

Instituto Tecnológico y de Estudios
Superiores de Occidente – ITESO



Fundamentos de Ciencias Computacionales

“FCC ToolKit C&V” – Entrega final

Profesor: Fernando Velasco Loera

Integrantes:

Cinthya Lorena González García

Valeria Guadalupe Ramírez López

Primavera 2021

14 de mayo de 2021

Contenido

Menú Principal.....	3
A nivel usuario.....	3
Documentación técnica y funcionalidad.....	3
Generador de tablas de verdad	6
A nivel usuario.....	6
Documentación técnica y funcionalidad.....	6
Calculadora de conjuntos	12
A nivel usuario.....	12
Documentación técnica y funcionalidad.....	12
Sucesiones.....	16
A nivel usuario.....	16
Documentación técnica y funcionalidad.....	16
Relaciones y funciones.....	24
A nivel usuario.....	24
Documentación técnica y funcionalidad.....	24
Relación de la materia con nuestra carrera.....	28
Conclusiones	28
Bibliografía	29

A nivel usuario

```
|
|
|          FCC TOOLKIT 2021   "C & V"
|
|-----|
|
|          MENÚ PRINCIPAL
|
|-----|
|
|  a. Generador de tablas de verdad
|  b. Calculadora de conjuntos
|  c. Sucesiones
|  d. Relaciones y funciones
|  e. Salir del programa
|  >>>
```

Documentación técnica y funcionalidad

```
# Menú inicial
def menu_inicial():
    t = encabezados("MENÚ PRINCIPAL")
    print("\t\tta. Generador de tablas de verdad\n\t\ttb. Calculadora de conjuntos\n\t\ttc. Sucesiones")
    print("\t\ttd. Relaciones y funciones\n\t\tte. Salir del programa")
    opcion = input("\t\t>>> ").lower()
    return opcion
```

3

Si no se introduce un valor correcto, se ciclará hasta que se cumpla. Cada submenú siguiente funciona igual: no tiene argumentos de entrada, se encarga de imprimir y retorna una letra (a-c, a-d en algunos casos):

- **Función encabezados**

Brinda una mejor apariencia a la salida en consola, pues se encarga de centrar y añadir separaciones con guiones (-) y tuberías (|) a los títulos de cada sección del programa. Recibe un *string* como argumento y sólo imprime.

```
# Formato de encabezados
def encabezados(titulo):
    print(Fore.YELLOW + "-" * 78)
    print(Fore.YELLOW + "|", end="")
    print(Fore.YELLOW + titulo.center(76, " "), end="")
    print(Fore.YELLOW + "|")
    print(Fore.YELLOW + "-" * 78)
```

Imagen 3. Código de la función encabezados.

- **Función tabla_menu**

Permite ingresar al menú del generador de las tablas de verdad. No recibe ningún argumento de entrada, únicamente despliega información sobre los próximos menús a utilizar, los cuales son:

- Instrucciones para hacer uso del generador
- El generador de la tabla
- Regresar al menú principal

```
# Menú generador de tablas (opción 'a' del inicial)
def tabla_menu():
    t = encabezados("GENERADOR DE TABLAS DE VERDAD")
    print("\t\ta. Instrucciones y operadores\n\t\tb. Generar tabla\n\t\tc. Volver al menú principal")
    eleccion = "-"
    while eleccion != "a" or eleccion != "b" or eleccion != "c":
        eleccion = input("\t\t>>> ").lower()
        if eleccion == "a" or eleccion == "b" or eleccion == "c":
            break
    return eleccion # devuelve la letra al menú inicial
```

Imagen 4. Código de la función `tabla_menu`; despliega el submenú de las tablas de verdad.

- **Función calculadora_menu**

Permite ingresar al menú de la calculadora de conjuntos. No recibe ningún argumento de entrada, únicamente despliega información sobre los próximos menús a utilizar, los cuales son:

- [illegible]

- b. Calculadora de conjuntos
- c. Regresar al menú principal

Permite ingresar al menú de las sucesiones. No recibe ningún argumento de entrada, únicamente despliega información sobre los próximos menús a utilizar, los cuales son:

- Instrucciones para hacer uso de las sucesiones
- Calcular sucesión
- Determinar tipo de sucesión
- Regresar al menú principal

```
# Menú Sucesiones - Letra 'c' del menú inicial  
def sucesiones_menu():  
    t = encabezados("SUCESIONES")  
    print("\t\tInstrucciones\n\t\ttb. Calcular sucesión\n\t\ttc. Determinar tipo de sucesión\n\t\td. Volver al menú principal")  
    eleccion = ""  
    while eleccion != "a" or eleccion != "b" or eleccion != "c" or eleccion != "d":  
        eleccion = input("\t\t>> ").lower()  
        if eleccion == "a" or eleccion == "b" or eleccion == "c" or eleccion == "d":  
            break  
    return eleccion # devuelve la letra al menú inicial
```

- **Función relaciones menu**

Permite ingresar al menú de las sucesiones. No recibe ningún argumento de entrada, únicamente despliega información sobre los próximos menús a utilizar, los cuales son:

- 5

c. Regresar al menú principal

```
# Menú Relaciones - Letra 'd' del menú inicial
def relaciones_menu():
    t = encabezados("RELACIONES Y FUNCIONES")
    print("\t\t\u0309a. Instrucciones\n\t\t\u0309b. Determinar relación \n\t\t\u0309c. Volver al menú principal")
    eleccion = "_"
    while eleccion != "a" or eleccion != "b" or eleccion != "c":
        eleccion = input("\t\t>>> ").lower()
        if eleccion == "a" or eleccion == "b" or eleccion == "c":
            break
    return eleccion # devuelve la letra al menú inicial
```

Imagen 7. Código de la función `relaciones_menu`; despliega el submenú de relaciones y funciones.

Generador de tablas de verdad

A nivel usuario

Al entrar al submenú de “Generador de tablas de verdad” con la letra “a”, se desplegarán tres posibles acciones: a) Instrucciones y operadores, b) Generar tabla, y c) Volver al menú principal. Se sigue la misma lógica de teclear la letra correspondiente a la opción que se desea ejecutar. La opción “a” sólo imprimirá un listado de instrucciones, consejos y operadores disponibles para el funcionamiento del apartado. La opción “b” llevará a la sección de entrada de texto para escribir el enunciado compuesto e imprimir la tabla de manera inmediata. La última opción, “c”, retorna al menú principal.

```
|-----|
| GENERADOR DE TABLAS DE VERDAD |
|-----|
a. Instrucciones y operadores
b. Generar tabla
c. Volver al menú principal
>>> █
```

Imagen 8. Submenú de "Generador de tablas de verdad"

Documentación técnica y funcionalidad

Esta es considerada nuestra primer herramienta que se integra a nuestra “Caja de Herramientas de FCC”. Su objetivo es llevar a cabo cálculos proposicionales a partir de enunciados de alta complejidad. Toma en cuenta las proposiciones p , q , r , s , t . Esta herramienta nos permitirá ingresar infinitas sentencias al contar con los principales operadores lógicos:

Negación (~)	Conjunción (^)	Disyunción (v)	Implicación (→)	Bicondicional (↔)
-----------------	----------------	----------------	-----------------	----------------------

A manera de un adelanto, el generador de las tablas de verdad funciona de la siguiente manera:

- Primero que nada, se debe de contar con signos reservados. Tenerlos asignados en algún lugar, estos corresponden a los mencionados en la tabla de arriba.
- Una vez teniéndolos identificados se procede a eliminarlos de la expresión para solo quedarnos con las variables que fueron introducidas; el tamaño de la tabla será correspondiente a la cantidad de letras.
- Se procede a realizar las tablas de verdad donde a partir de la formula conocida, se encarga de agregar los verdaderos y falsos correspondientes.
- En caso de encontrarse una negación, simplemente basta con identificar cual fue el valor aportado y a partir de comparaciones añadir el contrario a la tabla.
- El programa suele resultar muy repetitivo, pues como ya se tiene las tablas de verdad principales almacenadas es muy sencillo el proceso. Habrá comparaciones por cada parte, un ejemplo sería que tuvimos que hacer los resultados de la tabla de verdad de la conjunción, y así para cada uno.
- **Función `inst`**

Al seleccionar la opción “a” del menú del generador, esta función se encargará de imprimir en consola las indicaciones necesarias para que el programa corra adecuadamente (imagen 3), así como la lista de los operadores lógicos. No tiene argumentos o valores de retorno.

```
| INSTRUCCIONES Y OPERADORES |
-----
1. Indicar jerarquía de operaciones con paréntesis ( ), utilizar sólo los NECESARIOS.
2. Escribir un único enunciado utilizando las proposiciones:
   p, q, r, s, t.
OPERADORES POR JERARQUÍA:
   Negación: ~ (alt+126)
   Conjunción: ^ (alt+94)
   Disyunción: v (letra)
   Implicación: > (alt+62)
   Bicondicional: ⇔ (copiar y pegar)

# Instrucciones - Letra 'a' del menú generador de tablas
def inst():
    t = encabezados("INSTRUCCIONES Y OPERADORES")
    print("\t1. Indicar jerarquía de operaciones con paréntesis ( ), utilizar sólo\n\tlos NECESARIOS.")
    print("\t2. Escribir un único enunciado utilizando las proposiciones: \n\t\tp, q, r, s, t.")
    print("\t3. La tabla se imprimirá verticalmente (se lee de arriba a abajo).")
    print("\tOPERADORES POR JERARQUÍA: ")
    print("\t\tNegación: ~ (alt+126) \n\t\tConjunción: ^ (alt+94)\n\t\tDisyunción: v (letra)")
    print("\t\tImplicación: > (alt+62)\n\t\tBicondicional: ⇔ (copiar y pegar)")
```

Imagen 9. Impresión de instrucciones de la tabla de verdad (arriba), código de la función inst (abajo).

- **Función validacion**

Esta es una de las funciones principales del programa ya que inicia con el proceso de validación del enunciado proposicional. Guarda todas sus variables de forma que se puedan utilizar tanto fuera como dentro de otras funciones. Esta función quita espacios del enunciado, obtiene las proposiciones y lleva a cabo el cálculo de renglones y filas de la tabla. Aquí suceden una de las partes muy importantes, pues se encarga de acomodar las proposiciones en el orden

correcto para hacer que, si en el enunciado proposicional se repite una misma literal el generador no la tome como una proposición nueva.

- **Función OperadoresLogicos**

Con las mismas variables globales de la función **validacion**, lleva a cabo la comparación con los cinco principales operadores lógicos. Se basa en listas y

```
#--- Inician funciones de TABLAS DE VERDAD ---
def validacion(): # funcion principal
    global simbolos, enunciado, lista_en, argumentos, matriz
    simbolos = ["~", "^", "v", ">", "<", "(", ")"]
    enunciado = str(input("\tEscribe el enunciado proposicional: "))
    enunciado = enunciado.replace(" ", "") # enunciado sin espacios
    lista_en = list(enunciado)
    lista_en = [q for q in lista_en if q not in simbolos] # enunciado sin simbolos
    # print(lista_en)
    argumentos = set(lista_en)
    # print(argumentos)
    argumentos = sorted(list(argumentos)) # orden alfabético de proposiciones
    col = len(argumentos) # no. de columnas x proposición
    matriz = []

    for a in argumentos:
        renglones = 2 ** len(argumentos) # fórmula 2^n para el no. de proposiciones
        num_iteracion = 2 ** (col - 1) # total de proposiciones a comparar para iterarlas
        tabla = []
        tabla.append(a) # las agrega a la tabla
        while renglones > 0:
            for x in range(num_iteracion):
                tabla.append("V")
                renglones -= 1
            for x in range(num_iteracion):
                tabla.append("F")
                renglones -= 1
            matriz.append(tabla)
        col = col - 1
```

Imagen 10. Código de la función validacion.

en almacenar primero las proposiciones negadas en caso de haberlas y hace su negación. Estas mismas negaciones se transforman en su inversa y se almacena en la lista de la tabla.

El validador de cada operador (conjunción, disyunción, negación, bicondicional e implicación) es parecido entre sí. No tienen argumentos previos, pero sí distintos valores de retorno que permanecen sin cambios por su declaración global. Prácticamente se itera la cadena y se buscan coincidencias con el operador AND para su respectiva equivalencia.

```

def OperadoresLogicos():
    global enunciado, posicion, posicion2, tabla_verdad, lista, posicion_argumento, matriz, argumentos, encabezado, a, end, e
    vueltas = 0
    lista = list(enunciado) # se pone en lista la proposición
    simbolos = ["^", "v", ">", "≡"]
    a = 0
    argumentos_negados = []
    encabezado = argumentos + argumentos_negados

    # Identificar posición de su argumento negativo
    posicion_argumento = []
    for arg in argumentos_negados:
        if arg[1] in argumentos:
            a = argumentos.index(arg[1])
            posicion_argumento.append(a)
    # Hacer tabla de verdad de negación
    a = 0
    for arg in argumentos_negados: ##Cada negación se transforma en su inverso
        tabla_verdad = []
        tabla_verdad.append(arg)
        posicion = posicion_argumento[a]
        for x in range(len(matriz[posicion])):
            if matriz[posicion][x] == "V":
                tabla_verdad.append("F")
            if matriz[posicion][x] == "F":
                tabla_verdad.append("V")
        a += 1
        matriz.append(tabla_verdad) # se añade a la matriz principal que sera la que se va a imprimir

# Conjunción
a = 0
while a != len(lista):
    posicion_argumento = []
    if lista[a] == "^":
        if lista[a + 1] in encabezado and lista[a - 1] in encabezado:
            JuntarParentesisExteriores()
            for x in range(len(matriz[posicion])):
                if matriz[posicion][x] == "V" and matriz[posicion2][x] == "V":
                    tabla_verdad.append("V")
                if matriz[posicion][x] == "V" and matriz[posicion2][x] == "F":
                    tabla_verdad.append("F")
                if matriz[posicion][x] == "F" and matriz[posicion2][x] == "V":
                    tabla_verdad.append("F")
                if matriz[posicion][x] == "F" and matriz[posicion2][x] == "F":
                    tabla_verdad.append("F")
            matriz.append(tabla_verdad)
        elif lista[a + 1] == "(" and lista[a - 1] == ")":
            e += 1
            pass
        elif lista[a - 1] == ")" and lista[a + 1] == "~":
            e += 1
            pass
        elif lista[a - 1] in encabezado and lista[a + 1] == "~":
            e += 1
            pass
        elif lista[a - 1] in encabezado and lista[a + 1] == "(":
            e += 1

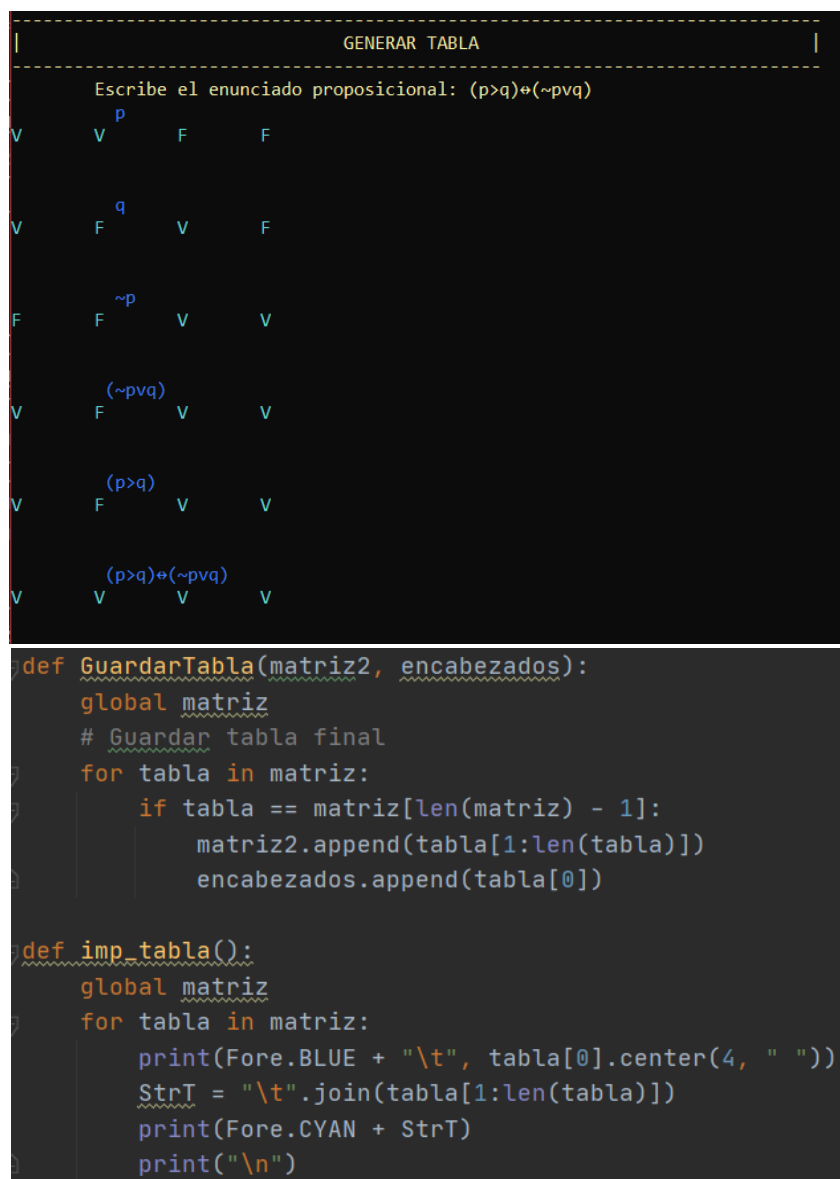
```

Imagen 11. Código de la función OperadoresLogicos. No se muestra completa (arriba) debido a que es muy extensa; la parte que valida la conjunción (abajo) se muestra una breve descripción de su funcionamiento.

- **Función GuardarTabla e imp_tabla**

La primera únicamente llama a los datos almacenados globalmente que se dieron como resultado de la validación de la función pasada (**OperadoresLogicos**) y los itera para imprimirlos.

En la segunda, **imp_tabla**, se llama a la matriz iterada en la función anterior y la acomoda en renglones para formar una tabla en orientación vertical (las tablas se leen de arriba hacia abajo, como se indica al usuario en la sección de instrucciones del generados). A la tabla se le da un formato de color distinto (tonos azules) al resto del texto en el programa (imagen 4).



```

|----- GENERAR TABLA -----|
Escribe el enunciado proposicional: (p>q)<math>\oplus</math>(<math>\sim</math>p<math>\vee</math>q)
V   p
V   V   F   F
V   q
F   F   V   F
F   <math>\sim</math>p
F   F   V   V
V   (<math>\sim</math>p<math>\vee</math>q)
F   F   V   V
V   (p>q)
F   F   V   V
V   (p>q)<math>\oplus</math>(<math>\sim</math>p<math>\vee</math>q)
V   V   V   V

def GuardarTabla(matriz2, encabezados):
    global matriz
    # Guardar tabla final
    for tabla in matriz:
        if tabla == matriz[len(matriz) - 1]:
            matriz2.append(tabla[1:len(tabla)])
            encabezados.append(tabla[0])

def imp_tabla():
    global matriz
    for tabla in matriz:
        print(Fore.BLUE + "\t", tabla[0].center(4, " "))
        StrT = "\t".join(tabla[1:len(tabla)])
        print(Fore.CYAN + StrT)
        print("\n")
  
```

Imagen 12. Impresión de prueba de una tabla de verdad (arriba); código de las funciones GuardarTabla e imp_tabla (abajo).

Calculadora de conjuntos

A nivel usuario

Al entrar al submenú de “Calculadora de conjuntos” con la letra “a” (imagen 2), se desplegarán tres posibles acciones: a) Instrucciones, b) Calcular conjuntos, y c) Volver al menú principal. Se sigue la misma lógica de teclear la letra correspondiente a la opción que se desea ejecutar. La opción “a” sólo imprimirá un listado de instrucciones, consejos para el funcionamiento del apartado. La opción “b” llevará a la sección de entrada de texto para escribir los elementos de cada conjunto y posteriormente a otro submenú para seleccionar la operación que se quiere realizar. La última opción, “c”, retorna al menú principal.

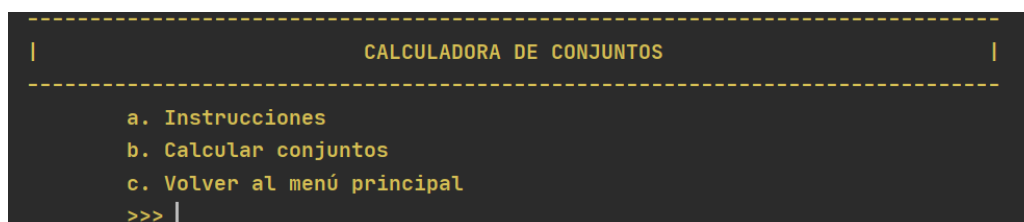


Imagen 13. Submenú de la calculadora de conjuntos.

Documentación técnica y funcionalidad

Ésta es considerada la segunda herramienta que se integra a nuestra “Caja de Herramientas de FCC”. Su objetivo es llevar a cabo operaciones a partir de los elementos proporcionados en cada conjunto, a continuación, se presenta la información calculable a partir de este programa:

Cardinalidad $ C $	Unión (\cup)	Intersección (\cap)	Diferencia $(-)$	Diferencia Simétrica (Δ)
-----------------------	-------------------	--------------------------	---------------------	------------------------------------

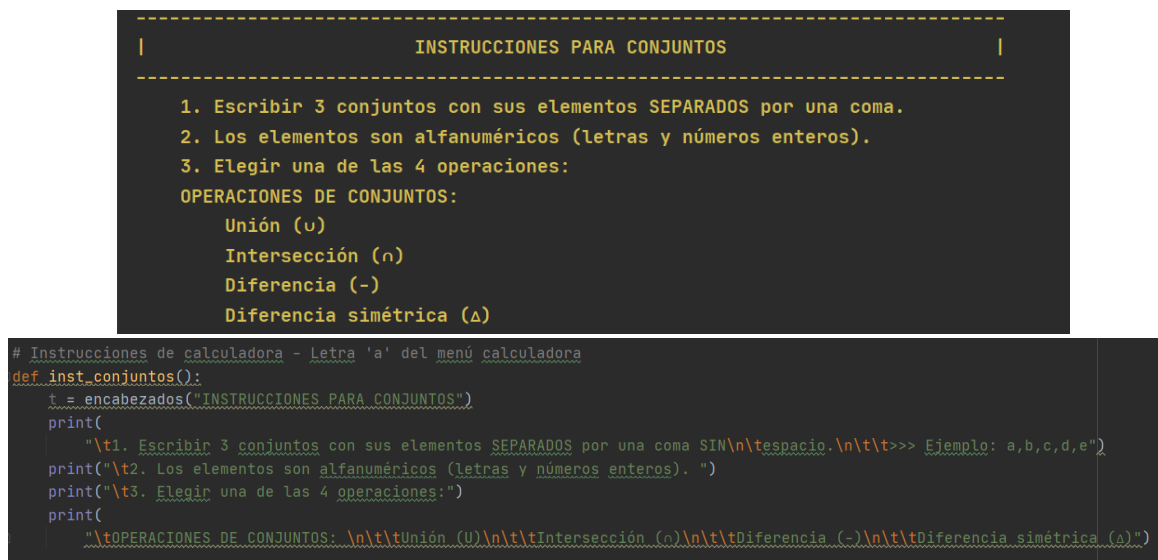
A manera de adelanto, la calculadora de conjuntos funciona de la siguiente manera:

- Se deben de ingresar los elementos de cada conjunto, separándolos por coma. Posteriormente, esa entrada de texto es convertida a una lista donde se eliminan las comas introducidas.
- El proceso que realiza la calculadora es relativamente sencillo, pues sólo debe de ordenarse la lista correspondiente a cada conjunto para así eliminar si es que existe algún elemento repetido y este no interfiera en las operaciones a realizar.

- Por cada conjunto se hacen búsquedas y comparaciones gracias a lo que contengan en cada índice de la lista, por lo que solo se necesita conocer bien la funcionalidad de cada operación para obtener el resultado deseado.
- **Operación de unión:** Se realiza la suma de 2 listas de conjuntos.
- **Operación de intersección:** Tomando en cuenta los elementos del conjunto 1 se debe de hacer una comparación y condicionar, a manera de que si no se encuentran en la lista a mostrar de esa intersección junto con el conjunto que se desea realizar la comparación entonces pasa a ser resultado de esa operación.
- **Operación de diferencia:** Tomando en cuenta los 2 conjuntos a los cuales se les requiere aplicar la operación basta con confirmar que si esta en el primer conjunto no aparezca en el segundo.
- **Operación de diferencia simétrica:** Tomando en cuenta los 2 conjuntos a los cuales se les requiere aplicar la operación se tomarán en cuenta a los elementos que estén en el primer conjunto, pero no en el segundo a la vez que en sentido contrario, es decir los elementos que están en el segundo conjunto pero no en el primero.

- **Función inst_conjuntos**

Al seleccionar la opción “a” del menú de la calculadora de conjuntos, esta función se encargará de imprimir en consola las indicaciones necesarias para que el programa corra adecuadamente. No tiene argumentos o valores de retorno.



```

|          INSTRUCCIONES PARA CONJUNTOS          |
|-----|
1. Escribir 3 conjuntos con sus elementos SEPARADOS por una coma.
2. Los elementos son alfanuméricos (letras y números enteros).
3. Elegir una de las 4 operaciones:
OPERACIONES DE CONJUNTOS:
    Unión (u)
    Intersección (∩)
    Diferencia (-)
    Diferencia simétrica (Δ)

# Instrucciones de calculadora - Letra 'a' del menú calculadora
def inst_conjuntos():
    t = encabezados("INSTRUCCIONES PARA CONJUNTOS")
    print(
        "\t1. Escribir 3 conjuntos con sus elementos SEPARADOS por una coma SIN\n\tespacio.\n\t\t>>> Ejemplo: a,b,c,d,e")
    print("\t2. Los elementos son alfanuméricos (letras y números enteros). ")
    print("\t3. Elegir una de las 4 operaciones:")
    print(
        "\tOPERACIONES DE CONJUNTOS: \n\t\tUnión (U)\n\t\tIntersección (∩)\n\t\tDiferencia (-)\n\t\tDiferencia simétrica (Δ)")
  
```

Imagen 14. Impresión de las instrucciones para la calculadora de conjuntos (arriba); código de la función `inst_conjuntos` (abajo).

• Función conjuntos

Esta es la función principal de este módulo del programa, ya que dentro de esta misma el proceso se realiza de manera completa de inicio a fin. Se ingresan los elementos de cada conjunto donde posteriormente se debe de tomar la decisión de cuál es la operación que se desea mostrar.

```

-----
|                                CALCULAR CONJUNTOS                                |
-----

Introduzca su conjunto A: 1,1,4,1
Introduzca su conjunto B: a,b,c,2,3,4,5,6
Introduzca su conjunto C: 5,6,7,8,9,a

Seleccione el inciso de la operación a realizar:
a. Unión (u)
b. Intersección (∩)
c. Diferencia (-)
d. Diferencia simétrica (Δ)
>>> d

-----
|                                DIFERENCIA SIMÉTRICA (Δ) DE 'A', 'B' Y 'C'                                |
-----

CARDINALIDAD:      |A| = 2          |B| = 8          |C| = 6

-----
DIFERENCIA SIMÉTRICA:
AΔB / BΔA : ['1', '2', '3', '5', '6', 'a', 'b', 'c']
AΔC / CΔA : ['1', '4', '5', '6', '7', '8', '9', 'a']
BΔC / CΔB : ['2', '3', '4', '7', '8', '9', 'b', 'c']
-----

# Calculadora de conjuntos
def conjuntos():
    A = str(input("\tIntroduzca su conjunto A: "))
    B = str(input("\tIntroduzca su conjunto B: "))
    C = str(input("\tIntroduzca su conjunto C: "))

    conjuntoA = []
    conjuntoB = []
    conjuntoC = []
    conjuntoA = A.split(",")
    conjuntoB = B.split(",")
    conjuntoC = C.split(",") # Aquí se separan los elementos de la lista cada que se detecta una coma

    # Menú para elegir operación para los conjuntos escritos previamente
    print("\n\tSeleccione el inciso de la operación a realizar:")
    print("\t\t\ta. Unión (U)\n\t\t\tb. Intersección (∩)\n\t\t\tc. Diferencia (-)\n\t\t\td. Diferencia simétrica (Δ)")
    eleccion = "-"
    while eleccion != "a" or eleccion != "b" or eleccion != "c" or eleccion != "d":
        eleccion = input("\t\t>>> ").lower()
        if eleccion == "a" or eleccion == "b" or eleccion == "c" or eleccion == "d":
            break
    if eleccion == "a": # Valores para el encabezado de cada cálculo
        t = "UNIÓN (U) DE 'A', 'B' Y 'C'"
    elif eleccion == "b":
        t = "INTERSECCIÓN (∩) DE 'A', 'B' Y 'C'"
    elif eleccion == "c":
        t = "DIFERENCIA (-) DE 'A', 'B' Y 'C'"
    elif eleccion == "d":

```

Imagen 15. Muestra del ingreso de datos para calcular los conjuntos (arriba); parte del código de la función conjuntos que valida las agrupaciones ingresadas (abajo).

```

# Cardinalidad
print("    CARDINALIDAD: \t", end="")
cA = set(sorted(conjuntoA))
list_cA = list(cA)
if list_cA[0] == "": # Si se detecta que el primer índice la lista está vacío, la cardinalidad será 0
    print("    |A| = 0", end="")
else:
    print("    |A| =", len(cA), end="")

cB = set(sorted(conjuntoB))
list_cB = list(cB)
if list_cB[0] == "":
    print("\t |B| = 0", end="")
else:
    print("\t |B| =", len(cB), end="")

cC = set(sorted(conjuntoC))
list_cC = list(cC)
if list_cC[0] == "":
    print("\t |C| = 0", end="")
else:
    print("\t |C| =", len(cC), end="")
print("\n    " + "-" * 72 + " ")

if eleccion == "a":
    print("\tUNIÓN:") # Enlista los valores de los dos conjuntos que se están comparando
    print("\tAUB / BUA : ", sorted(set(conjuntoA + conjuntoB)))
    print("\tAUC / CUA : ", sorted(set(conjuntoA + conjuntoC)))
    print("\tBUC / CUB : ", sorted(set(conjuntoB + conjuntoC)))
    print("\tAU(BUC) : ", sorted(set(conjuntoA + sorted(set(conjuntoB + conjuntoC)))))

elif eleccion == "b":
    # Intersección
    print("\tINTERSECCIÓN:")
    AinterseccionB = []
    AinterseccionC = []
    BinterseccionC = []
    interseccionABC = []
    # intersección a y b
    for i in conjuntoA:
        if (i not in AinterseccionB) and (i in conjuntoB):
            AinterseccionB.append(i) # Añade a la lista los elementos que comparten A y B sin repetirlos
    # intersección a y c
    for i in conjuntoA:
        if (i not in AinterseccionC) and (i in conjuntoC):
            AinterseccionC.append(i) # Añade a la lista los elementos que comparten A y C sin repetirlos
    # intersección b y c
    for i in conjuntoB:
        if (i not in BinterseccionC) and (
            i in conjuntoC): # Añade a la lista los elementos que comparten C y B sin repetirlos
            BinterseccionC.append(i)
    for i in AinterseccionB:
        if (i not in interseccionABC) and (
            i in conjuntoC): # Compara la intersección de A y B con los elementos de C y los añade a la lista
            interseccionABC.append(i)

```

Imagen 16. Continuación del código de la función conjuntos. Arriba se muestra el proceso para la cardinalidad; abajo el proceso de UNIÓN e INTERSECCIÓN. No se muestra la INTERSECCIÓN, DIFERENCIA ni DIFERENCIA SIMÉTRICA por espacio.

Sucesiones

A nivel usuario

Al entrar en el apartado de “Sucesiones” desde el menú principal con la letra “c”, se desplegará un submenú con cuatro opciones: a) Instrucciones, b) Calcular sucesión, c) Determinar tipo de sucesión y d) Volver al menú principal. El usuario tecleará una de estas cuatro letras y se desplegará lo correspondiente a su opción.

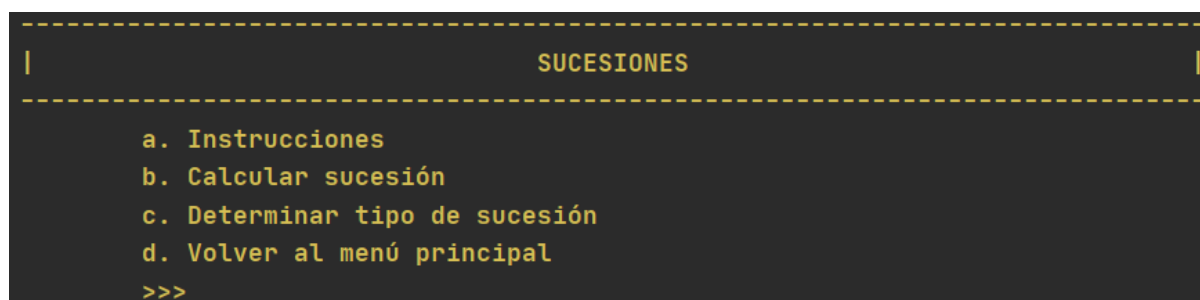


Imagen 17. Submenú de "Sucesiones".

Para “a” únicamente se mostrará un listado de reglas e instrucciones básicas para hacer uso de la herramienta. La opción “b” llevará al cálculo directo de las sucesiones (el usuario ingresará los límites inferior y superior de una sucesión preestablecida previamente elegida); la letra “c” los llevará a un clasificador de sucesiones (dependiendo de los elementos ingresados determinará si es geométrica o aritmética); y, por último, la letra “d” lo hará regresar al menú principal.

Documentación técnica y funcionalidad

Esta tercera herramienta añadida al ToolKit tiene como propósito calcular:

- a) Los “k” elementos de una sucesión;
- b) La sumatoria de todos los elementos de dicha sucesión;
- c) La multiplicación de todos los elementos de dicha sucesión;
- d) Determinar si una sucesión es del tipo aritmética o geométrica dependiendo de los elementos que la conforman.

- **Función `inst_sucesiones`**

Al seleccionar la letra “a” del submenú de sucesiones, se mandará a llamar a esta función en el código global de nuestro programa. Tiene como objetivo imprimir en pantalla las instrucciones pertinentes para el correcto funcionamiento de la herramienta de sucesiones. No recibe ni retorna ningún tipo de valor, por lo que es sólo ilustrativa.

```

|-----|
|              INSTRUCCIONES PARA SUCESIONES              |
|-----|

PARA CÁLCULO DE SUCESIONES:
1. Escribir la fórmula en términos de 'k', los operadores son:
    '**' exponente      '*' multiplicación
    '/' división        '-' resta
    '+' suma            '()' jerarquía
2. Escribir enteros para los límites superior (n) e inferior (m); m < n.
3. 0 si se quiere calcular una preestablecida, ingresar opción y límites.

PARA DETERMINAR TIPO DE SUCESIÓN:
1. Introduce el número de elementos que usarás para la comprobación
   (deben de ser mínimo 4).
2. Introduce los elementos de la sucesión, sólo ENTEROS.
  
```

```

# Instrucciones Sucesiones - Letra 'a' del submenú de sucesiones
def inst_sucesiones():
    t = encabezados("INSTRUCCIONES PARA SUCESIONES")
    print("\tPARA CÁLCULO DE SUCESIONES: ")
    print("\t1. Escribir la fórmula en términos de 'k', los operadores son:")
    print("\t\t '**' exponente\t\t '*' multiplicación\n\t\t '/' división\t\t '-' resta\n\t\t '+' suma\t\t\t '()' jerarquía")
    print("\t2. Escribir enteros para los límites superior (n) e inferior (m); m < n.")
    print("\t3. 0 si se quiere calcular una preestablecida, ingresar opción y límites.")
    print("\n\tPARA DETERMINAR TIPO DE SUCESIÓN: ")
    print("\t1. Introduce el número de elementos que usarás para la comprobación \n\t\t (deben de ser mínimo 4).")
    print("\t2. Introduce los elementos de la sucesión, sólo ENTEROS.")
  
```

Imagen 18. Impresión de las instrucciones para sucesiones (arriba); código de la función `inst_sucesiones` (abajo).

- **Función `lista_sucesiones`**

Al seleccionar la letra “b” del submenú de sucesiones se llama a mandar la función correspondiente a un listado, la cual destacan 2 variantes principales; ya sea introducir una formula personalizada por el usuario en términos de k o alguna de las fórmulas previamente establecidas.

```

-----
|                                CALCULAR SUCESIÓN                                |
-----

Seleccione la sucesión a iterar:
1. Escribir fórmula
2. 1/k          (Armónica)
3. (-1)^k/(k+1)
4. 1/k(k+1)
5. 1+(1/2)^k
6. 1/k^(n-1)   (Zenon)
>>>

def lista_sucesiones():
    t = encabezados("CALCULAR SUCESIÓN")
    print("\tSeleccione la sucesión a iterar: ")
    print("\t1. Escribir fórmula") #op_sucesiones()
    print("\t2. 1/k          (Armónica)") #suc2()
    print("\t3. (-1)**k/(k+1)") #suc3
    print("\t4. 1/k(k+1)") #suc4
    print("\t5. 1+(1/2)**k") #suc5
    print("\t6. 1/k^(n-1)   (Zenon)")
    s = input("\t>>> ")
    s = int(s)
    while s < 0 or s > 10:
        s = input("\t>>>")
        s = int(s)
    return s

```

Imagen 19. Impresión del submenú de selección para calcular la sucesión (arriba); código de la función lista_sucesiones (abajo).

• Función op_sucesiones

Al seleccionar la opción “1” del submenú calcular sucesiones, el programa solicita al usuario introducir tanto su límite inferior como el exterior. Posteriormente, se debe de introducir la formula deseada siguiendo cuidadosamente las instrucciones previamente señaladas en uno de los submenús anteriores.

```

Introduzca límite inferior (n): 2
Introduzca límite superior (m): 7
Escriba la fórmula en función de 'k': k*(2*k)**2
-----
|                                k*(2*k)**2                                |
-----

m = 2 | n = 7

Fórmula: k*(2*k)**2 con k = 7   Resultado: 1372
Fórmula: k*(2*k)**2 con k = 6   Resultado: 864
Fórmula: k*(2*k)**2 con k = 5   Resultado: 500
Fórmula: k*(2*k)**2 con k = 4   Resultado: 256
Fórmula: k*(2*k)**2 con k = 3   Resultado: 108
Fórmula: k*(2*k)**2 con k = 2   Resultado: 32
> Sumatoria = 3132
> Multiplicación = 524386566144000

```

Imagen 20. Función personalizada introducida por el usuario.

La funcionalidad de esta función es a partir de un ciclo for y acumuladores, que permiten llevar la suma y multiplicación de los términos. *Eval* juega una parte muy importante, pues ejecuta e interpreta el string proporcionado por el usuario

```
def op_sucesiones(inf, sup, form):
    t = encabezados(f"{form}")
    print(f"\tm = {inf} | n = {sup}\n")
    original = form
    n = sup
    suma = 0
    mult = 1
    for i in range(0, sup-inf+1):
        k = sup-i
        print(f"\tFórmula: {original} con k = {n} Resultado: {eval(form)}")
        suma += eval(form)
        mult *= eval(form)
        n -= 1
    print("\t> Sumatoria = ", suma, "\n\t> Multiplicación = ", mult)
```

Imagen 21. Código de la función op_sucesiones.

• Funciones de sucesiones

Las siguientes funciones tienen una estructura muy similar. Se conforman de ciclos “for” en los cuales se evalúan los límites superior (n) e inferior (m) otorgados por el usuario para llevar a cabo diferentes tipos de sucesiones previamente establecidas. Cada una de ellas mantiene la misma estructura.

○ Función suc2

```
Introduzca límite inferior (n): 4
Introduzca límite superior (m): 8

-----
|                               1/k                               |
-----
m = 4 | n = 8

Fórmula: 1/8 = 0.125
Fórmula: 1/7 = 0.14285714285714285
Fórmula: 1/6 = 0.16666666666666666
Fórmula: 1/5 = 0.2
Fórmula: 1/4 = 0.25
> Sumatoria: 0.8845238095238095
> Multiplicación: 0.00014880952380952382

def suc2(inf, sup):
    t = encabezados("1/k")
    print(f"\tm = {inf} | n = {sup}\n")
    suma = 0
    mult = 1
    for i in range(inf, sup+1):
        f = 1/sup
        print(f"\tFórmula: 1/{sup} = {f}")
        suma += 1/sup
        mult *= 1/sup
        sup -= 1
    print(f"\t> Sumatoria: {suma}\n\t> Multiplicación: {mult}")
```

Imagen 22. Resultado de la expresión preestablecida (arriba); código de la función suc2 (abajo).

○ Función suc3

```

    Introduzca límite inferior (n): 2
    Introduzca límite superior (m): 6

-----
|                                     (-1)^k/(k+1)                                     |
-----

m = 2      |      n = 6

Fórmula: (-1)^6/(6+1) = 0.14285714285714285
Fórmula: (-1)^5/(5+1) = -0.16666666666666666
Fórmula: (-1)^4/(4+1) = 0.2
Fórmula: (-1)^3/(3+1) = -0.25
Fórmula: (-1)^2/(2+1) = 0.33333333333333333
> Sumatoria: 0.2595238095238095
> Multiplicación: 0.0003968253968253968
def suc3(inf, sup):
    t = encabezados("(-1)^k/(k+1)")
    print(f"\tm = {inf}      |      n = {sup}\n")
    suma = 0
    mult = 1
    for i in range(inf, sup+1):
        f = ((-1)**(sup))/(sup+1)
        print(f"\tFórmula: (-1)^{sup}/{(sup)+1} = {f}")
        suma += ((-1)**(sup))/(sup+1)
        mult *= ((-1)**(sup))/(sup+1)
        sup -= 1
    print(f"\t> Sumatoria: {suma}\n\t> Multiplicación: {mult}")

```

Imagen 23. Cálculo de una expresión preestablecida (arriba); código de la función suc3 (abajo).

○ Función suc4

```

    Introduzca límite inferior (n): 5
    Introduzca límite superior (m): 10

-----
|                                     1/k(k+1)                                     |
-----

m = 5      |      n = 10

Fórmula: 1/10(10+1) = 0.00909090909090909
Fórmula: 1/9(9+1) = 0.011111111111111112
Fórmula: 1/8(8+1) = 0.013888888888888888
Fórmula: 1/7(7+1) = 0.017857142857142856
Fórmula: 1/6(6+1) = 0.023809523809523808
Fórmula: 1/5(5+1) = 0.03333333333333333
> Sumatoria: 0.10909090909090907
> Multiplicación: 1.988262570273152e-11
def suc4(inf, sup):
    t = encabezados("1/k(k+1)")
    print(f"\tm = {inf}      |      n = {sup}\n")
    suma = 0
    mult = 1
    for i in range(inf, sup+1):
        f = 1/(sup*(sup+1))
        print(f"\tFórmula: 1/{sup}({sup}+1) = {f}")
        suma += 1/(sup*(sup+1))
        mult *= 1/(sup*(sup+1))
        sup -= 1
    print(f"\t> Sumatoria: {suma}\n\t> Multiplicación: {mult}")

```

Imagen 24. Cálculo de una expresión preestablecida (arriba); código de la función suc4 (abajo).

○ Función suc5

```

Introduzca límite inferior (n): 4
Introduzca límite superior (m): 9

-----
|                                     1+(1/2)^k
|-----
|
m = 4      |      n = 9

Fórmula: 1+(1/2)^9 = 1.001953125
Fórmula: 1+(1/2)^8 = 1.00390625
Fórmula: 1+(1/2)^7 = 1.0078125
Fórmula: 1+(1/2)^6 = 1.015625
Fórmula: 1+(1/2)^5 = 1.03125
Fórmula: 1+(1/2)^4 = 1.0625
> Sumatoria: 6.123046875
> Multiplicación: 1.1280973674456618

def suc5(inf, sup):
    t = encabezados("1+(1/2)^k")
    print(f"\tm = {inf}      |      n = {sup}\n")
    suma = 0
    mult = 1
    for i in range(inf, sup + 1):
        f = 1 + ((1/2)**sup)
        print(f"\tFórmula: 1+(1/2)^{sup} = {f}")
        suma += 1 + ((1/2)**sup)
        mult *= 1 + ((1/2)**sup)
        sup -= 1
    print(f"\t> Sumatoria: {suma}\n\t> Multiplicación: {mult}")

```

Imagen 25. Cálculo de una expresión preestablecida (arriba); código de la función suc5 (abajo).

○ Función suc6

```

Introduzca límite inferior (n): 2
Introduzca límite superior (m): 7

-----
|                                     Zenon 1/2^(k-1)
|-----
|
m = 2      |      n = 7

Fórmula: 1/2^(7-1) = 0.015625
Fórmula: 1/2^(6-1) = 0.03125
Fórmula: 1/2^(5-1) = 0.0625
Fórmula: 1/2^(4-1) = 0.125
Fórmula: 1/2^(3-1) = 0.25
Fórmula: 1/2^(2-1) = 0.5
> Sumatoria: 0.984375
> Multiplicación: 4.76837158203125e-07

def suc6(inf, sup):
    t=encabezados("Zenon 1/2^(k-1)")
    print(f"\tm = {inf}      |      n = {sup}\n")
    suma = 0
    mult = 1
    for i in range(inf, sup+1):
        f = 1/(2**(sup-1))
        print(f"\tFórmula: 1/{2}^{(sup)-1} = {f}")
        suma += 1/(2**(sup-1))
        mult *= 1/(2**(sup-1))
        sup -= 1
    print(f"\t> Sumatoria: {suma}\n\t> Multiplicación: {mult}")

```

Imagen 26. Cálculo de la serie de Zenon (arriba); código de la función suc6 (abajo).

- **Función determinar_tipo**

Al ingresar a la letra “c” del submenú de sucesiones se puede acceder esta función (imagen 11). Su objetivo es determinar, por medio de elementos ingresados por el usuario, si la función a la que estos pertenecen es del tipo aritmética o geométrica.

```

-----
|                               DETERMINAR EL TIPO DE SUCESIÓN                               |
-----

¿Con cuántos elementos quieres hacer la comprobación? 2
[!] Debes de incluir mínimo 4 términos de la sucesión
¿Con cuántos elementos quieres hacer la comprobación? 5
>>>Elemento 1 : 2
>>>Elemento 2 : 4
>>>Elemento 3 : 6
>>>Elemento 4 : 8
>>>Elemento 5 : 10
La sucesión es ARITMÉTICA

```

Imagen 27. Ejemplo de ejecución de la función "determinar_tipo".

Es importante recalcar que primero validará que los elementos introducidos sean cuatro o más. De lo contrario, seguirá pidiendo este valor hasta que se cumpla su condición. En segundo lugar, los elementos introducidos por el usuario deberán ser del tipo entero (tal y como se especificó en las instrucciones de la herramienta de sucesiones). Una vez introducidos procederán a guardarse en una lista, la cual irá hacia la siguiente función: **tipo2**.

```

def determinar_tipo():
    t=encabezados("DETERMINAR EL TIPO DE SUCESIÓN")
    while True:
        try:
            elementos=int(input("\t¿Con cuántos elementos quieres hacer la comprobación? "))
            if (elementos>=4):
                break
            else:
                print("\t[!] Debes de incluir mínimo 4 términos de la sucesión")
        except ValueError:
            print("\t[!] Debes de incluir mínimo 4 términos de la sucesión")
    s = list([])
    c=1
    for i in range(elementos):
        print("\t\t>>>Elemento",c,"": "end=")
        a=int(input())
        c+=1
        s.append(a)
    tipo2(s)
# ---- Terminan funciones de SUCESIONES E INDUCCIÓN ----

```

Imagen 28. Código de la función determinar_tipo.

- **Función tipo2**

Esta función es el complemento para que la función anterior arroje un resultado. La lista que tiene almacenados los elementos de la sucesión son su valor de entrada, mas no retorna ningún valor, sólo imprime si la sucesión es geométrica o aritmética. El proceso que lleva a cabo es simple: compara los primeros tres índices del listado. Si sus diferencias no son iguales, entonces por defecto se sabe que no es aritmética, pues se trata de una razón de cambio y no de una constante que se suma o resta.

```
def tipo2(lista):  
    if lista[1]-lista[0]==lista[2]-lista[1]:  
        print("\t\t\t\t\t La sucesión es ARITMÉTICA")  
    else:  
        print("\t\t\t\t\t La sucesión es GEOMÉTRICA")
```

Imagen 29. Código de la función tipo2.

Relaciones y funciones

A nivel usuario

Al entrar en el apartado de “Relaciones y funciones” desde el menú principal con la letra “d”, se desplegará un submenú con tres opciones: a) Instrucciones, b) Determinar sucesión y c) Volver al menú principal. El usuario tecleará una de estas cuatro letras y se desplegará lo correspondiente a su opción.

Para “a” únicamente se mostrará un listado de reglas e instrucciones básicas para hacer uso de la herramienta. La opción “b” hará que el usuario ingrese un conjunto de pares entre paréntesis [(2,3),(0,2) por ejemplo] para determinar si es una función y sus propiedades; mientras que la letra “c” lo hará regresar al menú principal.

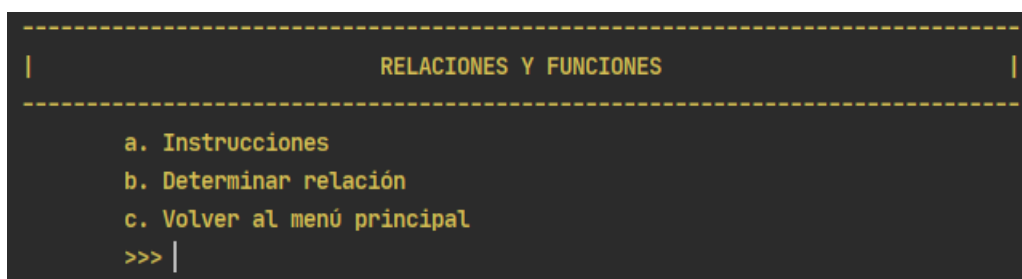


Imagen 30. Submenú de “Relaciones y funciones”.

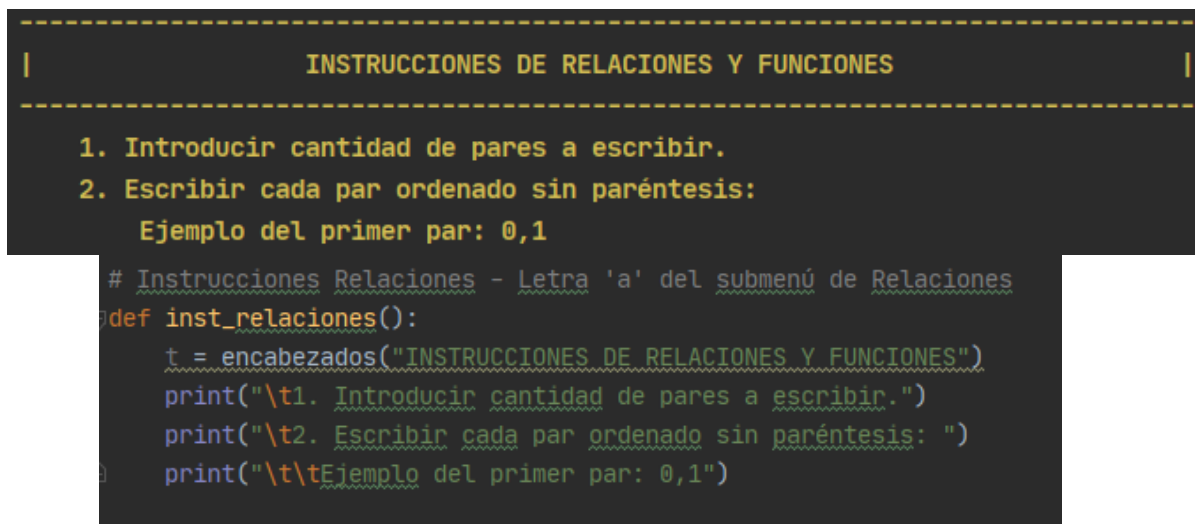
Documentación técnica y funcionalidad

Esta cuarta herramienta añadida al ToolKit busca determinar si una relación de parejas, ingresadas como si fueran coordenadas, se clasifica como función o no. Además de mostrar su dominio, codominio y si cuenta con las siguientes propiedades:

Sea R una relación definida sobre el conjunto A , es...		
Reflexiva si...	Simétrica si...	Transitiva si...
Para toda 'x' en A , hay un (x, x) .	Para toda 'x' y 'y' en A , si hay un punto (x, y) , habrá uno (y, x) .	Para toda 'x', 'y', 'z' en A , si (x, y) , entonces hay un (y, z) .

- **Función `inst_relaciones`**

Al seleccionar la letra “a” del submenú de relaciones, se mandará a llamar a esta función en el código global de nuestro programa. Tiene como objetivo imprimir en pantalla la manera en la que se deben recibir los datos para usar el resto de la utilidad en la sección. No recibe ni retorna ningún tipo de valor.



```
# Instrucciones Relaciones - Letra 'a' del submenú de Relaciones
def inst_relaciones():
    t = encabezados("INSTRUCCIONES DE RELACIONES Y FUNCIONES")
    print("\t1. Introducir cantidad de pares a escribir.")
    print("\t2. Escribir cada par ordenado sin paréntesis: ")
    print("\t\tEjemplo del primer par: 0,1")
```

Imagen 31. Impresión de las instrucciones de "Relaciones y funciones" (arriba); código de la función `inst_relaciones` (abajo).

- **Función `tipo_relaciones`**

Al seleccionar la letra “b” del submenú de relaciones, se mandará a llamar a esta función en el código global de nuestro programa. En la cual se piden el numero de pares a introducir junto con todos los parámetros que son necesarios para realizar de una manera correcta cada una de las diferentes propiedades de las relaciones (Reflexividad, simetría, transitividad, dominio y rango).

```
-----
|                               DETERMINAR RELACIÓN                               |
|-----|

Cantidad de pares ordenados a escribir >>> 3
Formato requerido: x,y

Ingresar el 1° par >>> 1,0
Ingresar el 2° par >>> 2,3
Ingresar el 3° par >>> 3,3

No es reflexiva
No es simétrica
Sí es transitiva

Dominio: {1, 2, 3}
Rango: {0, 3}
-> Es una función
```

```
def tipo_relaciones():
    # Pide cuántos pares separados escribirá el usuario
    cantidad = int(input("\tCantidad de pares ordenados a escribir >>> "))
    pares, x, y = [], 0, 0
    print("\tFormato requerido: x,y\n")

    for i in range(cantidad):
        x, y = input("\tIngresar el %d° par >>> " % (i + 1)).split(",")
        pares.append((x, y)) # Almacena los pares en dos listas distintas

    dominio = []
    codominio = []

    for i, j in pares: # Itera la lista de los pares para obtener dominio
        y rango
        if (i not in dominio):
            dominio.append(i)

        if (j not in codominio):
            codominio.append(j)
    print("\n")
    # PARTE REFLEXIVA
    reflexiva = True
    for i in dominio:
        if ((i, i) not in pares):
            reflexiva = False
            break

    if (reflexiva == True):
        print("\tSí es reflexiva")
    else:
        print("\tNo es reflexiva")

    # PARTE SIMÉTRICA
    simetria = True
    for i, j in pares:
        if ((j, i) not in pares):
            simetria = False
            break
```

```

if (reflexiva == True):
    print("\tSí es simétrica")
else:
    print("\tNo es simétrica")

# PARTE TRANSITIVA
transitiva = True
pares.sort()
h1, h2, h3 = 0, 0, 0
for i in range(len(pares)):
    h1 = pares[i][0]
    h2 = pares[i][1]
    for j in range(len(pares)):
        if (pares[j][0] == h2):
            h3 = pares[j][1]
            if ((h1, h3) not in pares):
                transitiva = False
                break
    if (transitiva == False):
        break

if (transitiva == True):
    print("\tSí es transitiva")
else:
    print("\tNo es transitiva")

dominio.sort()
f_dominio = ", ".join(dominio)
codominio.sort()
f_codominio = ", ".join(codominio)
print("\n\tDominio: {" + f_dominio + "}\n\tRango: {" + f_codominio +
"}")
# Imprime el dominio y codominio en forma de string, separados por
comas

# Convierte la lista de pares en un string para quitar los caracteres
que no sean numéricos
pares = str(pares)
numeros = ' '.join(filter(str.isalnum, pares))
numeros_lista = numeros.split(" ") # Convierte en lista dichos números
longitud = len(numeros_lista)
x = []
y = []
cont_x = 0

for i in range(0, longitud - 1, 2): # Obtiene los valores de las
abscisas
    if numeros_lista[i] not in x:
        x.append(numeros_lista[i])
    else:
        cont_x += 1 # Suma 1 si una 'x' se repite
for j in range(1, longitud, 2): # Obtiene los valores de las ordenadas
    if numeros_lista[j] not in y:
        y.append(numeros_lista[j])
if cont_x != 0:
    print("\t-> No es una función")
else:
    print("\t-> Es una función")

```

Imagen 32. Impresión de "Determinar Relación" (arriba); código de la función tipo_relciones (abajo).

Relación de la materia con nuestra carrera

Conclusiones

Como recordaremos, al hablar de relaciones nos referimos a una correspondencia entre dos elementos de conjuntos con ciertas propiedades. Es aquí donde la matemática nos explica mediante procesos de abstracción poder estudiarlas y expresarlas de una manera correcta.

Nosotras, al estar estudiando la carrera en Ing. en Sistemas Computacionales la aplicación es muy amplia, ya que, su aplicación aparece en bases de datos, estructuras de datos, redes y diversos lenguajes de programación. (Vidal, 2018).

A su vez, podemos encontrar una aplicación mucho más específica, que, de hecho, fue aplicada en la asignatura *Interconexión de Redes*, en ella se revisó un algoritmo denominado SPF (Shortest Path First), el cual está basado en las propiedades que presentan las relaciones. En él, se determinan costos de ruta acumulados a lo largo de cada ruta, y de esta manera se identifica cuál es el camino más corto, pues construye una base de datos de enlace-estado. (CCNA, 2020)

Hablando en términos generales de este proyecto, sin duda la realización de esta herramienta no fue sencilla, sin embargo, la buena distribución de tiempos permitió obtener un resultado como este. Como lo dice el nombre de esta asignatura, ahora tenemos los fundamentos de las Ciencias de la Computación para entender cómo es que funciona toda esa lógica detrás de este gran mundo.

Bibliografía

CCNA. (10 de Agosto de 2020). *Introducción a OSPF*. Recuperado el 13 de Mayo de 2021, de <https://ccnadesdecero.com/curso/ospf/>

Vidal, M. T. (2018). *Relaciones y Funciones*. Recuperado el 13 de Mayo de 2021, de <http://mtovar.cs.buap.mx/doc/MatDiscA/Funciones%20y%20Relaciones1.pdf>