2.6 Sets o Conjuntos

Recordemos que podemos definir ***strings*** con comillas simples o dobles.

Los sets o conjuntos son estructuras especiales que nos ayudan a almacenera un grupo de elementos. Cuando el orden de los elementos no es relevante podemos utilizar **sets** en Python.

```
>>> conjunto1 = {"a", "b", "c"}
```

```
>>> conjunto2 = {'d','e','f','a'}
```

Como en matemáticas básicas podemos operar con conjuntos.

- **Uso del método add( ) en conjuntos**

```
>>> conjunto1.add('d')
```

```
>>> print(conjunto1)
{'a', 'b', 'c', 'd'}
```

- **Uso del método update( )**

Este método actualiza el conjunto con los elementos únicos que vienen de otro conjunto. Como en todo conjunto, los elementos únicos permanecen.

```
>>> conjunto1.update(conjunto2)
```

```
>> conjunto1
{'a', 'b', 'c', 'd', 'e', 'f'}
```

- **Uso del método union( )**

Este método retorna un conjunto con todos los elementos que vienen de otro conjunto. Este método no actualiza el conjunto actual.

```
>>> conjunto3 = {1,2,3,4}
```

```
>>> conjunto4 = {3,4,5,6}
```

No se actualiza el conjunto actual al utilizar ***union***.

```
>>> conjunto4.union(conjunto3)
{1, 2, 3, 4, 5, 6}
```

```
>>> conjunto4
{3, 4, 5, 6}
```

- **Uso del método remove( )**

Remueve el elemento especificado como argumento.

```
>>> conjunto4.remove(5)

>>> conjunto4

{3, 4, 6}

>>> conjunto4.remove(8)

KeyError Traceback (most recent call last)
Input In [54], in <cell line: 1>()
----> 1 conjunto4.remove(8)

KeyError: 8
```

El mensaje de error nos indica que no encuentra el elemento buscado para remover.

- **Uso del método discard( )**

Si un elemento se encuentra este método lo remueve del conjunto. La diferencia estará en la forma como este método maneja el error, si el elemento buscado no se encuentra, no devolverá nada.

discard no trata el error de la misma manera que lo hace el método ***remove***.

```
>>> conjunto4.discard(8)

>>> print(conjunto4)
{3, 4, 6}
```

- **Uso del método pop( )**

Este método remueve un elemento de manera aleatoria de un conjunto. Este método no requiere de argumentos para operar.

```
>>> conjunto5 = {'a',8,3,'d'}

>>> print(conjunto5)
{3, 8, 'a', 'd'}

>>> conjunto5.pop()
8
```


29

```
>>> print(conjunto5)
{3, 'a', 'd'}
```

- **Uso del método clear( )**

Este método elimina todos los elementos de un conjunto dado.

```
>>> conjunto5.clear()
```

Set() es la representación de un conjunto vacío.

```
>>> print(conjnuto5)
set()
```

- **intersection( ) method**

Este método devuelve los elementos que se encuentran en ambos conjuntos. La intersección se define como los elementos en común.

```
>>> conjunto6 = {2,4,6,8}
```

```
>>> conjunto7 = {'b','c',2}
```

```
>>> conjunto6.intersection(conjunto7)
{2}
```

```
>>> print(conjunto6)
{2, 4, 6, 8}
```

Ejercicio propuesto

SI desea actualizar un conjunto intente utilizar el método **intersection_update()**

- **Uso del método isupperset( )**

Este método devuelve **True** si todos los **items** de un conjunto se encuentran en otro conjunto dado.

```
>>> conjunto7.issuperconjunto(conjunto6)
False
```

- **Uso del método issubset( )**

Este método devuelve **True** si todos los ítems de otro conjunto se encuentran en el conjunto actual.

```
>>> conjunto9 = {1,2,3}
```

```
>>> conjunto10 = {4,1,5,2,6,3}

>>> conjunto9.issubset(conjunto10)
True



>>> conjunto0.issubset(conjunto9)
False
```

- **Uso del método isdisjoint()**

Este método devuelve **True** en el caso de que todos los ítems de un conjunto no se encuentren en un conjunto dado. Si al menos uno de ellos existe en otro conjunto este método devuelve **False.**

```
>>> set11 = {'a','b','c','d'}

>>> set12 = {'a','b','c','e'}

>>> set12.isdisjoint(set11)
False

>>> set13 = {'f','g','h'}

>>> set13.isdisjoint(set12)
True
```

El uso de estos métodos también permite encontrar elementos en común entre conjuntos.

2.7 Diccionarios

Los diccionarios tienen una sintaxis que relaciona **key-value**

```
>>> dict1 = {'clave1':'valor1', 'clave2':'valor2'}
```

Los diccionarios no aceptan claves duplicadas.

```
>>> dict2 = {'clave1':'valor1', 'clave1':'valor2',
'clave3':'valor2'}

>>> print(dict2)
{'clave1': 'valor2', 'clave3': 'valor2'}
```

Los **key** en un diccionario deben ser únicos. Cuando hay duplicados se sobre escriben.

- **Operaciones con Diccionarios**
- **Uso del método update( )**

31

Este método se emplea para actualizar un par clave-valor.

```
>>> estudiantes = {'phill':35, 'Sam': 32, 'Diana':28}

>>> print(estudiantes)
{'phill': 35, 'Sam': 32, 'Diana': 28}

>>> estudiantes.update({'phill': 37})

>>> print(estudiantes)
{'phill': 37, 'Sam': 32, 'Diana': 28}

>>> estudiantes.update({'karol': 22})

>>> print(estudiantes)
{'phill': 37, 'Sam': 32, 'Diana': 28, 'karol': 22}
```

- **Uso del método keys( )**

```
>>> estudiantes.keys()
dict_keys(['phill', 'Sam', 'Diana', 'karol'])
```

- **Uso del método values( )**

```
>>> estudiantes.values()
dict_values([37, 32, 28, 22])
```

- **Uso del método copy( )**

```
>>> estudiantes2 = estudiantes.copy()

>>> estudiantes2
{'phill': 37, 'Sam': 32, 'Diana': 28, 'karol': 22}
```

- **-Uso del método fromkeys( )**

Este método es útil para crear un diccionario llamando variables previamente definidas.

```
>>> keys = {1, 2, 3, 4, 5, 6, 7, 8, 9}

>>> import random

>>> values = random.random()

>>> print(values)
0.8560380475919955
```

Ten en cuenta que el valor generado puede ser distinto. Estamos usando la librería **random**.

```
>>> dict3 = dict.fromkeys(keys,values)
```

```
>>> print(dict3)
{1: 0.8560380475919955,
 2: 0.8560380475919955,
 3: 0.8560380475919955,
 4: 0.8560380475919955,
 5: 0.8560380475919955,
 6: 0.8560380475919955,
 7: 0.8560380475919955,
 8: 0.8560380475919955,
 9: 0.8560380475919955}
```

- **Uso del método get ( )**

Este método retorna el valor para una clave dada.

```
>>> dict3.get(5)
0.8560380475919955
```

Si la clave no existe en el diccionario el método no retornará ningún error. Pero si incluirá una nueva clave-valor al diccionario.

```
>>> dict3.get(20)
```

```
>>> print(dict3)
{1: 0.8560380475919955,
 2: 0.8560380475919955,
 3: 0.8560380475919955,
 4: 0.8560380475919955,
 5: 0.8560380475919955,
 6: 0.8560380475919955,
 7: 0.8560380475919955,
 8: 0.8560380475919955,
 9: 0.8560380475919955}
```

- **Uso del método pop( )**

```
>>> dict3.pop(5)
0.8560380475919955
```


33

Este método devuelve el valor que eliminamos del diccionario.

```
>>> print(dict3)
{1: 0.8560380475919955,
 2: 0.8560380475919955,
 3: 0.8560380475919955,
 4: 0.8560380475919955,
 6: 0.8560380475919955,
 7: 0.8560380475919955,
 8: 0.8560380475919955,
 9: 0.8560380475919955}
```

- **Uso del método clear ( )**

Este método eliminar todos los elementos de un diccionario.

```
>>> dict3.clear()
```

```
>>> print(dict3)
{}
```

REFERENCIAS
SELECCIONADAS

35

Ejercicios de repaso

Ejercicio 2.1

Cree una variable llamada ***cadena*** que contenga una cadena de texto "Python puro". Al crearla verifique el tipo de dato que queda guardado en memoria utilizando la función ***type***.

Ejercicio 2.2

Utilice un método que le permita poner en mayúscula la palabra puro.

Ejercicio 2.3

Qué diferencia encuentra entre el método ***titlle*** y el método ***capitalize.***

Ejercicio 2.4

Aplique un método para verificar si la letra ***o*** es la que finaliza la cadena de caracteres.

Ejercicio 2.5

Investigue cómo utilizar el método ***casefold*** sobre la cadena de texto y para qué serviría.

Ejercicio 2.6

Cree un programa que defina una lista con las primeras cinco letras del alfabeto en minúscula. Posterior a la creación de la lista, aplique un método para insertar en la posición 2 la letra ***a***. Ahora, aplique un método para contar el número de veces que la letra ***a*** se encuentra en la lista creada.

Ejercicio 2.7

Utilizando la lista creada en el punto anterior aplique el método que le permita eliminar la letra ***c*** que se encuentra en la posición 3. Confirme que el método utilizado devuelva la letra eliminada y que al llamar nuevamente la lista el elemento no se encuentre en la posición indicada.

Ejercicio 2.8

Cree una lista con los números del 1 al 10 considerando el orden de su preferencia. Llame a la lista ***valores.*** Aplique un método que le permita ordenar los valores desde el mayor al menor. Si ahora utiliza la función ***sorted*** sobre la lista, ¿qué diferencia encuentra?.

Ejercicio 2.9

Considere la creación de una nueva lista llamada ***duplicada*** que sea una copia de la lista ***valores***. Posteriormente ejecute una instrucción para que la copia quede ordenada desde los valores más bajos a los más altos.

Ejercicio 2.10

Cree una tupla llamada ***ventas*** que considere en la primera posición el nombre de un producto y su precio. Para el ejemplo puede considerar ***leche*** como el nombre y su precio ***5.*** Utilice el método ***index*** para consultar el índice correspondiente al precio. Ahora en otra instrucción y utilizando el índice intente reemplazar el elemento asignándole un nuevo valor ***8***. ¿Qué interpreta del error que sale al ejecutar la instrucción?

EJERCICIOS DE REPASO – CAPÍTULO: 2
LIBRO: Python Puro
AUTOR: Andrés Salazar

Ejercicio 2.11

Con la tupla definida en el ejercicio anterior aplique el método ***count*** para consultar cuántas veces

aparece la letra ***"e"*** dentro de la tupla, de esta manera:

>>> ventas.count("e")

Interprete el resultado

Posteriormente indexe la tupla y consulte cuántas veces aparece la letra ***"e"*** dentro del primer elemento de la tulpa.

>>> ventas[0].count("e")

¿Qué diferencias encuentra en los dos resultados?

Ejercicio 2.12

Defina un conjunto vacío denominado ***figuras.*** A continuación, utilice un método para que el conjunto contenga los nombres de tres figuras geométricas, por ejemplo: ***circulo***, ***cuadrado***, ***rectángulo***. Aplique un método que permita agregar un nuevo elemento utilizando exactamente uno de los nombres ya definidos. Consulte el estado actual del conjunto ***figuras***, ¿qué interpreta del resultado?

Ejercicio 2.13

Emplee el conjunto ***figuras*** definido en el ejercicio anterior. Emplee la siguiente instrucción:

>>> figuras[0]

¿Qué interpreta del resultado?

Ejercicio 2.14

Utilice el conjunto ***figuras*** definido anteriormente. Aplique el método ***unión*** agregando dentro del argumento otro conjunto que incluya más figuras geométricas. Consulte de nuevo el conjunto ***figuras.*** ¿qué interpreta del resultado?

Ejercicio 2.15

Cree un conjunto llamado ***personas*** con nombres de personas que conozca***.*** Aplique el método ***update*** de la siguiente manera:

>>> personas.update({ "pedro", "ramiro "})

Posteriormente aplique el método ***pop.*** ¿Cómo describiría el uso de este método y qué argumentos necesita para operar?

Ejercicio 2.16

Utilice el conjunto definido ***personas*** de los ejercicios anteriores. Cree ahora un nuevo conjunto llamado ***estudiantes*** definido de la siguiente manera:

>>> estudiantes = {' luciana', 'ramiro '}

Aplique el método que permite establecer los elementos diferentes entre los dos conjuntos tomando como base el conjunto ***personas***, ¿Qué logra identificar de los resultados?, ¿para qué serviría el método?

Ejercicio 2.17

Ejecute la última instrucción del ejercicio anterior pero ahora tomando como base el conjunto ***estudiantes***. ¿Qué diferencia encuentra entre los dos resultados?

vez creado compruebe el tipo de objeto que queda almacenado en la variable ***calificaciones.*** Utilice para esto la función **t*ype.***

Ejercicio 2.18

Se desea conocer si el conjunto ***estudiantes*** es un subconjunto de las personas. Subconjunto significa que todos los elementos que son parte de ***estudiantes*** están igualmente definidos en el conjunto ***personas***. Aplique un método para conocer la respuesta.

Ejercicio 2.19

Actualmente los conjuntos no tienen un método ***built-in*** en Python 3.8 para ordenar los elementos que lo componen. Esto se debe a que en un conjunto el orden no interesa, sin embargo, es posible aplicar la función ***sorted***. Ejecute una instrucción que utilice esta función e interprete el resultado.

Ejercicio 2.20

Para la construcción de una aplicación de notas escolares es necesario crear una base de datos con los nombres de los estudiantes y la nota obtenida en un curso. Para almacenar los datos se le ha pedido que cree un diccionario donde las claves sean los nombres de los estudiantes y los valores sean las notas obtenidas. Tenga en cuenta los datos presentados en la siguiente tabla:

Nombre de estudiante	Nota asignada
Liliana	4.5
Carmen	3.3
Josefina	4.1
Daniela	4.9
Pedro	2.9
José	4.6
Mario	3.3

Tabla resumen de datos

Para efectos de desarrollar los siguientes ejercicios se definirá el diccionario como ***calificaciones*.** Una

Ejercicio 2.21

Aplique la función ***sorted*** sobre el diccionario calificaciones. Discuta la necesidad de usar esta función en el diccionario analizado.

Ejercicio 2.22

Parece ser que el método pop permite eliminar un elemento de un diccionario. Ejecute la siguiente instrucción:

```
>>>
calificaciones.popitem({'Daniela': 4.9})
```

¿Qué interpretación se le puede dar al mensaje de error que nos entrega el intérprete cuando ejecutamos?:

```
TypeError: popitem() takes no arguments (1 given)
```

¿Cuándo es conveniente aplicar este método?

Ejercicio 2.23

Utilice el método ***ítems*** sobre el diccionario. ¿qué retorna este método?, ¿qué tipo de estructura de datos parece, una lista?, una lista de listas?, ¿una lista de tuplas?. Investigue para más información sobre los objetos de tipo ***dict_items.***

Ejercicio 2.24

Sobre el diccionario ***calificaciones*** aplique el método update verificando los argumentos necesarios para el

correcto funcionamiento si se quiere modificar la nota asignada a la estudiante Liliana a un valor de 4.7.

39

41

Capítulo 3

[Las barras laterales son perfectas para remarcar puntos importantes del texto o agregar información adicional de referencia rápida como, por ejemplo, una programación.

Por lo general, se colocan en la parte izquierda, derecha, superior o inferior de la página. No obstante, se pueden arrastrar fácilmente a cualquier posición que prefiera.

Cuando esté listo para agregar contenido, haga clic aquí y empiece a escribir.]

Python Puro

Felipe Salazar

www.profefelipe.com

3. Control de Flujo y bucles

3.1 Condicionales

3.1.1 Uso de la expresión if

```
>>> x = 100
>>> y = 200
>>> if x > y:
        print("x es mayor que y")
```

Como la condición no se cumple no tendremos salida al ejecutar el **if.** Usar la declaración **if** en Python significa que debe considerar la indentación. Se recomienda utilizar cuatro espacios de indentación.

Recuerda: "**Simple is better than complex".** Otros lenguajes de programación usan símbolos específicos para definir el alcance de una pieza de código.

Recuerda: "**Simple is better than complex".** thon se basa en este rincipio para que sea un guaje fácil de mplementar, scribir y leer.

Cuando usas una expresión **if** debes definir al menos el caso favorable. De lo contrario no se permitirá ejecutar la condición por un error de sintaxis.

3.1.2 Uso del control if -else

```
>>> if x < y:
        print("x es menor que y")
    else:
        print("x es menor que y")
```

```
Output: x es menor que y
```

La expresión **else** debe estar identada al nivel que se le hace referencia en **if**.

3.1.3 Uso del control if-elif-else

elif es una palabra reservada que podemos usar en caso de evaluar diferentes alternativas antes de usar el **else.**

```
>>> if x < y:
        print("x es menor que y")
    elif x == y:
```


43

```
        print("x es igual a y")
    else:
        print("x es mayor a y")
```

Output: **x es menor que y**

Queremos comprobar una condición usando el símbolo de doble igualdad **==** para especificar un caso.

```
>>> w = 50

>>> z = 25

>>> if w < z:
       print("w es menor que z")
    elif w == 50:
       print("w es igual a 50")
    elif z == 10:
       print("z es igual a 10")
    else:
       print("w es mayor que z")
```

Output: **w es igual a 50**

Podemos usar múltiples expresiones **elif** antes del **else**. Incluso podemos ejecutar un condicional sin la necesidad de este último.

3.1.4 If anidados

Podemos escribir un script que represente el cumplimiento de varias condiciones sucesivas. A esto se le conoce como **if** anidados.

```
>>> a = 'azúl'
>>> b = 'rojo'
>>> if a == 'azúl':
       if b == 'rojo':
           print('Si mezclas esos colores tendrás un purpura')
```

Output: **Si mezclas esos colores tendrás un purpura**

```
>>> if a == 'azúl' and b == 'rojo':
       print('Si mezclas esos colores tendrás un purpura')
```

Output: **Si mezclas esos colores tendrás un purpura**

En un if no necesario u el **else** com condición f

3.2 Bucles

3.2.1 Ciclo For

El bucle for es una expresión para el control del flujo usada en programación donde puedes definir el número de iteraciones en una parte del código.

```
>>> var1 = "Jupyter"

>>> for i in var1:
        print(i)
```

```
Output:
J
u
p
y
t
e
r
```

El bucle **for** utiliza indentación, así como lo hace la expresión **if** Usamos una variables o contador para manejar el número de iteraciones a través del código identado. En el anterior ejemplo "i".

demos crear distintas structuras de datos con el ciclo for.

Podemos usar el ciclo con múltiples propósitos.

```
>>> mi_lista = [] # Definimos una lista vacía

>>> for paso in range(1,10):
        mi_lista.append(paso)
    print(mi_lista)
    print(type(mi_lista))
```

```
Output:
[1, 2, 3, 4, 5, 6, 7, 8, 9]
<class 'list'>
```

Ahora utilicemos un ciclo **for** para crear un conjunto.

```
>>> mi_conjunto = set() # Definimos un conjunto vacío

>>> for k in range(1,10):
        mi_conjunto.add(k)
    print(mi_conjunto)
    print(type(mi_conjunto))
```


45

```
Output:
{1, 2, 3, 4, 5, 6, 7, 8, 9}
<class 'set'>
```

Podríamos crear una estructura como un diccionario con un ciclo **for**.

```
>>> mi_dic = {} # Definimos un diccionario vacío
>>> valor = "elemento"
>>> for key in range(1,10):
        mi_dic.update({key:valor})
    print(mi_dic)
    print(type(mi_dic))
```

```
Output:
{1: 'elemento', 2: 'elemento', 3: 'elemento', 4: 'elemento', 5:
'elemento', 6: 'elemento', 7: 'elemento', 8: 'elemento', 9:
'elemento'}
<class 'dict'>
```

3.2.2 Ciclo While

Esta es otra expresión que nos permite controlar el flujo a través de una pieza de código.

Un ciclo wh
expresa mi
con lo cual
controlamo
proceso mi
que se cum
una condici

```
>>> j = 10
>>> while j < 20:
        print(j)
        j = j + 1
```

```
Output:
10
11
12
13
14
15
16
17
18
19
```

En cada
ón se está
niendo un
xto y una
variable.

```
>>> var1 = 10

>>> while var1 > 1:
        print("El valor actual es: ", var1)
        var1 = var1 - 1
        print("restando 1, ahora el valor es: ", var1)
```

```
Output:
El valor actual es:  10
restando 1, ahora el valor es:  9
El valor actual es:  9
restando 1, ahora el valor es:  8
El valor actual es:  8
restando 1, ahora el valor es:  7
El valor actual es:  7
restando 1, ahora el valor es:  6
El valor actual es:  6
restando 1, ahora el valor es:  5
El valor actual es:  5
restando 1, ahora el valor es:  4
El valor actual es:  4
restando 1, ahora el valor es:  3
El valor actual es:  3
restando 1, ahora el valor es:  2
El valor actual es:  2
restando 1, ahora el valor es:  1
```

- Uso de la expresión **break**

```
>>> var1 = 10

>>> while var1 > 1:
        print("El valor actual es: ", var1)
        var1 = var1 - 1
        print("restando 1, ahora el valor es: ", var1)
        if var1 == 5:
            break
```

```
Output:
El valor actual es:  10
restando 1, ahora el valor es:  9
El valor actual es:  9
restando 1, ahora el valor es:  8
```


47

```
El valor actual es:  8
restando 1, ahora el valor es:  7
El valor actual es:  7
restando 1, ahora el valor es:  6
El valor actual es:  6
restando 1, ahora el valor es:  5
```

REFERENCIAS SELECCIONADAS

Ejercicios de repaso

Ejercicio 3.1

Escriba un programa que permita al usuario ingresar un texto utilizando la función ***input***. Guarde la información en una variable denominada ***entrada***. Mediante una expresión ***if*** evalúe si el texto entregado tiene en su primera letra una mayúscula.

Ejercicio 3.2

Se requiere confirmar si el texto que contiene la variable ***entrada*** puede usarse como identificador para una variable. Recuerde que una de las condiciones de las variables validas en Python es que no deben empezar con números. Imprima este texto en la expresión ***if*** en caso de que ***entrada*** sea válida:
`"Puede ser usado como nombre de variable".`

Ejercicio 3.3

Ejecute nuevamente la instrucción donde se define la variable ***entrada***. Asigne en el ***input*** un valor numérico. Utilice una expresión ***if*** para validar si el dato ingresado es numérico. Utilice en otra instrucción la función ***type*** sobre la variable ***entrada*****.** ¿Qué diferencias encuentra entre lo que entrega la expresión if y lo que entrega en este caso la función ***type*****?**

Ejercicio 3.4

Escriba un programa que permita recibir dos números en entradas distintas. Guarde cada valor en una variable distinta. Ejecute la suma de los dos valores ingresados, asegúrese de que los valores se puedan sumar. Si el resultado de la operación es un número par utilice una expresión ***if*** para imprimir un mensaje acorde a esto. Utilice la expresión ***else*** para entregar una respuesta en caso de que no se cumpla la primera condición.

Ejercicio 3.5

Utilice la expresión ***if*** para validar si el cálculo del ejercicio anterior es un valor entero, en caso de que no lo sea utilice el ***else*** para entregar el tipo de dato que representa la variable.

Ejercicio 3.6

Un semáforo peatonal en una avenida tiene normalmente dos posiciones. Detenerse o Avanzar. Desarrolle un programa que genere un número aleatorio mediante el método ***randint*** de la librería ***random*** donde escoja uno de dos posibles valores ***0*** ó ***1*** y se guarde en una variable denominada ***peatonal***. Utilice la expresión ***if*** y la expresión ***else*** para controlar el semáforo a partir del número aleatorio que se genera. En el caso de que la variable **peatonal** tome un valor de ***1*** se debe presentar un mensaje que indique que el peatón puede cruzar. En caso de que la variable tome un ***0*** se debe presentar un mensaje que indique que el peatón no puede cruzar.

Ejercicio 3.7

Genere un programa que controle las tres posiciones posibles para un semáforo vehicular tradicional. Verde para avanzar, Amarillo en precaución o preparación, Rojo para detenerse. Utilice el método ***randint*** para generar uno de tres posibles valores: ***0***, ***1*** o ***2*** que representará en el

49