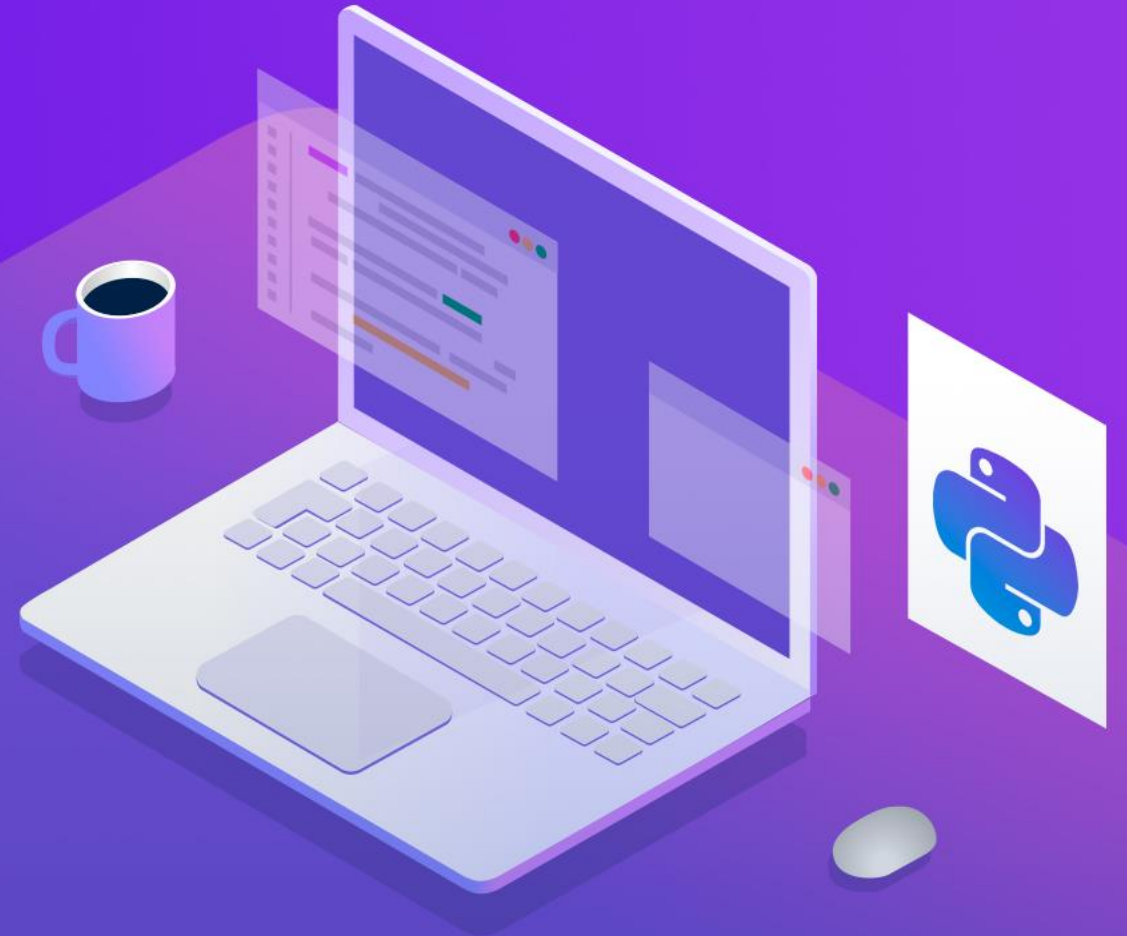


TELEDUC

Educación a Distancia

CONCEPTOS BÁSICOS DE PROGRAMACIÓN

>>> Parte 1: Operadores Lógicos



Recordemos

Ya revisamos algunos operadores básicos:

+	Suma los valores de la izquierda y la derecha.
-	Resta el valor de la derecha al valor de la izquierda.
*	Multiplica el valor de la izquierda por el de la derecha.
/	Divide el valor de la izquierda por el de la derecha.
**	Calcula la potencia. El valor de la izquierda es la base y el de la derecha es el exponente.
//	Divide el valor de la izquierda por el de la derecha, y entrega el valor entero del resultado.
%	Divide el valor de la izquierda por el de la derecha. No entrega el resultado, sino que el resto de esta división.



Python también tiene otro tipo de operadores, **los operadores lógicos**, y los revisaremos a continuación.

Operadores lógicos de comparación

Se llaman así porque sirven para comparar dos valores.
Revisemos algunos ejemplos:

CÓDIGO	RESULTADO
<code>print (9==9)</code>	True
<code>print (10==9)</code>	False
<code>print (9==0)</code>	False

== Compara el valor de la derecha y la izquierda

Si son iguales, retorna TRUE (que es la traducción al inglés de "verdadero") y si son distintos, retorna FALSE (que es la traducción al inglés de "falso").

Operadores lógicos de comparación

Se llaman así porque sirven para comparar dos valores.
Revisemos algunos ejemplos:

CÓDIGO	RESULTADO
<code>print (9!=9)</code>	False
<code>print (10!=9)</code>	True
<code>print (9!=0)</code>	True

!= Compara el valor de la derecha y la izquierda

Si son distintos, retorna TRUE y si son iguales, entrega FALSE.

Operadores lógicos de comparación

CÓDIGO	RESULTADO
<code>a="texto1"</code>	<code>False</code>
<code>b="texto2"</code>	<code>True</code>
<code>c="texto1"</code>	<code>True</code>
<code>print (a==b)</code>	<code>False</code>
<code>print (a!=b)</code>	
<code>print (a==c)</code>	
<code>print (a!=c)</code>	

Es muy importante notar que los operadores `==` y `!=` sirven también para comparar textos.

Operadores lógicos de comparación

Veamos un ejemplo (equivalente al anterior):

CÓDIGO	RESULTADO
<code>print (11<7)</code>	False
<code>print (7<7)</code>	False
<code>print (4<7)</code>	True

< Menor que

Compara el valor de la derecha y la izquierda. Si el de la izquierda es menor que la derecha, entonces retorna TRUE. Si el valor de la izquierda es mayor o igual al de la derecha, entonces retorna FALSE.

Operadores lógicos de comparación

Veamos un ejemplo (equivalente al anterior):

CÓDIGO	RESULTADO
<code>print(11>7)</code>	True
<code>print(7>7)</code>	False
<code>print(4>7)</code>	False

> Mayor que

Compara el valor de la derecha y la izquierda. Si el de la izquierda es mayor que la derecha, entonces retorna TRUE. Si el valor de la izquierda es menor o igual al de la derecha, entonces retorna FALSE.

Operadores lógicos de comparación

Veamos un ejemplo (equivalente al anterior):

CÓDIGO	RESULTADO
<code>print(11>=7)</code>	True
<code>print(7>=7)</code>	True
<code>print(4>=7)</code>	False

>= Mayor o igual que

Compara el valor de la derecha y la izquierda. Si el de la izquierda es mayor o igual que el de la derecha, entonces retorna TRUE. Si el valor de la izquierda es menor al de la derecha, entonces retorna FALSE.

Operadores lógicos de comparación

Veamos un ejemplo (equivalente al anterior):

CÓDIGO	RESULTADO
<code>print (11<=7)</code>	False
<code>print (7<=7)</code>	True
<code>print (4<=7)</code>	True

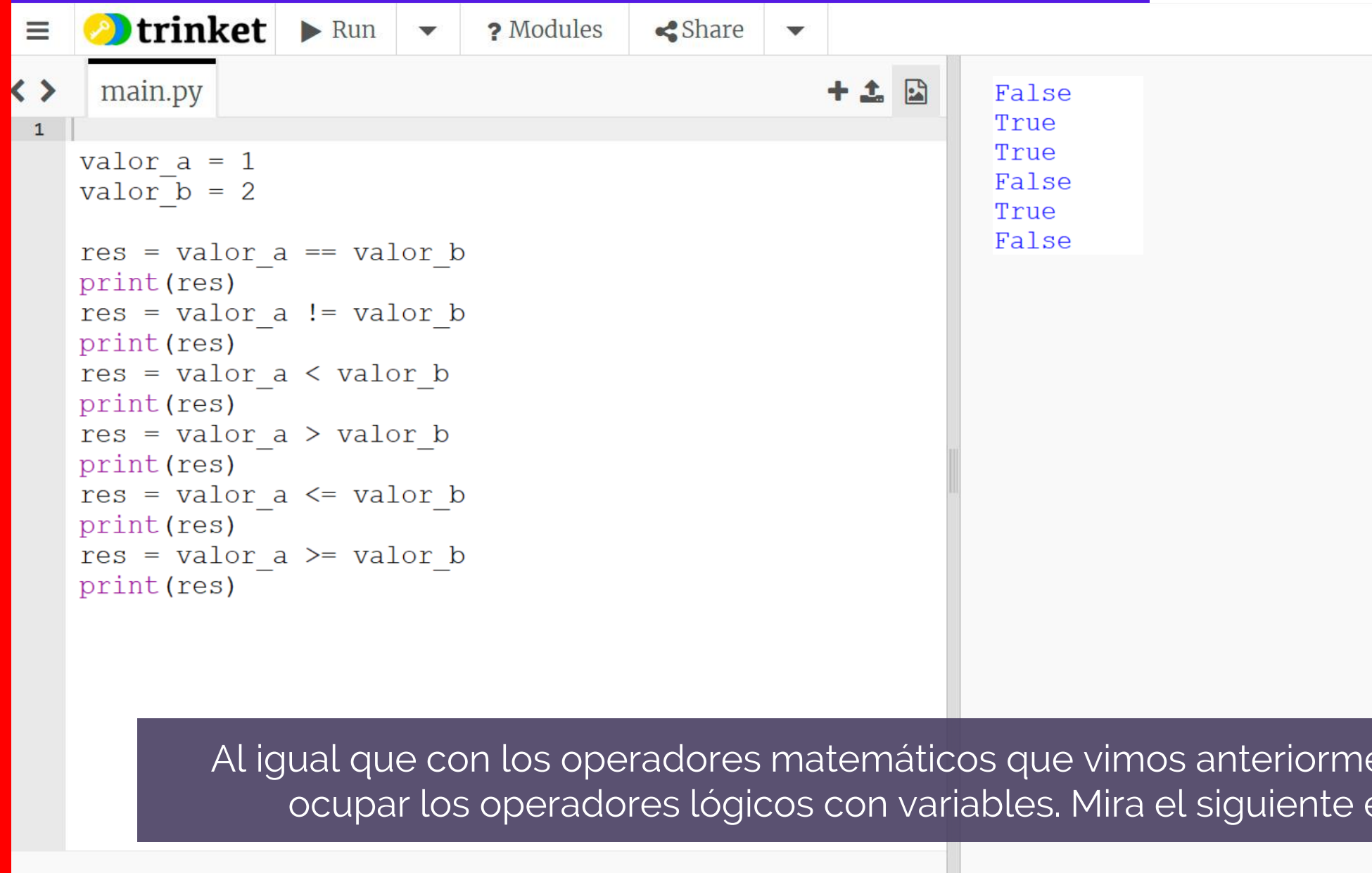
<= Menor o igual que

Compara el valor de la derecha y la izquierda. Si el de la izquierda es menor o igual que la derecha, entonces retorna TRUE. Si el valor de la izquierda es mayor al de la derecha, entonces retorna FALSE.

Operadores lógicos de comparación y variables

Recuerda revisar la
Ruta de ejercicios.

Ejercicio EM1-24 →



The screenshot shows the Trinket Python IDE interface. The top bar includes the Trinket logo, a 'Run' button, and links for 'Modules' and 'Share'. The main editor area displays a Python script in a file named 'main.py'. The script defines two variables, 'valor_a' and 'valor_b', with values 1 and 2 respectively. It then performs seven comparison operations, storing the results in 'res' and printing them. The output on the right shows the results of these operations: False, True, True, False, True, False, and False.

```
1 valor_a = 1
   valor_b = 2

   res = valor_a == valor_b
   print(res)
   res = valor_a != valor_b
   print(res)
   res = valor_a < valor_b
   print(res)
   res = valor_a > valor_b
   print(res)
   res = valor_a <= valor_b
   print(res)
   res = valor_a >= valor_b
   print(res)
```

False
True
True
False
True
False
False

Al igual que con los operadores matemáticos que vimos anteriormente, puedes ocupar los operadores lógicos con variables. Mira el siguiente ejemplo:

Operador lógico `not`

El operador lógico `not` sirve para poder “cambiar” el valor de una operación lógica. Es decir, si uno le aplica este operador a una operación lógica, cambia el valor de `TRUE` a `FALSE` o viceversa.

Por ejemplo:

CÓDIGO	RESULTADO
<code>print(3>=2)</code>	<code>True</code>
<code>print(not 3>=2)</code>	<code>False</code>
<code>print(1!=1)</code>	<code>False</code>
<code>print(not 1!=1)</code>	<code>True</code>

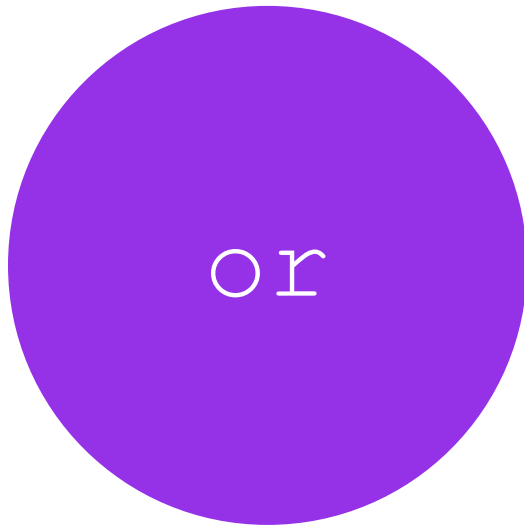
Volvamos al operador lógico \leq

Al hacer las siguientes operaciones:



Para que se cumpla la operación que dice el operador, se debe cumplir que sea menor o que sea igual. Si se cumple cualquiera de las dos condiciones, entonces la operación completa es verdadera.

Operadores lógicos binarios



Compara dos valores y entrega un resultado. Para esto, ocupa el operador `or`. Si al menos uno de los valores es `TRUE`, entonces el resultado total también lo es.

En esta tabla se resume lo anterior:

A	B	A <code>or</code> B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Veamos algunos ejemplos de `or` en Python:

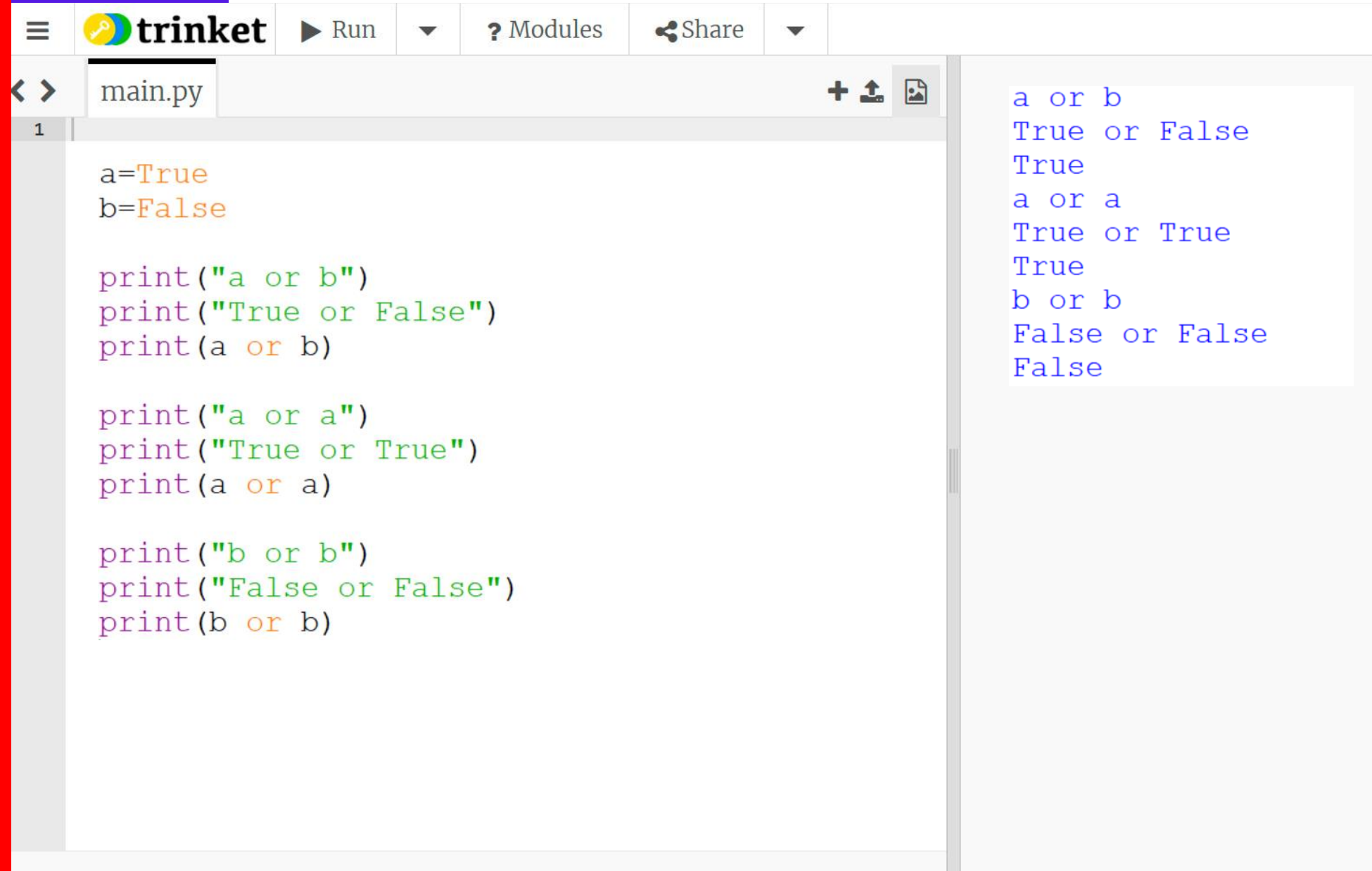
CÓDIGO	RESULTADO
<code>print(True or True)</code>	True
<code>print(True or False)</code>	True
<code>print(False or True)</code>	True
<code>print(False or False)</code>	False

Esto demuestra lo expuesto en la tabla anterior. No obstante, muy rara vez trabajaremos con los valores explícitos de `True` o `False`. En general trabajaremos con variables.

or

Recuerda revisar la
Ruta de ejercicios.

Ejercicio EM1-27 →



The screenshot shows the Trinket Python IDE interface. The top bar includes a menu icon, the Trinket logo, a 'Run' button, a 'Modules' dropdown, and a 'Share' button. Below the bar, the file 'main.py' is open. The code in the editor is as follows:

```
1
a=True
b=False

print("a or b")
print("True or False")
print(a or b)

print("a or a")
print("True or True")
print(a or a)

print("b or b")
print("False or False")
print(b or b)
```

The output on the right side of the IDE shows the results of the code execution:

```
a or b
True or False
True
a or a
True or True
True
b or b
False or False
False
```

¿Cuál es el uso práctico de or?

El verdadero uso práctico de esta función es que las variables con las que ocupamos `or` sean operaciones lógicas.

Por ejemplo, digamos que queremos saber si un número es menor a 3 o mayor que 10,

¿Cómo podríamos escribir un programa que resolviera este problema?

`or`

or

Recuerda revisar la
Ruta de ejercicios.

Ejercicio EM1-28 →

 **trinket** Run ? Modules Share
main.py
1

```
numero = int(input("Ingrese el número que desea comprobar que sea menor que 3 o mayor que 10\n"))  
menor_que_3 = numero < 3  
mayor_que_10 = numero > 10  
print(menor_que_3 or mayor_que_10)
```

Ingrese el número que desea comprobar que sea menor que 3 o mayor que 10

6

False

Ingrese el número que desea comprobar que sea menor que 3 o mayor que 10

27

True

and



Es un operador que compara dos valores y entrega un resultado. Si ambos valores son TRUE, entonces el resultado total también lo es.

En esta tabla se resume lo anterior:

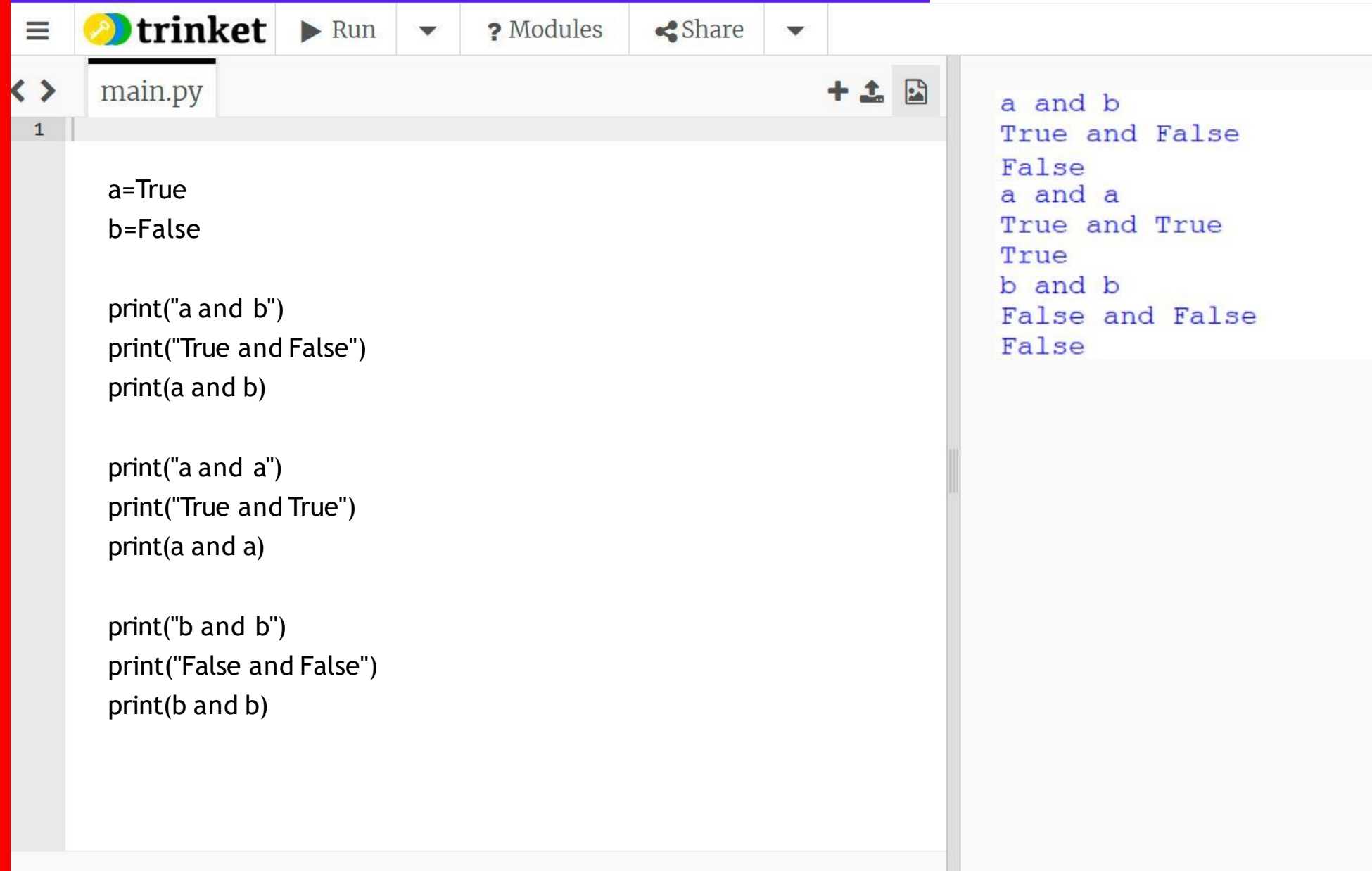
A	B	A and B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

Veamos algunos ejemplos de and

CÓDIGO	RESULTADO
<code>print(True and True)</code>	True
<code>print(True and False)</code>	False
<code>print(False and True)</code>	False
<code>print(False and False)</code>	False

Veamos algunos ejemplos sobre `and`

Recuerda revisar la
Ruta de ejercicios.
Ejercicio EM1-30 →



The image shows a screenshot of the Trinket Python IDE. The interface includes a top bar with the Trinket logo, a 'Run' button, and a 'Modules' dropdown. Below this is a file explorer showing 'main.py'. The main editor area contains the following Python code:

```
1 a=True
  b=False

  print("a and b")
  print("True and False")
  print(a and b)

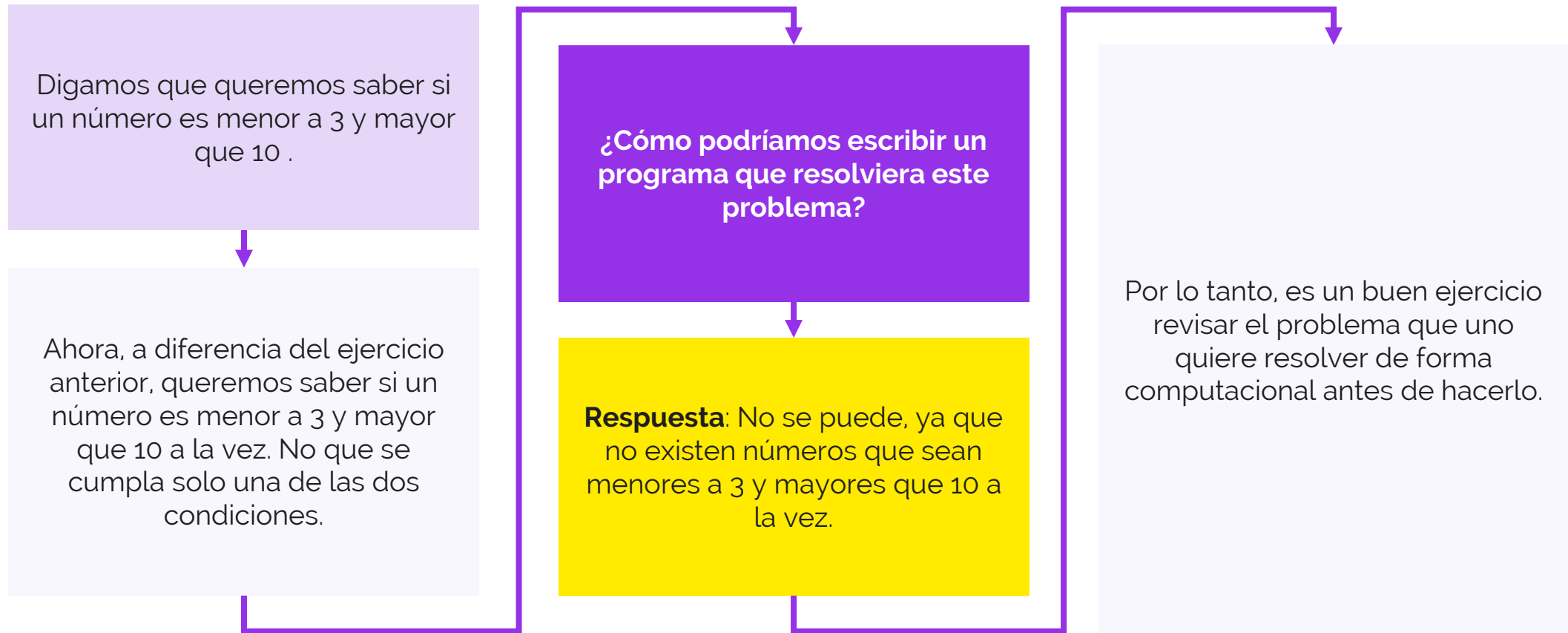
  print("a and a")
  print("True and True")
  print(a and a)

  print("b and b")
  print("False and False")
  print(b and b)
```

To the right of the editor, the output of the code is displayed:

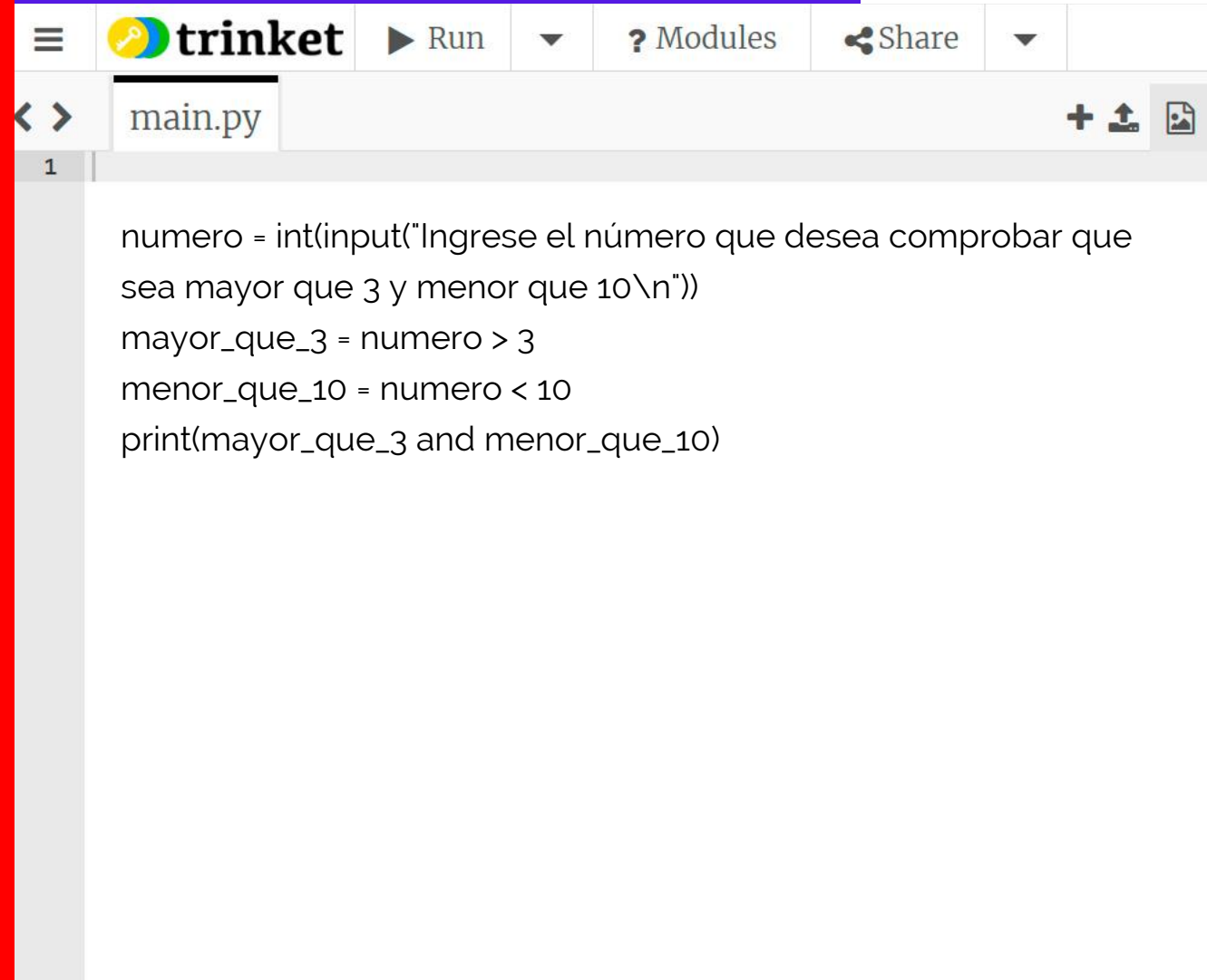
```
a and b
True and False
False
a and a
True and True
True
b and b
False and False
False
```

Ejercicio propuesto



Revisemos otro ejercicio

Recuerda revisar la
Ruta de ejercicios.
Ejercicio EM1-31 →



```
1 numero = int(input("Ingrese el número que desea comprobar que  
2 sea mayor que 3 y menor que 10\n"))  
3 mayor_que_3 = numero > 3  
4 menor_que_10 = numero < 10  
print(mayor_que_3 and menor_que_10)
```

```
Ingrese el número que desea comprobar que sea menor que 3 o mayor que 10  
5  
True
```

```
Ingrese el número que desea comprobar que sea menor que 3 o mayor que 10  
3  
False
```

```
Ingrese el número que desea comprobar que sea menor que 3 o mayor que 10  
10  
False
```

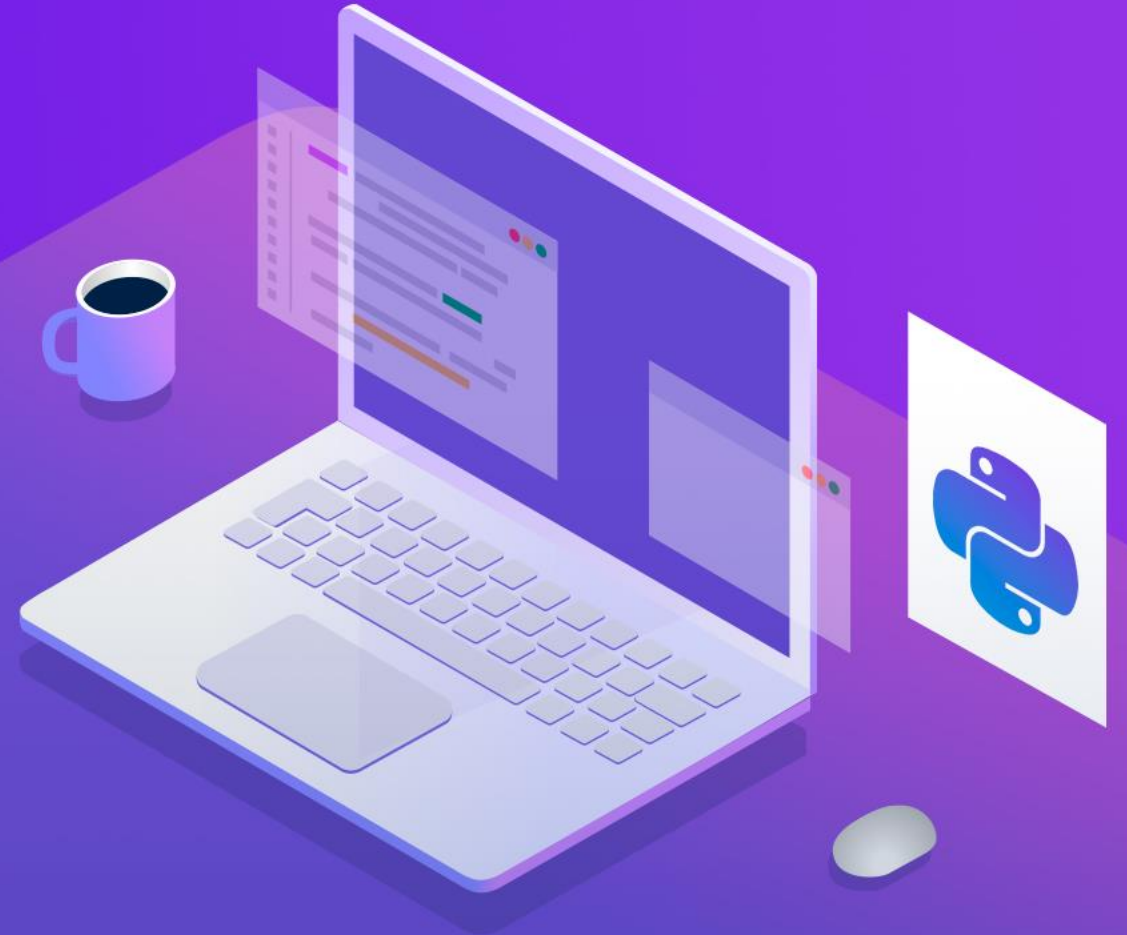
Ahora queremos saber si un número que un usuario ingresa es **mayor a 3** y **menor que 10**.

TELEDUC

Educación a Distancia

CONCEPTOS BÁSICOS DE PROGRAMACIÓN

>>> Parte 2: Control de flujo



Reflexionemos

Si retomamos uno de los ejercicios propuestos anteriormente, podemos proponer un nuevo flujo para nuestro código.

Si una persona ingresa un número que es mayor a 3 y menor que 10, entonces se debe imprimir en la consola:

“El número sí es mayor que 3 y menor que 10”.

En caso contrario, se debe imprimir en la consola:

“El número no es mayor que 3 y menor que 10”.

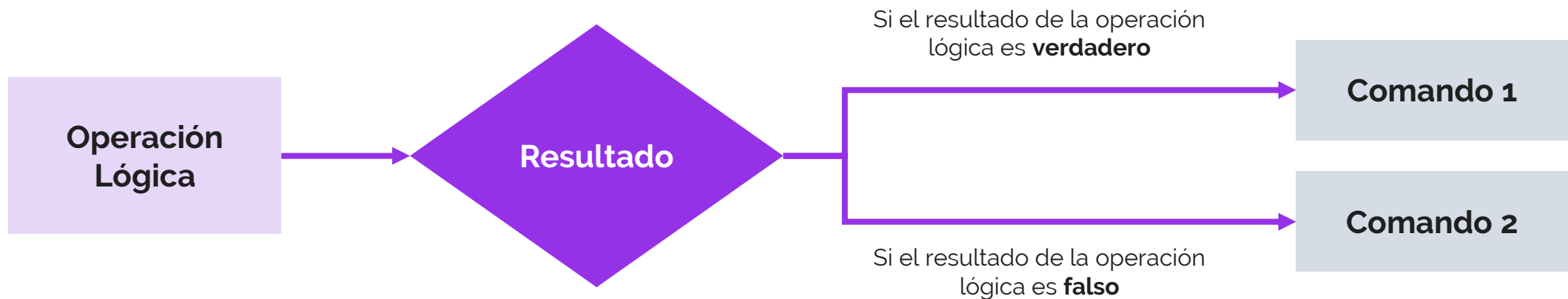
¿Podríamos “**simular**” este flujo en Python?



if

Para simular el flujo anterior, ocuparemos el comando `if`.

Éste sirve para poder ejecutar comandos de acuerdo al resultado de una operación lógica. Se puede caracterizar por el siguiente esquema:



if

En código Python se vería de la siguiente manera:

<code>if operación lógica:</code>	←	De acuerdo al resultado de esta operación lógica
<code> comando 1</code>	←	Si el resultado de la operación lógica es TRUE, se ejecuta el comando 1 (puede ser más de un comando)
<code>else:</code>		
<code> comando 2</code>	←	Si el resultado de la operación lógica es FALSE, se ejecuta el comando 2 (puede ser más de un comando)

if

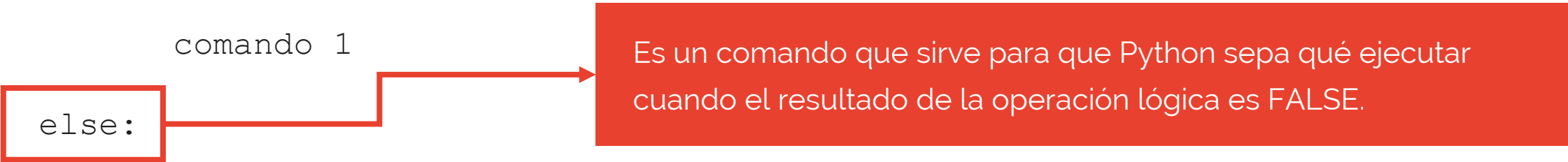
En código Python se vería de la siguiente manera:

```
if operación lógica:
```

comando 1

else:

comando 2



Es un comando que sirve para que Python sepa qué ejecutar cuando el resultado de la operación lógica es FALSE.

if

En código Python se vería de la siguiente manera:

```
if operación lógica:
```

```
    comando 1
```

```
else:
```

```
    comando 2
```

Pueden notar que en el código, comando 1 está "más adentro" que el `if` y el `else`. Esto no solo es ayuda visual, sino que es un aspecto sumamente importante.

Se denomina **indentación** y le está diciendo a Python que comando 1 está "dentro" del `if`, así como comando 2 está dentro del `else`.

De esta forma, Python sabe qué comando ejecutar de acuerdo al resultado de la operación lógica.

Siempre hay que respetar la indentación cuando se trabaje con `if` y `else`. Para añadir indentación, se debe ocupar la tecla *Tab*.

Ejemplo `if`

CÓDIGO	RESULTADO
<pre>if True: print("resultado de la operación cuando es True") else: print("resultado de la operación cuando es False")</pre>	<pre>resultado de la operación cuando es True</pre>

Casos prácticos de `if`

En general `if` y `else` se ocupan para verificar el valor de una variable, que puede:



Diagram illustrating two practical cases for using the `if` statement, represented by two purple circles. The first circle on the left contains the text 'Ingresar el usuario'. The second circle on the right contains the text 'Cambiar durante la ejecución del código'.

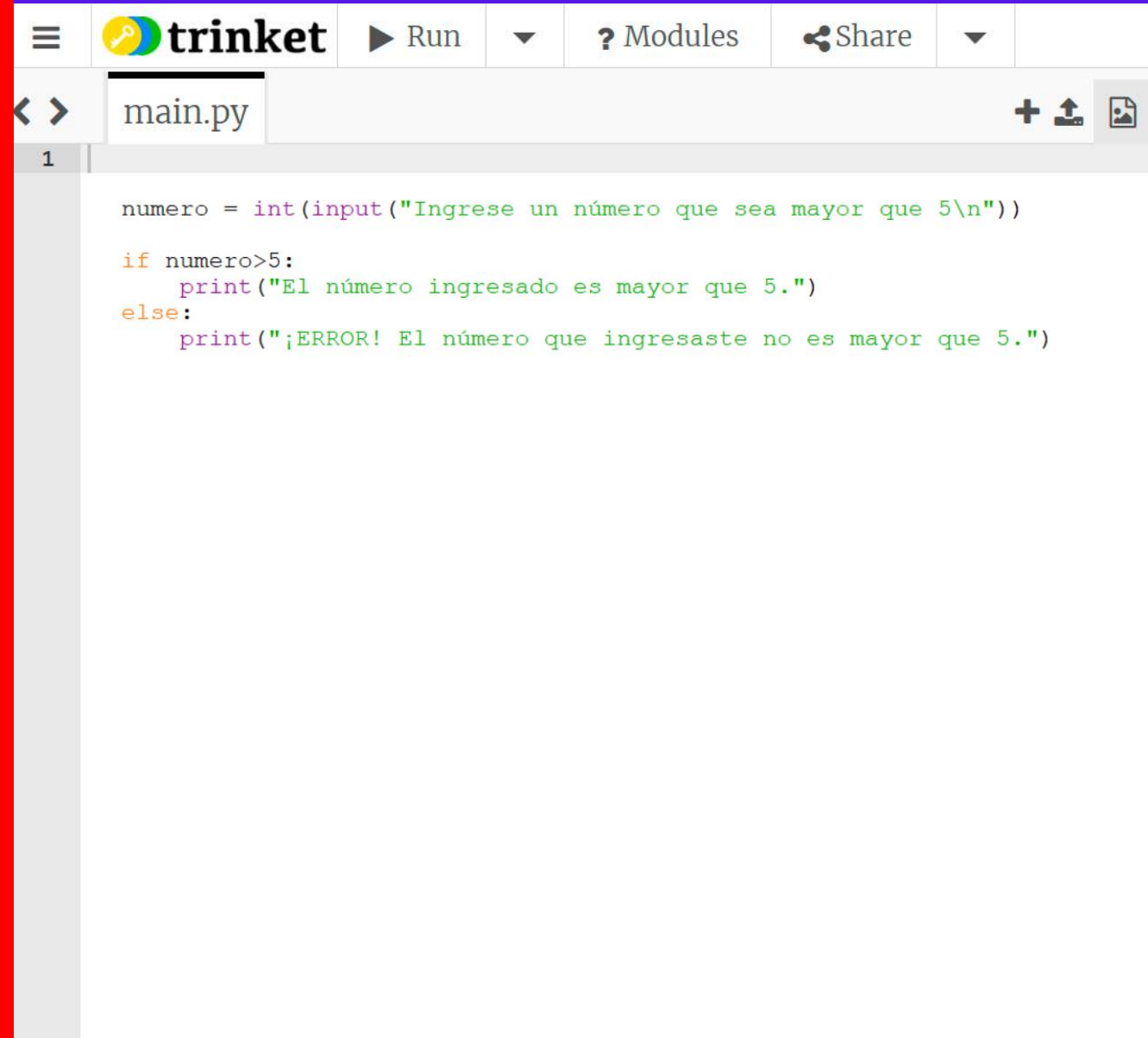
Ingresar el
usuario

Cambiar
durante la
ejecución del
código

Veamos un ejemplo para el primer caso

Recuerda revisar la
Ruta de ejercicios.

Ejercicio EM1-33 →



```
1
numero = int(input("Ingrese un número que sea mayor que 5\n"))

if numero>5:
    print("El número ingresado es mayor que 5.")
else:
    print(";ERROR! El número que ingresaste no es mayor que 5.")
```

Ingrese un número que sea mayor que 5

9

El número ingresado es mayor que 5.

Ingrese un número que sea mayor que 5

1

;ERROR! El número que ingresaste no es mayor que 5.

Veamos un ejemplo para el segundo caso

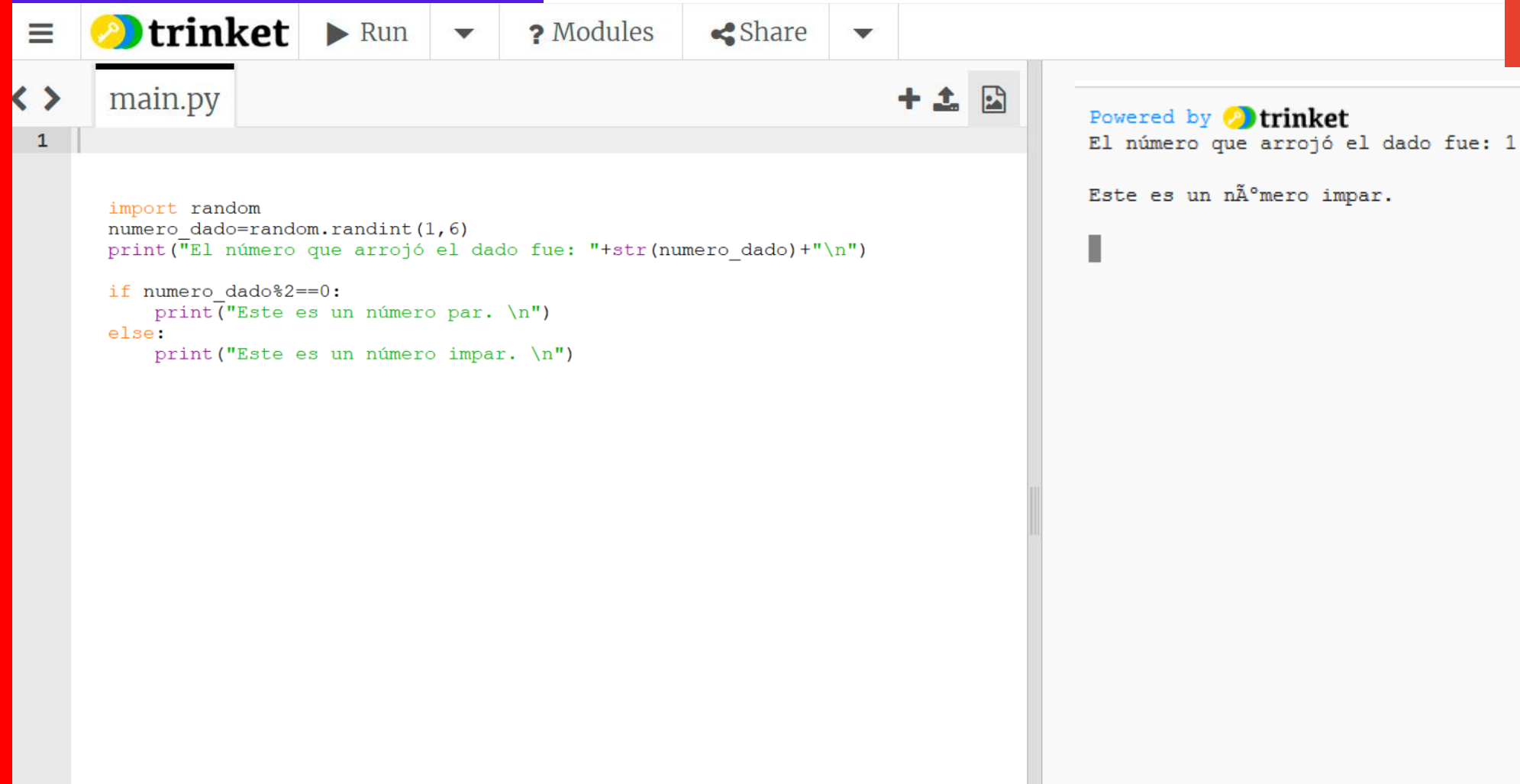
Para el segundo caso, imaginemos que queremos simular el lanzamiento de un dado. Queremos saber si el número que salió en el lanzamiento es par o impar.



Solución ejercicio


Recuerda revisar la
Ruta de ejercicios.

Ejercicio EM1-34 →



```
import random
numero_dado=random.randint(1,6)
print("El número que arrojó el dado fue: "+str(numero_dado)+"\n")

if numero_dado%2==0:
    print("Este es un número par. \n")
else:
    print("Este es un número impar. \n")
```

Powered by  trinket

El número que arrojó el dado fue: 1

Este es un número impar.

Esta solución tiene varios elementos nuevos e interesantes.
Los analizaremos uno a uno a continuación.

Analicemos la solución del ejercicio

Estos dos comandos sirven para poder generar un número aleatorio entre 1 y 6.

Veamos en qué consisten ambos.

```
import random  
numero_dado=random.randint(1,6)
```

En general, los lenguajes de programación trabajan con paquetes, son algoritmos y funcionalidades.



Es posible añadir (o **importar**, que es el término exacto en este caso) estos paquetes para aportar funcionalidades extras a tu código.



En este caso, trabajaremos con el paquete `random` que sirve para poder generar números aleatorios.

Analicemos la solución del ejercicio

Estos dos comandos sirven para poder generar un número aleatorio entre 1 y 6.

Veamos en qué consisten ambos.

```
import random  
numero_dado=random.randint(1,6)
```

Para generar un número aleatorio, debes escribir:

```
random.randint(límite inferior, límite superior)
```



Donde límite inferior y límite superior indican entre qué números quieres acotar las posibilidades de generar el número aleatorio. Este número debes asignarlo a alguna variable, porque sino no podrás ocuparlo después.

Continuemos analizando la respuesta:

```
numero_dado=random.randint(1,6)
print("El número que arrojó el dado fue: "+str(numero_dado)+"\n")
```

Asumamos entonces que `numero_dado` es una variable que contiene un número aleatorio entre 1 y 6. Lo que queremos hacer con la línea de código que sigue es poder mostrar ese número en la consola.

Para hacerlo, podemos ocupar el comando `print` que vimos anteriormente. Sin embargo, este no se puede ocupar directamente con la variable.

Anteriormente, vimos el comando `int()`. Éste sirve para decirle a Python que el texto que ingresaba un usuario lo transformara a número.

En este caso, hacemos una operación similar. Tomamos el número aleatorio entre 1 y 6, y para poder imprimirlo en la consola (como un texto), tenemos que decirle a Python que lo transforme a texto.

Para esto, ocupamos el comando `str()`, que toma lo que está dentro del paréntesis y lo transforma a un texto.

Continuemos analizando la respuesta:

```
numero_dado=random.randint(1,6)
print("El número que arrojó el dado fue: "+str(numero_dado)+"\n")
```

Notemos también que para poder unir este texto con otros, ocupamos el operador +.

Es decir, si se quiere unir distintos textos (ya sea de forma literal o por medio de variables), se puede usar el comando +

CÓDIGO	RESULTADO
<pre>variable1="texto1" variable2="texto2" print("Podemos unir "+variable1+" con "+variable2+" de esta manera")</pre>	<pre>Podemos unir texto1 con texto2 de esta manera</pre>

Continuemos analizando la respuesta:

Antes de analizar el `if`, detengámonos en la operación lógica. Puedes observar que ocupamos el operador `%`, que vimos anteriormente. Recordemos que este operador entrega el resto de una división.

Una buena aplicación de este operador es para saber si un número es divisible por otro. En este caso, queremos saber si el número del dado es divisible por 2. Si la división de un número por 2 da resto igual a 0, entonces es una división exacta y el número es par. Por lo tanto, comparamos el resultado de esta operación con 0.

```
if numero_dado%2==0:
    print("Este es un número par. \n")
else:
    print("Este es un número impar. \n")
```

Continuemos analizando la respuesta:

```
if numero_dado%2==0:  
    print("Este es un número par. \n")  
else:  
    print("Este es un número impar. \n")
```

La operación lógica del `if` evalúa si el resto de la división del número del dado por 2 es igual a 0. Como explicamos antes, esto indicaría que el número es par.

Entonces, cuando esto es `TRUE`, se imprime en la consola "Este es un número par."

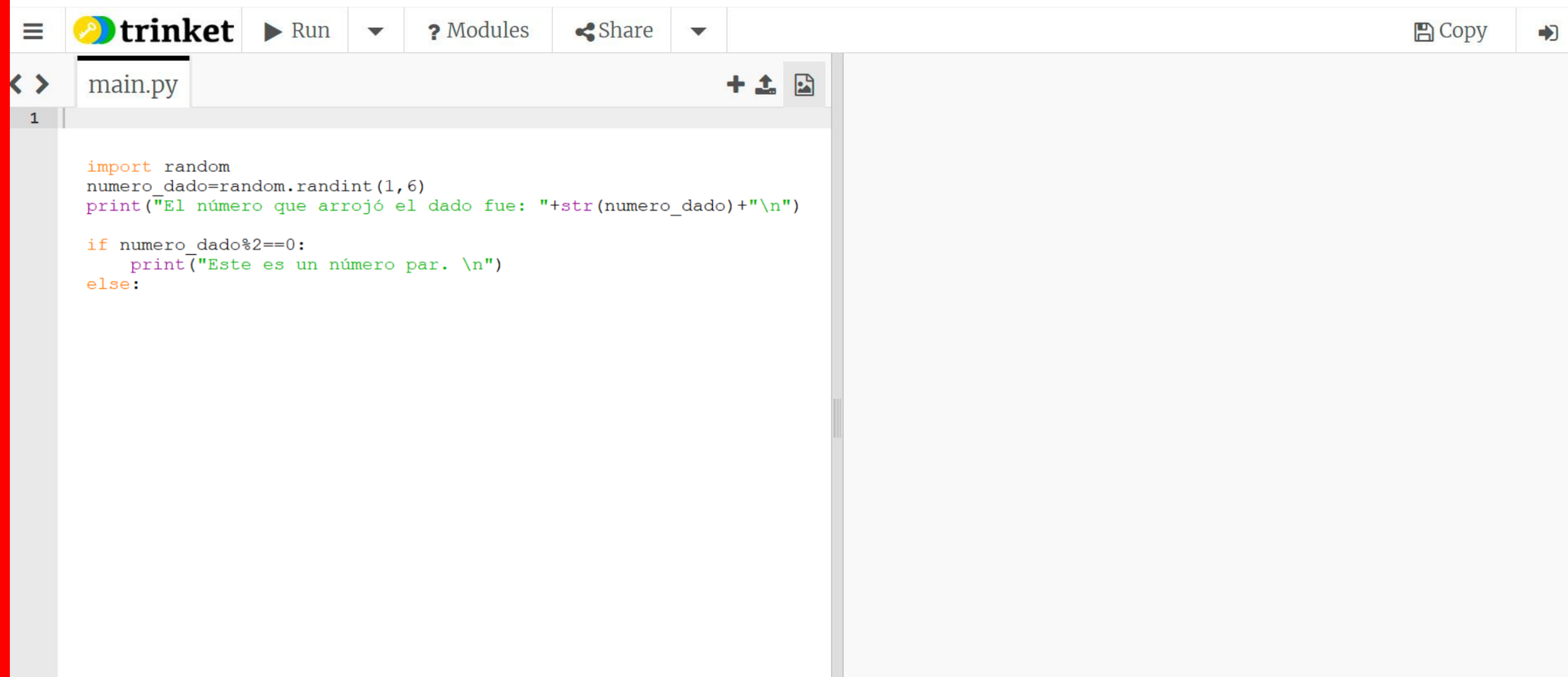
Y cuando es `FALSE`, se imprime "Este es un número impar".

Revisemos un caso particular

Digamos que en el mismo ejercicio anterior, **SOLO** queremos imprimir en la consola cuando el número es par. Es decir, cuando el número que arroja el dado es impar no queremos hacer nada

¿Cómo podríamos modificar la solución al ejercicio anterior para poder lograrlo?





```
import random
numero_dado=random.randint(1,6)
print("El número que arrojó el dado fue: "+str(numero_dado)+"\n")

if numero_dado%2==0:
    print("Este es un número par. \n")
else:
```

La solución obvia sería dejar el `else` vacío. No obstante, no es la solución más eficiente.

Recuerda revisar la
Ruta de ejercicios.

Ejercicio EM1-34 →



The screenshot shows the Trinket Python IDE interface. The top bar includes the Trinket logo, a 'Run' button, a 'Modules' dropdown, and a 'Share' button. Below the bar, the file 'main.py' is open. The code in the editor is as follows:

```
1
import random
numero_dado=random.randint(1,6)
print("El número que arrojó el dado fue: "+str(numero_dado)+"\n")

if numero_dado%2==0:
    print("Este es un número par. \n")
```

At the bottom of the image, there are two text boxes. The first is a dark purple box with the text: "Se puede escribir un `if` sin un `else`. Y esta sería la mejor solución." The second is a red box with the text: "❗ **Importante:** Puedes escribir un `if` sin un `else` pero no un `else` sin un `if`."

Propongamos un nuevo caso

Digamos que queremos asignar aleatoriamente a personas de una empresa a tres grupos. Los tres grupos se denominarán "Alerce", "Boldo" y "Cerezo".

Para eso, generaremos un número aleatorio entre uno y tres.

Si el número generado es 1, entonces asignamos a la persona al grupo "Alerce".

Si el número generado es 2, entonces asignamos a la persona al grupo "Boldo".

Si el número generado es 3, entonces asignamos a la persona al grupo "Cerezo".

¿Podemos resolver esto solo con `if-else`? Casi...



if anidados

Es posible escribir una secuencia de `if-else` (o solo `if`) dentro de otro. Esto sirve para poder generar más casos, pero es poco eficiente en cuanto a líneas de códigos y en especial si se quieren hacer muchos casos.

Una solución posible para el ejercicio propuesto anteriormente podría ser la siguiente:

Recuerda revisar la Ruta de ejercicios.

Ejercicio EM1-35 →



The screenshot shows the Trinket Python IDE interface. At the top, there is a navigation bar with a menu icon, the Trinket logo, a 'Run' button, a dropdown arrow, a 'Modules' button with a question mark, and a 'Share' button with a share icon. Below this is a file explorer showing 'main.py'. The main editor area contains the following Python code:

```
1 import random
   numero_aleatorio = random.randint(1,3)
   nombre_persona = input("Ingresa tu nombre para poder asignarte a
   un grupo.")

   if numero_aleatorio == 1:
       print(nombre_persona + " fuiste asignado/a al grupo ALERCE\n")
   elif numero_aleatorio == 2:
       print(nombre_persona + " fuiste asignado/a al grupo BOLDO\n")
   else:
       print(nombre_persona + " fuiste asignado/a al grupo CEREZO\n")
```

if-elif-else

Para poder hacer una solución más eficiente del caso anterior, podemos ocupar `elif`.

A diferencia del `else`, que no lleva ninguna operación lógica, `elif` tiene una operación lógica distinta al `if`.

Se pueden ocupar todos los `elif` que se deseen.

La estructura general es la siguiente:

```
if operación lógica 1:
    comando 1
elif operación lógica 2:
    comando 2
...
elif operación lógica N:
    comando N
else:
    comando 2
```

Volvamos al caso anteriormente expuesto

Digamos que queremos asignar aleatoriamente a personas de una empresa a tres grupos. Los tres grupos se denominarán "Alerce", "Boldo" y "Cerezo".

Para eso, generaremos un número aleatorio entre uno y tres.

Si el número generado es 1, entonces asignamos a la persona al grupo **"Alerce"**.

Si el número generado es 2, entonces asignamos a la persona al grupo **"Boldo"**.

Si el número generado es 3, entonces asignamos a la persona al grupo **"Cerezo"**.

Resolvamos este ejercicio con `if-elif-else`

Recuerda revisar la
Ruta de ejercicios.

Ejercicio EM1-36 →

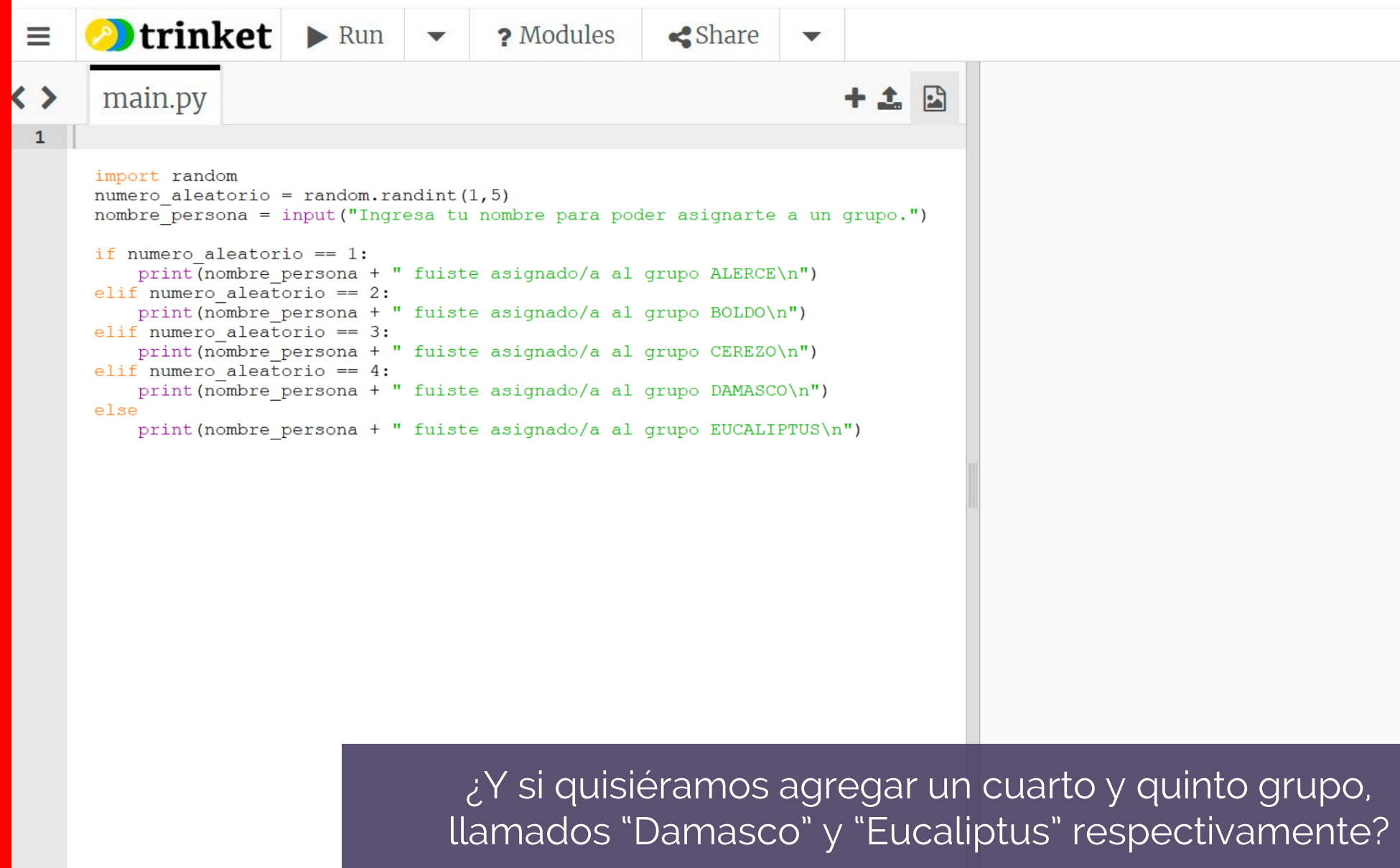


The image shows the Trinket Python IDE interface. At the top, there is a navigation bar with a menu icon, the Trinket logo, a 'Run' button, a dropdown arrow, a 'Modules' button with a question mark, and a 'Share' button with a share icon. Below this is a tab labeled 'main.py' with navigation arrows on the left and icons for adding files, uploading, and viewing on the right. The main area contains a Python script starting at line 1. The script imports the 'random' module, generates a random integer between 1 and 3, and prompts the user for their name. It then uses an if-elif-else structure to assign the user to one of three groups: ALERCE, BOLDO, or CEREZO.

```
1
import random
numero_aleatorio = random.randint(1,3)
nombre_persona = input("Ingresa tu nombre para poder asignarte a un grupo.")

if numero_aleatorio == 1:
    print(nombre_persona + " fuiste asignado/a al grupo ALERCE\n")
elif numero_aleatorio == 2:
    print(nombre_persona + " fuiste asignado/a al grupo BOLDO\n")
else:
    print(nombre_persona + " fuiste asignado/a al grupo CEREZO\n")
```


Recuerda revisar la
Ruta de ejercicios.
Ejercicio EM1-37 →



The screenshot shows the Trinket Python IDE interface. At the top, there is a navigation bar with a menu icon, the Trinket logo, a 'Run' button, a dropdown arrow, a '? Modules' button, a 'Share' button, and another dropdown arrow. Below this is a file explorer showing 'main.py'. The main editor area contains the following Python code:

```
1
import random
numero_aleatorio = random.randint(1,5)
nombre_persona = input("Ingresa tu nombre para poder asignarte a un grupo.")

if numero_aleatorio == 1:
    print(nombre_persona + " fuiste asignado/a al grupo ALERCE\n")
elif numero_aleatorio == 2:
    print(nombre_persona + " fuiste asignado/a al grupo BOLDO\n")
elif numero_aleatorio == 3:
    print(nombre_persona + " fuiste asignado/a al grupo CEREZO\n")
elif numero_aleatorio == 4:
    print(nombre_persona + " fuiste asignado/a al grupo DAMASCO\n")
else
    print(nombre_persona + " fuiste asignado/a al grupo EUCALIPTUS\n")
```

At the bottom of the image, there is a dark blue text box with the following text:

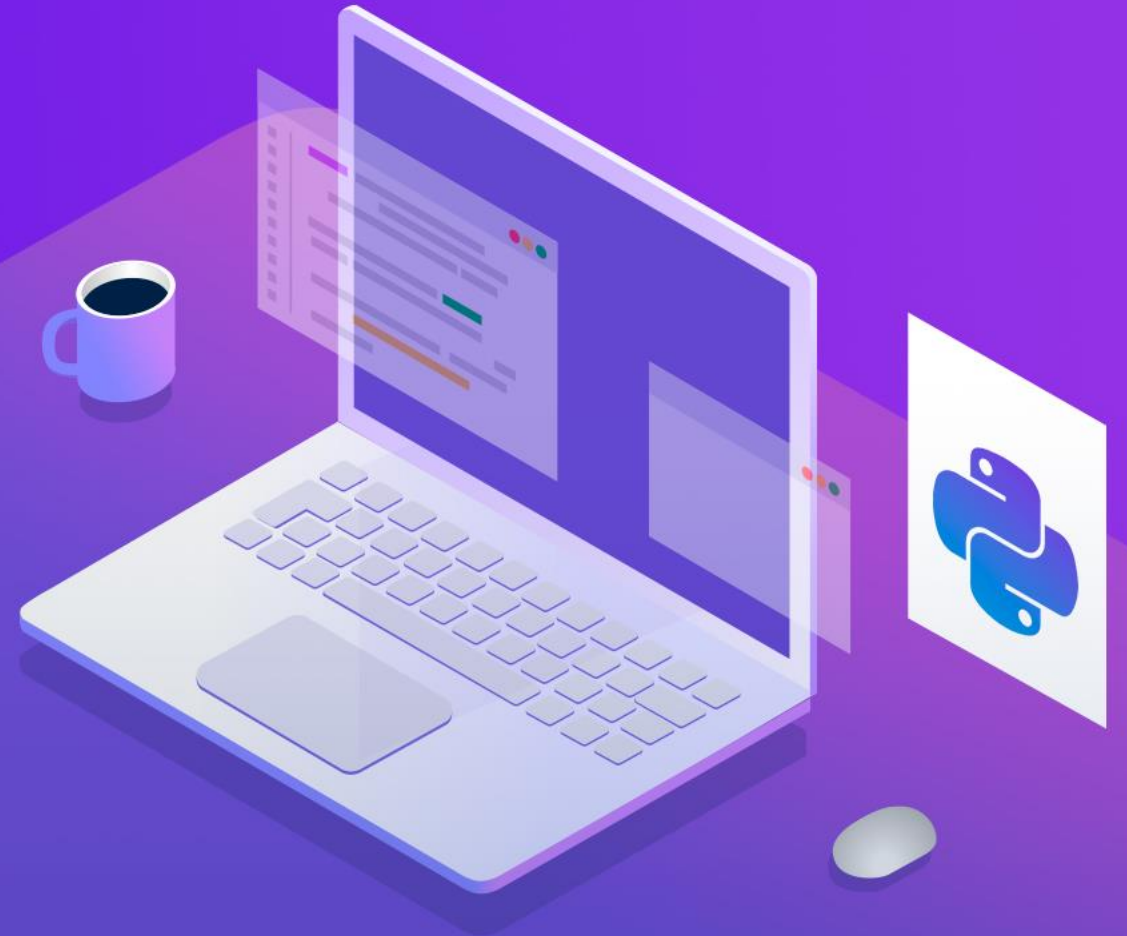
¿Y si quisiéramos agregar un cuarto y quinto grupo, llamados "Damasco" y "Eucaliptus" respectivamente?

TELEDUC

Educación a Distancia

CONCEPTOS BÁSICOS DE PROGRAMACIÓN

>>> Parte 3: Ciclos



Revisemos desde otra perspectiva (los ciclos) el caso

Si ejecutamos la solución propuesta anteriormente, es útil ya que sirve para poder asignar una persona aleatoriamente a alguno de los grupos.

Sin embargo:

¿Cómo podríamos hacerlo para N personas?

Ejecutar el código cada vez que una persona quiera estar asignada a un grupo no es muy eficiente (imaginen si se quisiera asignar a 1000 personas aleatoriamente a alguno de estos grupos).



Ciclos

Python posee herramientas para poder ejecutar el mismo código un determinado número de veces.

Esto se hace a través del comando `while`.

Al igual que `if`, funciona con una operación lógica.

Se evalúa al "entrar" al ciclo, es decir, si la operación lógica se cumple, se ejecuta el ciclo por primera vez.

Si la operación lógica es falsa, entonces el ciclo termina.

Si la operación lógica se cumple, o es verdadera, el código se repite.

Ciclos

La estructura general de un `while` es la siguiente:

```
while operación lógica:  
    comando 1
```

Ciclos

Revisemos un ejemplo:

```
while True:  
    print("Hola \n")
```

Hay que responder dos preguntas:

1

¿Se va a ejecutar el ciclo?

Sí, porque la operación lógica es verdadera

2

¿Cuántas veces se ejecutará?

Infinitas veces, porque la operación lógica siempre será verdadera

Responder estas preguntas es
siempre un buen ejercicio
cuando se trabaja con ciclos

Ciclos

Revisemos otro ejemplo:

```
while False:  
    print("Hola \n")
```

Hay que responder dos preguntas:

1

¿Se va a ejecutar el ciclo?

No, porque la operación lógica es verdadera

2

¿Cuántas veces se ejecutará?

0 veces, ya que la operación lógica es falsa y nunca pasará a ser verdadera

Revisemos un caso

En este caso, queremos imprimir en consola los números del 1 al 20.

Escribir 20 veces `print("1")`, `print("2")`, y así sucesivamente, claramente no es eficiente.

Por lo tanto, ocuparemos un ciclo para poder hacerlo de una forma más fácil.

1
20

Definimos una variable que será la que aumenta y se imprime en la consola

```
variable_numerica = 1
```

```
while variable_numerica <= 20:
```

```
    print(str(variable_numerica))
```

```
    variable_numerica = variable_numerica + 1
```

La operación lógica es que
variable_numérica sea menor o
igual que 20.

En esta línea imprimimos en la
consola la variable que va
aumentando con cada ciclo.

Notemos que `variable_numerica` parte en 1, y como es la variable que se está imprimiendo en consola necesitamos que aumente en 1. Con este código se logra lo anterior.

Así, el ciclo se ejecuta 20 veces, y así es posible imprimir en consola los números del 1 al 20.

Analicemos la solución del ejercicio

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1
```

Podemos volver a responder estas dos importantes preguntas:

1

¿Se va a ejecutar el ciclo?

Sí, porque la operación lógica es verdadera. Su valor es 1 y como 1 es menor o igual que 20, entonces sí se cumple

2

¿Cuántas veces se ejecutará?

20 veces, dado que la operación lógica se cumple mientras la `variable_numerica` sea menor o igual que 20. Cuando el valor de ésta es igual a 21, entonces la operación lógica ya no se cumple y el ciclo se deja de ocupar.

Analicemos la solución del ejercicio paso a paso

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1

print("terminó el conteo")
```

- | | |
|---|---|
| 1 | Se asigna un 1 a variable_numerica |
| 2 | Se evalúa la operación lógica; 1 es menor o igual que 20, lo que es verdadero, por lo tanto se entra "dentro" del ciclo |
| 3 | Se imprime el 1 en la consola |

Analicemos la solución del ejercicio paso a paso

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1

print("terminó el conteo")
```

- | | |
|---|--|
| 4 | variable_numerica aumenta en 1, ahora es 2. |
| 5 | Se evalúa la operación lógica; 2 es menor o igual que 20, lo que es verdadero, por lo tanto se entra "dentro" del ciclo. |
| 6 | Se imprime el 2 en la consola |

Analicemos la solución del ejercicio paso a paso

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1

print("terminó el conteo")
```

- | | |
|---|---|
| 7 | variable_numerica aumenta en 1, ahora es 3. |
| 8 | (... se repiten los pasos 5, 6 y 7 hasta que variable_numerica es 19) |
| 9 | Se evalúa la operación lógica; 19 es menor o igual que 20, lo que es verdadero, por lo tanto se entra "dentro" del ciclo. |

Analicemos la solución del ejercicio paso a paso

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1

print("terminó el conteo")
```

10	Se imprime el 19 en la consola
11	variable_numerica aumenta en 1, ahora es 20.
12	Se evalúa la operación lógica; 20 es menor o igual que 20, lo que es verdadero, por lo tanto se entra "dentro" del ciclo.

Analicemos la solución del ejercicio paso a paso

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1

print("terminó el conteo")
```

- | | |
|----|---|
| 13 | Se imprime el 20 en la consola |
| 14 | variable_numerica aumenta en 1, ahora es 21. |
| 15 | Se evalúa la operación lógica; 21 NO es menor o igual que 20. La operación lógica es falsa, por lo tanto no entra dentro del ciclo. |

Analicemos la solución del ejercicio paso a paso

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1

print("terminó el conteo")
```

16

Se imprime en consola "terminó el conteo".

Ciclos

Hay veces que no sabemos a priori cuándo debería terminar el ciclo.

En general se ocupa cuando esperamos que el usuario ingrese cierta información.

Por ejemplo, escribamos un programa que imprima en consola un texto hasta que el usuario indique lo contrario.


aa

Recuerda revisar la
Ruta de ejercicios.

Ejercicio EM1-41 →



```
main.py
1
input_del_usuario = input("Escribe 1 para que termine el programa")
while input_del_usuario != "1":
    input_del_usuario = input("Escribe 1 para que termine el programa")
```

Powered by  **trinket**
Escribe 1 para que termine el programa

Desde la primera vez que se le pide el input al usuario, además de cada iteración del ciclo, se verifica si el usuario ha ingresado el valor "1". En caso contrario, la operación lógica del `while` se cumple y se vuelve a ejecutar una nueva iteración.

Ciclos

Si por alguna razón extra a la operación lógica de un `while` uno quisiera terminarlo, uno puede ocupar el comando `break`.

Por ejemplo,

```
while True:  
    print("test")  
    break
```

```
print("test2")
```

En este ejemplo, la operación lógica es verdadera (y siempre lo es ya que no varía). Al entrar a la primera iteración del ciclo se imprime en consola "test". Sin embargo, y por el comando `break` se termina el `while` y se imprime en consola "test2".

Existe un caso especial del `while`, denominado `for`

Una de las ventajas de este comando es que tiene integrado un índice que va aumentando, desde un límite inferior a un límite superior. Sirve especialmente para poder ejecutar un código un cierto número de veces.

La estructura general de un `for` es:

```
for i in range(a,b):  
    comando 1
```

Donde:

`i` es la variable que incrementa

`a` es el valor inicial de `i`

`i` aumenta hasta `b-1`

Retomemos uno de los ejemplos anteriores

Ejercicio EM1-40

```
variable_numerica = 1

while variable_numerica <= 20:
    print(str(variable_numerica))
    variable_numerica = variable_numerica + 1
```

Ejercicio EM1-43

```
for variable_numerica in range(1,21):
    print(str(variable_numerica))
```

Se puede observar que se cumple lo mismo que el ejemplo del while anterior. En este caso, variable_numérica parte en el valor 1 y aumenta hasta llegar a 20.

>>> Cierre

Has finalizado la revisión de los contenidos de esta clase.

A continuación, te invitamos a realizar las actividades y a revisar los recursos del módulo que encontrarás en plataforma.