Introducción a la Programación con Python

Contenido

Condicionales

- Los condicionales te permiten, como programador, permitir que tu programa tome decisiones: Como si tu programa tuviera la opción de tomar el camino de la izquierda o el de la derecha basándose en ciertas condiciones.
- Los condicionales permiten que tu programa tome decisiones, eligiendo un camino sobre otro dependiendo de condiciones específicas.
- Integrados en Python hay un conjunto de "operadores" que se utilizan para hacer preguntas matemáticas.
- Los símbolos `>` y `<` probablemente te son bastante familiares.
- `>=` denota "mayor que o igual a."
- `<=` denota "menor que o igual a."
- `==` denota "igual a." Nota el signo de igual doble: un solo signo igual asigna un valor, mientras que dos signos iguales comparan valores.
- `!=` denota "no igual a."
- Las declaraciones condicionales comparan un término del lado izquierdo con un término del lado derecho.

Declaraciones if

- En tu ventana de terminal, escribe `code comparar.py`. Esto creará un archivo completamente nuevo llamado "comparar."
- En la ventana del editor de texto, comienza con lo siguiente:

```
x = int(input("¿Cuál es x? "))

y = int(input("¿Cuál es y? "))

if x < y:

    print("x es menor que y")
```

Nota cómo tu programa toma la entrada del usuario para ambos x e y, los convierte en enteros y los guarda en sus respectivas variables x e y. Luego, la declaración `if` compara x e y. Si la condición de `x < y` se cumple, la declaración `print` se ejecuta.

- Las declaraciones `if` usan valores `bool` (Booleanos) (`True` o `False`) para decidir si ejecutar código o no. Si la comparación `x > y` es `True`, el intérprete ejecuta el bloque indentado.

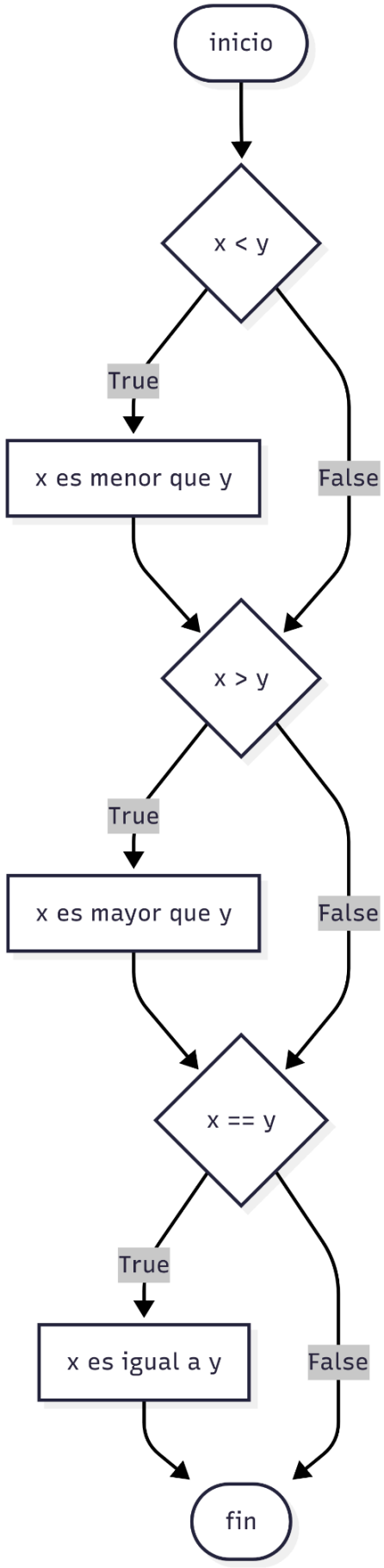
Flujo de Control, elif, y else

- Modifique su código como se indica a continuación:

```
x = int(input("¿Cuál es x? "))

y = int(input("¿Cuál es y? "))

if x < y:

    print("x es menor que y")

if x > y:

    print("x es mayor que y")

if x == y:

    print("x es igual a y")
```

Nota cómo estás proporcionando una serie de declaraciones `if`. Primero, la primera declaración `if` es evaluada. Luego, la segunda declaración `if` ejecuta su evaluación. Finalmente, la última declaración `if` ejecuta su evaluación. Este flujo de decisiones se llama "flujo de control."

- Nuestro código puede ser representado como sigue:

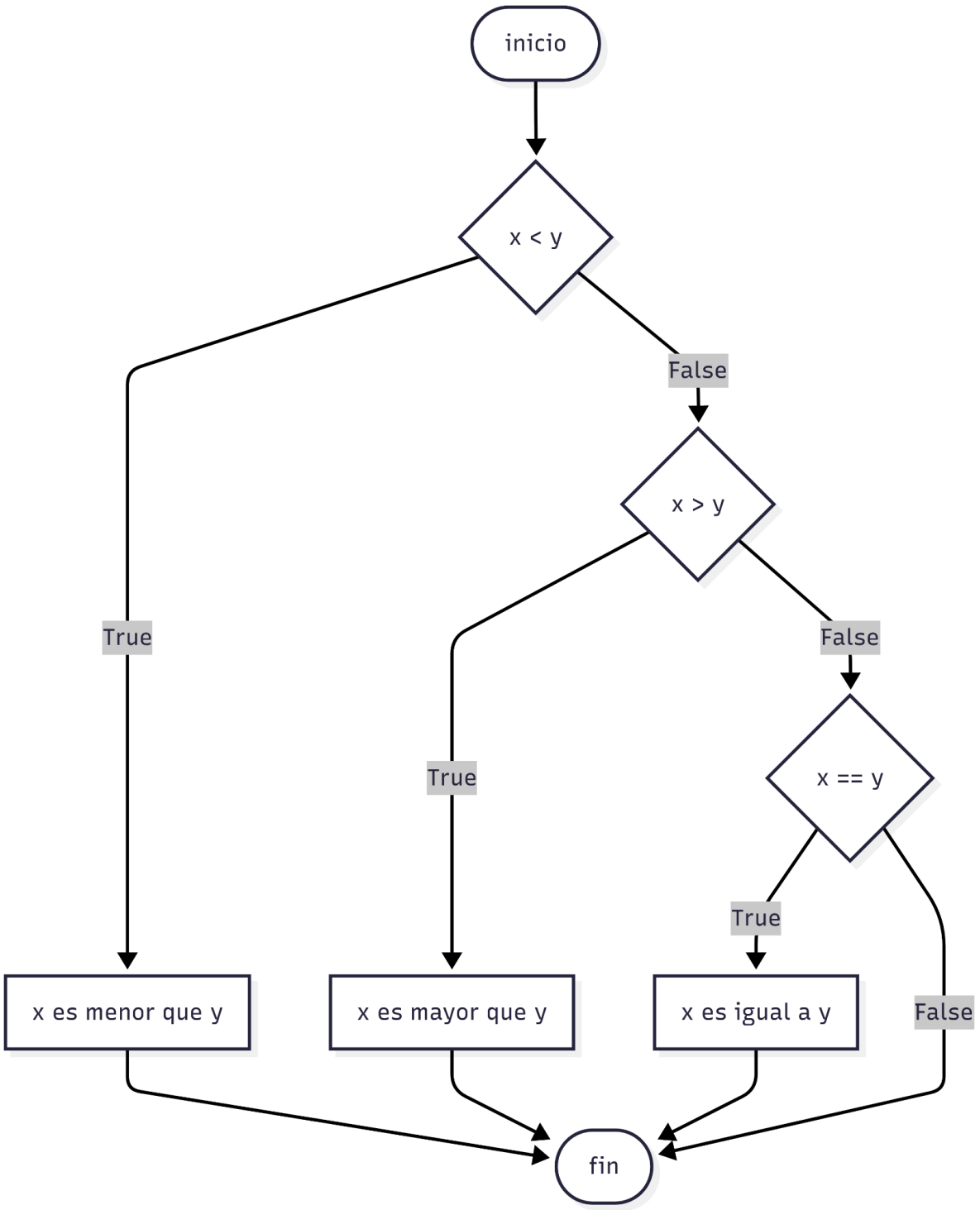

inicio
x < y
True
x es menor que y
False
x > y
True
x es mayor que y
False
x == y
True
x es igual a y
False
fin

- Este programa puede ser mejorado al no hacer tres preguntas consecutivas. Después de todo, ¡no todas las tres preguntas pueden tener un resultado `true`! Revisa tu programa como sigue:

```
x = int(input("¿Cuál es x? "))

y = int(input("¿Cuál es y? "))

if x < y:

    print("x es menor que y")

elif x > y:

    print("x es mayor que y")

elif x == y:

    print("x es igual a y")
```

Nota cómo el uso de `elif` permite al programa tomar menos decisiones. Primero, la declaración `if` es evaluada. Si esta declaración se encuentra verdadera, todas las declaraciones `elif` no se ejecutarán en absoluto. Sin embargo, si la declaración `if` es evaluada y se encuentra falsa, el primer `elif` será evaluado. Si esto es verdadero, no ejecutará la evaluación final.

- Nuestro código puede ser representado como sigue:



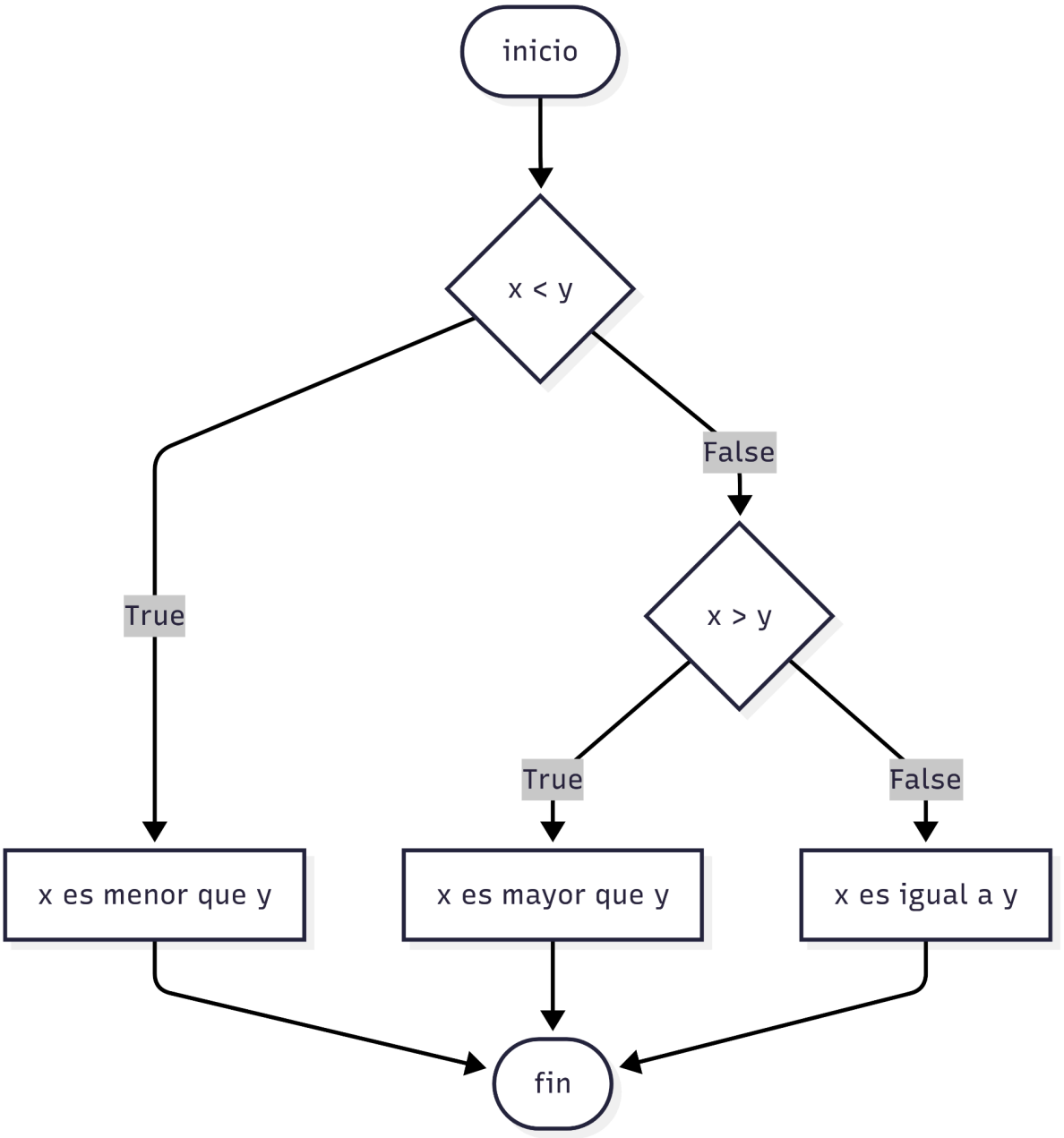
- Aunque tu computadora puede no notar una diferencia en velocidad entre nuestro primer programa y este programa revisado, considera cómo un servidor en línea ejecutando miles de millones o billones de estos tipos de cálculos cada día definitivamente podría ser impactado por una decisión de codificación tan pequeña.

- Hay una mejora final que podemos hacer a nuestro programa. Nota cómo lógicamente `elif x == y` no es una evaluación necesaria para ejecutar. Después de todo, si lógicamente x no es menor que y Y x no es mayor que y, x DEBE ser igual a y. Por lo tanto, no tenemos que ejecutar `elif x == y`. Podemos crear un resultado "catch-all", por defecto usando una declaración `else`. Podemos revisar como sigue:

```
x = int(input("¿Cuál es x? "))

y = int(input("¿Cuál es y? "))

if x < y:

    print("x es menor que y")

elif x > y:

    print("x es mayor que y")

else:

    print("x es igual a y")
```

Nota cómo la complejidad relativa de este programa ha disminuido a través de nuestra revisión.

- Nuestro código puede ser representado como sigue:



or

- `or` permite que tu programa decida entre una o más alternativas. Por ejemplo, podríamos editar nuestro programa como sigue:

```
x = int(input("¿Cuál es x? "))

y = int(input("¿Cuál es y? "))

if x < y or x > y:
```

```
    print("x no es igual a y")

else:

    print("x es igual a y")
```

Nota que el resultado de nuestro programa es el mismo, pero la complejidad disminuye. La eficiencia de nuestro código se incrementa.

- En este punto, nuestro código está bastante bien. Sin embargo, ¿podría el diseño ser mejorado aún más? Podríamos editar nuestro código como sigue:

```
x = int(input("¿Cuál es x? "))

y = int(input("¿Cuál es y? "))

if x != y:

    print("x no es igual a y")

else:

    print("x es igual a y")
```

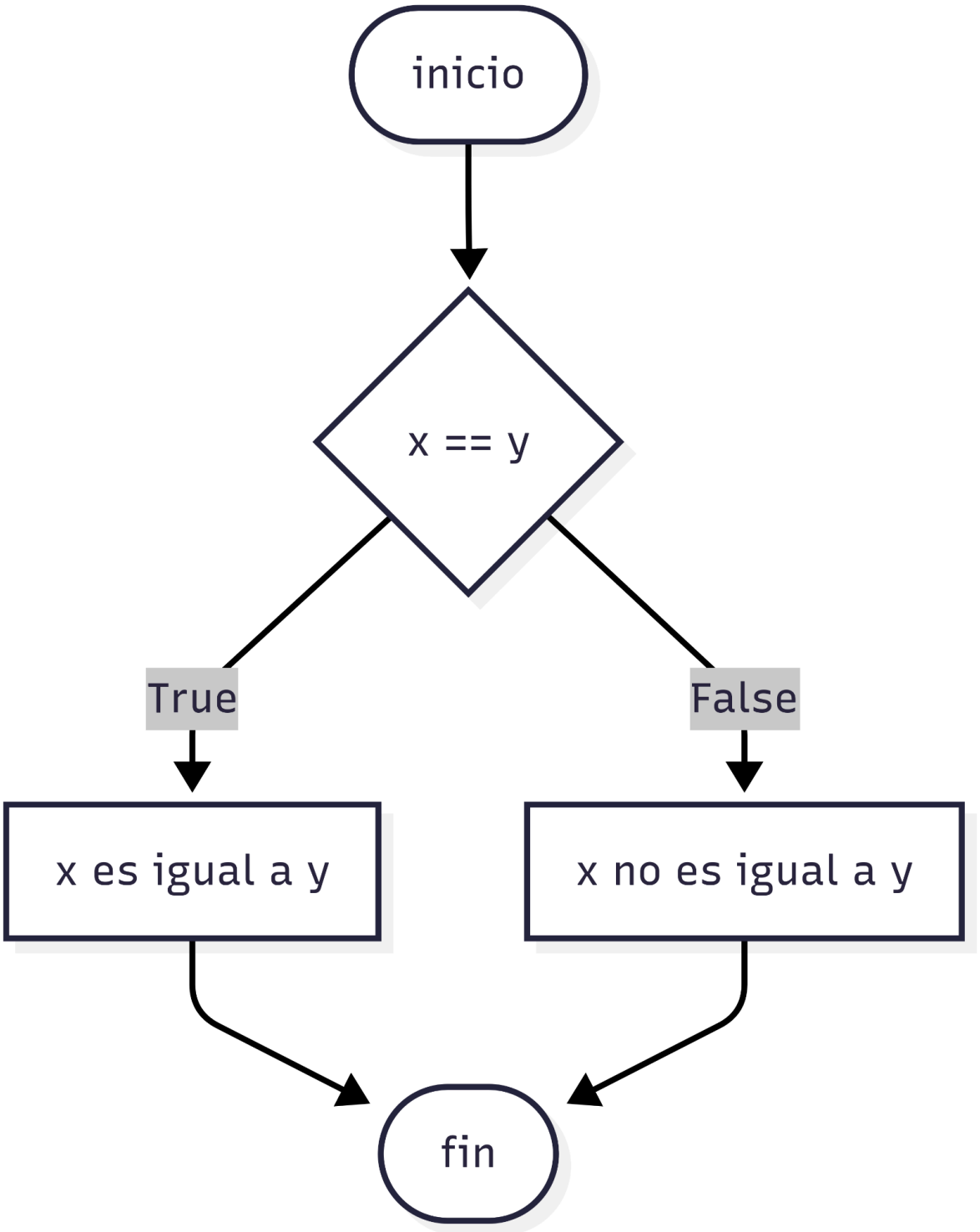
Nota cómo removimos el `or` completamente y simplemente preguntamos, "¿x no es igual a y?" Hacemos una y solo una pregunta. ¡Muy eficiente!

- Para propósitos de ilustración, también podríamos cambiar nuestro código como sigue:

```
x = int(input("¿Cuál es x? "))

y = int(input("¿Cuál es y? "))

if x == y:

    print("x es igual a y")

else:

    print("x no es igual a y")
```

Nota que el operador `==` evalúa si lo que está a la izquierda y a la derecha son iguales entre sí. El uso de signos de igual dobles es muy importante. Si usas solo un signo igual, probablemente se lanzará un error por el intérprete.

- Nuestro código puede ser ilustrado como sigue:



and

- Similar a `or`, `and` puede ser usado dentro de declaraciones condicionales.
- Ejecuta en la ventana de terminal `code grado.py`. Comienza tu nuevo programa como sigue:

```
score = int(input("Puntuación: "))

if score >= 90 and score <= 100:

    print("Calificación: A")

elif score >= 80 and score < 90:

    print("Calificación: B")

elif score >= 70 and score < 80:

    print("Calificación: C")

elif score >= 60 and score < 70:

    print("Calificación: D")

else:

    print("Calificación: F")
```

Nota que al ejecutar `python grado.py`, podrás ingresar una puntuación y obtener una calificación. Sin embargo, nota cómo hay potencial para errores.

- Típicamente, no queremos confiar nunca en que nuestros usuarios ingresen la información correcta. Podríamos mejorar nuestro código como sigue:

```
score = int(input("Puntuación: "))

if 90 <= score <= 100:

    print("Calificación: A")

elif 80 <= score < 90:
```

```
    print("Calificación: B")

elif 70 <= score < 80:

    print("Calificación: C")

elif 60 <= score < 70:

    print("Calificación: D")

else:

    print("Calificación: F")
```

Nota cómo Python te permite encadenar los operadores y condiciones de una manera bastante poco común en otros lenguajes de programación.

- Aún así, podemos mejorar nuestro programa aún más:

```
score = int(input("Puntuación: "))

if score >= 90:

    print("Calificación: A")

elif score >= 80:

    print("Calificación: B")

elif score >= 70:

    print("Calificación: C")

elif score >= 60:

    print("Calificación: D")

else:

    print("Calificación: F")
```

Nota cómo el programa es mejorado al hacer menos preguntas. Esto hace nuestro programa más fácil de leer y mucho más mantenible en el futuro.

- Puedes aprender más en la documentación de Python sobre flujo de control.

Módulo

- En matemáticas, la paridad se refiere a si un número es par o impar.
- El operador módulo `%` en programación permite ver si dos números se dividen uniformemente o se dividen y tienen un residuo.
- Por ejemplo, 4 % 2 resultaría en cero, porque se divide uniformemente. Sin embargo, 3 % 2 no se divide uniformemente y resultaría en un número distinto de cero.
- En la ventana de terminal, crea un nuevo programa escribiendo `code parity.py`. En la ventana del editor de texto, escribe tu código como sigue:

```
x = int(input("¿Cuál es x? "))

if x % 2 == 0:

    print("Par")

else:

    print("Impar")
```

Nota cómo nuestros usuarios pueden escribir cualquier número 1 o mayor para ver si es par o impar.

Creando Nuestra Propia Función de Paridad

- Como se discutió en la Lectura 01, ¡encontrarás útil crear una función propia!
- Podemos crear nuestra propia función para verificar si un número es par o impar. Ajusta tu código como sigue:

```
def main():

    x = int(input("¿Cuál es x? "))

    if es_par(x):

        print("Par")

    else:
```

```
        print("Impar")

def es_par(n):
    if n % 2 == 0:
        return True
    else:
        return False

main()
```

Nota que nuestra declaración `if es_par(x)` funciona aunque no hay un operador ahí. Esto es porque nuestra función devuelve un `bool` (Booleano), `True` o `False`, de vuelta a la función principal. La declaración `if` simplemente evalúa si `es_par` de x es verdadero o falso o no.

Pythonic

- En el mundo de la programación, hay tipos de programación que se llaman "Pythonic" por naturaleza. Es decir, hay maneras de programar que a veces solo se ven en la programación de Python. Considera la siguiente revisión a nuestro programa:

```
def main():
    x = int(input("¿Cuál es x? "))
    if es_par(x):
        print("Par")
    else:
        print("Impar")

def es_par(n):
    return True if n % 2 == 0 else False
```

```
main()
```

Nota que esta declaración de retorno en nuestro código es casi como una oración en inglés. Esta es una manera única de codificar que solo se ve en Python.

- Podemos revisar nuestro código aún más y hacerlo más y más legible:

```
def main():
    x = int(input("¿Cuál es x? "))
    if es_par(x):
        print("Par")
    else:
        print("Impar")

def es_par(n):
    return n % 2 == 0

main()
```

Nota que el programa evaluará lo que está ocurriendo dentro de `n % 2 == 0` como `True` o `False` y simplemente devolver eso a la función principal.

match

- Similar a las declaraciones `if`, `elif`, y `else`, las declaraciones `match` pueden ser usadas para ejecutar condicionalmente código que coincida con ciertos valores.
- Considera el siguiente programa:

```
name = input("¿Cuál es tu nombre? ")

if name == "Harry":
    print("Gryffindor")
```

```
elif name == "Hermione":
    print("Gryffindor")
elif name == "Ron":
    print("Gryffindor")
elif name == "Draco":
    print("Slytherin")
else:
    print("¿Quién?")
```

Nota que las primeras tres declaraciones condicionales imprimen la misma respuesta.

- Podemos mejorar este código ligeramente con el uso de la palabra clave `or`:

```
name = input("¿Cuál es tu nombre? ")
if name == "Harry" or name == "Hermione" or name == "Ron":
    print("Gryffindor")
elif name == "Draco":
    print("Slytherin")
else:
    print("¿Quién?")
```

Nota que el número de declaraciones `elif` ha disminuido, mejorando la legibilidad de nuestro código.

- Alternativamente, podemos usar declaraciones `match` para mapear nombres a casas. Considera el siguiente código:

```
name = input("¿Cuál es tu nombre? ")
```

```
match name:
    case "Harry":
        print("Gryffindor")
    case "Hermione":
        print("Gryffindor")
    case "Ron":
        print("Gryffindor")
    case "Draco":
        print("Slytherin")
    case _:
        print("¿Quién?")
```

Nota el uso del símbolo `_` en el último caso. Esto coincidirá con cualquier entrada, resultando en un comportamiento similar a una declaración `else`.

- Una declaración match compara el valor que sigue a la palabra clave `match` con cada uno de los valores que siguen a las palabras clave `case`. En el caso de que se encuentre una coincidencia, la respectiva sección de código indentada se ejecuta, y el programa detiene la coincidencia.

- Podemos mejorar el código:

```
name = input("¿Cuál es tu nombre? ")

match name:
    case "Harry" | "Hermione" | "Ron":
        print("Gryffindor")
    case "Draco":
```

```
        print("Slytherin")

    case _:

        print("¿Quién?")
```

Nota, el uso de la barra vertical única `|`. Al igual que la palabra clave `or`, esto nos permite verificar múltiples valores en la misma declaración `case`.

Resumen

Ahora tienes el poder dentro de Python para usar declaraciones condicionales para hacer preguntas y hacer que tu programa tome acción en consecuencia. En esta lección, discutimos…

- Condicionales;
- Declaraciones `if`;
- Flujo de control, `elif`, y `else`;
- `or`;
- `and`;
- Módulo;
- Creando tu propia función;
- Codificación Pythonic;
- y `match`.