Lenguaje de Programación Python

@SENAComunica

www.sena.edu.co

Método index()

Utilizamos el método index( ) para explorar strings, ya que permite hallar el índice de aparición de un carácter o cadena de caracteres dentro de un texto dado.

- **string.index(value, start, end)**: Busca caracteres desde hasta una posición.

- **string.rindex(value, start, end)**: Busca caracteres desde hasta una posición pero en sentido inverso.

- **String[i]**: Devuelve el carácter con el índice i

Usar el índice de una cadena

Escribe el siguiente código en tu Pycharm y comprueba los resultados.

```
1  mi_texto = "Esta es una prueba"
2  resul1 = mi_texto[0]
3  resul2 = mi_texto[-4]
4
5  print(resul1)
6  print(resul2)
```

Método index()

Escribe el siguiente código en tu Pycharm y comprueba los resultados.

```
1  mi_texto = "Esta es una prueba"
2  resul1 = mi_texto.index("n")
3  print(resul1)
4  resul1 = mi_texto.index("prueba")
5  print(resul1)
6  resul1 = mi_texto.index("a")
7  print(resul1)
8  resul1 = mi_texto.index( __sub: "a", __start: 5, __end: 12)
9  print(resul1)
```

Método rindex()

Escribe el siguiente código en tu Pycharm y comprueba los resultados.

```
1  mi_texto = "Esta es una prueba"
2  resul1 = mi_texto.rindex("n")
3  print(resul1)
4  resul1 = mi_texto.rindex("prueba")
5  print(resul1)
6  resul1 = mi_texto.rindex("a")
7  print(resul1)
8  resul1 = mi_texto.rindex( __sub: "a", __start: 5, __end: 12)
9  print(resul1)
```

Ejercicios index()

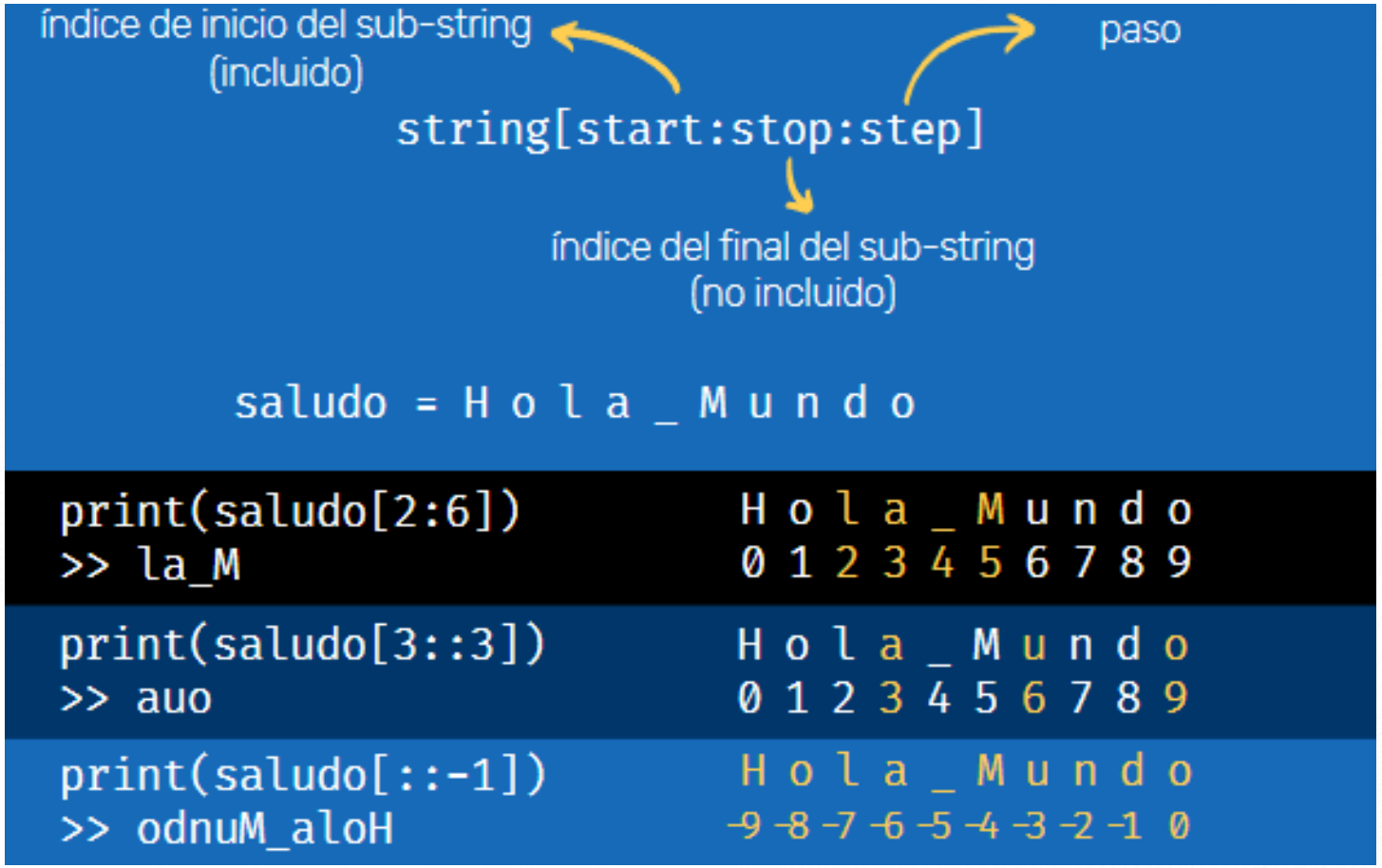
Encuentra y muestra en pantalla que carácter ocupa la quinta posición dentro de la siguiente palabra: “**ordenador**”.

Encuentra y muestra en pantalla el índice de la primera aparición de la palabra “práctica” en la siguiente frase: “**En teoría, la teoría y la práctica son los mismos. En la práctica, no lo son.**”

Encuentra y muestra en pantalla el índice de la ultima aparición de la palabra “práctica” en la siguiente frase: “**En teoría, la teoría y la práctica son los mismos. En la práctica, no lo son.**”

Extraer Sub-Strings

Podemos extraer porciones de texto utilizando las herramientas de manipulación de strings conocidas como slicing (rebanar).



Extraer Sub-Strings

Ejecuta el siguiente código en tu Pycharm y comprueba los resultados:

```
1   texto = "ABCDEFGHIJKLMN"
2   fragmento = texto[2]
3   print(fragmento) # C
4
5   texto = "ABCDEFGHIJKLMN"
6   fragmento = texto[2:5]
7   print(fragmento) # CDE
8
9   texto = "ABCDEFGHIJKLMN"
10  fragmento = texto[2:]
11  print(fragmento) #CDEFGHIJKLMN
```

Extraer Sub-Strings

Ejecuta el siguiente código en tu Pycharm y comprueba los resultados:

```
13      texto = "ABCDEFGHIJKLMN"
14      fragmento = texto[:2]
15      print(fragmento) #AB
16
17      texto = "ABCDEFGHIJKLMN"
18      fragmento = texto[2:10:2]
19      print(fragmento) #CEGI
20
21      texto = "ABCDEFGHIJKLMN"
22      fragmento = texto[::3]
23      print(fragmento) #ADGJM
```

Extraer Sub-Strings

Ejecuta el siguiente código en tu Pycharm y comprueba los resultados:

```
25      texto = "ABCDEFGHIJKLMN"
26      fragmento = texto[::-1]
27      print(fragmento) #NMLKJIHGFEDCBA
28
29      texto = "ABCDEFGHIJKLMN"
30      fragmento = texto[::-2]
31      print(fragmento) #NLJHFDB
```

Ejercicios Sub-Strings

Extrae la primera palabra de la siguiente frase utilizando slicing, y muéstrala en pantalla: frase = "**Controlar la complejidad es la esencia de la programación**“

Toma **cada tercer carácter empezando desde el noveno hasta el final de la frase**, e imprime el resultado. Frase=“**Nunca confíes en un ordenador que no puedas lanzar por una ventana**”.

Invierte la posición de todos los caracteres de la siguiente frase y muestra el resultado en pantalla. Frase=“**Es genial trabajar con ordenadores. No discuten, lo recuerdan todo y no se beben tu cerveza**”.

Métodos de String

Métodos de análisis

count( ): retorna el número de veces que se repite un conjunto de caracteres especificado.

```
x = "Hola Hola mundo".count("Hola")
print(x) # 2
```

find( ) e **index( )** retornan la ubicación (comenzando desde el cero) en la que se encuentra el argumento indicado. Difieren en que index lanza **ValueError** cuando el argumento no es encontrado, mientras **find** retorna -1.

```
x = "Hola mundo".find("world")
print(x) # -1
```

Métodos de String

Métodos de análisis

rfind( ) y **rindex( ).**Para buscar un conjunto de caracteres pero desde el final.

```
x = "C:/python36/python.exe".rfind("/")
print(x) # 11
```

startswith( ) y **endswith( )** indican si la cadena en cuestión comienza o termina con el conjunto de caracteres pasados como argumento, y retornan True o False en función de ello.

```
x = "Hola mundo".startswith("Hola")
print(x) # True
```

Métodos de String

Métodos de análisis

isdigit( ): determina si todos los caracteres de la cadena son dígitos, o pueden formar números, incluidos aquellos correspondientes a lenguas orientales.

```
x = "abc123".isdigit()
print(x) # False
```

isnumeric( ): determina si todos los caracteres de la cadena son números, incluye también caracteres de connotación numérica que no necesariamente son dígitos (por ejemplo, una fracción).

```
x = "1234".isnumeric()
print(x) # True
```

Métodos de String

Métodos de análisis

isdecimal( ): determina si todos los caracteres de la cadena son decimales; esto es, formados por dígitos del 0 al 9.

```
x = "1234".isdecimal()
print(x) # True
```

isalnum( ): determina si todos los caracteres de la cadena son alfanuméricos.

```
x = "abc123".isalnum()
print(x) # True
```

isalpha( ): determina si todos los caracteres de la cadena son alfabéticos.

```
x = "abc123".isalpha()
print(x) # False
```

Métodos de String

Métodos de análisis

islower( ): determina si todos los caracteres de la cadena son minúsculas.

```
x = "abcdef".islower()
print(x) # True
```

isupper( ): determina si todos los caracteres de la cadena son mayúsculas.

```
x = "ABCDEF".isupper()
print(x) # True
```

isprintable( ): determina si todos los caracteres de la cadena son imprimibles (es decir, no son caracteres especiales indicados por \...).

```
x = "Hola \t mundo!".isprintable()
print(x) # False
```

Métodos de String

Métodos de transformación

En realidad los strings son inmutables; por ende, todos los métodos a continuación no actúan sobre el objeto original sino que retornan uno nuevo.

capitalize( ): retorna la cadena con su primera letra en mayúscula.

```
x = "hola mundo".capitalize()
print(x) # Hola mundo
```

encode( ): codifica la cadena con el mapa de caracteres especificado y retorna una instancia del tipo bytes.

```
x = "Hola mundo".encode("utf-8")
print(x) # b'Hola mundo'
```

Métodos de String

Métodos de transformación

replace( ): reemplaza una cadena por otra.

```
x = "Hola mundo".replace( __old: "mundo",  __new: "world")
print(x) # Hola world
```

lower( ): retorna una copia de la cadena con todas sus letras en minúsculas.

```
x = "Hola Mundo!".lower()
print(x) # Hola mundo!
```

upper( ): retorna una copia de la cadena con todas sus letras en mayúsculas.

```
x = "Hola Mundo!".upper()
print(x) # HOLA MUNDO!
```

Métodos de String

Métodos de transformación

swapcase( ): cambia las mayúsculas por minúsculas y viceversa.

```
x = "Hola Mundo!".swapcase()
print(x) # hOLA mUNDO!
```

strip( ), lstrip( ) y **rstrip( )**: remueven los espacios en blanco que preceden y/o suceden a la cadena.

```
x = " Hola mundo! ".strip()
print(x) # Hola mundo!
```

Métodos de String

Métodos de transformación

Los métodos **center( )**, **ljust**( ) y **rjust**( ) alinean una cadena en el centro, la izquierda o la derecha. Un segundo argumento indica con qué caracter se deben llenar los espacios vacíos (por defecto un espacio en blanco).

```
x = "Hola".center( __width: 10,  __fillchar: "*")
print(x) # ***hola***
```

Métodos de String

Métodos de separación y unión

split( ): divide una cadena según un caracter separador (por defecto son espacios en blanco). Un segundo argumento en split( ) indica cuál es el máximo de divisiones que puede tener lugar (-1 por defecto para representar una cantidad ilimitada).

```
x = "Hola mundo!\nHello world!".split()
print(x) # ['Hola', 'mundo!', 'Hello', 'world!']
```

splitlines( ): divide una cadena con cada aparición de un salto de línea.

```
x = "Hola mundo!\nHello world!".splitlines()
print(x) # ['Hola mundo!', 'Hello world!']
```

Métodos de String

Métodos de separación y unión

partition( ): retorna una tupla de tres elementos: el bloque de caracteres anterior a la primera ocurrencia del separador, el separador mismo, y el bloque posterior.

```
x = "Hola mundo. Hello world!".partition(" ")
print(x) # ('Hola', ' ', 'mundo. Hello world!')
```

rpartition( ): opera del mismo modo que el anterior, pero partiendo de derecha a izquierda.

```
x = "Hola mundo. Hello world!".rpartition(" ")
print(x) # ('Hola mundo. Hello', ' ', 'world!')
```

Métodos de String

Métodos de separación y unión

join( ): debe ser llamado desde una cadena que actúa como separador para unir dentro de una misma cadena resultante los elementos de una lista.

```
x = ", ".join(["C", "C++", "Python", "Java"])
print(x) # C, C++, Python, Java
```

Ejercicios Métodos de Strings

Imprime el siguiente texto en mayúsculas, empleando el método específico de strings.

Une la siguiente lista en un string, separando cada elemento con un espacio. Utiliza el método apropiado de listas/strings, y muestra en pantalla el resultado.
lista_palabras = ["La","legibilidad","cuenta."]

Reemplaza en la siguiente frase: “Si la implementación es difícil de explicar, puede que sea una mala idea.” los siguientes pares de palabras:
“difícil → “”fácil”
“mala → “buena”
Y muestra en pantalla la frase con ambas palabras modificadas.

Propiedades de Strings

Esto es lo que debes tener presente al trabajar con strings en Python:

- **Son inmutables**: una vez creados, no pueden modificarse sus partes, pero sí pueden reasignarse los valores de las variables a través de métodos de strings.
- **Concatenable**: es posible unir strings con el símbolo +
- **Multiplicable**: es posible concatenar repeticiones de un string con el símbolo *.
- **Multilineales**: pueden escribirse en varias líneas al encerrarse entre triples comillas simples (''' ''') o dobles (""" """).
- **Determinar su longitud**: a través de la función len(mi_string).
- **Verificar su contenido**: a través de las palabras clave in y not in. El resultado de esta verificación es un booleano (True / False).

Ejercicios Propiedades de Strings

Concatena 15 veces el texto “Repetición” y muestra el resultado en pantalla. Por suerte, conoces que los strings son multiplicables y puedes realizar esta actividad de forma simple y elegante.

Verifica si la palabra “agua” no se encuentra en el siguiente texto. Debes imprimir el booleano:
Tierra mojada,
Mis recuerdos de viaje
Entra las lluvias

Muestra en pantalla el largo (en números de caracteres) de la palabra **electroencefalografista**.

Listas

Las listas son secuencias ordenadas de objetos. Estos objetos pueden ser datos de cualquier tipo: strings, integers, floats, booleanos, listas, entre otros. Son tipos de datos mutables.

```
lista_1 = ["C", "C++", "Python","Java"]
lista_2 = ["PHP", "SQL", "VBA","JavaScript"]
```

indexado: podemos acceder a los elementos de una lista a través de sus índices:
[inicio : fin : paso]

```
lista_1 = ["C", "C++", "Python", "Java"]
print(lista_1[1:3])
```

Listas

cantidad de elementos: a través de la propiedad len( ).

```
lista_1 = ["C", "C++", "Python", "Java"]
print(len(lista_1))
```

concatenación: sumamos los elementos de varias listas con el símbolo +

```
lista_1 = ["C", "C++", "Python", "Java"]
lista_2 = ["PHP", "SQL", "VBA","JavaScript"]
print(lista_1 + lista_2)
```

función append( ): agrega un elemento a una lista en el lugar.

```
lista_1 = ["C", "C++", "Python", "Java"]
lista_1.append("R")
print(lista_1)
```

Listas

función pop( ): elimina un elemento de la lista dado el índice, y devuelve el valor quitado.

```
lista_1 = ["C", "C++", "Python", "Java", "R"]
eliminado = lista_1.pop(4)
print(lista_1)
print("Eliminado: " + eliminado)
```

función sort( ): ordena los elementos de la lista en el lugar.

```
lista_3 = ["d", "a", "c","b", "e"]
lista_3.sort()
print(lista_3)
```

función reverse( ): invierte el orden de los elementos en el lugar.

```
lista_4 = [5, 4, 3, 2, 1]
lista_4.reverse()
print(lista_4)
```

Ejercicios con Listas

Crea una lista con 5 elementos, dentro de la variable **mi_lista**. Puedes incluir strings, booleanos, números, etc.

Agrega el elemento “motocicleta” a la siguiente lista de medios de transporte:
medios_transporte = ["avión", "auto", "barco", "bicicleta"]

Utiliza el método pop( ) para quitar el tercer elemento de la siguiente lista llamada frutas:
frutas = ["manzana", "banana", "mango", "cereza", "sandía"]
y almacénalo en una variable llamada eliminado. Utiliza métodos de listas sin alterar la linea de código ya suministrada.

Diccionarios

Los diccionarios son estructuras de datos que almacenan información en pares **clave:valor**. Son especialmente útiles para guardar y recuperar información a partir de los nombres de sus claves (no utilizan índices).
Son mutables, no están ordenados por un índice, no admite valores duplicados.

```
mi_dic = {"curso":"Python","clase":"Diccionarios"}
```

Agregar nuevos datos, o modificarlos

```
mi_dic["recursos"] = ["notas","ejercicios"]
print(mi_dic)
```

Acceso a valores a través del nombre de las claves

```
mi_dic = {"curso":"Python","clase":"Diccionarios","recursos":"Notas"}
print(mi_dic.keys())
```

```
dict_keys(['curso', 'clase', 'recursos'])
```

Diccionarios

Acceso a valores a través del nombre de los valores:

```
mi_dic = {"curso":"Python","clase":"Diccionarios","recursos":"Notas"}
print(mi_dic.values())
```

Acceso a valores a través del par, **clave:valor**

```
mi_dic = {"curso":"Python","clase":"Diccionarios","recursos":"Notas"}
print(mi_dic.items())
```

Imprimir el tipo de dato de **mi_dic**:

```
mi_dic = {"clave1":"valor1","clave2":"valor2","clave3":"valor3"}
print(type(mi_dic))
```

Diccionarios

Imprimir por completo **mi_dic**:

```
mi_dic = {"clave1":"valor1","clave2":"valor2","clave3":"valor3"}
print(mi_dic)
```

Acceder a un **valor** de un diccionario por medio de su **clave**:

```
mi_dic = {"clave1":"valor1","clave2":"valor2","clave3":"valor3"}
valor = mi_dic["clave1"]
print(valor)
```

```
valor1
```

Mas ejemplos:

```
persona = {'nombre':'Felipito','apellido':'Martinez','email':'felo@correo.com','telefono':'6678956'}
consulta = persona['email']
print(consulta)
```

Diccionarios

Los diccionarios pueden almacenar listas completas, así accedemos a la lista que se encuentra asociada a la clave2.

```
dic = {'clave1':55,'clave2':[10,20,30],'clave3':{'s1':100,'s2':200}}
print(dic['clave2'])
```

Así accedemos a uno de los elemento que se encuentra dentro de la lista asociada a la clave 2

```
dic = {'clave1':55,'clave2':[10,20,30],'clave3':{'s1':100,'s2':200}}
print(dic['clave2'][1])
```

```
C:\PycharmPr
20
```

Diccionarios

Imprimir el valor del diccionario asociado a la clave3 según su clave “s2”

```
dic = {'clave1':55,'clave2':[10,20,30],'clave3':{'s1':100,'s2':200}}
print(dic['clave3']['s2'])
```

Imprimir el valor 2 de la clave2, y convertirlo en mayúsculas:

```
dic = {'clave1':['a','b','c'],'clave2':['d','e','f']}
print(dic['clave2'][1].upper())
```

Ejercicios con Diccionarios

Crea un diccionario llamado **mi_dic** que almacene la siguiente información de una persona:

- **nombre**: Karen
- **apellido**: Jurgens
- **edad**: 35
- **ocupacion**: Periodista

Los nombres de las claves y valores deben ser iguales a lo solicitado.

Ejercicios con Diccionarios

Crea una función **print** que devuelva el segundo **item** de la lista llamada **points2**, dentro del siguiente diccionario.

mi_dict = {"valores_1":{"v1":3,"v2":6},"puntos":{"points1":9,"points2":[10,300,15]}}
print(mi_dict[])

Si el valor 300 cambiara en el futuro, el código debería funcionar igual para devolver el valor que se encuentre en esa misma posición. Para ello, deberás hacer referencia a los nombres de las claves y/o índices según corresponda.

Ejercicios con Diccionarios

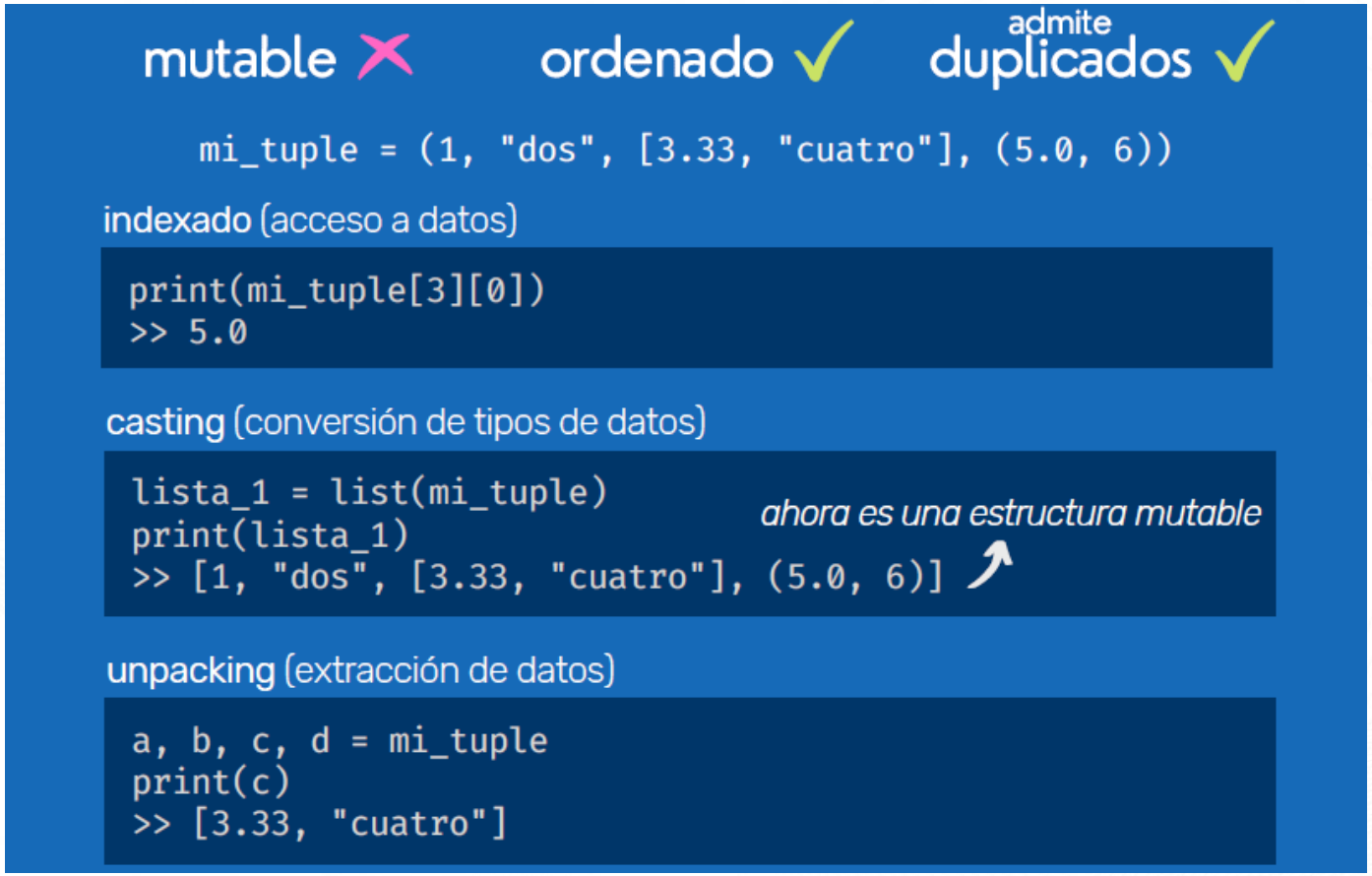
Actualiza la información de nuestro diccionario llamado mi_dic (reasignando nuevos valores a las claves según corresponda), y agrega una nueva clave llamada "pais" (sin tilde). Los nuevos datos son:

- **nombre**: Karen
- **apellido**: Jurgens
- **edad**: 36
- **ocupacion**: Editora
- **pais**: Colombia

Para ello, no debes cambiar la línea de código ya escrita, sino actualizar los valores mediante métodos de diccionarios.

Tuples

Los tuples o tuplas, son estructuras de datos que almacenan múltiples elementos en una única variable. Se caracterizan por ser ordenadas e inmutables. Esta característica las hace más eficientes en memoria y a prueba de daños.



Tuples

Ejecuta los siguientes códigos en tu Pycharm y comprueba los resultados.

```
mi_tuple = (1,2,3,4)
print(type(mi_tuple))
print(mi_tuple)
```

```
t = (1,2,3)
x,y,z = t
print(x,y,z)
```

```
mi_tuple = (1,2,(10,20),4)
print(mi_tuple[2]) # (10, 20)
```

```
mi_tuple = (1,2,3,4)
print(mi_tuple[0])
```

```
t = (1,2,3,1)
print(t.count(1))
```

```
mi_tuple = (1,2,(10,20),4)
print(mi_tuple[2][0]) # 10
```

```
mi_tuple = (1,2,3,4)
print(mi_tuple[-2])
```

```
mi_tuple = (1,2,(10,20),4)
print(type(mi_tuple))
mi_tuple = list(mi_tuple)
print(type(mi_tuple)) # <class 'list'>
```

```
mi_tuple = (1,2,3,4)
mi_tuple[0] = 5 # Esto dara un error
print(mi_tuple)
```

Ejercicios con Tuples

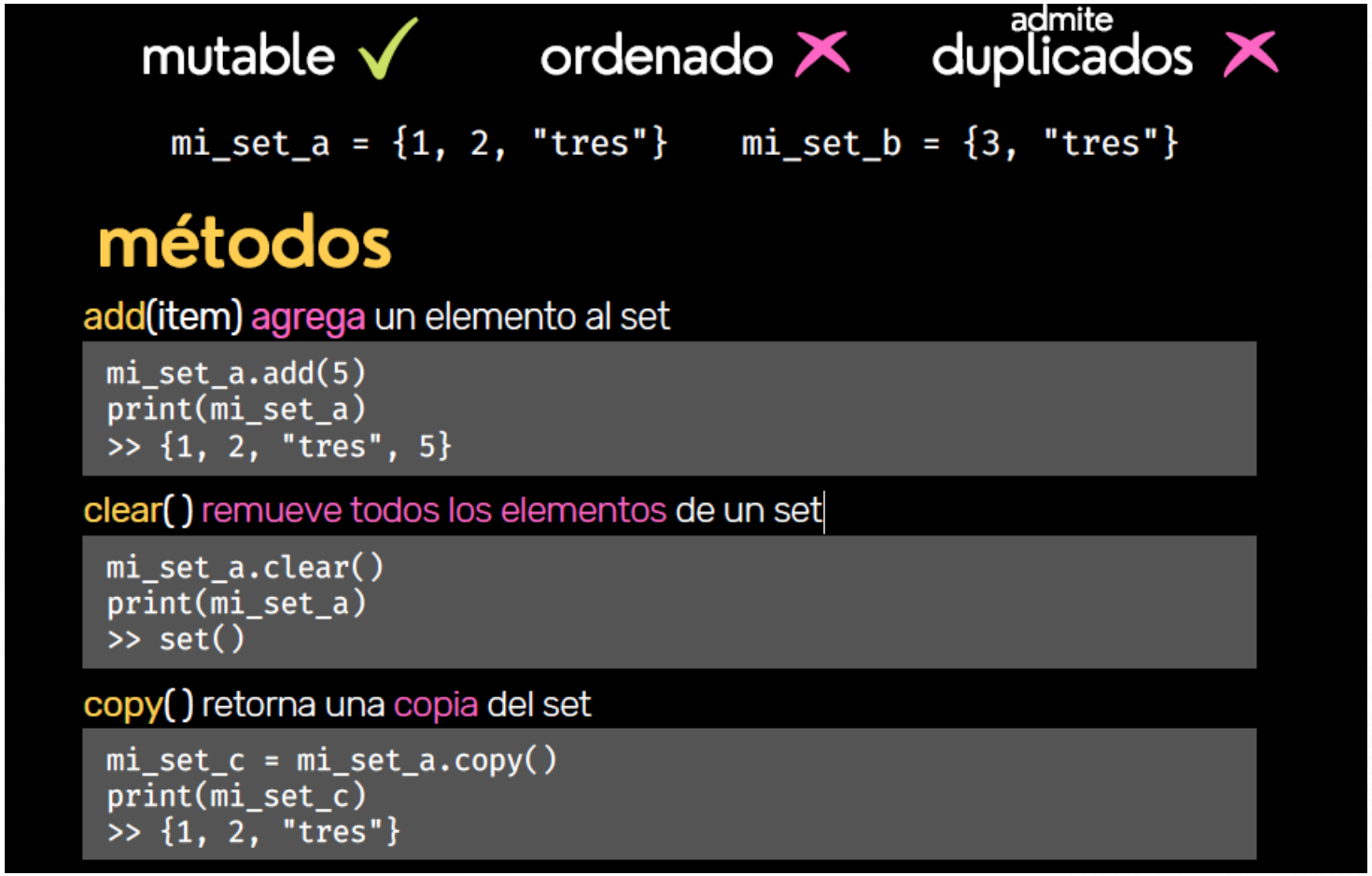
Utiliza un método de tuplas para contar la cantidad de veces que aparece el valor 2 en la siguiente tupla, y muestra el resultado (integer) en pantalla:
mi_tupla = (1, 2, 3, 2, 3, 1, 3, 2, 3, 3, 3, 1, 3, 2, 2, 1, 3, 2)

Convierte a lista la siguiente tupla, y almacénala en una variable llamada **mi_lista**.
mi_tupla = (1, 2, 3, 2, 3, 1, 3, 2)

Extrae los elementos de la siguiente tupla en cuatro variables: a, b, c, d
mi_tupla = (1, 2, 3, 4)

Sets

Los sets son otro tipo de estructuras de datos. Se diferencian de listas, tuplas y diccionarios porque son una colección mutable de elementos inmutables, no ordenados y sin datos repetidos.



Sets

```
mi_set_a = {1, 2, "tres"}    mi_set_b = {3, "tres"}
```

difference(set) retorna el set formado por todos los elementos que únicamente existen en el set A

```
mi_set_c = mi_set_a.difference(mi_set_b)
print(mi_set_c)
>> {1, 2}
```

difference_update(set) remueve de A todos los elementos comunes a A y B

```
mi_set_a.difference_update(mi_set_b)
print(mi_set_a)
>> {1, 2}
```

discard(item) remueve un elemento del set

```
mi_set_a.discard("tres")
print(mi_set_a)
>> {1, 2}
```

intersection(set) retorna el set formado por todos los elementos que existen en A y B simultáneamente.

```
mi_set_c = mi_set_a.intersection(mi_set_b)
print(mi_set_c)
>> {'tres'}
```

Sets

```
mi_set_a = {1, 2, "tres"}    mi_set_b = {3, "tres"}
```

intersection_update(set) mantiene únicamente los elementos comunes a A y B

```
mi_set_b.intersection_update(mi_set_a)
print(mi_set_b)
>> {"tres"}
```

isdisjoint(set) devuelve True si A y B no tienen elementos en común

```
conjunto_disjunto = mi_set_a.isdisjoint(mi_set_b)
print(conjunto_disjunto)
>> False
```

issubset(set) devuelve True si todos los elementos de B están presentes en A

```
es_subset = mi_set_b.issubset(mi_set_a)
print(es_subset)
>> False
```

issuperset(set) devuelve True si A contiene todos los elementos de B

```
es_superset = mi_set_a.issuperset(mi_set_b)
print(es_superset)
>> False
```

Sets

```
mi_set_a = {1, 2, "tres"}    mi_set_b = {3, "tres"}
```

pop( ) elimina y retorna un elemento al azar del set

```
aleatorio = mi_set_a.pop()
print(aleatorio)
>> {2}
```

remove(item) elimina un item del set, y arroja error si no existe

```
mi_set_a.remove("tres")
print(mi_set_a)
>> {1, 2}
```

symmetric_difference(set) retorna todos los elementos de A y B, excepto aquellos que son comunes a los dos

```
mi_set_c = mi_set_b.symmetric_difference(mi_set_a)
print(mi_set_c)
>> {1, 2, 3}
```

symmetric_difference_update(set) elimina los elementos comunes a A y B, agregando los que no están presentes en ambos a la vez

```
mi_set_b.symmetric_difference_update(mi_set_a)
print(mi_set_b)
>> {1, 2, 3}
```

Sets

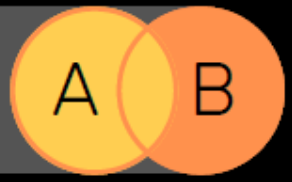
```
mi_set_a = {1, 2, "tres"}    mi_set_b = {3, "tres"}
```

union(set) retorna un set resultado de combinar A y B (los datos duplicados se eliminan)

```
mi_set_c = mi_set_a.union(mi_set_b)
print(mi_set_c)
>> {1, 2, 3, 'tres'}
```

update(set) inserta en A los elementos de B

```
mi_set_a.update(mi_set_b)
print(mi_set_a)
>> {1, 2, 3, 'tres'}
```



Ejercicios con Sets

Une los siguientes sets en uno solo, llamado mi_set_3:
{1, 2, "tres", "cuatro"}
{"tres", 4, 5}

Elimina un elemento al azar del siguiente set, utilizando métodos de sets.
sorteo = {"Camila", "Margarita", "Axel", "Jorge", "Miguel", "Mónica"}

Agrega el nombre Damián al siguiente set, utilizando métodos de sets:
sorteo = {"Camila", "Margarita", "Axel", "Jorge", "Miguel", "Mónica"}

Booleanos

Los booleanos son tipos de datos binarios (True/False), que surgen de operaciones lógicas, o pueden declararse explícitamente.

operadores lógicos

```
==  igual a
!=  diferente a
 >  mayor que
 <  menor que
>=  mayor o igual que
<=  menor o igual que

and y (True si dos declaraciones son True)
 or o (True si al menos una declaración es True)
not no (invierte el valor del booleano)
```

Ejercicios con Booleanos

Realiza una comparación que arroje como resultado un booleano y almacena el resultado (True/False) en una variable llamada prueba

---

Verifica si 17834/34 es mayor que 87*56 y muestra el resultado (booleano) en pantalla utilizando print()

Verifica si la raíz cuadrada de 25 es igual a 5 y muestra el resultado (booleano) en pantalla utilizando print()