

Programación



pythonTM

APUNTES DE PROGRAMACIÓN BÁSICA

2021



UNIVERSIDAD DE CONCEPCIÓN



Facultad de Ingeniería
Universidad de Concepción

ÍNDICE GENERAL

1. Cómo funciona un computador	1
1.1. Procesador	2
1.2. Almacenamiento	2
1.3. Memoria RAM	2
1.4. Placa base	3
2. Introducción a la programación básica en Python	4
2.1. Variables	5
2.2. Asignación de variables	6
2.3. Tipos de variables	7
2.4. Operadores de cálculo aritmético	8
2.5. Operadores lógicos y de comparación	9
2.6. Ingreso y salida de datos	10
2.7. Comandos de cambio de tipo de dato	12
2.8. Estructuras de selección y de repetición	13
2.8.1. Ejemplos de la estructura if-elif-else	13
2.8.2. Ejemplos del ciclo while	14
2.8.3. Ejemplos del ciclo for	15
2.9. Ejercicios de programación básica	17
2.10. Solución sugerida a los ejercicios	18
3. Funciones	20
3.1. Crear y llamar a una función	21
3.2. Funciones con retorno	22
3.3. Ejercicios usando funciones	23
3.4. Soluciones propuestas de los ejercicios	24
4. Strings	27
4.1. Operaciones y funciones básicas con strings	28
4.2. Indexación y cómo recorrer un string	28
4.2.1. Sub-cadena	29

4.3.	Métodos más comunes con strings	30
4.4.	Ejercicios usando strings	32
4.5.	Soluciones propuestas de los ejercicios	33
5.	Listas	35
5.1.	Crear listas	36
5.1.1.	Lista de listas	36
5.2.	Operadores básicos en listas	36
5.3.	Indexación en listas	37
5.4.	Funciones básicas para listas numéricas	38
5.5.	Métodos más comunes con listas	39
5.5.1.	Método split	40
5.5.2.	Método join	41
5.6.	Ejercicios	42
5.7.	Soluciones	43
6.	Arreglos	45
6.1.	Introducción al paquete Numpy	46
6.2.	Creación manual de arreglos	47
6.2.1.	Vectores con el comando array	47
6.2.2.	Matrices con el comando array	48
6.3.	Indexación en arreglos	49
6.4.	Funciones y operaciones con arreglos	50
6.4.1.	Creaciones de arreglos por medio de funciones	50
6.4.2.	Atributos de un arreglo	51
6.4.3.	Operaciones con arreglos	51
6.5.	Ejercicios	53
6.6.	Soluciones	54
7.	Archivos	56
7.1.	Creación de archivos	57
7.2.	Lectura de archivos	59
7.3.	Agregar información a un archivo	61
7.4.	Ejercicios	62
7.5.	Soluciones	63

ÍNDICE DE CUADROS

2.1.	Tipos de datos más comunes	7
2.2.	Operadores de cálculo aritméticos en Python	8
2.3.	Operadores lógicos en python	9
2.4.	Operadores de comparación en Python	9
2.5.	Comandos para cambiar tipo de datos más comunes	12
4.1.	Algunos métodos de análisis para strings	30
4.2.	Algunos métodos de transformación para strings	31
5.1.	Métodos para listas	39
6.1.	Funciones para crear arreglos	50
6.2.	Atributos de un arreglo A	51

CAPÍTULO

1

CÓMO FUNCIONA UN COMPUTADOR

Computador, con raíz del latín *computare*, es como llamamos al aparato que la mayoría dispone en sus hogares. Este, como dice su nombre, realiza cálculos en fracciones de segundo que permiten la realización de diversas tareas.

Entender cómo funciona un computador se vuelve necesario al momento de enfrentarse a problemas o nuevos desafíos. Así, en este capítulo se describen las partes más importantes que constituyen un computador y sus funciones dentro del mismo.

1.1. Procesador

Si simplificamos las cosas, se puede decir el procesador es el computador.

El procesador, también conocido como CPU, es quien lleva a cabo casi la totalidad del trabajo que un computador necesita, lo cual consiste en realzar cálculos y cumplir instrucciones; todo lo que ocurre en el dispositivo pasa por el procesador.

Este se encuentra formado por:

- Placa: Permite conectar con el PC, mediante unos conectores.
- placa de silicio: memoria

1.2. Almacenamiento

Las unidades de almacenamiento son las encargadas de guardar los datos, esto en lenguaje binario. Hoy, dada la alta velocidad de transferencia de datos que los usuarios requieren, suelen ser de uso común los discos SSD y memorias M.2, dejando ya de lado los clásicos discos duros mecánicos (HDD).

Esta unidad cumple la función de almacenar información para el largo plazo.

1.3. Memoria RAM

Para efectuar sus computos y dar vida al computador, el procesador debe ser alimentado constantemente de datos. Si bien estos tiene su unidad de almacenamiento, para el uso cotidiano, ese envío de datos sería muy lento y dificultaría un uso efectivo del dispositivo.

La memoria RAM, de la sigla en inglés *Random Access Memory*, fue concebida como solución al problema antes expuesto. Esta actúa como mediadora entre el almacenamiento y el procesador, permitiendo cargar datos para su uso en el corto plazo, agilizando el funcionamiento del sistema. Esta memoria almacena los datos de aquello que el procesador se encuentre ejecutando en el momento, en tiempo real, por ejemplo, este libro.

La capacidad de la RAM que nuestro dispositivo posea establece el límite de información disponible para su uso rápido, por ejemplo, para mantener muchas pestañas abiertas en el buscador de internet o jugar y permitirnos streamear.

1.4. Placa base

La placa base es la interfaz que permite unir los componentes antes mostrados, constando esta de una arquitectura compleja. Para la incorporación del procesador se encuentra el socket y para las memorias RAM se dispone de slots (o ranuras). el método de conexión de las unidades de almacenamiento depende del tipo de estas que se disponga: para discos mecánicos y de estado sólido se suelen utilizar una conexión mediante cable SATA, mientras para las memorias M.2, las placas actuales disponen de un slot dedicado.

CAPÍTULO

2

INTRODUCCIÓN A LA PROGRAMACIÓN BÁSICA EN PYTHON

Python es uno de los lenguajes de programación más utilizados en los últimos tiempos, siendo este, uno de los más versátiles y sencillos de aprender.

Hay una gran variedad de entornos de desarrollo para este lenguaje, uno de ellos es Jupyter notebook, que será usado en este libro para mostrar algunos ejemplos.

En este capítulo se presentarán la sintaxis de la asignación de variables, los operadores básicos matemáticos, los comandos de entrada y de salida de datos, el uso de comentarios, las estructuras de control de selección y de repetición, y algunos ejemplos del uso de cada uno de estos.

En la parte final de este y los demás capítulos se presentarán ejercicios que, a diferencia de los ejemplos, presentarán un formato como las tareas certámenes del curso y con una dificultad ascendente, es decir cada ejercicio con mayor dificultad que el anterior.

2.1. Variables

Para iniciar es necesario entender qué es una variable dentro del contexto de programación, dado que, a pesar de ser un nombre común dentro de distintas disciplinas, para nuestros intereses se refiere a algo particular.

Las variables son usadas para almacenar información, con la finalidad de ser manipulada y referenciada. Estas son comúnmente alojadas en la memoria RAM de nuestros dispositivos (VALIDAR CON EL PROFE).

Al generar una variable dentro de un entorno de programación, esta ha de contener

- Identificador único.
- Tipo de dato.
- Valor.

Así, la instrucción dada se puede resumir en: podemos llamar `a` al objeto número entero 2. La variable `a` es como una etiqueta que permite referenciar al objeto "2", más cómoda de recordar y utilizar que el identificador del objeto.

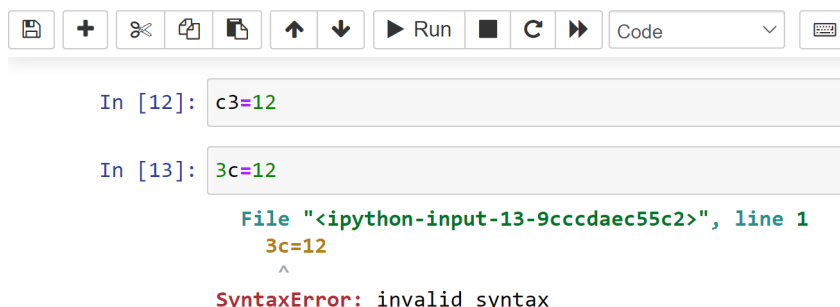
En Python, las variables actúan como etiquetas que establecen referencia a los datos, ya que estos son guardados en objetos. (REVISAR Y PREGUNTAR AL PROFE)

2.2. Asignación de variables

Existen ciertas condiciones para la asignación de variables, referido esto a la forma en que escribimos el código, siendo esto dado por el entorno en el cual se trabaje.

Python reconoce como variable a las cadenas de caracteres, es decir, conjunto de letras y números, que cumplan lo siguiente:

- Que no coincidan con el nombre de un comando del mismo entorno.
- Que no contenga caracteres especiales, tales como &, %, \$, #, -, entre otros. La única excepción es el guión bajo (_), que es de uso común para dar nombre a las variables.
- Que no inicien con un número.



```

In [12]: c3=12

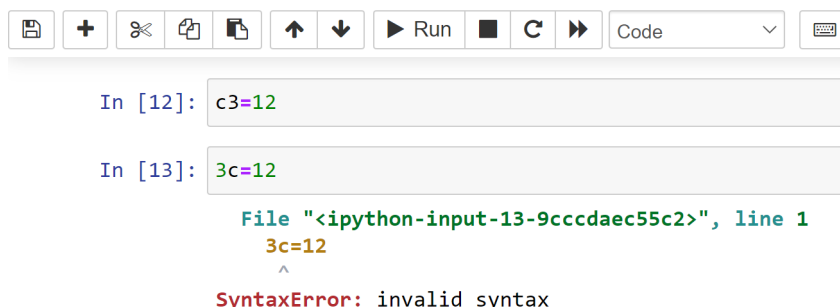
In [13]: 3c=12
File "<ipython-input-13-9cccdac55c2>", line 1
      3c=12
        ^
SyntaxError: invalid syntax

```

Figura 2.1: Declarando un valor a una variable con Jupyter notebook

Para asignar un valor o dato a una variable, luego de nombrarla se agrega un = seguido de la expresión a asignar. Esto se llama expresión de asignación.

Veamos un ejemplo asignando el valor 12 a la variable antes mencionada. Esto sería:



```

In [12]: c3=12

In [13]: 3c=12
File "<ipython-input-13-9cccdac55c2>", line 1
      3c=12
        ^
SyntaxError: invalid syntax

```

Figura 2.2: Declarando un valor a una variable con Jupyter notebook

2.3. Tipos de variables

En Python las variables pueden ser de distintos tipos, siendo los básicos los presentados a continuación:

Tipo	Significado
int	Un número entero
float	Un número en punto flotante
complex	Un número complejo
str	String o cadena de símbolos
list	Una lista

Cuadro 2.1: Tipos de datos más comunes

Para conocer el tipo de dato que es una variable se usa el comando `type()`

Los tipos de datos en un lenguaje de programación determinan los valores y las operaciones que se pueden aplicar a una variable. Por ejemplo `div` y `mod` son válidas para números enteros pero no para valores de punto flotante.

Python usa tipado dinámico, es decir, una variable puede tomar un tipo de dato a través de la asignación de un valor como resultado de la evaluación de una expresión aritmética, y luego, dentro del mismo programa, esa misma variable puede cambiar el tipo de dato al asignarle otro valor, resultado de la evaluación de otra expresión.

2.4. Operadores de cálculo aritmético

Para realizar operaciones aritméticas simples, ya sea suma, resta, multiplicación o división, la simbología de ellas en Python es casi igual a la usual.

Operación	Sintaxis
$a + b$	<code>a+b</code>
$a - b$	<code>a-b</code>
ab	<code>a*b</code>
a/b	<code>a/b</code>
a^b	<code>a**b</code>
$a \text{ div } b$	<code>a//b</code>
$a \text{ mod } b$	<code>a%b</code>

Cuadro 2.2: Operadores de cálculo aritméticos en Python

De las operaciones presentadas en el cuadro 2.2 es probable que las últimas dos sean poco recordadas o desconocidas. Por ello, a continuación, presentamos una breve explicación.

- División entera ($a \text{ div } b$): Consiste en realizar la división obteniendo como resultado solo la parte entera del cociente.
- Módulo ($a \text{ mod } b$): Esta operación entrega el resto de realizar una división entera.

Ahora, como ejemplo de aplicación de la sintaxis, calcularemos el siguiente valor:

$$3^2(1 + 3 - 2) \text{ mod } (5 \text{ div } 2)$$

```
In [25]: (3**2*(1+3-2))%(5//2)
Out[25]: 0
```

Figura 2.3: Sintaxis del cálculo en Jupyter Notebook

El resultado de la operación es 0, como muestra la figura 2.3.

2.5. Operadores lógicos y de comparación

Los operadores lógicos son aquellos que entregan un valor de verdad al trabajar con una proposición lógica, aquí en python los valores de verdad son **True** o **False**, en el siguiente cuadro se muestra la sintaxis de estos operadores y su significado:

Sintaxis del operador	Significado
<code>a and b</code>	La expresión es verdadera si se cumple a y se cumple b.
<code>a or b</code>	La expresión es verdadera si se cumple a o se cumple b.
<code>not a</code>	La expresión es verdadera si a no se cumple.

Cuadro 2.3: Operadores lógicos en python

Los operadores de comparación, como bien dice su nombre, se usan para comparar datos entre sí, si son numéricos se puede ver si son mayores o menores que otros, mientras que si son de cualquier tipo se puede comparar si son iguales o distintos. A continuación, se muestra el cuadro con las sintaxis de estas comparaciones:

Sintaxis	Significado
<code>==</code>	Igual
<code>!=</code>	Distinto
<code><</code>	Menor
<code>></code>	Mayor
<code><=</code>	Menor o igual
<code>>=</code>	Mayor o igual

Cuadro 2.4: Operadores de comparación en Python

Al igual que los operadores lógicos, estas comparaciones nos entregan un valor de verdad. Veremos el siguiente ejemplo:

```
In [2]: a=300  
        b=200  
        a>b and a!=b
```

```
Out[2]: True
```

```
In [3]: c=504  
        d=738  
        d<=c or c==d
```

```
Out[3]: False
```

Figura 2.4: Ejemplo de uso de operadores lógicos y de comparación

2.6. Ingreso y salida de datos

En la programación, el ingreso y salida de datos resulta de suma importancia. En Python, el ingreso de datos tipo strings se hace con el comando `input()`, se puede definir una variable con un determinado string a ingresar y además se puede añadir un mensaje al momento de pedir el ingreso del string usando comillas dentro del paréntesis, como se ve en el ejemplo:

```
In [*]: s=input('Ingresar un string de ejemplo: ')  
        print(s)
```

Ingresar un string de ejemplo:

(a) Al momento de ejecutar

```
In [4]: s=input('Ingresar un string de ejemplo: ')  
        print(s)
```

Ingresar un string de ejemplo: ABC

ABC

(b) Después de ingresar el string

Figura 2.5: Ejemplo de entrada y salida de datos

Como se puede ver en la figura 2.5 el comando `print()` despliega el valor de la variable ingresada dentro de los paréntesis, en caso de que se quiera desplegar más de un valor de una variable y un texto, se debe ingresar cada variable a desplegar separadas por una coma (,) y encerrarlos en comillas si se pone un texto:

```
In [5]: a=1234  
        b='ejemplo'  
        print('Para este',b,'a=',a)  
  
        Para este ejemplo a= 1234
```

Figura 2.6: Ejemplo de múltiple despliegue de datos

2.7. Comandos de cambio de tipo de dato

Para convertir un tipo de dato a otro, se pueden usar los siguientes comandos de la tabla:

Sintaxis	Tipo de dato a convertir
<code>float()</code>	Número en punto flotante
<code>int()</code>	Número entero
<code>complex()</code>	Número complejo
<code>str()</code>	Cadena de símbolos
<code>list()</code>	Lista

Cuadro 2.5: Comandos para cambiar tipo de datos más comunes

Ahora se mostrará el uso de cada uno de estos comandos, como observación, sólo se puede convertir de un tipo a otro si es posible hacerlo, por ejemplo, no se puede convertir un string a un número tipo float si está formado con letras.

```
In [12]: a,b,c,d,e='1.2343','123456','2+2j',928,'abcde'
a,b,c,d,e=float(a),int(b),complex(c),str(d),list(e)
print(a,type(a),'\n',b,type(b),'\n',c,type(c),d,type(d),'\n',e,type(e))

1.2343 <class 'float'>
123456 <class 'int'>
(2+2j) <class 'complex'> 928 <class 'str'>
['a', 'b', 'c', 'd', 'e'] <class 'list'>
```

Figura 2.7: Diferentes cambios de tipo de datos

Observación 1: El símbolo `\n` representa un salto de línea.

Observación 2: Existen algunas operaciones como: `+` que está definido para otros tipos de tados no numéricos como los strings y las listas, en el caso de los strings, esto representa una concatenación como se ve el el capítulo 4 donde `'a'+'b'='ab'`.

2.8. Estructuras de selección y de repetición

Las estructuras de selección más comunes en Python son **if-elif-else**.

Estos comandos permiten generar bloques en el programa, los que se ejecutan o no dependiendo de una o más condiciones. En el caso de la estructura generada por el comando **if** se ejecutará sólo si se cumple la condición escrita a la derecha del comando. Para el comando **elif**, se ejecutará sólo si no se ejecutó la estructura del **if** (o en alguna anterior estructura generada en otro **elif**) y además se cumpla la condición puesta a su derecha. La estructura generada por el comando **else** se ejecutará siempre que no se hayan ejecutado en las demás estructuras, en este caso no hay que escribir condiciones.

Mientras que las estructuras de repetición en Python son **while** y **for**, la estructura **while** se le agrega una o más condiciones, y mientras esa condición se siga cumpliendo, el bucle seguirá en marcha. El ciclo se detendrá cuando la condición deje de cumplirse. En cambio, la estructura **for** se le debe asignar un rango con el cual se trabajará.

En las estructuras de repetición, el comando **break** se usa para interrumpir la secuencia de iteraciones del ciclo.

2.8.1. Ejemplos de la estructura if-elif-else

Ejemplo 1.1: Crear un programa que en el cual al ingresar “on”, se despliegue “Encendido”, y en caso de que se ingrese cualquier otra cosa, el programa no haga nada.

```
1 A=input('Ingresar algo: ')
2 if A=='on':
3     print('Encendido')
```

Código 2.1: Solución ejemplo 1.1 Estructura if

Ejemplo 1.2: Hacer un programa en el cual si el usuario ingresa un “Hola”, se le responda “Chao”, y si se ingresa cualquier otra cosa, el programa responda con un (?)

```
1 I=input('Ingresar algo: ')
2 if I=='Hola':
3     print('Chao')
4 else:
5     print(' (?)')
```

Código 2.2: Solución ejemplo 1.2 Estructura if-else if

Ejemplo 1.3: Hacer un programa que verifique si un número a ingresar es mayor o igual que 10, en caso de que cumpla con esa condición, se despliegue un mensaje diciendo “Lo cumple”, caso contrario, desplegar “No cumple”, además si el número no cumple la condición, el programa debe indicar al usuario si el número es positivo, negativo o cero.

```
1 Numero=float(input('Ingresar un número: '))
2 if Numero >= 10:
3     print('Lo cumple')
4 else:
5     print('No cumple')
6     if Numero > 0:
7         print('Es positivo')
8     elif Numero < 0:
9         print('Es negativo')
10    else:
11        print('Es cero')
```

Código 2.3: Solución ejemplo 1.3 Estructura if-elif-else if

2.8.2. Ejemplos del ciclo while

Ejemplo 2.1: Crear un programa de tal forma que le pida al usuario ingresar un número menor que 100, y si se le ingresa un número mayor o igual que 100 el programa pida otra vez el número.

```
1 Numero=float(input('Ingresar un número menor que 100: '))
2 while Numero >= 100:
3     Numero=float(input('Vuelva a ingresar: '))
```

Código 2.4: Solución ejemplo 2.1 Estructura while

Ejemplo 2.2: Crear un programa que al ingresar un número entero n , verifique de que sea mayor que 1, en caso que no lo sea, se vuelva a pedir el ingreso del número. Luego, el programa debe desplegar el resultado de:

$$\sum_{i=1}^n \left(i^3 + \frac{1}{i} \right)$$

```
1 n=int(input('Ingrese el entero n: '))
2 while n <= 1:
3     n=int(input('Volver a ingresar n: '))
4 i,suma=1,0
5 while i <= n:
6     suma,i=suma+(i**3+1/i),i+1
7 print('El resultado es:',suma)
```

Código 2.5: Solución ejemplo 2.2 Estructura while

Ejemplo 2.3: Crear un programa que al ingresar un número entero m , lo pida nuevamente hasta que se ingrese un número positivo y par, y cuando se cumpla esto, el programa entregue el valor de su semifactorial. (El semifactorial de un número par es la multiplicación entre él y todos sus antecesores positivos pares)

```
1 m=int(input('Ingrese m: '))
2 while m <= 0 or m%2!=0: #Aquí si m%2!=0 indica que m es impar
3     m=int(input('Vuelva a ingresar m: '))
4 semifactorial=1
5 c=2
6 while c <= m:
7     semifactorial=semifactorial*c
8     c=c+2 #Se le suman 2 para sólo considerar los pares
9 print('El valor de su semifactorial es:',semifactorial)
```

Código 2.6: Solución ejemplo 2.3 Estructura while

2.8.3. Ejemplos del ciclo for

Ejemplo 3.1: Crear un programa en Python, usando el ciclo for que despliegue los primeros 10 números naturales.

```
1 for i in range(1,11):
2     print(i)
```

Código 2.7: Solución ejemplo 3.1 Estructura for

En el código 1.7, la función `range(a,b,c)` donde a, b, c son enteros, en el caso de que $a < b$, y $c > 0$ esta función crea un rango de valores c -espaciados desde a hasta antes de b , esto quiere decir, que `range(1,7,2)` crea un rango donde tiene los valores: $\{1, 3, 5\}$, en otro caso, si $a > b$, y $c < 0$ entonces el rango creado son de valores desde a hasta antes de llegar a b $|c|$ -espaciados, es decir, `range(7,0,-3)` crea un rango donde los valores son: $\{7, 4, 1\}$. Si no se pone el valor c , entonces se considera que es 1, es decir: `range(a,b)=range(a,b,1)`.

Ejemplo 3.2: Realizar un programa que calcule la siguiente sumatoria, considerando que el n a ingresar ya es un número entero mayor que 1:

$$\sum_{k=1}^n \frac{2^k}{5 + k^3}$$

```
1 n=int(input('n= '))
2 suma=0
3 for k in range(1,n+1):
4     suma=suma+(2**k)/(5+k**3)
5 print(suma)
```

Código 2.8: Solución ejemplo 3.2 Estructura for

Ejemplo 3.3: De forma parecida al ejemplo anterior, se debe construir un programa que al ingresar un entero $n > 1$, despliegue el resultado de la siguiente productoria (sin usar la fórmula de la suma de los primeros k naturales):

$$\prod_{k=1}^n \left(\sum_{i=1}^k i \right)$$

Para resolver esto se utiliza un doble ciclo for, uno para el producto y otro para la suma:

```
1 n=int(input('n= '))
2 producto=1
3 for k in range(1,n+1):
4     suma=0
5     for i in range(1,k+1):
6         suma=suma+i
7     producto=producto*suma
8 print(producto)
```

Código 2.9: Solución ejemplo 3.3 Estructura for

2.9. Ejercicios de programación básica

Ejercicio 1.1: Área de un cuadrado Desarrollar un programa que al ingresar las coordenadas de un punto inicial y otro punto final que tiene un lado de un cuadrado, se calcule el área de este cuadrado y despliegue el resultado. El programa debe verificar que el punto (x_1, y_1) sea distinto de (x_2, y_2) , si son iguales se debe volver a pedir el ingreso de datos de x_2 e y_2

Entradas: El programa sólo debe tener 4 entradas, los cuales son los valores x_1, y_1, x_2, y_2 que representan las coordenadas de los puntos.

Salida: La única salida es un número que represente el área del cuadrado.

Ejemplo de entrada: 0 0 2 0

Ejemplo de salida: 4

Ejercicio 1.2: Sumando primos Crear un programa capaz de calcular la suma total de los primeros n números primos, con $100 > n > 1$ y declarar si esta suma resulta ser un número par o impar.

Entradas: El programa sólo debe tener como entrada un número entero n que representa cuántos primos se van a sumar.

Salidas: Deben haber 2 salidas, una que entregue la suma total y un mensaje diciendo que es par o impar.

Ejemplo de entrada: 3

Ejemplo de salida: 10, es par.

Ejercicio 1.3: Potencia de una potencia como sumas Implemente un programa Python que calcule la operación $(x^y)^z$ con sólo sumas, donde x, y y z son enteros positivos.

Entradas: Debe haber 3 entradas, uno para el valor de x , para y , y otro para el valor de z .

Salidas: Como salida sólo se debe entregar el resultado de la operación $(x^y)^z$.

Ejemplo de entradas: 3, 5, 2

Ejemplo de salida: 59049

2.10. Solución sugerida a los ejercicios

```

1  #Inicia el programa solicitando los valores de las coordenadas
2  x1=float(input('x1= '))
3  y1=float(input('y1= '))
4  x2=float(input('x2= '))
5  y2=float(input('y2= '))
6  #Se asegurará que (x1,y1) sea distinto de (x2,y2)
7  while x1==x2 and y1==y2:
8      x2=float(input('Vuelva a ingresar x2= '))
9      y2=float(input('Vuelva a ingresar y2= '))
10 #Se calculará la distancia entre el punto (x1,y1) y (x2,y2)
11 D=((x1-x2)**2+(y1-y2)**2)**(1/2)
12 #Como D representa un lado del cuadrado, el área es su cuadrado
13 Area=D**2
14 print(Area) #Se despliega el resultado

```

Código 2.10: Solución ejercicio 1.1

```

1  #Inicia el programa solicitando la cantidad de primos
2  n=int(input('n= '))
3  #Verifica que n esté en el intervalo (1,100)
4  while n >= 100 or n<=1:
5      n=int(input('Vuelva a ingresar n= '))
6  #Encontrar y sumar los n primeros primos
7  suma=2 #Como n es mayor que 1, el primer primo es 2
8  c,d=1,3 #Contadores auxiliares
9  while c < n:
10     while True:
11         for i in range(2,d):
12             if d%i == 0:
13                 p=0
14                 break
15             elif i==d-1:
16                 p=1
17         if p==0:
18             d=d+1
19         else:
20             break
21     suma=suma+d
22     d,c=d+1,c+1
23 #Desplegar la suma y si es par o impar
24 if suma%2 == 0:
25     print(suma,'Es par')
26 else:
27     print(suma,'Es impar')

```

Código 2.11: Solución ejercicio 1.2


```
1  # Solicitar los enteros X e Y
2  x=int(input('Ingresar x= '))
3  # Comprobar que es positivo
4  while x <= 0:
5      x=int(input('Vuelva a ingresar x= '))
6  y=int(input('Ingresar y= '))
7  while y <= 0:
8      y=int(input('Vuelva a ingresar y= '))
9  z=int(input('Ingresar z= '))
10 while z <= 0:
11     z=int(input('Vuelva a ingresar z= '))
12 # calcular potencia x elevado a y
13 if y == 1:
14     Potencia=x
15 else:
16     i,aux=1,x
17     while i < y:
18         Potencia=0
19         for k in range(0,x):
20             Potencia=Potencia+aux
21         aux,i=Potencia,i+1
22 # Ahora se calculará la potencia de la potencia
23 if z == 1:
24     Potencia2=Potencia
25 else:
26     j,aux2=1,Potencia
27     while j < z:
28         Potencia2=0
29         for k in range(0,Potencia):
30             Potencia2=Potencia2+aux2
31         aux2,j=Potencia2,j+1
32 # Desplegamos el resultado
33 print(Potencia2)
```

Código 2.12: Solución ejercicio 1.3

CAPÍTULO

3

FUNCIONES

Las funciones en python son muy útiles para crear subprogramas que deben realizar una misma acción en varias ocasiones, debido a que estas cumplen el rol de realizar esa acción sin la necesidad de programar todo cada vez que se necesite.

En este capítulo se verá en forma simple, el cómo crear y utilizar las funciones.

3.1. Crear y llamar a una función

Para crear una nueva función, la sintaxis es muy sencilla, primero se debe escribir el comando `def` dejar un espacio, escribir el nombre de la función e inmediatamente después, colocar entre paréntesis todos los parámetros de la función, para después colocar los dos puntos y empezar a crear la estructura de la función.

Los parámetros son variables que reciben valores desde la llamada de la función y que permiten la ejecución de la tarea para la cual la función fue construida.

Para llamar a una función, es decir, utilizarla en algún momento del programa, sólo se tiene que escribir el nombre de la función junto con los parámetros a usar entre paréntesis.

Sí se admite que una función no tenga parámetros, aun así, se deben necesariamente colocar los paréntesis.

Ejemplo 1: Desplegar un saludo

```
1 def hola():  #Se crea la función
2     print('Hola')
3 hola()      #Se llama a la función
```

Código 3.1: Ejemplo de función sin parámetros

Esta función al ser llamada, sólo despliega la cadena “Hola”.

Ahora se mostrará un ejemplo de una función que dependa de un parámetro:

Ejemplo 2: Factorial

```
1 def factorial(n):  #Se crea la función
2     if n==0 or n==1:
3         fn=1
4         print('n!=', fn)
5     else:
6         fn=1
7         for i in range(0,n):
8             fn=fn*(n-i)
9         print('n!=', fn)
10 n=int(input('Ingresar entero positivo: '))
11 factorial(n)      #Se llama a la función
```

Código 3.2: Ejemplo de función factorial

Esta función al ser llamada despliega el factorial de un número entero no negativo n ingresado.

3.2. Funciones con retorno

Al momento de crear una función existe la opción de que esta devuelva su resultado al programa u otra función que la invoca o llama. Para esto se utiliza el comando `return`, el cual después de un espacio debe incluir todas las variables o valores devueltos por la función.

A continuación se ve un ejemplo de una función que retorna el valor de la siguiente sumatoria:

$$\sum_{i=0}^a b^i$$

Donde el programa despliega el caso particular donde $a = 4$ y $b = 5$

```
1 def sumadepotencia(a,b):  
2     s=0  
3     for i in range(0,a+1):  
4         s=s+b**i  
5     return s  
6 c=sumadepotencia(4,5)  
7 print(c) #El resultado que despliega es 781
```

Código 3.3: Ejemplo de función con retorno

3.3. Ejercicios usando funciones

Ejercicio 2.1: Sumar números de Fibonacci Crear un programa capaz de sumar dos elementos de la serie de Fibonacci con sólo ingresar sus posiciones en la serie, y que despliegue el resultado.

Recordar que la sucesión de Fibonacci es de la forma:

$$0, 1, 1, 2, 3, 5, 8, 13 \dots$$

Donde el elemento de posición $n \geq 3$ es el resultado de la suma entre el elemento $(n - 1)$ y el $n - 2$

Entradas: El programa sólo debe tener 2 entradas, las cuales son números enteros positivos que indican la posición de los números de Fibonacci a sumar.

Salidas: Debe haber 1 salida, la cual es el resultado de esa suma.

Ejemplo de entrada: 5, 7

Ejemplo de salida: 11

Ejercicio 2.2: Ecuación de la recta Crear un programa que encuentre la ecuación de la recta al ingresar 2 puntos (x_1, y_1) , (x_2, y_2)

Entradas: Deben haber 4 entradas, correspondientes a las variables x_1, y_1, x_2 e y_2 (en ese orden).

Salidas: Debe haber 1 salida, la ecuación de la recta.

Ejemplo de entrada: 0, 1, 2, 5

Ejemplo de salida: $y = 2.0x - 1.0$

Ejercicio 2.3: Serie de Taylor Construir un programa tal que despliegue una aproximación del valor de $\sin(x)$ y de $\cos(x)$ donde x es un valor real a ingresar, usando sumas parciales de las series de Taylor:

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}, \quad \cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$$

Entradas: 2 entradas, una para la cantidad de sumas parciales $p > 1$ y otra para el valor x

Salidas: El resultado de la aproximación para $\sin(x)$ y el para $\cos(x)$

Ejemplo de entrada: 100, 3.14

Ejemplo de salida: seno: 0,001592652916... coseno: -0,9999987317...

3.4. Soluciones propuestas de los ejercicios

```
1  #Es recomendable que siempre se empiece colocando las funciones a
2  #usar, para que el programa sea más fácil de entender.
3
4  #Función para leer y validar que el entero positivo sea
5  #ingresado de manera correcta
6
7  def LyV(minimo,msg):
8      valor=int(input(msg))
9      while valor < minimo:
10         valor=int(input('Error',msg))
11     return valor
12
13 #Función para tener el número de Fibonacci según su posición
14
15 def fibonacci(n):
16     q=1
17     if n == 1:
18         return 0
19     elif n == 2 or n == 3:
20     else:
21         p=1
22         while n > 2:
23             r=p+q
24             p=q
25             q=r
26             n=n-1
27     return r
28
29 #Programa Principal
30
31 a=LyV(1,'Vuelva a ingresar el primer número: ')
32 b=LyV(1,'Vuelva a ingresar el segundo número: ')
33 a,b=fibonacci(a),fibonacci(b)
34 print(a+b)
35 #Fin del programa
```

Código 3.4: Solución ejercicio 2.1

```
1  #Una función para leer las variables y verificar
2  #Aquí verifica que (x1,y1) y (x2,y2) son distintos
3  def LyV():
4      x1=float(input('Ingresar x1: '))
5      y1=float(input('Ingresar y1: '))
6      x2=float(input('Ingresar x2: '))
7      y2=float(input('Ingresar y2: '))
8      while x1 == x2 and y1 == y2:
9          print('Ingrese de nuevo: ')
10         x1=float(input('Ingresar x1: '))
11         y1=float(input('Ingresar y1: '))
12         x2=float(input('Ingresar x2: '))
13         y2=float(input('Ingresar y2: '))
14     return x1,y1,x2,y2
15
16 #Luego se crea la función que encuentra la recta
17 def recta(x1,y1,x2,y2):
18     if x1 == x2:
19         L='x = '+str(x1)
20     else:
21         m=(y1-y2)/(x1-x2)
22         c=y1-x1*m
23         if c > 0:
24             L='y = '+str(m)+'x + '+str(c)
25         elif c < 0:
26             L='y = '+str(m)+'x - '+str(abs(c))
27         else:
28             L='y = '+str(m)+'x'
29     return L
30
31 #Finalmente se hace el programa principal
32 x1,y1,x2,y2=LyV()
33 print(recta(x1,y1,x2,y2))
34 #Fin del programa
```

Código 3.5: Solución ejercicio 2.2

```
1  #Función para ingresar correctamente p
2
3  def LyV(minimo,msg):
4      valor=int(input(msg))
5      while valor < minimo:
6          valor=int(input('Ingrese de nuevo: '))
7      return valor
8  #Función para calcular el factorial de un número
9
10 def factorial(n):
11     if n == 0 or n == 1:
12         fn=1
13         return fn
14     else:
15         fn=1
16         for i in range(0,n):
17             fn=fn*(n-i)
18         return fn
19 #Aproximación del seno
20
21 def seno(x,p):
22     s=0
23     for n in range(0,p+1):
24         s=s+((-1)**n/factorial(2*n+1))*x**(2*n+1)
25     return s
26 #Aproximación del coseno
27
28 def coseno(x,p):
29     s=0
30     for n in range(0,p+1):
31         s=s+((-1)**n/factorial(2*n))*x**(2*n)
32     return s
33 #Programa principal
34
35 p=LyV(1, 'Ingresar p: ')
36 x=float(input('Ingresar valor de x: '))
37 print('seno:', seno(x,p), 'coseno:', cos(x,p))
38 #Fin del programa.
```

Código 3.6: Solución ejercicio 2.3

CAPÍTULO

4

STRINGS

Se sabe que a las cadenas de símbolos se les llama strings, en este capítulo se verán varias formas en las que se puede trabajar con ellas, ejemplos de cómo usar la indexación de una, los métodos relacionados a estas, y unos ejercicios donde se mezclará lo aprendido en este capítulo y los dos anteriores.

4.1. Operaciones y funciones básicas con strings

En Python también existen operaciones con cadenas de símbolos, como las mostradas a continuación:

Concatenación: Para unir dos strings en uno solo, se usa el operador “+”

Es decir, si se tiene lo siguiente: ‘Uno’ + ‘Dos’, el resultado será: ‘UnoDos’

Multiplicar cadena por un número: Al usar el operador “*” como multiplicación entre un número entero positivo y una cadena, el resultado será otra cadena donde se repite la cadena de la multiplicación, la misma cantidad de veces que indica el número por el cual se multiplicó. Como ejemplo, si se realiza la siguiente operación: 3 * ‘Hoy’ o ‘Hoy’ * 3, el resultado es: HoyHoyHoy

Largo de una cadena: Cuando se habla del largo de una cadena, hacemos referencia a la cantidad de símbolos que tiene, para calcularlo se usa el comando `len()` donde el string a calcular su cantidad de símbolos se coloca dentro de los paréntesis, como ejemplo, si hacemos: `len(‘ABCD12$’)` el resultado sería 7

4.2. Indexación y cómo recorrer un string

Python tiene 2 formas de enumerar la posición de cada cadena, uno de ellos es con un índice partiendo del 0 que se va enumerando de izquierda a derecha y otra forma es el índice inverso, partiendo del -1 que es el último símbolo de la cadena y disminuyendo de derecha a izquierda, como se ve en la figura:

Caracteres :	P	y	t	h	o	n
Índice :	0	1	2	3	4	5
Índice inverso :	-6	-5	-4	-3	-2	-1

Figura 4.1: Índices de los elementos de la cadena ‘Python’

Para acceder a un elemento en específico de una cadena, se guarda la cadena en una variable, y luego se llama a la variable y a su derecha colocar los corchetes `[]` donde dentro de ellos se pone el número del índice o índice inverso del elemento al cual se quiere acceder.

Ejemplo 1.1: Recorrer una cadena de izquierda a derecha

Se debe hacer un programa que despliegue cada elemento de una cadena de símbolos a ingresar, de izquierda a derecha.

```
1 a=input('Ingresar cadena: ')
2 for i in range(0,len(a)):
3     print(a[i])
```

Código 4.1: Solución de recorrer una cadena de izquierda a derecha

Ejemplo 1.2: Recorrer una cadena de derecha a izquierda

Hacer un programa análogo al anterior pero con el recorrido inverso.

```
1 a=input('Ingresar cadena: ')
2 for i in range(-1,-len(a)-1,-1):
3     print(a[i])
```

Código 4.2: Solución de recorrer una cadena de derecha a izquierda

4.2.1. Sub-cadena

Además de poder acceder a un elemento específico de una cadena usando los índices, se puede acceder a una sub-cadena o segmento de una cadena al colocar un intervalo como índice. Estos intervalos son de la forma: `[a:b:c]` donde `a` es el número del índice por donde empieza el intervalo, `b` es el índice donde termina y `c` indica cómo se irán tomando los elementos. si sólo se coloca `[a:b]` entonces se asume el `c` como 1.

Ejemplo 2.1: El siguiente programa despliega la cadena 'aea' que son elementos del string 'cadena1234' con índices 1, 3 y 5 respectivamente. No considera el elemento con índice 7 porque el último elemento del intervalo no se considera.

```
1 a='Cadena1234'
2 print(a[1:7:2])
```

Código 4.3: Ejemplo 1 de sub-cadena

Ejemplo 2.2: Usando la misma cadena del ejemplo anterior se desplegará '4321' dos veces, uno sin usar el comando `len` y otro donde sí se usa:

```
1 a='cadena1234'
2 print(a[-1:-5:-1])
3 print(a[len(a)-1:5:-1])
```

Código 4.4: Ejemplo 2 de sub-cadena

4.3. Métodos más comunes con strings

Un método es un tipo de función que tienen los objetos. Las cadenas en Python, por ejemplo, representan un tipo de objeto y existe una gran variedad de estos para ellas.

Para ejecutar un método se debe escribir el objeto al cual se le va a ejecutar, y en su derecha escribir un punto (.) y el nombre del método a usar.

Ejemplo: cadena.metodo()

En las siguientes tablas se mostrarán algunos de los nombres de estos métodos y lo que realizan:

Métodos de análisis:

Nombre	Uso	Nombre	Uso
<code>count()</code>	Retorna el número de veces que se repite una cadena especificada dentro de los paréntesis.	<code>isnumeric()</code>	Determina si todos los caracteres de la cadena ingresada son números, y entrega el valor de verdad.
<code>find()</code> <code>index()</code>	Ambos devuelven el índice en donde empieza la cadena especificada dentro de los paréntesis, en caso de que no se encuentre el índice, la de abajo da un error y la de arriba entrega un <code>-1</code>	<code>rfind()</code> <code>rindex()</code>	Hacen lo mismo que los comandos de la izquierda, pero ahora considerando los índices de derecha a izquierda.

Cuadro 4.1: Algunos métodos de análisis para strings

Métodos de transformación:

Nombre	Uso
<code>capitalize()</code>	Retorna la cadena con su primera letra en mayúscula.
<code>center()</code>	Alinea una cadena en el centro, tomando un argumento en los paréntesis que indica la cantidad de caracteres respecto de la cual se producirá la alineación.
<code>ljust()</code>	Misma función de alineación, pero ahora hacia la izquierda.
<code>rjust()</code>	En este caso la alineación es hacia la derecha.
<code>lower()</code>	Retorna una copia de la cadena con todas sus letras en minúsculas.
<code>upper()</code>	De manera parecida a la anterior, pero ahora en mayúsculas.
<code>strip()</code>	Remueve los espacios en blanco que preceden y suceden a la cadena.
<code>replace()</code>	Reemplaza una subcadena por otra. Se deben ingresar ambas cadenas dentro de los paréntesis en el orden del cambio, y separándolas con una coma. La cantidad máxima de cambios (en caso de que halla más de una subcadena igual a la que se quiera cambiar), se coloca después de las dos cadenas.

Cuadro 4.2: Algunos métodos de transformación para strings

4.4. Ejercicios usando strings

Ejercicio 3.1: Mensaje secreto Un famoso programador quiere regalar como premio un Bitcon a la primera persona que logre traducir el siguiente mensaje de forma exitosa:

n1oct18 nu od4n4g 74h ,ohc3h n318

Alguien se dió cuenta que el mensaje se podría decifrar realizando los siguientes cambios: 0 = o, 1 = i, 8 = B, 3 = e, 7 = z, 4 = a y leerlo de derecha a izquierda, pero ¿Qué programa usó para entregar su respuesta?, recree ese programa en Python que pueda decifrar ese mensaje y cualquier otro mensaje con ese tipo de encriptación.

Entrada y salida: Sólo se debe ingresar el mensaje a decifrar y desplegar el resultado

Ejemplo de entrada: *n1oct18 nu od4n4g 74h ,ohc3h n318*

Ejemplo de salida: Bien hecho, te haz ganado un Bitcoin.

Ejercicio 3.2: Verificación de clave Un banco ha decidido que las claves de sus clientes tuvieran una alta complejidad para así disminuir los hackeos de cuenta. Esta clave debe tener al menos 25 caracteres, y ser una combinación de minúsculas, mayúsculas y números, además de no tener espacios. Crea un programa que verifique si una clave ingresada satisface estas condiciones, en caso de que lo cumpla o no, que se despliegue el mensaje diciéndolo.

Entradas: Sólo 1 la cual es la clave por verificar

Salidas: Un mensaje diciendo si la clave cumple o no cumple las condiciones.

Ejemplo de entrada: 11111ContrAse232slp230001

Ejemplo de salida: Clave válida.

Ejercicio 3.3: De binario o quinario a decimal Crear un programa que le permita al usuario elegir entre traducir un número binario o quinario (en base 5) a un número decimal, y que además le permita elegir si quiere traducir otro o no.

Entradas: Principalmente las respuestas del usuario a las sugerencias del programa y sus números a traducir.

Salidas: Todos los números que el usuario coloque y quiera pasarlos a decimal.

4.5. Soluciones propuestas de los ejercicios

```

1  # En este primer problema no es necesario usar funciones
2
3  F=input('Ingresar código: ')
4  F=F.replace('0','o')
5  F=F.replace('4','a')
6  F=F.replace('1','i')
7  F=F.replace('3','e')
8  F=F.replace('5','s')
9  F=F.replace('8','B')
10 F=F.replace('7','z')
11 #Aquí se creará el mensaje de derecha a izquierda
12
13 k=len(F)-1
14 B=''
15 for i in range(0,len(F)):
16     B=B+''+F[k]
17     k=k-1
18 print(B) #Se despliega el mensaje real

```

Código 4.5: Solución ejercicio 3.1

```

1  Con=input('Ingresar c: ')
2  #Se comprobará que tenga al menos 25 caracteres
3  if len(Con) < 25:
4      print('Clave inválida')
5  else:
6      # Se probará si tiene números
7      c=0
8      for i in range(0,len(Con)):
9          V=Con[i].isnumeric()
10         if V == False:
11             c=c+1
12     if c == len(Con) or c == 0:
13         print('Clave inválida')
14     else:
15         # Ahora se verá que tenga minúsculas, mayúsculas y no ←
16         tenga espacios
17         if Con.upper() != Con and Con.lower() != Con and Con.←
18             replace(' ','') == Con:
19                 print('Clave válida')
20         else:
21             print('Clave inválida')

```

Código 4.6: Solución ejercicio 3.2

```

1  # Función para convertir un número en base b a decimal
2
3  def traducir(b,cadena):
4      Decimal=0
5      potencia=0
6      for i in range(-1,-len(cadena)-1,-1):
7          Decimal=Decimal+int(cadena[i])*b**potencia
8          potencia=potencia+1
9      return Decimal
10
11 #Programa Principal
12
13 while True:
14     print('Elija el tipo de traducción \n 1) Binario a decimal \n ←
15         2) De quinario a decimal ')
16     # Recordar que \n es el símbolo del salto de línea
17     n=input('Ingrese 1 o 2:')
18     while n != '1' and n != '2':
19         n=input('Ingresar un opción válida: ')
20     if n == '1':
21         B=input('Ingrese número binario: ')
22         while True:
23             c=0
24             for i in range(0,len(B)):
25                 if B[i] != '0' and B[i] != '1':
26                     c=1
27             if c== 0:
28                 print('En decimal es:',traducir(2,B))
29                 break
30             else:
31                 B=input('Ingresar número válido: ')
32     else:
33         Q=input('Ingrese quinario: ')
34         while True:
35             c=0
36             S='012345'
37             for i in range(0,len(Q)):
38                 if Q[i] not in S:
39                     c=1
40             if c == 0:
41                 print('En decimal es:',traducir(5,Q))
42                 break
43             else:
44                 Q=input('Ingresar número válido: ')
45     print('Elija si quiere traducir algo más \n 1) Si \n 2) No')
46     s=input('Ingrese 1 o 2:')
47     if s == '2':
48         break

```

Código 4.7: Solución ejercicio 3.3

CAPÍTULO

5

LISTAS

Una lista es otro tipo de dato, que a la vez es una estructura que puede almacenar una gran variedad de elementos de iguales o distintos tipos de datos. Debido a esto resultan de mucha utilidad en la creación de programas.

En este capítulo se mostrará la indexación de ellas, métodos, y los respectivos ejercicios donde se utilizan.

5.1. Crear listas

Para crear una lista en Python, se puede hacer de forma manual o bien con el comando `list`. Para hacer una lista en forma manual, se usan los corchetes, y en su interior se colocan los datos separados por una coma, como por ejemplo, crear la lista: `L=[1, 'A', 1.03, 2+3j]`, en el caso del comando `list` se debe acompañar de unos paréntesis a la derecha, y dentro de ellos escribir una cadena con los caracteres que se quieren poner como elementos de la lista, si se usa de esta forma, se debe tener cuidado ya que cada símbolo lo guardará como un elemento distinto.

Otra manera de crear una lista es la de elemento por elemento, la cual se puede hacer de diferentes formas, que se verán más adelante.

5.1.1. Lista de listas

Como el nombre dice, en Python se pueden crear listas que tiene por elementos otras listas y así sucesivamente. Para esto sólo basta colocar los corchetes de la sub-lista y sus elementos separados por comas de la misma forma que se hizo con la lista más grande, como ejemplo:

```
ListaL=[[1,2,3],[3,2],[[1,5],[2,4]]]
```

5.2. Operadores básicos en listas

Al igual que los datos numérico y los strings, las listas en Python tienen los operadores `+` y `*`, los cuales están definidos de la siguiente forma:

Suma entre listas: Los elementos de la lista sumada, pasarán a ser elementos de la primera lista, ordenadas de la misma manera y después del último elemento que tenía originalmente la primera lista.

Ejemplo:

```
1 # La operación suma:
2
3 [1,2,'a']+['a',3,34,'b']
4
5 # Entrega como resultado: [1, 2, 'a', 'a', 3, 34, 'b']
```

Código 5.1: Suma de listas

Lista por un natural: Si se multiplica una lista por un número natural, el resultado entrega una lista con los mismos elementos, pero repetidos las veces que indica el número por el cual se multiplicó, en el mismo orden.

Ejemplo:

```
1 # La operación multiplicación:
2
3 3*['a', 3]
4
5 # Entrega como resultado:['a', 3, 'a', 3, 'a', 3]
```

Código 5.2: Natural por una lista

Cabe mencionar, que al igual que las cadenas, el comando `len()` se puede usar en las listas, y entrega la cantidad de elementos que tiene la lista.

5.3. Indexación en listas

La indexación de las listas es de cierta manera análoga al caso de los strings, con la diferencia que en estas se recorre por elementos y asu vez, si el elemento es una cadena o una lista, se puede subrecorrer.

Ejemplo 1: Si tenemos una lista `L=[1, 2, 3]` para obtener el valor entero 3, se puede hacer de las dos maneras siguientes:

```
1 # Primera forma, enumeración de índices de izquierda a derecha
2 L=[1, 2, 3]
3 a=L[2]
4
5 # Segunda forma, enumeración de índices de derecha a izquierda
6 L=[1, 2, 3]
7 a=L[-1]
```

Código 5.3: Ejemplo 1 Índices de elementos en listas

Ejemplo 2: Si se quiere obtener un elemento de una lista que está dentro de otra lista, se debe colocar primero la posición en la que está esa lista en la lista más grande y después la posición del elemento a buscar en la lista pequeña, como se ve en el ejemplo:

```
1 # Si se quiere obtener el elemento '1' de:
2 LL=[1, 2, ['1', 1, 'c'], 'a']
3 # La lista más chica tiene el índice 2 como elemento
4 # El '1' tiene como índice 0 en la sub-lista
5
6 ElementoPedido=LL[2][0]
```

Código 5.4: Ejemplo 2 Índices de elementos en listas

5.4. Funciones básicas para listas numéricas

Si se tiene una lista, la cual todos sus elementos son números, entonces las siguientes funciones en Python pueden resultar muy útiles:

Función min: La función `min()` lo que hace es calcular el mínimo valor que está en la lista colocada dentro de los paréntesis.

```
1 L=[1.5,2.7,9,-3.4]
2 Minimo=min(L)
3 print(Minimo)
4 #Despliega -3.4
```

Código 5.5: Ejemplo de la función min

Función max: Parecida a la función `min()`, la función `max()` calcula el máximo valor que está en la lista colocada dentro de los paréntesis.

```
1 L=[1.5,2.7,9,-3.4]
2 Maximo=max(L)
3 print(Maximo)
4 #Despliega 9
```

Código 5.6: Ejemplo de la función max

Función sum: Calcula la suma total de todos los elementos de la lista ingresada.

```
1 L=L=[1.5,-3.4,0.9,4]
2 Suma=sum(L)
3 print(Suma)
4 #Despliega 3.0
```

Código 5.7: Ejemplo de la función sum

5.5. Métodos más comunes con listas

Así como existen diversos métodos para los strings, los hay para las listas, la notación para usarlos es análoga al caso de los strings, con la diferencia que al usar un método en una lista sólo para modificarla, no se puede asignar el valor del dato modificado a otra variable.

Nombre	Uso
<code>append()</code>	Añade un elemento puesto dentro de los paréntesis al final de la lista (string, o numérico).
<code>clear()</code>	Elimina todos los elementos de la lista.
<code>extend()</code>	Añade todos los elementos de una lista ingresada dentro de los paréntesis a la otra.
<code>count()</code>	Cuenta todas las veces que un elemento específico aparece en la lista.
<code>index()</code>	Entrega el número del índice del primer elemento de la lista igual al ingresado, si no hay ninguno, marca error
<code>insert()</code>	Se ingresan 2 variables, primero el índice donde se quiere guardar un nuevo dato para la lista, y el segundo, es el nuevo dato. Si se ingresa un índice fuera de rango, el dato lo añade al final.
<code>pop()</code>	Extrae el elemento solicitado de la lista y después lo elimina de ella.
<code>remove()</code>	Borra el primer elemento de la lista que coincida con el puesto dentro de los paréntesis.
<code>reverse()</code>	Ordena los elementos de la lista de forma inversa. (“Da la vuelta a la lista”)
<code>sort()</code>	Ordena los elementos de la lista de menor a mayor los datos numéricos (int y float), y alfabéticamente los strings

Cuadro 5.1: Métodos para listas

5.5.1. Método split

Así como existe el comando `list` para convertir una cadena en una lista, también existen el método `split` para transformar una cadena en una lista de cadenas, a pesar de que se puede pensar que hacen lo mismo, este método resulta ser más útil debido a los siguientes puntos:

- **Separadores:** El método separa una cadena en varios elementos por separadores. Por defecto: `cadena.split()` separa los símbolos de la cadena por espacios en blanco y saltos de línea, pero se puede personalizar el separador colocándolo como argumento.
- **Cantidad de divisiones:** Un segundo argumento en el método (después del separador) indica cuál es el máximo de divisiones posibles se harán para crear la lista.

Ejemplo 1: Se creará una lista donde cada elemento es una palabra de la frase “Esta es una cadena de ejemplo”, como se quiere separar por espacios, se puede usar el comando por defecto.

```
1 A='Esta es una cadena de ejemplo'
2 L=A.split()
3 print(L)
4 # Se despliega: ['Esta', 'es', 'una', 'cadena', 'de', 'ejemplo']
```

Código 5.8: Ejemplo 1 método split

Ejemplo 2: Si ahora, se tiene la cadena: `'a-b-c-d-e'`, si se quiere hacer una lista de las 3 primeras letras separadas y lo que sobra todo junto como otro elemento, entonces se debe indicar como argumentos el string de separación y la cantidad de separaciones:

```
1 A='a-b-c-d-e'
2 L=A.split('-',3)
3 print(L)
4 # Se despliega: ['a', 'b', 'c', 'd-e']
```

Código 5.9: Ejemplo 2 método split

5.5.2. Método join

El método `join` es lo contrario al `split`, ya que aquí se define primero el separador como string, y dentro de los paréntesis se le ingresa la lista con elementos a unir en una sola cadena.

En general, se puede ver que:

```
separador.join(cadena.split(separador))
```

Regresa la misma cadena.

Ejemplo 1: Se volverá a unir el string que en el ejemplo 1 de la subsección 5.5.1 se convirtió en una lista.

```
1 L=['Esta', 'es', 'una', 'cadena', 'de', 'ejemplo']
2 C=' '.join(L)
3 print(C)
4 # Se despliega: 'Esta es una cadena de ejemplo'
```

Código 5.10: Ejemplo 1 método join

Ejemplo 2: De la misma manera se volverá a construir la cadena del anterior ejemplo 2 con la lista obtenida.

```
1 L=['a', 'b', 'c', 'd-e']
2 C='-'.join(L)
3 print(C)
4 # Se despliega: 'a-b-c-d-e'
```

Código 5.11: Ejemplo 2 método join

5.6. Ejercicios

Ejercicio 4.1: Conjunto potencia Crear un programa en Python capaz de generar el conjunto potencia (o de partes) de un conjunto de 3 elementos.

Entrada: Una cadena con 3 elementos separados por espacios (se asumirá que el ingreso sea correcto).

Salida: El conjunto de partes del conjunto con los datos ingresados.

Ejemplo de entrada: 1 2 3

Ejemplo de salida:

`{'', {'1'}, {'2'}, {'3'}, {'1', '2'}, {'1', '3'}, {'2', '3'}, {'1', '2', '3'}}`

Ejercicio 4.2: Medidas de dispersión Crear un programa en Python el cual, cuando un usuario ingrese un conjunto de datos numéricos positivos (separados por comas), este calcule el rango y la varianza de esos datos, y los entregue en una lista en ese orden.

Entrada: Una entrada donde se ingresen todos los datos separados por comas.

Salida: Una cadena de dos elementos, con el rango y la varianza.

Ejemplo de entrada: 0.5,5,2.5,10,6.5

Ejemplo de salida: [9.5, 10.74]

Ejercicio 4.3: Notas y estudiantes Un profesor quiere ordenar todas las notas finales con respecto a cada estudiante en una sola lista, para esto le pide a un conocido que le creara un programa en Python donde él pueda ingresar cada nombre de sus estudiantes con sus respectivas 2 notas, y que el programa le despliegue una lista donde cada elemento es una lista que tiene el nombre y la nota promedio de cada estudiante y que estén ordenadas por el valor de la nota (escalada del 1.0 al 7.0) de mayor a menor, y que además se despliegue una lista con los nombres de quienes reprobaron. (Hay que tener en cuenta que la lista de alumnos es de 30)

Entrada: 30 entradas para cada nombre y 60 entradas de cada nota.

Salida: Las listas pedidas por el profesor.

5.7. Soluciones

```
1 C=input('Ingresar elementos del conjunto: ')
2
3 # Crear lista de elementos
4 L=C.split()
5
6 # Crear lista de subconjuntos
7 K=['']
8 for i in range(0,len(L)):
9     K.append([L[i]])
10 for l in range(0,len(L)-1):
11     for s in range(l+1,len(L)):
12         K.append([L[l],L[s]])
13 K.append(L)
14 K=str(K).replace('[','{').replace('}','}')
15 print(K) #Despliega lo pedido
```

Código 5.12: Solución ejercicio 4.1

```
1 n=input('Ingresar datos: ')
2
3 # Verificación de que sean datos numéricos positivos
4 while True:
5     L=n.split(',')
6     c=0
7     for i in range(0,len(L)):
8         if L[i].replace('.', '', 1).isnumeric() == True:
9             c=c+1
10    if c == len(L):
11        Ln=[]
12        for i in range(0,len(L)):
13            Ln.append(float(L[i]))
14        break
15    else:
16        n=input('Error, Ingresar datos de nuevo: ')
17 # Calcular el rango
18 R=max(Ln)-min(Ln)
19
20 # Calcular la varianza
21 N=len(Ln)
22 promedio=sum(Ln)/N
23 Sdif=0
24 for i in range(0,N):
25     Sdif=Sdif+(Ln[i]-promedio)**2
26 varianza=Sdif/N
27
28 # Desplegar la lista
29 print([R,varianza])
```

Código 5.13: Solución ejercicio 4.2

```

1  # Función para leer y validar las notas
2  def LyV(mini,maxi,msg):
3      valor=float(input(msg))
4      while valor < mini or valor > maxi:
5          valor=float(input('Ingreso de nuevo: '))
6      return valor
7
8  # Programa principal
9
10 ListaT=[]
11 ListaTF=[]
12 ListaRe=[]
13 #Se crea el ciclo de repetición para el ingreso de los 30 nombres
14 for i in range(0,30):
15     A1=input('Ingresar nombre: ')
16     N1=LyV(1.0,7.0,'Ingresar su primera nota: ')
17     N2=LyV(1.0,7.0,'Ingresar su segunda nota: ')
18     NP=(N1+N2)/2
19     # Se Crea cada lista elemento
20     ListaE=[NP,A1]
21     # Si reprueba, se agrega a la lista de reprobados
22     if NP < 4.0:
23         ListaRe.append(A1)
24     # Se agrega cada lista elemento a la lista total
25     ListaT.append(ListaE)
26     # Se ordena de mayor nota a menor nota
27     ListaT.sort()
28     ListaT.reverse()
29
30 # Se ordena para que aparezca el nombre primero, luego la nota
31
32 for j in range(0,len(ListaT)):
33     ListaT[j].reverse()
34     ListaTF.append(ListaT[j])
35 # Desplegamos las listas pedidas
36 print(ListaTF)
37 print('Reprobados:',ListaRe)

```

Código 5.14: Solución ejercicio 4.3

CAPÍTULO

6

ARREGLOS

Los arreglos en Python son los equivalentes de las matrices y vectores de las matemáticas. Una gran motivación para usar arreglos, es que hay mucha teoría detrás de ellos que puede ser usada en el diseño de algoritmos para resolver problemas verdaderamente interesantes.

Los arreglos tienen una gran utilidad (superior a las listas) ya que es una estructura de datos que sirve para almacenar grandes secuencias de números (generalmente de tipo float)

Sin embargo, tienen restricciones que se deben tomar muy en cuenta antes de trabajar con ellos:

- Todos los elementos del arreglo deben tener el mismo tipo de dato
- En general, el tamaño del arreglo es fijo (no va cambiando su tamaño como las listas a través de métodos)
- Para trabajar con arreglos en Python, es necesario importar el paquete Numpy.

6.1. Introducción al paquete Numpy

Un paquete en Python, en palabras simples es como un conjunto de varios comandos y funciones extras que se pueden usar, para usarlos se deben instalar primero en el dispositivo con el cual se está trabajando y después importar el paquete cuando se quiera usar en algún programa. En este caso Numpy (Numerical Python) es un paquete que tiene una gran variedad de comandos y funciones relacionadas al área del cálculo numérico y los arreglos. Para trabajar con Numpy, se importará el paquete siempre al principio de cada programa, para tener un orden y no haya problemas con el programa. Se puede importar todo el paquete o bien sólo algunas funciones que se van a usar, para esto, generalmente se usan las siguientes formas:

```
import numpy as nombre
from numpy import función
```

Donde (nombre) es el nombre que se le quiere colocar al paquete (usualmente se le pone “np” por comodidad) y (función) es el nombre de la función de Numpy que se quiera usar.

Ejemplo: Una de las funciones de Numpy es `median` lo que hace es calcular la mediana de un conjunto de números. Para llamar a esta función y calcular la mediana de un conjunto `x` se puede hacer lo siguiente:

```
1 import numpy as np
2 x=[1,2,3,1]
3 s=np.median(x)
4 print(s)
5 #Se despliega 1.5
```

Código 6.1: Ejemplo 1 de llamar una función en Numpy

Para llamar sólo a la función y evitar escribir el (`np .`) cada vez que se usa la función, es la siguiente:

```
1 from numpy import median
2 x=[1,2,3,1]
3 s=median(x)
4 print(s)
5 #Se despliega 1.5
```

Código 6.2: Ejemplo 2 de llamar una función en Numpy

6.2. Creación manual de arreglos

Los arreglos en Python se pueden crear de forma manual con la función `array`, o también, de manera predeterminada con alguna función y luego modificarla si así se quiere. Existen distintas formas de crear un arreglo predeterminado, pero esto se dejará para la sección 6.4

6.2.1. Vectores con el comando `array`

Los arreglos unidimensionales en programación son el equivalente a un vector fila, así la construcción de un vector de forma manual, se realiza con la misma sintaxis que se construía una lista (considerando la restricción de que todos los datos deben ser de un mismo tipo), pero dentro de los paréntesis de la función `array`.

Ejemplo: Se mostrará a continuación, la construcción manual de 4 vectores con los tipos de dato: `int`, `string`, `float`, y `complex`.

```
1 # Se importa el paquete
2 import numpy as np
3 # Se crearán los vectores
4 V1=np.array([1,2,3,4]) # Vector de enteros
5 V2=np.array(['a','b','c']) # Vector de strings
6 V3=np.array([1.2,3.045]) # Vector de números punto flotante
7 V4=np.array([2+3j,4-2j]) # Vector de números complejos
```

Código 6.3: Vectores con Numpy

Las listas también se pueden definir en una variable y luego convertirla en vector con sólo colocar el nombre de la variable asignada, es decir:

```
1 import numpy as np
2 # Escribir:
3 L=[1,2,3,4]
4 V=np.array(L)
5 # Es lo mismo que hacer:
6 V=np.array([1,2,3,4])
```

Código 6.4: Convertir una lista en vector con su variable asignada

6.2.2. Matrices con el comando array

Como se había dicho al principio del capítulo, los arreglos bidimensionales en programación son el equivalente a las matrices. Así, que de ahora en adelante se le dirá matriz a este tipo de arreglo (de la misma forma como le decíamos vectores a los de una dimensión). Para construir una matriz, se debe crear una lista de listas dentro de los paréntesis del comando `array`, donde todas sus sublistas son del mismo tamaño y con el mismo tipo de datos en todas.

Ejemplo: De manera análoga con el ejemplo de los vectores, se crearán matrices de strings, enteros, números en punto flotante y números complejos.

```
1 import numpy as np
2 # Matriz de strings
3 M1=np.array([[ 'ab', 'cd'], [ 'be', 'da' ]])
4
5 # Matriz de enteros
6 M2=np.array([ [-1, 2, 3], [3, -4, 5] ])
7
8 # Matriz de números en punto flotante
9 M3=np.array([ [0.34, -1.45], [-3.67, 4.1] ])
10
11 # Matriz de números complejos
12 M4=np.array([ [3+2j, 1+1j], [-1+0j, 0-3j] ])
```

Código 6.5: Matrices con Numpy

Y al igual que los vectores, también se puede definir primero la lista de listas en una variable y luego convertirla en matriz con sólo colocar la variable dentro de los paréntesis del `array`.

Observación: Para crear hipermatrices (arreglos de 3 o más dimensiones), basta con hacer más sublistas dentro de las sublistas, donde cada nivel de las sublistas ellas sean del mismo tamaño, y todos sus datos sean del mismo tipo.

6.3. Indexación en arreglos

La indexación en arreglos, es análoga a la indexación de listas, esto se puede apreciar de mejor manera por la forma en la que se construyeron las matrices en la sección 6.2, en el caso de los vectores, se recorre exactamente de la misma forma que una lista normal, en el caso de las matrices, el primer índice entre corchetes indicará la fila, y el segundo índice indicará la columna en la cual buscará el dato. (Hay que tener en cuenta que en los índices la primera fila es la fila 0 al igual que la primera columna)

Ejemplo: Sea la matriz:

$$A = \begin{bmatrix} 1 & 5 & -2 \\ 0 & 4 & -5 \end{bmatrix}$$

Entonces, los índices de cada uno de sus elementos se pueden escribir como:

```
1 import numpy as np
2 A=np.array([[1, 5, -2], [0, 4, -5]])
3 a11=A[0][0] # Aquí a11 = 1
4 a12=A[0][1] # a12 = 5
5 a13=A[0][2] # a13 = -2
6 a21=A[1][0] # a21 = 0
7 a22=A[1][1] # a22 = 4
8 a23=A[1][2] # a23 = -5
```

Código 6.6: Índices de una matriz en Python

Observación: Para el caso de las hipermatrices n -dimensionales, se deben colocar los n índices con corchetes para encontrar el elemento específico.

6.4. Funciones y operaciones con arreglos

6.4.1. Creaciones de arreglos por medio de funciones

Como se había mencionado antes, Numpy tiene una gran variedad de funciones para crear arreglos sin hacerlo manualmente, algunas de ellas se muestran en el cuadro 6.1:

Nombre	Tipo de arreglo que devuelve
<code>zeros((n,m))</code>	Arreglo de n filas y m columnas con todos sus elementos iguales a 0.
<code>ones((n,m))</code>	Parecido a la función zeros, con la diferencia de que todos sus elementos son iguales a 1.
<code>full((n,m),v)</code>	Generaliza las funciones anteriores, ya que crea un arreglo de $n \times m$ donde todos sus elementos son iguales al valor v.
<code>identity(n)</code>	Matriz identidad de orden n.
<code>random.random(n,m)</code>	Arreglo de n filas y m columnas, donde sus elementos son números aleatorios entre 0 y 1.
<code>random.randint(a,b,(n,m))</code>	Arreglo de n filas y m columnas, donde sus elementos son números enteros aleatorios que pueden ser desde a hasta b (sin incluir a este último).
<code>arange(a,b,c)</code>	Arreglo unidimensional de números que comienzan en a hasta b (sin incluir este último) equiespaciados por c.
<code>linspace(a,b,n)</code>	Arreglo unidimensional de n números equiespaciados desde a hasta b.

Cuadro 6.1: Funciones para crear arreglos

6.4.2. Atributos de un arreglo

Los atributos de un arreglo son las características generales de un arreglo específico, para ver estos atributos se pueden usar los métodos del cuadro 6.2

Nombre	Atributo devuelto
<code>A.size</code>	Número de elementos que tiene el arreglo.
<code>A.shape</code>	Una upla con las dimensiones del arreglo.
<code>A.dtype</code>	El tipo de dato que tienen los elementos.

Cuadro 6.2: Atributos de un arreglo A

Observación: Para usar estos métodos no se deben importar de Numpy ni mencionarlos con un `np.` u otro nombre con el que se le haya nombrado al paquete.

6.4.3. Operaciones con arreglos

Las operaciones matemáticas $+$, $-$, $*$, $/$, $\%$, $**$, (suma, resta, multiplicación, división, resto y potencia respectivamente) entre un arreglo y un número, entrega un arreglo del resultado de la operación elemento a elemento.

Ejemplo: Se mostrará el como funciona estos tipos de operaciones:

```

1 import numpy as np
2 A=np.array([[1,3,2],[-1,3,0]])
3 print(A**2, '\n', A+2, '\n', A-2, '\n', A/2, '\n', A*2, '\n', A%2)
```

Código 6.7: Operaciones por elemento

El código 6.7 despliega las matrices:

$$\begin{bmatrix} 1 & 9 & 4 \\ 1 & 9 & 0 \end{bmatrix}, \begin{bmatrix} 3 & 5 & 4 \\ 1 & 5 & 2 \end{bmatrix}, \begin{bmatrix} -1 & 1 & 0 \\ -3 & 1 & -2 \end{bmatrix}, \begin{bmatrix} 0,5 & 1,5 & 1 \\ -0,5 & 1,5 & 0 \end{bmatrix}, \begin{bmatrix} 2 & 6 & 4 \\ -2 & 6 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Además de estas operaciones (entre un arreglo y un número), las operaciones entre 2 arreglos también son posibles, en estos casos $+$, $-$, $*$, $/$, son elemento a elemento a elemento, y es por eso que las matrices al relacionarlas de estas formas deben ser del mismo orden. En cambio, la multiplicación real entre matrices se define con un método.

Producto matricial: Sean las matrices A y B, de orden $n \times m$ y $m \times l$ respectivamente, entonces el producto matricial $A \cdot B$ que genera una matriz de orden $m \times l$ en Python se escribe: `A.dot(B)`

Ejemplo: Sean las matrices:

$$A = \begin{bmatrix} 1 & 3 & 0 \\ -2 & 0 & 1 \end{bmatrix} \text{ y } B = \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 2 & 5 \end{bmatrix}$$

Y se quiere calcular AB

```
1 import numpy as np
2 A=np.array([[1,3,0],[-2,0,1]])
3 B=np.array([[1,-1],[0,1],[2,5]])
4 print(A.dot(B))
```

Código 6.8: Producto matricial

El código 6.8 despliega la matriz:

$$AB = \begin{bmatrix} 1 & 2 \\ 0 & 7 \end{bmatrix}$$

Transposición: Para tener transpuesta de una matriz matriz A en Python, se usa el método: `A.T`

Ejemplo: Si se pide obtener la matriz transpuesta de A (la usada en el ejemplo anterior), entonces:

```
1 import numpy as np
2 A=np.array([[1,3,0],[-2,0,1]])
3 print(A.T)
```

Código 6.9: Matriz transpuesta

Donde lo que entrega es:

$$A^T = \begin{bmatrix} 1 & -2 \\ 3 & 0 \\ 0 & 1 \end{bmatrix}$$

6.5. Ejercicios

Ejercicio 5.1: Matrices nilpotentes Una matriz A cuadrada, se dice que es nilpotente si existe un número $k \in \mathbb{N}$ tal que A^k es la matriz nula (Θ), en cuyo caso se le llama índice de nilpotencia al $k_0 = \min\{k \in \mathbb{N} : A^k = \Theta\}$. Realice un programa en Python que compruebe que una matriz es o no es nilpotente con índice de nilpotencia menor o igual a 50.

Entrada: Se debe ingresar un número entero $n > 1$ que indique el tamaño de la matriz a estudiar, seguido de n^2 números en punto flotante que serán los elementos de la matriz.

Salida: La salida será un mensaje que afirme que es una matriz nilpotente con el respectivo valor de su índice de nilpotencia, en caso de que no se encuentre un índice de nilpotencia menor que 50, deberá indicar en un mensaje que no se encontró tal valor.

Ejemplo de entrada: $n = 2$ y la matriz: $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$

Ejemplo de salida: Es una matriz nilpotente con un índice de nilpotencia $k_0 = 2$

Ejercicio 5.2: Redefinir una matriz Uno de los pasos que se requieren en los algoritmos para resolver un sistema de ecuaciones lineales consiste en intercambiar las filas de una matriz cuadrada para colocar en la diagonal principal los elementos de mayor magnitud de cada columna. Escriba un programa Python que permita ingresar una matriz cuadrada de $n \times n$, que luego intercambie las filas, desde arriba hacia abajo, de manera que el elemento de mayor magnitud de cada columna se ubique en la diagonal y sustituya con ceros el resto de la fila hacia la derecha.

Entrada: La entrada al programa consiste en una secuencia de $n^2 + 1$ números enteros, el primero es el valor de n ($n \geq 3$) y luego n^2 números enteros, correspondientes a los elementos de la matriz.

Salida: La salida será una matriz de $n \times n$ cuya estructura corresponde a la descripción del enunciado.

Ejemplo de entrada: $n = 3$ y la matriz: $\begin{bmatrix} 2 & 7 & 6 \\ 4 & 5 & 3 \\ 9 & 8 & 1 \end{bmatrix}$

Ejemplo de salida: $\begin{bmatrix} 9 & 0 & 0 \\ 2 & 7 & 0 \\ 4 & 5 & 3 \end{bmatrix}$

6.6. Soluciones

```

1 import numpy as np
2 n=int(input('Ingrese el orden de la matriz: '))
3 while n <= 1:
4     n=int(input('Ingrese un valor correcto: '))
5 O=np.zeros((n,n))
6 A=O
7 for i in range(0,n):
8     for j in range(0,n):
9         A[i][j]=float(input(f'Ingresar valor del elemento ({i,j})←
            : '))
10 A2=A
11 for k in range(1,50):
12     c=0
13     for i in range(0,n):
14         for j in range(0,n):
15             if A2[i][j] == 0:
16                 c=c+1
17     if c == n**2:
18         print('Es nilpotente, su índice de nilpotencia es:',k)
19         break
20     elif k == 49:
21         print('No se encontró índice para probar su nilpotencia')
22     A2=A2.dot(A)

```

Código 6.10: Solución ejercicio 5.1

```

1 import numpy as np
2 ## Funciones a usar ##
3 def LyV(mini,msg):
4     m=int(input(msg))
5     while m < mini:
6         print('Error, valor fuera de rango')
7         m=int(input(msg))
8     return m
9 def construccion(n):
10    B=np.zeros((n,n))
11    for i in range(n):
12        for j in range(n):
13            B[i][j]=int(input(f'Ingrese el valor del elemento ({i←
14                                ,j}): '))
15    return B
16 def hacerlecambios1(C,B,n,i):
17     maxi=abs(B[0][i])
18     j=0
19     while j < n:
20         if abs(B[j][i]) > maxi:
21             maxi=abs(B[j][i])
22             C[i][i]=B[j][i]
23             h=j
24             j=j+1
25     if i > 0:
26         for k in range(i):
27             C[i][k]=B[h][k]
28         for l in range(n):
29             B[h][l]=B[i][l]
30             B[i][l]=C[i][l]
31     return C,B
32 ##### Programa principal #####
33 n=LyV(3,'Ingrese el orden de la matriz: ')
34 B=construccion(n)
35 C=np.zeros((n,n))
36 print(B)
37 for i in range(n):
38     C,B=hacerlecambios1(C,B,n,i)
39 print(C)

```

Código 6.11: Solución ejercicio 5.2

CAPÍTULO

7

ARCHIVOS

En este último capítulo se mostrará de manera sencilla el cómo se trabaja en Python con archivos (de texto), ya sea en la creación o lectura de ellos.

Trabajar con archivos es muy útil debido a su facilidad de manejar y los datos del archivo y a la persistencia de estos, es decir, para no perder la información útil y para facilitar la ejecución de un programa.

7.1. Creación de archivos

Para crear un archivo de texto en Python, es necesario usar la función con tres entradas: `open('Nombre.txt', 'w', encoding='utf-8')`, donde, la primera entrada es el nombre que se le quiera colocar al archivo, el `'w'` es el indicador de que se escribirá el archivo, y la tercera entrada es la codificación de los símbolos a usar (es recomendable usar el utf-8). Hay que tener en consideración que el archivo se creará en la misma carpeta que se encuentra el programa. Para manipular el archivo creado (escribir en él), hay que guardarlo en una variable, y posterior a la manipulación de él, es recomendable cerrar la memoria del archivo en el programa con el método: `close()`, como se ve en el siguiente código:

```
1 # Creación del archivo
2 A=open('archivo.text', 'w', encoding='utf-8')
3
4 # Se manipula después del open y antes del close
5
6 # Cerramos la memoria del archivo en el programa
7 A.close()
```

Código 7.1: Estructura de la creación de un archivo

Para escribir en el documento se usa el método `write()` donde el texto a escribir se debe colocar dentro de los paréntesis y entre comillas.

Ejemplo 1: A continuación, se creará un archivo de texto de 10 líneas que en cada una de ellas tenga escrito el número de su línea correspondiente.

```
1 A=open('lineas.text', 'w', encoding='utf-8')
2 for i in range(1,10):
3     A.write(f'{i}\n')
4 A.write(f'{i+1}')
5 A.close()
```

Código 7.2: Ejemplo 1 de escribir en un archivo

En este caso se escribe algo netamente indicado en el programa, sin embargo, también puede depender de quien ejecute el programa.

Ejemplo 2: Para crear un archivo de texto de una cantidad n de líneas con texto, donde en cada línea esté escrito lo que el usuario desea escribir, con una longitud máxima de 20 caracteres por línea, se realiza el siguiente programa:

```
1 n=int(input('Cantidad de líneas a escribir: '))
2 while n <= 0:
3     n=int(input('Ingresar una cantidad de líneas correcta: '))
4 A=open('archivo.text', 'w', encoding='utf-8')
5 for i in range(n):
6     lineai=input(f'Escritura en la línea {i+1}: ')
7     while len(lineai) > 20:
8         print('Cantidad de caracteres no permitida')
9         lineai=input(f'Escritura en la línea {i+1}: ')
10    A.write(f'{lineai}\n')
11 A.close()
```

Código 7.3: Ejemplo 2 de escribir en un archivo

También puede ser útil el método `writelines()`, ya que al ingresarle dentro de los paréntesis una lista con strings terminados con `'\n'`, escribe en cada línea lo que cada elemento cadena de la lista dice.

Ejemplo: Si se quiere crear un archivo de 3 líneas, donde la primera dice 'Hola', la segunda 'ABC', y la tercera '718', entonces se puede hacer lo siguiente:

```
1 archivo=open('code.txt', 'w', encoding='utf-8')
2 archivo.writelines(['Hola\n', 'ABC\n', '718'])
3 archivo.close()
```

Código 7.4: Ejemplo de uso del método `writelines`

7.2. Lectura de archivos

Para leer un archivo se hace uso de la misma función vista para crearlo, con la diferencia que ahora es: `open('Nombre.txt', 'r', encoding='utf-8')`, ya que la 'r' es el indicador de que se leerá el archivo llamado "Nombre.txt".

Para almacenar todo el texto de un archivo desde el inicio hasta un número de símbolos b, se usa el método `read(b)`, en caso de querer guardar toda la información de un archivo se omite ingresar el b en el método, de la siguiente forma:

```

1  # Se abre el archivo que se quiere leer
2  archivotexto=open('archivo.text', 'r', encoding='utf-8')
3
4  # Guardamos en la variable texto toda la información
5  # (en forma de strings) del archivo
6  texto=archivotexto.read()
7
8  archivotexto.close()
9
10 # Se puede desplegar la información obtenida
11 print(texto)

```

Código 7.5: Guardar toda la información de un archivo

En caso de no querer guardar toda la información de un archivo sino que una línea de él, se usa el método `readline()`, cada vez que se vuelve a usar este método en el mismo archivo, se leerá la información que hay en la siguiente línea. Hay que tener en cuenta que también considera los saltos de línea. **Ejemplo:** Considerando que tenemos un archivo de texto llamado "code.tex", que tiene escrito los siguientes 4 códigos:

```

1 1 0 0
1 0 1 1
1 0 0 1
0 1 0 1

```

Si se quiere crear un programa capaz de almacenar cada uno de estos códigos en una lista llamada L, una forma de hacerlo es:

```

1 L=list()
2 archivo=open('code.txt', 'r', encoding='utf-8')
3 for i in range(4):
4     texto=archivo.readline()
5     texto=texto.replace('\n', '') # Se eliminan los saltos de línea
6     L.append(texto)
7 archivotexto.close()
8 print(L)

```

Código 7.6: Ejemplo del uso del método readline

Sin embargo, existe una forma un poco más rápida de resolver el ejemplo anterior, y esa forma es haciendo el uso de otro método, llamado `readlines()`, el cual crea una lista donde cada elemento es un string de toda la información de cada línea del archivo (incluyendo los saltos de línea). Si se quiere hacer un programa que haga lo mismo que el código 7.6, usando este nuevo método, sería:

```
1 archivo=open('code.txt', 'r', encoding='utf-8')
2 L=archivo.readlines()
3 for i in range(len(L)):
4     L[i]=L[i].replace('\n', '')
5 archivo.close()
6 print(L)
```

Código 7.7: Ejemplo del uso del método `readlines`

Estos métodos de lectura siempre comienzan en donde esté la posición del puntero (si no se ha usado ningún método de lectura antes, se comienza del principio del archivo, si se ha usado uno antes, el nuevo comienza donde terminó el anterior. En caso de haber leído todo el archivo, el puntero se colocará al final del archivo y ya no podrá leer nada más si se usa otro método más de lectura), la posición de este puede ser modificada manualmente para comenzar a leer de diferentes lugares usando el método: `seek(a)` donde `a` es la nueva posición del puntero. La posición del puntero se puede ver como el índice de la cadena del archivo total, y como los índices de estos, igual empieza en 0.

Usando el archivo del ejemplo anterior, para desplegar sólo el segundo código como string, es decir, '1011', podemos hacer el siguiente programa:

```
1 archivo=open('code.txt', 'r', encoding='utf-8')
2 archivo.seek(7)
3 L=archivo.read(8)
4 archivo.close()
5 print(L)
```

Código 7.8: Uso del método `seek`

7.3. Agregar información a un archivo

Si ya se tiene un archivo con información, y se quiere agregar información a él, se usa: `open('Nombre.txt', 'a', encoding='utf-8')`, ya que al volver a usar la función con el `'w'`, este borra toda la información anterior para volver a escribir desde cero.

Ejemplo: Usando el archivo creado anteriormente “code.txt”, y se necesita agregar en una siguiente línea el nuevo código: `'1 0 1 0'` se puede hacer el siguiente programa:

```
1 archivo=open('code.txt', 'a', encoding='utf-8')
2 archivo.write('\n1 0 1 0')
3 archivo.close()
```

Código 7.9: Ejemplo de agregar información a un archivo

Observación: Aunque estas tres maneras de abrir un archivo son para realizar cosas específicas (`'w'` = escribir, `'r'` = leer `'a'` = agregar), también existe una manera de abrir el archivo de forma que se pueda hacer más de una: `'r+'` = leer y escribir en el archivo. De esta forma, para hacer lo mismo que en el ejemplo recién visto, y desplegar la cantidad de códigos que tiene, se realiza lo siguiente:

```
1 archivo=open('code.txt', 'r+', encoding='utf-8')
2 L=archivo.readlines()
3 archivo.writelines(['\n1 0 1 1'])
4 archivo.close()
5 print('La cantidad de códigos son:', len(L)+1)
```

Código 7.10: Ejemplo de leer y escribir información en un archivo

7.4. Ejercicios

Ejercicio 6.1: Verificar números primos y encontrar siguiente Usando un archivo donde cada una de sus líneas sólo tenga escrito un número entero positivo, se debe construir un programa en Python que verifique que cada uno de esos números sean primos o no, en cualquiera de los casos, debe entregar un mensaje que diga si lo es o no indicando la línea en la que se encuentra el número, luego, se colocará al final del archivo (en una nueva línea) el número primo que viene después del primo más grande encontrado en el archivo. El archivo se llamará “NumerosP.txt”.

Ejemplo de archivo a usar:

3
9
13
7

Ejemplo de salida:

En la línea 1 hay un primo.
En la línea 2 no hay un primo.
En la línea 3 hay un primo.
En la línea 4 hay un primo.
(Por último, se agrega el número 17 al archivo)

Ejercicio 6.2: Ecuación de la recta (versión con archivos) Recordando el ejercicio 2.2, se pide ahora que los puntos (x_1, y_1) , (x_2, y_2) deben estar en un archivo, sin embargo, el archivo tiene una cantidad de líneas por determinar, donde hay 4 números en cada línea separados por espacios: $x_1 \ y_1 \ x_2 \ y_2$, el programa debe ser capaz de entregar la cantidad de rectas igual a la cantidad de líneas de puntos distintos, e indicar con un mensaje los pares que son iguales (si es que hay). El archivo a usar se llamará “puntos.txt”

Ejemplo de archivo a usar:

1 0 2 0
2 3 2 3
-2 4 0 -2
2 90 2 15

Ejemplo de salida:

Recta del par de la línea 1: $y = 0x$
Recta del par de la línea 2: Es un par idéntico.
Recta del par de la línea 3: $y = -3x - 2$
Recta del par de la línea 4: $x = 2$

7.5. Soluciones

```
1  # Consideraremos que el archivo tiene los datos correctos,
2  # es decir, no se comprobará que sólo son números enteros;0
3  # Función a usar:
4  def primo(p):
5      if p == 1:
6          M='no hay un primo'
7      elif p == 2:
8          M='hay un primo'
9      else:
10         c=0
11         for j in range(2,p//2+1):
12             if p%j == 0:
13                 c=1
14             if c == 0:
15                 M='hay un primo'
16             else:
17                 M='no hay un primo'
18         return M
19 # Programa principal:
20 S=open('NumerosP.txt', 'r+', encoding='utf-8')
21 L,L2=S.readlines(),[]
22 for i in range(len(L)):
23     L[i]=int(L[i].replace('\n', ''))
24     print(f'En la línea {i+1}',primo(L[i]))
25     if primo(L[i]) == 'hay un primo':
26         L2.append(L[i])
27 m=max(L2)
28 while True:
29     m=m+1
30     if primo(m) == 'hay un primo':
31         S.writelines(['\n'+str(m)])
32         break
33 S.close()
```

Código 7.11: Solución ejercicio 6.1

```

1  # (Los datos en el archivo se considerarán correctos,
2  # es decir, no hay que verificar nada)
3  # La función a usar es:
4  def recta(x1,y1,x2,y2):
5      if x1 == x2 and y1 != y2:
6          L='x = '+str(x1)
7      elif x1 == x2 and y1 == y2:
8          L='Es un par idéntico.'
9      else:
10         m=(y1-y2)/(x1-x2)
11         c=y1-x1*m
12         if c > 0:
13             L='y = '+str(m)+'x + '+str(c)
14         elif c < 0:
15             L='y = '+str(m)+'x - '+str(abs(c))
16         elif m == 0:
17             L='y = 0'
18         else:
19             L='y = '+str(m)+'x'
20     return L
21 # Programa principal:
22 O=open('puntos.txt','r',encoding='utf-8')
23 L=O.readlines()
24 O.seek(0)
25 for i in range(len(L)):
26     XY,x1,y1,x2,y2,c=O.readline().replace('\n',''),'', '', '', '',0
27     while XY[c] != ' ':
28         x1,c=x1+XY[c],c+1
29     c,x1=c+1,float(x1)
30     while XY[c] != ' ':
31         y1,c=y1+XY[c],c+1
32     c,y1=c+1,float(y1)
33     while XY[c] != ' ':
34         x2,c=x2+XY[c],c+1
35     c,x2=c+1,float(x2)
36     while c < len(XY):
37         y2,c=y2+XY[c],c+1
38     c,y2=c+1,float(y2)
39     print(f'Recta del par de la línea {i+1}:',recta(x1,y1,x2,y2))

```

Código 7.12: Solución ejercicio 6.2

BIBLIOGRAFÍA

- [1] A. Downey, J. Elkner y C. Meyers, *Aprenda a pensar como un programador con Python*. Green Tea Press, Wellesley, Massachusetts
- [2] The NumPy community, *NumPy User Guide*, Release 1.20.0
- [3] Charles R. Severance, *Python para todos, Explorando la información con Python*
3
- [4] A. Marzal e I. Gracia, *Introducción a la programación con Python*, Universitat Jaume