

Guía de estudio N.º 2 – Conociendo Python (Parte II)



¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase, además de profundizar temas adicionales que complementan aquellos vistos en la clase.

Esta guía de ejercicios te ayudará a comprender los conceptos básicos de **diagramas de flujo**, **operadores**, control de flujo: **if-else-elif** y ciclos **for** y **while** de una manera práctica.

Los ejercicios están diseñados para que puedas resolverlos paso a paso y a tu propio ritmo, sin importar tu nivel de experiencia en programación. Además, cada ejercicio viene con una explicación detallada de cómo funciona el código, para que puedas entenderlo fácilmente.

¡Vamos con todo!



Tabla de contenidos

Guía de estudio N°2— Conociendo Python	1
¿En qué consiste esta guía?	1
Tabla de contenidos	2
Algoritmos	3
¿Qué es un algoritmo?	3
Diagramas de flujo	3
Operadores	4
Estructuras de control	5
Ciclos	1
Ciclo For	1
Ciclo While	1
Actividad guiada N°1: La contraseña	1
Algunos trucos	1
La función range	1
Compresión de lista	1
Actividad guiada N°2: Listas por compresión	1
Comprensión de diccionarios	1
¡Manos a la obra! - ¡A poner en práctica!	1



¡Comencemos!

Algoritmos

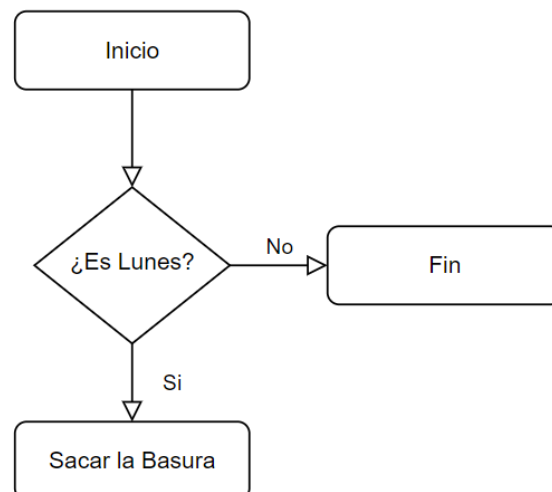
¿Qué es un algoritmo?

Se llama **algoritmo** a una serie de pasos ordenados y bien definidos que se utilizan para resolver un problema específico.



- Debe entregar, efectivamente, una solución o respuesta.
- Debe ser finito, es decir, terminar (esto no es trivial).
- Debe ser preciso, es decir, no prestarse a interpretaciones.

Diagramas de flujo

Los diagramas de flujo son una herramienta gráfica que se utiliza para representar la secuencia de pasos o procesos en un programa o algoritmo. Se pueden utilizar para planificar la lógica de un programa antes de escribir el código, lo que puede ayudar a identificar errores y simplificar la tarea de programación. Por ejemplo, a partir del día que es hoy, queremos saber si sacar la basura o no:



Hay varios símbolos que se utilizan comúnmente en los diagramas de flujo para representar diferentes tipos de instrucciones y decisiones, las tres figuras más básicas son:

-  **Rectángulos:** se utilizan para representar procesos o actividades.
-  **Rombos:** se utilizan para representar decisiones, es decir, puntos en el proceso en los que se debe tomar una decisión basada en cierta condición.

- ➡ **Flechas:** se utilizan para conectar las diferentes figuras geométricas y mostrar la dirección del flujo del proceso.

En los **diagramas de flujo**, las acciones se representan mediante **rectángulos**, los cuales contienen la descripción de la acción a realizar. Por otro lado, las decisiones se denotan como **rombos** y representan un punto de bifurcación en el flujo del programa. Es decir, dependiendo de la respuesta a la pregunta planteada en la decisión, el programa tomará un camino u otro. Existen otras formas, además, para representar entrada de información o salidas, por ejemplo, que pueden depender de la costumbre o el uso. Pero estas son las fundamentales.



¿Te gustaría construir tus propios diagramas personalizados? Visita draw.io

Operadores

En Python, al igual que en SQL, se utilizan **valores booleanos** (True y False) y **operadores lógicos** (and, or, not) para construir expresiones booleanas que son muy útiles para combinar múltiples condiciones y crear expresiones lógicas más complejas, de esta forma nos permiten tomar decisiones en nuestro código.

1. **Operadores aritméticos:** Estos operadores se utilizan para realizar operaciones matemáticas básicas en Python. Los operadores aritméticos incluyen = (asignación), + (suma), - (resta), * (multiplicación), / (división), % (módulo) y ** (potencia).

```
a = 10
b = 4
suma = a + b
resta = a - b
multiplicacion = a * b
division = a / b
división entera: a // b
modulo = a % b
potencia = a ** b
```

2. **Operadores de comparación:** Estos operadores se utilizan para comparar dos valores o variables. Los operadores de comparación incluyen == (igual a), != (no igual a), > (mayor que), < (menor que), >= (mayor o igual que) y <= (menor o igual que).

```
a = 10
```

```
b = 5
resultado1 = a == b # Compara si a es igual a b
resultado2 = a != b # Compara si a es diferente de b
resultado3 = a > b # Compara si a es mayor que b
resultado4 = a < b # Compara si a es menor que b
resultado5 = a >= b # Compara si a es mayor o igual que b
resultado6 = a <= b # Compara si a es menor o igual que b
```

3. **Operadores lógicos:** Estos operadores se utilizan para combinar dos o más condiciones. Los operadores lógicos incluyen and (y), or (o) y not (no).

```
a = 10
b = 5
c = 15
# Comprueba si a es mayor que b, y si c es mayor que a
if a > b and c > a:
    print("Ambas condiciones son verdaderas")
```

Estructuras de control

Las estructuras de control de flujo más importantes en Python son **if**, **else** y **elif**. Ellas nos permiten tomar decisiones en nuestro código dependiendo de ciertas condiciones.

Al igual que en SQL, donde utilizamos operadores de comparación para filtrar los datos que necesitamos, en Python podemos utilizar estos mismos operadores en nuestras condiciones para controlar el flujo de nuestro programa. Por ejemplo, si queremos verificar si una persona es mayor de edad antes de permitirle entrar a un bar, podemos utilizar la siguiente estructura de control de flujo:

```
if edad >= 18:
    print("Puede entrar al bar")
else:
    print("No se le permite entrar al bar")
```

En este caso, estamos utilizando el operador de comparación **>=** para verificar si la edad es mayor o igual a 18. Si la condición es verdadera, se ejecutará la primera parte del código, y si no, se ejecutará la segunda parte.

La estructura **"if-elif-else"** nos permite manejar múltiples condiciones de forma ordenada y efectiva. Por ejemplo, si queremos calificar a los estudiantes usando letras en vez de puntaje numérico en una prueba, podemos utilizar la siguiente estructura de control de flujo:

```
puntaje = 86
if puntaje >= 90:
    print("Nota A")
elif puntaje >= 80:
    print("Nota B")
elif puntaje >= 70:
    print("Nota C")
else:
    print("Nota D")
```



¡Cada ejercicio resuelto es un paso más hacia el éxito!

Ciclos

En Python, los ciclos nos permiten repetir una sección de código varias veces, lo que puede ser útil cuando queremos realizar una tarea repetitiva o cuando queremos iterar a través de una estructura de datos.

Existen dos tipos principales de ciclos en Python: el **ciclo for** y el **ciclo while**.

Ciclo For

El **ciclo for** se utiliza para iterar sobre una secuencia de elementos, como una lista o una cadena de caracteres. La sintaxis básica de un ciclo for es la siguiente:

```
for elemento in secuencia:
    # Código a ejecutar en cada iteración
```

Aquí, **elemento** es una variable que toma el valor de cada elemento en la secuencia en cada iteración del ciclo. El código dentro del ciclo se ejecuta una vez por cada elemento en la secuencia. Por ejemplo, si queremos imprimir cada número en una lista, podemos usar un ciclo for de la siguiente manera:

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

for num in numeros:
    print(num)
```

Esto imprimirá los números del 1 al 5 en la consola.

Ahora, para poder filtrar una lista, crearemos otra lista que contenga sólo ciertos elementos de la primera. Así, por ejemplo, a partir de la lista **numeros** podemos crear una lista nueva llamada **pares**.

```
numeros = [1, 2, 3, 4, 5]
pares = []

for num in numeros:
    if num % 2 == 0:
        pares.append(num)

print(pares)
```

Ciclo While

El **ciclo while** se utiliza para repetir una sección de código mientras se cumpla una condición. La sintaxis básica de un ciclo while es la siguiente:

```
while condición:
    # Código a ejecutar mientras se cumpla la condición
```

Aquí, condición es una expresión booleana que se evalúa al comienzo de cada iteración del ciclo. Si la condición es verdadera, el código dentro del ciclo se ejecuta. El ciclo continuará repitiéndose mientras la condición siga siendo verdadera.

```
num = 1

while num <= 5:
    print(num)
    num += 1
```

Esto imprimirá los números del 1 al 5 en la consola.



Actividad guiada N°1: La contraseña

Crea un programa que solicite una contraseña al usuario y la verifique para permitir el acceso, el programa debe repetirse hasta que la contraseña sea correcta. 🗝️🤖.

```
password = "contraseña"

while True:
    entrada = input("Ingrese la contraseña: ")
    if entrada == password:
        print("Contraseña correcta. Bienvenido.")
        break
    else:
        print("Contraseña incorrecta. Intente de nuevo.")
```

Algunos trucos

La función range

La función **range** nos permite obtener, rápidamente, una lista de números con un máximo de tres parámetros. Su sintaxis es **range(inicio, fin, paso)**, donde solo **fin** es obligatorio. Podemos obtener así las siguientes combinaciones:

- `range(4)` genera la lista 0, 1, 2, 3. Observa que comienza de cero y llega hasta fin-1
- `range(2, 7)` genera la lista 2, 3, 4, 5, 6, 7
- `range(9, 4, -1)` genera la lista 9, 8, 7, 6, 5
- `range(3, 12, 3)` genera la lista 3, 6, 9

Puedes probar las diferentes combinaciones que permite esta función, imprimiendo los resultados como se muestra.

```
for p in range(3,18,2):
    print(p)
```


Compresión de lista

La **compresión de lista** es una herramienta muy útil para simplificar el código y hacerlo más legible. Las **compresiones de lista** son una forma concisa y expresiva de crear una lista en Python. En lugar de crear una lista vacía y luego usar un **ciclo for** para agregar elementos a la lista, una **compresión de lista** nos permite definir la lista y su contenido en una sola línea de código.

```
# sintaxis básica de la compresión de lista  
[expresión for elemento in iterable if condición]
```

Por ejemplo, si queremos crear una lista de los números pares de otra lista dada, podemos hacerlo de esta manera con un **ciclo for**:

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
pares = []  
  
for numero in numeros:  
    if numero%2 == 0:  
        pares.append(numero)
```

Pero podemos usar en lugar de esto una **compresión de lista** de la siguiente manera:

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
pares = [num for num in numeros if num % 2 == 0]
```



Actividad guiada N°2: Listas por compresión

Crearemos un programa que compare dos listas y muestre los elementos que están presentes en ambas listas.

```
# Ejercicio de comparación de listas  
  
lista1 = ["manzana", "naranja", "uva", "fresa", "kiwi"]  
lista2 = ["sandía", "mango", "kiwi", "fresa", "limón"]
```

Solución:

Aquí está cómo se podría resolver el ejercicio utilizando un **ciclo for**. Observa que, además, utilizamos el operador **in** para verificar si un elemento está en una lista o no::

```
comunes = []
for elemento in lista1:
    if elemento in lista2:
        comunes.append(elemento)

print("Elementos comunes en ambas listas: ", comunes)
```

Aquí está cómo se podría resolver el ejercicio utilizando **comprensión de listas**:

```
comunes = [elemento for elemento in lista1 if elemento in lista2]
print("Elementos comunes en ambas listas: ", comunes)
```

La **comprensión de listas** en general es más eficiente que los **ciclos for**, ya que se ejecutan más rápido y requieren menos código. Además, la **comprensión de listas** puede ayudar a reducir la complejidad y la cantidad de errores en el código.

Comprensión de diccionarios

La **comprensión de diccionarios** en Python es una técnica que permite construir diccionarios de manera concisa y eficiente a partir de iterables. Es similar a la **comprensión de listas**, pero en lugar de crear una lista, se crea un diccionario.

La sintaxis básica es la siguiente:

```
{clave: valor for variable in iterable}
```

Aquí, **clave** y **valor** son expresiones que definen la **clave** y el **valor** de cada elemento en el diccionario. **variable** es la **variable** que toma los valores de cada elemento en el iterable, y iterable es una secuencia de elementos (por ejemplo, una lista, un rango, etc.).

Para mostrarlo, consideraremos un programa en el que el usuario debe ingresar una cantidad *n* de lanzamientos de dados que desea generar. Luego, el programa generará *n* lanzamientos de dos dados y guardará los resultados en un diccionario con nombres de clave de lanzamiento. Esto se podría resolver utilizando un **ciclo for**, de la siguiente manera:

```
n = int(input("Ingrese la cantidad de lanzamientos: "))












dicc = {}
for i in range(1,n):
    dicc[f"lanzamiento_{i}"] = random.randint(1,6) + random.randint(1,6)
```


Ahora, en lugar de usar un **ciclo for**, utilizaremos una **comprensión de diccionarios** para generar los lanzamientos y guardar los resultados en un diccionario.





```
dicc = {"lanzamiento_{i}": random.randint(1,6) + random.randint(1,6)  
for i in range(1, n)}
```



¡Manos a la obra! - ¡A poner en práctica!

1.  Crea un programa que tome una cadena de texto y cuente la cantidad de veces que aparece cada palabra en la cadena. Utiliza un diccionario para almacenar los conteos .
2. El profesor quiere que practiquemos nuestras tablas de multiplicar. Para hacerlo, vamos a crear un programa que pregunte al usuario por un número  y muestre la tabla de multiplicar correspondiente . El programa seguirá pidiendo números hasta que el usuario ingrese un número negativo .
3.  En una consulta médica se han registrado los datos de pacientes que han asistido a un examen general. Los datos se encuentran almacenados en una lista de diccionarios, donde cada diccionario representa los datos de un paciente y contiene las claves:
 - **nombre:** el nombre del paciente 
 - **edad:** la edad del paciente en años 
 - **peso:** el peso del paciente en kilogramos 
 - **altura:** la altura del paciente en metros 
 - **temperatura:** la temperatura corporal del paciente en grados Celsius 

El médico desea saber cuál es el promedio de **edad**, **peso**, **altura**, **presión** y **temperatura** de los pacientes que han asistido a la consulta. ¿Podrás ayudarlo a obtener esta información? 

4.  Crea un programa interactivo que permita al usuario elegir su destino interplanetario favorito entre la Luna y Marte, y luego seleccionar entre dos emocionantes actividades: hacer una caminata espacial o pilotar un rover   .

Soluciones

Ejercicio 1

```
texto = "En un lugar de la Mancha, de cuyo nombre no quiero acordarme,  
no ha mucho tiempo que vivía un hidalgo"
```

```
conteo = {}  
palabras = texto.split()  
for palabra in palabras:  
    if palabra in conteo:  
        conteo[palabra] += 1  
    else:  
        conteo[palabra] = 1
```

```
print(conteo)
```

Por otro lado, podemos usar una **comprensión de diccionario** de la siguiente manera:

```
texto = "En un lugar de la Mancha, de cuyo nombre no quiero acordarme,  
no ha mucho tiempo que vivía un hidalgo"
```

```
conteo = {palabra: texto.count(palabra) for palabra in texto.split()}
```

```
print(conteo)
```

Ejercicio 2

```
while True:  
    num = int(input("Ingresa un número (negativo para salir): "))  
  
    if num < 0:  
        print("Adiós, hasta la próxima!")  
        break  
  
    print(f"Tabla de multiplicar del {num}:")  
    for i in range(1, 11):  
        print(f"{num} x {i} = {num*i}")
```

Ejercicio 3

```
pacientes = [  
    {'nombre': 'Juan', 'edad': 30, 'peso': 80, 'altura': 1.75,  
    'temperatura': 36.5},
```

```
{'nombre': 'María', 'edad': 25, 'peso': 60, 'altura': 1.65,
'temperatura': 37.0},
{'nombre': 'Pedro', 'edad': 40, 'peso': 90, 'altura': 1.80,
'temperatura': 36.8},
{'nombre': 'Ana', 'edad': 35, 'peso': 70, 'altura': 1.70,
'temperatura': 37.2}
]

# Promedio de edad
edad_promedio = sum([paciente['edad'] for paciente in pacientes]) /
len(pacientes)

# Promedio de peso
peso_promedio = sum([paciente['peso'] for paciente in pacientes]) /
len(pacientes)

# Promedio de altura
altura_promedio = sum([paciente['altura'] for paciente in pacientes]) /
len(pacientes)

# Promedio de temperatura
temperatura_promedio = sum([paciente['temperatura'] for paciente in
pacientes]) / len(pacientes)

#Impresión de resultados
print(f"Promedio de edad: {edad_promedio}")
print(f"Promedio de peso: {peso_promedio}")
print(f"Promedio de altura: {altura_promedio}")
print(f"Promedio de temperatura: {temperatura_promedio}")
```

#Viaje al espacio

Ejercicio 4

```
destino = input("¿Quieres viajar a la 🌍 Luna o a 🚀 Marte? ")

# Verificar la respuesta del usuario y preguntar si quiere hacer una
caminata espacial o pilotar un rover

if destino == "Luna":
    actividad = input("¿Quieres hacer una 🚶 caminata espacial o pilotar
un 🚗 rover? ")
    if actividad == "caminata espacial":
        print("Prepárate para poner un pie en la Luna 🚀🌍👤")
```

```
elif actividad == "rover":
    print("¡A pilotar el rover y explorar la Luna! 🚀🌕🚗")
else:
    print("Lo siento, no entiendo esa actividad 😞")
elif destino == "Marte":
    actividad = input("¿Quieres hacer una 🚶 caminata espacial o pilotar un 🚗 rover? ")
    if actividad == "caminata espacial":
        print("¡Una caminata espacial en Marte! Esto es un sueño hecho realidad 🚀🌌👤")
    elif actividad == "rover":
        print("¡A pilotar el rover y explorar Marte! 🚀🌌🚗")
    else:
        print("Lo siento, no entiendo esa actividad 😞")
else:
    print("Lo siento, no entiendo ese destino 😞")
```



¡Continúa aprendiendo y practicando!