Funciones en Python

Jorge Mario López Pereira

2023

Contenido

1. Introducción
2. Elementos del Lenguaje Python
3. Funciones
4. Numpy
5. Números Aleatorios
6. Algoritmo Genético

Introducción

Antes de Comenzar

NO se requiere ser experto en programación de computadores SI se requiere saber los conceptos básicos de álgebra lineal

Nota 2

Este documento está diseñado para Introducirte al lenguaje Python y muestra cómo hacer operaciones con matrices de una forma fácil y rápida.

¿Qué es el Lenguaje Python?

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma. Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Instalación

Para desarrollar adecuadamente este curso recomendamos instalar el software Anaconda, el cual contiene Python y los paquetes que usaremos:

- Descargar Anaconda Python: https://www.anaconda.com/download/
- Video Tutorial Instalar Anaconda: https://www.youtube.com/watch?v=YQu4OPmQ8Q0
- Tutorial Python (ultima versión): https://docs.python.org/3/tutorial/
- Jupyter Notebook, viene incluido en Anaconda: http://jupyter.org/
- Paquetes relacionados (Incluidos en Anaconda): Numpy, Scipy, Matplotlib, Pandas

Entorno de trabajo Python

Anaconda trae múltiples entornos de programación donde puedes realizar tus código, los recomendados en este curso son Spyder y Jupyter Notebook.

Spyder

Spyder es un entorno de desarrollo clásico, puedes empezar hacer el código de tu programa, ejecutarlo y ver los resultados en el terminal incluido en el programa.

Jupyter Notebook

Es un entorno de trabajo interactivo que permite desarrollar código en Python (puede permitir otros lenguajes) de manera dinámica, a la vez que integrar en un mismo documento tanto bloques de código como texto, gráficas o imágenes.

Contenido

1 Introducción

2 Elementos del Lenguaje Python

3 Funciones

4 Numpy

5 Números Aleatorios

6 Algoritmo Genético

Variables

Una variable es un espacio para almacenar datos modificables, en la memoria de un ordenador. En Python, una variable se define con la sintaxis:

variable = valor

Ejemplos

a = 5
b = 'hola'
c = [1 ,2 , 3]

Los tres casos muestra como se realiza la asignación de variables en Python, lo cual es igual independiente del tipo de datos.

Tipos de Datos

Una variable (o constante) puede contener valores de diversos tipos. Entre ellos:

- Cadena de texto (string): a = 'Hola Mundo'
- Número Entero: b = 45
- Número Real: c = 23.4566
- Booleano (Verdadero / Falso):

d = True

e = False

- Lista (varios elementos):

g = [3, 4, 5.34, 'a']

Operadores Aritméticos

Simbolo	Significado	Ejemplo	Resultado
+	Suma	a = 3 + 4	a es 7
-	Resta	b = 5 - 7	b es -2
*	Multiplicación	c = 4 * 3	c es 12
/	División	d = 10 / 2	d es 5
**	Potenciación	e = 2 ** 3	e es 8
//	Division entera	f = 10 // 3	f es 3
%	Modulo de la división	g = 10 % 3	g es 1

Ejemplos

a = 5
b = 7
c = a * b
c toma el valor de 35

Listas, Tuplas y Diccionario

Estos tres tipos, pueden almacenar colecciones de datos de diversos tipos y se diferencian por su sintaxis y por la forma en la cual los datos pueden ser manipulados.

Listas

Las listas son variables que pueden tener en su interior otros tipos de datos para usarlos se usan [ ] por ejemplo:

```
a = [3, 5, 7]
```

Para acceder a un elemento de la lista se usa el nombre de la variable y la posición que se quiere acceder (las posiciones se cuentan desde 0), por ejemplo:

```
b = a[1]
```

En este caso b toma el valor de 5

Listas, Tuplas y Diccionarios II

Listas

Las Tuplas son variables que pueden tener en su interior otros tipos de datos inmutables (no pueden ser modificados), para asignarlos se usan ( ) y para acceder a un dato en ella se usan [ ], por ejemplo:

```
a = (3, 5, 7)
```

Para acceder a un elemento de la lista se usa el nombre de la variable y la posición que se quiere acceder (las posiciones se cuentan desde 0), por ejemplo:

```
b = a[1]
```

En este caso b toma el valor de 5

Listas, Tuplas y Diccionarios III

Diccionarios

Mientras que a las listas y tuplas se accede solo y únicamente por un número de índice, los diccionarios permiten utilizar una clave para declarar y acceder a un valor:. para usarlos se usan llaves { } por ejemplo:

a = {'Id_1':3, 'Id_2': 5, 'Id_3' : 7 }

Para acceder a un elemento del diccionario se usa el nombre de la variable y la identificación que se quiere acceder, por ejemplo:

b = a['Id_2']

En este caso b toma el valor de 5

Operadores Lógicos

Para describir la evaluación a realizar sobre una condición, se utilizan operadores relacionales (o de comparación):

Símbolo	Significado	Ejemplo	Resultado
==	Igual que	7 == 3 + 4	True
!=	Diferente que	-2 != 5 - 7	False
<	Menor que	11 < 4 * 3	True
>	Mayor que	4 > 10 / 2	False
<=	Menor o Igual que	8 <= 2 ** 3	True
>=	Mayor o Igual que	2 >= 10 // 3	False
and	y(lógico)	4>3 and 5<6	True
or	O (Lógico)	2>5 or 5<6	True
extasciicircum{}	O exclusivo (xor)	2>5 ^ 5<6	True

Estructura de Control Condicional (if)

Las estructuras de control condicionales, son aquellas que nos permiten evaluar si una o más condiciones se cumplen, para decir qué acción vamos a ejecutar. La evaluación de condiciones, solo puede arrojar 1 de 2 resultados: verdadero o falso (True o False).

Ejemplo

```
a = 7
b = 5
if a < b:
    print('a es menor que b')
elif a > b:
    print('a es mayor que b')
else:
    print('a y b son iguales')
```

Estructura de Control Iterativa

A diferencia de las estructuras de control condicionales, las iterativas (también llamadas cíclicas o bucles), nos permiten ejecutar un mismo código, de manera repetida, mientras se cumpla una condición. En Python se dispone de dos estructuras cíclicas:

- Ciclo While
- Ciclo For

Ciclo While

Este bucle, se encarga de ejecutar una misma acción "mientras que" una determinada condición se cumpla:

Ejemplo Imprimir los números del 5 al 7

Código

```
a = 5
while a <= 7:
    print(a)
    a = a +1
```

Salida 5
6
7

Ciclo For

El bucle for, en Python, es aquel que nos permitirá iterar sobre una variable compleja, del tipo lista o tupla:

Ejemplo Imprimir las siguientes letras ‘a’, ‘b’ y ‘c’

Código

```
letras = ['a', 'b', 'c']
for i in letras:
    print(i)
```

Salida a
b
c

Ciclo For II

También podemos hacer una lista iterable con la función range por ejemplo range(3) puede dar una lista iterable con los números 0, 1 y 2:

Ejemplo Imprimir los números de 0 al 3

Código

```
for i in range(3):
    print(i)
```

Salida 0
1
2

Contenido

1. Introducción
2. Elementos del Lenguaje Python
3. Funciones
4. Numpy
5. Números Aleatorios
6. Algoritmo Genético

Funciones

- Las funciones son bloques de código que se pueden ejecutar múltiples veces.
- Permiten encapsular código y reutilizarlo en distintas partes de un programa.
- En Python, se definen utilizando la palabra clave `def`.

Sintaxis de funciones

- La sintaxis para definir una función en Python es la siguiente:

Sintaxis

```
def nombre_de_la_funcion(argumentos):
    # Cuerpo de la función
    return valor_de_retorno
```

- El nombre de la función debe seguir las mismas reglas que el nombre de las variables.
- Los argumentos son opcionales y se separan por comas.
- El cuerpo de la función es el bloque de código que se ejecuta cada vez que se llama a la función.
- La instrucción `return` indica el valor que se devuelve al llamar a la función.

Ejemplo de función

- Veamos un ejemplo sencillo de función que suma dos números:

Ejemplo

```
def sumar(a, b):
    resultado = a + b
    return resultado
```

- Para llamar a la función, simplemente utilizamos su nombre y los argumentos correspondientes:

Ejemplo

```
resultado = sumar(3, 5)
print(resultado) # Imprime 8
```

Argumentos por defecto

- En Python, es posible asignar valores por defecto a los argumentos de una función:

Ejemplo

```
def saludar(nombre, saludo="Hola"):
    print(saludo, nombre)
```

- Si no se especifica el valor del segundo argumento, se utiliza el valor por defecto:

```
saludar("Juan") # Imprime "Hola Juan"
```

Funciones con número variable de argumentos

- En Python, es posible definir funciones que acepten un número variable de argumentos:

Ejemplo

```
def sumar_varios(*numeros):
    resultado = 0
    for numero in numeros:
        resultado += numero
    return resultado
```

- El argumento que permite aceptar un número variable de argumentos se define utilizando el operador $*$.
- Dentro de la función, el argumento se comporta como una tupla.

Contenido

1 Introducción

2 Elementos del Lenguaje Python

3 Funciones

4 Numpy

5 Números Aleatorios

6 Algoritmo Genético

Importar Librerías

La comunidad de Python es muy amplia y han elaborado paquetes (librerías) que aumentan las funcionalidades en Python, para llamar una librería en Python se usa la función import seguida del paquete, también se pueden usar alias usando la palabra as seguida del alias deseado.

Ejemplo

```
import numpy as np
```

El paquete numpy Introduce objetos multidimensionales vectores(array) y matrices , también tiene funciones avanzadas de matemáticas y estadísticas

Crear Arreglos con numpy

crearemos un array en NumPy a través de la función array(), pasándole una lista con los elementos que queremos pertenezcan al mencionado array.

Ejemplo

```
import numpy as np
a = np.array([3, 4, 5])
b = np.array([[3 , 4 , 5],[6, 7, 8]])
print(a)
print(b)
```

En este ejemplo la variable a es un vector de 3 elementos y la variable b es una matriz de 2 filas 3 columnas

Crear Arreglos Especiales

La librería NumPy contiene una serie de funciones orientadas a generar arrays especiales como, por ejemplo, aquellos cuyas componentes son todas nulas, todas de unos.

Ejemplo

```
import numpy as np
a = np.zeros((3,2))
b = np.ones((2,3))
c = np.identity(3) #Tambien se puede con np.eye(3)
print(a)
print(b)
```

En este ejemplo la variable a es una matriz de ceros de 3 filas y 2 columnas, la variable b es una matriz de unos de tamaño 2 filas 3 columnas, la variable c es una matriz identidad de tamaño 3x3.

Manipulación de Arreglos

para acceder a un dato almacenado en una variable arreglo de numpy es necesario usar el nombre de la variable y entre corchetes colocar la pocision de la fila y columna que queremos separada por coma.

Ejemplo

```
import numpy as np
a = np.array([[3, 4, 5],[6, 7, 8]])
print(a[1,2])
```

En este ejemplo la salida del código es 8, se debe recordar que el indice de la posición en Python inicia en 0, por lo tanto a[1, 2] esta accediendo a la fila 2 columna 3.

Manipulación de Arreglos II

Se puede modificar el valor de una dato en un arreglo, solo se requiere la posición en fila y columna que se quiere cambiar.

Ejemplo

```
import numpy as np
a = np.array([[3, 4, 5],[6, 7, 8]])
a[1, 2] = 4
print(a[1,2])
```

En este ejemplo la salida del código es 4

Manipulación de Arreglos III

Si necesitamos seleccionar un rango de filas, podemos especificar el rango usando ":", este operador puede ser utilizado de la siguiente forma 0:2, indica que muestra la lista [0, 1]

Ejemplo

```
import numpy as np
a = np.array([[3, 4, 5],[6, 7, 8]])
print(a[1, 0: 2])
```

En este ejemplo la salida del código es [6, 7], la expresión a[1, 0:2] muestra la fila 2, columnas 1 y 2 (no se incluye el ultimo dato).

Manipulación de Arreglos IV

Otros Ejemplos del uso del operador : son los siguientes

Ejemplo

```
a[0]  # primera fila
a[i]  #(i+1)esima fila
a[-1] # última fila
a[:, 0]  # Primera columna
a[:, -1] # Última columna
a[0:7]        #Primeras 7 filas
a[:, 0:2]     #primeras 2 columnas
a[1:3, 0:2]
#Segunda hasta la tercera fila y primeras 2 columnas
a[[0,5], [1,3]]
#1ra y 6ta fila and 2da y 4ta columnas
```

Manipulación de Arreglos V

Otros Ejemplos del uso del operador : son los siguientes

Ejemplo

```
a[1:, 0]
# filas desde 2 hasta la ultima fila, Primera columna
a[:3, -1] # primeras tres filas, última columna
a[1:3,1:5:2] #filas 2 y 3, columnas 2 y 4
```

Resumen de los : Como se pudo observar en los ejemplos del uso de los : la sintaxis que se necesita tiene el siguiente orden, a:b:c, donde a es la posición inicial, b es la posición final (la cual no se incluye) y c es el salto, por ejemplo, 1:10:2, da como resultado los números [1, 3, 5, 7, 9]

Eliminación y Concatenación de Arreglos

Para eliminar elementos de un arreglo se usa np.delete(a, index), donde a es el arreglo original y el index es una lista que indica la posición de los elementos a borrar.

Ejemplo

```
import numpy as np
a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
index = [1, 4, 6]
new_a = np.delete(a, index)
print(new_a)
```

Salida El código anterior imprime en pantalla los números [1, 3, 4, 6, 8, 9]

Eliminación de Filas y Columnas

Para eliminar filas o columnas de un arreglo se usa np.delete(a, index, axis=b), donde a es el arreglo original, index es una tupla que indica la posición de los elementos a eliminar y b es 0 para borrar filas ó 1 para columnas.

Ejemplo

```
import numpy as np
x = np.array([[1,2,3],[4,5,6],[7,8,9]])
y = np.delete(x, (0), axis=0)
# Elimina la primera fila
z = np.delete(x, (2), axis=1)
# Elimina la tercera columna
```

Ejemplo

variable y muestra [[4, 5, 6],[7, 8, 9]]
En la variable z da como resultado [[1, 2],[4, 5],[7, 8]]

Insertar de Elementos en Arreglos

Para eliminar elementos de un arreglo se usa np.insert(a, index, v), donde a es el arreglo original y el index es una lista que indica la posición donde se insertará el elemento y v es el valor a insertar.

Ejemplo

```
import numpy as np
a = np.array([1,2,3,4])
new_a= np.insert(a, 2, 14)
print(new_a)
```

Salida

El código anterior imprime en pantalla los números [1, 2, 14, 3, 4]

Insertar Filas o Columnas

Para insertar filas o columnas de un arreglo se usa np.insert(a, index, v, axis=b), donde a es el arreglo original, index es un entero que indica la posición de los elementos a insertar, v es una lista con los datos y b es 0 para agregar filas ó 1 para columnas.

Ejemplo

```
import numpy as np
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
y= np.insert(a, 1, [10, 11, 12], axis=0)
# agrega en vector a la segunda fila
z= np.insert(a, 2, [10, 11, 12], axis=1)
# inserta el vector en la tercera columna
```

El código anterior da como resultado en la variable y lo siguiente [[1, 2, 3],[10, 11, 12],[4, 5, 6],[7, 8, 9]]

En la variable z da como resultado [[1, 2, 10, 3],[4, 5, 11, 6],[7, 8, 12, 9]]

Operaciones Matemáticas con Arreglos

Los arrays permiten hacer operaciones aritméticas entre ellos haciéndolo elemento a elemento; para ello ambos arrays deben tener la misma longitud.

Ejemplo

```
import numpy as np
a = np.array([ 0, 1,  2,  3,  4,  5])
x = np.array([5.6, 7.3, 7.7, 2.3, 4.2, 9.2])
print('Suma: ')
print(x + a)
print('Multiplicación: ')
print(x * a)
```

Salida:

Suma: [5.6, 8.3, 9.7, 5.3, 8.2, 14.2]
Multiplicación: [0., 7.3, 15.4, 6.9, 16.8, 46.]

Operaciones Matemáticas con Arreglos II

Los arrays de numpy tienen métodos o funciones específicas, algunas de ellas son:

Métodos:

```
x.max()   # Valor máximo de los elementos del array
Out[1]: 9.2
x.min()   # Valor mínimo de los elementos del array
Out[2]: 2.3
x.mean()  # Valor medio de los elementos del array
Out[3]: 6.05
x.std()   # Desviación típica de los elementos
Out[4]: 2.305609102457165
x.sum()   # Suma de todos los elementos del array
Out[5]: 36.3
np.median(x) # Mediana de los elementos del array
Out[6]: 6.449999999999999
```

Multiplicación Matricial

Además de las operaciones aritméticas básicas, los arrays de numpy tienen métodos o funciones específicas para álgebra lineal, por ejemplo para multiplicar dos matrices se usa:

Ejemplo:

```
import numpy as np
a = np.array([[1, 0], [2, -1]])
b= np.array([[2, 1], [-1, 0]])
c= np.dot(a, b)
print(c)
```

En el ejemplo anterior se usa el método np.dot() para hacer la multiplicación de la matriz a por b, dando como resultado [[2, 1],[5, 2]].

Potenciación Matricial

Para elevar una matriz a una potencia se usa np.linalg.matrix_power:

Ejemplo:

```
import numpy as np
a = np.array([ [1,1],
[0,1] ])
result = np.linalg.matrix_power(a, 4)
print(result)
```

Dando como resultado la matriz[[1, 4],[0, 1]]

Inversa de una Matriz

Para conseguir la inversa de una matriz se usa np.linalg.inv:

Ejemplo:

```
import numpy as np
a = np.array([ [1,1],[0,1] ])
result = np.linalg.inv(a)
print(result)
```

Dando como resultado la matriz [[1., -1.],[0., 1.]]

Transpuesta de una Matriz

Para conseguir la transpuesta de una matriz se usa el atributo .T, por ejemplo a.T:

Ejemplo:

```
import numpy as np
a = np.array([ [1,2],[3,4] ])
result = a.T
print(result)
```

Dando como resultado la matriz [[1, 3],[2, 4]]

Ejercicio

Resolver el siguiente sistema de ecuaciones:

$$2x + y - z = 1$$
$$x + 3y + z = 4$$
$$x + y + z = 6$$

Ejemplo:

```
import numpy as np
a = np.array([ [2,1,-1],[1,3,1],[1,1,1] ])
b= np.array([[1],[4],[6]])
inversa = np.linalg.inv(a)
resultado = np.dot(inversa, b)
print(resultado)
```

Dando como resultado la el vector columna [[3.],[-1.],[4.]] por lo tanto

Contenido

1. Introducción
2. Elementos del Lenguaje Python
3. Funciones
4. Numpy
5. Números Aleatorios
6. Algoritmo Genético

Números Aleatorios

- Utilice la función `numpy.random.rand()` para generar un número aleatorio entre 0 y 1.
- Puede proporcionar un argumento para especificar la forma de la matriz que desea generar.

Ejemplo:

```
import numpy as np

# Generar un número aleatorio entre 0 y 1
rand_num = np.random.rand()
print(rand_num)

# Generar una matriz de 3 filas y 2 columnas de números aleatorios
rand_matrix = np.random.rand(3, 2)
print(rand_matrix)
```

Generación de Matrices Aleatorias

- Utilice la función `numpy.random.randn()` para generar una matriz de números aleatorios con una distribución normal (gaussiana).
- Puede proporcionar un argumento para especificar la forma de la matriz que desea generar.

Ejemplo:

```
import numpy as np

# Generar una matriz de 3 filas y 2 columnas de números aleatorios
#con una distribución normal (gaussiana)
randn_matrix = np.random.randn(3, 2)
print(randn_matrix)
```

Generación de Matrices Aleatorias Enteras

- Utilice la función `numpy.random.randint()` para generar una matriz de números enteros aleatorios.
- Puede proporcionar un argumento para especificar el valor mínimo, valor máximo y la forma de la matriz que desea generar.

Ejemplo:

```
import numpy as np

# Generar una matriz de 3 filas y 2 columnas de
# números enteros aleatorios entre 0 y 10
randint_matrix = np.random.randint(0, 10, size=(3, 2))
print(randint_matrix)
```

Elegir de Forma Aleatoria (Muestra)

- La función `numpy.random.choice()` se utiliza para generar una muestra aleatoria de valores de una matriz dada.
- Puede proporcionar un argumento para especificar la matriz de entrada y la cantidad de muestras que desea generar.

Ejemplo:

```
import numpy as np

# Generar una muestra aleatoria de 3 elementos de la matriz dada
my_array = np.array([1, 2, 3, 4, 5])
random_sample = np.random.choice(my_array, size=3)
print(random_sample)
```

Elegir de Forma Aleatoria (Muestra) II

La función `np.random.choice()` utiliza estas probabilidades para generar una muestra aleatoria de elementos de la matriz `my_array`, donde los elementos más probables tienen una mayor probabilidad de ser seleccionados.

Ejemplo:

```
import numpy as np

# Generar una muestra aleatoria de 3 elementos de la matriz dada con proba
my_array = np.array([1, 2, 3, 4, 5])
probabilidad = [0.1, 0.2, 0.3, 0.2, 0.2]
random_sample = np.random.choice(my_array, size=3, p=probabilidad)
print(random_sample)
```

En este ejemplo, `probabilidad` es una lista que especifica las probabilidades de cada valor en `my_array`. La suma de todas las probabilidades debe ser 1.

Generación de Aleatorios Permutados

La permutación aleatoria de un conjunto de datos implica reordenar los elementos en un orden aleatorio.

En Python, puede utilizar la función `numpy.random.permutation()` para generar una permutación aleatoria de una matriz o secuencia.

Ejemplo:

```
import numpy as np

# Generar una permutación aleatoria de la matriz dada
my_array = np.array([1, 2, 3, 4, 5])
random_permutation = np.random.permutation(my_array)
print(random_permutation)
```

La función devuelve una nueva matriz con los elementos en un orden aleatorio.

Contenido

1. Introducción
2. Elementos del Lenguaje Python
3. Funciones
4. Numpy
5. Números Aleatorios
6. Algoritmo Genético

Algoritmo Genético

- Los algoritmos genéticos son una técnica de búsqueda y optimización inspirada en la evolución biológica.
- Se utilizan para encontrar soluciones aproximadas a problemas difíciles.
- A menudo se aplican en problemas de optimización combinatoria, como el problema del viajante o el problema de la mochila, programación de la producción.

Conceptos básicos

Cromosomas

- Un cromosoma es una cadena de genes que representan una solución candidata al problema.
- Cada gen representa una característica de la solución.

Población

- Una población es un conjunto de cromosomas que se utilizan para la evolución.
- Al principio, la población se genera aleatoriamente o utilizando algún método heurístico.

Función de aptitud

- La función de aptitud (fitness) mide cuán buena es una solución candidata.
- Cuanto mayor sea la aptitud, mejor será la solución.

Operadores genéticos

Selección

- La selección es el proceso de elegir los cromosomas más aptos de la población actual para la reproducción.
- Los cromosomas más aptos tienen más probabilidades de ser seleccionados.

Cruzamiento

- El cruzamiento (crossover) es el proceso de crear nuevos cromosomas a partir de dos cromosomas padres.
- Se seleccionan dos padres y se intercambia información genética entre ellos para crear uno o varios hijos.

Mutación

- La mutación es el proceso de cambiar aleatoriamente uno o varios genes en un cromosoma.
- La mutación es importante para mantener la diversidad genética en la población.

Pseudocódigo del algoritmo genético

Algoritmo Genético

1. Generar una población inicial aleatoria o utilizando algún método heurístico.
2. Evaluar la aptitud de cada cromosoma en la población.
3. Repetir hasta que se alcance un criterio de parada:
   1. Seleccionar una matriz de padres de la población actual utilizando algún método de selección (del mismo tamaño de la población actual).
   2. Crear una matriz de hijos a partir de los padres utilizando algún método de cruzamiento (mismo tamaño de la población).
   3. Mutar algunos genes de los hijos utilizando algún método de mutación.
   4. Evaluar la aptitud de los hijos.
   5. Reemplazar algunos cromosomas de la población actual con los hijos más aptos.
4. Devolver el cromosoma más apto como solución.

Conclusiones y Reto

- Los algoritmos genéticos son una técnica poderosa para la búsqueda y optimización de soluciones aproximadas a problemas difíciles.
- Se inspiran en la evolución biológica y utilizan conceptos como cromosomas, poblaciones, función de aptitud, selección, cruzamiento y mutación.
- Programa algoritmos genéticos en Python que busque minimizar la función rastrigin.

Muchas Gracias