

Python para principiantes

Módulo B

1. Empezando
 - Escribir Código
 - Memoria y Variables
 - Datos de texto
 - Datos Numéricos
 - Trabajando con Variables
 - Módulo 1 Prueba
2. Profundizando
 - Debugging
 - Estándares y Mejores Prácticas
 - Aplicando las mejores prácticas
 - Entradas y Salidas
 - Tipos de datos
 - Módulo 2 Prueba
3. Trabajando con Datos
 - Verificación de Datos
 - Conversión de datos
 - Corrigiendo datos
 - Operaciones de Comparación
 - Operaciones Lógicas
 - Combinación de operaciones de comparación y lógicas
 - Módulo 3 Prueba
4. Control de flujo
 - Control de flujo
 - Bucles For
 - Bucles While
 - Más sobre Iteración
 - Declaraciones condicionales
 - Más sobre Declaraciones Condicionales
 - Módulo 4 Prueba
5. Trabajando con Listas
 - Listas
 - Indexación
 - Uso de la indexación
 - Rebanado
 - Uso de la segmentación
 - Segmentación e indexación avanzada
 - Módulo 5 Prueba

- 6. Funciones
 - Funciones
 - Argumentos de funciones
 - Funciones de cadenas
 - Funciones de listas
 - Funciones personalizadas
 - Más sobre funciones personalizadas
 - Módulo 6 Prueba

Módulo B

Empezando

Python es un lenguaje de programación muy conocido, sencillo y potente que se utiliza en software y desarrollo web, así como en ciencia de datos, aprendizaje automático y muchos otros campos. En este curso, cubriremos los aspectos básicos de Python, crearemos proyectos reales y solucionaremos varios retos de programación. Python para principiantes no requiere experiencia previa en programación, ¡así que no esperes más!

1. Escribir Código

Python es uno de los lenguajes más populares y útiles utilizados para dar instrucciones a las computadoras.

Python tiene una sintaxis simple, lo que significa que es fácil de escribir, leer y aprender.

```
print("Welcome")
```

La instrucción `print()` es la forma más fácil de enviar un mensaje a la pantalla u otro dispositivo de visualización.

```
print("Level Up!")
```

La instrucción `print()` requiere el uso de paréntesis alrededor del mensaje.

- Los humanos usan código para dar instrucciones a las máquinas.
- Asegúrate de usar comillas alrededor de los mensajes de texto. (Si te olvidas de las comillas alrededor del texto, la computadora no entenderá tus instrucciones.)
- Todo comienza con una línea de código.
- Los números no requieren comillas.

Python se utiliza ampliamente para construir software y juegos. analizar datos y programar inteligencia artificial. Aprendiste que:

☀ Los humanos usan código para dar instrucciones a las máquinas

☀ La instrucción `print()` muestra un mensaje en la pantalla

2. Memoria y Variables

Los programas informáticos utilizan variables para recordar información importante, como los artículos en un carrito de compras, precios y descuentos.

La línea de código a continuación le indica a la computadora que almacene información en una variable llamada item.

```
item = "bike"
```

- Para crear una variable, solo necesita darle un nombre.
- La información que necesita almacenar se agrega a la derecha.
- Las variables tienen un nombre y un valor. Están conectados con el signo igual =.

Completa el código para almacenar phone como información en una variable llamada item

Puedes pensar en una variable como una caja que contiene alguna información.



Los humanos usan código para dar instrucciones a las computadoras, pero el código también será leído por otros humanos (¡y por ti en el futuro!).

La línea de código a continuación le indica a la computadora que almacene información en la memoria.

```
message = "Level Up"
```

Identifique los elementos de la variable

nombre: -mensaje

valor: -"Level up"

¿Verdadero o falso? Las computadoras pueden almacenar y recuperar información más rápido y con mayor precisión que los humanos.

Trabajo increíble! Aprendió que:

☀ Los programas informáticos utilizan variables para recordar información importante

☀ Una variable tiene un nombre y un valor

☀ Puede crear una variable conectando el nombre y el valor con un signo igual =

3. Datos de texto

Una gran cantidad de información consiste en texto. Un dato de texto se llama cadena.

- Las cadenas (o strings) en Python deben estar rodeadas por comillas. Usamos comillas para indicarle a Python que estamos trabajando con un dato de texto.
- Las cadenas se pueden almacenar en variables.

Una variable está almacenando una cadena si el valor está... rodeado por comillas

En Python, tanto las comillas simples como las dobles pueden usarse para definir cadenas. No importa si eliges comillas dobles o simples. Las comillas solo deben coincidir.

Un programa de computadora está hecho de líneas de código. Puedes agregar tantas

líneas y variables como necesites para darle instrucciones a la computadora.

El código en los programas de computadora está hecho de declaraciones. Las declaraciones son las instrucciones que la computadora debe seguir. Los programas reales pueden contener miles de declaraciones.

¿Cuáles es el número de declaraciones en este código?

```
book = "Harry Potter"
author = "J. K. Rowling"
```

PY

Copiar

2

1

2

La instrucción `print()` es la forma más fácil de enviar un valor a la pantalla.

¿Qué instrucción le está dando este código a la computadora?

```
country = "Iceland"
```

PY

Copiar

mostrar un valor en la pantalla

crear una variable

Aprendiste que:

☀ Un dato de texto se llama cadena

☀ Las cadenas requieren comillas

☀ La instrucción `print()` se utiliza para enviar un valor a la pantalla

4. Datos Numéricos

Los datos numéricos son información que se presenta en forma de números.

Los valores numéricos se pueden almacenar directamente en variables.

Selecciona el nombre de la variable que almacena un valor numérico

```
book = "The Hobbit"
pages = 310
```

PY Copiar

Los datos numéricos no deben estar entre comillas. La línea de código a continuación declara una variable numérica.

```
points = 500
```

- Puedes enviar un número a la pantalla con la instrucción `print()`. Solo necesitas insertar el número entre paréntesis.
- *Puedes realizar operaciones matemáticas con números. Cada instrucción `print()` mostrará el valor del resultado en una nueva línea.*
- Puedes usar la instrucción `print()` para comprobar que la computadora está siguiendo tus instrucciones.

Ambas líneas de código a continuación harán que la computadora realice el cálculo.

¿Cuál mostrará también el resultado en la pantalla?

```
print(3 + 7)
```

```
3 + 7
```

El nombre de una variable se utiliza para identificar dónde se almacena esa información. Puedes acceder al valor que una variable está almacenando llamando a su nombre.

El fragmento a continuación es parte de un prototipo para un nuevo videojuego. ¿Qué 2 valores mostrará este código en la pantalla?

```
username = "magician"
points = 50
lives = 3
print(username)
print(points)
```

Aprendiste que:

- ☀ Los valores numéricos se pueden almacenar en variables
- ☀ Puedes acceder al valor almacenado en una variable llamando a su nombre
- ☀ Los datos numéricos no deben estar entre comillas

Tarea

Completa el código para mostrar el mensaje "Game over" en la pantalla

```
message = "Game over"
message
```

5. Trabajando con Variables

Las variables son clave para el desarrollo de software. Te permiten almacenar, etiquetar y jugar con datos.

Completa el código para almacenar la cadena en la variable

Completa el código para almacenar el número en la variable

Puedes acceder al valor almacenado en una variable llamando a su nombre.

¿Qué enviará este código a la pantalla?

```
price = 150  
print(price)
```

PY

Copiar

Tarea

```
budget = 20  
print(budget + 10)
```

PY

Copiar

```
price = 5  
amount = 3  
print(price * amount)
```

PY

Copiar

Puedes almacenar el resultado de un cálculo en una variable.

¿Qué enviará este código a la pantalla?

```
score = 7 + 8  
print(score)
```

Puedes crear una nueva variable para almacenar el resultado de un cálculo hecho usando otras variables.

Completa el código para declarar una variable llamada total

```
price = 5  
amount = 6
```

Puedes actualizar el valor almacenado en una variable. La variable olvidará el valor almacenado previamente.

```
price = 99  
price = 100  
print(price)
```

Actualizar el valor de una variable se llama reasignar una variable.

¿Qué enviará este código a la pantalla?

```
points = 35  
points = 45  
print(points)
```

Tarea

Reorganiza las declaraciones para que primero el código declare la variable price, luego declare la variable discount y finalmente muestre el valor del descuento en la pantalla.

<code>print(discount)</code>	A	⋮
<code>discount = 0.2 * price</code>	B	⋮
<code>price = 14</code>	C	⋮

¿Qué valores enviará el código a la pantalla?

```
name = "Tom"
level = 14
print(name)
level = level + 1
print(level)
```

Selecciona todas las respuestas correctas.

<input type="checkbox"/> Tom
<input type="checkbox"/> 15
<input type="checkbox"/> level

Aprendiste que:

- ☀ Puedes hacer cálculos usando los valores almacenados en variables
- ☀ Puedes almacenar el resultado de un cálculo en una variable
- ☀ Actualizar el valor de una variable se llama reasignar una variable

6. Modulo de prueba

¿Cuál es la forma correcta de crear una variable llamada score con el valor de 100?

- A. 100 = score
- B. score 100
- C. score = 100

¿Cómo podrías nombrar una variable que almacena información sobre el clima?

- D. número
- E. precio
- F. temperatura

¿Cómo se llama el tipo de dato que viene como una pieza de texto?

- G. párrafo
- H. cadena
- I. palabra

Produce una línea de código para almacenar el valor "Paris" en una variable llamada city

city= "Paris"

Completa el código para crear una variable, luego enviar su valor a la pantalla

XXXXXXXXXX

¿Qué enviará este código a la pantalla?

```
name = "Tom"  
salary = 75000  
print(name)
```

XXXXXX

¿Qué enviará este código a la pantalla?

```
player = "James"
```

```
score = 55  
level = 14
```

Reorganiza las declaraciones para que primero el código declare la variable movie, luego declare la variable year y finalmente muestre el valor de la variable movie en la pantalla.

- A. print(movie)
- B. year = 1997
- C. movie = "Titanic"

orden: xxxxxxxxxxxx

¿Qué valor enviará este código a la pantalla?

```
band = "Beatles"  
song = "Yesterday"  
song = "Let it be"  
print(song)
```

Se espera que el siguiente código calcule el número de elementos por caja. Completa el código para declarar las variables items y boxes, luego muestra el resultado de la división en la pantalla

```
items  18  
boxes = 6  
 (items / boxes)
```

A:xxxx,B:xxxx

=====

===

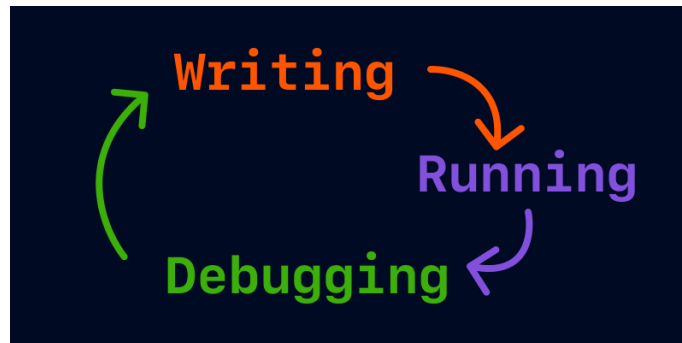
Profundizando

1. Debugging

La programación consta de 3 pasos:

- Escribir
- Ejecutar
- Corregir errores (o debugging)

En esta lección, se profundizará con el tercer paso: identificar y corregir errores.



Las máquinas se negarán a seguir las instrucciones de un humano si contienen errores.

Este código contiene un error. Python devolverá un mensaje de error. Ejecuta el código para obtener el mensaje de error.

```
message = "Debugging"
print(message)
```

Incluso los programadores experimentados cometen errores de programación (o bugs) constantemente.

Así que si estás cometiendo errores, estás en el camino correcto.

La línea de código a continuación contiene un error. ¿Puedes identificarlo?

```
print 5 + 3
```

Incluso el error tipográfico más pequeño resultará en un error.

```
message = "Debugging"
print(message)
```

La línea de código para crear una nueva variable contiene un error.

```
color -> "red"
```

Crea una línea de código para almacenar un título de película "Avatar" en una variable llamada movie

Selecciona el nombre de la variable que almacena una cadena

```
surname = "Smith"
age = 32
```

Los errores son una parte esperada de escribir código. El truco para ser un buen programador es mantener la calma al tratar con errores.

Un fragmento de código puede contener múltiples errores. ¿Puedes corregir el código para mostrar la cadena?

```
print"Great Progress)
```

Si tu código contiene múltiples errores, debes corregirlos todos para que el código se ejecute correctamente.

¿Qué líneas contienen errores?

```
salary = 50000
role = Analyst
age -> 29
print(salary)
```

La computadora lee y ejecuta instrucciones línea por línea, de arriba hacia abajo. La ejecución del programa se interrumpirá en el primer error encontrado.

¿En qué línea se detendrá la ejecución de este código?

```
salary = 50000
role = Analyst
age -> 29
```

```
print(salary)
```

El código a continuación contiene errores en las líneas 2 y 3. La computadora se quedará atascada en la línea 2. ¿Por qué?

```
salary = 50000  
role = Analyst  
age -> 29  
print(salary)
```

Después de encontrar un error, la computadora dejará de leer y ejecutar el código.

¿En qué línea se detendrá la ejecución de este código?

```
salary = 70000  
print(salary)  
age -> 32  
role = "Product Manager"  
print(role)
```

Llamar al nombre de una variable que no ha sido definida es un error muy común.

¿Qué enviará este código a la pantalla?

```
print(balance)
```

El código a continuación devolverá un error. ¿Por qué?

```
print(revenue)  
revenue = 300
```

Completa el código para declarar 2 variables y enviar el modelo de automóvil a la pantalla

```
brand __ "Audi"  
model = "Q5"  
__(model)
```

EJERCICIO DE PRÁCTICA

Se espera que el código dado muestre los resultados de una competencia de baile en la pantalla, pero algo está mal con las instrucciones.

Tarea

Arregla los errores para mostrar el siguiente resultado en la pantalla

```
Anna
96
```

```
name = Anna
score -> 96
print name
print score
```

Aprendiste que:

- ☀ Los errores en el código se conocen como bugs
- ☀ El código se ejecuta línea por línea de arriba hacia abajo
- ☀ La ejecución del código se interrumpe por errores.

2. Estándares y Mejores Prácticas

Millones de programadores desarrollan y mantienen programas todos los días. Se necesitan estándares universales y buenas prácticas para la colaboración.

En esta lección, se aplicaran algunos de estos estándares y mejores prácticas utilizados en la industria tecnológica.

Los desarrolladores profesionales usan comentarios para agregar descripciones y explicaciones a sus programas. Puedes agregar comentarios a tu código Python con el símbolo #. Los comentarios hacen que el código sea más fácil de leer para los humanos.

Completa el código para crear un comentario que explique lo que está haciendo el código.

```
☐ a variable
transaction = "accepted"
```

#

Declaring

Displaying

Los comentarios ayudarán a otros (¡y a ti en el futuro!) a comprender el código. Los comentarios no son instrucciones, son ignorados por las máquinas.

```
# Declarando variables de pago
currency = "USD"
amount = 200
status = "overdue"
# Mostrar estado
print(status)
```

Los comentarios en el código son:

explicaciones para los humanos

instrucciones para la computadora

¿Es una buena práctica usar comentarios en tu código?

Puedes usar comentarios para desactivar temporalmente una instrucción. De esta manera, la computadora omitirá la instrucción.

¿Qué enviará este código a la pantalla?

```
#print("Game Over")
```

Este código dará como resultado un error. ¿Por qué?

```
# client = "loyal"  
print(client)
```

PY

Copiar

El valor almacenado en la variable debería ser un número

El valor almacenado en la variable es una cadena

La declaración de variable está comentada y es ignorada

Python es un lenguaje sensible a mayúsculas y minúsculas, lo que significa que "A" y "a" se tratan como diferentes.

¿Cuántas variables diferentes se están declarando en el siguiente código?

```
credit = 300  
Credit = 280  
CREDIT = 320
```

Python es un lenguaje sensible a mayúsculas y minúsculas.

Verdadero o Falso? Se pueden cometer errores si no prestas atención al uso de mayúsculas y minúsculas.

Completa el código para mostrar el valor de la variable declarada en la pantalla

```
campaign = "email"  
print( )
```

Campaign

CAMPAIGN

campaign

Snake case es una forma popular de crear nombres de variables de manera

consistente. Snake case utiliza guiones bajos “_” para separar palabras en el nombre de una variable.

Selecciona el nombre de variable recomendado en el estilo snake case

- A. *dog-name*
- B. *DogName*
- C. *dog_name*

Los programadores usan snake case para dar nombres descriptivos a las variables con varias palabras. El guión bajo hace que el nombre de la variable sea más fácil de leer.

¿Cómo llamarías a una variable que almacena una identidad de usuario

- A. *user/id*
- B. *user_id*
- C. *user#id*
- D. *user*id*

Completa el código a continuación para agregar un comentario y declarar una variable usando el estilo snake case.

```
Using snake case  
credit__score = 700
```

EJERCICIO DE PRÁCTICA

El código dado es parte de una aplicación de rastreo de vuelos. El equipo que trabaja en este proyecto ha agregado explicaciones al código, pero la computadora devuelve un error.

Tarea

Usa comentarios para agregar las explicaciones de una manera que no resulte en

errores.

Salida esperada

```
BA0117
New York
1580
```

```
almacenando el número de vuelo
flight_n = "BA0117"

almacenando la información del vuelo

destination = "New York"
distance = 1580

print(flight_n)
print(destination)
print(distance)
```

Aprendiste que:

- ☀️ Puedes agregar comentarios a tu código con el símbolo #
- ☀️ Python es un lenguaje sensible a mayúsculas y minúsculas
- ☀️ Snake case es la mejor práctica para crear nombres de variables con varias palabras

3. Aplicando las mejores prácticas

Es hora de poner en práctica las habilidades que se han aprendido. Evitarás errores y resolverás problemas relacionados con variables y diferentes tipos de datos.

- No se permiten espacios en los nombres de las variables. Python devolverá un error si los nombres de tus variables contienen espacios.

¿Cómo crearías un nombre de variable válido?

```
account ☐ balance = 350
```

- Recuerda que Python es un lenguaje sensible a las mayúsculas y minúsculas.

¿Cómo declaramos la variable?

```
☐ = "mp3"  
print(format)
```

Usa el símbolo correcto para hacer que la primera línea sea un comentario

```
☐ Declaring a variable  
client_type = "new"
```

- Un nombre de variable puede contener números, pero no puede comenzar con un número.

Selecciona el nombre de variable que dará como resultado un mensaje de error

- A. dog_name1
- B. user_id
- C. 2nd_user

Completa el código para que se pueda ejecutar sin errores

```
☐ = "iPad"  
print(device_type)
```

- Puedes usar una variable para construir una nueva variable.

¿Qué enviará este código a la pantalla?

```
destination = "New York"  
distance = 1580
```

```
salary = 900
new_salary = salary + 200
print(new_salary)
```

¿Qué enviará este código a la pantalla?

```
salary = 1000
pay_raise = 100
new_salary = salary + pay_raise
print(new_salary)
```

¿Qué enviará este código a la pantalla?

```
salary = 1000
pay_raise = 100
print(new_salary)
new_salary = salary + pay_raise
```

¿Qué enviará este código a la pantalla?

```
salary = 1000
pay_raise = 100
new_salary = salary + pay_raise
print(salary)
```

Reordena las líneas para crear un código que envíe el valor 220 a la pantalla

new_budget = budget + 20

A

budget = 200

B

print(new_budget)

C

¿Qué mostrará este código en la pantalla?

```
payment_status = "returned"
payment_status = "paid"
print(payment_status)
```

¿Qué mostrará este código en la pantalla?

```
a = 3
a = 5
a = 7
print(a)
```

Selecciona los nombres de variables inválidos que darían como resultado mensajes de error

<input type="checkbox"/> dog_name1	1
<input type="checkbox"/> user_id	2
<input type="checkbox"/> 2nd_user	3
<input type="checkbox"/> user*name	4
<input type="checkbox"/> client type	5

Corrige los errores en el código. Usa snake case para el nombre de la variable
dog name : "Goofy"

EJERCICIO DE PRÁCTICA

Imagina que eres responsable del sistema de gestión de recursos humanos en una empresa. El siguiente fragmento de código calcula los aumentos de salario.

Tarea

Completa el código para calcular el nuevo salario, guarda su valor en una variable y muéstralo en la pantalla.

160000

```
salary = 150000
pay_raise = 10000
#Crea una variable llamada new_salary
print(new_salary)
```


Aprendiste que:

☀ No se permiten espacios en los nombres de las variables

☀ Un nombre de variable no puede comenzar con un número

☀ Las buenas prácticas pueden ayudarte a evitar errores

4. Entradas y Salidas

Los programas informáticos están diseñados para interactuar con los usuarios y el mundo exterior. Hemos visto cómo crear un código que reciba y envíe información.

Una entrada es cualquier información que ingresa a una computadora. Presionar una tecla y hacer clic en un botón son ejemplos de entradas.

¿Cuál es otro ejemplo de entrada?

- A. el sonido de un altavoz
- B. el video capturado por una cámara conectada a una computadora

La instrucción **input()** es la forma más fácil de permitir que un usuario inserte un valor en un programa.

Ejecuta el código para probarlo

```
#Obtener un mensaje del usuario
message = input()
print(message)
```

Explica qué hace el código

```
user_entry = input()
```

- A. muestra el mensaje "user_entry" en la pantalla
- B. pide al usuario un valor que se almacena en una variable llamada user_entry

Completa el código para tomar una entrada del usuario

```
#Asks the user for age
...
#Age, input(), =
```

Una salida es una forma en que la computadora se comunica con el mundo exterior. Un mensaje que se muestra en la pantalla y el sonido de un altavoz son ejemplos de salidas.

¿Cuál es otro ejemplo de salida?

- A. presionar un botón
- B. un documento que sale de una impresora

La instrucción **print()**, que ya conoces, es la forma más fácil de hacer que tu programa genere una salida.

Completa el código para generar una salida

```
-- -- --
#print(), "welcome2, (
```

El código a continuación contiene un error. ¿Puedes arreglarlo para generar la salida?

```
print 360
```

Una aspiradora robot hace un sonido de pitido cuando está atascada. Este sonido es un ejemplo de...

- A. error en el código
- B. entrada
- C. salida

Una aspiradora robot está equipada con un sensor que le ayuda a navegar por su entorno y detectar obstáculos.

La información que proviene del sensor es un ejemplo de...

- A. entrada
- B. salida
- C. error

El programa dentro de una aspiradora robot declara una variable relacionada con el sensor de proximidad.

Selecciona la forma correcta de nombrar la variable utilizando el estilo snake case

- A. proximity_sensor
- B. Proximity Sensor
- C. proximity/sensor

Completa la línea para crear una variable que almacene información del sensor

```
--  
#proximity_sensor,"obstacle",=
```

¿Cuál es el valor almacenado en la variable sensor_proximidad después de las actualizaciones?

```
proximity_sensor = "obstacle"  
proximity_sensor = "clear"
```

- A. "obstacle"
- B. "clear"

Los programas informáticos pueden tener múltiples entradas y salidas. ¿Qué imprimirá el código? Selecciona todas las respuestas correctas.

```
username = "zombie"  
points = 50  
lives = 3  
print(points)  
print(lives)
```

Completa el código para obtener una entrada del usuario y almacenarla en una variable llamada address

`address =`

EJERCICIO DE PRÁCTICA

Escribe un programa que le pida al usuario una entrada y la muestre en la pantalla.

Continuarás trabajando en este código después de la próxima lección.

Tarea

Completa el código para pedirle al usuario una entrada, guárdala en la variable `name` y muéstrala en la pantalla.

input Example -> `name = "Tom"`

input Output -> Tom

Input Example	Expected Output
<code>name = "Tom"</code>	Tom
<code>name = "Bob"</code>	Bob

```
#pídele al usuario una entrada y guárdala en una variable
name =

#muestra la entrada del usuario en la pantalla
```

Aprendiste que:

- 🌟 las entradas y salidas ayudan a las máquinas a comunicarse con el mundo exterior
- 🌟 la instrucción `input()` permite al usuario ingresar un valor en tu programa
- 🌟 la instrucción `print()` se utiliza para generar una salida

5. Tipos de datos

Los datos vienen en diferentes formatos. Las computadoras tratan diferentes tipos de datos de diferentes maneras.

Selecciona la cadena de texto

A. Nintendo

B. 2

C. 2.5

¿Cómo sabes que una variable está almacenando una cadena de texto?

A. La línea comienza con un símbolo de almohadilla #

B. Hay un signo igual =

C. El valor está rodeado por comillas

La cadena es el tipo de dato para un fragmento de texto. Las comillas le indican a la computadora que un valor debe ser almacenado como una cadena de texto.

¿Cuál de las siguientes afirmaciones es verdadera con respecto a las cadenas de texto?

A. Las comillas son opcionales y decorativas

B. Puedes usar comillas simples o dobles siempre y cuando coincidan

Las computadoras almacenan y procesan diferentes tipos de números. Los enteros son números enteros sin un punto decimal. Pueden ser positivos, negativos o cero.

Selecciona el único valor entero

A. "Apple"

B. 2

C. 3.14

El flotante es el tipo de dato para números con decimales, pueden ser positivos o negativos.

Selecciona el único valor flotante

A. 3.14

B. 4

C. "Número"

¿Qué está almacenando esta variable?

variable = 5/2

A. "2.5" - una cadena de texto

B. 2 - un entero

C. 2.5 - un flotante

Se presentan 3 tipos de datos diferentes. Asocia cada tipo de dato con su valor de ejemplo.

- *cadena de texto:*

- *número entero:*

- *número flotante:*

Opciones: 2.5, 5, "accepted"

La forma en que las computadoras operan con valores depende del tipo de dato.

Cuando usas el operador de suma + con valores de cadena de texto, las dos cadenas se unen. Esto se conoce como **concatenación**.

¿Qué producirá el código?

```
a = "basket"
```

```
b = "ball"
```

```
print(a+b)
```

Cualquier cosa entre comillas será tratada como una cadena de texto, incluso los números.

¿Qué tipo de valor está almacenado en la variable?

```
var = "360"
```

No podrás realizar operaciones matemáticas si los números están rodeados por comillas. Serán tratados como cadenas de texto.

¿Cuál sería la salida?

```
print("360" + "360")
```

A. "720"

B. 360360

C. 720

Selecciona todas las cadenas de texto

A. 0

B. "Wednesday"

C. "Avatar"

D. "2.0"

E. 5

Completa el código para mostrar WinterPark en la pantalla

```
## opción: +,),print, "Park, ("Winter"
```

Completa el código para concatenar las cadenas de texto y mostrar el resultado.

```
a = "foot"  
_ = "ball"  
print(a_b)
```

EJERCICIO DE PRÁCTICA

Modifica el código para mostrar un mensaje amigable al usuario

Tarea

Utiliza la concatenación de cadenas para unir las cadenas y mostrar un mensaje amigable personalizado.

input Example -> name ="Mary"

input Output -> Nice to meet you, Mary

Input Example	Expected Output
name = "Mary"	Nice to meet you, Mary
name = "John"	Nice to meet you, John

```
#Toma el nombre como entrada  
name = input()  
#Utiliza la concatenación para unir 2 cadenas  
message =  
#muestra la entrada del usuario en la pantalla
```

Aprendiste que:

☀ las computadoras almacenan y procesan diferentes tipos de datos de manera diferente

☀ la cadena de texto es el tipo de dato para el texto

☀ el entero y el flotante son tipos de datos para los números

6. prueba

6.1. ¿Por qué hay un error en el código?

```
message = "Debugging"
```

```
print(message)
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

6.2. Selecciona la línea que contiene un error

```
band = "Beatles"
```

```
song = Yesterday
```

```
year = 1965
```

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

6.3. Reordena para explicar lo que hace el código

```
movie = "Titanic"
```

```
print(movie)
```

```
year = 1997
```

Se declara una variable llamada movie	A
Se declara una variable llamada year	B
Se muestra el valor de la variable movie	C

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

6.4. ¿Qué mostrará este código en la pantalla?

```
score = 37
```

```
score = 38
```

```
#score = 15
```

```
print(score)
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

6.5. ¿Cuál es el resultado de la ejecución de este código?

```
car_type = "sedan"  
print(sedan)
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

6.6. Declara la variable con un nombre válido

```
_ = "Tom"  
## opción:user1 ,1user , user 1
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

6.7. Completa el código para tomar una entrada del usuario

```
_____  
## opción:name, input(), 2name, =, user 1
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

6.8. Selecciona todos los ejemplos de una salida

Selecciona todas las respuestas correctas.

☒ 1 mensaje mostrado en la pantalla

☒ 2 video de un proyector

☒ 3 clic en un botón

☒ 4 pulsación de una tecla

XXXXXXXXXXXXXXXXXXXXXXXXXXXX

6.9. Asocia cada tipo de dato con su valor de ejemplo:

- cadena:
- entero:
- flotante:

opción:- 5, "great", 3.14, user_1

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

6.10. Completa el código para mostrar honeymoon en la pantalla

__ = "moon"

b = "guide"

__ = "honey"

print(a __ c)

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

=====

===

Trabajando con Datos

1. Verificación de Datos

Los datos tienen variedad de formatos diferentes. Trabajar con datos en el formato incorrecto puede resultar en pérdida o corrupción de datos.

En esta lección, aprenderás a verificar el tipo de datos almacenado en una variable.

Relaciona cada valor con su tipo de dato

cadena:

número entero:

número flotante:

Opcion : 15.32, 365, "Aladin"

Selecciona el entero

A. 530

B. 9.57

C. "The Jungle Book"

D. "22"

¿Qué resultado se mostrará en pantalla?

a = "note"

```
b = "book"
```

```
print(a+b)
```

No puedes sumar un número a una cadena de texto.

```
a = 3
b = "8"
print(a + b)
```

¿Por qué necesitamos diferentes tipos de datos?

- A. Para marcar comentarios y explicaciones que serán ignorados por la computadora
- B. Las computadoras almacenan y manejan diferentes tipos de datos de diferentes maneras

Los datos pueden llegar en un formato incorrecto. Puedes usar la instrucción **type()** para verificar el tipo de datos almacenado en una variable

```
balance = "780"
type(balance)
```

¿Qué está almacenando la variable balance?

- A. número entero
- B. número decimal
- C. cadena de texto

Ejecuta el código para mostrar el tipo de datos almacenado en las variables

```
city = "Berlin" #stores a string
age = 42 #stores an integer
balance = 830.29 #stores a float
print(type(city)) #outputs <class 'str'>
print(type(age)) #outputs <class 'int'>
print(type(balance)) #outputs <class 'float'>
```

¿Qué mostrará este código?

```
balance = 234.3
print(type(balance))
```

Completa el código para mostrar el tipo de dato almacenado en la variable

```
balance = 300
__(__(balance))
## opción: type, print
```

¿Por qué utilizamos tipos de datos?

- A. Para enviar un mensaje a la pantalla
- B. Para indicarle a una computadora cómo almacenar y manejar un valor

El código devolverá un error. ¿Por qué?

```
budget = 200
expenses = "180"
savings = budget - expenses
```

- A. no puedes hacer cálculos matemáticos con cadenas de texto*
- B. budget debería almacenar un número decimal*
- C. savings es un nombre de variable inválido*
- D. budget debería almacenar un número entero*

La división de dos números enteros siempre produce un número decimal (flotante).

```
a = 4
b = 2
c = 4/2
print(c) #displays the result
print(type(c)) #displays data type
```



¿Qué mostrará este código en la pantalla?

```
x = 9
y = 3
print(x / y)
```

Completa el código para mostrar el tipo de datos

```
print(__(3.0))
```

Aprendiste que:

-  *la instrucción type() se utiliza para verificar el tipo de datos*
-  *la división de dos números enteros siempre produce un número decimal*

2. Conversión de datos

Los datos pueden venir en un formato incorrecto. Los datos de encuestas y formularios web pueden llegar con problemas de calidad. En esta lección, aprenderás a convertir datos para obtenerlos en el formato correcto.

Relaciona cada valor con su tipo de dato

cadena: __
entero: __
flotante: __
opcion: 84, 7.7, "BMW"

Completa el código para tomar la entrada del usuario

```
birth_year = __
```

La instrucción `input()` siempre convierte la entrada del usuario en una cadena de texto, sin importar lo que el usuario ingrese.

Ejecuta el código para verificar el tipo de dato

```
birth_year=input() #tomar un valor
print(type(birth_year)) #muestra el tipo de dato
```

¿Qué almacena esta variable después de que el usuario ingresa un número?

```
project_budget = input()
```

Los valores numéricos no deben almacenarse como cadenas de texto. ¿Por qué?

- A. Las cadenas de texto solo deben usarse con números decimales
- B. No podrás realizar operaciones matemáticas con los valores

Puede convertir datos de un tipo a otro para solucionar problemas de calidad de datos.

La instrucción `int()` convierte cualquier tipo de valor en un número entero

```
x = "55" # x es una cadena de texto
print(type(x))
y = int(x) # y es un entero
print(type(y))
```

Completa el código para convertir la cadena de texto en un entero

```
a = "14"
```

```
b = __ (a)
```

```
## opción: num, int
```

Puedes usar la instrucción `int()` para convertir la entrada del usuario en un entero

```
# Convierte el número en un entero
height = int(input())
print(type(height))
# La línea anterior es una forma efectiva de combinar 2 instrucciones en una.
```

```
# height1 = input()
# height2 = int(height1)
# print(type(height2))
```

Relaciona la variable con el tipo de dato que almacenará cuando el usuario ingrese un número

```
a = input()
b = int(input())
```

a: __

b: __

opcion: entero, cadena

Reordena para explicar qué hace el código

```
age = int(input())
```

A. La entrada del usuario se convierte en un entero

B. El entero se almacena en una variable

C. Pide al usuario una entrada

Hay situaciones en las que necesitas que los valores se traten como flotantes. La instrucción `float()` convierte valores en flotantes.

Completa el código para convertir el valor en un flotante

```
a = 15
b = __ (a)
## opcion: dec, float
```

¿Qué mostrará el código?

```
a = 3
b = float(a)
print(b)
```

De manera similar, puedes asegurarte de que los valores se conviertan en cadenas de texto con la instrucción `str()`.

Completa el código para convertir el valor en una cadena de texto

```
a = 15
b = __ (a)
## opcion: dec, float
```

Las instrucciones `int()`, `str()` y `float()` son ejemplos de conversión explícita, lo que significa que son realizadas por una instrucción dada por un programador (como tú).

Por otro lado, ejecuta el código para ver algunos ejemplos de conversiones de tipo de dato implícitas (automáticas)

```
# Ejemplos de conversión automática de tipos de datos

x = 5 # entero
y = 2 # entero
z = x/y # flotante (conversión implícita)
print(z)

a = 3 # entero
b = 1.5 # flotante
c = a + b # flotante
print(c)
```

¿Qué mostrará el código?

`x = input()`

`y = input()`

`print(x+y)`

A. la suma de los números

B. la concatenación de dos cadenas

Completa el código para convertir el valor en un entero

`a = "6"`

`b = __ (a)`

EJERCICIO DE PRÁCTICA

Estás desarrollando una aplicación financiera que ayuda a los usuarios a hacer crecer sus ahorros.

Tarea

Completa el código para tomar los ahorros, calcular el monto final y luego mostrar un mensaje en la pantalla.

input Example -> savings = 150

input Output -> Amount in 1 year: 157.5

savings = 250

Amount in 1 year: 262.5


```
# Pide al usuario que ingrese los ahorros
savings = input()
# Convierte la entrada del usuario en un valor decimal y actualiza la
variable
savings =
# Los ahorros crecen después de 1 año con una tasa de interés anual del 5%
balance = savings * 1.05
# Convierte el saldo en una cadena y actualiza la variable
balance =

# Concatena las 2 cadenas para producir un mensaje
message = "Amount in 1 year: " + balance

# Muestra el mensaje
```

Aprendiste que:

- ☀️ puedes cambiar el tipo de dato de un valor con `int()`, `float()` y `str()`
- ☀️ existen conversiones de tipo de dato implícitas y explícitas en Python
- ☀️ las instrucciones `str()`, `int()` y `float()` son conversiones explícitas

3. Corrigiendo datos

La ejecución de tu programa puede fallar si tus datos están en el formato incorrecto. En esta lección, pondrás en práctica tus habilidades de conversión de tipos de datos para corregir problemas de calidad de datos.

¿Cuál es la instrucción utilizada para convertir datos en un entero?

- A. `number()`
- B. `num()`
- C. `int()`

Debe asegurarse de que la score ingresada por el usuario se almacene como un número entero.

Completa el código

```
role = input()
score = __ (input())
## opción: num, int, str
```

Completa el código para tomar dos números y sumarlos

```
score1 = __ (input())
score2 = int(__)
total_score = score1 + __
## opcion: input(), score2, int
```

¿Qué mostrará este código después de que el usuario ingrese un número?

```
age = int(input())
print(type(age))
```

El código mostrará <class 'int'>

```
x = 15
print(type(x))
```

¿Qué significa?

- A. la instrucción type() convierte los datos a un tipo de dato entero
- B. la variable x está almacenando un número entero

Completa el código para mostrar snowball en la pantalla

```
a = "ball"
b = "snow"
c = "foot"
print(__ __ __)
## opción: c, a, +, b
```

Este código para una aplicación de navegación resultará en un error. ¿Por qué?

```
distance = 14
units = "km"
print(distance + units)
```

- A. Las dos variables almacenan cadenas de texto
- B. Los enteros no se pueden concatenar con cadenas de texto
- C. Nombres de variables inválidos

¿Qué mostrará este código?

```
print(14 + "km")
```

- A. 14 km
- B. Un error
- C. 14

¿Qué mostrará el código?

```
print("14" + "km")
```

- A. 14
- B. Un error
- C. 14km

El comando `str()` puede ayudarte con las concatenaciones.

Completa el código para mostrar un mensaje en la pantalla

```
distance = 14
units = "km"
print( __ (distance) + units)
## opción: text, str
```

El código contiene múltiples errores. ¿Qué líneas contienen errores?

```
investment = 120000
rate -> 0.1
print investment * rate
```

Verifica y muestra el tipo de dato almacenado en la variable

```
x = 5.6
__ ( __ )
## opción: x, data, type(x), print
```

Las operaciones matemáticas entre enteros y flotantes producen un flotante.

¿Qué mostrará el código en la pantalla?

```
x = 9
y = 3.0
print(x+y)
## opción: 9.0, 12, 12.0,
```

Este código mostrará un flotante

```
print(12 / 6)
```

Esto es un ejemplo de...

- A. conversión explícita de tipo de datos
- B. conversión implícita de tipo de datos

Completa el código para asegurarte de que el valor ingresado por el usuario se almacene como un entero

```
amount = __ (input())
```

Aprendiste que:

- ☀️ puedes usar conversiones explícitas de tipo de datos para evitar errores en tus programas
- ☀️ `int()` asegura que la entrada del usuario se trate como un número entero
- ☀️ `str()` puede ayudarte a concatenar números con texto

4. Operaciones de Comparación

Las computadoras son más rápidas y precisas que los humanos en ciertas operaciones.

En esta lección, aprenderás sobre un tipo de operación que hace que las máquinas evalúen diferentes escenarios y tomen decisiones.

Las operaciones de comparación son clave para el desarrollo de programas. La línea de código a continuación muestra un ejemplo de una operación de comparación.

`5 < 9`

¿Es 5 menor que 9?

A. No (False)

B. Sí (True)

Una operación de comparación siempre resulta en uno de estos dos resultados: Sí o No

`50 > 100`

¿Es 50 mayor que 100?

Sí (True)

No (False)

Ambas declaraciones instruirán a la computadora a realizar la operación de comparación.

¿Cuál de ellas también mostrará el resultado de la operación como salida?

A. `print(30 < 25)`

B. `30 < 25`

Ejecuta el código para verificar el resultado de diferentes operaciones de comparación en Python

```
print(30 < 25)
```

```
print(5 < 9)
```

```
print(50 > 100)
```

El resultado de una operación de comparación en Python es True o False.

Elige el operador de comparación que hará que el código produzca un valor True.

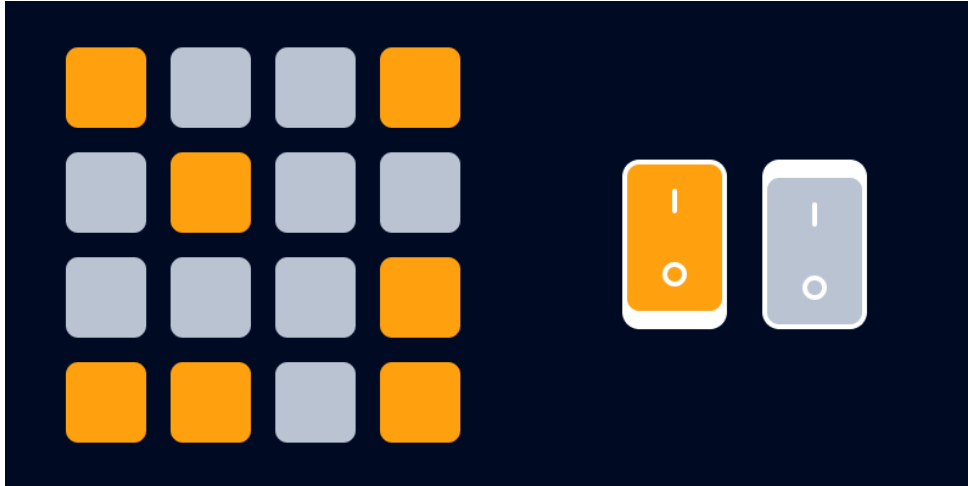
```
print(70__30)
```

¿Cuáles son las salidas de estas declaraciones?

`print(3 > 15): ____`

```
print(3 < 15): ____  
## opción: True, False
```

Los circuitos electrónicos dentro de las computadoras utilizan millones de interruptores pequeños para almacenar estos valores True/False.



¿Cuántas posiciones posibles tienen estos interruptores?

- A. 3 (LOW, MEDIUM and HIGH)
- B. 2 (ON and OFF)



Las computadoras utilizan código binario para representar información. Al encender y apagar interruptores, cambiamos la información almacenada en una computadora.

¿Qué significa binario en este contexto?

- A. el código consiste en dos líneas
- B. dos posibilidades para el estado de un interruptor

Para simplificar las cosas, se utilizan dos números para representar los estados de OFF/ON de un interruptor.

Estos números son:

A. 1 y 2

B. 0 y 1

El código binario está compuesto por 0 y 1, y los pequeños interruptores permiten que las computadoras realicen miles de millones de operaciones..rápidamente

Tareas complejas como la transmisión de video o las transacciones bancarias en línea se pueden descomponer en cálculos simples y pequeños.

Dispositivos modernos como telescopios y cámaras digitales almacenan y procesan información utilizando...código binario

Ahora estás listo para conocer otro tipo de dato. El booleano es un tipo de dato que solo tiene dos valores posibles: Verdadero (True) o Falso (False).

Ejecuta el código para ver el tipo de dato de los valores

```
print(type(5 < 9))  
print(type(50 > 100))  
print(type(True))  
print(type(False))
```

Este tipo de dato lleva el nombre de George Boole, quien creó la teoría que es la base de las computadoras modernas.

Las operaciones de comparación y los valores booleanos permiten que las máquinas tomen decisiones.

Selecciona todos los valores booleanos

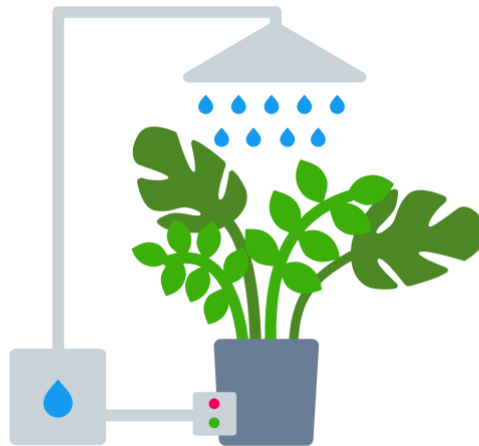
A. True

B. 3.24

C. "avatar"

D. False

Veamos un ejemplo real. Un sistema de riego de plantas utiliza un sensor para medir la humedad del suelo.



Comencemos creando una variable que almacene la humedad del suelo (soil moisture)

```
__ __ __  
## opcion: 80, =, soil_moisture
```

Cuando la humedad del suelo es inferior a 100 unidades, el sistema regará automáticamente la planta. El valor True activará la acción.

Completa el código con el operador de comparación que hará que el código produzca el valor True.

```
soil_moisture = 80  
print(soil_moisture __ 100)
```

Puedes usar booleanos para activar acciones. Después de regar la planta, la lectura del sensor de humedad del suelo aumentará.

¿Qué producirá el código?

```
soil_moisture = 80  
soil_moisture = 120  
print(soil_moisture < 100)
```

EJERCICIO DE PRÁCTICA

El fragmento de código dado para un videojuego verifica si el jugador está listo para el siguiente nivel. Solo los jugadores con una puntuación mayor a 100 pueden pasar al siguiente nivel.

Tarea

Completa el código para mostrar True cuando la puntuación sea mayor a 100 y False en caso contrario.

Input Example	Expected Output
score = 120	True
score = 95	False

codigo:

```
# El programa toma la puntuación como entrada
score = int(input())
# Agrega la operación de comparación dentro de los paréntesis
print()
```

Aprendiste que:

- ☀ el tipo de dato booleano tiene dos valores posibles: Verdadero (True) o Falso (False)
- ☀ una operación de comparación siempre produce un valor booleano

5. Operaciones Lógicas

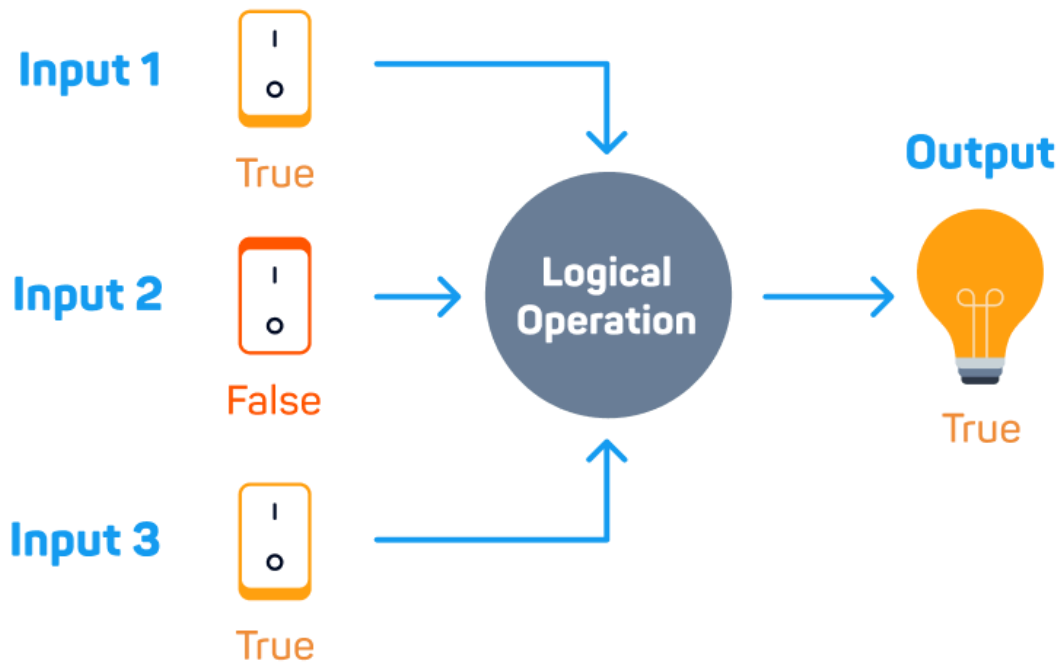
Las computadoras modernas pueden realizar tareas complejas rápidamente porque estas pueden descomponerse en muchas operaciones simples.

En esta lección, aprenderás sobre otro tipo de operación que las computadoras pueden hacer más rápido que los humanos.

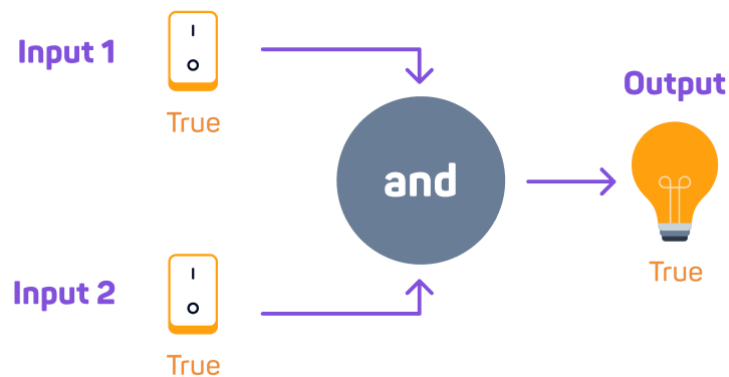
Las operaciones lógicas son necesarias para que las máquinas evalúen escenarios complejos.

Las operaciones lógicas utilizan valores booleanos. *¿Recuerdas qué es un booleano?*

- A. Cualquier número entero
- B. Un tipo de dato con dos posibles valores: True o False
- C. Otro tipo de operación



Una operación lógica...
toma varios valores booleanos como _____
produce solo 1 valor booleano como _____



La operación "and" resulta en un valor Verdadero solo cuando todas las entradas son Verdaderas al mismo tiempo. ¿Cuál es el resultado de esta operación lógica "and"?
True and True

Una operación lógica combina entradas booleanas para producir una salida booleana.

True and True = True

Cualquier otra combinación = False

¿Cuál es el resultado de la operación lógica "and"?

True and False

Completa la operación lógica para producir False como resultado

___ and True

La operación "or" resulta en un valor Verdadero si al menos una de las entradas es Verdadera.

¿Cuál es el resultado de esta operación lógica "or"?

True or False

A. False

B. True

Completa la operación lógica para producir False como resultado

False or ___

¿Cuál es el resultado de esta operación lógica?

True and True

Completa el código para producir True como resultado

False ___ True

Completa el código para obtener un valor True

___ ___ False

Las operaciones lógicas deben insertarse en las instrucciones print() para que el resultado se muestre.

```
print(True and False)
print(False and True)
print(True or False)
print(False or True)
```

Ambas líneas de código a continuación realizarán la operación lógica. *¿Cuál de ellas también mostrará el resultado en la pantalla?*

A. `print(True and False)`

B. `True and False`

Completa la operación lógica para que el resultado sea False

False ___ True

Completa la operación lógica para que el resultado sea True

False ___ True

Aprendiste que:

- ☀ las operaciones lógicas toman múltiples valores booleanos como entrada
- ☀ las operaciones lógicas producen un solo valor booleano como salida
- ☀ "and" y "or" son ejemplos de operaciones lógicas

6. Combinación de operaciones de comparación y lógicas

En esta lección, combinarás operadores de comparación y lógicos para crear programas más rápidos y precisos.

Relaciona el valor con su tipo de dato

cadena de texto: _____

entero: _____

flotante: _____

Booleano: _____

15, False, 10.4, "Menu", "2"

El resultado de una operación de comparación siempre es...

- A. una cadena de texto
- B. un entero
- C. un flotante
- D. un Booleano

El código dentro de un reloj inteligente almacena la frecuencia cardíaca del usuario en una variable.



Elige el operador de comparación que hará que el código muestre un valor True cuando la frecuencia cardíaca del usuario sea mayor a 75 latidos por minuto.

```
heart_rate = 78
```

```
print(heart_rate ____ 75)
# # , > , <
```

Puedes almacenar valores booleanos en variables al igual que lo haces con otros tipos de datos.

El reloj inteligente puede detectar cuando el usuario está durmiendo. Completa el código para almacenar el booleano en la variable

```
sleep = ____
# # =, True
```

Puedes almacenar el resultado de una operación de comparación en una variable.

```
heart_rate = 165
____ ____ heart_rate > 160
# # = , peak_rate
```

¿Qué mostrará el código?

```
heart_rate = 165
peak_rate = heart_rate > 160
print(peak_rate)
```

Puedes almacenar el resultado de una operación lógica en una variable.

¿Qué mostrará este código?

```
a = True and False
print(a)
```



El código dentro de una casa inteligente almacena el estado de las luces en una variable.

Completa el código

```
light_on ____  
# # True, =
```

Puedes operar con valores almacenados en variables.

¿Qué mostrará este código?

```
light_on = True  
door_locked = False  
print(light_on or door_locked)  
# # False  
# # True
```

Python es un lenguaje sensible a mayúsculas y minúsculas.

- Tanto "True" como "False" comienzan con una letra mayúscula.
- Los operadores "and" y "or" son en minúsculas en Python.

Corrige los errores en el código

```
light_on = true  
door_locked = false  
print(light_on OR door_locked)  
print(light_on AND door_locked)
```

La casa inteligente almacena la temperatura en grados Celsius (°C) en una variable. El sistema de control de temperatura enciende el aire acondicionado (AC) cuando la temperatura es mayor a 30 °C.

¿Qué mostrará este código?

```
temp = 35
ac_on = temp > 30
print(ac_on)
```

Puedes poner paréntesis alrededor de las operaciones que deben realizarse primero. Esto hace que el código sea más fácil de leer.

```
a = (3 > 2) or False
```

*Selecciona el valor que la variable **a** está almacenando.*

A. False

B. True

C. 3

La casa inteligente puede detectar la presencia de personas en la casa.

¿Cuándo dará como resultado True la siguiente operación?

```
(temp > 30) or presence
```

El sistema de control de temperatura solo debe encender el aire acondicionado cuando la temperatura es mayor a 30°C y hay personas en la casa.

Haz que la siguiente operación dé como resultado True bajo estas condiciones

```
presence ____ (temp > 30)
```

Completa el código para que el aire acondicionado se encienda cuando la temperatura sea mayor a 30 y haya personas en la casa

```
ac_on = (temp ____ 30) ____ presence
```

EJERCICIO DE PRÁCTICA

Se considera que un usuario ha alcanzado la meta diaria de fitness cuando el número de pasos es mayor a 10000 o el número de minutos activos es mayor a 30.

Tarea

Completa el código para mostrar *True* si el usuario ha alcanzado la meta diaria de fitness, y *False* en caso contrario.

Input Example	Expected Output
steps = 10035 active_minutes = 15	True
steps = 9850 active_minutes = 45	True
steps = 9850 active_minutes = 25	False

código:

```
# Tomar los pasos y los minutos como entradas
steps = int(input())
active_minutes = int(input())

# Almacenar el resultado de las operaciones en la variable
goal_achieved =

# Mostrar el resultado en la pantalla
```

Aprendiste que:

- ☀️ Puedes almacenar valores booleanos en variables
- ☀️ Puedes almacenar el resultado de operaciones lógicas y de comparación en variables
- ☀️ Puedes combinar operaciones con operadores lógicos y de comparación

7. Prueba

7.1 Selecciona el entero

- 9.57
- "22"
- 530

d. "Luz de luna"

R:_____

7.2 Completa el código para verificar y mostrar el tipo de dato almacenado en la variable

```
movie = "The Matrix"
____ ( ____ ( ____ ))
# # movie, type, print
```

R:_____

7.3 La división de dos enteros siempre produce un...

- a. cadena
- b. entero
- c. flotante

R:_____

7.4 Relaciona el nombre de la variable con el tipo de dato que almacena

```
name = input()
age = int(input())
is_adult = True
```

- name: ____
- age: ____
- is_adult: ____

7.5 Completa el código para una aplicación de pronóstico del clima

```
temp = 24
unit = "°C"
print( ____ ( ____ ) + ____ )
# # unit, str, temp
```

7.6 Un sistema de cámaras de tráfico detecta vehículos que exceden el límite de velocidad para generar automáticamente una multa. ¿Qué mostrará este código?

```
speed = 65
ticket = speed > 70
print(ticket)
```

R:_____

7.7 ¿Qué mostrará este código?

```
active = True
registered = False
print(active or registered)
```

R:_____

7.8 Una aplicación recomienda una película si tiene una calificación superior a 70 y más de 5000 reproducciones. Completa el código para mostrar True si se cumplen esas condiciones


```
rating = 74
views = 5400
print( ____ > 70 ____ views ____ 5000)
# # >, and , rating
```

7.9 ¿Cuáles de las variables almacenan un valor True?

```
a = 5
b = True
c = a > 8
d = b or c
```

R:_____

7.10 Un fragmento de código para un videojuego verifica si el jugador ha completado el nivel. El nivel se completa si el jugador ha recolectado más de 30 coins o ha encontrado al menos 1 magic key.

Completa el código para verificar si el jugador ha completado el nivel

```
coins = 36
keys = 2
print(coins > ____ ____ keys > 0)
```

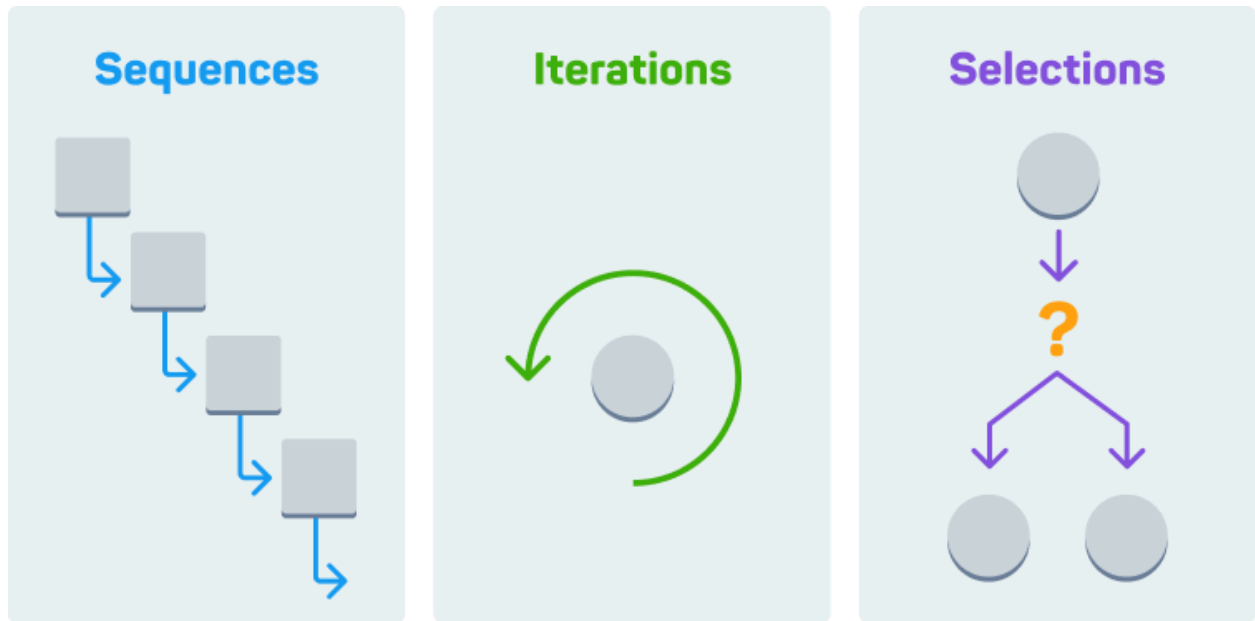
```
=====
===
```

Control de flujo

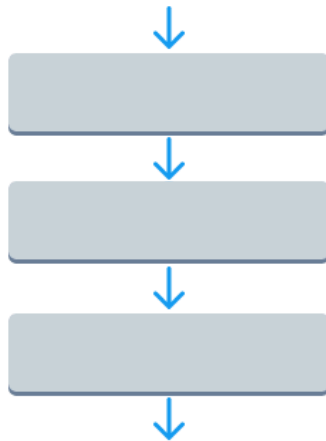
1. Control de flujo

Las computadoras son capaces de tomar decisiones y nos ayudan a resolver problemas del mundo real porque son muy buenas siguiendo instrucciones.

En esta lección, aprenderás a controlar el flujo de instrucciones que sigue una computadora utilizando 3 técnicas: secuenciación, iteración y selección.



Ya conoces la secuenciación. Significa que la computadora ejecuta tu código en orden, de arriba a abajo.

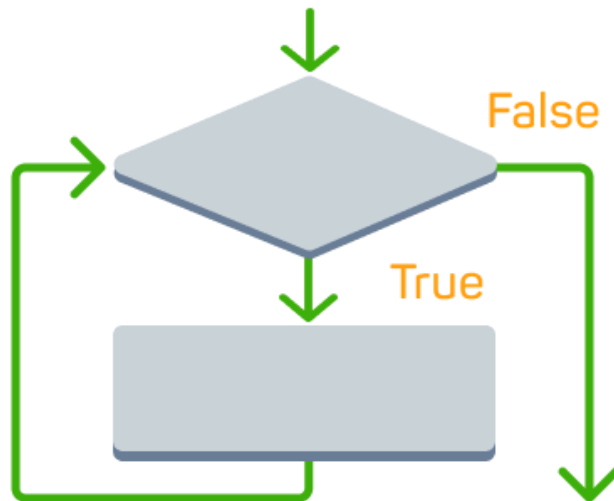


```
print("3")
print("building")
print("blocks")
```

Coloca las salidas en el orden en que se mostrarán

- A. 3
- B. building
- C. block

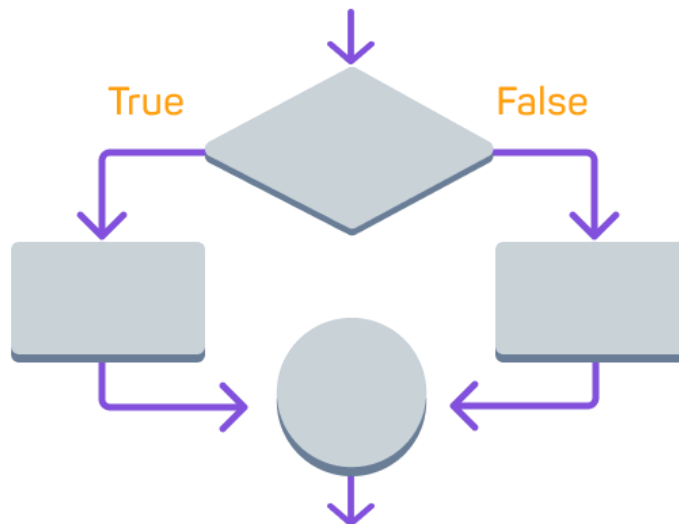
a, b, c



La iteración se trata de ejecutar una instrucción repetidamente. La iteración comúnmente se representa como un bucle.

Poner tu canción favorita en repetición es un ejemplo de...

- A. iteración
- B. secuenciación



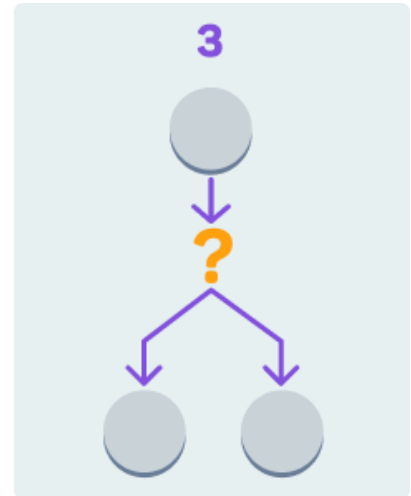
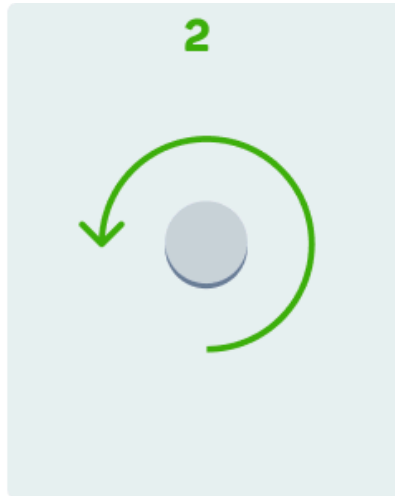
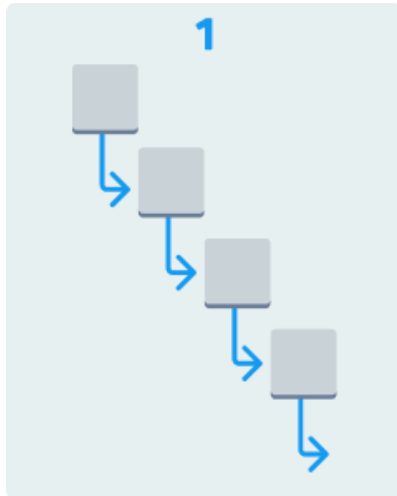
La selección especifica cuándo seguir cada camino.

Los relojes inteligentes notifican al usuario si su ritmo cardíaco sale del rango normal. Esto es un ejemplo de...

- A. iteración
- B. secuenciación
- C. selección

Los programas de computadora reales realizan tareas complejas combinando las 3 técnicas.

Relaciona el diagrama con la técnica correspondiente



Iteración: _____

Selección: _____

Secuenciación: _____

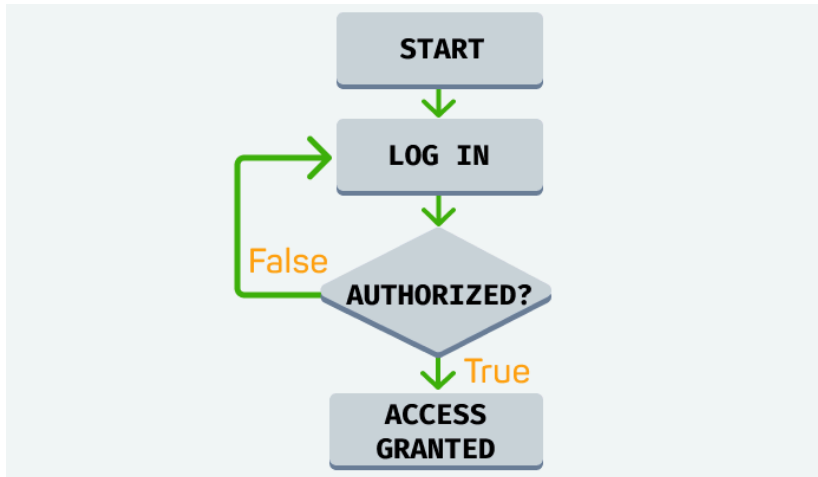
Las máquinas pueden completar tareas complejas por nosotros, pero primero necesitan saber cómo. Un algoritmo es un conjunto de instrucciones paso a paso para completar una tarea, colocadas en un cierto orden.

Los algoritmos existen en nuestra vida cotidiana.

Los pasos de instrucciones en una receta de cocina son un ejemplo de...

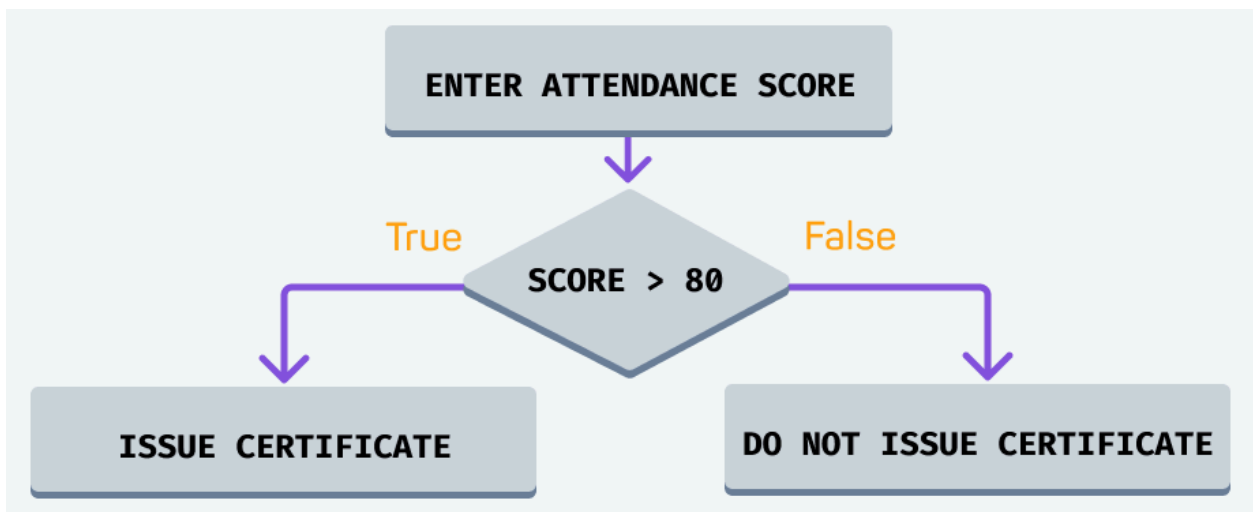
- A. un error
- B. datos
- C. un algoritmo

Los algoritmos pueden representarse de muchas formas. Por ejemplo, los diagramas de flujo ayudan a visualizar los algoritmos.



¿Qué técnicas puedes identificar en el diagrama de flujo dado?

- A. iteración
- B. conteo
- C. selección
- D. secuenciación



Se necesitan operaciones de comparación y lógicas para controlar el flujo de tus programas.

En el diagrama, una universidad otorga certificados a los estudiantes con altas puntuaciones de asistencia.

¿Qué técnicas puedes identificar en el diagrama de flujo anterior?

- A. Selección
- B. Secuenciación
- C. Iteración

Otra forma de representar un algoritmo es con pseudocódigo. El pseudocódigo es un lenguaje simplificado que se acerca un poco más a un lenguaje de programación.

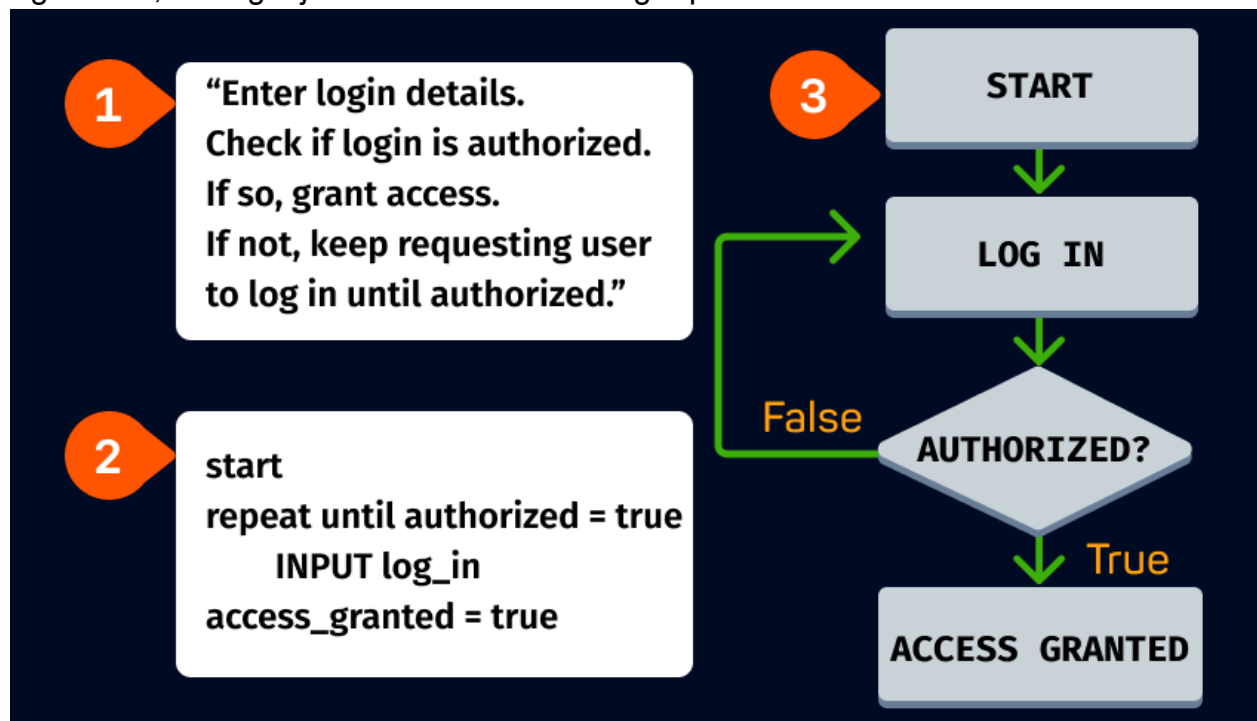
```
start
repeat until authorised = true
  INPUT log_in
  access_granted = true
```

¿Puedes adivinar la respuesta correcta?

A. El pseudocódigo es específico de Python

B. El pseudocódigo no es específico de un lenguaje de programación

Los algoritmos pueden representarse de diferentes formas. Si eres nuevo en los algoritmos, el lenguaje natural es un buen lugar para comenzar.



Identifica las diferentes representaciones de algoritmos

Diagrama de flujo: _____

Lenguaje natural: _____

Pseudocódigo: _____

¿Cuáles son ejemplos de algoritmos en la vida real?

- A. baile improvisado*
- B. indicaciones para llegar a un lugar*
- C. receta de cocina*

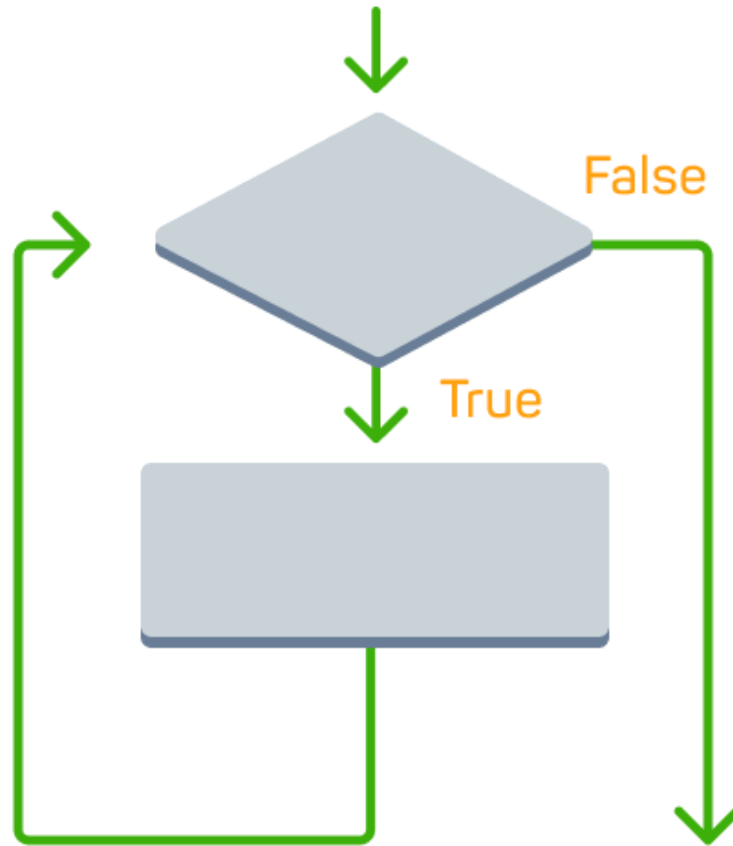
Los humanos usan código para comunicarse con las máquinas. Para que las máquinas completen una tarea, las instrucciones deben ser...

- A. en un lenguaje que la maquina pueda entender*
- B. dadas en orden correcto*
- C. libres de errores*
- D. escritas en lenguaje natural*

Aprendiste que:

- ★ Utilizas secuenciación, iteración y selección para controlar el flujo de instrucciones
- ★ Un algoritmo es un conjunto de instrucciones paso a paso para completar una tarea
- ★ Los algoritmos pueden representarse de diferentes formas

2. Bucles For



La iteración se utiliza para automatizar tareas que deben realizarse una y otra vez. La iteración hace que tus programas sean más simples, más rápidos y reduce los errores. En esta lección, aprenderás cómo crear bucles for.

Completa el programa para que imprima Hello 3 veces

```
print("____")
____("Hello")
print(____ Hello")
```

Un bucle for se utiliza para ejecutar la misma instrucción una y otra vez, un número específico de veces.

¿Puedes hacer que el código imprima el mensaje Hello 10 veces?

```
#Definir el número de iteraciones
for i in range(3):
    #Declaración que se repite
    print("Hello")
```


El bucle for comienza con la palabra clave for.

```
for i in range(100):  
    print("Hello")
```

¿Cuántas veces se imprimirá Hello?

- A. 10
- B. 3
- C. 100

range() crea 10 números en una secuencia, comenzando desde 0.

```
for i in range(10):  
    print(i)
```

La función range(5) generará una secuencia de 5 números.

Selecciona la secuencia que generará

- A. 1, 2, 3, 4, 5
- B. 0, 1, 2, 3, 4

¿Cuántos números generará range(5)?

- A. 5
- B. 4
- C. 3

range() genera una serie de números enteros. La variable i se utiliza para iterar sobre los números. Puedes reemplazar i con cualquier otra variable de tu elección.

```
#Bucle usando i como variable interna  
for i in range(3):  
    print(i)  
  
print("--")    #Esto es solo un separador  
  
# Bucle usando "something" como variable interna  
for something in range(3):  
    print(something)
```

1

```
for i in range(3):  
    print("Hello")
```

2

```
for i in range(3):  
print("Hello")
```

El código que se repite en el bucle for debe tener sangría. La sangría son los espacios al principio de las líneas.

Selecciona el código que muestra el uso correcto de la sangría

A. 2

B. 1

El código que no está correctamente sangrado dará como resultado errores.

A Python no le importa si usas 2 espacios o 4 espacios (o cualquier otro número de espacios), siempre y cuando seas consistente.

¿Qué se repite en el código del bucle for?

```
for i in range(3):  
    print("Hello")
```

A. range(3)

B. print("Hello")

Completa el código del bucle

```
____ i in ____ :  
    ____("Congrats!")
```

¿Qué imprimirá el código?

```
for i in range(5):  
    print("Congrats!")
```

A. Un error

C. Congrats! 5 veces

El código dará como resultado un error. ¿Por qué?

```
for num in range(5):  
    print("Coding is fun!")
```

A. El código que se repite debe tener sangría

B. La variable num no ha sido declarada

La declaración inicial del bucle debe ir seguida de dos puntos ":". Esto indica el inicio del bloque de iteración.

```
for x in range(10):  
    print(x)
```

Este código dará como resultado un error. ¿Por qué?

```
for num in range(6)  
    print(num)
```

- A. Falta el símbolo de dos puntos*
- B. Falta sangría*
- C. Uso incorrecto de la sangría*

¿Cuál de las siguientes afirmaciones son verdaderas sobre los bucles for?

- A. te ayuda evitar errores*
- B. es opcional usar sangría*
- C. hace que tu código se mas simple*

EJERCICIO DE PRÁCTICA

El recordatorio del cinturón de seguridad para un automóvil muestra un mensaje repetidamente para informar a los pasajeros.

Tarea

Completa el programa para mostrar el mensaje "Fasten your seat belt" 3 veces.

```
Fasten your seat belt  
Fasten your seat belt  
Fasten your seat belt
```

código:

Aprendiste que:

- ☀️ puedes implementar la iteración en tus programas con el bucle for
- ☀️ la declaración inicial del bucle debe ir seguida de dos puntos :
- ☀️ el código que se repite debe tener sangría

3. Bucles While

Los bucles while son poderosos porque se pueden utilizar incluso cuando no se sabe cuántas iteraciones serán necesarias.

¿Cuántas veces imprime este programa For Loop?

```
for i in range(5):  
    print("For Loop")
```

Los bucles while repiten el código mientras se cumple una condición.

Por ejemplo, un vendedor de boletos en un teatro venderá boletos repetidamente hasta que todos los asientos estén ocupados.

```
seats= 500 # número inicial de asientos  
while seats > 0: # ¿hay asientos disponibles?  
    print("Sell ticket") # boleto vendido  
    seats = seats - 1 # número de asientos actualizado
```

Los bucles while comienzan con la palabra clave while.

Completa el código

```
____ seats > 0:  
print(____Sell ticket"____  
seats = seats - 1
```

La palabra clave while va seguida de la condición bajo la cual se repite el código.

```
while ____:  
    ____("Sell ticket")  
    seats = seats - 1
```

Cuando la condición deja de ser verdadera, salimos del bucle while.

```
seats = 300  
while seats > 0:  
    print("Sell ticket")  
    seats = seats - 1
```

¿Cuándo dejará de imprimir el mensaje el bucle?

- A. seats = 1
- B. seats = 0

¿Cuántas veces se mostrará el mensaje "Sell ticket"?

```
seats = 300  
while seats > 0:  
    print("Sell ticket")  
    seats = seats - 1
```

Si el código que se repite dentro del bucle no está indentado, el código dará error.

¿Puedes corregir el error?

```
seats = 600
while seats > 0:
print("Sell ticket")
    seats = seats - 1
```

Al igual que con los bucles for, la declaración inicial del bucle while debe ir seguida de dos puntos “:”.

Los bucles generalmente incluyen contadores. Un contador es una variable que lleva un registro del número de iteraciones.

```
seats = 300
while seats > 0:
    print("Sell ticket")
    seats = seats - 1
```

¿Qué contador se utiliza en el ejemplo de los boletos?

- A. *print*
- B. *seats*
- C. *while*

Las variables contador se actualizan dentro del bucle, por lo que cambian con cada iteración. Se establece un valor inicial fuera del bucle, como punto de partida.

¿Qué sucede con el valor de los asientos en cada iteración?

- A. *Permanecen iguales*
- B. *Aumentan*
- C. *Disminuyen*

Con los bucles while, puedes encontrarte con lo que se conoce como un bucle infinito. Esto ocurre cuando la condición es siempre verdadera y el código nunca deja de repetirse.

Los contadores te ayudan a evitar bucles infinitos.

```
seats = 300
while seats > 0:
    print("Sell ticket")
    #seats = seats - 1
```

Veamos más de cerca lo que sucede dentro de un bucle.

```
counter = 0
while counter < 4:
    print(counter)
    counter = counter + 1
```

iteration #	counter	condition <i>counter < 4</i>	output <i>print(counter)</i>	update counter <i>counter = counter + 1</i>
1	0	True	0	1 = 0 + 1
2	1	True	1	2 = 1 + 1
3	2	True	2	3 = 2 + 1
4	3	True	3	4 = 3 + 1
5 (END)	4	False	END OF LOOP	END OF LOOP

```
counter = 0
while counter < 4:
    print(counter)
```

El código a continuación dará como resultado...

- A. un bucle con 4 iteraciones*
- B. un bucle infinito*
- C. un error*

El código a continuación dará como resultado un error. ¿Por qué?

```
counter = 0
while counter < 4
print(counter)
counter = counter + 1
```

Selecciona todas las respuestas correctas.

- A. Bucle infinito*
- B. Falta de indentación*
- C. Falta de los dos puntos*

En general, utiliza bucles for cuando ya conoces el número de iteraciones y bucles while cuando hay una condición que debe cumplirse.

¿Cuáles de las siguientes afirmaciones sobre los bucles while son verdaderas?

- A. A veces puede generar bucle infinito*
- B. Solo se pueden usar cuando se conoce el número de iteraciones*
- C. simplifican el código*

Termina de codificar este bucle while

```
i = 0
____ i < 5 ____
    print(i)
    ____ = i + 1
```

EJERCICIO DE PRÁCTICA

Tarea

Crea un programa de temporizador que tomará el número de segundos como entrada y hará una cuenta regresiva hasta 0.

Input Example	Expected Output
number = 3	3 2 1 0
number = 5	5 4 3 2 1 0

```
# Toma el número como entrada
number = int(input())

# Usa un bucle while para la cuenta regresiva
```

Aprendiste que:

- ☀️ puedes aplicar la iteración a tus programas con el bucle `while`
- ☀️ los contadores llevan un registro del número de iteraciones y evitan bucles infinitos
- ☀️ la indentación y el símbolo de dos puntos `:` son necesarios para que el código se ejecute

4. Más sobre Iteración

La iteración ahora está en tu caja de herramientas. Vamos a ponerla en práctica para resolver algunos ejemplos de la vida real.

```
for i in range(3):
    print(i < 1)
```

¿Qué enviará este programa a la pantalla?

- A. 3 booleanos
- B. 3 números

Un bucle puede repetir múltiples declaraciones. Todas deben estar indentadas.

```
for i in range(3):  
    print(i < 1)
```

¿Cuántas palabras imprimirá este código?

- A. 3
- B. 2
- C. 6

Después de que el bucle haya terminado, la computadora continuará ejecutando las declaraciones en secuencia.

```
for i in range(3):  
    print("A")  
print("B")
```

Cuántas veces se repetirá cada salida?

- A: ____
- B: ____

```
for i in range(5):  
    print("Hello")  
print("Goodbye")
```

El mensaje Goodbye solo se mostrará una vez. ¿Por qué?

- A. Se debe usar un bucle while para repetir múltiples declaraciones
- B. La falta de indentación muestra que la declaración está fuera del bucle

¿Qué secuencia de números genera range(3)?

- A. 0, 1, 2
- B. 0, 1, 2, 3
- C. 1, 2, 3

Ya has utilizado los operadores de comparación > y <.

Ejecuta este código para explorar un poco más:

```
print (5 == 5) # ¿es 5 igual a 5?  
print (5 ==7) # ¿es 5 igual a 7?  
print (5 != 7) # ¿es 5 diferente de 7?  
print (5 != 5) # ¿es 5 diferente de 5?  
print (5 <= 5) # ¿es 5 menor o igual a 5?  
print (5 >= 5) # ¿es 5 mayor o igual a 5?
```

Como regla general, ¿cuándo usarías un bucle for? ¿Y un bucle while?

- Sabes cuántas iteraciones: ____
- No sabes cuántas iteraciones: ____

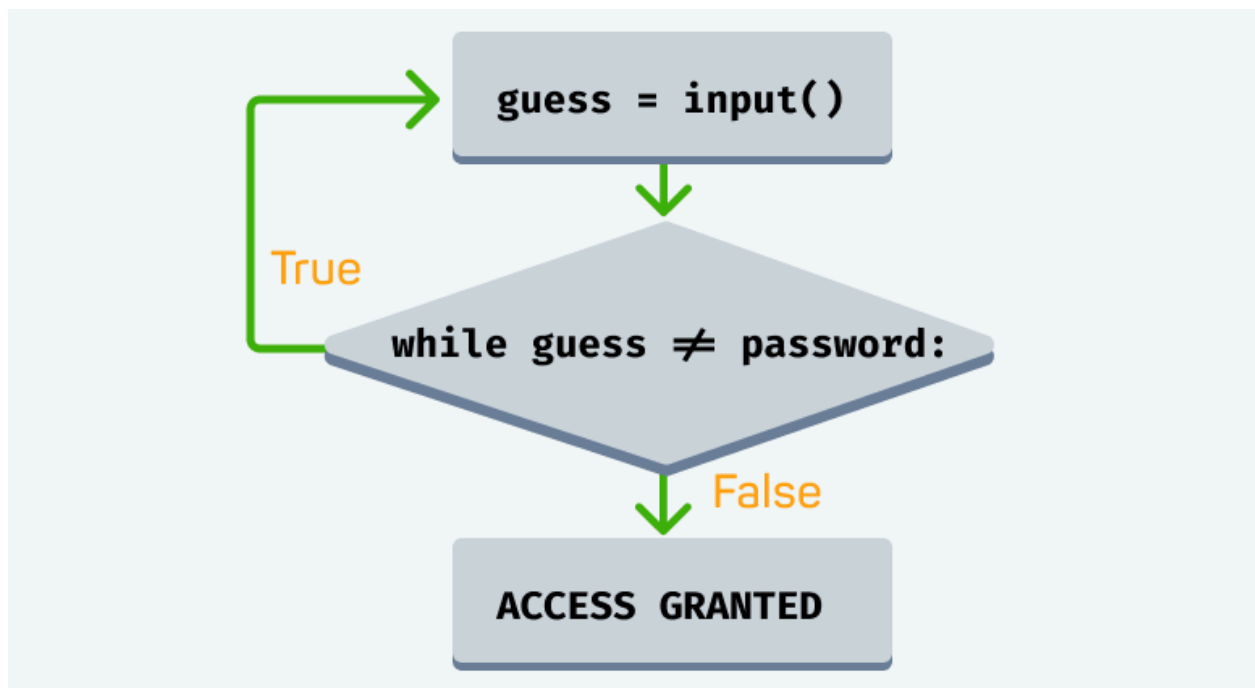
Veamos un ejemplo de la vida real. Un programa le pide repetidamente al usuario que ingrese una contraseña hasta que la contraseña sea correcta.

¿Qué bucle se debe usar?

- A. bucle while
- B. bucle for

¿Qué comando usarías para permitir que el usuario ingrese una contraseña?

- A. `enter()`
- B. `print()`
- C. `input()`



La entrada del usuario se compara con la contraseña correcta. Cuando el usuario ingresa correctamente la contraseña, inicia sesión con éxito.

¿Qué imprimirá el código?

```
password = "SecretWord"
guess = "1234"
print(guess != password)
```

```
password = "SecretWord"
guess = input()
while guess != password:
    guess = input()
print("Access Granted")
```

¿Cuándo terminará el bucle?

- A. cuando la entrada del usuario sea diferente de la contraseña correcta*
- B. cuando la entrada del usuario coincida con la contraseña correcta*

Un robot imprime una etiqueta Package A para cada uno de los 50 paquetes en una colección. ¿Qué bucle se debe usar para programar este robot?

- A. bucle while*
- B. bucle for*

El robot debe mostrar Task Complete una vez que se hayan impreso todas las etiquetas.

```
for box in range(50):
    print("Package A")
    print("Task complete")
```

¿Cuál es el error en el código?

- A. la indentación es incorrecta*
- B. box no ha sido declarado*
- C. falta de los dos puntos*

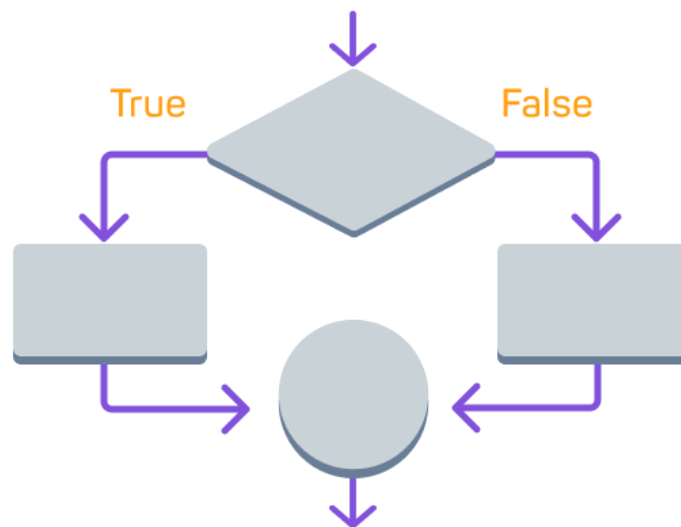
Completa el código para un bucle for

```
___ i in range(20) ___  
print(i)
```

Aprendiste que:

- ☀ los bucles for se usan cuando se conoce el número de iteraciones
- ☀ puedes resolver problemas reales combinando operaciones de comparación e iteraciones

5. Declaraciones condicionales



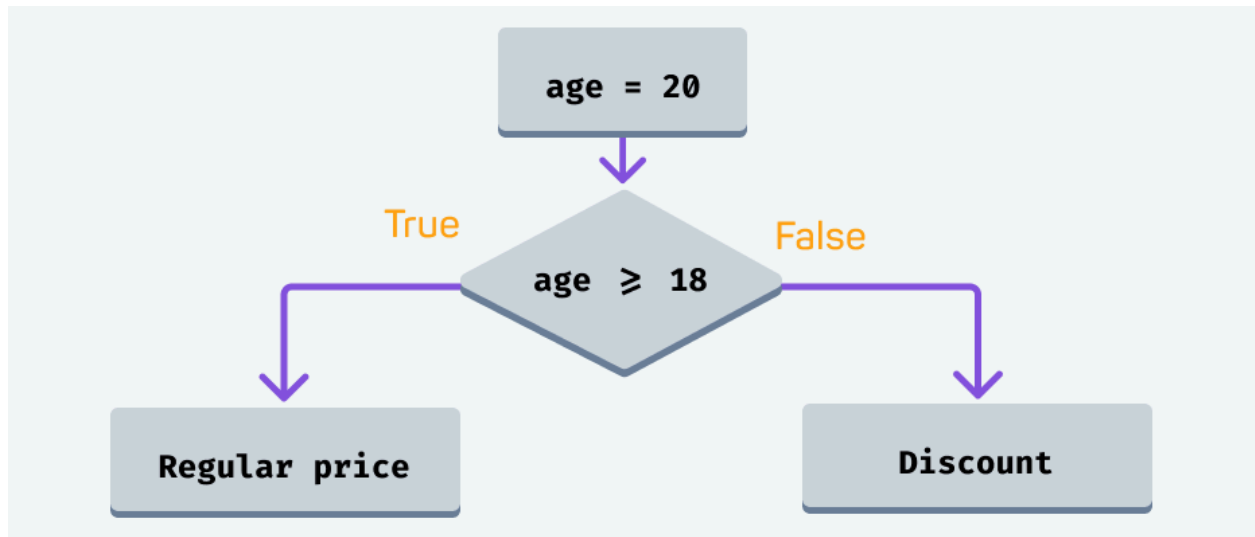
La selección es como un camino en el que se bifurca. Permite a tus programas decidir qué camino tomar.

En esta lección, aprenderás a construir código que utiliza la selección para tomar decisiones.

Las declaraciones condicionales, o declaraciones if-else, permiten que los programas realicen diferentes acciones basadas en las condiciones.

```
age = 22  
if age >= 18:  
    print("Regular price")  
else:  
    print("Discount")
```

¿Cómo cambia la salida si estableces la edad en 14?



El código decide si se debe aplicar un descuento basado en la edad.

¿Cuál es la salida cuando la edad es 20?

```
age = 16
if age >= 18:
    print("Regular price")
else:
    print("Discount")
```

¿Cuál es la salida cuando la edad es 16?

- A. Discount
- B. Regular price

Relaciona la salida con la operación de comparación cuando la edad es 16

- `print (age >= 18) :` ____
- `print (age < 18) :` ____

Relaciona la salida con la operación de comparación cuando la edad es 18

- `print (age >= 18) :` ____
- `print (age < 18) :` ____

Las declaraciones condicionales **if** comienzan con la palabra clave **if** seguida de la condición y el símbolo de dos puntos `:`.

```
__ age >= 18 __
    print("Regular price")
else:
    print("Discount")
```

La declaración condicional **else** comienza con la palabra clave else seguida de dos puntos .:

El código indentado debajo de la línea else se ejecuta cuando la condición no se cumple.

```
if age >= 18:
    print("Regular price")
    _____
    print("Discount")
```

El código que se ejecuta dentro de los bloques if y else debe estar indentado. De lo contrario, habrá errores.

Ejecuta el código para obtener el mensaje de error. Luego, corrige los errores.

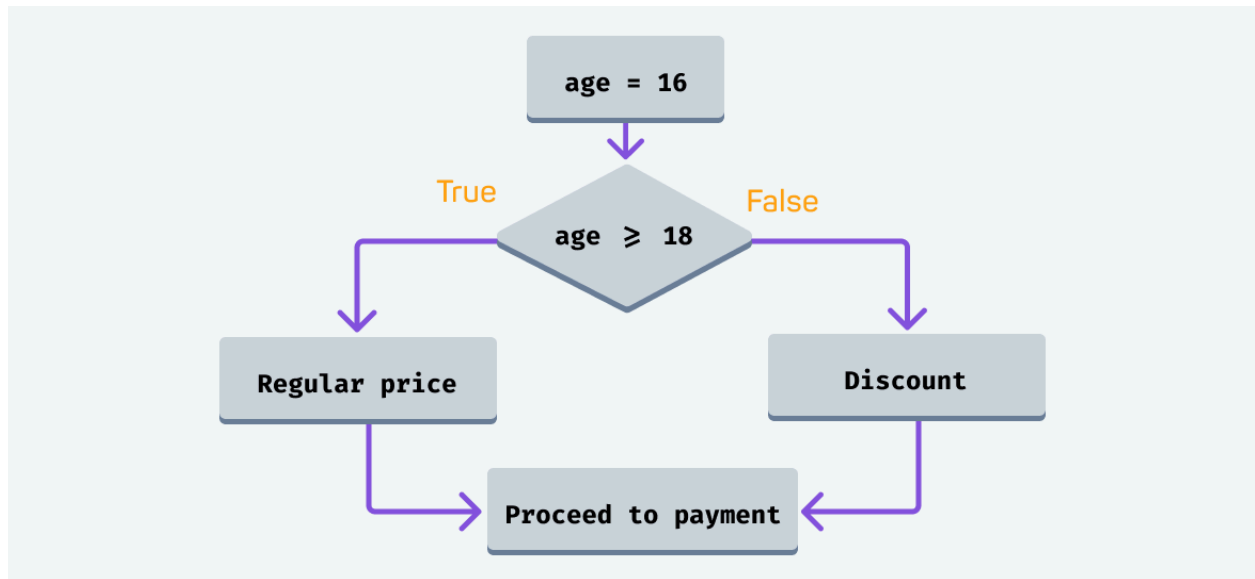
```
age = 41
if age >= 18:
    print("Regular price")
else:
    print("Discount")
```

Después de que la computadora haya terminado de ejecutar la declaración if-else, continuará ejecutando cualquier declaración siguiente en secuencia.

```
age = 30
if age >= 18:
    print("Regular price")
else:
    print("Discount")
print("Proceed to payment")
```

¿Qué imprimirá el código?

- A. Discount
- B. Proceed to payment
- C. Regular price



¿Qué imprimirá el programa?

- A. Proceed to payment
- B. Regular price
- C. Discount

El uso correcto de las operaciones de comparación te permite construir programas que resuelven problemas de la vida real. Practiquemos tus habilidades en operaciones de comparación.

Selecciona el símbolo que hará que la operación resulte en True cuando la edad sea 20.

age ____ 18
== , !=

Selecciona el símbolo que hará que la operación resulte en False cuando la edad sea 18.

age ____ 18
>= , <

Puedes construir programas que tomen decisiones más complejas si combinas operaciones lógicas y de comparación.

Selecciona la operación lógica que hace que la expresión combinada resulte en True.

```
is_student = True  
age = 20
```

```
is_student ____ (age < 18)
```

```
is_student = False  
age = 16  
print(is_student and (age < 18))
```

¿Qué imprimirá este código?

- A. False
- B. True

El programa aplica un descuento si el cliente tiene menos de 18 años o es estudiante.

```
age = 32  
  
is_student = True  
if age < 18 or is_student:  
    print("Discount")  
else:  
    print("Regular price")
```

¿Qué imprimirá el código?

```
age = 32  
  
is_student = True  
if age < 18 or is_student:  
    print("Discount")  
else:  
    print("Regular price")
```

EJERCICIO DE PRÁCTICA

Un estacionamiento inteligente muestra diferentes mensajes al visitante según el número de espacios disponibles.

Tarea

Completa el programa para informar al usuario sobre los espacios disponibles en el estacionamiento

Input Example	Output
space = 20	Available spaces
space = 0	Sorry, the parking lot is full

```
# Tomar el número de espacios disponibles como entrada
spaces = int(input())

# Mostrar mensaje si hay espacios disponibles
if
print("Available spaces")

# Mostrar un mensaje diferente si no hay espacios disponibles
print("Sorry, the parking lot is full")
```

Aprendiste que:

- ☀ Las declaraciones if-else se utilizan para implementar la selección en tus programas
- ☀ El símbolo de dos puntos : y el uso de la indentación son necesarios para evitar errores

6. Más sobre Declaraciones Condicionales

Las declaraciones condicionales te permiten programar máquinas que toman decisiones.

En esta lección, aprenderás a programar decisiones más complejas.

Completa este código

```
is_student = True
__ is_student:
print("Student discount")
__ :
print("No discount")
```

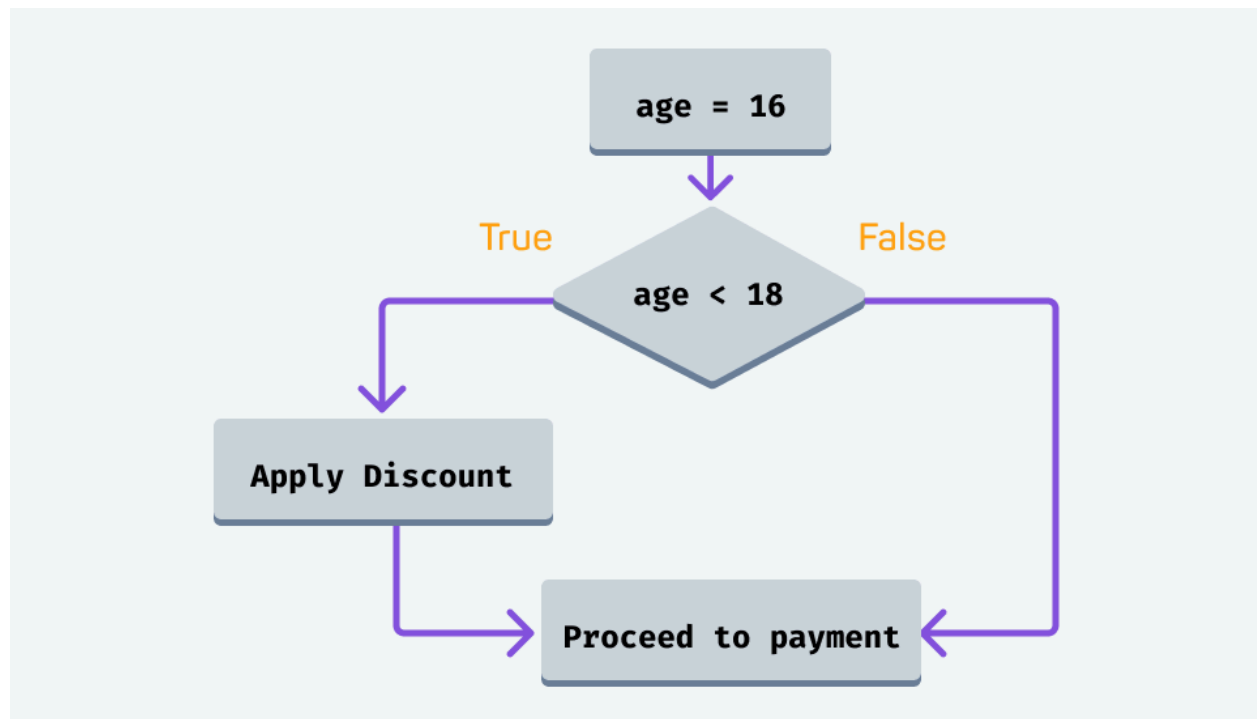
```
temperature = 36
if temperature > 39:
    print("High temperature")
else:
    print("No fever")
```

¿Cuál será la salida?

- A. No fever
- B. High temperature

Habrán situaciones en las que no necesitarás la declaración else. Este programa solo aplica un descuento cuando la edad es menor de 18. El programa no hace nada más si no se cumple la condición, por lo que se puede omitir la declaración else.

```
age = 16
if age < 18:
    print("Apply Discount")
print("Proceed to payment")
```



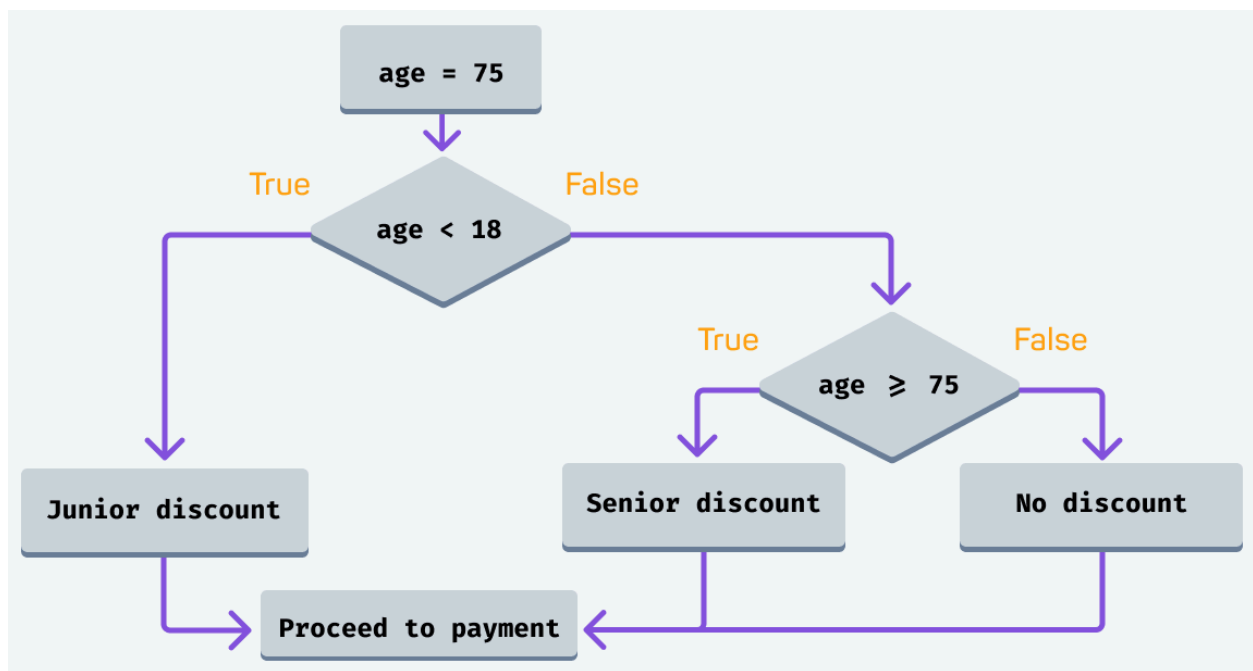
Cuando no se necesita la declaración else, puedes simplificar y tener el código para el bloque de selección en 1 línea.

```
age = 16
```

```
if age < 18: print("Apply Discount")
print("Proceed to Payment")
```

Puedes usar la declaración elif (abreviatura de "else if") para verificar más condiciones si no se cumple la primera condición.

```
age = 75
if age < 18:
    print("Junior discount")
elif age >= 75:
    print("Senior discount")
else:
    print("No discount")
print("Proceed to payment")
```



¿Cuándo se aplica el descuento para adultos mayores?

- A. When age >= 75
- B. When age > 75

Al igual que con cualquier otra declaración condicional, elif requiere el símbolo de dos puntos : y que el código que se ejecuta debajo esté indentado.

Completa el código

```
if age < 18:
print("Junior discount")
    ___ age >= 75 ___
        print("Senior discount")
else:
    print("No discount")
```

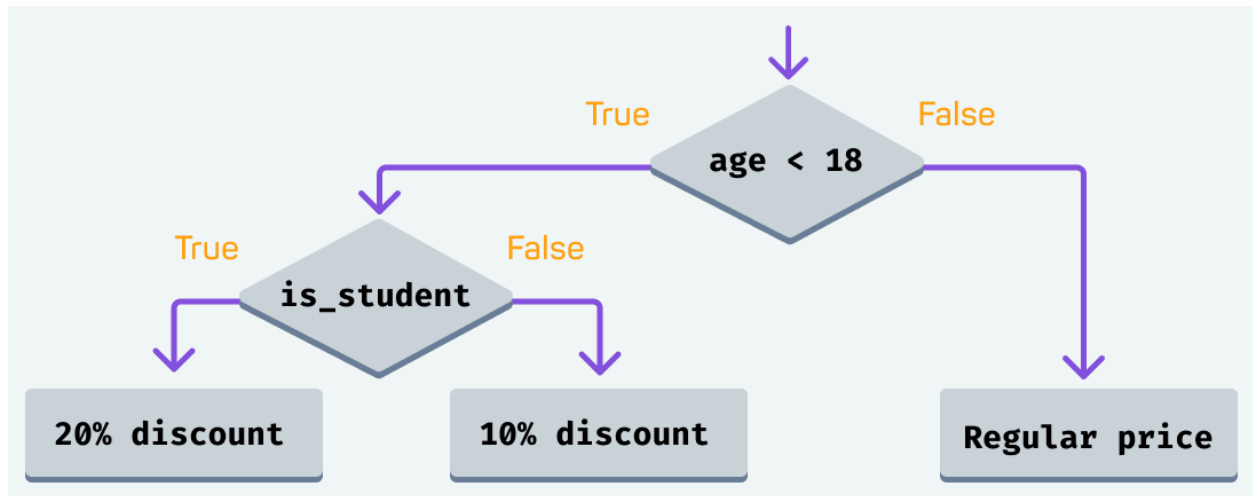
Las declaraciones if, elif y else deben estar en el orden correcto.

```
___ age < 18:
    print("Junior discount")
___ age >= 75:
    print("Senior discount")
___ :
    print("No discount")
```

Puedes anidar declaraciones if-else dentro de otras.

```
age = 16
is_student = True

if age < 18:
    #se ejecuta si la edad es menor de 18
    if is_student:
        #se ejecuta si es menor de 18 y también es estudiante
        print("20% discount")
    else:
        #se ejecuta si es menor de 18 y no es estudiante
        print("10% discount")
else:
    #se ejecuta este código si el cliente tiene 18 años o más
    print("Regular price")
```



¿Cuál sería el descuento cuando el cliente tiene 17 años y no es estudiante?

- A. No discount, regular price
- B. 20% discount
- C. 10% discount

```
if age < 18:
    if is_student:
        print("20% discount")
    else:
        print("10% discount")
else:
    print("Regular price")
```

Se utilizan diferentes niveles de indentación para...

- A. comentar el código
- B. mostrar bloques anidados

Indentar correctamente el código para que el programa se ejecute sin errores

```
age = 16
is_student = True

if age < 18:
    if is_student:
        print("20% discount")
    else:
        print("10% discount")
else:
    print("Regular price")
```

Completa el código para que se ejecute sin errores

```
_____  
if age < 18:  
    print("Junior discount")  
else:  
    print("No discount")
```

```
if age < 18:  
    print("Junior discount")  
else:  
    print("No discount")
```

¿Por qué el programa devolverá un error?

- A. La variable edad no ha sido declarada
- B. Uso incorrecto de la indentación

```
age = 80  
if age < 18:  
    print("Junior discount")  
elif age >= 75:  
    print("Senior discount")  
print("Proceed to payment")
```

¿Qué imprimirá el código?

- A. Junior discount
- B. Senior discount
- C. Proceed to payment

```
age = 29
if age < 18:
    if is_student:
        print("20% discount")
else:
    print("Regular price")
print("Proceed to payment")
```

¿Qué imprimirá el código?

- A. 20% discount
- B. Regular price
- C. Proceed to payment

Completa el código

```
temperature = 40
___ temperature > 39 ___

print("High temperature")
```

Completa el bloque de selección

```
___ age < 18:
    print("Junior discount")
___ age >= 75:
    print("Senior discount")
___ :
    print("No discount")
```

EJERCICIO DE PRÁCTICA

Estás desarrollando un software para un dispositivo médico que informa a los pacientes sobre sus niveles de azúcar en la sangre.

Tarea

Completa el código dado para mostrar diferentes mensajes al paciente según los niveles de azúcar en la sangre

Input Example	Expected Output
glucose_level = 60	Low glucose level
glucose_level = 100	Normal range
glucose_level = 150	High glucose level

```
# El nivel de glucosa es una entrada para este software
glucose_level = int(input())
# Mostrar mensaje si el nivel de glucosa es menor a 70
print("Low glucose level")
# Mostrar mensaje si el nivel de glucosa es mayor a 140
print("High glucose level")
# Mostrar mensaje si no se cumplen ninguna de las condiciones
anteriores
print("Normal range")
```

Aprendiste que puedes:

- ✨ omitir la declaración else cuando no es necesaria
- ✨ verificar más condiciones con la declaración elif
- ✨ anidar declaraciones if-else dentro de otras

7. aszzxzx

