



INTRODUCCIÓN A PYTHON

E3

Contenido

PRORAMACIÓN I	¡Error! Marcador no definido.
Día 03	5
Entrada de datos	5
Salida de datos	6
Operadores Aritméticos.....	8
Operadores relacionales	9
Operadores Binarios	9
Comentarios.....	11
Librerías.....	12
OS	12
Math.....	13
Conjunto de instrucciones	14
Instrucciones de Decisión	14
Bibliografía	17

“

Este espacio curricular permite fortalecer el espíritu crítico y la actitud creativa del futuro graduado. Lo alienta a integrar los conocimientos previos, recreando la práctica en el campo real, al desarrollar un proyecto propio.

”



PYTHON

Programación en Python

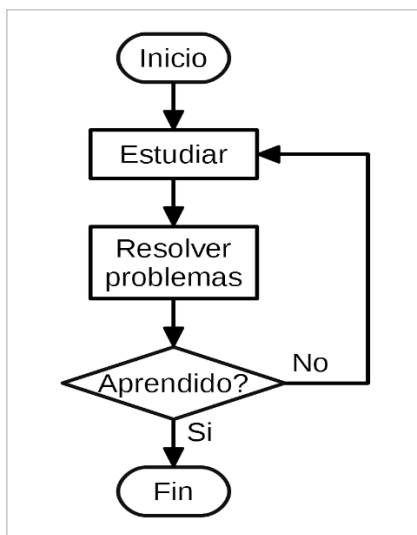
Comenzamos el tercer encuentro, vas a notar que se vuelve un poco más complejo. Pero no te preocupes, es normal que esto pase. Vamos a trabajar con nuevas instrucciones que le suman dificultad. En este encuentro vamos a profundizar en la instrucción `print`, pero el tema más importante es como usar las instrucciones de repetición.

¡¡Así que... de a poco vamos avanzando!!

Día 03

Antes de comenzar, vamos a definir el concepto de flujo de información. Ya habíamos visto que un algoritmo, es una secuencia ordenada de pasos que se ejecutan en forma secuencial. entonces, para indicar cuál es ese recorrido, usamos el flujo de información.

El flujo, es el camino que recorre la computadora al ejecutar cada línea de código. Existen muchas formas de representar esto, en la etapa de análisis. Una de las maneras más simple es usando los diagramas de flujos. como vemos a continuación:



Si se observa el diagrama, vas a reconocer dos cosas. La primera son Las figuras, cada una de las figuras representa una instrucción de Python. La segunda, son las flechas. Esa flecha, representa el flujo de información. Es decir, te

indica el recorrido que seguirá la computadora al ejecutar cada parte del código.

Para terminar, es importante comprender la importancia de este recorrido. en este sentido, más adelante vamos a conocer instrucciones que nos van a permitir cambiar el recorrido del flujo de información.

Entrada de datos

En este apartado, nos vamos a concentrar solo en la entrada de datos por teclado. Para programar una entrada de datos, usaremos la palabra reservada **input**.

Toda instrucción, tiene una sintaxis. Para la instrucción input, la podríamos representar de la siguiente manera:

Input([Mensaje])

Esta instrucción permite ingresar datos por teclado, y convierte la entrada del teclado en una cadena de caracteres. Un ejemplo simple, lo podemos ver en el siguiente código:

```
a=input()
print(f"El valor ingresado es {a}")
```

el código anterior, pide un valor por teclado. Luego, el valor ingresado se asigna a la variable **a**. Es simple verdad, para terminar, se muestra por pantalla el valor de la variable. cómo se puede ver en la siguiente imagen:



Las entradas y salidas de datos se gestión con la instrucción input y print

```
2
El valor ingresado es 2
```

Vamos a realizar algunas mejoras en el código anterior. Observarán en la imagen que no se proporciona información al usuario. Por lo tanto, al ingresar un valor, es posible que este último no se dé cuenta de que estamos solicitando una entrada por teclado.

Mediante la instrucción `input`, es posible agregar una descripción que será visualizada en la pantalla.

```
a=input("Ingrese un valor : ")
print(f"El valor ingresado es {a}")
```

El resultado en pantalla se vería de la siguiente manera:

```
Ingrese un valor : 2
El valor ingresado es 2
```

Dado que la instrucción `input` siempre devuelve un valor de tipo cadena, es posible utilizar la conversión de datos para transformarlo en un entero, como se muestra en este ejemplo

```
a=int(input("Ingrese un valor : "))
print(f"El valor ingresado es {a}")
```

Salida de datos

La instrucción `print` se utiliza para mostrar datos en la pantalla, su sintaxis se resume de la siguiente manera:

```
print("mensaje", sep="",end="")
```

En la función **`print`** existen varios componentes. En primer lugar, tenemos la cadena de texto, seguida de **`sep`**, un argumento opcional que representa el carácter utilizado para separar dos mensajes. Contamos con una amplia gama de opciones para mostrar información en la pantalla.

Mostrar un mensaje

es la opción más común, donde se muestra una cadena de texto que se encuentra entre comillas. Si deseamos mostrar el valor de una variable, no utilizamos comillas.

```
print("El valor ingresado es ")
print(a)
```

Mostrar dos mensajes concatenados

La instrucción `print`, permite incluir más de un argumento, en este caso se separan con una coma

```
print("El valor","ingresado es ")
```

una línea como separador

Cuando queremos crear espacio para mejorar la visualización del mensaje, podemos utilizar la instrucción **print** sin parámetros.

```
print()
```

salto de línea

La instrucción **print**, ejecuta un salto de línea siempre.

```
print("Hola")  
print("Mundo")
```

Pueden notar que la cadena Mundo, se muestra en la línea de abajo

```
Hola  
Mundo
```

Cómo Evitar el salto de línea

Agregamos el argumento `end=""` al final:

```
print("Hola",end="")  
print("Mundo")
```

a continuación, puede ver la diferencia

```
HolaMundo
```

Si queremos poner un espacio, sólo lo modificamos y colocamos `end=" "`

Ignorar texto

Muchas veces, tenemos que poner un carácter que puede destruir la estructura del mensaje o sólo deseamos que se muestre como tal. Para eso usamos el carácter `\` (ALT +92).

```
print("el \@ y la \" son caracteres")
```

Las cadenas f

Cuando se desea embeber texto dentro de otra cadena y que Python lo utilice como si estuviera formateado, se utilizan las f-strings. Para ello, simplemente se antepone la letra `f` al mensaje y se coloca el texto incorporado entre llaves `{}`.

```
edad=24  
print(f"su edad es {edad}")
```

el formato %

se utiliza para formatear la cadena de texto.

```
edad=str(24)  
print("su edad es %s" % f"{edad}")
```

se puede colocar más de un texto, y vinculo es respetando la secuencia. Con una variante en el format que también está permitida.

```
edad=str(24)  
print("la edad de %s es %s" % ("Pablo",edad))
```

el método `str.format`

es similar al `%`, pero para indicar los valores dentro de la cadena se usa `{}` ó `{#}`

```
edad=str(24)
print("la edad de {} es {}".format("Pablo",f"{edad}"))
```

variante con números permitido en el format.

```
edad=str(24)
print("la edad de {0} es {1}".format("Pablo",edad))
```

```
la edad de Pablo es 24
```

Separadores

Usamos un carácter para separar dos argumentos

```
print("Columna 1","Columna2", sep="-")
```

Separar en ancho de columna

```
print("{0:15}{1:15}{2:15}".format("Columna 1","Columna 2","Columna 3"))
```

```
Columna 1      Columna 2      Columna 3
```

Mostrar flotante con decimales

Es similar al anterior, pero se determina la cantidad de decimales y el tipo de dato numérico

```
temperatura=25.656566
print("La temperatura es {0:.2f}".format(temperatura))
```

Operadores Aritméticos

Tenemos 3 tipos de operadores, uno de ellos es el operador aritmético. Como su nombre lo indica, permite hacer operaciones aritméticas con las variables.

Operador	Descripción
+	Suma dos números, o concatena dos string
-	Resta
*	Multiplicación
/	División con resultado siempre float
%	Módulo de la división
//	División con resultado int
**	Potencia

Por ejemplo, el siguiente código suma los valores de ambas variables:

```
coordenadaX=10
mover=2
coordenadaX= coordenadaX + mover
```

el código anterior, cambiar el valor de 10 en la `coordenadaX` a un nuevo valor de 12. Que tiene de interesante el `+`, que actúa diferente cuando las variables son cadenas:

```
cadena1="Hola"
cadena2="Mundo"
cadena3=cadena1 + " " + cadena2
print(cadena3)
```


este código me muestra en pantalla:

```
Hola Mundo
```

Operadores relacionales

Tenemos otro grupo de operadores llamados relacionales, también podemos conocerlos como comparativos. Su utilidad es comparar dos valores y devolver *True* o *False* como resultado según corresponda. Veamos un ejemplo:

```
print(2==3)
```

Este código nos muestra en pantalla:

```
False
```

La explicación es simple, como 2 “no es igual a” 3, el resultado es Falso.

Para entender mejor estos operadores, les presentamos la tabla con cada uno de ellos:

Operador	Descripción
<code>==</code>	Igual
<code>!=</code>	Distinto
<code>></code>	Mayor
<code><</code>	Menor
<code>>=</code>	Mayor o igual
<code><=</code>	Menor o igual

Operadores Binarios

Terminamos esta descripción de Operadores, presentando los operadores binarios `and` -or y `not`. También tenemos los operadores bit a bit, `&` y `|`.

La tabla de operadores lo mostramos a continuación:

Operador bit a bit	Descripción
<code>&</code>	And (Y)
<code> </code>	Or (o)
	Not

Ahora bien, el uso de los operadores lógicos o bit a bit tiene dos caras, una compleja y una simple. Veamos la compleja primero. La comparación bit a bit con cada operando, obliga a convertir el valor en binario.

Veamos el resultado del código siguiente:

```
print(2 & 3)
```

Resultado es

```
2
```

¿No ayuda verdad?

Esto es porque tenemos que convertir cada operando en números binarios. En este caso el número 2 en binario es 10, y el 3 en binario es 11. Entonces, el código anterior quedaría

como¹:

```
print(bin(0b10 & 0b11))
0b10
```

¿Esto ayuda menos verdad?

Qué largo lo estoy haciendo, pero antes de continuar te voy a mostrar la tabla de verdad² del **operador &**:

Operando		Salida
A	B	A&B
1	1	1
1	0	0
0	1	0
0	0	0

Ok, antes de entrar en desesperación veamos lo siguiente. El operador compara bit a bit

Entonces:

10
11

10

Al comparar bit a bit, el resultado de 1 & 1 es → 1, por otro lado, el resultado de 0 & 1 es → 0. Ahora, podemos decir que el resultado de **0b10 & 0b11 es 0b10**.

Sabiendo esto último, vamos a cambiar la tabla de verdad. Donde teníamos 1 pondremos True. Donde teníamos 0, pondremos False. La tabla para el operador and nos queda así:

¹ Recuerda que anteponer el 0b (cero + b) para representar números binarios. Por su parte la función bin() de Python convierte un numero decimal en binario

¿Qué concluimos?

Sólo tendremos True como resultado, cuando los dos operandos son True.

Operando		Salida
A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

Los otros dos operadores, funcionan de la misma manera, pero con resultados diferentes.

El **operador OR bit a bit**

El símbolo que se usa es el pipe (|), representa el operador OR – se lee “o”, el

Operando		Salida
A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

carácter ASCII es ALT+124— funciona contrario al operador AND.

¿Cuál sería el resultado de...?

```
print(bin(2 | 3))
```

² La tabla de verdad proviene de la tecnología digital, nos muestra cual es el resultado final, cuando tenemos un determinado valor binario en la entrada de la compuerta.

Correcto 0b11 ☺

La tabla de verdad del operador OR es

Por último, tenemos el **operador NOT**

Esto es simple, a la salida obtenemos lo contrario de la entrada

Operador- Salida	
A	!A
True	False
False	True

Los operadores bit a bit junto con los operadores relacionales, se usan mucho para construir condiciones. Algo que veremos más adelante.

Comentarios

Los comentarios, no son parte del código y se eliminan al compilar. Estos comentarios, son información interna que podemos colocar en el código fuente. Tenemos que ser muy cautos al usar comentarios, el abuso puede ser contraproducente para la legibilidad del programa.

En Python tenemos dos tipos de comentarios, de una sola línea o de múltiples líneas.

Los comentarios de una sola línea se usan también, como comentarios inline. Para los comentarios en una sola línea usamos el carácter #.

```
x,a,b=2,3,2
x = a if b else 0 #esto es un if
```

Los comentarios, no se usan sólo como comentario inline, también puede estar solos en una línea del código fuente.

```
# Asignamos valores a las variables
x,a,b=2,3,2
x = a if b else 0
```

Continuando con los comentarios, otra forma es hacerlo en más de una línea. Para ello, usamos las tres comillas simples (""") al comienzo y final.

```
'''
    Ejercicio 1
    Python
    Autor: Pablo
    Fecha:02-03-22
'''
```

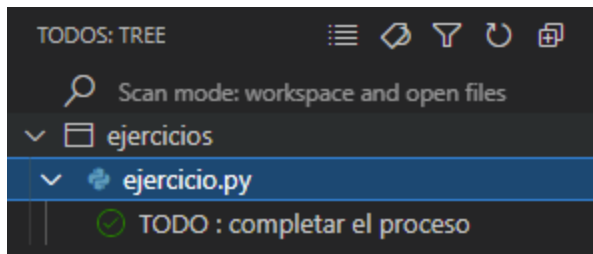
Continuamos con Python y comentarios, tenemos un par de opciones de “comentarios especiales”. Un caso es el TODO.

#TODO

Se utiliza para indicar acciones pendientes en el código:

```
# TODO: completar el proceso
x,a,b=2,3,2
x = a if b else 0
```

Te invito a instalar la librería **Todo Tree**, y podrás ver todos los comentarios #TODO del código y navegar entre ellos



Librerías

¡¡Bien!!

Estamos entrando en librerías, eso significa que ya vamos a estar haciendo cosas más potentes.

Las Librerías son como los powerups de los videojuegos, al importarlos en nuestro código fuente nos da más poderes. Desde luego, hay muchas librerías y no estamos obligados a importar todas. Sólo lo hacemos con aquellas que vamos a usar.

Para importar librerías usamos la palabra reservada import

```
import math
```

noten el detalle de indicar a qué librería pertenece, al completar la instrucción con math.pi:

```
import math
print (math.pi)
```

También, nos permite importar y renombrar la librería, este es un ejemplo

```
import math as mate
print (mate.pi)
```

Usar la palabra import, indica que importamos toda la librería math. Pero también podemos usar un import parcial. Esta acción se conoce como importación de submódulo. Si sólo queremos importar el método pi, usamos la instrucción from

```
from math import pi
print ((-pi))
```

en este caso, podemos comprobar que no todas las funciones de math, están disponibles.

```
from math import degrees
print (radians(pi))
```

En este caso el mensaje es que radians no está disponible. El código correcto para el ejemplo es:

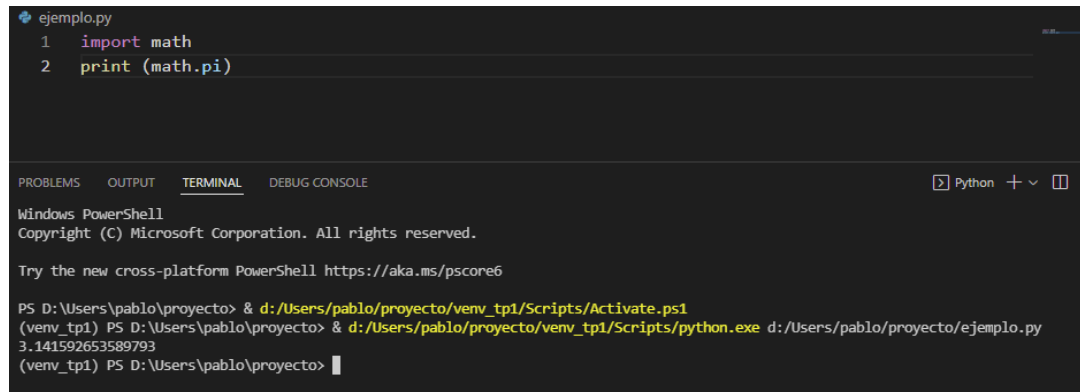
```
from math import degrees, radians, pi
print (radians(pi))
```

OS

Esta es una librería que provee funciones que me facilitan el trabajo con acciones dependientes del sistema operativo. Abarca desde abrir y gestionar archivos, crear carpetas o limpiar la consola.

En este nivel lo usaremos para incluir la función de limpiar pantalla:

representación de números, trigonométricas e hiperbólicas.



```
ejemplo.py
1 import math
2 print (math.pi)

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\Users\pablo\proyecto> & d:/Users/pablo/proyecto/venv_tp1/Scripts/Activate.ps1
(venv_tp1) PS D:\Users\pablo\proyecto> & d:/Users/pablo/proyecto/venv_tp1/Scripts/python.exe d:/Users/pablo/proyecto/ejemplo.py
3.141592653589793
(venv_tp1) PS D:\Users\pablo\proyecto>
```

Ilustración 1- Uso de la librería Math

```
#importar la librería
import os

'''
usando el método system, pasamos
como argumento cls para borrar la
consola
'''
os.system("cls")
```

Algo que, si es importante ver, es que las funciones trigonométricas no usan números enteros. Estas trabajan con radianes.

Un radian es una unidad de medida para ángulos, un sistema internacional de unidades de medida. Un radian es equivalente a

$$180^\circ/\pi$$

Python, por medio de la librería math, nos facilita la conversión de grados a radianes y, al contrario.

Math

Brinda acceso a todas las funciones matemáticas, y trigonométricas. Estas funciones no pueden ser usadas con números complejos. Como toda función, algo que veremos más adelante, devuelve un valor de retorno.

No es la idea describir cada una de las funciones, para eso te invito a visitar la página de documentación de Python en:

<https://docs.python.org/es/3/library/math.html>

En esa página, encontrar funciones logarítmicas, exponenciales, funciones de

```
import math
angulo=90
print (f"el angulo de {angulo} grados,
equivale a {math.radians(angulo)} radianes")
```

de igual modo, podemos pasar de radianes a grados

```
import math
print (math.degrees(1.57))
```

el resultado es 89.95437383553924, pero es claro que se debe al redondeo. También la librería math me brinda funciones como math.pi que devuelve el valor pi.

Conjunto de instrucciones

Todo lenguaje de programación debe tener, instrucciones secuenciales, de decisión y de repetición

En todos los lenguajes tenemos dos tipos de instrucciones, las simples y las compuestas. En Python, eso no es tan simple de describir. Pero podemos decir que por un lado estas las instrucciones simples como

```
print (radians(pi))
```

Es decir, cuando se trata de solo una línea de código. Y los bloques, cuando son varias líneas de código. En Python ese bloque, se representa con los dos puntos, y las líneas que están dentro del bloque separadas del margen izquierdo con un doble tab.

En general los bloques se usan en instrucciones con las decisiones y las repeticiones, pero también se usan en las funciones o las clases.

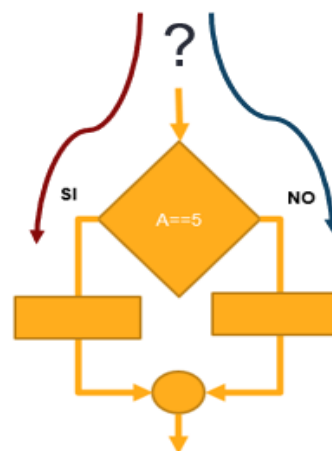
```
if a==b:
    print ("linea 1 del bloque")
    print ("linea 2 del bloque")
```

Instrucciones de Decisión

También conocidas como sentencias condicionales, se usan en todos los programas para crear caminos alternativos en el flujo del código. Estos condicionales nos permiten, que, dada una condición, elegir entre bloque de código u otro. Es importante saber que, no se pueden ejecutar ambos caminos a la vez.

Estas condiciones, son como preguntas. donde la respuesta a esa pregunta es sí o no... O también, verdadero o falso. Se usan con operadores lógicos y relacionales. Existen métricas que miden la complejidad del código contando la cantidad de expresiones condicionales. En el futuro, y sobre todo cuando programamos en objetos, es preferible tratar de evitar el uso de los IF.

Recordando conceptos de la primera clase, al diseñar algoritmos, habíamos hablado del flujo. Miremos el siguiente diagrama de flujo, aquí las flechas amarillas representan el flujo



del código.

entonces, que utilidad tienen las instrucciones de decisión. El rombo, representa una instrucción de repetición, es un punto de

inflexión que decide que camino se debe seguir.

La lógica funciona de la siguiente manera, el programa ejecuta cada instrucción hasta ese punto, entonces pregunta por la variable A. Dependiendo la respuesta elige el camino de la izquierda o de la derecha.

Las flechas rojas y azules representan el recorrido posible del flujo de información. Por qué aclaramos esto, porque al tener una **“instrucción de decisión (if)”** solo se puede ejecutar una de las dos ramas. Por lo que, el **“if”** determina cuál de las ramas se ejecutar y cual no.

Existen muchas formas de programar un **“if”**, y alternativas donde ponemos un **“if”** dentro de otro **“if”** (**“if”** anidado). Pero todo eso, depende de cada solución.

Todas las instrucciones **“if”**, se componen de una condición, una rama para la salida de verdadero y una rama para la salida del falso.

if [condición]:

Código cuando la condición da true

else:

Código cuando la condición da false

Las condiciones

Se trata de una comparación, generalmente se usan los operadores relacionales. También una condición puede estar compuesta por más de una comparación usando operadores lógicos

Las condiciones, tienen que estar armadas de tal forma que devuelvan como resultado verdadero o falso.

Condional solo con salida de verdadero

La instrucción **“if”** más simple, es aquella que se programa para decidir si un bloque de código se ejecuta o no. El flujo de datos sigue hasta la condición. Si la comparación en dicha condición es verdadera, el código se ejecuta. Si no, salta a la primera línea después del **“if”**.

```
if a==b:
    print ("linea 1 del bloque")
    print ("linea 2 del bloque")
print ("primera linea despues del if")
```

Analicemos el código anterior, después del **“if”** tenemos la condición (a==b). si eso es verdadero, es decir a “es igual a” b. Entonces se ejecuta el bloque. ¿cuál es el bloque? Luego de la condición tenemos el carácter “:”, eso indica que comienza el bloque. Todas las líneas de código, separadas con el doble **“tab”** es el bloque del **“if”**. Entonces, el bloque de código sería:

```
print ("linea 1 del bloque")
print ("linea 2 del bloque")
```

Luego de ejecutar el bloque avanza a la siguiente línea:

```
print ("primera linea despues del if")
```

si la condición (a==b), hubiese arrojado false. El bloque no se ejecuta, y salta a la línea:

```
print ("primera linea despues del if")
```

Simple...

Condional con salida de verdadero y falso

Una segunda opción, es programar bloques de código tanto para una posible respuesta verdadera, como para una falsa. El **“else”**, representa la rama falsa de la condición. Podría leerse como un “sino”. Es decir:

Si $a==b$ es verdadero entonces ejecutamos el bloque verdadero. Si no, ejecutamos el bloque falso.

Es importante aclarar, que “nunca” podremos ejecutar el bloque verdadero y el falso al mismo tiempo. En todos los casos, según la condición se ejecuta uno de los dos y sólo uno de los dos.

```
if a==b:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")
else:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")
print ("primera linea despues del if")
```

Condicional anidado

En muchas ocasiones, podemos programar un **“if”** dentro de un **“if”**. Esto es simple, porque el bloque de código puede contener cualquier instrucción de Python. Un ejemplo serio:

```
if a==b:
    if a > b:
        print ("linea 1 del bloque verdadero")
        print ("linea 2 del bloque verdadero")
    else:
        print ("linea 1 del bloque verdadero")
        print ("linea 2 del bloque verdadero")
print ("primera linea despues del if")
```

Podemos programarlo también en el bloque de falso:

```
if a==b:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")
else:
    if a > b:
        print ("linea 1 del bloque verdadero")
        print ("linea 2 del bloque verdadero")
print ("primera linea despues del if")
```

Elif

Cuando tenemos que anidar los **“if”** como el ejemplo anterior. Python nos ofrece una opción compactada, el **“elif”**

Es decir, juntamos el **“else”** con el **“if”** de este código:

```
if a==b:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")
elif a > b:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")
print ("primera linea despues del if")
```

Y nos queda algo así:

```
if a==b:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")
elif a>b:
    print ("linea 1 del bloque verdadero")
    print ("linea 2 del bloque verdadero")
print ("primera linea despues del if")
```

Versión más compacta del condicional

Para cerrar, y solo con el afán de mostrarte que a veces las cosas se pueden complicar. Te

dejo una forma compacta para programar un *"if" en Python.*

```
numero=6  
#acá decimos que si numero es igual a 5  
respuesta= "El número es 5" if (numero==5) else "El número no es 5"  
print (respuesta)
```

El resultado es:

```
El número no es 5
```

Bibliografía

- Varó, A. M., Sevilla, P. G., & Luengo, I. G. (2014). Introducción a la programación con Python 3. Recuperado de <http://dx.doi.org/10.6035/Sapientia93>