GUSTAVO CORONEL
DESARROLLA SOFTWARE

CIENCIA DE DATOS: TALLER DE FUNDAMENTOS DE MACHINE LEARNING CON PYTHON

Módulo 01
PRIMEROS PASOS EN LA CIENCIA DE DATOS CON PYTHON

Dr. Eric Gustavo Coronel Castillo
Docente UNI
gcoronel@uni.edu.pe

ÍNDICE

1 Introducción

Python es reconocido por su sintaxis clara y legible, lo que lo convierte en un lenguaje ideal para principiantes (Python Software Foundation, 2026). En este capítulo, exploraremos los conceptos fundamentales de Python que servirán como base para el análisis de datos. A través de ejemplos prácticos y ejercicios, aprenderás a escribir tus primeros programas y a comprender los elementos esenciales del lenguaje.

El enfoque será eminentemente práctico: aprenderemos haciendo. Se recomienda ejecutar cada ejemplo de código en tu entorno (Google Colab o Jupyter Notebook) para experimentar y ver los resultados inmediatamente (DataCamp, 2022).

2 Python como Calculadora

Python puede funcionar como una calculadora poderosa. El intérprete de Python puede evaluar expresiones aritméticas directamente (Python Software Foundation, 2026).

2.1 Operaciones Aritméticas Básicas

A continuación, tienes ejemplo de Python como calculadora en la consola de Python:

```
>>> x = 17
>>> y = 5
>>> x + y
22
>>> x - y
12
>>> x * y
85
>>> x / y
3.4
>>> x // y
3
>>> x % y
2
>>> x ** y
1419857
```

2.2 Orden de Operaciones

Python sigue el orden estándar de operaciones matemáticas (PEMDAS: Paréntesis, Exponentes, Multiplicación/División, Adición/Substracción) (Python Software Foundation, 2026):

```
>>> 2 + 3 * 4
14
>>> (2 + 3) * 4
20
>>> 3 * 3 ** 2 / 9
3.0
```

3 Variables y Tipos de Datos

Las variables son contenedores que almacenan valores. En Python, no necesitamos declarar el tipo de una variable explícitamente; el tipo se infiere automáticamente del valor asignado (Python Software Foundation, 2026).

3.1 Asignación de Variables

```
>>> x = 5
>>> y = 10
>>> z = x + y
>>> print("z =", z)
z = 15
```

3.2 Reglas para nombres de variables

- Deben comenzar con una letra (a-z, A-Z) o guión bajo (_).
- Pueden contener letras, números y guiones bajos.
- Son sensibles a mayúsculas y minúsculas (edad, Edad y EDAD son variables diferentes).
- No pueden ser palabras reservadas de Python (if, for, while, etc.).

3.3 Tipos de Datos Fundamentales

Tipo	Nombre en Python	Ejemplo
Entero	int	edad = 25
Decimal	float	altura = 1.75
Texto	str	nombre = 'Ana'
Booleano	bool	activo = True

Para verificar el tipo de una variable, usamos la función `type()`:

```
edad = 25
print(type(edad))  # <class 'int'>
altura = 1.75
print(type(altura))  # <class 'float'>
nombre = 'Ana'
print(type(nombre))  # <class 'str'>
```

4 Cadenas de Texto (Strings)

Las cadenas son secuencias de caracteres Unicode. En Python, se pueden definir con comillas simples (') o dobles (") (Python Software Foundation, 2026).

4.1 Creación y Concatenación

```
# Diferentes formas de definir strings
saludo1 = 'Hola'
saludo2 = "Mundo"
# Concatenación (unir strings)
mensaje = saludo1 + ' ' + saludo2
print(mensaje)  # Hola Mundo
```

4.2 Métodos útiles de Strings

```
texto = 'ciencia de datos'
# Convertir a mayúsculas
print(texto.upper())  # CIENCIA DE DATOS
# Convertir a minúsculas
print(texto.lower())  # ciencia de datos
# Capitalizar primera letra
print(texto.capitalize())  # Ciencia de datos
# Contar ocurrencias de un carácter
print(texto.count('a'))  # 2
# Reemplazar texto
print(texto.replace('datos', 'información'))  # ciencia de información
```

4.3 Formateo de Strings (f-strings)

Los f-strings (formatted string literals) son la forma moderna y recomendada de formatear strings en Python 3.6+ (Python Software Foundation, 2026):

```
nombre = 'Carlos'
edad = 30
altura = 1.80
# f-string permite insertar variables directamente
mensaje = f'Me llamo {nombre}, tengo {edad} años y mido {altura}m'
print(mensaje) # Me llamo Carlos, tengo 30 años y mido 1.8m
```

5 Listas

Las listas son colecciones ordenadas y modificables que pueden contener elementos de cualquier tipo. Son una de las estructuras de datos más versátiles en Python (Python Software Foundation, 2026).

5.1 Creación y Acceso

```
# Crear una lista
temperaturas = [23.5, 25.0, 22.8, 24.1, 26.3]
# Acceder a elementos (índice comienza en 0)
print(temperaturas[0]) # 23.5 (primer elemento)
print(temperaturas[-1])  # 26.3 (último elemento)
# Obtener sublista (slicing)
print(temperaturas[1:3])  # [25.0, 22.8]
```

5.2 Operaciones con Listas

```
# Dedinición de una lista
datos = [10, 20, 30]
# Agregar elemento al final
datos.append(40)
print(datos)  # [10, 20, 30, 40]
# Insertar en posición específica
datos.insert(0, 5)
print(datos)  # [5, 10, 20, 30, 40]
# Eliminar elemento
datos.remove(20)
print(datos)  # [5, 10, 30, 40]
# Longitud de la lista
print(len(datos))  # 4
```

6 Diccionarios

Los diccionarios almacenan pares clave-valor. Son extremadamente útiles para organizar datos relacionados (Python Software Foundation, 2026).

```
# Crear un diccionario
estudiante = {
     'nombre': 'María',
     'edad': 22,
     'carrera': 'Ingeniería de Sistemas',
     'promedio': 16.5
}
# Acceder a valores
print(estudiante['nombre'])  # María
# Agregar nueva clave-valor
estudiante['universidad'] = 'UNI'
# Obtener todas las claves
print(estudiante.keys())
# Obtener todos los valores
print(estudiante.values())
```

7 Estructuras de Control

7.1 Condicionales (if, elif, else)

Las estructuras condicionales permiten que el programa tome decisiones basándose en condiciones (Python Software Foundation, 2026):

```
# Fija una temperatura
temperatura = 28
# Imprime: El clima es agradable
if temperatura > 30:
     print('Hace mucho calor')
elif temperatura > 20:
     print('El clima es agradable')
else:
     print('Hace frío')
```

7.2 Bucle For

El bucle `for` se utiliza para iterar sobre secuencias (Python Software Foundation, 2026):

```
# Iterar sobre una lista
ciudades = ['Lima', 'Cusco', 'Arequipa', 'Trujillo']
for ciudad in ciudades:
     print(f'Ciudad: {ciudad}')

# Iterar con índice
# Imprime: Número: 0, 1, 2, 3, 4
for i in range(5):
     print(f'Indice: {i}')
```

7.3 Bucle While

El bucle `while` se ejecuta mientras una condición sea verdadera:

```
# ija el valor de contador
contador = 0
# Imprime: Contador: 0, 1, 2
while contador < 3:
     print(f'Contador: {contador}')
     contador += 1
```

8 Funciones

Las funciones son bloques de código reutilizables que realizan una tarea específica. Ayudan a organizar el código y evitar repetición (Python Software Foundation, 2026).

8.1 Definir y Llamar Funciones

```
# Definir una función
def saludar(nombre):
     return f'Hola, {nombre}!'

# Llamar la función
mensaje = saludar('Ana')
print(mensaje)  # Hola, Ana!
```

8.2 Funciones con Múltiples Parámetros

```
# Definir la función
def calcular_promedio(nota1, nota2, nota3):
     suma = nota1 + nota2 + nota3
     promedio = suma / 3
     return promedio

# Llamar la función
resultado = calcular_promedio(15, 17, 16)
print(f'Promedio: {resultado}')  # Promedio: 16.0
```

8.3 Parámetros con Valores por Defecto

```
# Definir la función
def presentar(nombre, edad=18):
     return f'{nombre} tiene {edad} años'

# Llamadar la función
print(presentar('Luis'))          # Luis tiene 18 años
print(presentar('María', 25))     # María tiene 25 años
```

9 Introducción a las Librerías

Una de las mayores fortalezas de Python es su ecosistema de librerías. Las librerías son colecciones de código escrito por otros programadores que podemos usar en nuestros proyectos (VanderPlas, 2016).

9.1 Importar Librerías

Caso 1: Importar librería completa

```
import math
# Utilizando la librería
print(math.pi)  # 3.141592653589793
print(math.sqrt(16))  # 4.0
```

Caso 2: Importar con alias

```
import numpy as np
# Crear un array de NumPy (usando el alias 'np')
temperaturas = np.array([22.5, 24.0, 23.8, 25.2, 21.9])
print("Temperaturas:", temperaturas)
print("Promedio:", np.mean(temperaturas))
```

Caso 3: Importar con alias

```
import pandas as pd
# Crear un DataFrame de Pandas (usando el alias 'pd')
datos = pd.DataFrame({
    'ciudad': ['Lima', 'Cusco', 'Arequipa'],
    'temperatura': [24.5, 18.2, 20.8],
    'humedad': [75, 45, 50]
})
# Imprimiendo el DataFrame
print("\nDataFrame:")
print(datos)
```

Caso 4: Importar funciones específicas

```
from math import sqrt, pi
print(sqrt(25))  # 5.0
print("PI:", pi)  # 3.141592653589793
```

9.2 Primer Ejemplo con NumPy

NumPy es la librería fundamental para computación numérica en Python (Harris et al., 2020):

```
import numpy as np
# Crear un array (arreglo)
temperaturas = np.array([23.5, 25.0, 22.8, 24.1, 26.3])
# Operaciones vectorizadas
print(f'Promedio: {temperaturas.mean()}')
print(f'Máximo: {temperaturas.max()}')
print(f'Mínimo: {temperaturas.min()}')
print(f'Desviación estándar: {temperaturas.std()}')
```

10 Primer Análisis de Datos Simple

Ahora que conocemos los fundamentos, realizaremos un análisis simple que integra los conceptos aprendidos:

```
# Datos de ventas de una semana
ventas = {
     'Lunes': 1500,
     'Martes': 1800,
     'Miércoles': 1200,
     'Jueves': 2100,
     'Viernes': 2500,
     'Sábado': 3000,
     'Domingo': 2200
}
# Análisis básico
total_semana = sum(ventas.values())
promedio_dia = total_semana / len(ventas)
mejor_dia = max(ventas, key=ventas.get)
# Reporte
print("REPORTE")
print(f'Total de ventas: S/. {total_semana}')
print(f'Promedio diario: S/. {promedio_dia:.2f}')
print(f'Mejor día: {mejor_dia} con S/. {ventas[mejor_dia]}')
# Encontrar días sobre el promedio
print('\nDías sobre el promedio:')
for dia, venta in ventas.items():
     if venta > promedio_dia:
         print(f'{dia}: S/. {venta}')
```

11 Mejores Prácticas y Consejos

- **Usar nombres descriptivos:** Preferir temperatura_maxima sobre tm.
- **Comentar el código:** Usar # para explicar la lógica compleja.
- **Seguir PEP 8:** La guía de estilo oficial de Python (van Rossum et al., 2001).
- **Experimentar:** La mejor forma de aprender es ejecutar código y ver qué pasa.
- **Leer documentación:** La documentación oficial de Python es excelente y muy completa (Python Software Foundation, 2026).

12 Ejercicios Propuestos

Para consolidar lo aprendido, se recomienda resolver los siguientes ejercicios:

1. Crear una función que reciba una lista de números y devuelva el promedio, el máximo y el mínimo.
2. Crear un diccionario con información de 5 estudiantes (nombre, edad, nota) y calcular el promedio de notas.
3. Escribir una función que convierta temperatura de Celsius a Fahrenheit y viceversa.
4. Crear una lista con los días de la semana y usar un bucle `for` para imprimir solo los días laborables.
5. Usar `NumPy` para crear un array de 10 números aleatorios entre 0 y 100, y calcular su estadística descriptiva básica.

13 Conclusiones

En esta lección hemos cubierto los fundamentos esenciales de Python que todo científico de datos debe conocer. Desde operaciones aritméticas básicas hasta estructuras de datos complejas como listas y diccionarios, pasando por estructuras de control y funciones, estos conceptos forman la base sobre la cual construiremos nuestras habilidades de análisis de datos.

La sintaxis clara y legible de Python, combinada con su poderoso ecosistema de librerías, lo convierte en la herramienta ideal para principiantes y profesionales por igual. Los ejercicios prácticos realizados demuestran cómo estos conceptos fundamentales se aplican a problemas reales de análisis de datos.

En los próximos módulos, profundizaremos en las librerías especializadas de ciencia de datos (NumPy, Pandas, Matplotlib) que nos permitirán realizar análisis más sofisticados y visualizaciones impactantes. El dominio de estos fundamentos facilitará significativamente ese aprendizaje.

14 Referencias

DataCamp. (2025). *Introduction to data science in Python*. DataCamp. https://www.datacamp.com/courses/introduction-to-data-science-in-python

Harris, C. R., Millman, K. J., Van der Walt, S. J., Gommers, R., Virtanen, P., & Cournapeau, D. (2020). Array programming with NumPy. *Nature, 585*(7825), 357-362. https://doi.org/10.1038/s41586-020-2649-2

Python Software Foundation. (2026). *The Python Tutorial*. https://docs.python.org/3.13/tutorial/

Van Rossum, G., Warsaw, B., & Coghlan, A. (2001). *PEP 8 – Style guide for Python code*. Python Enhancement Proposals. https://peps.python.org/pep-0008/

VanderPlas, J. (2016). *Python data science handbook: Essential tools for working with data.* O'Reilly Media. https://doi.org/https://jakevdp.github.io/PythonDataScienceHandbook/

15 Anexo: Ejemplos Aplicados por Disiplina

15.1 Ejemplo para Economistas: Cálculo de Inflación

```
# Precios de la canasta básica (en soles)
precios_2024 = {'arroz': 3.50, 'pan': 0.50, 'leche': 4.20, 'pollo': 8.50}
precios_2025 = {'arroz': 3.80, 'pan': 0.60, 'leche': 4.50, 'pollo': 9.20}
# Calcular inflación por producto
print("Inflación por producto:")
for producto in precios_2024:
     precio_anterior = precios_2024[producto]
     precio_actual = precios_2025[producto]
     inflacion = (precio_actual - precio_anterior)
     inflacion = (inflacion / precio_anterior) * 100
     print(f"{producto}: {inflacion:.2f}%")
# Inflación promedio
precios_ant = list(precios_2024.values())
precios_act = list(precios_2025.values())
inflacion_promedio = ((sum(precios_act) - sum(precios_ant)) /
                        sum(precios_ant)) * 100
print(f”\nInflación promedio: { nflación_promedio:.2f}%”)
```

Resultado:

```
Inflación por producto:
arroz: 8.57%
pan: 20.00%
leche: 7.14%
pollo: 8.24%

Inflación promedio: 8.38%
```

15.2 Ejemplo para Ingenieros Civiles: Análisis de Resistencia de Materiales

```
import numpy as np
# Resistencias de probetas de concreto (kg/cm²)
resistencias = np.array([210, 215, 205, 220, 218, 212, 208, 225, 213, 217])
# Análisis estadístico
print(f"Resistencia promedio: {resistencias.mean():.2f} kg/cm²")
print(f"Resistencia mínima: {resistencias.min()} kg/cm²")
print(f"Resistencia máxima: {resistencias.max()} kg/cm²")
print(f"Desviación estándar: {resistencias.std():.2f} kg/cm²")
# Verificar especificación (f'c = 210 kg/cm²)
especificacion = 210
probetas_ok = resistencias[resistencias >= especificacion]
porcentaje_ok = (len(probetas_ok) / len(resistencias)) * 100
print(f"\nProbetas que cumplen: {porcentaje_ok:.1f}%")
```

Resultado:

```
Resistencia promedio: 214.30 kg/cm²
Resistencia mínima: 205 kg/cm²
Resistencia máxima: 225 kg/cm²
Desviación estándar: 5.66 kg/cm²

Probetas que cumplen: 80.0%
```

15.3 Ejemplo para Psicólogos: Análisis de Encuesta de Ansiedad

```
import pandas as pd
# Respuestas escala de ansiedad
# (1=Nunca, 2=Rara vez, 3=A veces, 4=Frecuentemente, 5=Siempre)
datos_encuesta = pd.DataFrame({
     'participante': ['P1', 'P2', 'P3', 'P4', 'P5'],
     'item1': [2, 4, 3, 5, 2],  # Nerviosismo
     'item2': [3, 5, 4, 4, 3],  # Preocupación
     'item3': [1, 3, 2, 5, 2],  # Tensión física
     'item4': [2, 4, 3, 4, 1]   # Dificultad para relajarse
})
# Calcular puntaje total por participante, nueva columna
datos_encuesta['puntaje_total'] = datos_encuesta[
       ['item1', 'item2', 'item3', 'item4']].sum(axis=1)

# Función para clasificar el nivel de ansiedad
def clasificar_ansiedad(puntaje):
     if puntaje <= 8:
         return 'Baja'
     elif puntaje <= 12:
         return 'Media'
     else:
         return 'Alta'

# Obtener el nivel por participante – Estilo Pandas
# datos_encuesta['nivel'] = datos_encuesta['puntaje_total'] \
#                           .apply(clasificar_ansiedad)

# Obtener el nivel por participante – Estilo Pythonic
niveles = []
for puntaje in datos_encuesta['puntaje_total']:
    nivel = clasificar_ansiedad(puntaje)
    niveles.append(nivel)
datos_encuesta['nivel'] = niveles

# Reporte
print(datos_encuesta[['participante', 'puntaje_total', 'nivel']])
print(f"\nPromedio de ansiedad: {datos_encuesta['puntaje_total'].mean():.2f}")
```

Resultado:

```
  participante  puntaje_total  nivel
0           P1              8   Baja
1           P2             16   Alta
2           P3             12  Media
3           P4             18   Alta
4           P5              8   Baja

Promedio de ansiedad: 12.40
```

15.4 Ejemplo para Ingenieros Industriales: Optimización de Inventario

```
# Modelo EOQ (Cantidad Económica de Pedido)
import math
# Parámetros
demanda_anual = 12000  # unidades
costo_pedido = 50      # soles por pedido
costo_almacen = 2      # soles por unidad al año
# Calcular EOQ
eoq = math.sqrt((2 * demanda_anual * costo_pedido) / costo_almacen)
# Calcular número óptimo de pedidos
num_pedidos = demanda_anual / eoq
# Costo total anual
costo_pedidos = (demanda_anual / eoq) * costo_pedido
costo_almacenamiento = (eoq / 2) * costo_almacen
costo_total = costo_pedidos + costo_almacenamiento
print(f"Cantidad económica de pedido: {eoq:.0f} unidades")
print(f"Número óptimo de pedidos al año: {num_pedidos:.0f}")
print(f"Costo total anual: S/. {costo_total:.2f}")
```

Resultado:

```
Cantidad económica de pedido: 775 unidades
Número óptimo de pedidos al año: 15
Costo total anual: S/. 1549.19
```

15.5 Ejemplo para Estadísticos: Prueba de Hipótesis Simple

```
import numpy as np

# Muestra de alturas (cm) de estudiantes UNI
muestras = np.array([168, 172, 165, 170, 175, 169, 171, 173, 167, 174])

# Parámetros
media_muestral = muestras.mean()
desv_muestral = muestras.std(ddof=1)  # ddof=1 para muestra
n = len(muestras)

# Prueba de hipótesis: H0: μ = 170 vs H1: μ ≠ 170
media_hipotetica = 170
error_estandar = desv_muestral / np.sqrt(n)
t_estadistico = (media_muestral - media_hipotetica) / error_estandar

# Reporte
print(f"Media muestral: {media_muestral:.2f} cm")
print(f"Desviación estándar: {desv_muestral:.2f} cm")
print(f"Error estándar: {error_estandar:.2f}")
print(f"Estadístico t: {t_estadistico:.4f}")
print(f"\nInterpretación: {'Rechazar H0' if \
       abs(t_estadistico) > 2.262 else 'No rechazar H0'}")
print("(valor crítico t(9, 0.05) = 2.262)")
```

Resultado:

```
Media muestral: 170.40 cm
Desviación estándar: 3.20 cm
Error estándar: 1.01
Estadístico t: 0.3948

Interpretación: No rechazar H0
(valor crítico t(9, 0.05) = 2.262)
```