Universidad Tecnológica Nacional

Tecnicatura en Programación

Trabajo Práctico Integrador

Informe teórico

Gestión de Datos en Países en Python

Integrantes: Gonzalo Sanchez, Jerónimo Almagro, Serrano Danilo

Materia: Programación I

Docente: Gerardo Magni

Introducción

En el presente informe se analiza el uso de las estructuras de datos fundamentales en Python, tales como listas, diccionarios, funciones, condicionales, así como herramientas para la manipulación de información incluyendo ordenamientos, estadística y archivos csv o txt. Python es un lenguaje de programación de alto nivel, interpretado, conocido por su sintaxis clara y sencilla, lo que facilita el aprendizaje y desarrollo de programas.

En este informe se combina teoría y práctica, presentando explicaciones y diagramas de flujo que representan el proceso de las operaciones principales. Esto ayuda a entender de forma más clara cómo se manipulan y guardan los datos para la creación de mejores y más eficientes programas.

Desarrollo

Python es un lenguaje de programación de alto nivel, interpretado y de código abierto, creado por Guido van Rossum en 1991. Fue diseñado centrándose en la simplicidad y la legibilidad del código, hecho que permite a los desarrolladores expresar conceptos en menos líneas de código en comparación con otros lenguajes como C++ o Java.

Entre las características más importantes de Python se encuentran sus estructuras de datos, que permiten almacenar, organizar y manipular información de manera eficiente. Las más comunes son: listas, diccionarios, tuplas y conjuntos, a continuación, las detallaremos:

Listas:

Las listas son conjuntos ordenados de elementos, estas se delimitan por corchetes ([ ]) y los elementos se separan por comas. Las listas pueden contener elementos del mismo tipo, o de distintos tipos como números, cadenas o otras listas. Lo más llamativo de este tipo de estructuras de datos, es que las listas son mutables, es decir, que sus elementos pueden modificarse después de su creación.

Propiedades de las listas:

- Son ordenadas, mantienen el orden en el que han sido definidas
- Pueden ser formadas por tipos arbitrarios
- Pueden ser indexadas con [i].
- Se pueden anidar, es decir, meter una dentro de la otra.
- Son mutables, ya que sus elementos pueden ser modificados.
- Son dinámicas, ya que se pueden añadir o eliminar elementos.

Crear listas:

```
#Crear lista vacía
lista = []
#Crear lista con mismo tipo de dato
lista = ["elemento1", "elemento2", "elemento3"]
#Crear lista con distintos tipos de datos
lista = ["elemento1", 3, True]
#Otra manera de crear listas
lista = list("1234")
```

Acceder a la lista:

Se puede acceder a un elemento usando corchetes y un índice, que va desde 0 hasta la longitud de la lista -1.

```
lista = ["elemento1", "elemento2", "elemento3"]
#Acceder a un elemento de la lista
print(lista[0])
print(lista[1])
print(lista[2])
```

Modificar listas:

Para modificar un elemento de una lista, primero se debe acceder a él mediante su índice y luego asignarle un nuevo valor a través de =

```
lista = ["elemento1", "elemento2", "elemento3"]
#Modificar un elemento de la lista
item1 = lista[0] = 1
print(lista)
```

Recorrer una lista:

Para recorrer una lista en Python, se debe acceder uno a uno por sus elementos, generalmente mediante un bucle for o while.

```
lista = ["elemento1", "elemento2", True, 25]
#Recorrer una lista por el índice
for i in lista:
    print(i)
```

Métodos Listas:

Las listas en Python no solo permiten almacenar datos, sin que también poseen métodos incorporados que facilitan su manejo.

Por ejemplo:

```
#Agregar elemento al final de la lista
lista.append("Elemento nuevo")
#Eliminar elemento por valor
lista.remove(25)
#Eliminar último elemento de la lista
lista.pop()
#Devuelve la cantidad de elementos
len(lista)
```

Diccionarios:

Un diccionario es una colección no ordenada de objetos que permite almacenar su contenido en forma de clave y valor. Estas claves suelen ser números enteros o cadenas o cualquier valor inmutable. En cambio, los valores pueden ser de cualquier tipo. Los diccionarios se crean con llaves ({}) y las claves se separan e los valores con dos puntos (:).

Propiedades de los diccionarios:

- Son dinámicos, pueden crecer o decrecer, se pueden añadir o eliminar elementos.
- Son indexados, los elementos del diccionario son accesibles a través del key.
- Pueden anidarse, un diccionario puede contener a otro diccionario en su campo value.

Crear diccionarios:

```
diccionario = {
    "elemento1": "valor",
    "elemento2": 15,
    3: [2,5],
}
diccionario = dict(nombre = "Jorge", edad = 30)
```

Acceder al diccionario:

Para acceder a un valor, se debe usar la clave correspondiente, colocándola entre corchetes o usando el método get.

```
#Acceder a un valor del diccionario a través de su clave
print(diccionario["elemento1"])
print(diccionario.get("elemento2"))
```

Modificar diccionario:

Para modificar un elemento se debe acceder al elemento como explique anteriormente y luego asignarle un nuevo valor, si la clave no existe, se añade automáticamente.

```
#Modificar valor del diccionario
nuevo_item = diccionario["elemento1"] = 100
print(diccionario)
```

Recorrer diccionario:

Suele hacerse con un bucle for, para mostrar o modificar los elementos del diccionario, tanto las claves como valores.

```
#Recorrer claves del diccionario
for key in diccionario:
    print(key)
#Recorrer valores del diccionario
for value in diccionario:
    print(diccionario[value])
#Recorrer valores y claves del diccionario
for key,value in diccionario.items():
    print(key, value)
```

Métodos diccionario:

```
#Devuelve el valor de una clave
print(diccionario.get("nombre"))
#Devuelve las claves
print(diccionario.keys())
#Elimina todo lo del diccionario
print(diccionario.clear())
#Elimina el valor de nombre y la clave
print(diccionario.pop("nombre"))
```

Funciones:

Una función en programación es un bloque de código diseñado para realizar una tarea específica y que se puede reutilizar varias veces. En Python, una función incluye argumentos de entrada**,** el código a ejecutar y los valores de salida. Para

usarla, se llama por su nombre definido con def; por ejemplo, si hay una función Saludar() que imprime “Hola”, se ejecuta simplemente escribiendo Saludar().Crear una función:

```
def nombre_funcion(param1, param2):
    resultado = param1 + param2
    return resultado
```

Partes de una función

Toda función está compuesta por varias partes importantes: nombre de la función, sus parámetros, cuerpo y el retorno.

```
#Definimos la función con un nombre
def suma(a,b): #(a,b) son los parámetros de la función
    res = a + b
    return res #valor que devuelve la función
a = int(input("Ingresa un número: "))
b = int(input("Ingresa otro número:"))
print(suma(a,b)) #invocamos a la función y le pasamos sus parámetros
# y luego se escribe el valor retornado
```

Nombre de la función

El nombre de la función es el identificador con el cual podemos invocarla o reutilizarla en nuestro programa.

Parámetros y argumentos de una función:

Una función puede recibir valores cuando se invoca a través de unas variables conocidas como parámetros que se definen entre paréntesis en la declaración de la función. En el cuerpo de la función se pueden usar estos parámetros como si fuesen variables.

```
def calcular_area(base, altura):
    area = base * altura
    return area
# Llamamos a la función pasando argumentos
resultado = calcular_area(5, 3)
print("El área del rectángulo es:", resultado)
```

Retorno de una función:

Una función puede devolver un objeto de cualquier tipo tras su invocación. Para ello el objeto a devolver debe escribirse detrás de la palabra reservada return. Si no se indica ningún objeto, la función no devolverá nada.

```
# Ejemplo de función que retorna algo
def cuadrado(numero):
    return numero ** 2
```

```
resultado = cuadrado(5)
print("El cuadrado de 5 es:", resultado)
#Ejemplo de función que no retorna nada (función de proceso)
def saludar(nombre):
    print(f"¡Hola, {nombre}!")
resultado = saludar("Miguel")
print(resultado)
```

Funciones incorporadas

Conjunto de funciones que están disponibles para su uso directo en cualquier programa sin necesidad de importarlas. Python proporciona un conjunto extense de funciones integradas que facilitan el desarrollo de programas sin la necesidad de escribir código adicional.

```
#Print imprime un mensaje en la consola
print("Bienvenido al programa")
#Len devuelve la longitud de un objeto, una lista, tupla o cadena
lista = ["elemento1", 25, True]
lon = len(lista)
print(lon)
#Input le pide al usuario que ingrese un dato, se devuelve en string
input("Ingresa una letra o palabra: ")
#Range genera una secuencia de número dentro de un rango especificado
for numero in range(1,5):
    print(numero)
```

Funciones personalizadas

Son funciones creadas por el programador para realizar tareas específicas dentro de un programa. El usuario se encarga de diseñarlas según sus necesidades.

```
#Función personalizada creada para saludar a una persona
def saludar(nombre):
    return "Bienvenido " + nombre

nombre = input("Ingresa tu nombre ")
print(saludar(nombre))
```

Ventajas de usar funciones

Las funciones en programación permiten **abstraer la complejidad del código**, ya que podemos usarlas sin conocer su funcionamiento interno. Además, fomentan la **modularidad**, al dividir un programa en partes independientes y más manejables. Esto mejora la **reutilización del código**, evitando repetir instrucciones, y aumenta la **legibilidad**, facilitando la comprensión, el mantenimiento y la detección de errores en los programas.

Condicionales

En Python If-Else es una declaración condicional fundamental utilizada para la toma de decisiones en programación. De no ser por esta estructura el código en cualquier lenguaje de programación sería ejecutado de forma secuencial, una tarea detrás de otra, gracias a las condiciones podemos cambiar el flujo de ejecución de un programa, haciendo que ciertos bloques de código se ejecuten si se dan ciertas condiciones.

if

```
edad = 20
if edad > 18:
    print("Eres mayor")
```

elif

```
edad = 30
if edad < 18:
    print("eres menor de 18")
elif edad >= 18 and edad <= 30:
    print("tienes entre 18 y 30 años")
elif edad > 30 and edad <= 80:
    print("tienes entre 30 y 80 años")
```

else

Se ejecuta si ninguna de las condiciones fue verdadera, es opcional y siempre se encuentre al final.

```
edad = 90
if edad < 18:
    print("eres menor de 18")
elif edad >= 18 and edad <= 30:
    print("tienes entre 18 y 30 años")
elif edad > 30 and edad <= 80:
    print("tienes entre 30 y 80 años")
else:
    print("Eres mayor de 80")
```

Ordenamientos

El ordenamiento es el proceso de ordenar elementos de una colección siguiendo un criterio específico, generalmente de mayor a menor o alfabéticamente.

sort

Este método modifica la propia lista, ordenando los elementos de forma ascendente o alfabéticamente.

```
#lista numérica
listaNumeros = [4,26,1,6,2]
listaNumeros.sort()
print(listaNumeros)
#Lista con letras
listaLetras = ["e", "z", "a", "d"]
listaLetras.sort()
print(listaLetras)
```

sorted

Este método se utiliza para ordenar cualquier iterable, no solo listas, es una función muy parecida a sort(), pero a diferencia de que devuelve una nueva lista.

```
#lista numérica
listaNumeros = [4,26,1,6,2]
ordenarNumeros = sorted(listaNumeros)
print(ordenarNumeros)
#Lista con letras
listaLetras = ["e", "z", "a", "d"]
ordenarLetras = sorted(listaLetras)
print(ordenarLetras)
```

Estadísticas Básicas

Sum

La función sum() en Python está diseñada para sumar todos los elementos de un conjunto iterable de izquierda a derecha, ya sean listas, tuplas u otros y retorna el total.

```
lista = [1,3,5,2,5]
suma = sum(lista)
print(suma)
```

Min

La función min() de Python devuelve el valor más pequeño de un conjunto de valores o el elemento más pequeño de un iterable pasado como parámetro. Es útil cuando se necesita determinar rápidamente el valor mínimo de un grupo de números u objetos. Por ejemplo:

```
a = [ 23 , 25 , 65 , 21 , 98 ]
print( min(a))
b = ["plátano", "manzana", "mango", "kiwi"]
print( min(b))
```

Max

El max() La función devuelve el elemento más grande de un iterable. También se puede utilizar para encontrar el elemento más grande entre dos o más parámetros.

```
a = [4,69,2,5,19]
print(max(a))
b = ["Banana", "Durazno", "Naranja"]
print(max(b))
```

Statics.mean()

Retorna la media aritmética muestral de data, que puede ser una secuencia o un iterable. La media aritmética es la suma de los valores dividida entre el número de observaciones. Es comúnmente denominada «promedio», aunque hay muchas formas de definir el promedio matemáticamente. Es una medida de tendencia central de los datos.

```
import statistics
datos = [3, 5, 7, 9]
media = statistics.mean(datos)
print(media)
```

Statistics.median()

Retorna la mediana (valor central) de los datos numéricos. Cuando el número de casos es impar, se retorna el valor central y cuando el número de observaciones es par, la mediana se interpola calculando el promedio de los dos valores centrales.

```
import statistics as stats
# Caso 1: número impar de datos
datos1 = [3, 1, 9, 5, 7]
print(stats.median(datos1))
# Caso 2: número par de datos
datos2 = [10, 2, 6, 8]
print(stats.median(datos2))
```

statistics.mode()

Retorna el valor más común del conjunto de datos discretos o nominales. La moda (cuando existe) es el valor más representativo y sirve como medida de tendencia central.

```
import statistics as stats
datos1 = [3, 5, 5, 7, 9]
print(stats.mode(datos1))
```

Archivo csv

Para poder guardar datos de forma permanente y reutilizarlos en otras ejecuciones, utilizamos archivos. Un archivo es un conjunto de datos almacenado en el disco. Estos datos pueden ser texto, números, imágenes, configuraciones, entre otros. Algunos ejemplos comunes de archivos que se utilizan en el desarrollo de software son:

- Archivos .txt para guardar texto plano.
- Archivos .csv para almacenar datos en forma tabular, separados por comas.
- Archivos .json para registrar eventos o errores.

Crear archivos csv

El archivo CSV se abre como un archivo de texto con la función open() incorporada de Python, que devuelve un objeto de archivo.

```
Juan;Perez;53365;8
Maria;Lopez;55654;10
Pablo;Gomez;58999;6
```

Abrir archivos

```
archivo = open("alumnos.txt", "r")
```

Modos de apertura:

- 'r': lectura (read). El archivo debe existir.
- 'w': escritura (write). Si el archivo existe, se sobreescribe. Si no, se crea.
- 'a': agregar (append). Se abre para escribir al final del archivo sin borrar lo anterior.

Leer archivos

En este ejemplo se abre el archivo alumnos.csv en modo lectura “r” y se usa el método read() para obtener todo su contenido y mostrarlo por consola.

```
archivo = open("alumnos.csv", "r")
contenido = archivo.read()
print(contenido)
```

Además, podemos leer archivos línea por línea de esta manera.

```
with open("alumnos.csv", "r") as archivo:
    for linea in archivo:
        print(linea.strip())
```

Escribir en archivos

Como ya explicamos anteriormente, se utiliza el modo “w” para escribir en un nuevo archivo o reemplazar el contenido existente.

```
with open("alumnosNuevos.csv", "w") as archivo:
    archivo.write("Nuevo contenido")
    print(archivo)
```

Agregar a archivos

El modo “a” permite agregar información al final de un archivo sin borrar su contenido. Además, al igual que “w” si el archivo no existe crea una automáticamente.

```
with open("EjemploAgregarArchivos.csv", "a") as archivo:
    archivo.write("Nuevo contenido")
    print(archivo)
```

Métodos de lectura

Strip()

- Elimina los caracteres al principio o final de la línea.

```
with open("alumnos.csv", "r") as archivo:
    lineas = archivo.read()
    print(lineas.strip())
```

Split()

- Divide la línea en partes, separándola por las comas.
- Devuelve una lista con los elementos de la línea.

```
with open("alumnos.csv", "r") as archivo:
    lineas = archivo.read()
    print(lineas.split())
```

Cerrar archivos

Luego de abrir in archivo, es importante cerrarlo para que libere los recursos del sistema. Esto puede hacerse con el método close() o automáticamente con la instrucción with, que cierra el archivo al finalizar el bloque de manera automática.

```
#Ejemplo cerrar archivo manualmente
archivo = open("alumnos.csv", "r")
contenido = archivo.read()
archivo.close()
#Ejemplo cerrar archivos automáticamente
with open("alumnos.csv", "r") as archivo:
```

```
contenido = archivo.read()
print(contenido)
```

Bibliografía

https://ellibrodepython.com/diccionarios-en-python

https://recursospython.com/guias-y-manuales/diccionarios/

https://aprendeconalf.es/docencia/python/manual/funciones/

https://llego.dev/posts/creating-and-documenting-custom-functions-in-python/

https://j2logo.com/python/ordenar-una-lista-en-python/

https://developers.google.com/edu/python?hl=es-419

https://docs.python.org/es/3/tutorial/

https://aws.amazon.com/es/what-is/python/

https://www.godaddy.com/resources/latam/desarrollo/python-que-es

https://docs.python.org/es/3/library/functions.html#max