Universidad Nacional de Colombia - sede Manizales
Facultad de Ciencias Exactas y Naturales
Curso: Informática 3.

Repositorios

Michael López Parra

Presentado a:

Cristian Elías Pachón Pacheco

024 de marzo del 2023

---

1. **Objetivos**
   - ❖ Crear y Recordar el entorno Python y todos sus elementos básicos.
   - ❖ Comprender el funcionamiento de los elementos y funciones integradas en Python.
   - ❖ Comprender la creación de funciones y su importancia en las líneas de código.

2. **Temario**

2.1 Entorno Python.

Durante esta clase se configuró el editor de código, el cual será Visual Studio Code, se descargaron las extensiones necesarias tanto para GitHub como para Python con la intención de que podamos subir los archivos a la nube y el editor de código (VS Code) pueda interpretar el lenguaje Python, del mismo modo se procedió a crear una cuenta de GitHub donde semana a semana se irán subiendo las clases y ejercicios vistos.

Como paso siguiente se crearon los repositorios para así subirlos a la nube y poder obtenerlos de manera fácil y rápida, para ello se siguieron estos pasos:

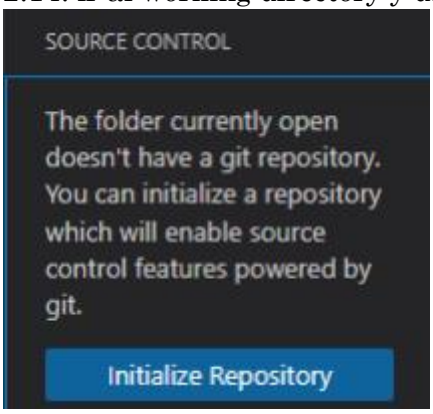
2.11. Tener la cuenta GitHub creada

2.12. configurar usuario y correo en Visual Studio Code (VS):

- para ello se abre el terminal y copiamos lo siguiente:
  - git config - - global user.name “username” (este nombre de usuario es el nombre que se creó con la cuenta de GitHub)
  - git config - - global email.user “useremail@unal.edu.co” (este correo debe ser el correo asociado a la cuenta de GitHub)

2.13. Ir al icono de “source control”

2.14. ir al working directory y dar clic en “initialize repository”



2.15. El archivo en cuestión aparecerá en pantalla y al lado tendrá el símbolo más (+), dar clic en él para llevar el archivo a la “stage area”

2.16. Luego llevar el archivo a la “commit area” dando clic en el botón commit

2.17. Por último, publicar el archivo en GitHub, puede ser como un repositorio privado o uno público

2.2 **Tipos de datos y funciones integradas**

En esta clase se comprendió como se puede dar instrucciones a Python para que opere lo conveniente en cada caso, se comprendió que este lenguaje, y en general en programación, trabaja bajo ciertos parámetros lógicos y cada uno con su función específica, en este campo los llaman “datos” y son: strings, enteros, flotantes y booleanos.

Los strings son cadenas de texto, los enteros son los números enteros, los flotantes, los números racionales e irracionales y los booleanos son datos, los cuales verifican si una sentencia es verdadera o falsa.

Estos a su vez se pueden almacenar en otro tipo de datos, los cuales son lista, tuplas diccionarios y conjuntos, todas puedan contener strings, enteros, flotantes o booleanos, así como combinaciones de estos, ejemplo de cómo se almacenan estos datos es:

```
lista=[1,2,3,4]
tupla = (10,20,20,30)
diccionario = {1:"uno", 2:"dos", 3:"tres"}
```

Teniendo los tipos de datos que maneja Python se necesita una manera para compararlos o usarlos dentro de las líneas de código, para ello existen los operadores:

- operador de asignación: =
- operadores aritméticos: +, -, *, /, //, %, ** (suma, resta, multiplicación, división, división entera, modulo (residuo de la división) y potencia)
- operadores lógicos: and, or, not
- operadores de comparación: <. <=, >, >=, !=, ==
- operadores de pertenencia: in not in
- operadores de conjuntos: |, & (unión, intersección)

Ahora bien, con todo lo anterior se puede trabajar en Python, pero no obtenemos nada por consola, solo se ejecuta el código, o a lo mejor si se quiere agregar datos a una lista o demás

operaciones que se requiera, Python cuenta con ciertas funciones integradas para ayudar a la elaboración de esas operaciones, de las cuales las más convenientes son:

- entrada y salida:
  - input( ): permite el ingreso de datos por consola
  - print( ): imprime en pantalla algo deseado
  - format( ): permite formatear y sustituir valores de una cadena (string)
- ayuda:
  - help( ): nos da información del elemento
  - dir( ): devuelve los elementos y funciones que se pueden aplicar al elemento
  - type( ): dice de qué tipo es el elemento (strings, enteros, etc)
- conversiones:
  - int( ): convierte a entero, no puede convertir un string alfabético a entero
  - float( ): convierte a decimal
  - str( ): convierte a string
  - bool( ): convierte a booleano
  - list( ): devuelve una lista
  - bin( ): convierte a binarios
  - oct( ): convertir a octal
  - hex( ): convierte a hexadecimal
- secuencias
  - range( ): genera un rango de números
  - enumerate( ):permite guardar los datos de forma ordenada en tuplas
- operaciones de secuencias:
  - len( ): da el tamaño de una lista o rango
  - sum( ): suma los datos

2.3 Métodos

Los métodos en Python son muy similares a las funciones integradas, ya que estas nos permiten trabajar con nuestros datos almacenados, y cada tipo de dato de almacenamiento tiene sus propios métodos, lo cual facilita la obtención de información y la actualización de los datos anteriores.

Métodos para strings:
Capitalize( ); coloca el primer carácter en mayúscula
Upper( ): coloca todo el string en mayúscula
Lower( ); coloca todo el string en minúscula
Title( ); coloca el primer carácter de cada palabra en mayúscula
Replace( ): reemplaza uno o varios caracteres por otros
Isalpha( ); verifica si la cadena es alfanumérica
Cadena[índice]: muestra el carácter en el índice deseado
Cadena[inicio:fin:salto]: muestra una parte de la cadena deseada

Métodos de listas:
Append( ): agrega un valor al final de la lista
Insert (índice, valor): agrega un valor en el índice dado
Copy( ): copia la lista
Pop( ): elimina el objeto en el índice dado
Remove( ): elimina el primer objeto con ese valor

Métodos de diccionarios:
Keys( ): retorna las claves del diccionario
Values( ): retorna los valores del diccionario
Items( ): retorna los pares clave-valor
Pop( ): elimina el valor y devuelve la clave
Popitem( ): elimina el par clave-valor

2.4 Condicionales y ciclos

En esta clase se aprendió sobre los condicionales, estas sirven principalmente para verificar sentencias mediante los diferentes operadores, con esta validación se puede hacer que el código siga una secuencia si ocurre la sentencia inicial, en caso contrario, de que la sentencia no se cumpla, hará otra diferente.
Esto nos permite clasificar los datos y almacenarlos en diferentes tipos.

Además de los condicionales se aprendió sobre los diferentes tipos de ciclos, los cuales ayudan a repetir una misma operación tantas veces se desee, llegando estas a ser infinitas, pero no se recomienda porque el código nunca convergería, en esta clase se explicó acerca del ciclo while, y su analogía es de "mientras pase la sentencia, hará una operación"; y el ciclo for, "para una variable en cierto rango, hacer esta operación" ambas se usarán de manera constante en el código ya que facilita algunos procedimientos y reduce las líneas de código. Ejemplo de esto es:
2.41. Se busca terminar la frase "había una vez una Í arbitraria" en una vocal con tilde y que cada carácter debe estar separado por "- -".
Entonces para hacer esto se declara una variable con la cadena de texto, un contador que inicie en 0 y buscamos el tamaño de la cadena de texto. Usando el ciclo while queda: mientras el contador sea menor al tamaño de la cadena se imprimirá el carácter en la posición del contador más "- -", luego si el carácter está en: "áéíóúÁÉÍÓÚ" terminar el programa.

```
texto = "habia una vez una í arbitraria"
cont = 0
limite = len(texto)

while cont < limite:
    print(texto[cont], end="--")
    if texto[cont] in "áéíóúÁÉÍÓÚ":
        break
    cont += 1
```

2.5 Funciones

Las funciones son una herramienta muy útil para la elaboración de código, ya que se pueden aplicar a todas las variables que tenga el programa, evitando la repetición de código y una mayor eficiencia en el mismo.
Estas funciones reciben ciertos parámetros con los cuales la función trabajará, y a su vez como resultado retornará una variable, la que es claro definir correctamente para evitar confusiones en el código.

Para tener un poco más claro el funcionamiento de estas se pueden comparar con las funciones integradas de Python, ya que en esencia son lo mismo, solo hay que tener cuidado a la hora de aplicar correctamente la función.
Lo siguiente es un ejemplo de una función que busca saber si un número es par:

```
def esUnNumeroPar(numero):
    esPar = (numero % 2 == 0)
    return esPar
```

Lo que retornará la función es la variable "esPar" y se evalúa de la siguiente manera:

```
print(esUnNumeroPar(9))
print(esUnNumeroPar(0))
```

Lo cual nos retornará dos booleanos, el primero será False y el otro un True.

3. **Conclusiones**
    - Se comprendió la creación del entorno de Python y el funcionamiento de las diferentes herramientas que Python nos brinda para la creación de código, sus facilidades y formas de interpretar la información y sus variables.
    - Se evidenció la facilidad que brindan las funciones integradas a la hora de crear código, ya que estas nos evitan trabajo innecesario y es la mejor herramienta a la hora de trabajar con datos.
    - La creación de funciones en Python es muy simple, facilitando el trabajo ya que al recurrir a ellas se logra reducir al mínimo las repeticiones de código, del mismo modo ayudan a la comprensión del programa para terceros ya que genera un programa con más orden y bien estructurado.