programa de BECAS
DGTIC
de FORMACIÓN en TIC

Seguridad Informática

Programación con Python

Servando Miguel López Fernández
Jesús Enrique Pacheco Franco

Estructuras de Datos - Listas

- Pueden almacenar elementos de diversos tipos de datos
- No tienen un tamaño definido, crecen y decrecen dependiendo de los elementos que contiene
- Se inicializan con corchetes
- Son mutables, pues pueden cambiar los elementos contenidos
- Se pueden operar con slices

```
lista_vacia = []
lista1 = ['alumno1','alumno2','alumno3']
lista2 = [0,1,2,3,4]
```

Descargar listas1.py

Estructuras de Datos - Tuplas

- Pueden almacenar elementos de diversos tipos de datos
- Tienen un tamaño definido, no se pueden modificar sus elementos una vez creada
- Son inmutables (no se pueden agregar o quitar elementos)
- Son más rápidas que las listas
- Se pueden operar con slices
- Tienen casi las mismas funciones que las listas

```
tupla_vacia = ()
tupla = (1,)
tupla1 = (0,1,2,3)
tupla2 = (4,5,6,7)
```

v = (’ a ’ , 2 , True)

(x , y , z) = v

x -> ’ a ’

y -> 2

z -> True

(LUNES, MARTES, MIERCOLES, JUEVES,VIERNES, SABADO, DOMINGO) = range (7)

LUNES -> 0

MARTES -> 1

Estructuras de Datos - Conjuntos

- Estructura que solo puede contener elementos no mutables (números, cadenas o tuplas)
- No soportan indexado (ni slices), pues no tienen un orden específico
- No puede tener elementos duplicados
- Buscar elementos en un conjunto es más rápido que hacerlo en una lista

```
conjunto_vacio = set()
conjunto = set([0,1,2,3])
conjunto = {4,5,6,7}
```

Modificación de Conjuntos

Hay dos maneras de añadir valores a un conjunto: el método **add()** y el método **update()**.

```
1  >>> un_conjunto = {1, 2}
2  >>> un_conjunto.add(4)
3  >>> un_conjunto
4  {1, 2, 4}
5  >>> len(un_conjunto)
6  3
7  >>> un_conjunto.add(1)
8  >>> un_conjunto
9  {1, 2, 4}
10 >>> len(un_conjunto)
11 3
```

El método **add()** recibe un parámetro, que puede ser de cualquier tipo, cuyo resultado es añadir el parámetro al conjunto.

```
1  >>> un_conjunto = {1, 2, 3}
2  >>> un_conjunto
3  {1, 2, 3}
4  >>> un_conjunto.update({2, 4, 6})
5  >>> un_conjunto
6  {1, 2, 3, 4, 6}
7  >>> un_conjunto.update({3, 6, 9}, {1, 2, 3, 5, 8, 13})
8  >>> un_conjunto
9  {1, 2, 3, 4, 5, 6, 8, 9, 13}
10 >>> un_conjunto.update([10, 20, 30])
11 >>> un_conjunto
12 {1, 2, 3, 4, 5, 6, 8, 9, 10, 13, 20, 30}
```

El método **update()** funciona como si llamaras al método **add()** con cada uno de los elementos del conjunto que pasas como parámetro.

Eliminar Elementos de un Conjunto

Existen tres formas de eliminar elementos individuales de un conjunto: **discard()**, **remove()** y **pop()**.

```
>>> un_conjunto = {1, 3, 6, 10, 15, 21, 28, 36, 45}
>>> un_conjunto
{1, 3, 36, 6, 10, 45, 15, 21, 28}
>>> un_conjunto.discard(10)
>>> un_conjunto
{1, 3, 36, 6, 45, 15, 21, 28}
>>> un_conjunto.discard(10)
>>> un_conjunto
{1, 3, 36, 6, 45, 15, 21, 28}
>>> un_conjunto.remove(21)
>>> un_conjunto
{1, 3, 36, 6, 45, 15, 28}
>>> un_conjunto.remove(21)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 21
```

```
1  >>> un_conjunto = {1, 3, 6, 10, 15, 21, 28, 36, 45}
2  >>> un_conjunto.pop()
3  1
4  >>> un_conjunto.pop()
5  3
6  >>> un_conjunto.pop()
7  36
8  >>> un_conjunto
9  {6, 10, 45, 15, 21, 28}
10 >>> un_conjunto.clear()
11 >>> un_conjunto
12 set()
13 >>> un_conjunto.pop()
14 Traceback (most recent call last):
15   File "<stdin>", line 1, in <module>
16 KeyError: 'pop from an empty set'
```

El método **pop()** elimina único valor del conjunto y retorna el valor. Sin embargo, como los conjuntos no están ordenados, no hay un "último" elemento, por lo que no hay forma de controlar qué elemento es el que se extrae. Es aleatorio.

El método **clear()** elimina todos los valores del conjunto dejándolo vacío.

Estructuras de Datos - Conjuntos

Operación	Equivalente	Resultado
len(s)		Número de elementos (cardinalidad)
x in s		Revisa si x pertenece al conjunto s
x not in s		Revisa si x no pertenece al conjunto s
s.issubset(t)	s <= t	Revisa si todos los elementos de s están en t
s.issuperset(t)	s >= t	Revisa si todos los elementos de t están en s
s.union(t)	s | t	Nuevo conjunto con los elementos de s y t
s.intersection(t)	s & t	Nuevo conjunto con los elementos en común de s y t
s.difference(t)	s - t	Nuevo conjunto con los elementos de s y no de t
s.symmetric_difference(t)	s ^ t	Nuevo conjunto con los elementos de s y t, que no sean communes en ambos

Estructuras de datos - diccionario

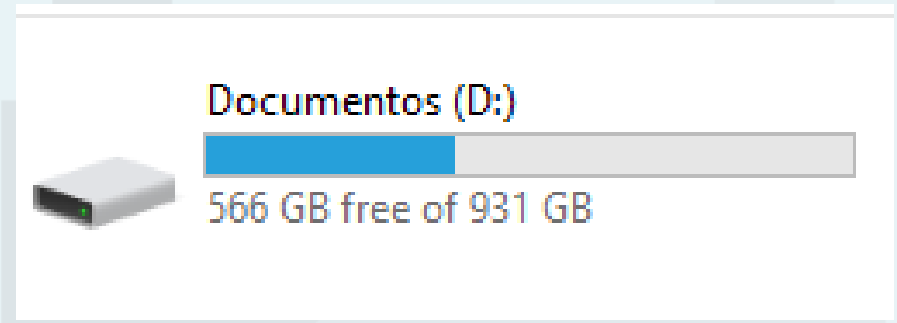
- Estructura que permiten mapear claves a valores.
- Las claves deben ser elementos no mutables (números, cadenas, tuplas)
- Se puede accede al valor, usando la clave

```
dic_vacio = {}
dic1 = {‘Pedro’:58, ‘Juan’:34,’Jose’:40}
dic1[‘Pedro] -> 58
```

Descargar: estructuras.py

Ejercicio de clase 3

- Descargar: ejercicio3.py
- Hacer función que regrese dos tuplas:
  - Nombres de los alumnos aprobados (calificación >= 8)
  - Nombres de los alumnos reprobados (calificación < 8)
- Hacer función que regrese el promedio de calificaciones de los alumnos (número real)
- Hacer función que regrese el conjunto de las calificaciones obtenidas.


Documentos (D:)
566 GB free of 931 GB

Archivos

- Con python se puede manipular archivos de una forma sencilla.
- Los archivos se pueden abrir en modo lectura, escritura o concatenación.
- Python puede manipular archivos de texto o binaries.
- Siempre que se abra un archivo, se debe cerrar una vez que se termina de manipular.

Archivos – Apertura

- Se usa la función “open”
- El primer argumento es la ruta (absoluta o relativa) del archivo.
- El segundo argumento es el modo de apertura.

Modo de apertura	Significado
r	Lectura: sólo puede leer el contenido del archivo, sin modificarlo.
w	Escritura: Reescribe el contenido de un archivo
a	Concatenación: Permite modificar e contenido de un archivo sin eliminar el contenido anterior
r+	Lectura y escritura simultánea
rb	Lectura binaria: igual que “r” pero con archivos binarios (fotos, videos, etc.)
wb	Escritura binaria: igual que “w” pero con archivos binarios (fotos, videos, etc.)
ab	Concatenación binaria: igual que “a” pero con archivos binarios (fotos, videos, etc.)

Archivos – Apertura

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

f1 = open('calificaciones.txt', 'r')

'''
Acciones a realizar con el contenido del archivo
'''


f1.close()
```

Archivos – Lectura

- Leer todo el contenido de un archivo.

```
f1 = open('becarios.txt','r')
contenido = f1.read()
f1.close()
```

- Leer una cantidad definida de bytes

```
f1 = open('becarios.txt','r')
contenido = f1.read(10)
f1.close()
```

Archivos – Lectura

- Obtener una lista con todos los renglones del archivo.

```
f1 = open('becarios.txt','r')
contenido = f1.readlines()
f1.close()
```

Archivos – Escritura

- Cuando un archivo se abre en modo escritura, se borra el contenido original.

```
f1 = open('becarios.txt','w')
f1.write('Esta cadena se escribirá en el archivo')
f1.close()
```

Archivos – Manejo Sencillo

- Siempre se debe de cerrar un archivo que se ha abierto, incluso si se produce una excepción.
- Para asegurarnos de cerrarlo siempre, podríamos hacer manejo de excepciones o utilizar la instrucción “with”.
- La instrucción “with” se asegura de cerrar un archivo sin importer la situación.

Archivos – Manejo Sencillo

- La sentencia ‘with’ genera un bloque de código nuevo.

```
with open(‘input.txt’,’r’) as input_file:
    for line in input_file.readlines():
        print line.upper()
```

Descargar: instructores.txt

Ejercicio Archivos

Descargar el archivo **instructores.txt**

Crear una función que califique a los instructores. La función tiene que abrir el archivo instructores.txt y escribir en otro archivo llamado **calificaciones.txt** el nombre del instructor seguido de un espacio y la oración ’rifa mucho’ si su calificación es mayor o igual a 8, ‘debe mejorar’ si su calificación es mayor o igual a 6 y ‘no debería dar clase’ si tiene calificación menor a 6.

instructores.txt

[nombre_del_instructor][-][calificacion]