



## ALGORITMOS Y ESTRUCTURAS DE DATOS (TSDS)

ASIGNATURA:

ALGORITMOS Y ESTRUCTURAS DE DATOS

PROFESOR:

Ing. Lorena Chulde MSc.

PERÍODO ACADÉMICO:

2023-B

### TALLER - TAREA

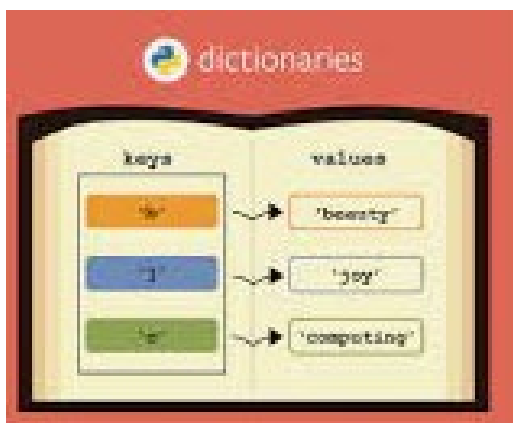
#### Grupal

### TÍTULO:

## DICCIONARIOS

#### Nombres de los estudiantes:

Guerra Lovato Josué Eduard  
Soria Ansa Richard Mauricio  
Pérez Orosco Carlos David



2023-B

# PROPÓSITO DE LA TAREA

Aplicar diccionario mediante funciones para implementar un CRUD

## OBJETIVO GENERAL

Desarrollar habilidades de programación en Python mediante la práctica de implementar algoritmos de ordenamiento como Quicksort y algoritmos de selección, así como también explorar y comprender el uso de estructuras de datos como diccionarios y conjuntos para resolver problemas de manipulación y gestión de datos.

## Parte I

Replicar los ejercicios del taller, capturar las pantallas y pegarlas en cada ejercicio

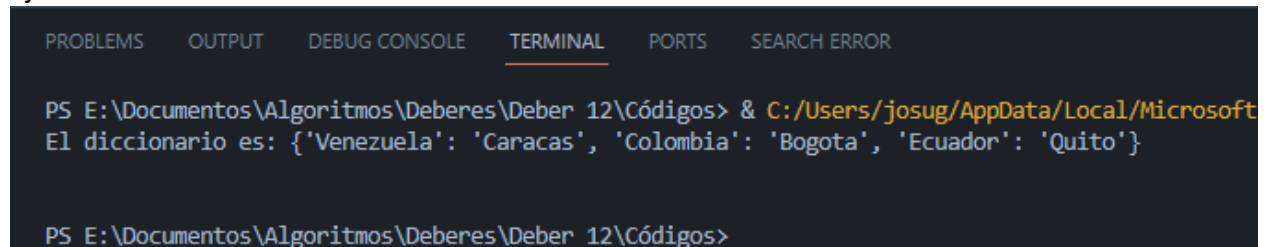
## GUIA DEL TALLER:

## EJERCICIOS CON DICCIONARIOS

### #Declarar el diccionario con clave y valor

```
diccionario = {"Venezuela": "Caracas", "Colombia": "Bogota", "Ecuador": "Quito"}  
print(diccionario)
```

Ejecución:

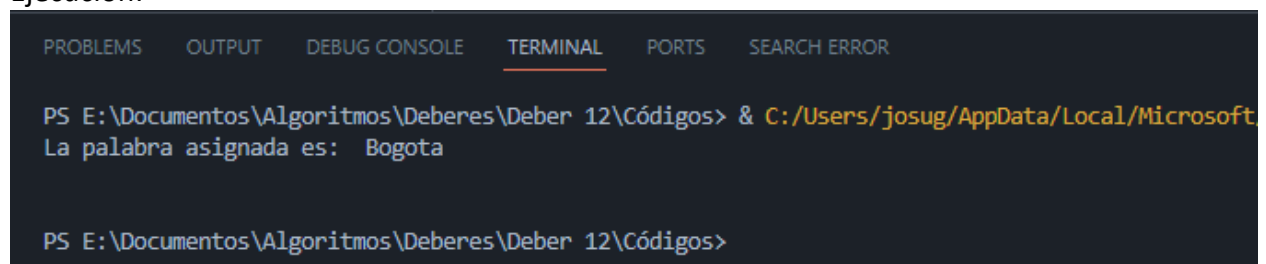


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR  
  
PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos> & C:/Users/josug/AppData/Local/Microsoft  
El diccionario es: {'Venezuela': 'Caracas', 'Colombia': 'Bogota', 'Ecuador': 'Quito'}  
  
PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos>
```

### #Acceder a un valor

```
diccionario = {"Venezuela": "Caracas", "Colombia": "Bogota", "Ecuador": "Quito"}  
print(diccionario["Colombia"])
```

Ejecución:

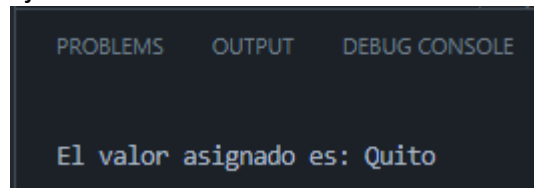


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR  
  
PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos> & C:/Users/josug/AppData/Local/Microsoft  
La palabra asignada es: Bogota  
  
PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos>
```

### #Acceder a un valor

```
valor = diccionario["Ecuador"]  
print(valor)
```

Ejecución:



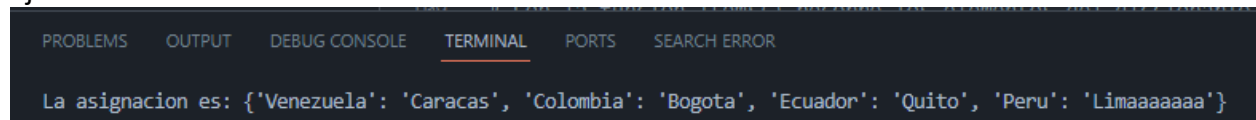
PROBLEMS OUTPUT DEBUG CONSOLE

El valor asignado es: Quito

### #Agregar un elemento

```
diccionario["Peru"] = "Limaaaaaaa"  
print(diccionario)
```

Ejecución:



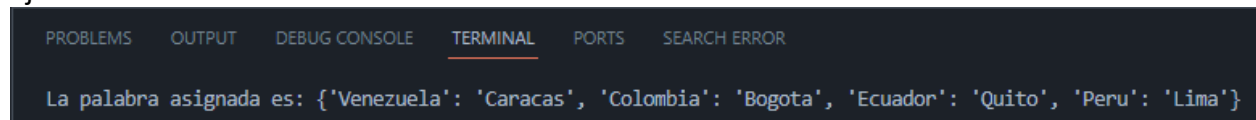
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR

La asignacion es: {'Venezuela': 'Caracas', 'Colombia': 'Bogota', 'Ecuador': 'Quito', 'Peru': 'Limaaaaaaa'}

### #Modificar un elemento

```
diccionario["Peru"] = "Lima"  
print(diccionario)
```

Ejecución:



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR

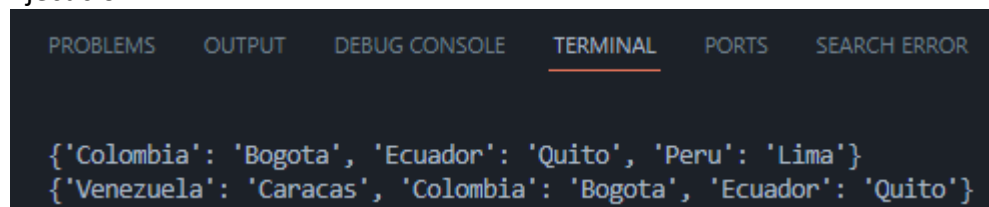
La palabra asignada es: {'Venezuela': 'Caracas', 'Colombia': 'Bogota', 'Ecuador': 'Quito', 'Peru': 'Lima'}

### #Eliminar un elemento con del

```
del diccionario["Venezuela"]  
print(diccionario)
```

```
diccionario2 = {"Venezuela": "Caracas", "Colombia": "Bogota", "Ecuador": "Quito",  
               "Peru": "Lima"}  
del(diccionario2["Peru"])  
print(diccionario2)
```

Ejecución:



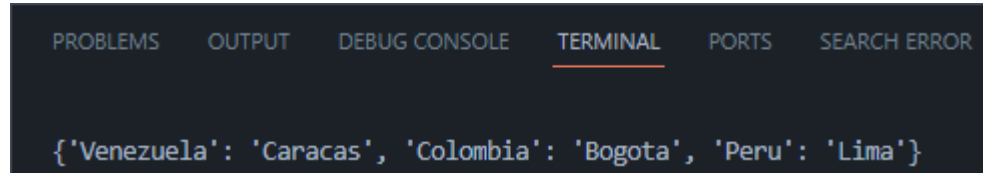
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR

{'Colombia': 'Bogota', 'Ecuador': 'Quito', 'Peru': 'Lima'}  
{'Venezuela': 'Caracas', 'Colombia': 'Bogota', 'Ecuador': 'Quito'}

### #Eliminar un elemento con la funcion pop

```
diccionario = {"Venezuela": "Caracas", "Colombia": "Bogota", "Ecuador": "Quito", "Peru": "Lima"}
diccionario.pop("Ecuador")
print(diccionario)
```

Ejecución:



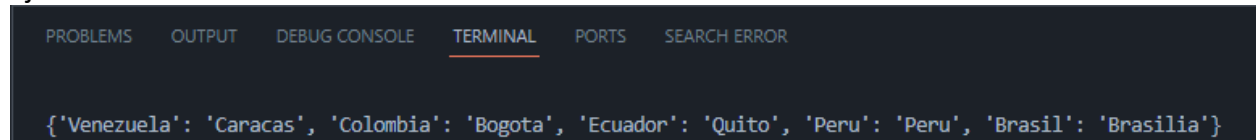
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

{'Venezuela': 'Caracas', 'Colombia': 'Bogota', 'Peru': 'Lima'}
```

### #con tuplas

```
tupla = ["Venezuela", "Colombia", "Ecuador", "Peru", "Brasil"]
dicPaises = {tupla[0]: "Caracas", tupla[1]: "Bogota", tupla[2]: "Quito", tupla[3]: "Peru",
tupla[4]: "Brasilia" }
print(dicPaises)
```

Ejecución:



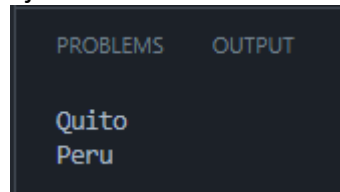
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

{'Venezuela': 'Caracas', 'Colombia': 'Bogota', 'Ecuador': 'Quito', 'Peru': 'Peru', 'Brasil': 'Brasilia'}
```

### #Acceder un elemento en concreto

```
print(dicPaises[tupla[2]]) # por el indice de la tupla
print(dicPaises["Peru"]) # por el valor de la tupla
```

Ejecución:



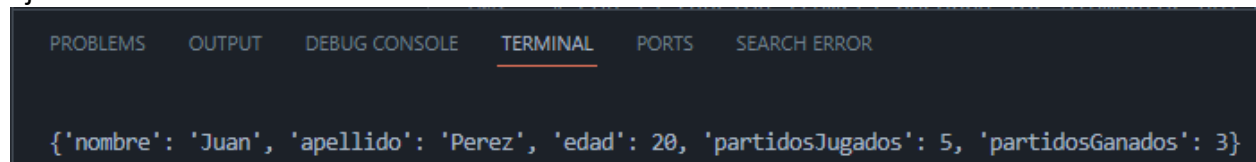
```
PROBLEMS  OUTPUT  TERMINAL

Quito
Peru
```

### #Crear diccionarios con diferentes tipo de datos

```
jugadores =
{"nombre": "Juan", "apellido": "Perez", "edad": 20, "partidosJugados": 5, "partidosGanados": 3}
print(jugadores)
```

Ejecución:



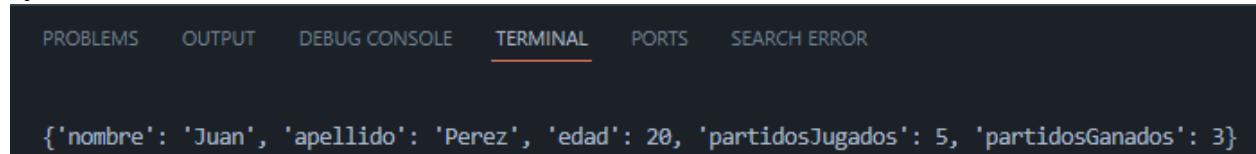
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

{'nombre': 'Juan', 'apellido': 'Perez', 'edad': 20, 'partidosJugados': 5, 'partidosGanados': 3}
```

### #con valores de tipo tupla

```
jugador =  
{ "nombre": "Juan", "apellido": "Perez", "edad": 25, "partidosJugados": 5, "partidosGanados": 3,  
  "detalleAnios": [2020, 2021, 2022, 2023, 2024] }  
print("Los partidos se jugaron en los años: ", jugador["detalleAnios"])
```

Ejecución:

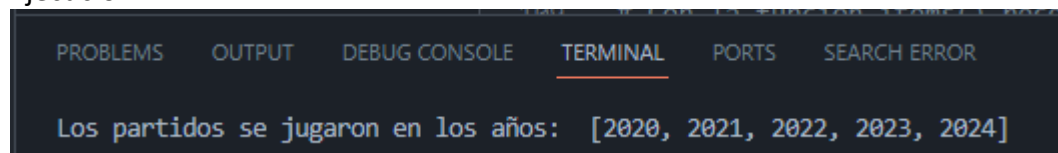


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  
  
{ 'nombre': 'Juan', 'apellido': 'Perez', 'edad': 20, 'partidosJugados': 5, 'partidosGanados': 3 }
```

### #un diccionario puede contener otro diccionario

```
jugador =  
{ "nombre": "Juan", "apellido": "Perez", "edad": 25, "partidosJugados": 5, "partidosGanados": 3,  
  "detalleAnios": { "temporadas": [2020, 2021, 2022, 2023, 2024] } }  
print("Los partidos se jugaron en los años: ", jugador["detalleAnios"])
```

Ejecución:

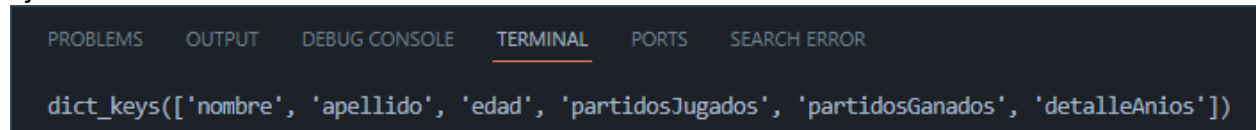


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  
  
Los partidos se jugaron en los años:  [2020, 2021, 2022, 2023, 2024]
```

### #Consultar las claves del diccionario

```
print(jugador.keys())
```

Ejecución:

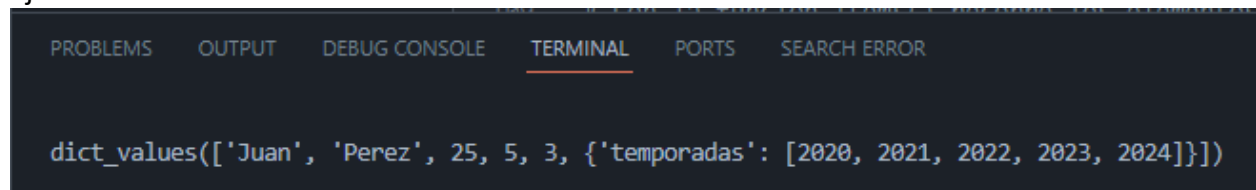


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  
  
dict_keys(['nombre', 'apellido', 'edad', 'partidosJugados', 'partidosGanados', 'detalleAnios'])
```

### #Consultar los valores del diccionario

```
print(jugador.values())
```

Ejecución:

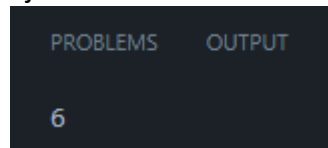


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  
  
dict_values(['Juan', 'Perez', 25, 5, 3, {'temporadas': [2020, 2021, 2022, 2023, 2024]}])
```

#Consultar la longitud del diccionario

```
print(len(jugador))
```

Ejecución:



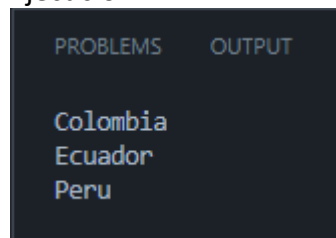
```
PROBLEMS OUTPUT
```

```
6
```

#Recorrer el diccionario con for e imprimir claves

```
diccionario_paises = {"Venezuela": "Caracas", "Colombia": "Bogota", "Ecuador": "Quito",  
"Peru": "Lima"}  
for pais in diccionario_paises:  
    print(pais)
```

Ejecución:



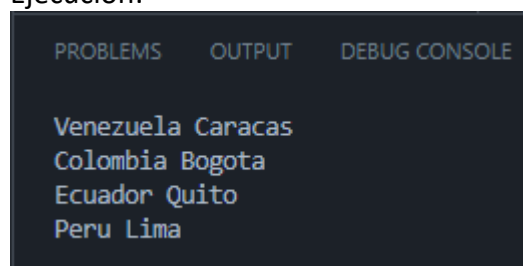
```
PROBLEMS OUTPUT
```

```
Colombia  
Ecuador  
Peru
```

#Recorrer el diccionario con for e imprimir clave y valor

```
diccionario_paises = {"Venezuela": "Caracas", "Colombia": "Bogota", "Ecuador": "Quito",  
"Peru": "Lima"}  
  
for clave, valor in diccionario_paises.items():  
    print(clave, valor)
```

Ejecución:



```
PROBLEMS OUTPUT DEBUG CONSOLE
```

```
Venezuela Caracas  
Colombia Bogota  
Ecuador Quito  
Peru Lima
```

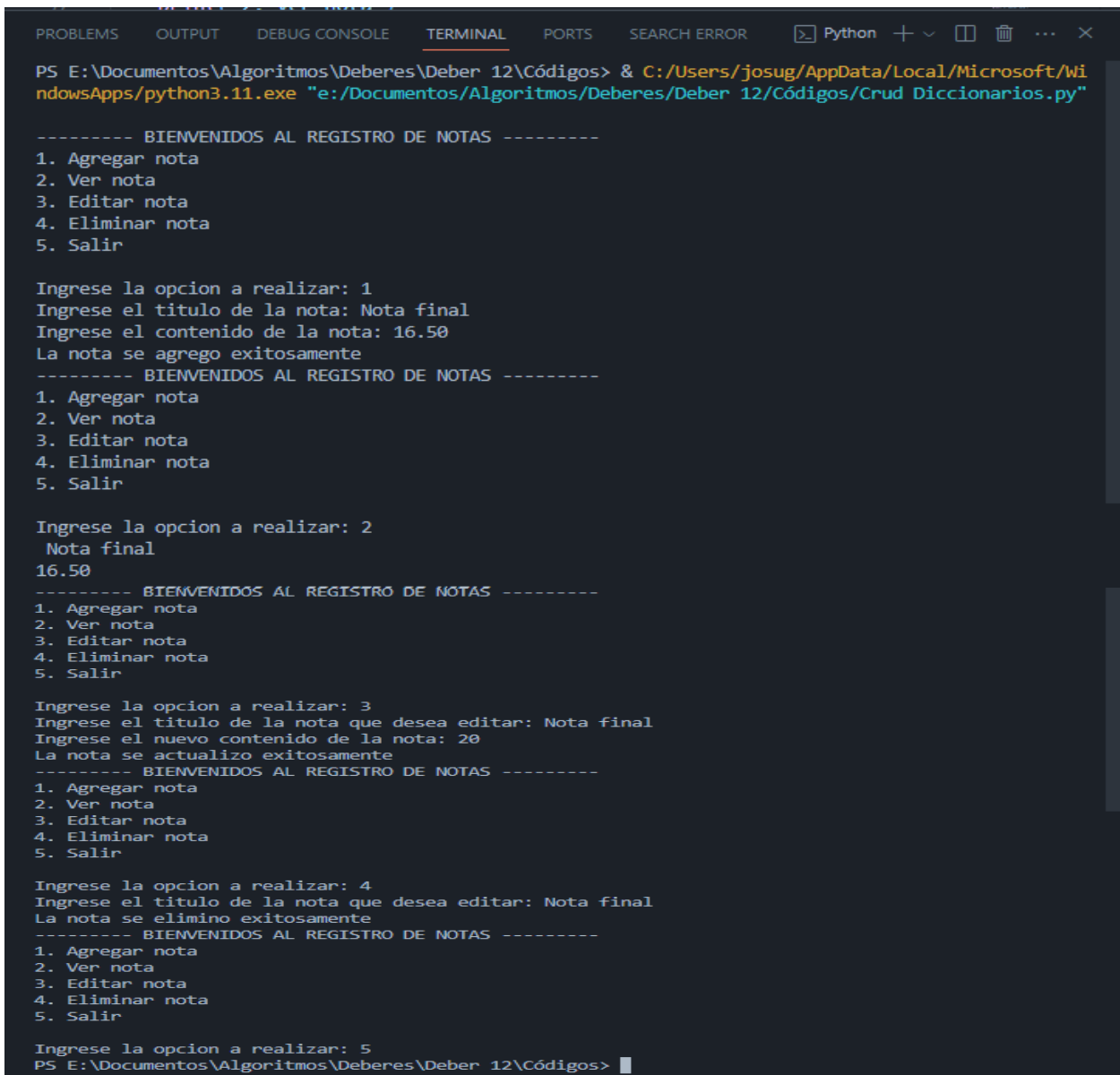
## CRUD CON DICCIONARIOS

#Diccionario que permite gestionar notas

```
notas = {}  
def agregarNota(notas, titulo, contenido):  
    notas[titulo] = contenido  
  
def verNotas(notas):  
    if not notas:  
        print("No existe ninguna nota ")  
    else:  
        for titulo, contenido in notas.items():  
            print(f" {titulo} ")  
            print(contenido)  
  
def editarNota(notas, titulo, nuevoContenido):  
    if titulo in notas:  
        notas[titulo] = nuevoContenido  
        print("La nota se actualizo exitosamente")  
    else:  
        print("La nota consultada no se encuentra en el registro")  
def eliminarNota(notas, titulo):  
    if titulo in notas:  
        del notas[titulo]  
        print("La nota se elimino exitosamente")  
  
    else:  
        print("La nota que desea eliminar no se encuentra en el registro")  
  
#MENU  
while True:  
    print("----- BIENVENIDOS AL REGISTRO DE NOTAS -----")  
    print("1. Agregar nota")  
    print("2. Ver nota")  
    print("3. Editar nota")  
    print("4. Eliminar nota")  
    print("5. Salir \n")  
  
    opcion = int(input("Ingrese la opcion a realizar: "))  
  
    if opcion == 1:  
        titulo = input("Ingrese el titulo de la nota: ")  
        contenido = input("Ingrese el contenido de la nota: ")  
        agregarNota(notas, titulo, contenido)  
        print("La nota se agrego exitosamente")
```

```
elif opcion == 2:
    verNotas(notas)
elif opcion == 3:
    titulo = input("Ingrese el titulo de la nota que desea editar: ")
    nuevoContenido = input("Ingrese el nuevo contenido de la nota: ")
    editarNota(notas, titulo, nuevoContenido)
elif opcion == 4:
    titulo = input("Ingrese el titulo de la nota que desea editar: ")
    eliminarNota(notas, titulo)
elif opcion == 5:
    break
else:
    print("La opcion ingresada no es valida")
```

Ejecución del programa:



```
PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos> & C:/Users/josug/AppData/Local/Microsoft/WindowsApps/python3.11.exe "e:/Documentos/Algoritmos/Deberes/Deber 12/Códigos/Crud Diccionarios.py"

----- BIENVENIDOS AL REGISTRO DE NOTAS -----
1. Agregar nota
2. Ver nota
3. Editar nota
4. Eliminar nota
5. Salir

Ingrese la opcion a realizar: 1
Ingrese el titulo de la nota: Nota final
Ingrese el contenido de la nota: 16.50
La nota se agrego exitosamente
----- BIENVENIDOS AL REGISTRO DE NOTAS -----
1. Agregar nota
2. Ver nota
3. Editar nota
4. Eliminar nota
5. Salir

Ingrese la opcion a realizar: 2
Nota final
16.50
----- BIENVENIDOS AL REGISTRO DE NOTAS -----
1. Agregar nota
2. Ver nota
3. Editar nota
4. Eliminar nota
5. Salir

Ingrese la opcion a realizar: 3
Ingrese el titulo de la nota que desea editar: Nota final
Ingrese el nuevo contenido de la nota: 20
La nota se actualizo exitosamente
----- BIENVENIDOS AL REGISTRO DE NOTAS -----
1. Agregar nota
2. Ver nota
3. Editar nota
4. Eliminar nota
5. Salir

Ingrese la opcion a realizar: 4
Ingrese el titulo de la nota que desea editar: Nota final
La nota se elimino exitosamente
----- BIENVENIDOS AL REGISTRO DE NOTAS -----
1. Agregar nota
2. Ver nota
3. Editar nota
4. Eliminar nota
5. Salir

Ingrese la opcion a realizar: 5
PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos>
```



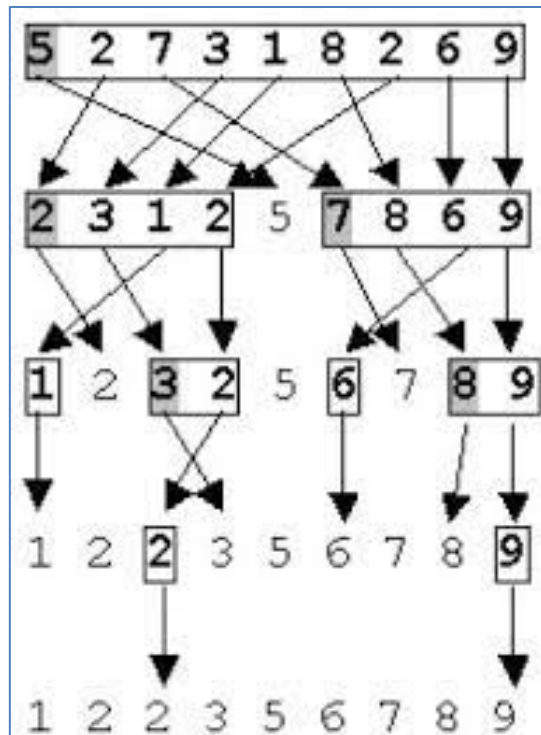
## Algoritmo de ordenamiento Quicksort

En qué consiste el Algoritmo de ordenamiento Quicksort:

El método Quick Sort es actualmente el más eficiente y veloz de los métodos de ordenación interna. Es también conocido con el nombre del método rápido y de ordenamiento por partición.

Para implementar este algoritmo se usan listas auxiliares vamos a aplicar la estrategia divide y vencerás sobre la lista para ir dividiendo la lista en partes más pequeñas y así tener que llevar a cabo menos comparaciones. Para esto se elige un elemento de la lista, a partir de este elemento se va a dividir la lista en los mayores y los menores.

El elemento se llama pivote y va el primer elemento de la lista. Se recorre el resto de los elementos para compararlos con el pivote si son mayores se los ubica a la derecha si son menores se los ubica a la izquierda.



### Ejemplo:

Tenemos la siguiente lista con valores desordenados:

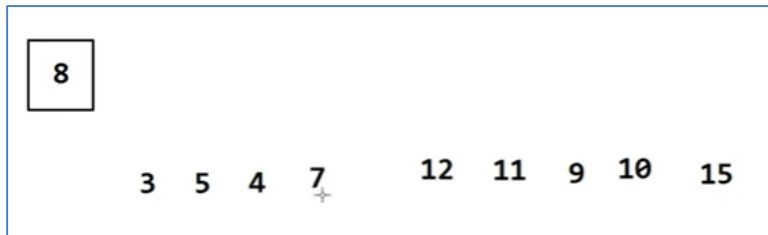
`valores = [8, 12, 3, 11, 5, 9, 10, 4, 15, 7]`

Elemento pivote el 8, tomamos el primer elemento:

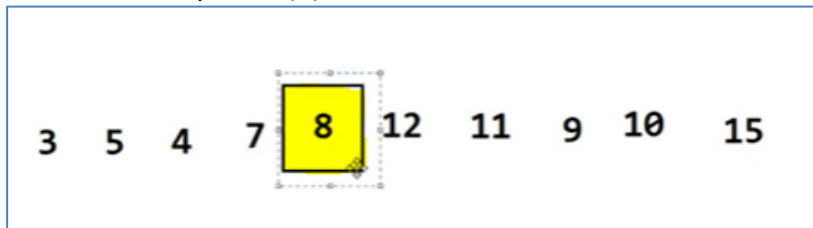
`valores = [8, 12, 3, 11, 5, 9, 10, 4, 15, 7]`

Ahora tomamos los elementos y los comparamos con el pivote (8):

- El 12 como es mayor lo ponemos a la derecha.
- El 3 es menor va a la izquierda
- El 11 es mayor va a la derecha
- El 5 es menor va a la izquierda
- El 9 es mayor va a la derecha
- El 4 es menor va a la izquierda
- El 15 es mayor va a la derecha
- El 7 es menor va a la izquierda



Finalmente, el pivote (8) lo ubicamos en el centro



Queda hacerle recursivo puesto que las instrucciones se han ejecutado una vez obteniendo el resultado de la figura anterior.

Al hacerle recursiva se ejecuta la misma operación con las dos partes de la lista hasta llegar al caso base.

### Caso Base:

Que la lista tenga un solo elemento con lo cual ya estará colocado o con una lista que no tenga elementos.

### Ejemplo:

Crear una función auxiliar dividir para particionar la lista; toma como parámetro la lista y que va a dividir la lista en dos partes a partir de un pivote que se tome en este caso es el primer elemento de la lista.

```
lista = [8, 12, 3, 11, 5, 9, 10, 4, 15, 7]
```

```
pivote = lista[0]
```

Vamos a necesitar dos listas auxiliares donde guardar los elementos menores y los elementos mayores. Declaramos dos listas vacías una para menores y otra para mayores.

```
lista = [8, 12, 3, 11, 5, 9, 10, 4, 15, 7]
```

```
def dividir(lista):  
    pivote = lista[0]  
    menores = []  
    mayores = []
```

Recorremos el resto de los elementos de la lista con for, desde el 1 hasta la longitud de la lista, puesto que el primero ya no se lo recorre porque es el pivote.

Comparamos si el elemento con el que vamos iterando es menor que el pivote entonces a la lista menor se le añade el elemento caso contrario se le añade a la lista mayores o si es igual también se lo incluye en mayores. Retorna la lista de elementos menores, mayores y el pivote.

```
lista = [8, 12, 3, 11, 5, 9, 10, 4, 15, 7]
```

```
def dividir(lista):  
    pivote = lista[0]  
    menores = []  
    mayores = []  
  
    for i in range(1, len(lista)):  
        if lista[i] < pivote:  
            menores.append(lista[i])  
        else:  
            mayores.append(lista[i])  
  
    return menores, pivote, mayores
```

Mediante esta función hemos dividido la lista en dos partes a partir del pivote de tal forma que los menores ya no hay que compararlos con los mayores estamos aplicando la técnica divide y vencerás.

Hemos dividido la lista en dos partes de tal forma que los menores ya no hay que compararlos con los mayores, ni los mayores con los menores, por lo tanto reducimos en una parte considerable las comparaciones.

A la lista menores volvemos a llevar a realizar la misma operación, la volvemos a dividir en dos partes volviéndose a reducir de nuevo las comparaciones. Una vez que hacemos esto se trata de aplicar la función de forma recursiva a las listas auxiliares.

Definimos la función “quicksort” que va a tomar la lista y si la longitud es menor que dos, es decir si tiene solo un elemento o 0 elementos llegamos al caso base y retorna la lista, caso contrario llamamos a la función “dividir” para dividir la lista en dos listas más pequeñas y un pivote, así que menores pivote y mayores va a ser igual a dividir(lista). Esta función devuelve la concatenación de estos tres elementos, por lo que ya se tiene la lista ordenada de tres elementos.

```
def quicksort(lista):  
    if len(lista) < 2:  
        return lista  
    menores, pivote, mayores = dividir(lista)  
    return menores + [pivote] + mayores
```

Salida:

```
lista = [8, 12, 3]
```

```
[3, 8, 12]
```

Pero si la lista tiene más elementos como la del ejemplo.

```
lista = [8, 12, 3, 11, 5, 9, 10, 4, 15, 7]
```

```
[3, 5, 4, 7, 8, 12, 11, 9, 10, 15]
```

Lo que hace es una pasada por eso se queda en el estado de la figura, el pivote 8 se coloca al centro y los menores a la izquierda y lo mayores a la derecha.

Para que recorra hasta el final de los elementos, debemos llamar a la función Quicksort en la lista menores y la lista mayores.

```
def quicksort(lista):  
    if len(lista) < 2:  
        return lista #devuelve los elementos ordenados porque solo tiene 1 o ninguno  
    menores, pivote, mayores = dividir(lista)  
    return quicksort(menores) + [pivote] + quicksort(mayores)
```

Ahora si se ordenan todos los elementos:

```
[3, 4, 5, 7, 8, 9, 10, 11, 12, 15]
```

**Código completo:**

```
lista = [8, 12, 3, 11, 5, 9, 10, 4, 15, 7]
```

```
#lista = [8, 12,3]
```

```
def dividir(lista):
```

```
    pivote = lista[0]
```

```
    menores = []
```

```
    mayores = []
```

```
    for i in range(1, len(lista)):
```

```
        if lista[i] < pivote:
```

```
            menores.append(lista[i])
```

```
        else:
```

```
            mayores.append(lista[i])
```

```
    return menores, pivote, mayores
```

```
def quicksort(lista):
```

```
    if len(lista) < 2:
```

```
        return lista #devuelve los elementos ordenados porque solo tiene 1 o ninguno
```

```
    menores, pivote, mayores = dividir(lista)
```

```
    return quicksort(menores) + [pivote] + quicksort(mayores)
```

```
print(quicksort(lista))
```

**Ejecución:**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos> & C:/Users/josug/AppData/Local/M
[3, 4, 5, 7, 8, 9, 10, 11, 12, 15]
PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos>
```

## Parte II

### TAREA:

Consultar sobre conjuntos

#### CONJUNTOS EN PYTHON

Es una colección desordenada de elementos únicos. Los conjuntos son útiles cuando se necesita almacenar una colección de elementos sin duplicados sin importar el orden en que se almacenan los elementos.

Los conjuntos en Python se definen utilizando llaves {} y los elementos se separan por comas. Por ejemplo:

1. `mi_conjunto = {1, 2, 3, 4, 5}`

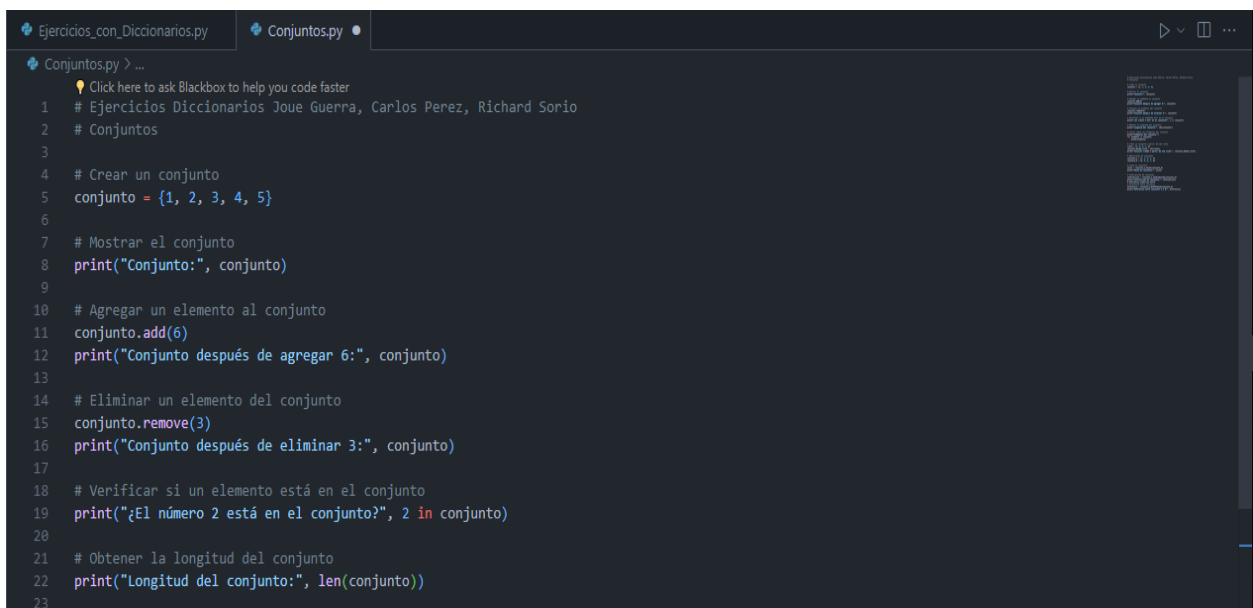
También se pueden crear conjuntos vacíos utilizando la función `set()`:

2. `conjunto_vacio = set()`

Los conjuntos tienen varias características útiles, como la capacidad de realizar operaciones de conjuntos como unión, intersección, diferencia y diferencia simétrica. También son eficientes en términos de búsqueda de elementos, ya que utilizan tablas hash internamente para almacenar los elementos únicos.

Es importante tener en cuenta que los conjuntos no conservan el orden de inserción de los elementos, por lo que no se puede acceder a los elementos mediante índices como en una lista. Además, los conjuntos no admiten elementos mutables como listas o diccionarios, pero sí admiten elementos inmutables como cadenas, números y tuplas [1].

#### EJEMPLO:

A screenshot of a code editor showing a Python script named 'Conjuntos.py'. The script demonstrates various set operations: creating a set, adding an element, removing an element, checking membership, and getting the length. The code is as follows:

```
1 # Ejercicios Diccionarios Joue Guerra, Carlos Perez, Richard Sorio
2 # Conjuntos
3
4 # Crear un conjunto
5 conjunto = {1, 2, 3, 4, 5}
6
7 # Mostrar el conjunto
8 print("Conjunto:", conjunto)
9
10 # Agregar un elemento al conjunto
11 conjunto.add(6)
12 print("Conjunto después de agregar 6:", conjunto)
13
14 # Eliminar un elemento del conjunto
15 conjunto.remove(3)
16 print("Conjunto después de eliminar 3:", conjunto)
17
18 # Verificar si un elemento está en el conjunto
19 print("¿El número 2 está en el conjunto?", 2 in conjunto)
20
21 # Obtener la longitud del conjunto
22 print("Longitud del conjunto:", len(conjunto))
23
```



```
Ejercicios_con_Diccionarios.py Conjuntos.py
Conjuntos.py > ...
24 # Iterar sobre los elementos del conjunto
25 print("Elementos del conjunto:")
26 for elemento in conjunto:
27     print(elemento)
28
29 # Crear un conjunto a partir de una lista
30 lista = [4, 5, 6, 7, 8]
31 conjunto_desde_lista = set(lista)
32 print("Conjunto creado a partir de una lista:", conjunto_desde_lista)
33
34 # Operaciones de conjuntos
35 conjunto_a = {1, 2, 3, 4, 5}
36 conjunto_b = {4, 5, 6, 7, 8}
37
38 # Unión de conjuntos
39 union = conjunto_a.union(conjunto_b)
40 print("Unión de conjuntos:", union)
41
42 # Intersección de conjuntos
43 interseccion = conjunto_a.intersection(conjunto_b)
44 print("Intersección de conjuntos:", interseccion)
45
46 # Diferencia entre conjuntos
47 diferencia = conjunto_a.difference(conjunto_b)
48 print("Diferencia entre conjuntos A y B:", diferencia)
```

Ejecución del programa de ejemplo:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR
PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos> & C:/Users/josug/AppData/Local/Python/Python39-64/Python.exe E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos\Conjuntos.py
Conjunto: {1, 2, 3, 4, 5}
Conjunto después de agregar 6: {1, 2, 3, 4, 5, 6}
Conjunto después de eliminar 3: {1, 2, 4, 5, 6}
¿El número 2 está en el conjunto? True
Longitud del conjunto: 5
Elementos del conjunto:
1
2
4
5
6
Conjunto creado a partir de una lista: {4, 5, 6, 7, 8}
Unión de conjuntos: {1, 2, 3, 4, 5, 6, 7, 8}
Intersección de conjuntos: {4, 5}
Diferencia entre conjuntos A y B: {1, 2, 3}
PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos>
```

## Algoritmos de ordenamiento por selección

## ALGORITMO DE ORDENAMIENTO POR SELECCIÓN



Es un algoritmo simple e intuitivo que consiste en seleccionar repetidamente el elemento más pequeño (o más grande, dependiendo del orden deseado) de una lista no ordenada y colocarlo al principio (o al final) de la lista ordenada. Este proceso se repite hasta que toda la lista esté ordenada [2].

#Ejemplo:

```

Ejercicios_con_Diccionarios.py  Conjuntos.py  Ordenamiento_por_Seleccion.py X
Ordenamiento_por_Seleccion.py > ordenamiento_seleccion
Click here to ask Blackbox to help you code faster
1 def ordenamiento_seleccion(lista):
2     n = len(lista)
3
4     # Iterar sobre todos los elementos de la lista
5     for i in range(n):
6         # Encontrar el índice del elemento mínimo en la parte no ordenada de la lista
7         indice_minimo = i
8         for j in range(i + 1, n):
9             if lista[j] < lista[indice_minimo]:
10                indice_minimo = j
11
12        # Intercambiar el elemento mínimo con el primer elemento de la parte no ordenada
13        lista[i], lista[indice_minimo] = lista[indice_minimo], lista[i]
14
15    return lista
16
17    # Ejemplo de uso
18    lista = [64, 25, 12, 22, 11]
19    print("Lista original:", lista)
20    print("Lista ordenada por selección:", ordenamiento_seleccion(lista))

```

Ejecución del programa de ejemplo:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR
PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos> & C:/Users/josug/AppData/Local/Python/Python39-64/Python.exe E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos\ordenamiento_seleccion.py
Lista original: [64, 25, 12, 22, 11]
Lista ordenada por selección: [11, 12, 22, 25, 64]
PS E:\Documentos\Algoritmos\Deberes\Deber 12\Códigos>

```



**ENTREGABLES:**

- Una vez culminada tu tarea, capturar las pantallas de la ejecución de cada ejercicio con tus datos y súbela en el apartado del aula virtual “S15-Taller-Tarea
- Subir los ejercicios al git o al drive y entrega la url de los archivos .py o, a su vez, entregue el archivo.
- Recordar que el nombre del archivo deberá ser: S15\_Taller-Tarea\_Algoritmos\_2023B\_NApellido(de todos los integrantes)

**RECURSOS NECESARIOS**

- Acceso a Internet.
- Imaginación.
- VSC

**RECOMENDACIONES:**

Se recomienda dedicar tiempo regularmente a practicar la implementación de algoritmos de ordenamiento, como Quicksort y algoritmos de selección, en diferentes contextos y con diversas variaciones de datos. Además, es fundamental dedicar tiempo a comprender en profundidad el funcionamiento de estas estructuras de datos y algoritmos, así como a explorar casos de uso reales donde se puedan aplicar de manera efectiva. Se sugiere también aprovechar recursos en línea, como tutoriales, cursos y comunidades de programadores, para obtener orientación y apoyo adicional durante el proceso de aprendizaje.

**CONCLUSIONES:**

El desarrollo de habilidades de programación en Python a través de la práctica de implementar algoritmos de ordenamiento y explorar estructuras de datos como diccionarios y conjuntos es fundamental para mejorar la capacidad de resolver problemas de manipulación y gestión de datos de manera eficiente. Al dedicar tiempo y esfuerzo a comprender en profundidad estos conceptos y practicar su implementación en diversos escenarios, se adquiere una base sólida para abordar problemas más complejos en el futuro y para desarrollar soluciones efectivas en el ámbito de la programación.

**ENLACES:**

Enlace GitHub:

<https://github.com/JosueGuerra2023B/Estructuras-Datos2023B/tree/master/Deberes/Deber%2012>

# 1 Bibliografía

- [1] A. H. Gonzales, «Recursos Python,» 29 06 2020. [En línea]. Available: <https://recursospython.com/guias-y-manuales/conjuntos-sets/>. [Último acceso: 2024 03 01].
- [2] parzibyte, «Parzibyte's blog,» 23 05 2021. [En línea]. Available: <https://parzibyte.me/blog/2021/05/23/python-ordenamiento-seleccion/>. [Último acceso: 03 01 2024].
- [3] H. E. HERRERA Monterroso, septiembre 2012. [En línea]. Available: <http://goo.gl/JcqRY>.
- [4] M. ATEHORT, marzo 2002. [En línea]. Available: <http://goo.gl/HbbfD>.