Esta clase va a ser

- grabada

Clase 04. PYTHON

Métodos de colecciones

CODERHOUSE

Temario

03

Conjuntos y diccionarios

- ✓ Conjuntos
- ✓ Diccionarios

04

Métodos de colecciones

- ✓ Cadenas
- ✓ Listas
- ✓ Conjuntos
- ✓ Diccionarios

05

Operadores y expresiones

- ✓ Operadores
- ✓ Expresiones anidadas

CODERHOUSE

Objetivos de la clase

- **Utilizar** funciones avanzadas de cadenas, listas, conjuntos y diccionarios

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ Repositorio Python

CODERHOUSE

Cadenas

Upper

```
>>> cadena = “Hola Mundo”
>>> cadena.upper()
“HOLA MUNDO”
>>> “hola amigo!”.upper()
“HOLA AMIGO!”
```

Esta función integrada sirve para hacer que se devuelva la misma cadena pero con sus caracteres en mayúscula, usando el método **upper()**.

Se escribe como: **string.upper()**

Lower

```
>>> cadena = “Hola Mundo”
>>> cadena.lower()
“hola mundo”
>>> “HoLa AmIgO!”.lower()
“hola amigo!”
```

Ya vimos cómo convertir a mayúsculas, pero también es útil convertir una cadena de caracteres a minúsculas, usando el método **lower()**.

Se escribe como: **string.lower()**

Capitalize

```
>>> cadena = “Hola Mundo”
>>> cadena.capitalize()
“Hola mundo”
>>> “HoLa AmIgO!”.capitalize()
“Hola amigo!”
```

Esta función integrada sirve para hacer que se devuelva la misma cadena pero con su primer carácter en mayúscula y el resto de caracteres hacerlos minúscula, usando el método **capitalize()**.

Se escribe como:
string.capitalize()

Title

```
>>> cadena = “hOLA mUNDO”
>>> cadena.title()
“Hola Mundo”
>>> “HoLa AmIgO!”.title()
“Hola Amigo!”
```

Esta función integrada sirve para hacer que se devuelva la misma cadena pero con el primer carácter de cada palabra en mayúscula y el resto de caracteres hacerlos minúscula, usando el método **title()**.

Se escribe como: **string.title()**

Count

```
>>> cadena = “hOLa mUNDO esta cadena tiene muchas a”
>>> cadena.count(“a”)
6
>>> “HoLa amigo como estas amigo!”.count(“amigo”)
2
```

Si necesitamos saber cuantas veces aparece una subcadena dentro de la misma cadena, usando el método **count()**.

Se escribe como: **string.count()**

Find

```
>>> cadena = “hOLa mUNDO esta cadena tiene muchas a”
>>> cadena.find(“esta”)
11
>>> “HoLa amigo como estas amigo!”.find(“chau”)
-1
```

Si necesitamos averiguar el índice en el que aparece una subcadena dentro de la misma cadena, usamos el método **find()**.

Se escribe como: **string.find()**.

Si no encuentra la cadena devuelve un -1.

Rfind

```
>>> “HoLa amigo como estas amigo!”.find(“amigo”)
5
>>> “HoLa amigo como estas amigo!”.rfind(“amigo”)
22
```

Es exactamente igual al método find() los diferencia en que **rfind()** devuelve el índice, pero de la última ocurrencia de la subcadena, es decir, la última vez que aparece en la cadena.

Se escribe como: **string.rfind()**.

Si no encuentra la cadena devuelve un -1.

Split

```
>>> cadena = “hOLA mUNDO”
>>> cadena.split()
[“hOLA”, “mUNDO”]
>>> “HoLa amigo como estas
amigo!”.split(“amigo”)
[“HoLa”, “como ”, “estas ”, “!”]
```

Esta función integrada sirve para devolver una lista con la cadena de caracteres separada por cada índice de la lista.

Se escribe como: **string.split(“cadena_a_separar”)**.

Si no se indica alguna cadena para separar separa por “**espacios**”.

Join

```
>>> cadena = “Hola mundo”
>>> “,”.join(cadena)
“H,o,l,a, ,m,u,n,d,o”
>>> “ ”.join(cadena)
“H o l a   m u n d o”
```

Esta función integrada sirve para devolver una cadena separada a partir de una especie de separador.
Se escribe como:
“separador”.join(“cadena”).

Nota: Si no se especifica el separador nos devuelve un error.

Strip

```
>>> cadena = “---------Hola mundo--------”
>>> cadena.strip(“-”)
“Hola mundo”
>>> “                Hola mundo “.strip()
“Hola mundo”
>>> “ssaddaHola mundoddsaa“.strip(‘asd’)
“Hola mundo”
```

Esta función integrada sirve para devolver una cadena borrando todos los caracteres ,de delante y detrás solamente, de la cadena especificados en la cadena de caracteres a borrar que se le pasan entre parentesis a strip. Se escribe como:
Se escribe como:
cadena.strip(“caracteres_a_borrar”).

Nota: Si no se especifica el carácter elimina los espacios

Replace

Esta función integrada sirve para devolver una cadena reemplazando los sub caracteres indicados.

Se escribe como:
cadena.replace(“caracter_a_remplazar”, “caracter_que_reemplaza”).

También podemos indicar cuantas veces lo reemplazaremos utilizando un índice.

Replace

```
>>> cadena = “Hola mundo”
>>> cadena.replace(“o”, “0”)
“H0la mund0”
>>> "Hola mundo mundo mundo mundo mundo".replace('
mundo','',4)
“Hola mundo”
```

Nota: En el último reemplazamos mundo 4 veces por un sólo carácter vacío

Listas

Clear

```
>>> letras = ['a', 'b', 'c', 'd', 'e', 'f']
>>> letras.clear()
[]
```

Como vimos en listas, para “eliminar” todos los elementos de una lista podíamos hacer lista **= []**, sin embargo, también podemos usar clear() para vaciar todos los ítems de la lista.

Se escribe como: **lista.clear()**

Extend

Como vimos en las listas, podemos sumar una lista con otra lista de la siguiente forma:

```
>>> numeros = [1,2,3,4]
>>> numeros + [5,6,7,8]
```

Pero también podemos hacer uso de extend, ya que une una lista con otra.

Se usa como **lista.extend(otra_lista)**

```
>>> lista1 = [1,2,3,4]
>>> lista2 = [5,6,7,8]
>>> lista1.extend(lista2)
```

Insert

Esta función integrada se usa para agregar un ítem a una lista, pero en un índice específico.

Se escribe como: **lista.insert(posición, ítem)**.

```
>>> lista = [1,2,3,4,5]
>>> lista.insert(0, 0)
[0,1,2,3,4,5]
>>> lista2 = [5,10,15,25]
>>> lista2.insert(-1, 20) # Anteúltima posición
[5,10,15,20,25]
>>> lista3 = [5,10,15,25]
>>> n = len(lista3)
>>> lista3.insert(n, 30) # Última posición
[5,10,15,25,30]
```

Reverse

```
>>> lista = [1,2,3,4]
>>> lista.reverse()
[4,3,2,1]
```

Esta función integrada sirve para dar vuelta una lista.

Se escribe como: **lista.reverse()**

Nota: Las cadenas no tienen la función reverse, pero se puede simular haciendo una conversión a lista y después usando el join

Sort

```
>>> lista = [5,-10,35,0,-65,100
>>> lista.sort()
[-65, -10, 0, 5, 35, 100]
>>> lista.sort(reverse=True)
[100, 35, 5, 0, -10, -65]
```

Esta función integrada sirve para ordenar una lista automáticamente por valor, de menor a mayor.

Se escribe como: **lista.sort()**.

Si ponemos el argumento reverse=True la lista se ordenará de mayor a menor.

Colecciones 1

Transforma el texto

Duración: **20 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Colecciones 1

Descripción de la actividad.

Utilizando todo lo que sabes sobre cadenas, listas y sus métodos internos, transforma este texto:

gordon lanzó su curva&strawberry ha fallado por un pie!
-gritó Joe Castiglione&dos pies -le corrigió
Troop&strawberry menea la cabeza como disgustado...
-agrega el comentarista

ACTIVIDAD EN CLASE

Colecciones 1

Transforma el texto en:

Gordon lanzó su curva...
- Strawberry ha fallado por un pie! -gritó Joe Castiglione.
- Dos pies le corrigió Troop.
- Strawberry menea la cabeza como disgustado... -agrega el comentarista.

Lo único prohibido es modificar directamente el texto

Break

¡10 minutos y volvemos!

Conjuntos

Copy

```
>>> set1 = {1,2,3,4}
>>> set2 = set1.copy()
>>> print(set2)
{1,2,3,4}
```

Esta función integrada sirve para hacer que se devuelva una copia de un set.

Se escribe como: **set.copy()**

Isdisjoint

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.isdisjoint(set2)
False
```

Esta función comprueba si el set es distinto a otro set, es decir, si no hay ningún ítem en común entre ellos.
Se escribe como: **set1.isdisjoint(set2)**

Nota: Devuelve False porque set1 y set2 comparten el 3

Issubset

```
>>> set3 = {-1,99}
>>> set4 = {1,2,3,4,5}
>>> set3.issubset(set4)
False
```

Esta función comprueba si el set es subset de otro set, es decir, si todos sus ítems están en el otro conjunto.

Se escribe como:
set1.issubset(set2)

Nota: Devuelve False porque set3 no está todo dentro de set4

Issuperset

```
>>> set5 = {1,2,3}
>>> set6 = {1,2}
>>> set5.issuperset(set6)
True
```

Esta función es muy similar al issubset, la diferencia es que esta comprueba si el set es contenedor de otro set, es decir, si contiene todos los ítems de otro set.

Se escribe como: **set1.issuperset(set2)**

Unión

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.union(set2)
{1,2,3,4,5}
```

Esta función une un set con otro, y devuelve el resultado en un nuevo set.

Se escribe como: **set1.union(set2)**

Difference

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.difference(set2)
{1,2}
```

Esta función encuentra todos los elementos no comunes entre dos set, es decir, nos devuelve un set de ítems diferentes entre cada set.

Se escribe como:
set1.difference(set2)

Nota: Acá devuelve 1 y 2 por qué le pregunta básicamente “que tengo de diferente al set2?”

Difference_update

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.difference_update(set2)
>>> print(set1)
{1,2}
```

Similar al difference, pero esta función nos guarda los ítems distintos en el set originales, es decir, le asigna como nuevo valor los ítems diferentes.
Se escribe como:
set1.difference_update(set2)

Nota: Ahora set1 vale {1,2} porque es la diferencia que tenía con set2

Intersection

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.intersection(set2)
{3}
```

Esta función devuelve un set con todos los elementos comunes entre dos set, es decir, nos devuelve un set de ítems iguales entre cada set.

Se escribe como: **set1.intersection(set2)**

Intersection_update

```
>>> set1 = {1,2,3}
>>> set2 = {3,4,5}
>>> set1.intersection_update(set2)
>>> print(set1)
{3}
```

Es exactamente igual al intersection, pero esta función actualiza el set original, es decir, le asigna como nuevo valor los ítems en común.
Se escribe como:
set1.intersection_update(set2)

Nota: Ahora set1 vale los ítems en común con set2

Colecciones 2

Analizar e interpretar los siguientes conjuntos en Python

Duración: **15 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Colecciones 2

¡Importante!
Ejecuta el código para analizar cada una de las salidas!

Ejemplo 1:

```
my_set_1 = set([1, 2, 3])
my_set_2 = set([3, 4, 5])
my_new_set =
my_set_1.union(my_set_2)
print(my_new_set)
```

Ejemplo 2:

```
my_set_1 = set([1, 2, 3])
my_set_2 = set([3, 4, 5])
my_new_set =
my_set_1.intersection(my_set_2)
print(my_new_set)
```

Ejemplo 3:

```
my_set_1 = set([1, 2, 3])
my_set_2 = set([3, 4, 5])
my_new_set =
my_set_1.difference(my_set_2)
print(my_new_set)
```

Diccionarios

Get

La función get sirve para poder buscar un elemento a partir de su key, en el caso de no encontrar devuelve un valor por defecto que le indicamos nosotros. Se escribe como: dict.get(key, “valor por defecto”)

```
>>> colores = { "amarillo":"yellow", "azul":"blue", "verde":"green" }
>>> colores.get(“rojo”, “no hay clave rojo”)
“no hay clave rojo”
>>> colores.get(“amarillo”, “no hay clave amarillo”)
“yellow”
```

Keys

```
>>> colores = { "amarillo":"yellow", "azul":"blue",
"verde":"green" }
>>> colores.keys()
dict_keys(['amarillo', 'azul', 'verde'])
```

La función key sirve para poder traer todas las claves de un diccionario en el caso de desconocerlas.

Se escribe como: **dict.keys()**

Values

```
>>> colores = { "amarillo":"yellow", "azul":"blue",
"verde":"green" }
>>> colores.values()
dict_values(['yellow', 'blue', 'green'])
```

La función values es similar a keys, pero esta sirve para poder traer todos los valores de un diccionario.

Se escribe como: **dict.values()**

Items

```
>>> colores = { "amarillo":"yellow", "azul":"blue", "verde":"green" }
>>> colores.items()
dict_items([('amarillo', 'yellow'), ('azul', 'blue'), ('verde', 'green')])

>>> for clave, valor in colores.items():
        print(clave, valor)
>>> amarillo yellow
>>> azul blue
>>> verde green
```

La función items es similar a keys y values, pero esta crea una lista con clave y valor de los ítems de un diccionario.

Se escribe como: **dict.items()**

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ Repositorio Python

CODERHOUSE

Colecciones 3

Duración: **15 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Colecciones 3

Descripción de la actividad.

Escribir un programa que guarde en una variable en un diccionario {'Dolar':'$','Euro':'€', 'Libra':'£'}.

Luego se le deberá solicitar al usuario que ingrese la divisa que desea visualizar.

En el caso de ingresar una divisa no existente en nuestro diccionario, deberemos indicarle con un mensaje de notificación que la divisa no se encuentra disponible.

#Codertraining

¡No dejes para mañana lo que puedes practicar hoy! Te invitamos a revisar la Guía de Ejercicios Complementarios, donde encontrarás un ejercicio para poner en práctica lo visto en la clase de hoy.

Actividad extra Nº 5

Colecciones práctica extra

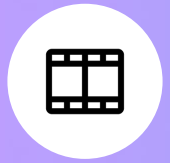
Consigna

A partir de una lista realizar las siguientes tareas sin modificar la lista original:

1. Borrar los elementos duplicados
2. Ordenar la lista de mayor a menor
3. Eliminar todos los números impares (for ---- if (%2==1) ---- pop, remove)
4. Realizar una suma de todos los números que quedan (sum(lista))
5. Añadir como primer elemento de la lista la suma realizada insert(0, suma)
6. Devolver la lista modificada
7. Finalmente, después de ejecutar la función, comprueba que la suma de todos los números a partir del segundo, concuerda con el primer número de la lista

lista = [29, -5, -12, 17, 5, 24, 5, 12, 23, 16, 12, 5, -12, 17]

Nota: Recuerda que para sumar todos los números de una lista puedes usar sum



¿Quieres saber más?
Te dejamos material ampliado de la clase

CODERHOUSE

MATERIAL AMPLIADO

Recursos multimedia

- ✓ Artículo: Cadenas
- ✓ Artículo: Listas
- ✓ Artículo: Tuplas
- ✓ Artículo: Diccionarios
- ✓ Artículo: Conjuntos
- ✓ EjemploClase 

Disponible en nuestro repositorio.

¿Preguntas?

Resumen
de la clase hoy

- ✓ Listas
- ✓ Tuplas
- ✓ Anidación
- ✓ Transformación de colecciones

Opina y valora esta clase

Muchas gracias.

#DemocratizandoLaEducación