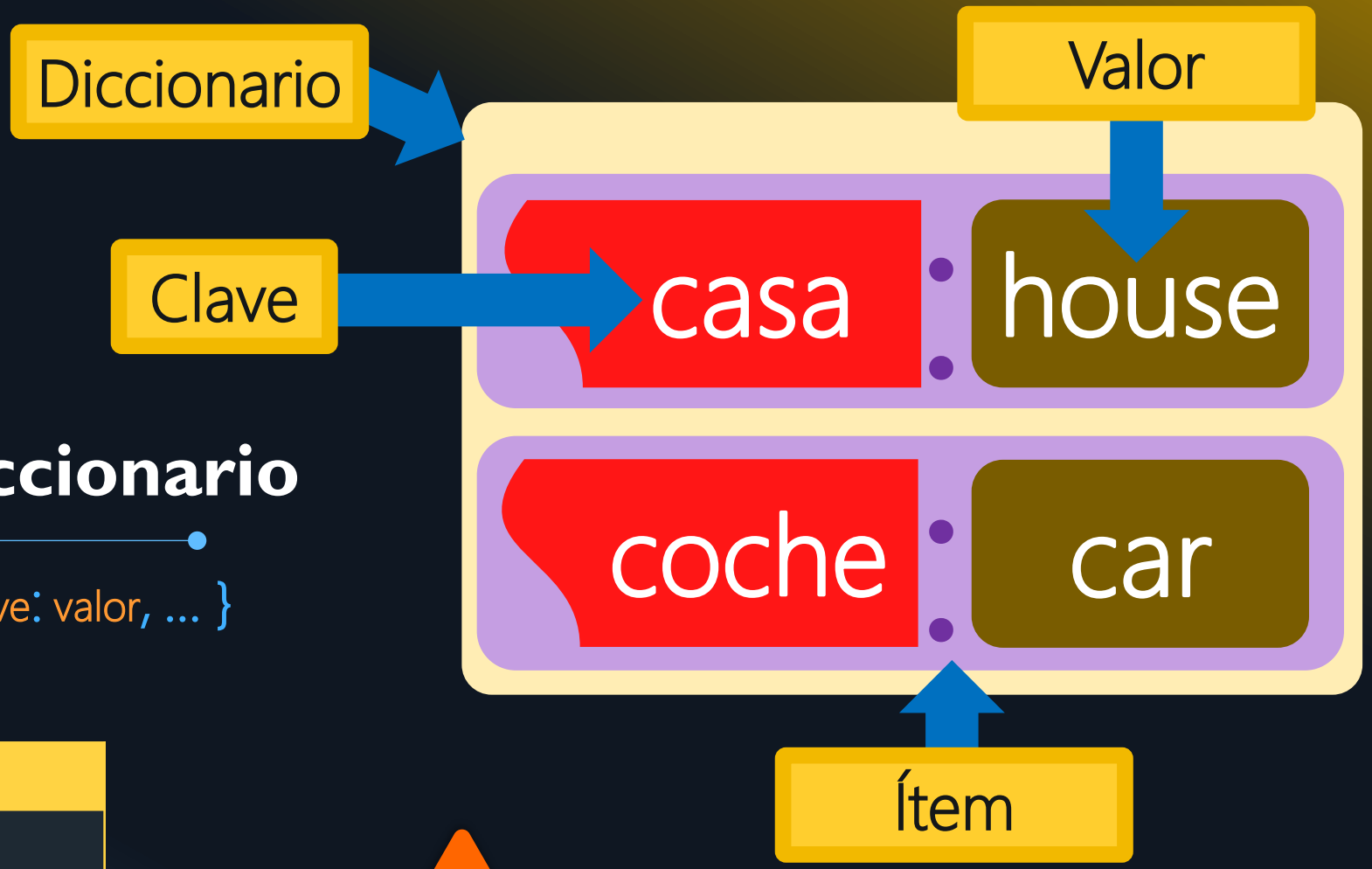
Inicializar un diccionario vacío

```
diccionario = {}

diccionario = dict()
```

- Un **diccionario vacío** se puede inicializar con **dict()** o **{}**

Código

```
diccionario = {}
diccionario = dict()
```



Inicializar un diccionario

diccionario = { clave: valor, clave: valor, ... }

Código

```
diccionario = {
    "casa": "house",
    "coche": "car"
}
```

Código

```
clave = "casa"
valor = "house"
item = ("casa", "house")
```

Añadir un nuevo item

diccionario[clave]: valor

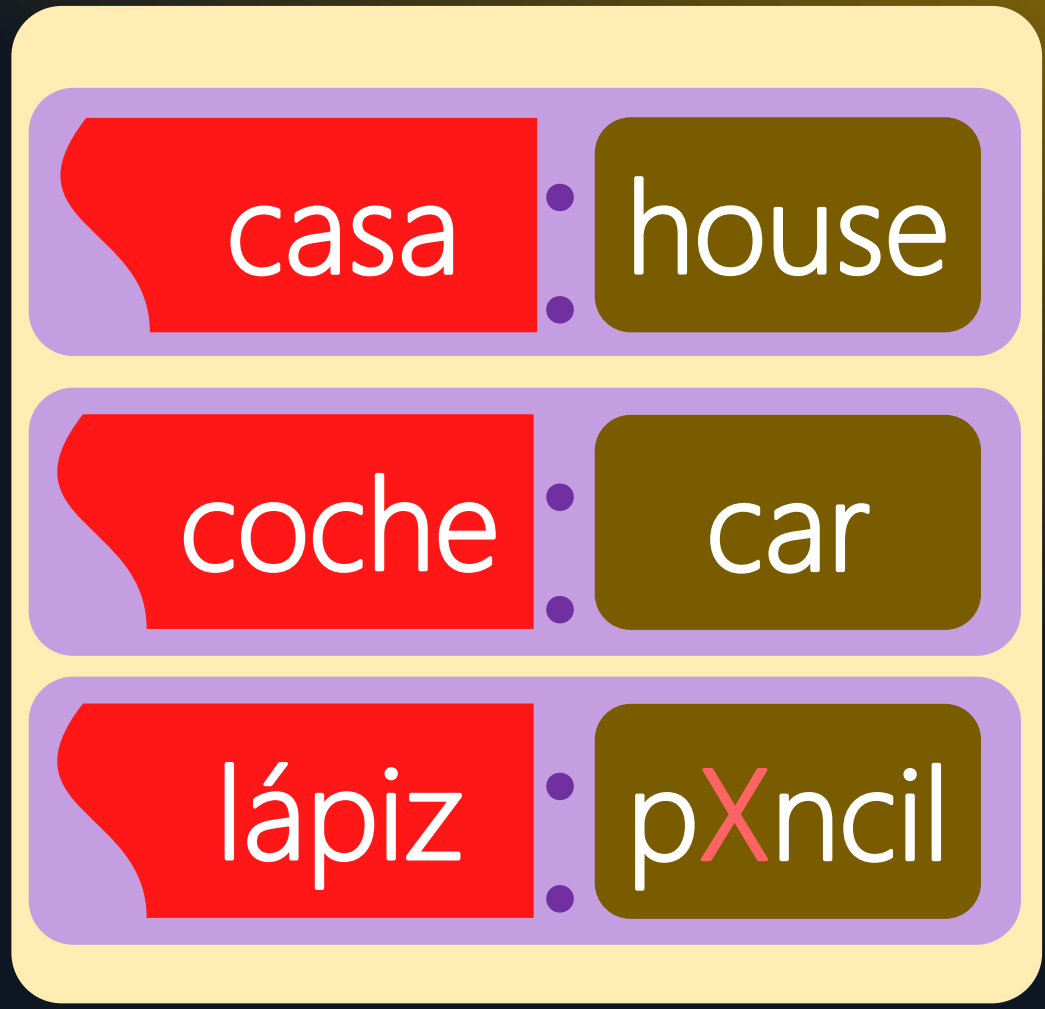
Código

```
diccionario["lápiz"] = "pXncil"
print(diccionario)
```

print

Consola

```
{
'casa': 'house',
'coche': 'car',
'lápiz': 'pXncil'
}
```



Modificar un ítem

diccionario[clave]: valor

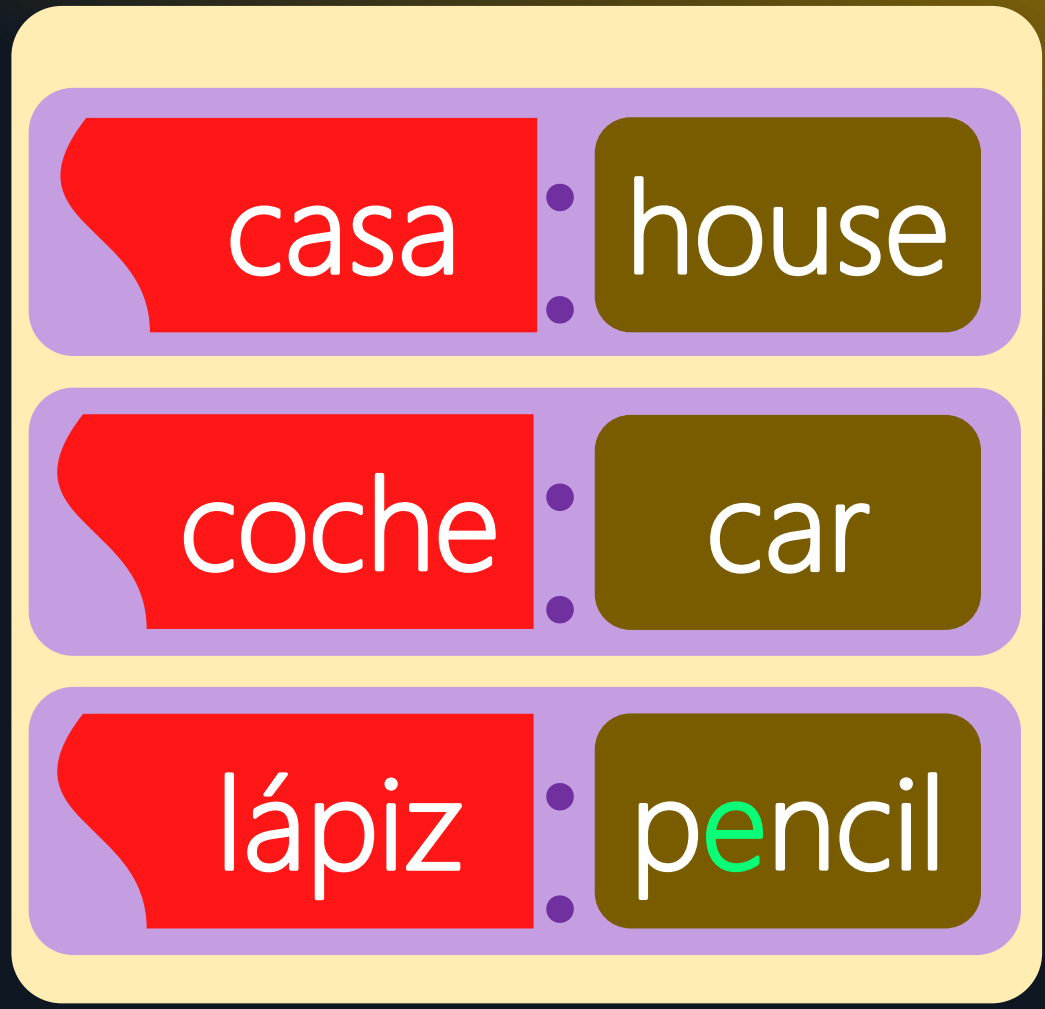
Código

```
diccionario["lápiz"] = "pencil"
diccionario
```

print

Consola

```
{
'casa': 'house',
'coche': 'car',
'lápiz': 'pencil'
}
```



Eliminar un ítem

del diccionario[clave]

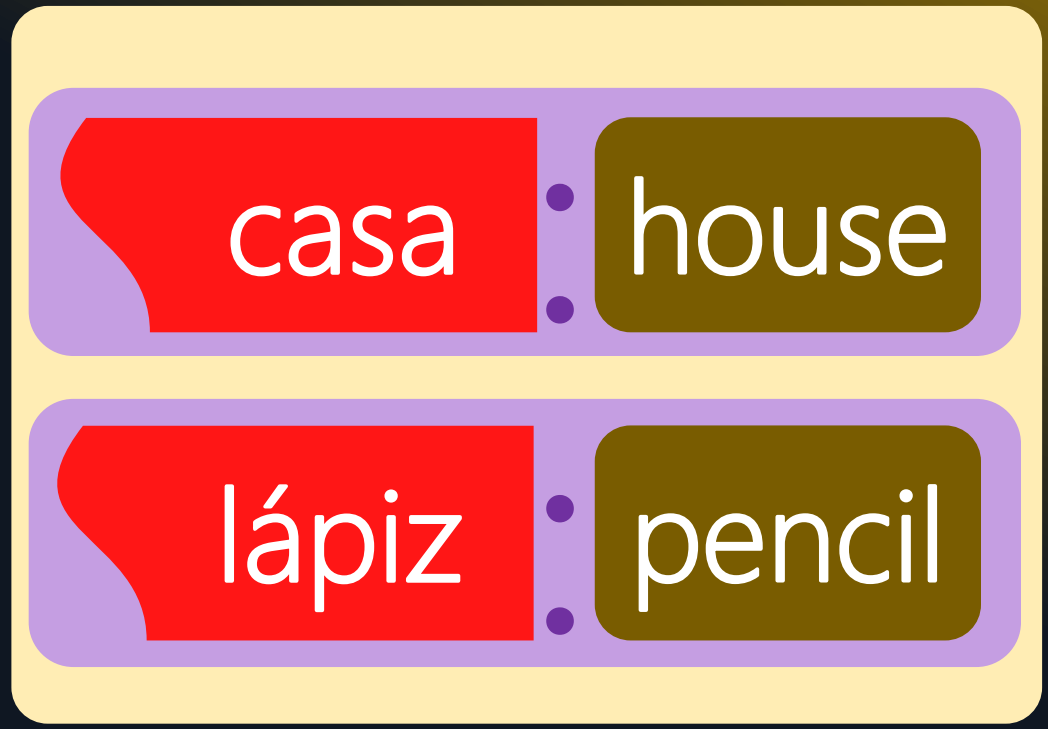
Código

```
del diccionario["coche"]
diccionario
```

print

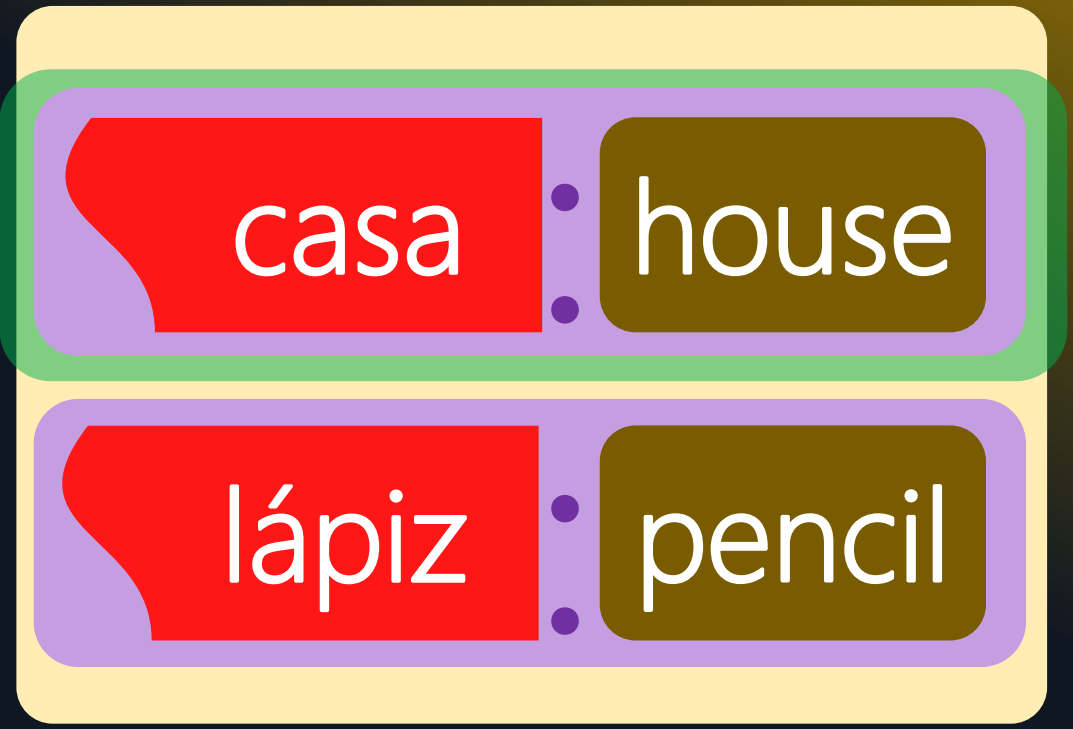
Consola

```
{
'casa': 'house',
'lápiz': 'pencil'
}
```



Seleccionar un valor existente

valor = diccionario[clave]



Código

```
casa_en_ingles = diccionario["casa"]
casa_en_ingles
```

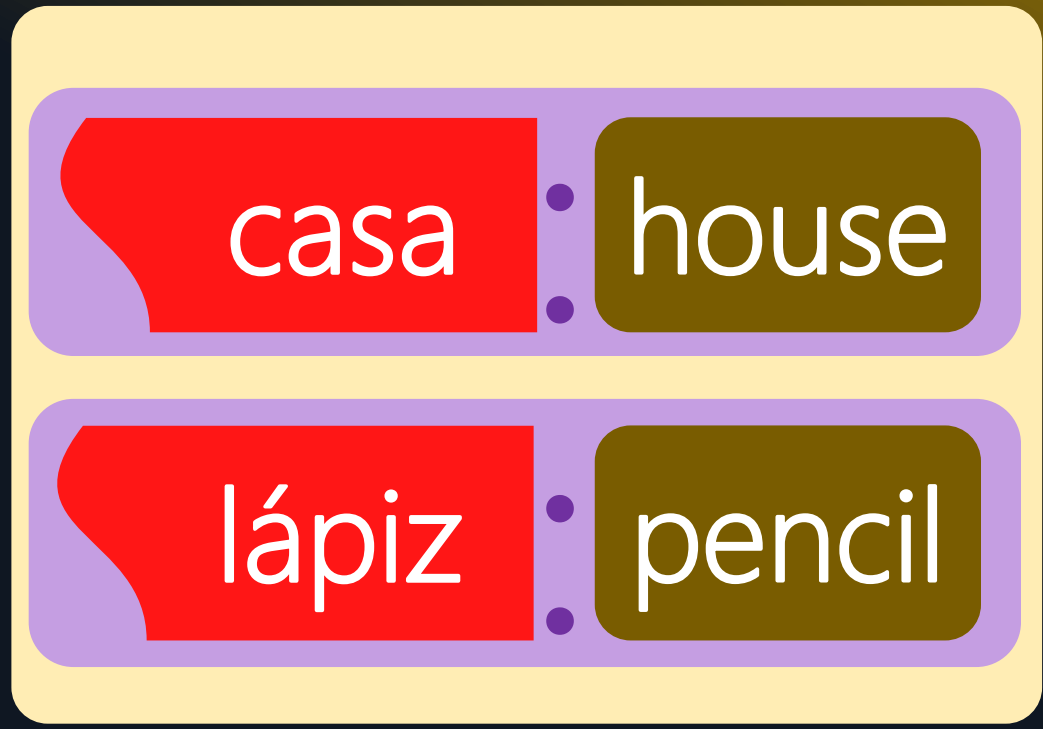
print

Consola

```
house
```

Seleccionar un valor NO existente

```
valor = diccionario[clave]
```



Código

```
valor_no_existe = diccionario["clave_no_existe"]
```

print

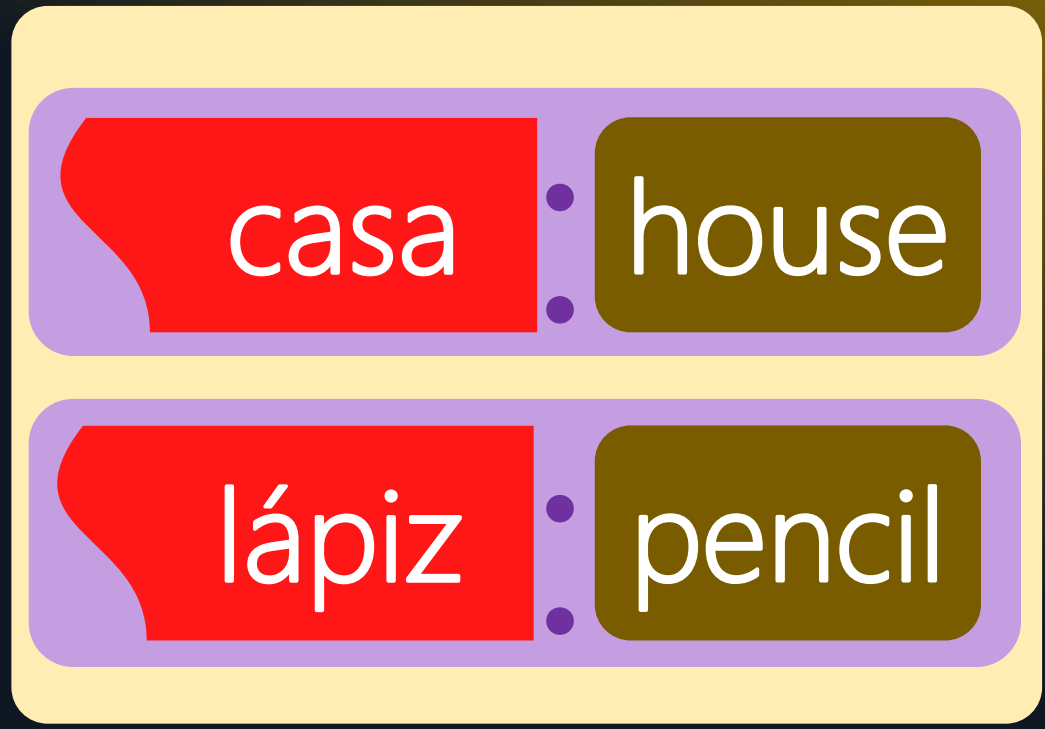
Consola

```
KeyError: 'clave_no_existe'
```

clave_no_existe

Preguntar si existe una clave

```
existe = clave in cdiccionario
```



Código

```
existe_la_clave = "casa" in diccionario
existe_la_clave
```

print

Consola

```
True
```

Longitud del diccionario

```
longitud = len(diccionario)
```

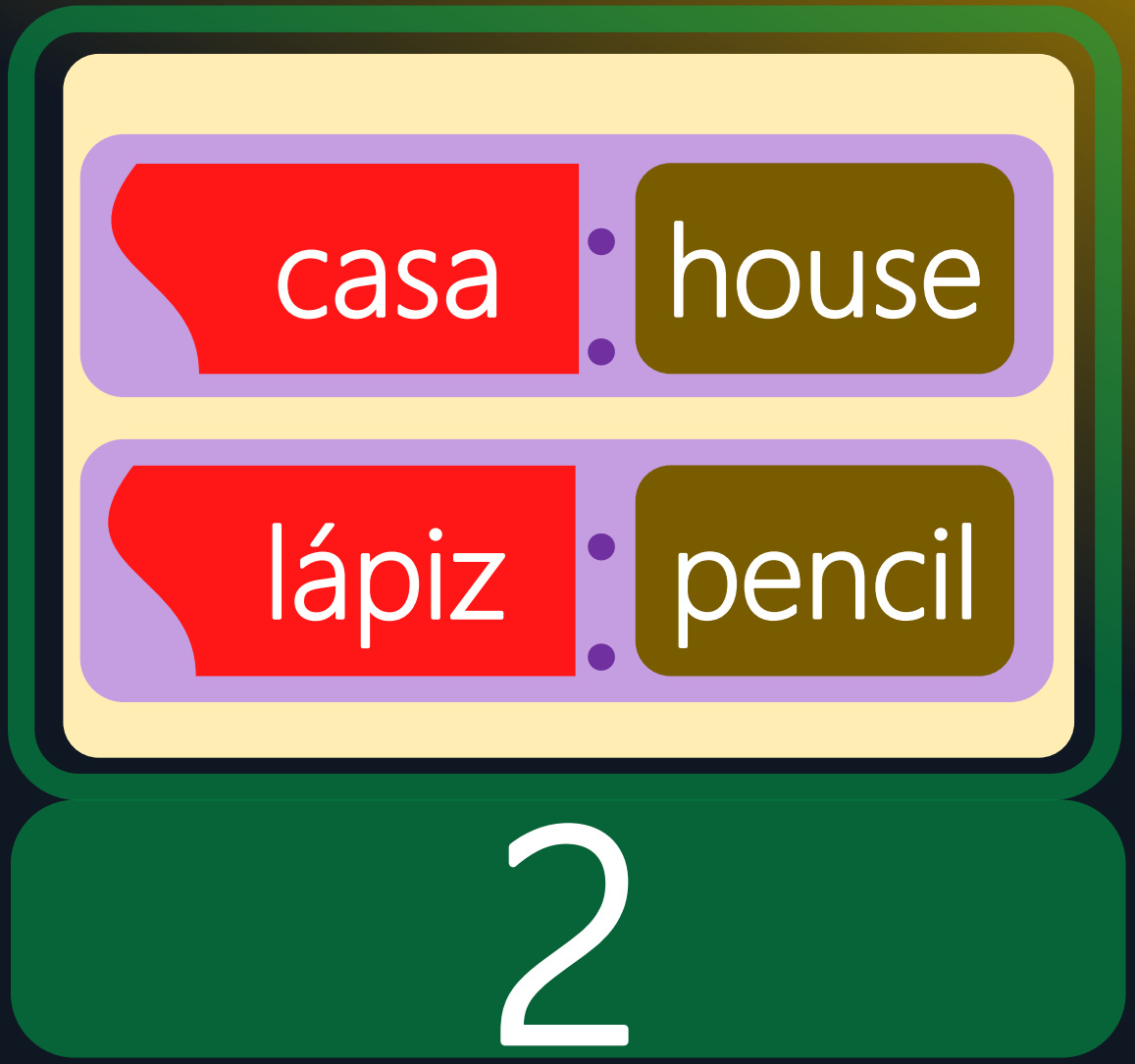
Código

```
longitud = len(diccionario)
longitud
```

print

Consola

```
2
```



Si modificamos algo de su diccionario asociado (algún valor o clave), se reflejará en el interior de las vistas

Vista de claves

vista_claves = diccionario.keys()

- KeysView: Vista de claves

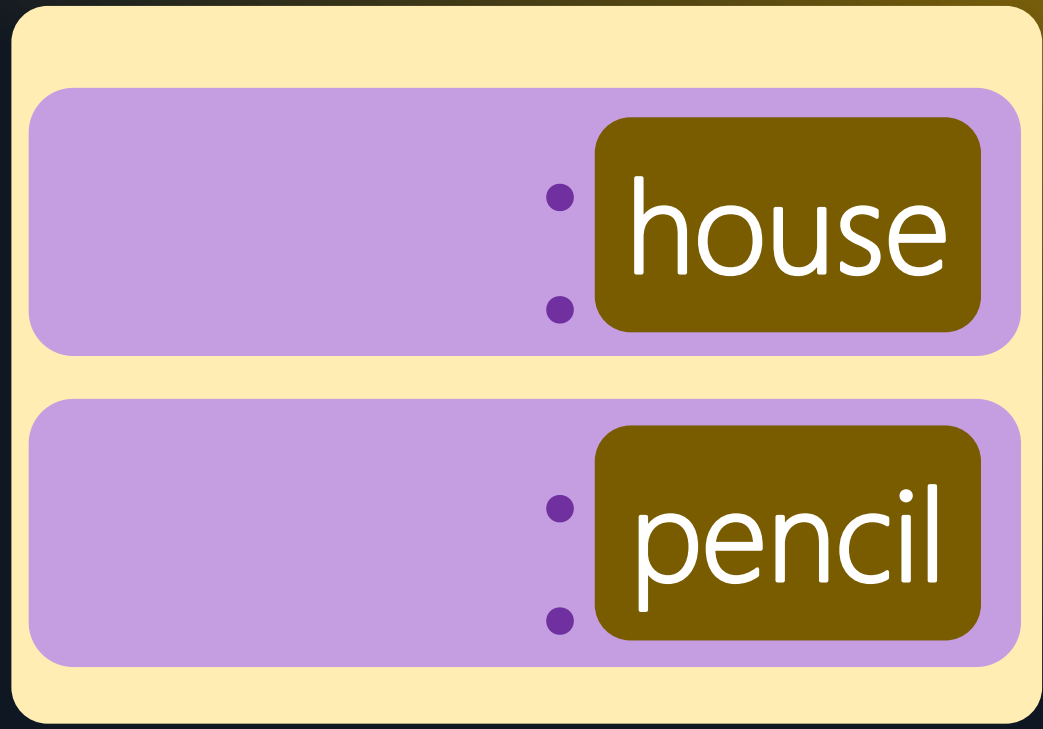
Código

```
for clave in diccionario.keys():
    print(clave)
```

print

Consola

```
casa
lápiz
```



Si modificamos algo de su diccionario asociado (algún valor o clave), se reflejará en el interior de las vistas

Vista de valores

vista_valores = diccionario.values()

- ValuesView: Vista de Valores

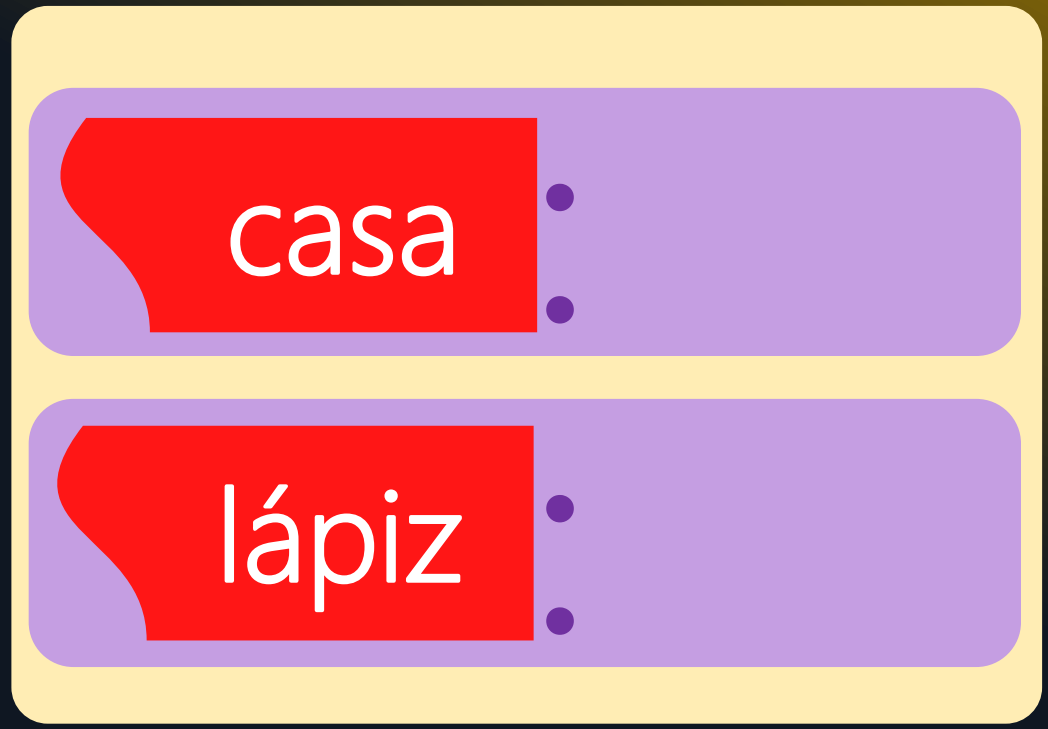
Código

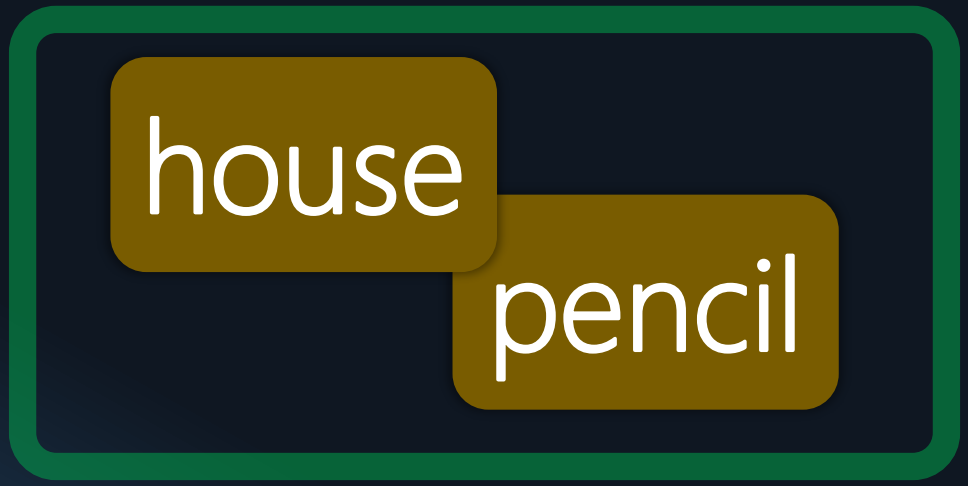
```
for valor in diccionario.values():
    print(valor)
```

print

Consola

```
house
pencil
```





Si modificamos algo de su diccionario asociado (algún valor o clave), se reflejará en el interior de las vistas

Vista de ítems

vista_items = diccionario.items()

- ItemsView: Vista de Item; tupla: (clave, valor)

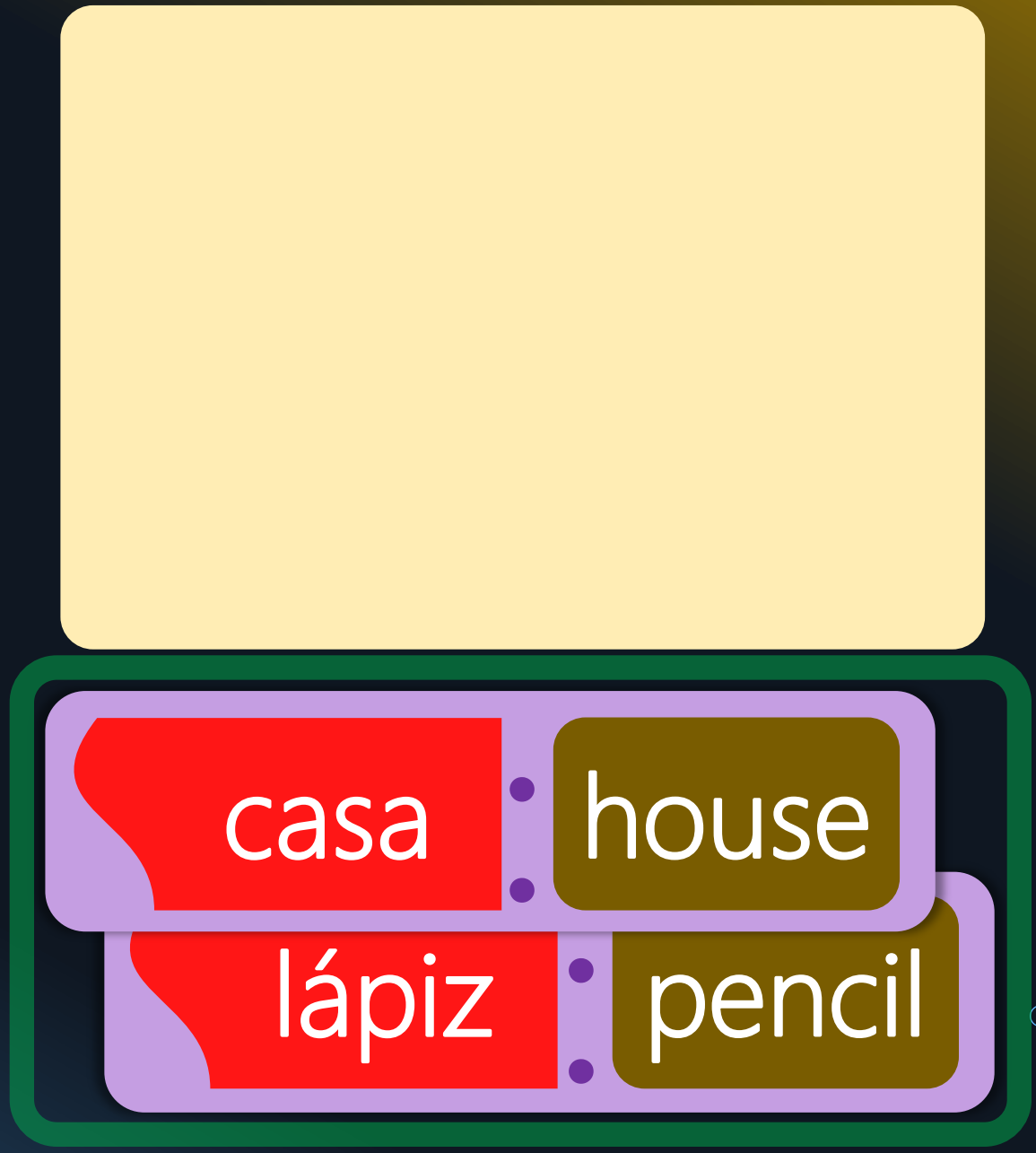
Código

```
for item in diccionario.items():
    print(item)
```

print

Consola

```
('casa', 'house')
('lápiz', 'pencil')
```



Iterar por las vistas

Con bucle "for"

- **Iterar por las claves;** sino se llama a ninguna vista y se usa el diccionario en un for, por defecto es keys()

Código

```
diccionario = {
    "casa": "house",
    "coche": "car",
    "lápiz": "pencil"
}
```

Código

```
for clave in diccionario.keys():
    print(clave)

for clave in diccionario:
    print(clave)
```

print

Consola

```
casa
coche
lápiz
```

- Iterar por los valores

Consola

```
house
car
pencil
```

print

Código

```
for valor in diccionario.values():
    print(valor)
```

- **Iterar por los ítems**; son tupla: (clave, valor). Los **ítems** al ser tuplas **se pueden extender** directamente al recorrer cada uno de los ítems en la cabecera del bucle **for**

Consola

```
('casa', 'house')
('coche', 'car')
('lápiz', 'pencil')
```

print

Código

```
for items in diccionario.items():
    print(items)

for clave, valor in diccionario.items():
    print(clave, valor)
```

Resumen de métodos de los diccionarios

Declaración de un diccionario
diccionario = dict()

Devuelve el numero de elementos que tiene el diccionario
len(dict)

Compara el número de elementos distintos que tienen los dos
cmp(dict1,dict2)

Devuelve una lista con las claves del diccionario
dict.keys()

Devuelve una lista con los valores del diccionario
dict.values()

Devuelve el valor del elemento con clave key. Sino devuelve default
dict.get(key, default=None)

Inserta un elemento en el diccionario clave:valor. Si la clave existe no lo inserta
dict.setdefault(key, default=None)

Insertamos un elemento en el diccionario con su clave:valor
dict['key'] = 'value'

Eliminamos el elemento del diccionario con clave key
dict.pop('key',None)

Devuleve la copia de un diccionario dict2 = dict.copy()
dict.copy()

Elimina todos los elementos de un diccionario
dict.clear()

Crea un nuevo diccionario poniendo como claves las que hay en la lista y los valores por defecto si se les pasa
dict.fromkeys(list, defaultValue)

Devuelve true si existe la clave. Sino devuelve false
key in dict

devuelve un lista de tuplas formadas por los pares clave:valor
dict.items()

Añade los elementos de un diccionario a otro
dict.update(dict2)

¡GRACIAS!

Web: **https://jarroba.com/**

Ramón Invarato Menéndez

Linked-in
https://www.linkedin.com/in/rinvarato/

Github
https://github.com/Invarato

Ricardo Moya García

Linked-in
https://www.linkedin.com/in/phdricardomoya

Github
https://github.com/RicardoMoya