IWI 131-PROGRAMACIÓN

Ayudantía 11

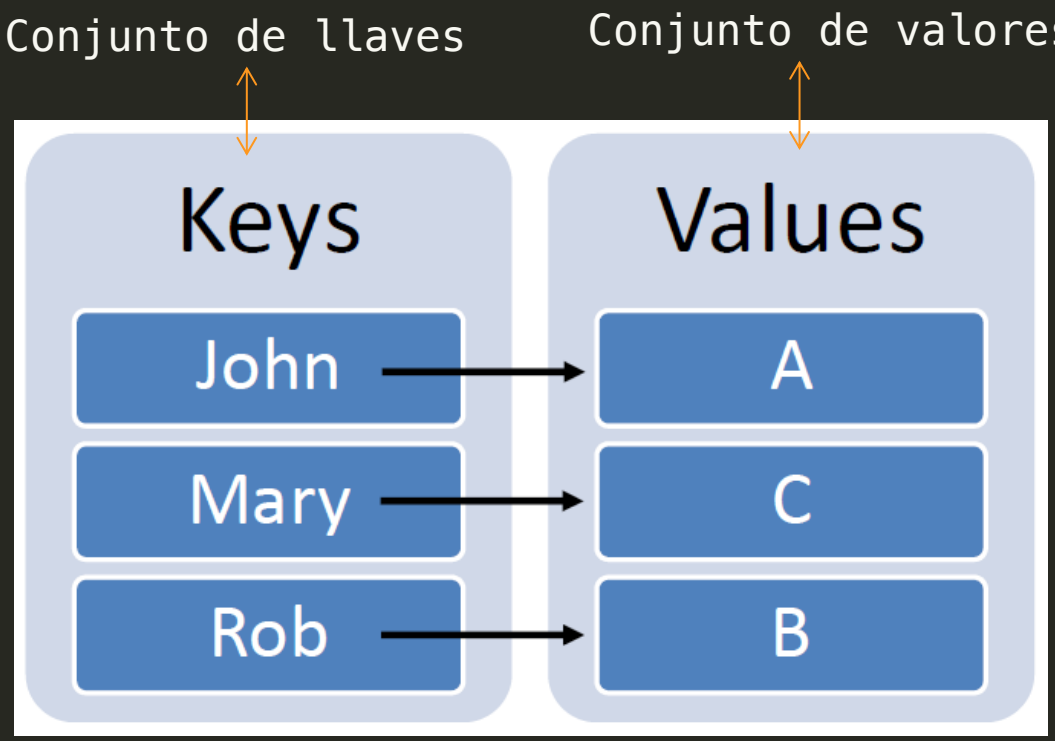
Ayudante: Anastasiia Fedorova

Paralelo: 212

Fecha: 1.07.2020

¿DICCIONARIOS?

Un diccionario es un tipo de dato, que asocia pares llave-valor (key-value). Las llaves no tienen orden algúno, y los valores entre sí se pueden repetir. La **única manera** de acceder a un valor es a través de su llave. **No** pueden haber llaves repetidas.




```
notas_alumnos =
 {'John': 'A', 'Mary': 'C', 'Rob': 'B'}
```

CREACIÓN

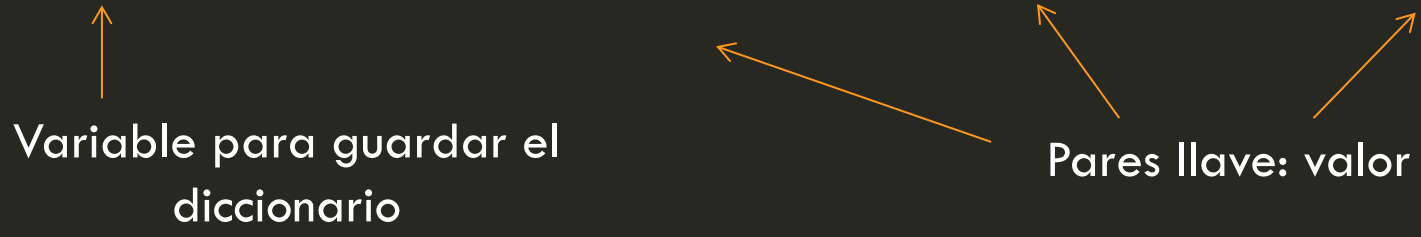
De nuevo, existen dos maneras de crear un diccionario vacío:

```
dicc = {}
```

```
dicc = dict()
```

Para crear un diccionario con valores explícitos, simplemente tenemos que poner los pares llave:valor dentro de las llaves:

```
notas_iwi = {'Leon': 55, 'Javier': 80, 'Maria': 93}
```



ACCESO

Como dijimos antes, en el diccionario podemos obtener el valor solo si conocemos la llave asociada. La sintaxis es muy parecida a la de acceso por índice en arreglos, solo que el ”índice” ahora es la llave:

```
notas_iwi = {“Leon”: 55, “Javier”: 80, “Maria”: 93}
print("Nota de Javier es:", notas_iwi[“Javier”])
>>> Nota de Javier es 80
```

ACCESO – KEYERROR

Si la llave no está presente en el diccionario, ocurre el KeyError – error de la llave:

```
>>> notas_iwi['Joaquin']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Joaquin'
```

AGREGAR UNA NUEVA LLAVE

Se puede agregar una llave nueva simplemente asignándole un valor. Recordar que las llaves no necesariamente mantienen el orden en que fueron ingresadas.

Si se asigna un valor a una llave que ya existe, el valor anterior **se sobreescribe**, pues un diccionario **no puede tener llaves repetidas.**

```
notas_iwi = {"Leon": 55, "Javier": 80, "Maria": 93}
notas_iwi["Joaquin"] = 78
notas_iwi["Leon"] = 67
print(notas_iwi)
>>> {'Leon': 67, 'Joaquin':78, 'Maria': 80, 'Javier': 80}
```

BORRAR LA LLAVE

Para borrar una llave del diccionario, podemos ocupar la sentencia del:

```
notas_iwi = {'Leon': 67, 'Joaquin':78, 'Maria': 80, 'Javier':
80}
del notas_iwi["Leon"]
notas_iwi = {'Joaquin':78, 'Maria': 80, 'Javier': 80}
```

LEN() Y IN

En diccionarios, `len(diccionario)` nos entrega la cantidad de pares llave-valor:

```
notas_iwi = {'Joaquin':78, 'Maria': 80, 'Javier': 80}
print(len(notas_iwi))
>>> 3
```

El `in` en diccionarios permite revisar si la llave está en este:

```
print('Pedro' in notas_iwi)
>>> False
```

RECORRER EL DICCIONARIO

Los diccionarios son iterables. Al iterar sobre un diccionario en un ciclo for, se obtienen las llaves:

```
notas_iwi = {'Joaquin':78, 'Maria': 80, 'Javier': 80}
```

```
lista_llaves = []
for k in notas_iwi:
    lista_llaves.append(k)
print(lista_llaves)
>>> ["Joaquin", "Maria", "Javier"]
```

La lista de mismas llaves se puede obtener con notas_iwi.keys()

```
lista_valores = []
for k in notas_iwi:
    lista_valores.append(notas_iwi[k])
print(lista_valores)
>>> [78, 80, 80]
```

La lista de los valores se puede obtener con notas_iwi.values()

RESTRICCIONES SOBRE LAS LLAVES

No cualquier cosa puede ser la llave. Las llaves deben ser de un tipo de dato inmutable.

Puede:

- Strings
- Tuplas
- Números (int & float)
- Booleanos

No puede:

- Listas
- Diccionarios
- Conjuntos

EJERCICIO 1

Sabemos que “**iwi**” tiene 2 caracteres únicos, mientras ”**pikachu**” tiene 7 caracteres únicos. Calcula el total de caracteres únicos de la palabra, y el total de repeticiones de cada carácter. La palabra será ingresada por el usuario. Utilice solamente un diccionario.

```
Ingrese su palabra: abracadabra
Total de caracteres unicos: 5
Total de repeticiones de a es 5
Total de repeticiones de b es 2
Total de repeticiones de r es 2
Total de repeticiones de c es 1
Total de repeticiones de d es 1
```

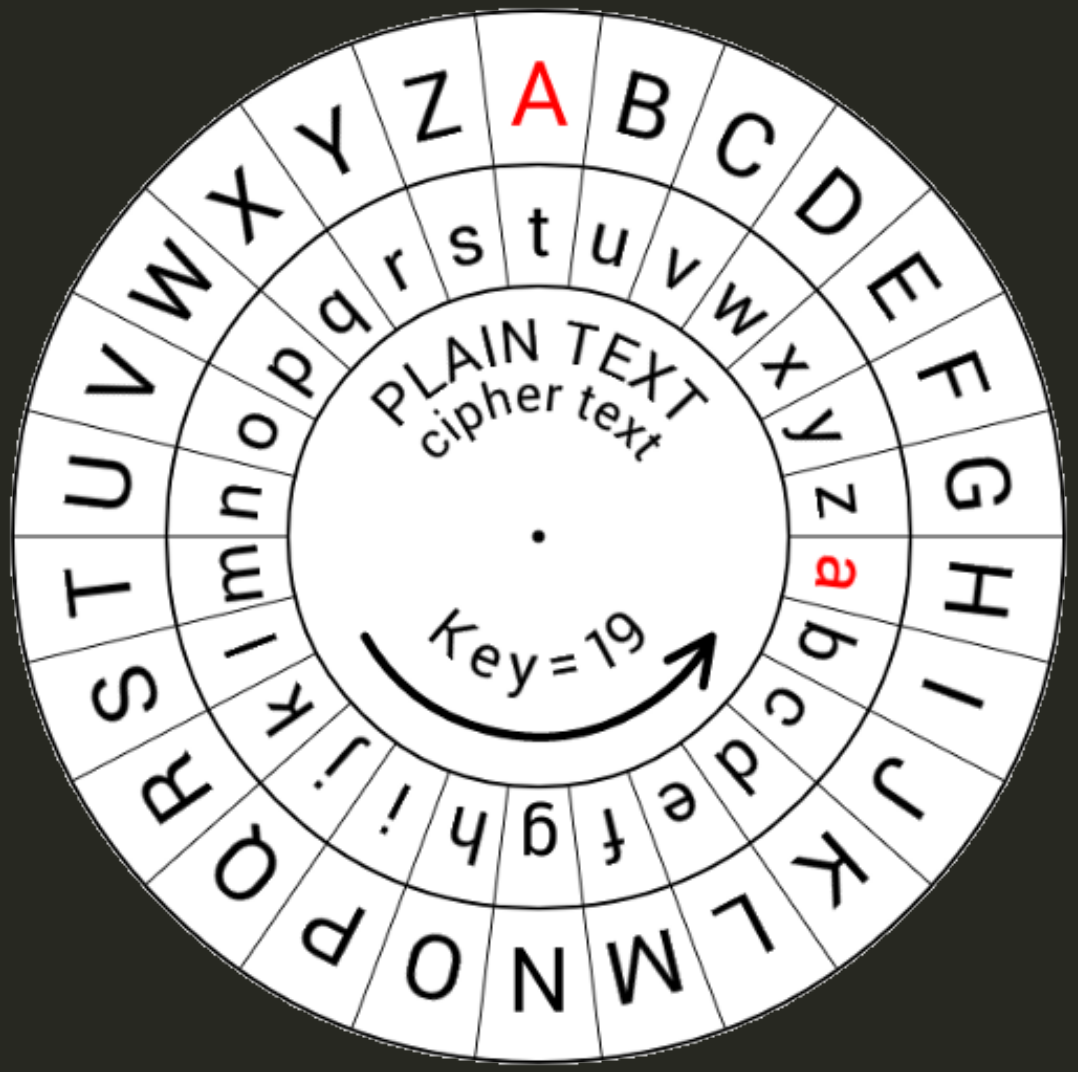
SOLUCIÓN

```
palabra = input("Ingrese su palabra: ")
d = {}
for letra in palabra:
    if letra not in d:
        d[letra] = 1
    else:
        d[letra] += 1
print("Total de caracteres unicos:", len(d))

for k in d:
    print("Total de repeticiones de", k, "es", d[k])
```

EJERCICIO 2.1

En criptografía, el Cifrado de Caesar es uno de los cifrados más simples y conocidos. Consiste en que una letra es sustituida por otra letra que está a **n** posiciones de la actual, de manera cíclica. Por ejemplo, ROT-3 corresponde al desplazamiento de cada letra en 3 posiciones. En este caso, la 'A' será sustituida por 'D', 'B' por 'E', etc.



EJERCICIO 2.1

En Python, ROT-13 puede ser representado por un diccionario. Crea una programa para descifrar el siguiente mensaje: **“Pvsenqb qr Pnrfne? Cersvreb yn rafnynqn qr Pnrfne!”**

```
caesar = {'a': 'n', 'b': 'o', 'c': 'p', 'd': 'q', 'g': 't', 'h': 'u',
          'i': 'v', 'j': 'w', 'm': 'z', 'n': 'a', 'o': 'b', 'p': 'c',
          's': 'f', 't': 'g', 'u': 'h', 'v': 'i', 'y': 'l', 'z': 'm',
          'A': 'N', 'B': 'O', 'E': 'R', 'F': 'S', 'G': 'T', 'H': 'U',
          'K': 'X', 'L': 'Y', 'M': 'Z', 'N': 'A', 'Q': 'D', 'R': 'E',
          'S': 'F', 'T': 'G', 'W': 'J', 'X': 'K', 'Y': 'L', 'Z': 'M',
          'e': 'r', 'f': 's', 'k': 'x', 'l': 'y', 'q': 'd', 'r': 'e',
          'w': 'j', 'x': 'k', 'C': 'P', 'D': 'Q', 'I': 'V', 'J': 'W',
          'O': 'B', 'P': 'C', 'U': 'H', 'V': 'I', 'W': 'J', 'X': 'K',
          'Y': 'L', 'Z': 'M'}
```

SOLUCIÓN

```
def ROT13(mensaje):
    decifrado = ""
    for letra in mensaje:
        if letra in caesar:
            decifrado += caesar[letra]
        else:
            decifrado += letra
    return decifrado


mensaje = "Pvsenqb qr Pnrfne? Cersvreb yn rafnynqn qr
Pnrfne!"
print(ROT13(cifrado))
```

EJERCICIO 2.2

Usando el mismo diccionario, ahora les pide crear la funcion para cifrar un mensaje cualquiera en ROT-13.

```
Hola!!!!
>>> Ubyn!!!!

IWI131
>>> VJV131

En un lugar de la Mancha, de cuyo nombre no quiero acordarme...
>>> Ra ha yhtne qr yn Znapun, qr phlb abzoer ab dhvreb npbeqnezr...
```

SOLUCIÓN

```
def cifrar(mensaje):
    cifrado = ""
    for letra in mensaje:
        if letra in caesar:
            for clave in caesar:
                letra_corrida = caesar[clave]
                if letra_corrida == letra:
                    cifrado += clave
        else:
            cifrado += letra
    return cifrado


a_cifrar = input("Ingrese el mensaje a cifrar: ")
print(cifrar(a_cifrar))
```

EJERCICIO 3

L@s sansan@s cada cierto tiempo compran un café para poder seguir estudiando, pero siempre eligen el lugar más cercano para efectuar la compra.

Se tiene un diccionario sansanos que asocia a cada nombre de alumn@ su ubicación en forma de una tupla (x,y). Además, se tiene un diccionario con tiendas, donde la llave es el nombre de lugar y el valor es una tupla, formada por otra tupla (x,y) que indica la posición, y una lista con productos vendidos. Crear un programa que recibe el nombre de alumn@ y que retorna la tienda más cercana donde se puede comprar un rico café.

```
sansanos = {"Ana": (3, 1), "Jose": (4, 5), "Gonzalo": (6, 1)...}

tiendas = {"Kiosko": ((3, 2), ["cafe", "jugos", "empanadas"]), "Food Truck":
((4, 1), ["hamburguesas"]), "Starbucks PUC": ((6, 4), ["cafe", "dulces"])...}
```

```
Ingrese el nombre de sansan@: Ana
Tienda mas cercana a sansan@ Ana es Kiosko
```

SOLUCIÓN

```
def distancia_euclidiana(punto1, punto2):
    x1, y1 = punto1
    x2, y2 = punto2
    return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5
```

SOLUCIÓN

```
nombre_sansano = input("Ingrese el nombre de sansan@: ")
if nombre_sansano in sansanos:
    pos_sansano = sansanos[nombre_sansano]

    mindist = float('inf')
    tienda_elegida = ""
    for nombre in tiendas:
        pos_tienda, productos = tiendas[nombre]
        if "cafe" in productos:
            dist_atienda = distancia_euclidiana(pos_sansano, pos_tienda)
            if dist_atienda < mindist:
                mindist = dist_atienda
                tienda_elegida = nombre

    print("Tienda mas cercana a sansan@", nombre_sansano, "es", tienda_elegida)
else:
    print("Nombre no encontrado!")
```