0. Practicando con Python

¡Bienvenidos a **Matemática Aplicada**! Este curso es una oportunidad para que aprendas a resolver situaciones problemáticas utilizando herramientas matemáticas como funciones, sucesiones y matrices, todo con el apoyo de Python.

El curso se desarrollará en el laboratorio de computación, donde lo que aprendas podrás aplicarlo para resolver problemas que podrías enfrentar en tu futuro profesional. Python no solo te ayudará a encontrar soluciones, si no que también facilitará la comprensión de los conceptos matemáticos claves.

Este curso te brindará herramientas valiosas. ¡Prepárate para desarrollar tus habilidades en Matemática Aplicada y Python!

0.1 Tipos de datos

Recordemos que en cursos anteriores, has visto que existen varios tipos de datos que se utilizan para representar diferentes tipos de valores y estructuras de información. Los tipos de datos que más utilizaremos en este curso son:

a. Números

- **Enteros (`int`):** Representan números enteros, positivos o negativos, sin decimales.
  Ejemplos: 92, -100, 0, etc.
- **Flotantes (`float`):** Representan números decimales, incluyendo números de punto flotante y notación científica.
  Ejemplos: 3.14; 2.0e3; 23.3333..., etc.

Si queremos pedirle a un usuario que ingrese un número, debemos declarar en nuestro código el tipo de dato que deberá ingresar.

b. Cadenas de caracteres (`str`)
Representan secuencias de caracteres encerradas entre comillas simples (') o dobles (").
Ejemplos: `"Bienvenidos estudiantes"`, `'¡Hola!'`.

c. Listas (`list`)
Son colecciones ordenadas de elementos, que pueden ser de diferentes tipos. Se definen mediante corchetes [ ] y los elementos se separan por comas.
Ejemplo: [1, 2, 'tres', 3.14].

Diferencia entre Lista y Arreglo

La principal diferencia entre una lista y un arreglo es que esta última estructura contiene elementos del **mismo tipo de datos**; en cambio, en una lista está permitido organizar elementos de distinto tipo (`int`, `float`, `bool`, `str`, `list`, etc.)

Estas estructuras tomarán relevancia a partir de la Guía 5, como aplicación para desarrollar el concepto de secuencia y matriz, en donde en este último concepto, se entenderá mayormente la diferencia entre una lista y un arreglo (o `array`).

Término *indexar*

En programación, *indexar* se refiere al acto de acceder a un elemento específico dentro de una estructura de datos. Los índices son números enteros que representan la posición o ubicación relativa de un elemento dentro de la estructura de datos.

Hay que tener en cuenta que las listas, arreglos o cadenas de caracteres, en programación se **indexan** desde 0. Por lo tanto, si queremos conocer el elemento de una lista, arreglo o cadena ubicado en la posición n, debemos ubicar en realidad el elemento $n-1$ en nuestro código. Veamos un ejemplo:

Código ejemplo 1

```
lista = [1, 2, 'tres', 3.14].

# Accediendo al primer elemento, "1"
print(lista[0])

# Accediendo al tercer elemento, "tres"
print(lista[2])
```

0.2 Bucles

Los *bucles* son fundamentales en la programación para automatizar tareas repetitivas y procesar datos de manera eficiente. En Python, un bucle se refiere a una estructura de control que permite repetir un bloque de código varias veces, dependiendo de una condición especificada. Hay dos tipos principales de bucles en Python:

a. Bucle `while`: Este bucle ejecuta repetidamente un bloque de código siempre que una condición dada sea verdadera. La estructura básica es:

Estructura de `while`

```
while condicion:
# código a ejecutar mientras la condición sea verdadera
```

Es importante destacar que en este bucle se deben utilizar **operadores** de comparación y/o lógicos para que se pueda llevar a cabo. Veamos un ejemplo donde a través de un bucle `while`, multiplicaremos los 5 primeros números naturales:

Operadores en condiciones del bucle `while`

`==` Igualdad
`!=` Diferente de
`<` Menor que
`<=` Menor o igual que
`>` Mayor que
`>=` Mayor o igual que
and Devuelve `True` si ambos operandos son `True`
or Devuelve `True` si al menos uno de los operandos es `True`
not Devuelve `True` si el operando es `False`, y viceversa

Código ejemplo 2

```
producto = 1
contador = 1
while contador <= 5:
    producto *= contador
    contador += 1

print(f'El producto de los primeros 5 números naturales
es: {producto}')
```

En el ejemplo anterior:

- Se inicializan las variables "`producto`" y "`contador`" en 1, para desarrollar correctamente la multiplicación, iniciando "`contador`" con el primer número natural.
- El bucle "`while contador <= 5`" se ejecutará mientras "`contador`" sea menor o igual a 5; es decir, para multiplicar los primeros 5 números naturales.
- "`producto *= contador`" multiplicará el valor actual de "`producto`" por el valor actual de "`contador`" y actualiza "`producto`" con el resultado.
- "`contador += 1`" incrementará "`contador`" en 1 en cada iteración para pasar al siguiente número natural.
- Por último, cuando no se cumpla la condición definida en `while`, la consola imprimirá el resultado de `producto`.

b. Bucle `for`: Este bucle itera sobre una secuencia (como una lista, tupla, string, o cualquier objeto iterable) y ejecuta un bloque de código una vez para cada elemento de esa secuencia. La estructura básica es:

Estructura de `for`

```
for elemento in secuencia:
    # código a ejecutar para cada elemento
```

Por ejemplo, un bucle `for` que itera sobre una lista de nombres sería

Código ejemplo 3

```
nombres = ["Agustín", "Germán", "Camila"]
for a in nombres:
print(a)
```

En el ejemplo anterior:

- "a" es la variable que tomará cada elemento de la lista "`nombres`" en cada iteración del bucle, y por otra parte
- en cada iteración, se imprimirá el nombre que contiene la variable "a".

Este bucle será de gran utilidad a partir de la Guía 5, para recorrer secuencias numéricas, las cuales serán organizadas a través de **listas**.

0.3 Condicionales

Los condicionales son estructuras de control que permiten tomar decisiones basadas en la evaluación de expresiones lógicas. Los condicionales más comunes en Python son `if`, `elif` (opcional) y `else`. En ellos, también se deben utilizar operadores de comparación y/o lógicos. Veamos un ejemplo de aplicación de condicionales:

Código ejemplo 4

```
# Solicitamos al usuario que ingrese la nota del estudiante
calificacion = float(input("Ingrese la nota del estudiante: "))

# Evaluamos la nota para determinar si aprueba o reprueba
if calificacion >= 4.0:
    print("El estudiante ha aprobado.")
else:
    print("El estudiante ha reprobado.")
```

Si bien la secuencia de estos condicionales suele ser `if-else`, también es posible sólo evaluar una sola condición (secuencia solo con un `if`), sin declarar nada cuando no

se cumpla; o también es posible evaluar más de una condición utilizando la secuencia **if–elif–else**. Veamos un ejemplo donde se cumple esta última:

Código ejemplo 5

```
# Solicitamos al usuario que ingrese su edad
edad = int(input("Ingrese su edad: "))

# Evaluamos la edad para determinar la categoría
if edad < 18:
    print("Usted es menor de edad.")
elif edad >= 18 and edad < 65:
    print("Usted es adulto.")
else:
    print("Usted es adulto mayor.")
```

0.4 Funciones

En Python, una función es un bloque de código, organizado y reutilizable que se utiliza para realizar una tarea específica. En otras palabras, una función es como una máquina virtual dentro de tu programa que acepta entradas (argumentos), realiza operaciones definidas y puede devolver resultados (retorno).

Para definir una función en Python, utilizaremos la palabra clave **def**, seguida del nombre de la función y un paréntesis (). Si la función acepta parámetros (entradas o argumentos), estos se colocan dentro de los paréntesis. A continuación, mostramos la sintaxis de una función:

Sintaxis de funciones en Python

```
def nombre_de_la_funcion(parametro1, parametro2):
    # o la cantidad de parámetros necesarios

    # Cuerpo de la función que puede contener definición de
    # variables, operaciones, bucles, condicionales, etc.
    return resultado
```

Una vez definida, una función se puede llamar (o invocar) en cualquier parte del programa utilizando su nombre seguido de paréntesis `()`. Las funciones permiten escribir código una vez y reutilizarlo cuantas veces sea necesario sin tener que repetir las mismas líneas de código.

Veamos un ejemplo de aplicación de función en Python:

Código ejemplo 6

```
def potencia(b, n):
    r = b**n
    return r

# Llamada a la función
resultado_potencia=potencia(3,5)
print("El resultado de la potencia es: ",resultado_potencia)
```

Variables definidas en funciones

Es importante destacar que todas las variables definidas dentro de una función son locales a esa función y no accesibles desde fuera de ella.

El código anterior declara una función llamada "`potencia`", la cual calcula y retorna el resultado "`r`" de una potencia de base "`b`" y exponente "`n`". Posteriormente se invoca la función para calcular la potencia 3^5 cuyo resultado es guardado en la variable "`resultado_potencia`" que posteriormente se imprime.

Las funciones en Python se utilizará como recurso, desde la Guía 1, para trabajar el concepto de función matemática, muy útil para optimizar procesos, organizar y gestionar códigos.

0.5 Librerías y Módulos

Una **librería** o biblioteca es un conjunto de *módulos* que proporcionan funcionalidades específicas para realizar tareas diversas; es decir, contiene colecciones de código reutilizable organizado en módulos. Por tanto, un **módulo** es un archivo que contiene definiciones y declaraciones de funciones, clases, variables y/o códigos ejecutables. Es la unidad básica de organización de código en Python.

Existen diversidad de librerías en el mundo Python. Sin embargo, la que utilizaremos frecuentemente en este curso con algunos de sus módulos son `matplotlib`, `numpy` y `scipy`. La primera librería mencionada nos permitirá representar gráficamente funciones matemáticas, la segunda principalmente nos permitirá trabajar operatoria matricial a través de *arreglos* (o `arrays`), y la última librería mencionada nos permitirá poder resolver ecuaciones, no necesariamente lineales y así poder resolver problemas que involucren optimización.

Para importar una librería, debes utilizar la siguiente sintaxis:

Sintaxis para importar librería

```
import nombre_de_la_librería as algún_prefijo
```

Por ejemplo, para importar la librería `numpy` que utilizaremos frecuentemente, se hará de la siguiente forma:

Código ejemplo 7

```
import numpy as np
```

Si bien, la forma anterior de importar una librería es la más utilizada, hay veces que será necesario sólo importar ciertas funciones de una librería y no toda la librería. Por lo tanto, para importar una función específica de una librería, debes hacerlo de la siguiente manera:

Sintaxis para importar una función desde librería

```
from nombre_de_la_librería import nombre_de_la_función
```

Veamos un ejemplo donde se importa una función desde una librería:

Código ejemplo 8

```
from math import e
```

En el ejemplo anterior, se importa el número de Euler a través de la simbología "e" ($e = 2,718281828459045\ldots$), para poder ser utilizado en algún cálculo posterior.

Es importante destacar que a pesar que Python es un lenguaje flexible y poderoso el cual tiene una biblioteca estándar que proporciona muchas funcionalidades básicas, como trabajar con archivos, manejar cadenas, matemáticas básicas, etc; esta biblioteca estándar puede quedar "limitada" con algunas funcionalidades más específicas o avanzadas. Por lo tanto, es crucial instalar librerías y sus dependencias para garantizar que el código funcione correctamente y se ajuste a los requisitos de tu proyecto. Comenta con tu profesor, cómo instalar ciertas librerías para poder llevar a cabo, el desarrollo de la asignatura Matemática Aplicada.

Guía Laboratorio 0 (Descargar)

P1. Escribe un código que pida al usuario ingresar un número real y determine si el número ingresado es positivo, negativo o cero.

P2. Escribe un código que pida al usuario ingresar un número entero y que determine si el número ingresado es par o impar.

P3. Escribe un código que, mediante un ciclo `while`, sume los primeros 100 números naturales.

P4. Escribe un código que almacene en una variable el `string` "Contraseña". Luego, el programa debe solicitar al usuario "Introducir contraseña", hasta que la palabra ingresada sea correcta[1].

P5. Escribe un código que, utilizando un ciclo `for`, pida al usuario ingresar un número entero y muestre la tabla de multiplicar desde el 1 al 12 de dicho número.

P6. *a)* Escribe una función que reciba dos números y retorne el producto de los números recibidos.
b) Escribe una nueva función que tome el producto calculado en el ítem anterior y redondee el valor al entero.
c) Pide al usuario que ingrese dos números, y utiliza ambas funciones para imprimir el valor de la multiplicación redondeado al entero.

P7. La fórmula que permite convertir los grados Celsius a Fahrenheit es la siguiente:

$$F = \frac{9}{5}C + 32\,.$$

a) Crea una función que permita realizar esta conversión.
b) Luego, pide al usuario que ingrese la temperatura en grados Celsius y utiliza la función para calcular su equivalente en grados Fahrenheit.
c) Imprime el resultado redondeado a la centésima.

P8. *a)* Escribe un código que importe el módulo `random` y utiliza la función `randint()` para generar diez números aleatorios entre 1 y 100, y almacénalos en una lista.
b) Imprime en pantalla la lista con los diez números generados.
c) Utiliza la indexación en Python para acceder al segundo y sexto elemento de la lista. Ten en cuenta que en Python los índices comienzan en 0. ¿Cómo imprimirías estos elementos?

P9. El Índice de Masa Corporal (IMC) es una medida que se utiliza para evaluar si una persona se encuentra en un peso saludable con respecto a su estatura. La fórmula para calcular esta medida es:

$$\text{IMC} = \frac{\text{peso (kg)}}{(\text{altura (m)})^2}\,.$$

El valor obtenido con el IMC, de acuerdo con la Organización Mundial de la Salud (OMS) indica si la persona tiene un peso bajo, normal, sobrepeso u obesidad. La clasificación es la siguiente:

- Bajo peso: IMC menor a $18{,}5\,\text{kg}/\text{m}^2$.

[1] String: Tipo de dato que almacena texto.

- Peso normal: IMC entre 18,6 $\mathrm{kg/m^2}$ y 24,9 $\mathrm{kg/m^2}$.
- Sobrepeso: IMC entre 25 $\mathrm{kg/m^2}$ y 29,9 $\mathrm{kg/m^2}$.
- Obesidad: IMC de 30 $\mathrm{kg/m^2}$ o mayor.

Aunque el IMC es una forma eficaz de evaluar el peso, no tiene en cuenta factores como la distribución de grasa, la masa muscular y otros rasgos de la salud física.

a) Considerando la información entregada, implementa un código que permita calcular mediante una función, el valor del IMC de una persona. Luego, solicita al usuario valores de masa y altura e indica, mediante el uso de condicionales, la categoría de la persona según el IMC calculado.

b) Considera la siguiente tabla de valores, que muestra el peso y el IMC de once estudiantes. Guarda los valores del IMC en una lista e imprime en pantalla los estudiantes que, según el criterio de la OMS, están en **bajo peso**. Además, para cada IMC que esté en bajo peso, índica también el índice en la lista donde se encuentra.

N°	Peso (kg)	IMC ($\mathrm{kg/m^2}$)
1	29,5	16,43
2	37,3	19,31
3	38	10,25
4	31	18,63
5	36	17,85
6	40,4	19,76
7	47	23,64
8	43	21,94
9	36	21,3
10	40,1	22,67
11	27	16,48

Problemas de sección 0

P1. Escribe un código que pida al usuario ingresar un número natural y determine si el número ingresado es múltiplo de 2.

P2. Escribe un código que, mediante un ciclo `while`, sume los 50 primeros números impares.

P3. Escribe un código el cual pida al usuario ingresar un *string* de largo mínimo 10, y devuelva otro *string* formado con el primer, el quinto y el décimo carácter del *string* ingreso por el usuario.

P4. Escribe un código que, utilizando un ciclo `for`, pida al usuario ingresar un número entero y muestre los resultados de las potencias de base dicho número entero, y exponentes 1 al 5.

P5. Escribe un código que permita a través de una función, obtener el área de un triángulo de base b y altura h.