



Comité Nacional Peruano de la CIER

Curso Virtual

**Python para el Análisis de Datos y la Automatización
en el Sector Eléctrico**

27, 29 y 31 de Octubre, 03, 05, 07, 10 y 12 de Noviembre 2025.

MARVIN COTO – FACILITADOR

E-mail: mcotoj@gmail.com



Comité Nacional Peruano de la CIER

SESIÓN 1

Parte 1

¿Por qué aprender un lenguaje de programación?

La programación es una herramienta esencial para optimizar procesos, analizar datos y automatizar tareas. Con un lenguaje de programación podemos:

- **Automatizar tareas repetitivas:** Reducir el tiempo dedicado a cálculos manuales o generación de informes. Ejemplo: Extraer información de documentos, bases de datos.



¿Por qué aprender un lenguaje de programación?

La programación es una herramienta esencial para optimizar procesos, analizar datos y automatizar tareas. Con un lenguaje de programación podemos:

- **Analizar datos de manera eficiente:** Procesar grandes volúmenes de datos, como mediciones de tensión, corriente o consumo energético. ¡Más allá de las posibilidades de hojas de cálculo!
- **Desarrollar herramientas personalizadas:** Crear programas adaptados a necesidades específicas, como simulaciones de circuitos, sistemas de monitoreo o algoritmos de inteligencia artificial.



¿Por qué aprender un lenguaje de programación?

- Todo lo que una computadora hace, lo realiza porque alguien la programó para hacerlo.
- ¿Qué cosas nos gustaría que nuestra computadora hiciera por nosotros?

Caso: Transcribir y resumir una lección, y generar un cuestionario de seguimiento de forma automática, todo en tiempo real durante el desarrollo de la lección. ¡Y gratis!



¿Por qué aprender un lenguaje de programación?

- Todo lo que una computadora hace, lo realiza porque alguien la programó para hacerlo.
- ¿Qué cosas nos gustaría que nuestra computadora hiciera por nosotros?

Caso: Realizar un estudio de capacidad de alojamiento (*hosting capacity*) sin modelado de la red, solamente a partir de los datos con algoritmos de inteligencia artificial. ¡Y gratis!



¿Qué hay del *vibe coding*?

Generar programas sin conocimientos de código.

Vibe Coding vs Real Coding

	VIBE CODING	REAL CODING
Method	Guesswork, AI prompts, or copying code	Logic, planning, understanding
Speed	Quick to start, slow/little long-term use	Slow to start, faster and reliable long-term use
Understanding	Often unclear	Clear and deliberate
Usefulness	Good for prototyping or experimenting	Best for complex and maintainable software
Risk	Higher risk: bugs, security issues, little scalability	Lower risk: more stable, easier to debug and build on



Introducción a la Filosofía de Python



- Se presentarán los principios que guían el desarrollo de Python y cómo gracias a sus características puede apoyar la mejorar en la eficiencia en tareas de análisis y automatización en el sector.

Lenguaje es una analogía útil en términos del aprendizaje y algunos de los elementos

Brevísima historia

- Python fue creado por Guido van Rossum (1956, Países Bajos)
- Lanzado por primera vez el 20 de febrero de 1991.
- El nombre del lenguaje proviene de la serie de comedia británica "Monty Python's Flying Circus".
- Python es en gran parte el resultado del trabajo de este autor, aunque con contribuciones significativas de una comunidad mundial de desarrolladores.



Características clave

- **Es un Lenguaje de propósito general:** Es usado en muchos dominios de aplicación, desde desarrollo web hasta inteligencia artificial.
- **Tiene sintaxis clara y legible:** Está diseñado para ser intuitivo y fácil de aprender.
- **Es multiplataforma y de código abierto:** Está disponible de forma gratuita para todos los sistemas operativos.
- **Gran ecosistema de bibliotecas:** Tiene herramientas para casi cualquier tarea imaginable.



Objetivos originales

- Ser un lenguaje fácil e intuitivo, pero igual de poderoso que sus competidores. (C, C++, Java, Perl, etc.)
- Ser de código abierto, permitiendo contribuciones comunitarias.
- **Tener un código casi tan legible como el inglés.**
- Ser adecuado para tareas cotidianas, con tiempos de desarrollo cortos.



Impacto y popularidad

Hoy, Python es uno de los lenguajes más populares del mundo, utilizado ampliamente en:

- Desarrollo web (Django, Flask)
- Ciencia de datos (SciPy, Pandas)
- Inteligencia artificial (PyTorch)
- Automatización y scripting



Python es mantenido por la Python Software Foundation, una organización sin fines de lucro dedicada al avance y promoción del lenguaje.

URL: <https://www.python.org/psf-landing/>

Filosofía de Python

- Python es un lenguaje de programación basado en una serie de principios que promueven la simplicidad, la legibilidad y la eficiencia.
- Estos principios están resumidos en el "Zen de Python", el cual se puede visualizar ejecutando el siguiente comando en un intérprete de Python:

```
import this
```

Podemos probarlo en:

<https://www.online-python.com/>

La sintaxis

- Python enfatiza la legibilidad del código utilizando indentación en lugar de llaves {} (como muchos otros lenguajes de programación) para definir bloques de código.
- El primer ejemplo de código de todo lenguaje de programación es el "¡Hola Mundo!"
- Se implementa para verificar que la computadora "comprende" el código y realiza un resultado.
- En Python esto se hace simplemente con `print("¡Hola Mundo!")`

Podemos probarlo en:

<https://www.online-python.com/>

La sintaxis

- Veamos este otro ejemplo:
- # Cálculo de la resistencia de un conductor según la Ley de Ohm
- tension = 220 # en voltios
- corriente = 10 # en amperios
-
- resistencia = tension / corriente # Ley de Ohm: $R = V / I$
- print(f"La resistencia es {resistencia} ohmios")

Note el uso de #

Podemos probarlo en:

<https://www.online-python.com/>

- Salida esperada:
La resistencia es 22.0 ohmios

La sintaxis

- La "f" significa que se está utilizando una cadena formateada (f-string). Otras posibilidades:

```
print("La resistencia es", resistencia, "ohmios")
```

```
print("El tiempo de operación es {}  
segundos".format(tiempo))
```

Podemos probarlo en:

<https://www.online-python.com/>

print es una función de Python que muestra mensajes en pantalla al usuario.

¿Está esto implementado en software que utilizamos?

La sintaxis

- Vemos que ésta forma de realizar cálculos es muy conveniente, pues cuenta con claridad una forma directa de realizarlos.
- Versión en Lenguaje de Programación C:

```
#include <stdio.h>

int main() {
    // Cálculo de la resistencia de un conductor según La Ley de Ohm
    float voltaje = 220.0; // en voltios
    float corriente = 10.0; // en amperios

    float resistencia = voltaje / corriente; // Ley de Ohm:  $R = V / I$ 
    printf("La resistencia es %.1f ohmios\n", resistencia);

    return 0;
}
```

Uso de espacios y tabulaciones

Python requiere el uso de espacios para definir bloques de código. Esto reduce errores y hace que los programas sean más fáciles de entender.

Ejemplo incorrecto:

```
voltaje=120
if voltaje > 0:
print("El voltaje es positivo") # Esto generará un error de indentación
```

Corrección:

```
voltaje=120
if voltaje > 0:
    print("El voltaje es positivo") # Correcto, con indentación adecuada.
```

El estándar de Python sugiere cuatro espacios para la indentación, y no utilizar tabulador.

Tipos de datos dinámicos

Las variables son propias de cualquier sistemas: parámetros que pueden tomar distintos valores. Python permite definir variables sin especificar su tipo, lo cual facilita la manipulación de datos en ingeniería eléctrica.

```
energia = 500.5 # Joules  
potencia = 100 # Watts  
  
tiempo = energia / potencia # t = E / P  
print(f"El tiempo de operación es {tiempo} segundos")
```

Salida esperada:

El tiempo de operación es 5.005 segundos

Esto permite a los programadores concentrarse en los cálculos en lugar de preocuparse por la declaración de tipos de datos.

Al finalizar esta introducción, sabemos que:

- Python es un excelente lenguaje de programación debido a su claridad, flexibilidad y capacidad de automatización.
- Siguiendo su filosofía, podemos escribir código eficiente y fácil de mantener para resolver problemas complejos.
- Si no se cuenta con experiencia en programación: Excelente opción para aprender un lenguaje. Posiblemente no se requiera aprender otro más.
- Si se cuenta con experiencia: Se pueden aprovechar sus ventajas por bibliotecas de funciones y características de alto nivel.

¿Qué ganamos haciendo nuestros propios programas?

Crear nuestros propios programas nos da control total sobre las herramientas que utilizamos. En lugar de depender de software comercial, podemos:

- **Adaptar los programas a nuestras necesidades:** Por ejemplo, crear un script que lea datos de sensores, logs de un SCADA y genere alertas en tiempo real.
- **Ahorrar costos:** Evitar la compra de licencias de software costoso.
- **Mejorar la eficiencia:** Automatizar tareas que consumen mucho tiempo, como la generación de informes o el análisis de datos.



Comité Nacional Peruano de la CIER

Ejemplos y ejercicios



colab.research.google.com

PECIER_dia1_parte1



Comité Nacional Peruano de la CIER

SESIÓN 1

Parte 2

Instalación de Python



Esta sección explica cómo instalar Python y configurar entornos de desarrollo adecuados para programación en el sector eléctrico. Las herramientas sugeridas son Jupyter Notebook y Spyder.



Descarga e instalar Python

- **Descargar Python:**
 - Visita python.org y descarga la última versión para tu sistema operativo. Esto permitirá utilizarlo en línea de comando (Terminal, Powershell).
- **Ejecutar el instalador:**
 - Asegúrate de marcar “**Add Python to PATH**” antes de hacer clic en “Install Now”.
- **Verificar la instalación:**
 - Abre una terminal y ejecuta:
`python --version`
 - Deberías ver algo como: Python 3.x.x.

Jupyter Notebook



Jupyter es útil para documentación y ejecución interactiva de código.

1. Instalar Jupyter:

```
pip install notebook
```

pip: manejador de paquetes
(extensiones) de Python

2. Ejecutar Jupyter:

```
jupyter notebook
```

Visual Studio Code (VS Code)

- VS Code es ideal para proyectos más estructurados.
 - **Descargar VS Code:** code.visualstudio.com
 - **Instalar extensión de Python:**
 - Ir a Extensiones (Ctrl+Shift+X)
 - Buscar Python y hacer clic en Instalar

Miniconda & Spyder

- Miniconda es un ecosistema de desarrollo, dentro del cual se encuentra Spyder, un intérprete de Python que facilita la edición y verificación de código.
- Descarga:** anaconda.com/download/

Ejemplo (para probar nuestros sistemas)

```
import matplotlib.pyplot as plt

# Datos de consumo por día
dias = ["Lunes", "Martes", "Miércoles",
"Jueves", "Viernes"]
consumo = [120, 150, 130, 160, 140]

# Generación del gráfico
plt.plot(dias, consumo, marker='o', linestyle='-
', color='b')
plt.title("Consumo Energético Semanal")
plt.xlabel("Día")
plt.ylabel("Consumo (kWh)")
plt.grid(True)
plt.show()
```

En resumen:

Para la siguiente sesión, idealmente debemos contar con:

- Instalación de Python en nuestras computadoras (version 3.9 o superior)
- Cuenta de Google para usar Google Colab.
- Spyder.

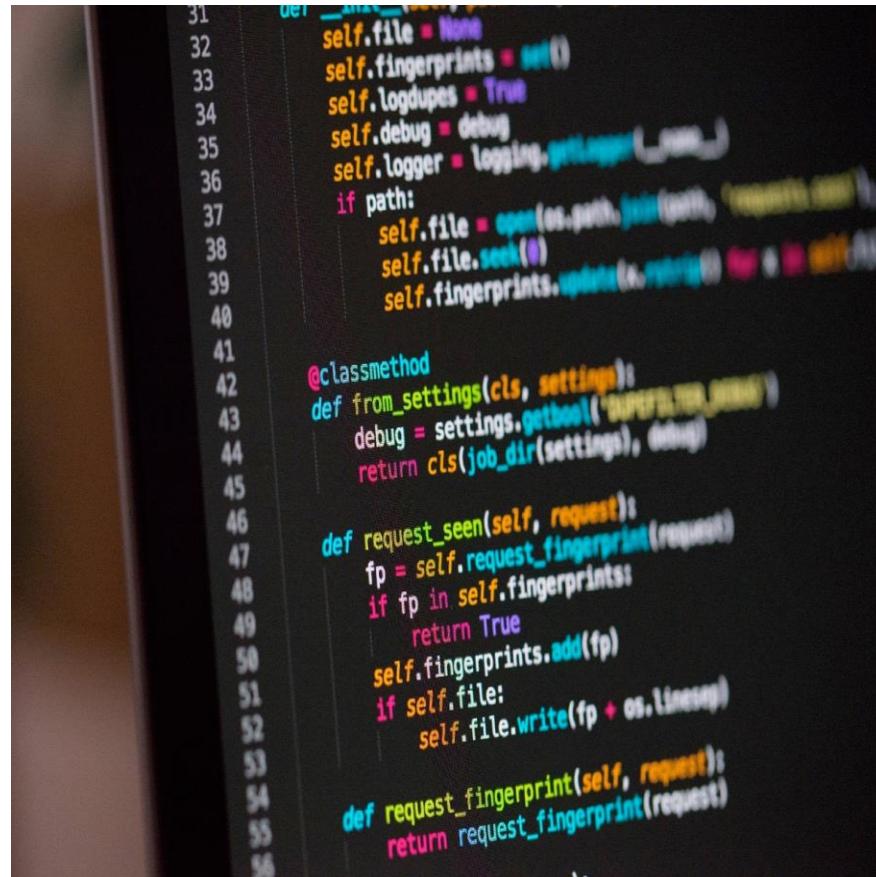


Comité Nacional Peruano de la CIER

SESIÓN 1 Parte 3

Sintaxis: Tipos de datos y operadores

Esta sección introduce la sintaxis básica de Python, la cual se refiere a la forma de introducir variables, tipos de datos y operadores matemáticos y lógicos de forma correcta para que sean interpretadas por la computadora



```
31     def __init__(self, path=None):
32         self.file = None
33         self.fingerprints = set()
34         self.logduplicates = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, 'fingerprint.log'), 'w')
39             self.file.seek(0)
40             self.fingerprints.update(fp for fp in self.file)
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool('FINGERPRINT_DEBUG')
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)
```

Introducción a las variables

Una **variable** es un contenedor que almacena datos en la memoria del computador, y que está sujetos a modificaciones.

Ejemplo: Definir Variables en un Circuito

```
# Variables de un circuito eléctrico
voltaje = 220 # Voltios
corriente = 5 # Amperios

# Imprimir valores
print(f"Voltaje: {voltaje} V")
print(f"Corriente: {corriente} A")

#O bien
print("Voltaje:", voltaje, "V")
print("Corriente:", corriente, "A")
```

Introducción a las variables



Importante: Todos los lenguajes de programación contemplan distintos tipos de variables, como cadenas de caracteres (texto) o formatos numéricos. Esto es importante porque las opciones que se pueden realizar sobre las variables dependen de su tipo.

Por ejemplo, sobre variables numéricos podemos realizar operaciones matemáticas, pero no sobre texto. Pero sobre texto podemos realizar concatenación.

Tipos de datos

Python tiene varios tipos de datos básicos:

- **int**: Números enteros (220)
- **float**: Números con decimales (3.14)
- **str**: Cadenas de texto ("Transformador")
- **bool**: Valores lógicos (True, False)

```
resistencia = 50.5          # Ohmios (float)
nombre = "Motor"            # Nombre del componente (str)
estado = True                # ¿Está encendido? (bool)

# Imprimir tipos
print(type(resistencia))
print(type(nombre))
print(type(estado))
```

Operadores matemáticos

Python permite realizar cálculos con operadores aritméticos.

Operador	Descripción	Ejemplo
+	Suma	$5 + 3 = 8$
-	Resta	$5 - 3 = 2$
*	Multiplicación	$5 * 3 = 15$
/	División	$5 / 2 = 2.5$
%	Módulo	$5 \% 2 = 1$
**	Exponente	$2 ** 3 = 8$

Ejemplo

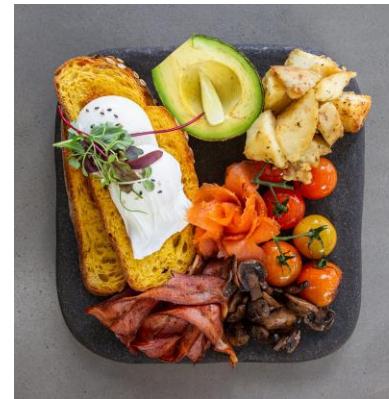
```
voltaje = 220
corriente = 5
potencia = voltaje * corriente # Ley de Watt: P = V * I
print("Potencia:", potencia, "W")
```

Estructuras de control

Resolver problemas:



Repetir acciones



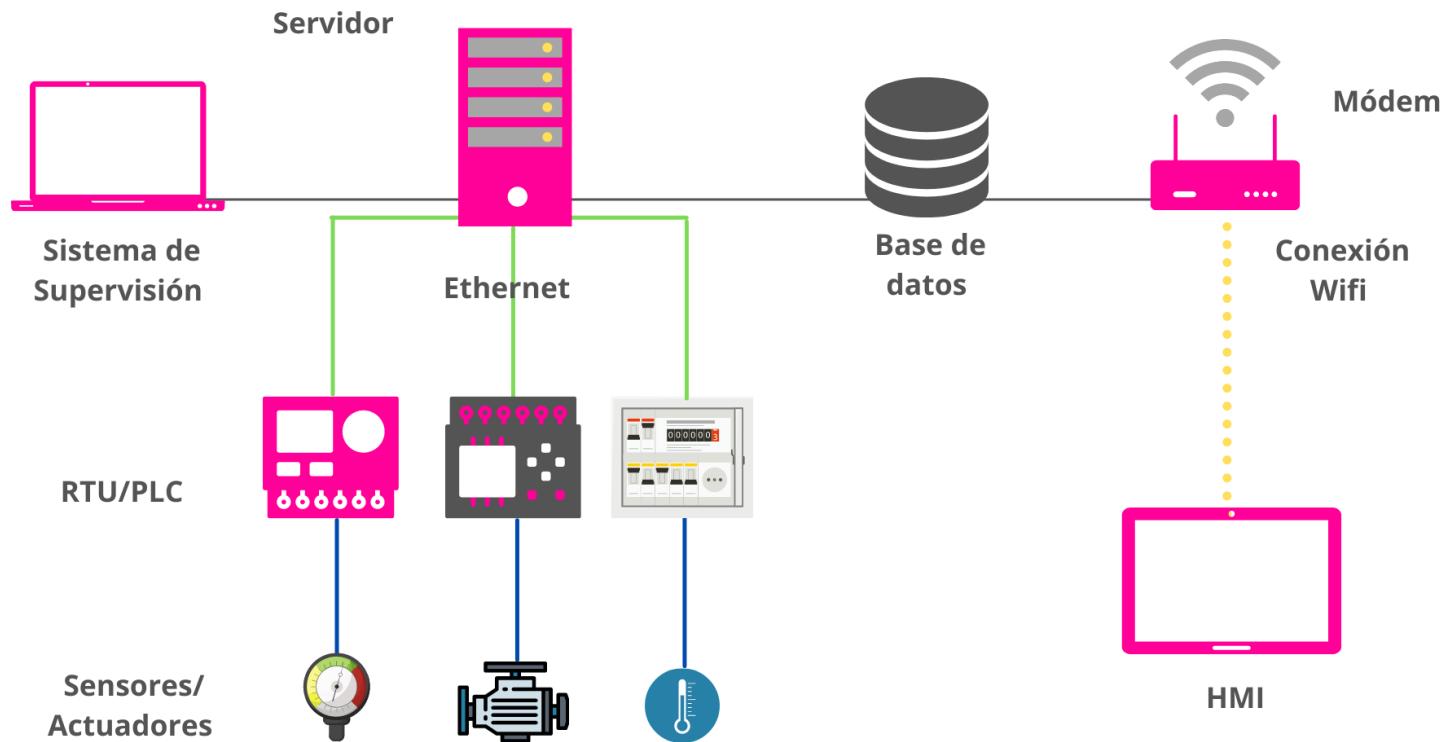
Realizar según condición



shutterstock.com · 594424715

Estructuras de control

Resolver problemas:



Estructuras de control



Resolver problemas:

Estudiaremos cómo controlar el flujo de ejecución en Python usando condicionales (if, else, elif) y bucles (for, while). Aplicaremos estos conceptos en el análisis de sistemas eléctricos y monitoreo de datos.

Estructuras de control

Ejemplo:

```
temperatura = 85
```

```
corriente = 12
```

```
if temperatura > 80 and corriente > 10:  
    print(" Advertencia: Riesgo de sobrecarga.")  
else:  
    print(" Circuito en condiciones seguras.")
```

En este caso, los valores de temperatura y corriente (o tensión), fueron definidas dentro del programa. Es usual que los valores sean recibidos de una fuente externa, por ejemplo: Son introducidos por el usuario en un formulario, o se reciben de una base de datos.

Estructuras de control

Ejemplo: Calculadora de resistencia

```
voltaje = float(input("Ingrese el voltaje (V): "))  
corriente = float(input("Ingrese la corriente (A): "))  
  
# Calcular resistencia  
resistencia = voltaje / corriente  
  
# Mostrar resultado  
print(f"La resistencia es: {resistencia} Ohm")
```

¡Nuevo elemento! Input: Función para que la persona usuaria ingrese los datos. Note que debe definirse el tipo de datos que se espera.

Estructuras de control

¿Cómo funcionan los condicionales?

Un condicional evalúa una expresión lógica y ejecuta un bloque de código si la condición es True.

```
if condicion:  
    # Código si la condición es verdadera  
elif condicion:  
    # Código: si la condición anterior no se cumple. Si se cumple, no se evalúa ésta.  
else:  
    # Código si las condiciones anteriores son falsas.
```

El intérprete de Python evalúa la condición y, dependiendo de si es `True` o `False`, ejecuta el bloque correspondiente.

Estructuras de control

Ejemplo:

```
# Evaluación de temperatura en un transformador
temperatura = float(input("Ingrese la temperatura del
transformador (°C): "))

if temperatura > 80:
    print(" Advertencia: Temperatura crítica.")
elif temperatura > 60:
    print(" Temperatura elevada, pero dentro del rango
seguro.")
else:
    print(" Temperatura normal.")
```



Comité Nacional Peruano de la CIER

Ejemplos y ejercicios



colab.research.google.com

PECIER_dia1_parte3



Comité Nacional Peruano de la CIER

SESIÓN 1

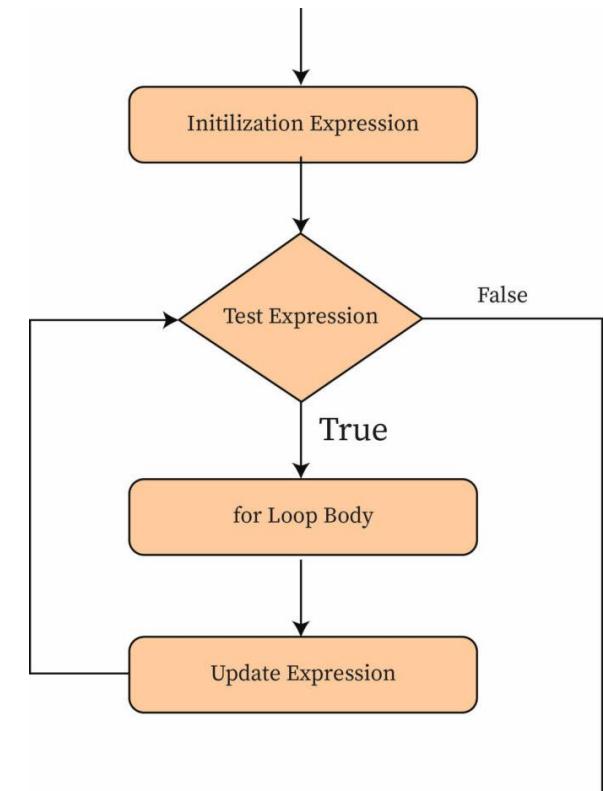
Parte 4

Bucles en Python

Los bucles permiten **repetir instrucciones varias veces**. Pensemos en un sistema que deba leer todas las líneas de una base de datos, o incluso todas las líneas de un archivo de Excel.

En lugar de programar la lectura de línea por línea, los bucles automatizan la realización de una tarea repetitiva.

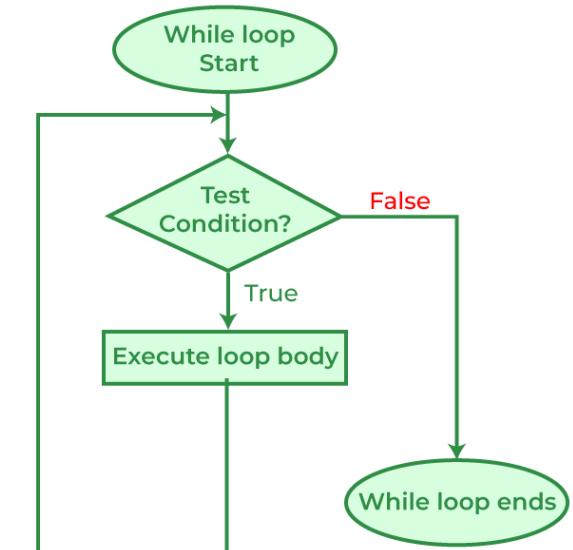
Se trata entonces de instrucciones para realizar una tarea una cierta cantidad de veces, o mientras se cumpla cierta condición en las variables.



Bucle `while`

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

```
while condicion:  
    # Código a repetir
```



Bucle `while`

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

`while condicion:`

`# Código a repetir`

En el siguiente código se muestra un bucle `while` que se ejecuta mientras se cumpla la condición de que la corriente sea menor o igual que 15 A.

```
# Simulación de un sistema que monitorea la corriente  
corriente = 5 # Amperios
```

```
while corriente <= 15:
```

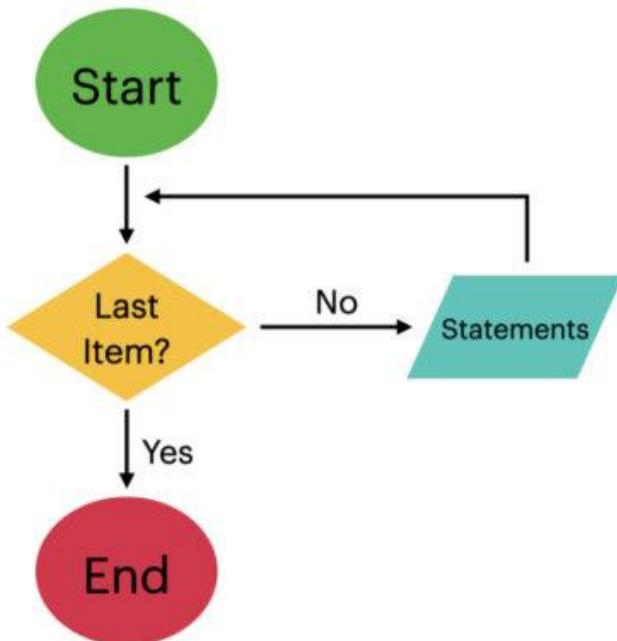
```
    print(f"Corriente actual: {corriente} A")
```

```
    corriente += 2 # Incremento gradual en dos unidades
```

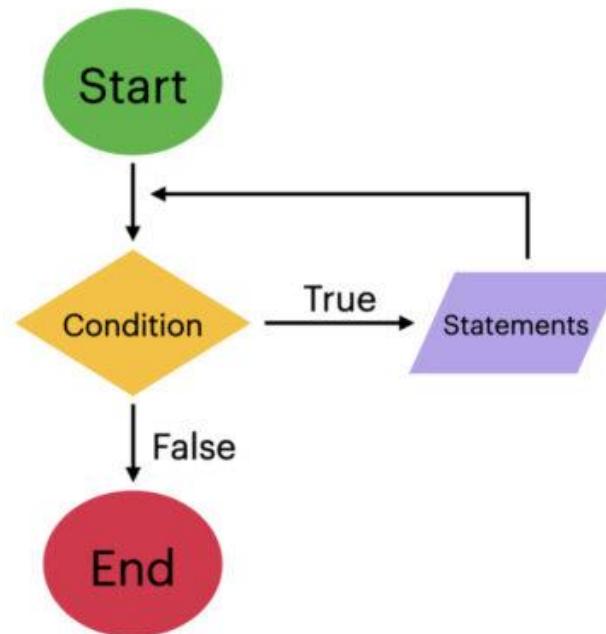
```
print(" Advertencia: Corriente crítica.")
```

`for` vs `while`

For Loop



While Loop



Mejor cuando: Cantidad de operaciones es conocida

Ejemplo Ejemplo

code

```
1 a = 1
2 while a < 10:
3     print (a)
4     a += 2
```

variables

output

Importante: Debemos tener cuidado con bucles infinitos:

```
# Simulación de un sistema que monitorea la corriente
corriente = 5 # Amperios

while corriente <= 15:
    print(f"Corriente actual: {corriente} A")
    print("Advertencia: Corriente crítica.")
```

Los bucles se pueden detener si sucede cierta condición interna, con el uso de `break`.

Ejemplo: Detener el bucle en un punto específico:

```
corriente = 20 # Amperios
while corriente <= 20:
    if corriente == 12:
        print(" Advertencia: Corriente alcanzó 12 A, deteniendo monitoreo.")
        break
    print(f"Corriente actual: {corriente} A")
    corriente = corriente - 1
```

Corriente actual: 20 A
Corriente actual: 19 A

Uso de for

El bucle `for` es útil para recorrer listas de datos. A diferencia de `while`, `for` se ejecuta para recorrer listas o elementos, en lugar de buscar que se cumpla una condición.

Ejemplo: Cálculo de Potencia en Diferentes Corrientes

```
# Potencia para diferentes valores de corriente
voltaje = 220
corrientes = [5, 10, 15, 20]

print("Corriente (A)\tPotencia (W)")
for corriente in corrientes:
    potencia = voltaje * corriente
    print(f"{corriente}\t\t{potencia}")
```

Uso de for

El bucle `for` es útil para recorrer listas de datos. A diferencia de `while`, `for` se ejecuta para recorrer listas o elementos, en lugar de buscar que se cumpla una condición.

Ejemplo: Cálculo de Potencia en Diferentes Corrientes

```
# Potencia para diferentes valores de corriente
voltaje = 220
corrientes = [5, 10, 15, 20]

print("Corriente (A)\tPotencia (W)")
for corriente in corrientes:
    potencia = voltaje * corriente
    print(f"{corriente}\t\t{potencia}")
```

En el ejemplo anterior, observe que `corriente` es variable, mientras que `voltaje` es una constante.

Bucles + condicionales

Marcar con `sobrecarga` si la potencia supera los `3000 W`:

```
# Potencia para diferentes valores de corriente
voltaje = 220
corrientes = [5, 10, 15, 20]
for corriente in corrientes:
    potencia = voltaje * corriente
    if potencia > 3000:
        print(f"{corriente} A - {potencia} W Sobrecarga")
    else:
        print(f"{corriente} A - {potencia} W")
```



Comité Nacional Peruano de la CIER

Ejemplos y ejercicios



colab.research.google.com

PECIER_dia1_parte4



PECIER

Comité Nacional Peruano de la CIER

E-mail: pecier@cier.org