RELATORIA SEGUNDO CORTE

INFORMATICA 2

JUAN SEBASTIAN RODRIGUEZ RODRIGUEZ

ESTUDIANTE

CRISTIAN ELIAS PACHON

DOCENTE

UNIVERSIDAD NACIONAL DE COLOMBIA

SEDE MANIZALES

2023

En este documento se relatará y se complementaran con experiencias y argumentos cada uno de los temas vistos en el segundo corte para la asignatura de informática II.

Temas vistos

1. Ciclo While (09-03-23)
2. Ciclo For (23-03-23)
3. Métodos (28-03-23)
4. Funciones (13-04-23)

1) Luego de haber comprendido ya en temas anteriores el funcionamiento menos complejo en Python, se complementa a nuestra lista de conocimientos el Ciclo while, que en palabras sencillas evalúa un parámetro y lo ejecuta infinitamente si la condición dada es verdadera o termina el ciclo si su condición es falsa.

   La sintaxis básica es:

   ```
   contador = 0
   while contador < 10:
     # Ejecuta el bloque de código aquí
     # Siempre que el contador sea inferior a 10
   ```

   A continuación, se muestra un ejemplo:

   ```
   dia = 0
   semana = ['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado',
   'Domingo']
   while dia < 7:
     print("Hoy es " + semana[dia])
     dia += 1
   ```

en clase principalmente fue utilizado para mediaciones en la que el parámetro fuera el mismo hasta un determinado tiempo, y aunque su uso se vio limitado debido al siguiente tema visto (ciclo For), sigue siendo una herramienta bastante útil a la hora de realizar un programa con estas características.

2) Nombrado anteriormente el siguiente tema visto en clase es el ciclo for, que para la orientación que va tomando nuestra clase, fue de mayor utilidad ya que se logra reevaluar problemas anteriores, que en su momento presentaron grandes dificultades por la cantidad inmensa de código que se debía realizar para evaluar una misma operación en distintas variables, véase como ejemplo el problema de calcular distancia mínima entre puntos, en el que se debía evaluar la formula euclidiana para cada uno de los puntos de manera individual y seguido de eso ir reevaluando cada de las distancias asociadas para así hallar la mínima, lo que resultaba en un código absurdamente largo dependiendo de la cantidad de puntos. Pero gracias al ciclo for, la misma cantidad de puntos que tomaba un código excesivamente largo, ahora se podría realizar con 4 a 6 líneas de código y este último calcularía de manera acertada la distancia mínima en ilimitados puntos.

   Antes de pasar al ejemplo en cuestión es importante recalcar como funciona exactamente el ciclo for y la sintaxis del mismo.

   Un bucle for establece la variable iteradora en cada valor de una lista, arreglo o cadena proporcionada y repite el código en el cuerpo del bucle for para cada valor de la variable iteradora.

   En el ejemplo a continuación, usamos un bucle for para imprimir cada número de nuestro arreglo.

```
# Ejemplo bucle for
for i in [1, 2, 3, 4]:
    print(i, end=", ") # prints: 1, 2, 3, 4,
```

este es un ejemplo del problema de los puntos

```
# Definir los puntos
P1 = (5, 2, 3)
P2 = (4, 1, 3)
P3 = (2.5, 1, 0)
P4 = (0.5, 0.5, 2)
P5 = (1, 2, 1)
P6 = (6, 2, 1)
P7 = (3, 2, 0.5)
P8 = (2, 6, 1)
P9 = (0, 0, 0)
P10 = (2, 0, 0.5)
P11 = (2, 2, 3)
P12 = (2, 3, 4)
P13 = (1, 1, 3)
P14 = (4, 4, 4)
P15 = (5, 5, 1)
P16 = (1, 0.5, 1)
P17 = (3, 4, 5)
P18 = (3, 1, 2)
P19 = (3, 9, 1)
P20 = (0.5, 0.5, 6)

# Definir una función para calcular la distancia entre dos puntos
def distancia(p1, p2):
    return ((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2 + (p1[2] -
p2[2])**2)**0.5

# Calcular todas las distancias entre cada par de puntos
puntos = [P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15,
P16, P17, P18, P19, P20]
distancias = {}
for i in range(len(puntos)):
    for j in range(i+1, len(puntos)):
        distancias[(i+1,j+1)] = distancia(puntos[i], puntos[j])

# Encontrar el par con la menor distancia
par_min = min(distancias, key=distancias.get)

# Crear el string con el par de puntos más cercanos
parCercano = f"P{par_min[0]} y P{par_min[1]}"
```

este último código ha sido realizado a fin de dar solución al problema planteado para el informe 2

3) Al entender ya los ciclos fundamentales y tener noción de cada uno de los tipos de datos, se implementa la nueva herramienta de métodos, que ha groso modo, sirve para sustraer eliminar o especificar qué información vamos a utilizar en una respectiva lista o diccionario para entender esto de la mejor manera se planteara la explicación igual a la vista en clase y se otorgara un ejemplo de una línea de código para los problemas del informe 2 en el que sin este tipo de métodos hubiera resultado imposible o muy complicado dar una solución.

Métodos

Objetos en python => strings, lista, diccionarios, tuplas

Todos los objetos tienen unos metodos

Los metodos son funcionalidades

Ejemplos:

lista.append(value) => agregar valor

cadenas.upper() => convierte a mayuscula

diccionario.keys() => extrae claves

Métodos de los strings

formateo: capitalize(), upper(), lower(), title() center(spaces), strip()

operaciones: count(subcadena),find(subcadena),replace(old, new)

verificacion: isalnum(), isalpha(), isdigit, isdecimal,

Indexado: [indice]

Slicing: [indice1:indice2:salto]

Métodos de las Listas

operaciones: append(value), insert(index, value), pop(index), remove(value),count(value)

ordenado: sort(), reverse()

almacenamiento: clear(), copy()

Indexado: [indice]

Slicing: [indice1:indice2:salto]

Métodos de las Tuplas

Operaciones: index(value), count(value)

Indexado: [indice]

Slicing: [indice1:indice2:salto]

Diccionarios

Extraccion: items(), keys(), values(), get(key)

Eliminar: pop(key)

Almacenamiento: clear(), copy()

Indexado: [key]

Ahora el ejemplo

```
for venta in ventas:
    for detalle in venta:
        tipo, cantidad, dia = detalle.split("_")
        cantidad = int(cantidad[0])
        if dia in dias_semana:
```

Se usa el .split para separar y almacenar las 3 distintas variables, esto aprovechándo de que el string en la lista separa los valores aprovechables con un _ ("2D_5NIÑOS_LUNES") Y, más adelante en el código se utiliza un Slicing para retornar únicamente el primer valor de cantidad en un entero ya que de lo contrario retornaría “5NIÑOS” pero al utilizar [0] se retorna solo el 5 y este si es un valor entero listo para ser utilizado en operaciones aritméticas.

4) Las funciones en Python concluirán nuestra temática de aprendizaje en cuanto a operadores en Python se refiere.
   A continuación, su sintaxis:

   Sintaxis
   En Python, una definición de función tiene las siguientes características:

   - La palabra clave def
   - Un nombre de función
   - Paréntesis ’()’, y dentro de los paréntesis los parámetros de entrada, aunque los parámetros de entrada sean opcionales.
   - Dos puntos ’:’
   - Algún bloque de código para ejecutar
   - Una sentencia de retorno (opcional)

Y un ejemplo sencillo de este ultimo

```
# función sin parámetros o retorno de valores
def diHola():
  print("Hello!")

diHola()  # llamada a la función, 'Hello!' se muestra en la consola
```

Las funciones en cuestión resultan de mayor utilizad en la realización de problemas aritméticos, viéndose así en la implementación de la clase y en la realización de varios ejercicios para el informe 2 en el que en este caso se logra crear una función para cada uno de los 5 ejercicios esto llevando consigo la optimización de código y un ambiente agradable a la vista para con el programador.

En conclusiones generales en este corte se logra evidenciar un notorio avance en la capacidad lógica de los estudiantes para resolver distintos problemas planteados, esto con ayuda de las distintas herramientas presentadas por el profesor y evidenciando de manera paralela que, muchos ejercicios con una dificultad bastante elevada debido a la cantidad de líneas de código, se ve simplificada a cantidades súper pequeñas de código con ayuda del aprovechamiento de los nuevos operadores predispuestos en la asignatura.