RELATORÍA 1

MICHAEL STIVEN VALENCIA MORA

mivalenciam@unal.edu.co

INFORMATICA III

DOCENTE: CRISTIAN ELIAS PACHON PACHECO

UNIVERSIDAD NACIONAL DE COLOMBIA SEDE MANIZALES

FACULTAD DE CIENCIAS EXACTAS Y NATURALES

INGENIERÍA FÍSICA

MANIZALES

2023

Objetivo

Recordar conceptos básicos de Python tales como tipos de datos, métodos, ciclos y funciones, así como sus respectivas sintaxis de cara a poder abordar temas más específicos en clases posteriores.

1. CREACIÓN DE REPOSITORIOS (17 – 02 - 2023):

El primer paso para crear un repositorio es crear la carpeta contenedora desde el gestor de archivos del computador en la dirección deseada, luego de esto se debe ingresar a Visual Studio Code y realizar las siguientes acciones.

Ir al apartado de “File” en la barra superior izquierda y seleccionar la opción “Open Folder” en el menú desplegable.

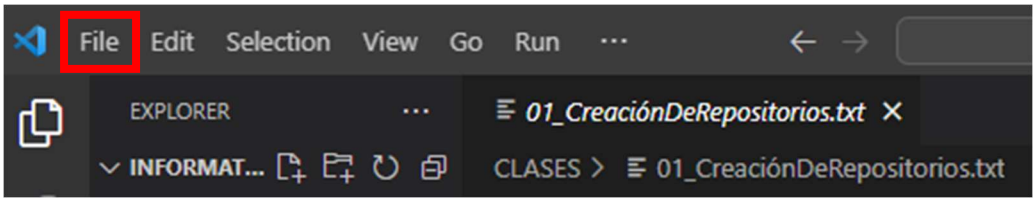

Figura 1. Barra de navegación.

Posteriormente se deberá buscar la carpeta creada anteriormente en el gestor de archivos, una vez seleccionada, se debe ir a la sección “Source Control” ubicada en la barra lateral izquierda.

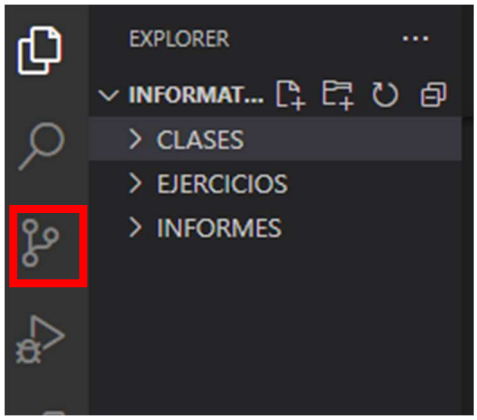

Figura 2. Barra lateral.

Una vez en el apartado “Source Control” seleccionar el botón “Initialize Repository”, luego de esto se debe poner un nombre al repositorio en la barra superior y definir si será público o privado.

Para enlazar el repositorio creado a la nube se debe tener instalado el software “Git”, una vez con el instalado se debe abrir la terminal desde Visual Studio Code y escribir las siguientes líneas de código.

- git config --global user.name "Usuario"
- git config --global user.email "Correo"

Luego de seguir estos pasos es posible subir los archivos creados a la nube (cada que se vaya a subir uno o varios archivos es necesario poner un comentario).

2. TIPOS DE DATOS (24 – 02 - 2023):

En Python es muy importante conocer los distintos tipos con los que se trabaja, ya que cada uno cumple una función específica y tiene sus propias formas de ser manipulados, se dividen en los siguientes:

Básicos:

```
String:         "", '', "Hola mundo", "1234"
Enteros:        3, 1, -512 2451
Flotantes:      2.4, 124.5, -12.5
Booleanos:      True, False
```

Figura 3. Datos básicos.

Estructurados:

```
Listas:         [1,2,3], ["a", "b", "c"]
Tuplas:         (1,2,3), ("a", "b", "c")
Diccionarios:   {"nombre": "Michael", "apellido": "Valencia", "edad": 22}
Conjuntos:      {"A", "B", "C", "D"} (a diferencia de los diccionarios estos solo tienen elementos independientes)
```

Figura 4. Datos estructurados.

Algunos de estos datos interactúan entre sí por medio de los siguientes operadores.

```
Asignacion:     = (a = 3, b = "hola")
Aritmeticos:    +, -, *, /, // (Division entera), % (Modulo de la division), ** (Potencia)
Logicos:        and (True and True => True), or (False or False => Flase), not (not False => True)
Comparativos:   > (3 > 5 => False), >= (3 >= 3 => True), < (-10 < 0 => True), <= (-10 <= 10 => True),
Pertenencia:    in (1 in [1,2,3] => True), not in (1 not in [1,2,3] => False)
```

Figura 5. Operadores

En la figura 5 se muestran algunos ejemplos de casos en los que se pueden usar los distintos tipos de operadores.

3. FUNCIONES INTEGRADAS (01 – 03 – 2023):

También conocidas como built – in functions son muy útiles a la hora de manipular cualquier tipo de dato en Python ya que realizan tareas que a menudo se presentan en el desarrollo de un programa de manera rápida, también existen varios tipos de funciones integradas que se acomodan a cada situación y tipo de dato, estas funciones necesitan de un parámetro de entrada que será sobre el que actuarán.

Funciones de entrada y salida:

- **input():** Permite que el usuario ingrese cualquier dato que se necesite y Python lo interpreta como una cadena.
- **print():** Muestra en la consola el argumento proporcionado.

Funciones de ayuda:

- **dir():** Muestra en la consola todos los métodos que se pueden aplicar al objeto pasado como argumento, más adelante se explicará que son los métodos en Python.
- **help():** Muestra una larga lista de información acerca del objeto suministrado como argumento, en esta información además de otras cosas se muestran métodos y una breve explicación de su funcionamiento y uso.

Funciones de conversión:

Estas funciones se dividen en los tipos de datos entre los que pueden realizar las conversiones.

- **Entre tipos de datos:** int(), float(), str(), list(), complex(), … Son bastante intuitivas además de útiles.

- **Entre sistemas numéricos:** int(), hex(), oct(), bin().

Funciones de secuencia:

Estas funciones crean secuencias a partir de los parámetros proporcionados.

- **range(inicio, final + 1, paso):** Crea una secuencia de números tomando el inicio, pero no el final, además se le puede proporcionar el tamaño del paso entre cada número.
- **enumerate(lista o tupla):** Crea un objeto donde cada elemento es una tupla que contiene como primer elemento el índice y como segundo elemento el correspondiente a dicho índice en el objeto argumento.
- **zip(objeto1, objeto2):** Los argumentos pueden ser listas o tuplas, y crea un objeto donde cada elemento es una tupla que contiene los elementos que comparten índice en cada uno de los objetos.

4. METODOS (03 – 03 - 2023):

Los métodos se aplican a los objetos, y como en Python todos los tipos de datos son objetos, entonces los métodos se pueden aplicar a cualquier tipo o estructura de datos, estos difieren de un objeto a otro ya que en cada uno realizan tareas distintas y específicas.

Sintaxis:

objeto.metodo()

Los métodos se dividen por el tipo de objeto al que se aplican, como sigue.

Métodos de las cadenas

Formateo: Le cambian el formato de diferentes formas a una cadena, ejemplos: capitalize(), upper(), lower(), title(), center(), strip().

Operaciones: Cambian o dan información acerca de cierto parámetro en la cadena, ejemplos: count(subcadena), replace(old, new), find(subcadena).

Verificaciones: Determinan si la cadena pertenece a cierto grupo de caracteres, ejemplo: isalpha(), isalnum(), isdigit(), isdecimal().

Indexing: Se ponen corchetes después del nombre de la cadena y permiten acceder al carácter asociado al índice que se ponga entre los corchetes.

Slicing: Permite extraer una sección determinada de la cadena al poner entre los corchetes el inicio, el fin y el paso.

Ejemplo:

```
cadena = "hola"

print(cadena.upper())
print(cadena.lower())
print(cadena.center(50, "-"))
```

Figura 6. Ejemplo de métodos de las cadenas.

Ejecutando el código anterior se obtiene.

```
HOLA
hola
----------------------hola----------------------
```

Figura 7. Ejecución de los métodos de las cadenas.

Métodos de las listas

Operaciones: Modifican la estructura de la lista, ejemplos: append(valor), remove(valor), pop(indice), insert(indice, valor), count().

Ordenado: Permiten acomodar los elementos de una lista a conveniencia, ya sea ascendente o descendente, ejemplos: sort(), reverse()

Almacenamiento: Permiten manipular la forma en la que se guarda la lista en la memoria del computador, ejemplos: clear(), copy()

Las listas también tienen los métodos de indexing y slicing, funcionan igual que para las cadenas.

Ejemplo:

```
lista1 = ["A", "B", "C"]
lista1.pop(-1)
lista1.append("Michael Stiven Valencia Mora")
print(lista1)
```

Figura 8. Ejemplo métodos de las listas.

Ejecutando el código anterior se obtiene.

```
['A', 'B', 'Michael Stiven Valencia Mora']
```

Figura 9. Ejecución de los métodos de las listas.

Métodos de los diccionarios

Extracción: Permiten obtener ya sea las claves o los valores de un diccionario, ejemplos: keys(), values(), get(<clave>).

Eliminación: Permiten remover pares clave, valor de un diccionario, ejemplo: pop(<clave>).

Estos también cuentan con métodos de almacenamientos y slicing igual que los casos anteriores, solo que en este caso el slicing se hace sobre las claves, y se obtienen los valores.

Ejemplo:

```
calificaciones = {"Juan": 5.0, "David": 2.4, "Maria": 2.9, "Esteban": 2.2, "Daniela": 2.0,
                  "Mario": 3.1, "Juanita": 2.1, "José": 3.0, "Sebastian": 2.3, "Cristian": 2.0,
                  "Alberto": 3.9, "Angélica": 4.2, "Angel": 2.0, "Marly": 2.5, "Mariana": 4.5, "Josefina": 2.7}

claves = calificaciones.keys()
valores  = calificaciones.values()

calificaciones["Marly"] = 4.3
calificaciones["Angel"] = 3.1
calificaciones["Juanita"] = 3.5

calificaciones.pop("Josefina")
calificaciones.pop("Juan")

print(valores)
print(claves)
print(calificaciones)
```

Figura 10. Ejemplo métodos de los diccionarios.

Ejecutando el código anterior.

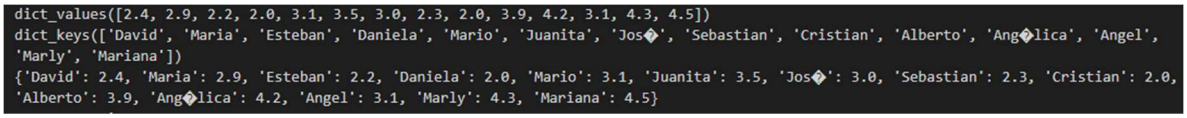

Figura 11. Ejecución de los métodos de los diccionarios.

5. CONDICIONAL IF (15 – 03 - 2023):

Es una de las herramientas más básicas de la mayoría de lenguajes de programación, funciona con una condición, si el resultado de evaluar dicha condición es True, entonces se ejecuta el código dentro del condicional, si existe una sentencia else, entonces se ejecuta el código dentro de else, sino simplemente se ignora.

Sintaxis:

if (<condición>):

    <sentencias>

else:

    <sentencias alternas>

También se puede tener un condicional if que examine varias condiciones con la sentencia elif, de la siguiente manera.

if (<condición 1>):

    <sentencias>

elif (<condición 2>)

    <sentencias>

else:

<sentencias alternas>

Ejemplo:

```
edad = 22

if(edad >= 18):
    print("Usted es mayor de edad")
else:
    print("Usted es menor de edad")
```

Figura 12. Ejemplo condicional if.

Ejecutando el código anterior.

```
Usted es mayor de edad
```

Figura 13. Ejecución ejemplo del condicional if.

6. CICLOS (17 – 03 – 2023):

Ciclo while:

En este ciclo se ejecuta una sentencia una y otra vez mientras se cumpla una sentencia, es útil cuando se desconoce el número de iteraciones que se deben hacer para cumplir con determinada tarea, se debe tener cuidado de no escribir un ciclo infinito garantizando que la condición se actualice en cada iteración.

Sintaxis

while <condición>:

<sentencia>

Ejemplo:

```
secuencia1 = [4,22,5,6,3,7,9,2,0,1,5]
par = 0
impar = 0
cont = 0

while cont < len(secuencia1):
    if(secuencia1[cont] == 0):
        break
    elif(secuencia1[cont]%2 == 0):
        par += 1
    elif(secuencia1[cont]%2 != 0):
        impar += 1
    cont += 1

print(par)
print(impar)
```

Figura 14. Ejemplo ciclo while.

Ejecutando el código anterior.

```
4
4
```

Figura 15. Ejecución del ejemplo del ciclo while.

Ciclo for:

Este ciclo es útil para cuando se quiere iterar a través de una secuencia o conjunto de datos de longitud conocida.

Sintaxis:

for <contador> in <iterable>

<sentencia>

Ejemplo:

```
iterable = range(0,10)

for numero in iterable:
    print(numero)
```

Figura 16. Ejemplo ciclo for.

Ejecutando el código anterior.

```
1
2
3
4
5
6
7
8
9
```

Figura 17. Ejecución del ejemplo del ciclo for.

7. FUNCIONES (22 – 03 – 2023):

Cuando se tiene un código en el que aparece recurrentemente un conjunto de sentencias lo recomendable es implementar una función, esto evitará tener que escribir códigos robustos que consuman más recursos de los necesarios para funcionar, además de ahorrar mucho tiempo.

Sintaxis:

def nombreDeLaFuncion(<parámetros>):

<sentencias>

return <elemento a retornar>

Ejemplo:

```
def esUnNumeroPar(numero):
    if(numero%2 == 0):
        return "El numero es par"
    else:
        return "El numero es impar"


lista = [[1,2,3],
         [1,2,3],
         [1,2,3],
         [10,20,30]]
```

Figura 18. Ejemplo función.

Ejecutando el código anterior.

```
[6, 6, 6, 60]
[13, 26, 39]
```

Figura 19. Ejecución ejemplo de función.

CONCLUSIONES

Las herramientas expuestas anteriormente constituyen la base de los distintos paradigmas de programación actualmente más utilizados y proporcionan una base fuerte para entender las librerías que se utilizan como complemento para la mayoría de aplicaciones.