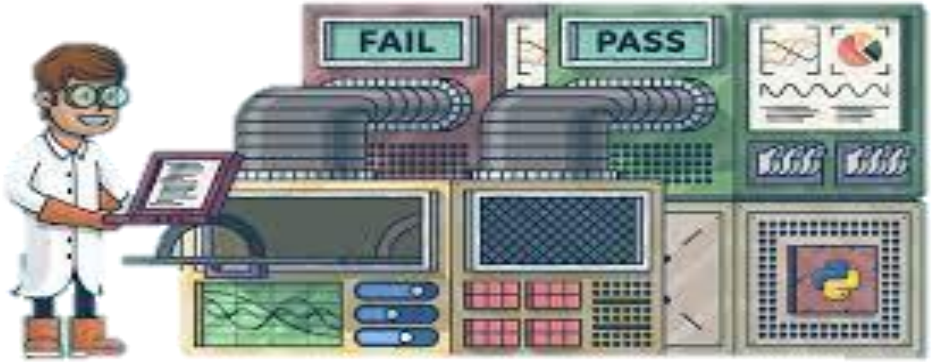
Introduction to Python Programming

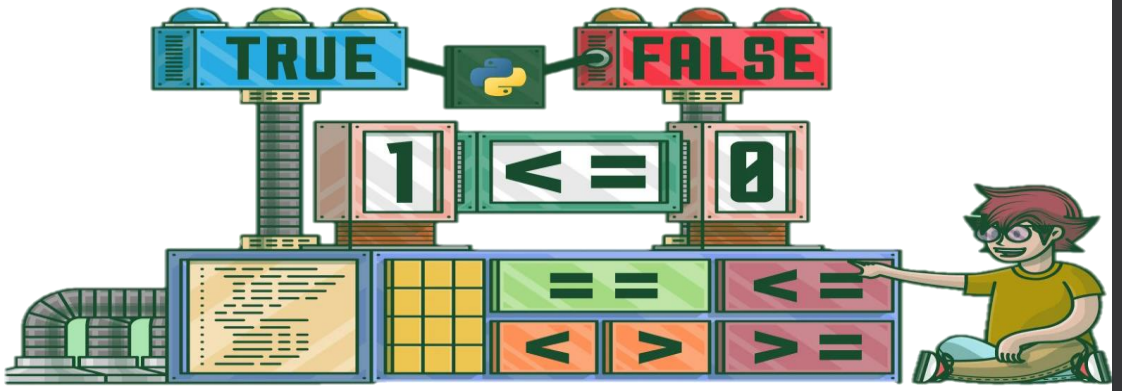
Dr.Muhammad Sajjad

RA.Muhammad Ayaz

Introduction to Conditional Statements

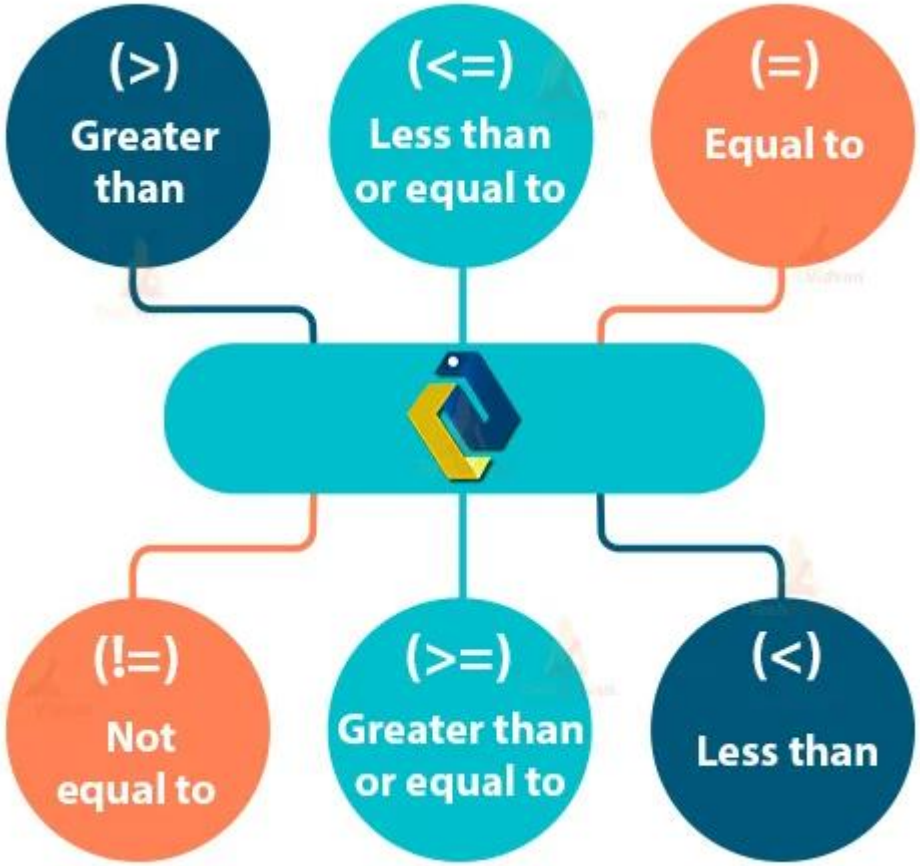
- In programming, conditional statements allow the program to take different actions based on whether an expression (condition) evaluates to **True** or **False**.
- A **condition** is a test that results in either True or False.
- Conditional statements help control the flow of a program.
- **Why Use Conditions?**
  Conditions allow the program to:
- Perform specific actions only if a condition is met.
- Skip actions when a condition is not met.
- **Real-Life Example**: A traffic light:
- If the light is green, cars go.
- If the light is red, cars stop.
  In Python, we use **if** statements to do the same with code.





Conditions in Python

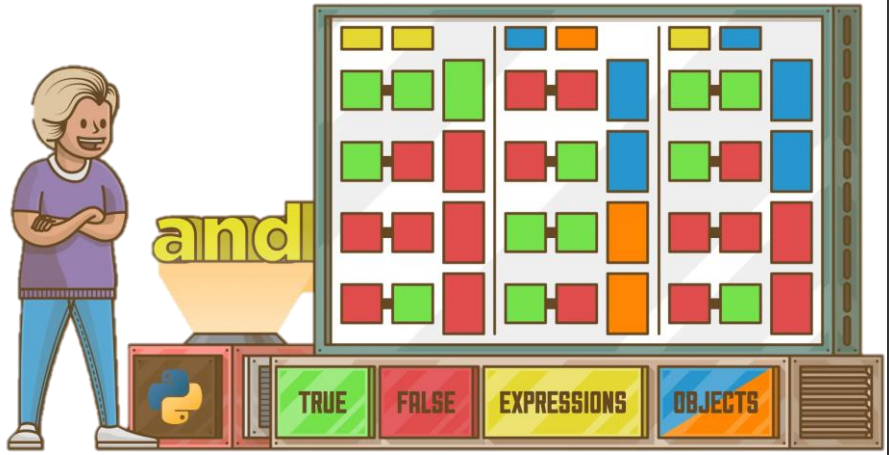
- A condition is an expression that evaluates to True or False. It is used in control flow statements like if, if-else, and if-elif-else.
- **Comparison Operators:**
  - ==: Equal to
  - !=: Not equal to
  - >: Greater than
  - <: Less than
  - >=: Greater than or equal to
  - <=: Less than or equal to
- **Example**:
  - x = 10
  - if x > 5:
  - print("x is greater than 5")
- In this example, **X > 5** is the condition that will evaluate to True or False.

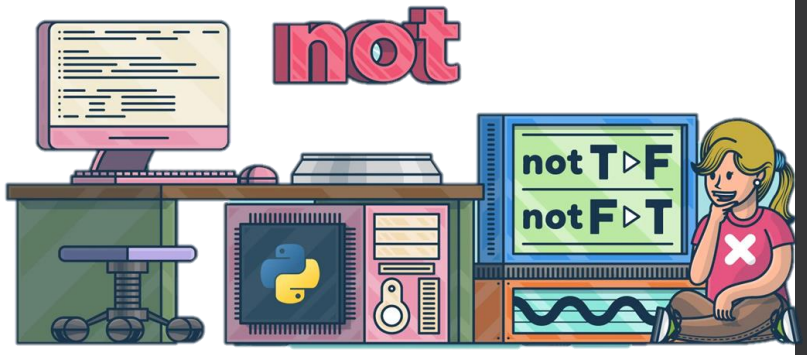


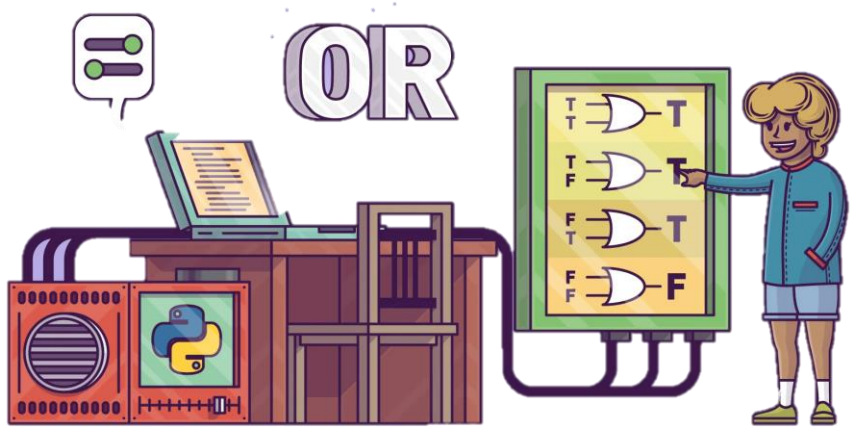
Logical Operators in Python

Logical operators are used to combine multiple conditions and return a True or False result.

- **and:** True if both conditions are True.
- **or:** True if at least one condition is True.
- **not:** Reverses the result of a condition (True becomes False and vice versa).
- **Examples**:
  - **# and operator**
  - age = 20
  - if age > 18 and age < 30:
    - print("You're a young adult")
  - **# or operator**
  - grade = 'B'
  - if grade == 'A' or grade == 'B':
  - print("You passed")







If Statements

- If Statements are used to execute a block of code only if a specified condition is True.
- The condition is placed after the if keyword, followed by a **colon :**.
- If the condition is true, the code inside the if block will be executed.
- If the condition is false, the code inside the block is skipped.
- **Example**:
  - num = 10
  - if num > 5:
    - print("The number is greater than 5")
- In this example, if num is greater than 5, the message is printed.
- If num is less than or equal to 5, nothing happens.

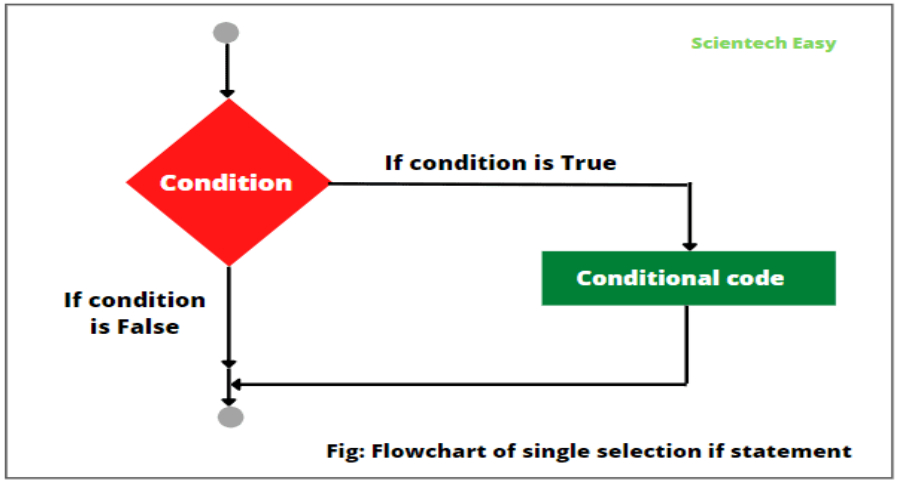
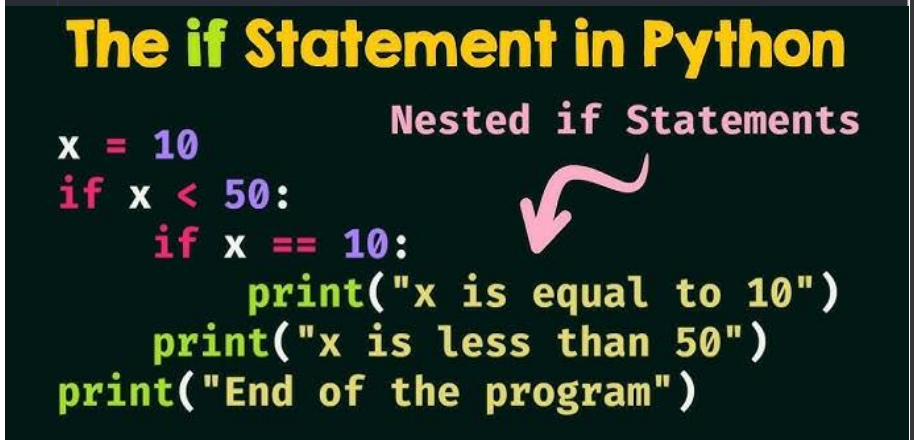


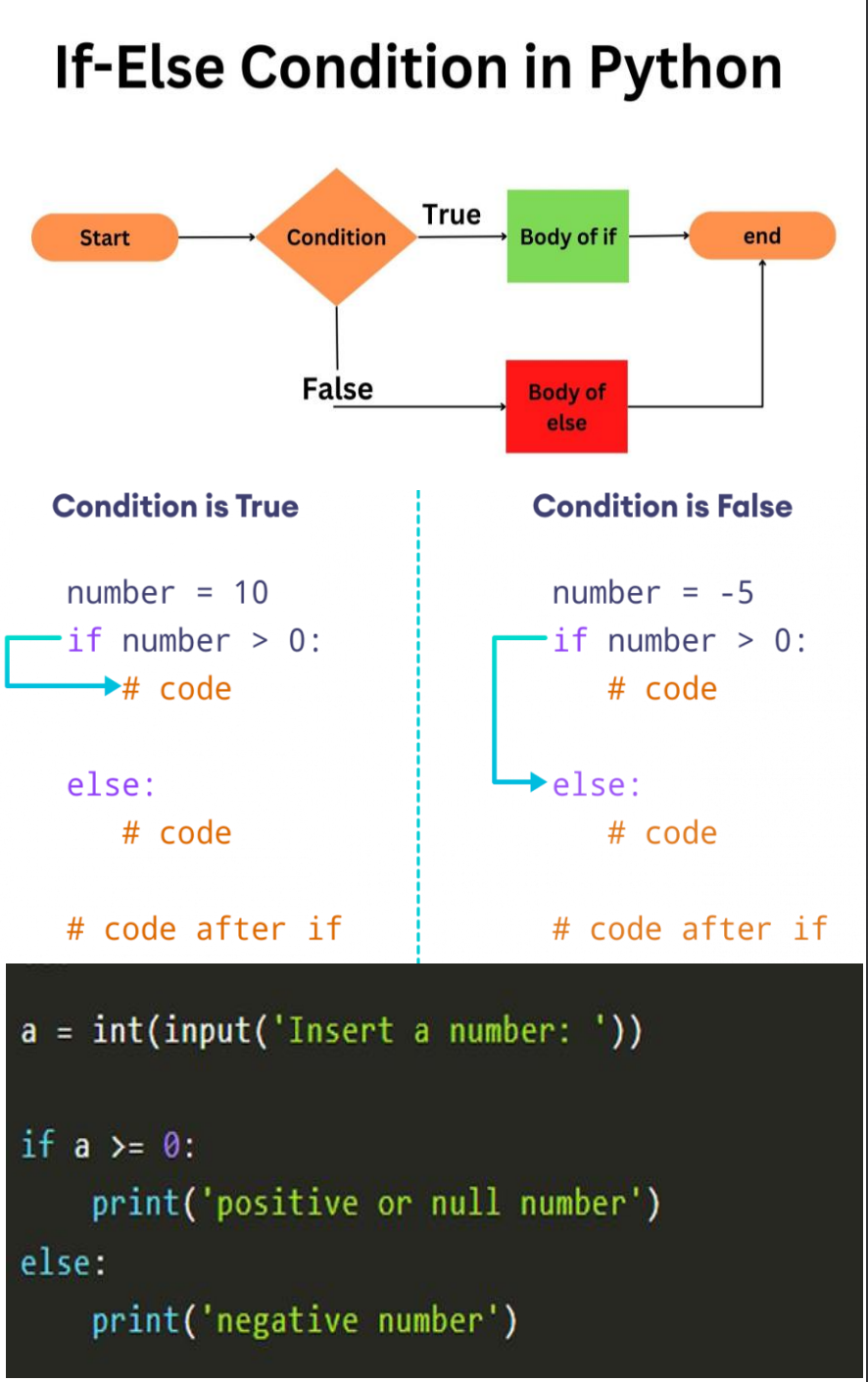
Fig: Flowchart of single selection if statement

```
name = 'Jason'
if name == 'Jason':
    print("Hello Jason, Welcome")
else:
    print("Sorry, I don't know you")
```



If-Else Statements

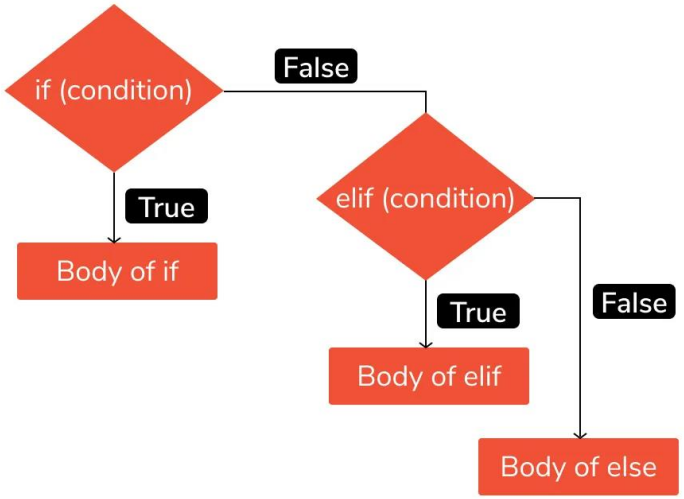
- If-Else Statements allow the program to take two possible actions:
  - One action when the condition is True.
  - Another action when the condition is False.
- The condition is checked with if, and if it's false, the code inside the else block is executed.
- The else block ensures that an alternative block of code runs if the condition is not met.
- **Example**:
  - num = 3
  - if num > 5:
    - print("The number is greater than 5")
  - else:
    - print("The number is not greater than **5")**
- In this case, if **num** is greater than 5, the first message is printed.
- If **num** is less than or equal to 5, the second message is printed from the else block.

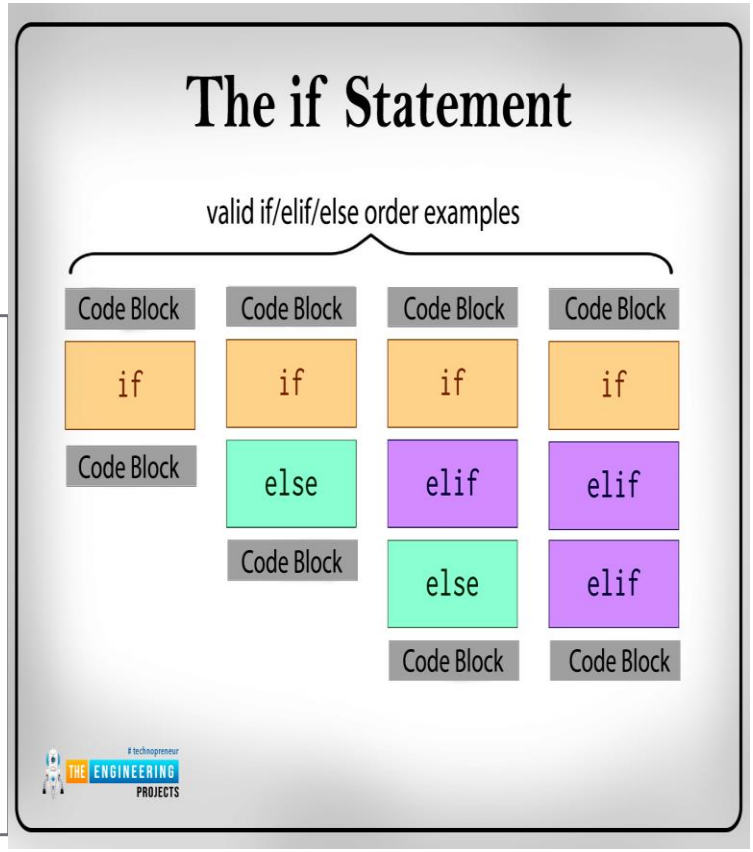


If-Elif-Else Statements

- If-Elif-Else Statements are used when multiple conditions need to be evaluated.
- The program checks the first condition with if.
- If the first condition is True, its block of code is executed.
- If the first condition is False, the program checks the next condition using elif (else if).
- You can have multiple elif conditions.
- If none of the conditions are True, the else block is executed as a fallback.
- **Example**
  - score = 75
  - if score >= 90:
    - print("Grade: A")
  - elif score >= 80:
    - print("Grade: B")
  - elif score >= 70:
    - print("Grade: C")
  - else:
    - print("Grade: D")

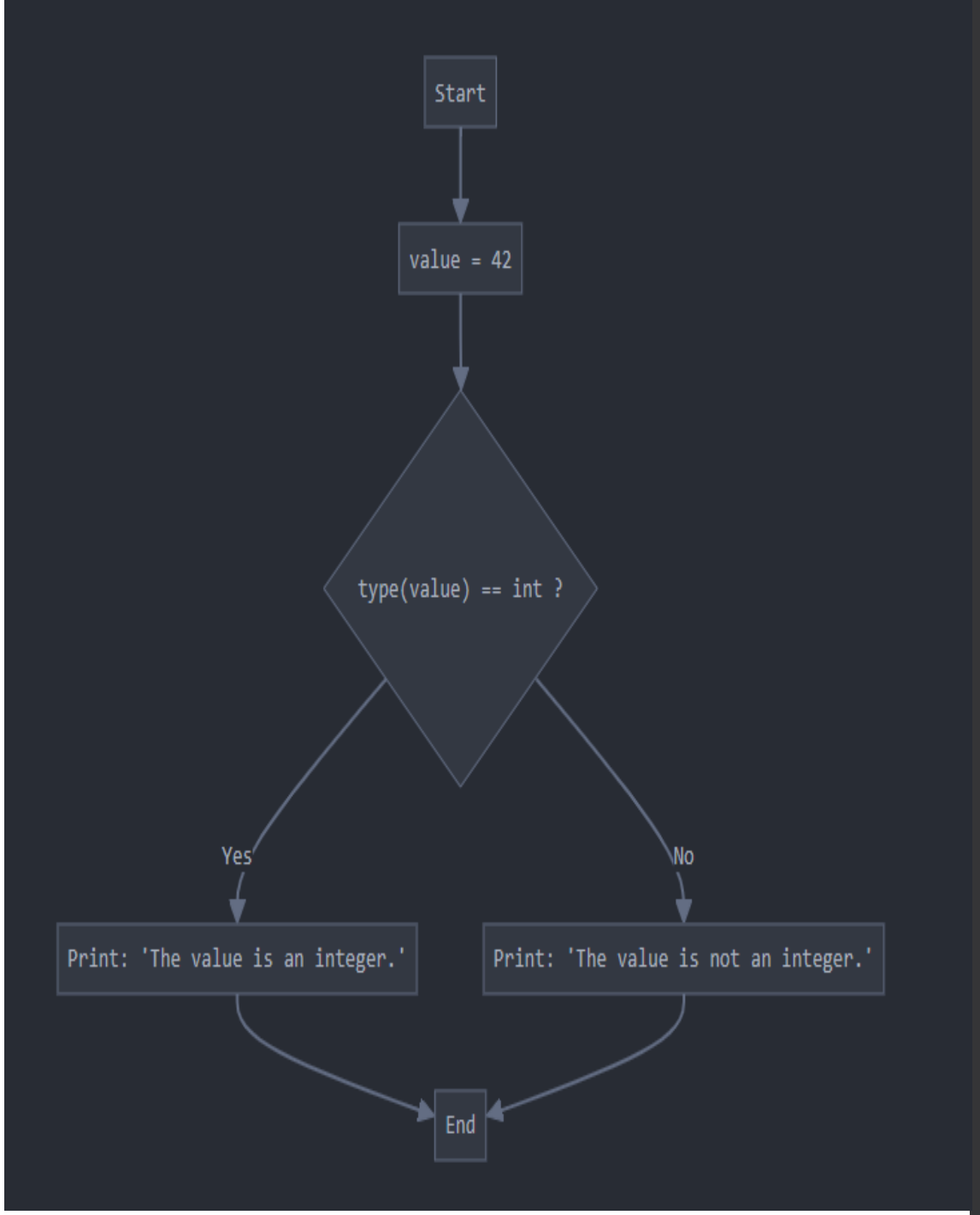
- If the score is 90 or higher, "Grade: A" is printed.
- If the score is between 80 and 89, "Grade: B" is printed.
- If the score is between 70 and 79, "Grade: C" is printed.
- If none of the conditions are met, "Grade: D" is printed from the else block.





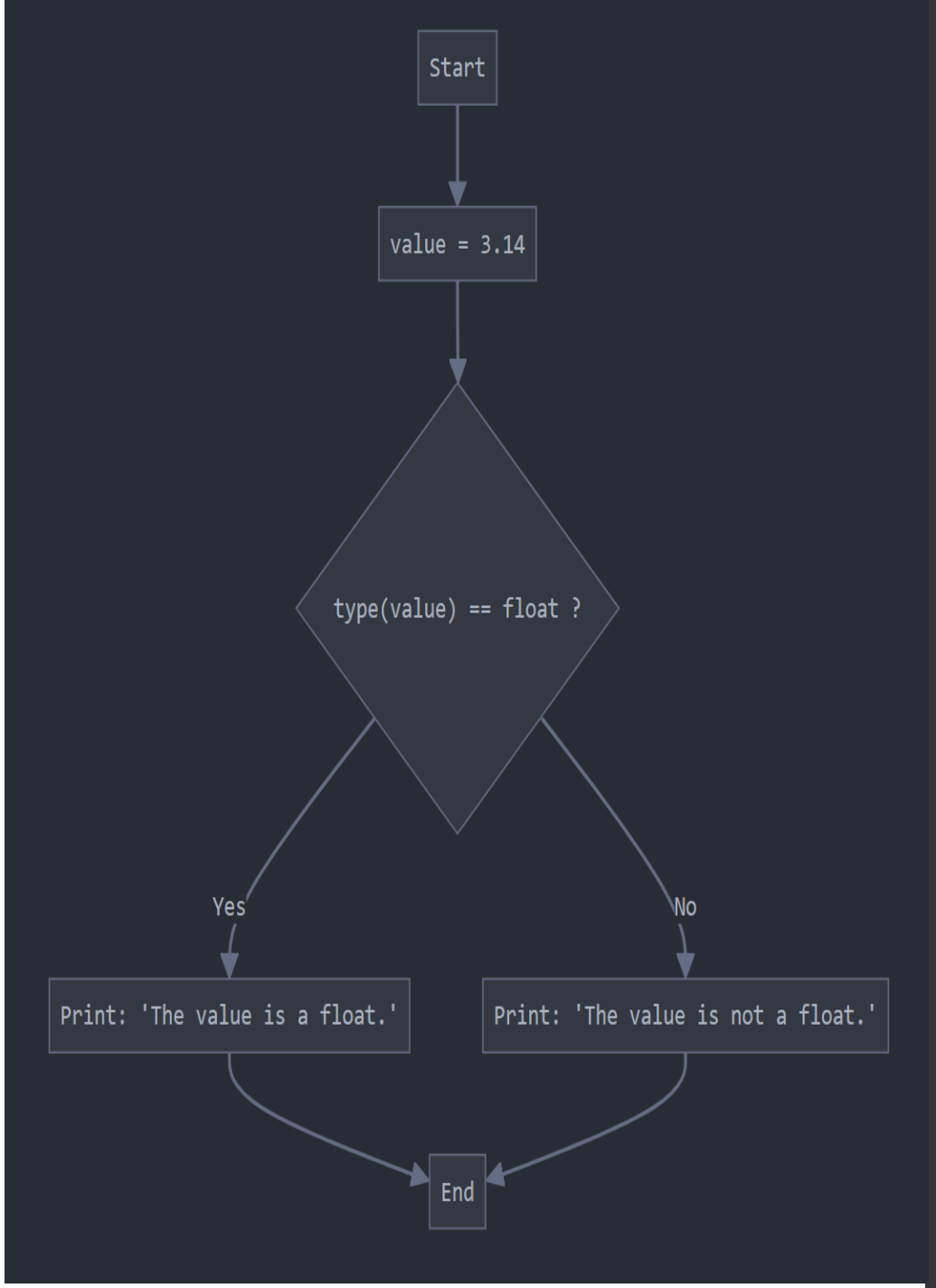
Integer Data Type with if,else

- **Objective:** To demonstrate how to use if-elif-else statements to check if a value is of type int.
- **Explanation:**
- **Data Type**: int (integer) is used for whole numbers, which can be positive or negative (e.g., 5, -3, 100).
- **Code Example**:
  - value = 42 # Example integer
  - if type(value) == int:
  - print("The value is an integer.")
  - else:
  - print("The value is not an integer.")
- **Explanation:** type(value) == int checks if the type of value is exactly int. If True, it prints "The value is an integer."



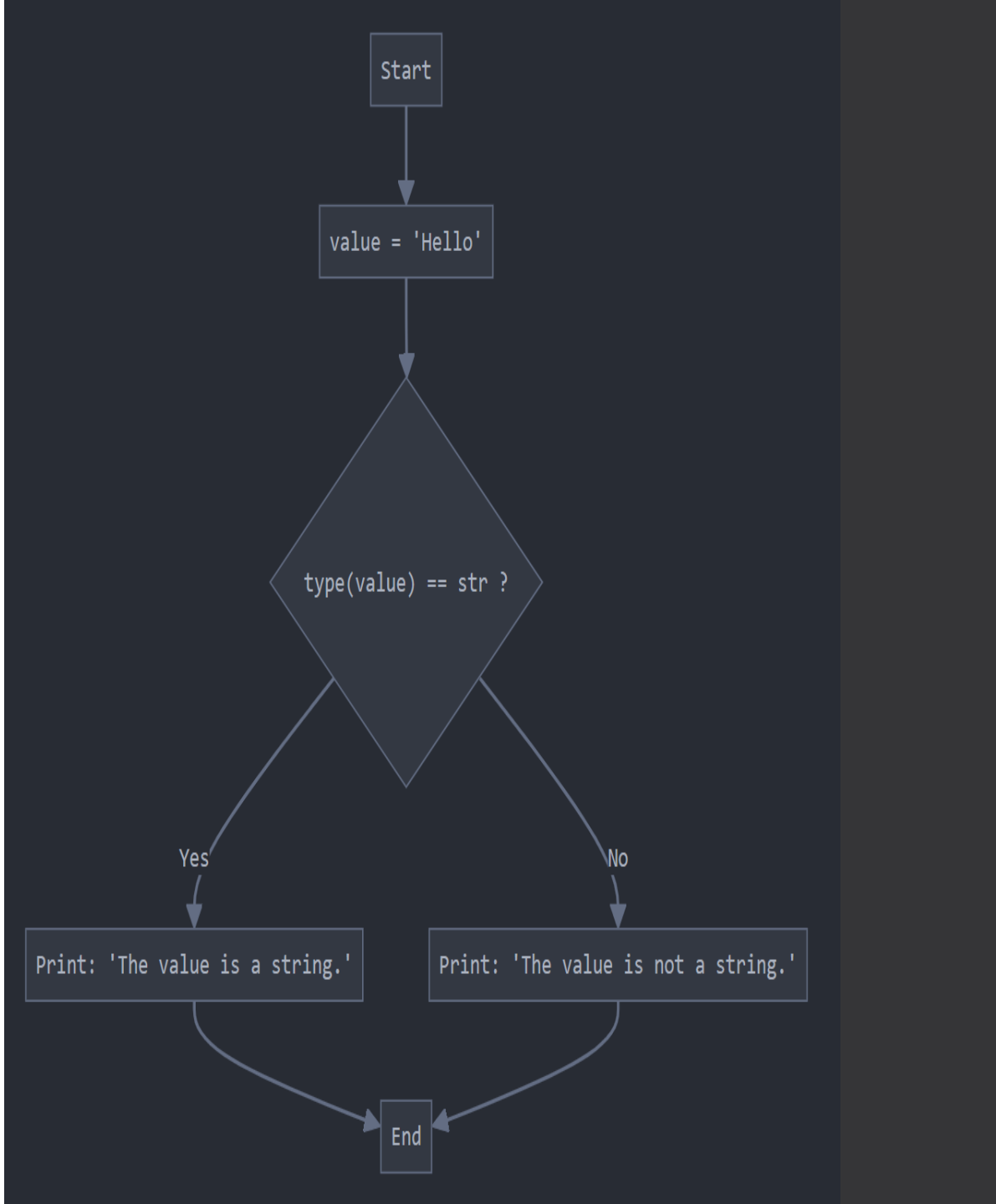
Float Data Type

- **Explanation:**
- **Data Type:** float represents decimal numbers (e.g., 3.14, -0.99).
- **Code Example:**
  - value = 3.14 # Example float
  - if type(value) == float:
  - print("The value is a float.")
  - else:
  - print("The value is not a float.")
- **Type Checking**: Use type(value) == float to directly check if the variable value is of type float.
- **Precision Handling**: float is used for numbers that require decimal precision, crucial in scientific calculations and financial applications.
- **Usage Context**: Commonly used in operations involving division or measurements where fractional values are expected.
- **Example Value:** 3.14 is a floating-point numbe.



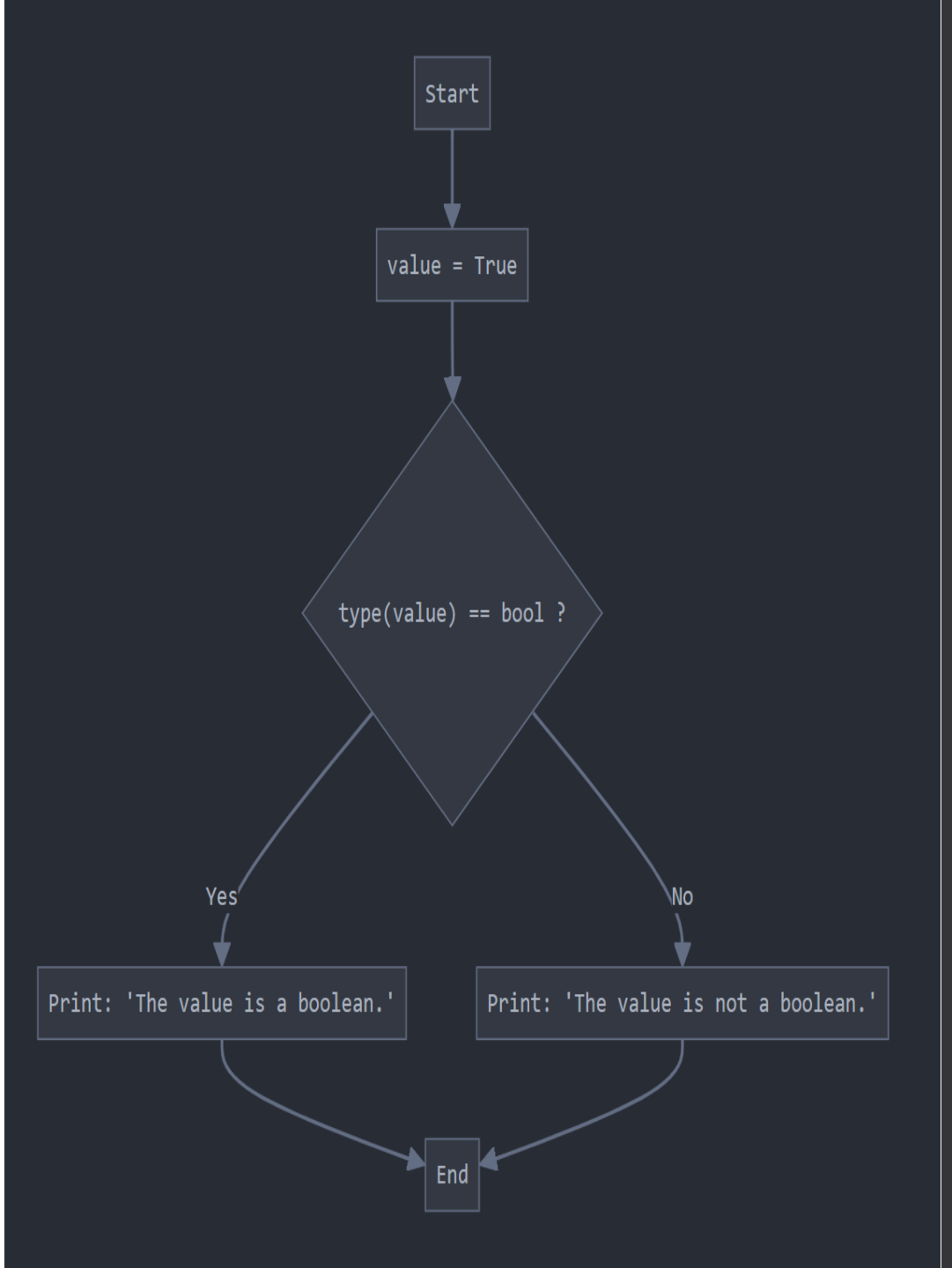
String Data Type

- **Explanation:**
- **Data Type:** str (string) represents text data (e.g., "Hello", "Python").
- **Code Example:**
  - value = "Hello" # Example string
  - if type(value) == str:
  - print("The value is a string.")
  - else:
  - print("The value is not a string.")
- **Type Checking:** Use type(value) == str to verify if the variable value is a string.
- **Text Handling:** Strings are used for storing and manipulating text, which is essential for user interfaces and data processing.
- **Concatenation:** Strings can be concatenated using +, for example, "Hello" + " World" results in "Hello World".
- **Example Value:** "Hello" is a string, while 100 is an integer.



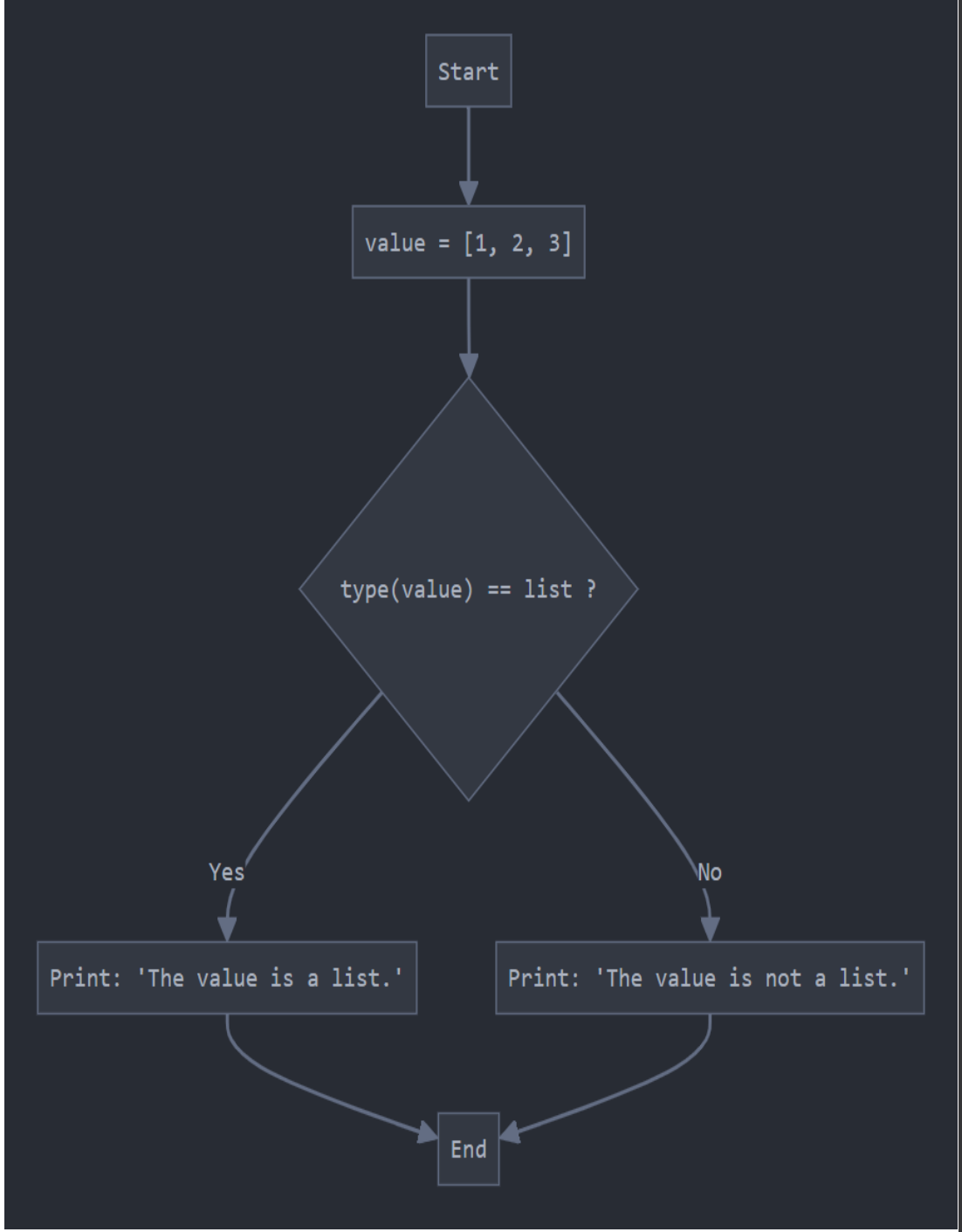
Boolean Data Type

- **Explanation:**
- **Data Type:** bool represents logical values True and False.
- **Code Example:**
  - value = True # Example boolean
  - if type(value) == bool:
  - print("The value is a boolean.")
  - else:
  - print("The value is not a boolean.")
- **Type Checking**: Use type(value) == bool to check if the variable value is of type bool.
- **Logical Operations**: Booleans are used in conditional statements and loops to control the flow of a program.
- **Boolean Operators:** Use logical operators like and, or, and not to combine boolean values.
- **Example Value:** True and False are boolean values, while "True" is a string.



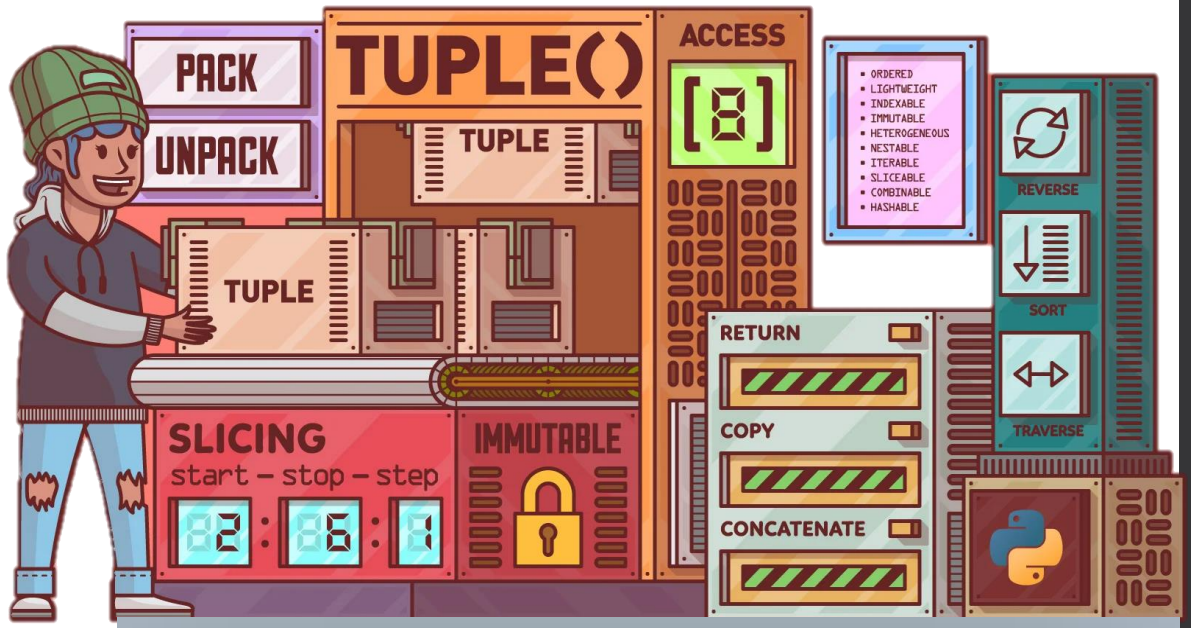
List Data Type

- **Explanation:**
- **Data Type:** list is an ordered, mutable collection of items (e.g., [1, 2, 3], ["apple", "banana"]).
- Code **Example**
  - value = [1, 2, 3] # Example list
  - if type(value) == list:
  - print("The value is a list.")
  - else:
  - print("The value is not a list.")
- **Type Checking:** Use type(value) == list to determine if value is a list.
- **Mutability:** Lists are mutable, meaning their contents can be changed after creation.
- **List Operations:** Lists support operations like indexing, slicing, appending, and extending.
- **Example Value:** [1, 2, 3] is a list



Tuple Data Type

- **Explanation:**
- **Data Type:** tuple is an ordered, immutable collection of items (e.g., (1, 2, 3), ("a", "b")).
- **Code Example**
  - value = (1, 2, 3) # Example tuple
  - if type(value) == tuple:
  - print("The value is a tuple.")
  - else:
  - print("The value is not a tuple.")
- **Type Checking:** Use type(value) == tuple to verify if value is a tuple.
- **Immutability**: Tuples are immutable, meaning their contents cannot be changed after creation.
- **Use Cases:** Tuples are often used for fixed collections of items and can be used as dictionary keys.
- **Example Value**: (1, 2, 3) is a tuple, while [1, 2, 3] is a list.




```
#declaring a list
list_a = [1,2,3]
print("Intial list_a -> ",list_a)

#declaring a tuple
tuple_a = (1,2,3)
print("Initial tuple_a ->", tuple_a)
```

Dictionary Data Type

- **Explanation:**
- **Data Type:** dict (dictionary) represents a collection of key-value pairs (e.g., {"key": "value"}, {1: "one"}).
- **Code Example**
  - value = {"key": "value"} # Example dictionary
  - if type(value) == dict:
  - print("The value is a dictionary.")
  - else:
  - print("The value is not a dictionary.")
- **Type Checking**: Use type(value) == dict to determine if value is a dictionary.
- **Key-Value Pairs**: Dictionaries store data in key-value pairs, allowing for efficient lookups and modifications.
- **Common Operations:** You can add, remove, and access values using their keys.
- **Example Value:** {"key": "value"} is a dictionary, while ["key", "value"] is a list.