El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

# CICLO I:

## Fundamentos de Programación en Python

MisionTIC2022

Hechos que CONECTAN

El futuro digital es de todos MinTIC

Mision TIC 2022

UNIVERSIDAD DEL NORTE
Vigilada Mineducación

# Sesión 16:

# Manejo de Archivos

## DATOS DE ARCHIVOS

Hechos QUE CONECTAN

# Objetivos de la sesión

Al finalizar esta sesión estarás en capacidad de:

1. Construir un programa con una interfaz de consola o grafica simple
2. Construir un programa que maneje archivos de texto para almacenar información persistente (formato TXT, JSON, CSV)

# Archivos: Acceso a archivos

- **Crear un archivo**
- **Modo de apertura de un archivo**
- **Leer desde un archivo de texto**
- Escribir en un archivo
- Cerrar un archivo
- Persistencia de datos
- Diccionarios

# Archivos: Escribir en un archivo

Presentamos dos formas comunes de escribir a un archivo:

- Mediante cadenas: escribe una cadena al archivo.

  **archivo.write(cadena)**

- Mediante listas de cadenas: recibe una lista de líneas para escribir.

  **archivo.writelines(lista_de_cadenas)**

# Archivos: Cerrar un archivo

Al terminar de trabajar con el archivo, se cierra. Para cerrar un archivo se realiza lo siguiente:

```
archivo.close()
```

Esto libera el archivo para ser usado por otros programas, y además asegura que los cambios sobre él se guarden.

Python se encarga de cerrar todos los archivos que queden abiertos al final del programa, sin embargo, se recomienda como una buena práctica no dejar nada al azar y cerrar el archivo tan pronto se termine de usar.

# Archivos: Persistencia de los datos almacenados

Persistencia es la capacidad de poder guardar la información de un programa para volver a utilizarla en otro momento.

Esto se refiere a lo que los usuarios llaman como Guardar el archivo y después Abrir el archivo.

Para un programador, esto puede significar suele involucrar un proceso de serialización de los datos a un archivo o a una base de datos o a algún otro medio similar, y el proceso inverso de recuperar los datos a partir de la información serializada.

# Archivos: ¿Cómo escribir sobre un archivo?

1. **Con write( )**

```
línea= ”Linea 1”
f = open(”archivo.txt”, ”w”)
f.write(linea)
f.close()
```

2. **Con writelines( )**

```
lista = [”linea 1\n”, ”linea 2”]
f = open(”archivo.txt”, ”w”)
f.writelines(lista)
f.close()
```

# DICCIONARIOS

# Diccionarios: Definición

- Son **objetos Python** que representan una estructura de datos que permite almacenar valores de **diferente tipos** (numérico, string o booleano) e incluso listas

- La principal características de los diccionarios es que los datos se almacenan asociados a una *clave*, creando una asociación **clave: valor** para cada elemento almacenado

- Los elementos almacenados **no están ordenados**. El orden es indiferente a la hora de almacenar la información en el diccionario.

# Diccionarios: Características

| Clase | Notas | Ejemplo |
|---|---|---|
| dict | Son mutables, sin orden. | Los diccionarios pueden ser creados colocando una lista separada por coma de pares «key : value» entre { }, por ejemplo:<br><br>miDic= {"nombre" : "Pedro", "edad" : 47, "active" : True } |

Algunas propiedades de los diccionarios son las siguientes:

- Son **dinámicos**, pueden crecer o decrecer, se pueden añadir o eliminar elementos.
- Son **indexados**, los elementos del diccionario son accesibles a través del **key**.
- Y son **anidados**, un diccionario puede contener a otro diccionario en su campo **value**.
- Sus **claves son únicas**, no puede haber un diccionario que tenga dos veces la misma clave, si se asigna un valor a una clave que ya existe, se sobrescribe el valor anterior.

# Diccionarios: Implementación en Python

- De la misma forma que con listas, es posible definir un diccionario directamente con los miembros que va a contener, o bien inicializar el diccionario vacío y luego agregar los valores de a uno o de a muchos.

- Para definirlo junto con los miembros que va a contener, se encierra el listado de valores entre llaves, las parejas de clave y valor se separan con comas, y la clave y el valor se separan con :

```
punto = {'x': 2, 'y': 1, 'z': 4}
```

- Para declararlo vacío y luego ingresar los valores, se lo declara como un par de llaves sin nada en medio, y luego se asignan valores directamente a los índices.

# Diccionarios: Operaciones

## Definición

```
miDic = {”nombre ” : ” Pedro ”, ” edad ” : 47, ” active ” : True } o

miDic = dict(nombre = ”Pedro”, edad = 47, active= True)
```

## Acceder a valor de clave

```
>> print(miDic[”nombre”])

Pedro
```

## Asignar a valor de clave

```
>> miDic[”edad”]= 25
>> print(miDic)

{’nombre’: ’Pedro’, ’edad’: 25, ’active’: True}
```

# Diccionarios: Operaciones

## Definición

```
miDic = {"nombre " : " Pedro ", " edad " : 47, " active " : True } o

miDic = dict(nombre = "Pedro", edad = 47, active= True)
```

## Crear nueva clave con valor clave

```
>> miDic["sexo"] = "M"
>> print(miDic)
{'nombre': 'Pedro', 'edad': 47, 'active': True, '
sexo'= ' M' }
```

## Iteración con el operador in

```
>> print(miDic)
{'nombre': 'Pedro', 'edad': 47, 'active': True, 'sexo'= ' M' }

>> genero = 'sexo' in miDic
>> print(genero)
'M'
```

# Diccionarios: Algunos métodos más usados

| | |
|---|---|
| **clear()** | **>>> versiones** = dict**(python**=2.7, **zope**=2.13, **plone**=5.1)<br>**>>>** print **(versiones)**<br>{'python': 2.7, 'zope': 2.13,  'plone': 5.1}<br>**>>> versiones**.**clear()**<br>**>>>** print(**versiones)**<br>{ } |
| **copy()** | >>> versiones = dict(python=2.7, zope=2.13, plone=5.1)<br>>>> otro_versiones = versiones.copy()<br>>>> versiones == otro_versiones<br>True |
| **get()** | >>> versiones = dict(python=2.7, zope=2.13, plone=5.1)<br>>>> versiones.get('plone')<br>5.1 |
| **has_key()** | >>> versiones = dict(python=2.7, zope=2.13, plone=5.1)<br>>>> versiones.has_key('plone')<br>True<br>>>> versiones.has_key('django')<br>False |
| **keys()** | >>> versiones = dict(python=2.7, zope=2.13, plone=5.1)<br>>>> versiones.keys()<br>['zope', 'python', 'plone'] |

# Diccionarios: Algunos métodos más usados

| | |
|---|---|
| **clear()** | **>>> versiones** = dict**(python**=2.7, **zope**=2.13, **plone**=5.1)<br>**>>>** print **(versiones)**<br>{'python': 2.7, 'zope': 2.13,  'plone': 5.1}<br>**>>> versiones**.**clear()**<br>**>>>** print(**versiones)**<br>{ } |
| **copy()** | >>> versiones = dict(python=2.7, zope=2.13, plone=5.1)<br>>>> otro_versiones = versiones.copy()<br>>>> versiones == otro_versiones<br>True |
| **get()** | >>> versiones = dict(python=2.7, zope=2.13, plone=5.1)<br>>>> versiones.get('plone')<br>5.1 |
| **has_key()** | >>> versiones = dict(python=2.7, zope=2.13, plone=5.1)<br>>>> versiones.has_key('plone')<br>True<br>>>> versiones.has_key('django')<br>False |
| **keys()** | >>> versiones = dict(python=2.7, zope=2.13, plone=5.1)<br>>>> versiones.keys()<br>['zope', 'python', 'plone'] |

# Diccionarios: Algunos usos

- Contar el número de veces que una palabra aparece dentro de un texto.

- Para tener una agenda donde la clave es el nombre de la persona, y el valor es una lista con los datos correspondientes a esa persona.

- En general, los diccionarios sirven para crear bases de datos muy simples, en las que la clave es el identificador del elemento, y el valor son todos los datos del elemento a considerar.

# COMPONENTE PRÁCTICO

El futuro digital es de todos

MinTIC

Mision TIC 2022

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

# ¡GRACIAS

## POR SER PARTE DE ESTA EXPERIENCIA DE APRENDIZAJE!

Hechos QUE CONECTAN