Seguridad Informática

Programación con Python

Virgilio Castro Rendón
Fernando Marcos Parra Arroyo

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]
```

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

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

```
conj_vacio = set()
conj1 = set([0,1,2,3])
conj2 = {4,5,6,7}
```

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.

Programación orientada a objetos

- Paradigma de programación popularizado en la década de los 90s
- Clases y objetos
  - Atributos y métodos
- Herencia
- Abstracción
- Polimorfismo
- Encapsulamiento

POO - Clases

- Una clase define lo que será un objeto.
- Una clase podría considerarse un plano o un molde para crear objetos.
- Define los atributos (características) y métodos (acciones) que tienen los objetos
- Se definen usando la palabra reservada *class*

POO - Clases

- Definición de una clase

```
class Becario:
    def __init__(self,nombre,calificacion):
            self.nombre = nombre
            self.calificacion = calificacion

    def ve_calificacion():
            if self.calificacion < 8: print ‘Debo estudiar mas’
            else: print ‘Aún me falta mucho por aprender’

b1 = Becario(‘Ignacio’, 9)
b2 = Becario(‘Luis’, 7)
b3 = Becario(‘Valeria’, 8)
```

Algunos conceptos

- Importar módulos
  - import *modulo*
  - import *modulo* as *alias*
  - from *modulo* import *
  - from *modulo* import *clase1,funcion1*
- Python interpretado?
- Función main?

POO – Ejercicio de clase 4

- Descargar: poo.py
- Descargar: ejercicio4.py
- Las clases deben mantenerse en un archivo separado (poo.py)
- Modificar el archivo ejercicio4.py para que la variable calificacion_alumno no sea un diccionario, sino una lista de objetos de la clase Becario.

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/python
# -*- coding: utf-8 -*-
#UNAM-CERT

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 importar 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: archivos.py

Tarea 3 – Generador de diccionarios

- Se deberá leer un archivo que contendrá en cada renglón una palabra sobre alguna persona (nombre, nombre de mascota, color favorito, etc). Al menos 10 renglones.
- Deberá generar una lista de posibles contraseñas usando diversas técnicas:
  - Cambiar letras por números
  - Cambiar números por letras
  - Pasar de mayúsculas a minúsculas
  - Pasar de minúsculas a mayúsculas
  - Concatenar usando números y símbolos especiales
  - Etcétera
- La lista generada deberá escribirse en un nuevo archivo (una contraseña por línea)