PREGRADO

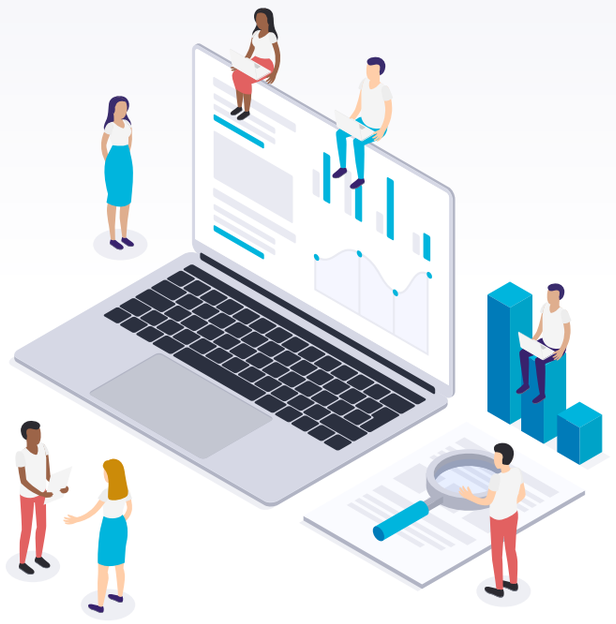
UNIDAD 1 | SEMANA 3

STRING, LISTAS, TUPLAS, FUNCIONES APLCABLES

1ACC0201| Programación Orientada a Objetos

Creando listas con el constructor list

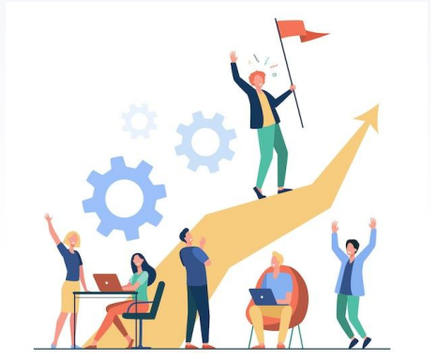
String, listas, tuplas



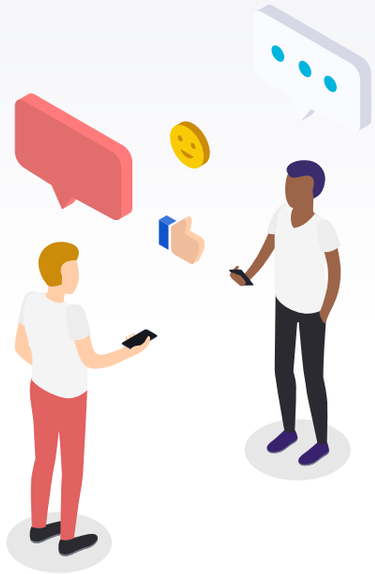
2024-01-Tema 04
Parte 1

Logro

Al final de la sesión el alumno entiende el uso de string, listas y tuplas, así como pasar las listas como parámetros de funciones y parámetros variables de funciones



listas



Listas

- Es una colección ordenada y modificable de diferentes tipos de datos.
- Permite miembros duplicados.
- En Python las listas se escriben entre corchetes.
- Los corchetes indican el inicio y el final de la lista
- Los elementos están separados por comas.
- Los elementos de las listas pueden ser cualquier tipo de dato: str, int, float, string, lista, bool
- Por lo tanto, puede haber listas dentro de las listas
- Una lista puede estar vacía

Creando Listas

```
lista1 = [1, 2, 3]
print(type(lista1))
print(lista1)
```

```
<class 'list'>
[1, 2, 3]
```

```
lista2 = [1, ['a', 'e', 'i', 'o', 'u'], 8.9, 'hola']
print(lista2)

lista3 = [[1, 2], [4, 5], [7, 4]]
print(lista3)
```

```
[1, ['a', 'e', 'i', 'o', 'u'], 8.9, 'hola']
[[1, 2], [4, 5], [7, 4]]
```

Creando listas con el constructor list

```
# crea una lista desde el string "aeiou"
c = list("aeiou")
print(type(c))
print(c)
```

```
<class 'list'>
['a', 'e', 'i', 'o', 'u']
```

Listas: Índices y slicing

Al igual que con los string, cuando se crea una lista también se crean índices para cada uno de los elementos de una lista

0	1	2	3	4	5	6
Alberto	Juan	Mario	Renan	Luis	Javier	Humberto
-7	-6	-5	-4	-3	-2	-1

Por lo tanto, al igual que con los string se puede aplicar el slicing

Listas: Índices y slicing

0	1	2	3	4	5	6
Alberto	Juan	Mario	Renan	Luis	Javier	Humberto
-7	-6	-5	-4	-3	-2	-1

```
lista = ["Alberto", "Juan", "Mario", "Renan", "Luis", "Javier", "Humberto"]

print(lista[2:4])
print(lista[-2])
print(lista[::-1])
```

['Mario', 'Renan']
Javier
['Humberto', 'Javier', 'Luis', 'Renan', 'Mario', 'Juan', 'Alberto']

Listas: el método append()

```
lista = ["Alberto", "Juan", "Mario", "Renan", "Luis", "Javier", "Humberto"]

lista.append("Miguel")
print(lista)
```

['Alberto', 'Juan', 'Mario', 'Renan', 'Luis', 'Javier', 'Humberto', 'Miguel']

Modificando y eliminando elementos

```
lis1 = [1, 2, 3, 4, 5, 6, 7, 8]
print(lis1)

lis1[1] = 20
print(lis1)

# usando el método pop() sín índice, se elimina el último elemento de la lista
lis1.pop()
print(lis1)

# Usando un índice, se elimina un elemento en particular

lis1.pop(1)
print(lis1)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 20, 3, 4, 5, 6, 7, 8]
[1, 20, 3, 4, 5, 6, 7]
[1, 3, 4, 5, 6, 7]
```

Copiando listas

En Python, para copiar una lista no basta con poner una sentencia como la siguiente:
Lista2 = lista1
Veamos un ejemplo

```
# Esto sólo pone otra etiqueta al objeto tipo lista
num1 = [1, 2, 3, 4]
num2 = num1
print(id(num1))
print(id(num2))
print(num1 is num2)
num1[0] = 10
print(num1)
print(num2)
```

```
1530503316096
1530503316096
True
[10, 2, 3, 4]
[10, 2, 3, 4]
```

Luego num1 y num2 son etiquetas del mismo objeto

Copiando listas

Copiar usando slicing

```
# Usando slicing
num1 = [1, 2, 3, 4]
num2 = num1[:]
print(id(num1))
print(id(num2))
print(num1 is num2)
num1[0] = 20
print(num1)
print(num2)
```

```
1530503405760
1530503315520
False
[20, 2, 3, 4]
[1, 2, 3, 4]
```

Los resultados muestran que ahora tenemos dos objetos diferentes

Copiando listas

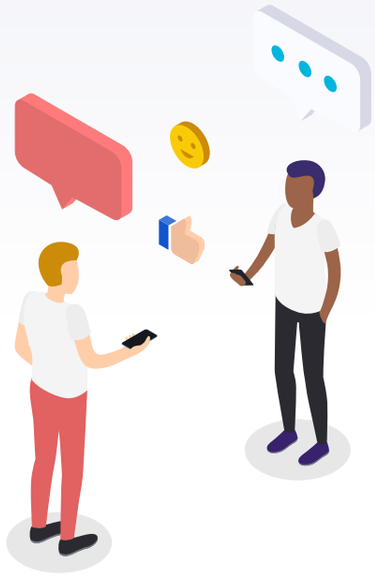
Copiar usando el método copy()

```
num1 = [1, 2, 3, 4]
num2 = num1.copy()
print(id(num1))
print(id(num2))
print(num1 is num2)
num1[0] = 20
print(num1)
print(num2)
```

```
1530503319744
1530503320192
False
[20, 2, 3, 4]
[1, 2, 3, 4]
```

En este caso también tenemos dos objetos diferentes

tuplas



tuplas

- Las tuplas son un conjunto de datos que tienen las siguientes características
  - Son iterables
  - Son inmutables.
  - Cualquier tipo de dato puede ser elemento de una tupla
- Para crearlas se puede usar paréntesis

```
# Creando Tuplas
mitupla1 = (1, 2, 3)
print(type(mitupla1))
print(mitupla1)
```

```
<class 'tuple'>
(1, 2, 3)
```

```
# Los paréntesis son opcionales
miTupla2 = 1, 2, 3
print(type(miTupla2))
print(miTupla2)
```

```
<class 'tuple'>
(1, 2, 3)
```

Acceso a elementos y Slicing en tuplas

- Las tuplas al igual que las listas, tienen dos grupos de índices, positivos y negativos y por lo tanto también se puede usar slicing

0	1	2	3	4	5
"Hola"	24	[1, 2]	("Python","3.10.7")	3.14159	True
-6	-5	-4	-3	-2	-1

Acceso a elementos y slicing en tuplas

- Veamos los algunos ejemplos

```
miTupla = ("Hola", 24, [1, 2], ("Python", "3.10.4"), 3.14159, 5 > 3)
print(miTupla)

print(miTupla[3])
print(miTupla[3:])
print(miTupla[2][0])
print(miTupla[::-1])
```

```
('Hola', 24, [1, 2], ('Python', '3.10.4'), 3.14159, True)
('Python', '3.10.4')
(('Python', '3.10.4'), 3.14159, True)
1
(True, 3.14159, ('Python', '3.10.4'), [1, 2], 24, 'Hola')
```

Analice y discuta los resultados
¿Por qué el tercer resultado es 1?

Tuplas: Empaquetado y desempaquetado

- Si a una variable se le asigna una secuencia de valores separados por comas, el valor de esa variable será la tupla formada por todos los valores asignados. A esta operación se la denomina **empaquetado** de tuplas.

```
# Ejemplo de empaquetado de tuplas
a = 125
b = 3.14159
c = "Ana"
d = a, b, c
print(type(d))
print(d)
print(len(d))  # len también funciona en
tuplas
```

```
<class 'tuple'>
(125, 3.14159, 'Ana')
3
```

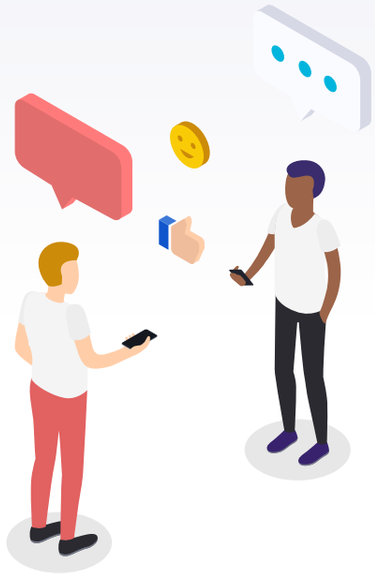
Tuplas: Empaquetado y desempaquetado

- Si se tiene una tupla de longitud k, se puede asignar la tupla a k variables distintas y en cada variable quedará una de las componentes de la tupla. A esta operación se la denomina **desempaquetado** de tuplas.

```
# Ejemplo de desempaquetado de tuplas
t = 125, 3.14159, "Ana"
x, y, z = t
print(x, y, z)
```

```
125 3.14159 Ana
```

Bucles for



La función len()

- Esta función también se aplica a listas y tuplas
- Devuelve el número de elementos de una lista o una tupla

```
cadena = "Hola mundo"
lista1 = [1, 2, 3, 4, 5, 6.5, 7, 8, cadena]
tupla1 = (1, 2, 3, 4, 5, 6, cadena, lista1)
print(len(cadena))
print(len(lista1))
print(len(tupla1))
```

```
10
9
8
```

Bucles “for” en listas y tuplas

- Los bucles **for** en Python son un tipo especial de sentencia que se utiliza para el recorrido secuencial.
- El bucle **for** de Python se utiliza para iterar sobre un elemento iterable como un string, una tupla, una lista, o un diccionario.
- En Python, no existe un bucle **for** al estilo C++, es decir, for (i=0; I <n; i++)
- La sintaxis básica es la siguiente:

```
for variable in iterable:
    # código python
```

- Hay dos palabras clave, **for** e **in**
- Observe también la indentación que señala el bloque de código que pertenece al **for**

Recorriendo strings, listas y tuplas con for

```
holaMundo = "Hola, mundo."
for caracter in holaMundo:
    print(caracter)
```

```
frutas = ["manzana", "pera", "mango", "sandía", "melón", "plátano"]
for elemento in frutas:
    print(elemento)
```

```
myTupla = (1, 2, 3, 4, 5)
for valor in myTupla:
    print(valor)
```

Ejecutar y analizar los resultados

Usando **in** como un operador lógico

En string permite saber si un substring está dentro de un string

```
x = "Curso Progamación Orientada a Objetos"
if "P" in x:
    print("Correcto")
else:
    print("Errado")
```

Resultado

Correcto

```
x = "Curso Progamación Orientada a Objetos"
if "Programacion" in x:
    print("Correcto")
else:
    print("Errado")
```

Resultado

Errado

```
x = "Curso Progamación Orientada a Objetos"
if "Objeto" in x:
    print("Correcto")
else:
    print("Errado")
```

Resultado

Correcto

enumerate()

- En Python, un bucle **for** generalmente se escribe como un bucle sobre un objeto iterable.
- Esto significa que no necesita una variable de recuento para acceder a los elementos del iterable.
- A veces, sin embargo, desea tener una variable que cambie en cada iteración del bucle.
- En lugar de crear e incrementar una variable, se puede usar **enumerate()** de Python para obtener un **contador** y el **valor** del iterable al mismo tiempo.
- La sintaxis es la siguiente:

```
enumerate(iterable, start=0)
```

Donde:

- Iterable: cualquier objeto que admita iteración.
- start: el valor del índice desde el cual se iniciará el contador, por defecto es 0

enumerate()

```
cadena = "Hola mundo"

for i, valor in enumerate(cadena):
    print(i, valor)

for i, valor in enumerate(cadena, 1):
    print(i, valor)
```

Ejecutar y analizar los resultados

```
frutas = ["manzana", "pera", "mango", "sandía", "melón", "plátano"]
for i, fruta in enumerate(frutas):
    print(i, fruta)

print()

for i, fruta in enumerate(frutas, 1):
    print(i, fruta)
```

```
frutas = ("manzana", "pera", "mango", "sandía", "melón", "plátano“)
for i, fruta in enumerate(frutas):
    print(i, fruta)

print()

for i, fruta in enumerate(frutas, 1):
    print(i, fruta)
```

Usando in como operador lógico

En listas permite saber si un valor es un elemento de la lista

```
colores = ["azul", "blanco", "negro"]
nuevo_color = "marrón"
if nuevo_color in colores:
    print("Elemento ya existe en la lista")
else:
    colores.append(nuevo_color)

print(colores)
```

['azul', 'blanco', 'negro', 'marrón']

```
colores = ["azul", "blanco", "negro"]
nuevo_color = "blanco"
if nuevo_color in colores:
    print("Elemento ya existe en la lista")
else:
    colores.append(nuevo_color)

print(colores)
```

Elemento ya existe en la lista
['azul', 'blanco', 'negro']

Usando in como operador lógico

- En tuplas permite saber si un valor es un elemento de la tupla

```
miTupla = (1, 2, 3, 4, 5)

if 3 in miTupla:
    print("Correcto")
```

```
Correcto
```

Creando secuencia de números: range()

Sintaxis de range:

range(inicio, final, paso)

- Si se omite el inicio, comienza con el valor 0
- Si se omite el paso, el valor por defecto es uno
- El valor final nunca está incluido en el iterador generado

```
# Ejemplo de len y range
holaMundo = "Hola, mundo."
print(len(holaMundo))
for i in range(len(holaMundo)):
    print(holaMundo[i])
```

```
12
H
o
l
a
,

m
u
n
d
o
.
```

Creando secuencia de números: range()

Sintaxis de range:
range(inicio, final, paso)

```
# Generar una lista con "n" números pares

def numPares(n):
    lista1 = list(range(2, n*2+1, 2))
    print(lista1)


# Programa principal
numero = 13
numPares(numero)
```

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26]

Números aleatorios

Número aleatorios enteros

Sintaxis: randint(a, b)

La función **randint(a, b)**, genera números aleatorios enteros entre a y b, ambos (a y b) incluidos

```
# usando generación de números aleatorios enteros
# Se una la biblioteca randon
import random

for i in range(5):
    print(random.randint(1, 3), end=" ")
```

3 2 1 1 2

```
# Otra forma de hacer el import es la siguiente
from random import randint

for i in range(5):
    print(randint(1, 3), end=" ")
```

1 1 2 3 1

Generando listas con números aleatorios y usando range

```
from random import randint

lista3 = []
for i in range(10):
    lista3.append(randint(1, 20))

print(lista3)
```

[1, 9, 6, 19, 14, 12, 19, 1, 10, 18]

Funciones de usuario – pasar parámetros por valor y por referencia

Argumentos de las funciones

- **Paso por valor y por referencia**:
  - En muchos lenguajes de programación existen los conceptos de paso por valor y por referencia que aplican a la hora de como trata una **función** a los **parámetros que se le pasan como entrada**. Su comportamiento es el siguiente:
    - Si usamos un parámetro pasado **por valor**, se creará una **copia** local **de la variable**, lo que implica que cualquier modificación sobre la misma no tendrá efecto sobre la original.
    - Con una variable pasada **por referencia**, se actuará directamente sobre la variable pasada, por lo que las modificaciones afectarán a la variable original.
- En Python el comportamiento estará definido por el tipo de variable con la que estamos tratando
- Las variables **numéricas, string** se pasan **por valor**
- Las **listas**, dado que son mutables, se pasan **por referencia**

Funciones de usuario – pasar parámetros por valor y por referencia

Numéricas y string

```
def func_numerica(entrada):
    entrada = entrada * 2
    print("func_numerica valor de x = ", entrada)


def func_cadena(entrada):
    entrada = entrada * 2
    print("func_cadena valor de y = ", entrada)


# Programa principal
x = 10
func_numerica(x)
print("Programa principal valor de x = ", x)
y = "Hola Mundo "
func_cadena(y)
print("Programa principal valor de y = ", y)
```

Pregunta: ¿Cómo se explican los resultados?

func_numerica valor de x = 20
Programa principal valor de x = 10
func_cadena valor de y = Hola Mundo Hola Mundo
Programa principal valor de y = Hola Mundo

Funciones de usuario – pasar parámetros por valor y por referencia

Listas, estas pasan por referencia

```
def funcion(entrada):
    entrada.append(40)
    print("De la función          ",
id(entrada))


# Programa principal
lista1 = [10, 20, 30]
print("Del programa principal", id(lista1))
funcion(lista1)
print(lista1)
```

```
Del programa principal 1917755610048
De la función          1917755610048
[10, 20, 30, 40]
```

Pregunta: ¿Cómo se explican los resultados?

Funciones de usuario – pasar parámetros por valor y por referencia

Listas, pasando una lista por valor

```
def funcion(entrada):
    entrada.append(40)
    print("Función id = ", id(entrada))
    print(entrada)


# Programa principal
lista1 = [10, 20, 30]
print("Principal id = ", id(lista1))
funcion(lista1[:])
print(lista1)
```

```
Principal id =  2756858898688
Función   id =  2756858892864
[10, 20, 30, 40]
[10, 20, 30]
```

Pregunta: ¿Cómo se explican los resultados?

Pasando argumentos de cantidad variable

- Para pasar argumentos de cantidad variable se usa un * antes del nombre del parámetro
- La función recibe el valor como una tupla

```
def suma(*numeros):
    print(type(numeros))
    return sum(numeros)


# Programa principal
res = suma(1, 2, 4, 10, 100, -34)
print(res)
res = suma(1, 2, 4, 10)
print(res)
```

```
<class 'tuple'>
83
<class 'tuple'>
17
```

La sentencia print(type(numeros)) solo se ha puesto para remarcar que se recibe los datos como una tupla

Métodos y funciones de strings/listas/tuplas

Método/Función	Descripción	Aplica
len(iterable)	La función **len()**, devuelve el número de caracteres del string o número de elementos de una lista o tupla	**string/lista/tupla**
string.**find(valor, inicio, final)**	El método **find()** devuelve el índice donde comienza el valor buscado, si no lo encuentra devuelve -1 **valor**: Requerido, es el valor a buscar **inicio**: Opcional, dónde comenzar la búsqueda, por defecto es 0 **fin**: Opcional, dónde termina la búsqueda, por defecto es el final del string	**string**
string.**isdigit()**	El método **isdigit()** devuelve True si cada carácter de la cadena es un número	**string**
string.**lower()**	El método **lower()** devuelve una cadena con cada letra del original en minúsculas	**string**
string.**replace(antiguo, nuevo, contador)**	El método **replace()** busca un string y lo reemplaza por otro **anterior**: Obligatorio. El string que se va a buscar **nuevo**: Obligatorio. El string que reemplazará al anterior **contador**: Opcional. Especifica cuántas apariciones del valor antiguo desea reemplazar. El valor predeterminado es todas las apariciones	**string**
string.**strip(caracteres)**	El método **strip()** elimina los **caracteres** entre paréntesis al inicio y al final del string. Si no se pone ningún carácter elimina espacios al inicio y al final de la cadena	**string**
string.**title()**	El método **title()** devuelve una cadena con la letra inicial de cada palabra en mayúscula	**string**
string.**upper()**	El método **upper()** devuelve una cadena con cada letra del original en mayúsculas	**string**

Métodos y funciones de strings/listas/tuplas

Método/Función	Descripción	Aplica
iterable.**count(valor)**	El método **count()** cuenta cuantas veces aparece el valor en el iterable (string/lista/tupla) **valor**: El valor buscado	**string/lista/tupla**
max(lista/tupla**)**	La función **max()** devuelve el mayor valor de una lista/tupla	**lista/tupla**
min(lista/tupla**)**	La función **min()** devuelve el mayor valor de una lista/tupla	**lista/tupla**
String_separador.**join(iterable)**	El método **join()** devuelve una cadena uniendo todos los elementos de un iterable (lista, cadena, tupla, diccionario), separados por el string_separador.	**lista/tupla**
sorted(iterable, key=Func, **reverse**=True/False**)**	La función **sorted()** **genera como resultado una lista** con los elementos ordenados. Por defecto **reverse es False**, eso significa que el **ordenamiento es de menor a mayor**, cuando **reverse es True**, se **ordena de mayor a menor**. **key**: Opcional. **Una función** que se ejecutará para decidir la orden. El valor predeterminado es Ninguno	**lista/tupla**
sum(lista/tupla)	La función **sum()** adiciona los elementos de una lista o tupla cuando los elementos son numéricos	**lista/tupla**
lista.**sort(reverse=True|False, key=miFunc)**	El método **sort()** ordena la lista sobre sí misma, no crea otra lista, por defecto **reverse es False** y esto ordenará la lista de menor a mayor, si **reverse es True**, ordenará la lista de mayor a menor **key**: Opcional, una función para especificar los criterios de clasificación	**lista**

find(valor, inicio, final)

```
txt = "Hola, bienvenido a mi mundo"
x = txt.find("e")
print(x)
```

8

```
txt = "Hola, bienvenido a mi mundo"
x = txt.find("e", 9, 15)
print(x)
```

11

```
txt = "Hola, bienvenido a mi mundo"
x = txt.find("q")
print(x)
```

-1

Analizar los resultados

isdigit(), lower(), upper()

```
# isdigit()
dni = "08804832"
if dni.isdigit():
    print("correcto")
else:
    print("error")
```

correcto

```
# lower()
cadena = "Hola Mundo"
print(id(cadena))
cadena = cadena.lower()
print(cadena)
print(id(cadena))
```

primer id 1978083617584
hola mundo
segundo id 1978083422256

Analice y discuta los resultados del segundo ejemplo

```
# upper()
cadena = "Hola Mundo"
print("primer id", id(cadena))
cadena = cadena.upper()
print(cadena)
print("segundo id", id(cadena))
```

primer id 2977384994800
HOLA MUNDO
segundo id 2977385001072

Analice y discuta los resultados del tercer ejemplo ejemplo

title(), strip()

```
# title
nombre = "juan esteban rodriguez"
nombre = nombre.title()
print(nombre)
```

Juan Esteban Rodriguez

```
# strip()
z = "   cadena de prueba   "
print(z, len(z))
z = z.strip()
print(z, len(z))
```

```
   cadena de prueba    22
cadena de prueba 16
```

```
# strip()
var1 = " "
if var1.strip() == "":
    print("La variable está vacía o en blanco")
else:
    print("La variable no está vacía")
```

La variable está vacía o en blanco

```
# strip()
holaMundo = "Hola Mundo!"
holaMundo = holaMundo.strip(" Mundo!")
print(holaMundo)
```

Hola

replace(antiguo, nuevo, contador)

```
# replace
cadenaSinGuion = "Hola Mundo Nuevo"
cadenaConGuion = cadenaSinGuion.replace(" ", "-")
print(cadenaConGuion)
```

Hola-Mundo-Nuevo

```
# replace
cadenaSinGuion = "Hola Mundo Nuevo"
cadenaConGuion = cadenaSinGuion.replace(" ", "-", 1)
print(cadenaConGuion)
```

Hola-Mundo Nuevo

count()

```
# count en string
cadena = "Hola alumnos de la UPC, Programación Orientada a Objetos"
print("Cantidad de 'o'", cadena.count("o"))
print("Cantidad de 'n'", cadena.count("n"))
```

Cantidad de 'o' 4
Cantidad de 'n' 3

¿Por qué no cuenta las mayúsculas y las acentuadas?

```
# count en listas
lista1 = [1, 2, 3, 4, 5, 6, 7, 8, 2, 3, 1, 4, 5, 6]
print("Cantidad de números 3 en la lista", lista1.count(3))
```

Cantidad de números 3 en la lista es 2

Ejercicio para el alumno: Haga la prueba con tuplas

sum(), max(), min()

```python
# sum, max, min
lista1 = [1, 2, 3, 4, 5, 6, 7, 8, 2, 3, 1, 4, 5, 6]

# Imprime suma, máximo y mínimo
print("La suma es", sum(lista1))
print("El valor máximo es", max(lista1))
print("El valor mínimo es", min(lista1))

# imprime promedio
print("El promedio es", sum(lista1) / len(lista1))
```

La suma es 57
El valor máximo es 8
El valor mínimo es 1
El promedio es 4.071428571428571

join()

```
# join
miTupla = ("Juan", "Pedro", "Vicky")
x = "#".join(miTupla)
print(x)
```

Juan#Pedro#Vicky

```
# join
miLista = ["Juan", "Pedro", "Vicky"]
x = "#".join(miLista)
print(x)
```

Juan#Pedro#Vicky

```
# join
miCadena = "Programación orientada a Objetos"
x = "-".join(miCadena)
print(x)
```

P-r-o-g-r-a-m-a-c-i-ó-n- -o-r-i-e-n-t-a-d-a- -a- -O-b-j-e-t-o-s

sorted(iterable, key=Func, reverse=True/False)

```
# Ordenamiento ascendente
numeros = [4, 2, 12, 8]
# ordena la lista en forma ascendente
numOrdAsc = sorted(numeros)
print(numOrdAsc)
```

[2, 4, 8, 12]

```
# Ordenamiento descendente
numeros = [4, 2, 12, 8]
# ordena la lista en forma descendente
numOrdDesc = sorted(numeros, reverse=True)
print(numOrdDesc)
```

[12, 8, 4, 2]

```
# Ordenamiento con key
frutas = ['manzana', 'mango', 'kiwi', 'granadilla']
# Ordena la lista por la longitud de los strings
frutasOrdenadas = sorted(frutas, key=len)
print('Lista ordenada:', frutasOrdenadas)
```

Lista ordenada: ['kiwi', 'mango', 'manzana', 'granadilla']

```
# Ordenamiento con key
def mifunc(n):
    return abs(10-n)


a = (5, 3, 1, 11, 2, 12, 17)
x = sorted(a, key=mifunc)
print(x)
```

[11, 12, 5, 3, 17, 2, 1]

Analice y discuta este resultado

sort()

```
# sort() ordena la lista sobre si misma
num1 = [1, 2, 3, 7, 0, 23]
print("Id antes del sort", id(num1))
num1.sort()
print(num1)
print("Id después del sort", id(num1))
```

Id antes del sort 1370090216128
[0, 1, 2, 3, 7, 23]
Id después del sort 1370090216128

```
# sort() ordena la lista sobre si misma
num1 = [1, 2, 3, 7, 0, 23]
print("Id antes del sort", id(num1))
num1.sort(reverse=True)
print(num1)
print("Id después del sort", id(num1))
```

Id antes del sort 2985824046144
[23, 7, 3, 2, 1, 0]
Id después del sort 2985824046144

```
# sort() ordena la lista sobre si misma
# Ordenamiento con key
frutas = ['manzana', 'mango', 'kiwi', 'granadilla']
# Ordena la lista por la longitud de los strings
frutas.sort(key=len)
print('Lista ordenada:', frutas)
```

Lista ordenada: ['kiwi', 'mango', 'manzana', 'granadilla']

```
# sort() ordena la lista sobre si misma
# Ordenamiento con key
def mifunc(n):
    return abs(10-n)


a = [5, 3, 1, 11, 2, 12, 17]
a.sort(key=mifunc)
print(a)
```

[11, 12, 5, 3, 17, 2, 1]

Analice y discuta los resultados

PREGRADO

Ingeniería de Sistemas de Información

Escuela de Ingeniería de Sistemas y Computación | Facultad de Ingeniería

UPC

Universidad Peruana
de Ciencias Aplicadas

Prolongación Primavera 2390,
Monterrico, Santiago de Surco

Lima 33 - Perú

T 511 313 3333

https://www.upc.edu.pe

exígete, innova

LAUREATE
INTERNATIONAL
UNIVERSITIES