**LENGUAJES DE PROGRAMACIÓN**

II Ciclo

**Laboratorio 6: Colecciones**

**Alumnos:** <colocar aquí los **apellidos** y nombres del alumno>

<colocar aquí los **apellidos** y nombres del alumno>

**Fecha de realización:** 00-00-2021

**Fecha de entrega:** 00-00-2021

1. **Objetivos**

* Comprender el uso de las colecciones
* Identificar los usos y tipos de Listas, Diccionarios y Tuplas
* Implementar aplicaciones con Listas, Diccionarios y Tuplas

1. **Seguridad**

|  | **Advertencia:**   * Obligatorio el uso de Mascarilla y careta de protección * Desinfectar las manos con alcohol al ingreso del laboratorio * Mantener siempre la distancia durante el laboratorio * No obstruir las vías de evacuación con mochilas, sillas en mala posición, etc. * Ubicar maletines y/o mochilas al final del aula o en los casilleros asignados al estudiante. * No ingresar con líquidos, ni comida al aula de Laboratorio. * Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas. * En este laboratorio está prohibida la manipulación del hardware, conexiones eléctricas o de red |
| --- | --- |

1. **Equipos y Materiales**

Una computadora con:

* Windows 7 o superior
* Conexión a la red
* Software de desarrollo

1. **Fundamento Teórico**

**Colecciones:** Una colección permite agrupar varios objetos bajo un mismo nombre. Por ejemplo, si necesitamos almacenar en nuestro programa los nombres de los alumnos de un curso de programación, será más conveniente ubicarlos a todos dentro de una misma colección de nombre alumnos, en lugar de crear los objetos alumno1, alumno2, etc.

En Python existen tres tipos de colecciones básicas: listas, diccionarios y tuplas.

**Lista:** Así como una cadena, una lista es una secuencia de valores. En una cadena, los valores son caracteres; en una lista, pueden ser cualquier tipo. Los valores en una lista son llamados elementos o a veces ítems. Hay varias formas de crear una nueva lista; la más simple es encerrar los elementos en corchetes (“[" y “]”):

[10, 20, 30, 40]

['rana crujiente', 'anticucho de corazón', 'choncholí frito']

**Diccionarios:** Un diccionario es como una lista, pero más general. En una lista, los índices de posiciones tienen que ser enteros; en un diccionario, los índices pueden ser (casi) cualquier tipo.

eng2sp = dict()

print(eng2sp)

{}

**Tuplas:** Las tuplas son otro tipo de secuencia que funciona de forma parecida a una lista, tienen elementos indexados empezando desde 0

A diferencia de una lista, una vez que creas una tupla, **no puedes alterar su contenido** de forma similar a una cadena

Puesto que Python no tiene que construir la estructura de una tupla de modo que sea modificable, las tuplas **son más simples y eficientes**, en términos de uso de memoria y desempeño, que una lista. Así que en nuestros programas, cuando creamos “variables temporales”, preferimos tuplas en vez de listas

(x, y) = (4, 'fred')

print(y)

fred

(a, b) = (99, 98)

print(a)

99

1. **Procedimiento:**

**Constantes en listas:**

Ejercicio 1: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| * Las constantes de una Lista están encerradas por corchetes y los elementos en la lista están separados por comas * Un elemento de una lista puede ser cualquier objeto de Python – incluso otra lista * Una lista puede estar vacía | #Constantes en listas  print([1, 24, 76])  print(['rojo', 'amarillo', 'azul'])  print(['rojo', 24, 98.6, True, False])  print([ 1, [5, 6], 7])  print([]) |
| --- | --- |

**Las listas son mutables:**

Ejercicio 2: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| A diferencia de las cadenas, las listas son mutables porque pueden cambiar el orden de los elementos en una lista o reasignar un elemento en una lista. Cuando el operador corchete aparece en el lado izquierdo de una asignación, éste identifica el elemento de la lista que será asignado.  Nota: Las cadenas (strings) son inmutables (no se pueden modificar), pero las listas si son modificables. | #Mutabilidad de listas  numeros = [17, 123]  print(numeros)  numeros[1] = numeros[1] + 1  print(numeros)  numeros[1] = 5  print(numeros) |
| --- | --- |

**Listas y bucles definidos (for):**

Ejercicio 3: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| El operador in funciona también en listas. | #in y for en listas  z = ['Kevin', 'Brayan', 'Britany']  for x in z:  print(f'¡Feliz año nuevo!: {x}')  print('¡Hecho!') |
| --- | --- |

**Búsqueda dentro de listas:**

Ejercicio 4: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Así como en las cadenas, podemos obtener cualquier elemento individual de una lista utilizando un índice especificado en corchetes | #Búsqueda  amigos = ['Kevin', 'Brayan', 'Britany']  print(amigos[1]) |
| --- | --- |

**Longitud de una lista:**

Ejercicio 5: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| La función len() toma una lista como parámetro y retorna el número de elementos en la lista  De hecho, len() nos da el número de elementos de cualquier conjunto o secuencia (tal como una cadena...) | #Longitud de una lista  saludo = 'Hola Lucho'  print(len(saludo))  x = [ 1, 2, 'Lucho', 99]  print(len(x)) |
| --- | --- |

**Función range:**

Ejercicio 6: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| * La función range retorna una lista de números que van desde cero hasta el número anterior al parámetro * Podemos construir un bucle por índices usando un for y un entero iterador | #Función range  print(range(4))  for x in range(4):  print(x)  amigos = ['Brayan', 'Kevin', 'Britany']  print("Tengo: " + str(len(amigos)) + " amigos")  print(f"Tengo: {len(amigos)} amigos")  print(range(len(amigos))) |
| --- | --- |

**Listas con operadores + y \*:**

Ejercicio 7: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Podemos crear una nueva lista al juntar dos listas creadas previamente usando el operador + para concatenar listas  El operador \* repite una lista un determinado número de veces | #Listas con el operador +  a = [1, 2, 3]  b = [4, 5, 6]  c = a + b  print(c)  #Listas con el operador \*  print([0] \* 4)  print([1, 2, 3] \* 3) |
| --- | --- |

**Recortar listas:**

Ejercicio 8: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Recuerda: Tal como en las cadenas, el segundo número es “hasta pero no incluyendo” | #Recortar listas  t = [9, 41, 12, 3, 74, 15]  print(t[1:3])  print(t[:4])  print(t[3:])  print(t[:])  t = ['a', 'b', 'c', 'd', 'e', 'f']  t[1:3] = ['x', 'y']  print(t) |
| --- | --- |

**Métodos de listas (append, extend, sort):**

Ejercicio 9: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| * Python provee métodos que operan en listas. Por ejemplo, append agrega un nuevo elemento al final de una lista: * Podemos crear una lista vacía y después agregar elementos usando el método append (agregar) * La lista mantiene su orden y los nuevos elementos son agregados al final de la lista | #Método append  t = ['a', 'b', 'c']  print(t)  t.append('d')  print(t)  #Crear lista desde cero  cosas = list()  cosas.append('libro')  cosas.append(99)  print(cosas)  cosas.append('galleta')  print(cosas)  cosas.append('galleta', 'chorizo')  print(cosas)  ncosas = ['galleta', 'chorizo']  cosas.append(ncosas)  print(cosas) |
| --- | --- |

Ejercicio 10: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| extend toma una lista como argumento y agrega todos los elementos: | #Método extend  t1 = ['a', 'b', 'c']  t2 = ['d', 'e']  t1.extend(t2)  print(t1) |
| --- | --- |

Ejercicio 11: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| sort ordena los elementos de la lista de menor a mayor o viceversa. | #Método sort  t = ['d', 'c', 'e', 'b', 'a']  print(t)  t.sort()  print(t)  t.sort(reverse=True)  print(t) |
| --- | --- |

**Eliminar elementos de una lista:**

Ejercicio 12: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| * Si sabes el índice del elemento que quieres, puedes usar pop. * pop modifica la lista y regresa el elemento que fue removido. Si no provees un índice, la función elimina y retorna el último elemento. * Si no necesitas el valor removido, puedes usar el operador del * Si sabes qué elemento quieres remover (pero no sabes el índice), puedes usar remove. * El valor de retorno de remove es None. * Para remover más de un elemento, puedes usar “del” con un índice de corte. | #Eliminar elementos  t = ['a', 'b', 'c']  x = t.pop(1)  print(t)  print(x)  t = ['a', 'b', 'c']  del t[1]  print(t)  t = ['a', 'b', 'c']  t.remove('b')  print(t)  t = ['a', 'b', 'c', 'd', 'e', 'f']  del t[1:5]  print(t) |
| --- | --- |

**Funciones y listas:**

Ejercicio 13: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Hay un cierto número de funciones nativas en Python que toman listas como parámetros ¿Recuerdas los bucles que hicimos? Son mucho más sencillos. | #Funciones nativas para listas  nums = [3, 41, 12, 9, 74, 15]  print(len(nums))  print(max(nums))  print(min(nums))  print(sum(nums))  print(sum(nums)/len(nums)) |
| --- | --- |

Ejercicio 14: Copiar el programa de abajo, compilar, **comparar**, analizar y comentar lo que sucede.

| # Funciones nativas para cadenas  total = 0  contador = 0  while True :  inp = input('Ingresa un número: ')  if inp == 'fin' : break  valor = float(inp)  total = total + valor  contador = contador + 1  promedio = total / contador  print('Promedio:', promedio) | #Funciones nativas para listas  numlista = list()  while True :  inp = input('Ingresa un número: ')  if inp == 'fin' : break  valor = float(inp)  numlista.append(valor)  print(numlista)  promedio = sum(numlista) / len(numlista)  print('Promedio:', promedio) |
| --- | --- |

**Cadenas y listas:**

Ejercicio 15: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Una cadena es una secuencia de caracteres y una lista es una secuencia de valores, pero una lista de caracteres no es lo mismo que una cadena. Para convertir una cadena en una lista de caracteres, puedes usar list. | #Cadenas y listas  s = 'spam'  t = list(s)  print(t)  l = list()  l.append(s)  print(l) |
| --- | --- |

**Partir listas (split):**

Ejercicio 16: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Split separa una cadena en partes y produce una lista de cadenas. Podemos verlas como palabras. Podemos acceder a una palabra en particular o iterar a través de todas ellas. | #Método split y operador in  abc = 'Con tres palabras'  cosas = abc.split()  print(cosas)  print(len(cosas))  print(cosas[0])  print(cosas)  for w in cosas :  print(w) |
| --- | --- |

Ejercicio 17: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Cuando no especificas un delimitador, múltiples espacios son tratados como un solo delimitador  Puedes especificar qué carácter delimitador utilizar al dividir llamando a la función split | #Delimitador  linea = 'Muchos espacios'  etc = linea.split()  print(etc)  linea = 'primero;segundo;tercero'  cosa = linea.split()  print(cosa)  print(len(cosa))  cosa = linea.split(';')  print(cosa)  print(len(cosa)) |
| --- | --- |

Ejercicio 18 (Reto Tecsup): Escribe un programa que pide al usuario una lista de números e imprime el máximo y el mínimo de los números, para finalizar se ingresará “fin”. encapsula el programa usando una función y modifica el programa para almacenar los números que el usuario ingrese en una lista, y utiliza las funciones max( ) y min( ) para calcular el máximo y el mínimo después de que el bucle termine, compilar, analizar y comentar lo que sucede.

| Ingresa un número: 6  Ingresa un número: 2  Ingresa un número: 9  Ingresa un número: 3  Ingresa un número: 5  Ingresa un número: fin  [6, 2, 9, 3, 5]  Máximo: 9.0  Mínimo: 2.0 |
| --- |

**Diccionarios:**

**Literales de Diccionarios (Constantes):**

Ejercicio 19: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Los literales de diccionarios se escriben con llaves y tienen una lista en par tipo clave : valor (keys : values). Puedes inicializar un diccionario vacío escribiendo llaves vacías | #Diccionarios  jjj = { 'Kevin' : 1 , 'Brayan' : 42, 'Jan': 100}  print(jjj)  ooo = { }  print(ooo) |
| --- | --- |

Ejercicio 20: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Las listas indexan sus entradas basadas en la posición en la lista.  Los diccionarios son como bolsas – no tienen orden.  Así que indexamos las cosas que ponemos en un diccionario con una “etiqueta de búsqueda” | #Diccionarios  bolsa = dict()  bolsa['dinero'] = 12  bolsa['dulce'] = 3  bolsa['papel'] = 75  print(bolsa)  print(bolsa['dulce'])  bolsa['dulce'] = bolsa['dulce'] + 2  print(bolsa) |
| --- | --- |

Ejercicio 21: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Los Diccionarios son como listas a excepción de que utilizan claves en vez de números para buscar valores (values) | #Listas VS Diccionarios  lst = list()  lst.append(21)  lst.append(183)  print(lst)  lst[0] = 23  print(lst)  ddd = dict()  ddd['edad'] = 21  ddd['curso'] = 182  print(ddd)  ddd['edad'] = 23  print(ddd) |
| --- | --- |

**Diccionarios como cuentas múltiples:**

Ejercicio 22: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| La aplicación más usual de los diccionarios es realizar cuentas de sus elementos.  podemos usar el operador in para saber si una clave se encuentra en el diccionario | #Diccionario como cuentas múltiples  ccc = dict()  ccc['Kevin'] = 1  ccc['Brayan'] = 1  print(ccc)  ccc['Kevin'] = ccc['Kevin'] + 1  print(ccc)  print('Kevin' in ccc) |
| --- | --- |

Ejercicio 23: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Cuando encontramos un nuevo nombre, necesitamos agregar una nueva entrada en el diccionario y si es la segunda vez o después encontramos de nuevo el nombre, simplemente sumamos uno al contador en el diccionario bajo ese nombre | #Diccionario como cuentas múltiples  contadores = dict()  nombres = ['Kevin', 'Brayan', 'Kevin', 'Jan', 'Brayan']  for nombre in nombres :  if nombre not in contadores:  contadores[nombre] = 1  else :  contadores[nombre] = contadores[nombre] + 1  print(contadores) |
| --- | --- |
| Podemos usar get( ) y proveer un valor por defecto de cero cuando la clave no existe aún en el diccionario - y después sumar uno | #Diccionario como cuentas múltiples  contadores = dict()  nombres = ['Kevin', 'Brayan', 'Kevin', 'Jan', 'Brayan']  for nombre in nombres :  contadores[nombre] = contadores.get(nombre,0) + 1  print(contadores) |

Ejercicio 24: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| El patrón general para contar las palabras en una línea de texto es dividir la línea en palabras, y después recorrer las palabras y usar un diccionario para mantener la cuenta de cada palabra de forma independiente.  podemos usar los métodos **keys** y values para obtener las **claves** y valores | #Diccionario como cuentas múltiples  contadores = dict()  linea = input('Ingresa el texto: ')  palabras = linea.split()  for palabra in palabras:  contadores[palabra] = contadores.get(palabra,0) + 1  print("----------------------------------------------")  print(contadores.keys())  print("----------------------------------------------")  print(contadores.values())  print("----------------------------------------------")  print(f'Total de palabras: {sum(contadores.values())}')  print(f'Total de palabras no repetidas: {len(contadores.values())}') |
| --- | --- |
| Ingresa el texto: Algunos profesores de la carrera de aviónica fueron invitados a entrar en un avión, después de que todos se sentaron cómodamente, se les fue informado que el avión había sido construído por sus alumnos. Todos ellos se levantaron y corrieron despavoridos, solamente un profesor permaneció sereno y sentado en su lugar. Cuando le preguntaron por qué no huía, él explicó: «Conozco la capacidad de mis alumnos, si fueron ellos quienes construyeron este avión, tengo total confianza de que esta chatarra ni siquiera va a prender...» | |

Ejercicio 25 (Reto Tecsup): Realizar un programa que cuente las palabras de un archivo de texto.txt

| Abra el archivo: texto.txt  Total de palabras: 85  Total de palabras no repetidas: 68 | |
| --- | --- |

**Tuplas**

Ejercicio 26: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Las tuplas son otro tipo de secuencia que funciona de forma parecida a una lista, tienen elementos indexados empezando desde 0 | x = ("Kevin", "Brayan", "Jan")  print(x)  print(x[2])  y = (1, 9, 2)  print(y)  print(max(y))  for i in y:  print(i) |
| --- | --- |

**Cosas que no se deben hacer con tuplas**

Ejercicio 27: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| A diferencia de una lista, una vez que creas una tupla, **no puedes alterar su contenido**, de forma similar a una cadena y solo tienen dos métodos que se pueden usar [‘count’, ‘index’] | x = (4, 3, 1)  x[2] = 2  x.sort()  x.append(2)  x.reverse() |
| --- | --- |

**Listas, Diccionarios y Tuplas**

Ejercicio 28: Copiar el programa de abajo, compilar, analizar y comentar lo que sucede.

| Las listas, Diccionarios y Tuplas, tienen cada uno de ellos, métodos que podemos mostrarlos con la función nativa dir( ) | l = list()  print(dir(l))  print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  d = dict()  print(dir(d))  print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  t = tuple()  print(dir(t)) |
| --- | --- |

Ejercicio 29 (Tarea para la casa video): Del ejercicio anterior, buscar en internet y poner un ejemplo de todos los métodos (sin sub guiones) de las Listas, Diccionarios y Tuplas que no se hicieron en clase.

###### **Entregables**: Capture la pantalla con la ejecución de los 30 ejercicios en el VS Code y comente lo que sucede.

1. **Observaciones:**

abc

1. **Conclusiones:**

abc

1. **Normas Generales:**
2. Presentar como informe de laboratorio el titulo, número de laboratorio, nombres, fechas, entregables(Lo resaltado en amarillo), observaciones, conclusiones y su propia bibliografía. **No colocar, objetivos, seguridad , equipos y materiales, fundamento teórico, procedimientos, normas generales y anexo**, ya que el turnitin va a reconocer un alto porcentaje de copia**.**
3. Por cada captura de pantalla que se realice, se deberá comentar en no menos de 3 líneas explicando la figura o procedimiento.
4. La presentación del informe se realiza por el Canvas, un día antes de que se cumpla la semana de realizado el laboratorio (6 días), no se revisa por otro medio.
5. Se penalizará con puntos en contra las faltas de ortografía en la redacción del informe, la presentación fuera del tiempo indicado, en especial la falta de orden.
6. Se evaluará con nota cero si el nombre de uno de los participantes no está incluido en el informe, incluir a un alumno que no ha participado en la experiencia de laboratorio, el nombre del profesor está mal escrito, la presentación del informe por otro medio que no sea el Canvas, o el más mínimo intento de plagio.
7. Se dará una bonificación de un punto a aquel alumno que publique en el foro de consultas, alguna información relevante del tema tratado en la semana o encuentre alguna falla de redacción u ortografía en las presentaciones y guías de laboratorio, así como de algún error en el Canvas.
8. Pasada una semana de la publicación de la nota en el SEVA, se dará por entendido la conformidad del alumno con su nota, es responsabilidad del alumno la verificación de la conformidad de su nota, en caso de algún reclamo pasada la semana, ya no será tomada en cuenta.
9. **Bibliografía**
10. Desarrollo inicial: Charles Severance, Facultad de Información de la Universidad de Michigan
11. Charles R. Severance (2020). Python para todos “Explorando la información con Python 3” Estados Unidos: Copyright ~2009- Charles Severance.
12. **Anexo**