Introduction to Programming and Problem Solving

Week 5: Decision Statements

By Lemi Agrey Oliver

Department of Information Technology

Kumi University

Email: codingissweet@gmail.com

Recap of week 4 lecture

- Welcome to our Week 5 lecture! Before we delve into the materials for this week, let's take a moment to recap our last lecture. As you may recall, in our previous session, we covered the fundamental topic of operators and expressions in programming.
- We defined an operator as a symbol used to perform various operations on data or values.

Recap of week 4 lecture+

- These operations can range from basic arithmetic calculations to more complex manipulations. It's important to note that the data or value being operated on by the operator is referred to as an operand.
- In our discussion, we explored a variety of operator types, including logical operators (such as AND, OR, and NOT), assignment operators (used to assign values to variables), bitwise operators (for working with individual bits of data), and membership operators (used to check if a value belongs to a set or sequence).

Recap of week 4 lecture++

- Additionally, we delved into the concept of operator precedence, which is essentially the order in which operators are evaluated within an expression. Understanding operator precedence is crucial as it dictates how the expression's components are combined and calculated.
- Furthermore, we covered operator associativity, which is a set of rules governing the order of evaluation when multiple operators of the same precedence appear in an expression.

Recap of week 4 lecture++

- This ensures that expressions are evaluated in a consistent and predictable manner.
- By mastering these concepts, you'll have a solid foundation for working with operators and expressions in programming.
- Now that we've briefly reviewed what we discussed last week, it's time to dive into our topic for this week, which is decision statements. Let us now look at our content for this week.

Content

- Introduction
- Boolean Type
- Boolean Operators
- Using Numbers with Boolean Operators
- Using String with Boolean Operators
- Boolean Expressions and Relational Operators
- Decision Making Statements
- Conditional Expressions

Introduction to Decision Statements

- Decision statements, also known as conditional statements, play a crucial role in programming.
- A statement can be thought as an instruction that can be interpreted by the Python interpreter. [1]
- They allow a program to make decisions based on certain conditions or criteria.
- This capability is important because it permits programs to behave dynamically and respond to different scenarios, making them more versatile and powerful tools for various tasks

Boolean type

- The **Boolean** type, often referred to as a "Boolean," is a fundamental data type in computer programming and computer science.
- In Python, the Boolean type is represented by the keyword bool.
- Boolean values can be created using the constants True and False. Boolean expressions can be created using the logical operators and, or, and not.
- It represents a binary value that can have one of two possible states: true or false. Booleans are typically used for making decisions and controlling the flow of a program through conditional statements and logical expressions.

Key characteristics and uses of the Boolean type

- **Values,** A Boolean variable can have one of two possible values: true or false. These values are often used to represent the truth or falsity of a condition or statement.
- **Logical Operations,** Booleans are commonly used in logical operations such as AND, OR, and NOT. These operations allow you to combine and manipulate Boolean values to make decisions in your code.
- AND, The result is true if both operands are true. A==5 and B==5 #True

Key characteristics and uses of the Boolean type

- **Comparison Operators,** Booleans are often used in conjunction with comparison operators to compare values and create conditions. Common comparison operators include "==", "!=", "<", ">", "<=", and ">=".
- **Conditional Statements,** Booleans are crucial for writing conditional statements like if statements, which allow you to execute different blocks of code depending on whether a condition is true or false.

Key characteristics and uses of the Boolean type

```
if condition:
    # Code to execute if the condition is true
else:
    # Code to execute if the condition is false
```

- **Loop Control,** Booleans are also used to control loops. For example, you can use a Boolean variable as a flag to determine when to exit a loop.
- **Function Returns,** Functions can return Boolean values to indicate success or failure. For example, a function that checks if a file was successfully opened might return true for success and false for failure.

Key characteristics and uses of the Boolean type+

- In many programming languages, including Python, JavaScript, C++, and Java, the Boolean type is a built-in data type, and you can create Boolean variables to store true or false values. For example, in Python:

```
x = True
y = False
```

- Boolean logic is a fundamental concept in computer science and programming, and it plays a crucial role in decision-making and control flow within programs

The not operator

- **The not operator** is a logical operator used to negate or reverse the value of a Boolean expression or variable. In most programming languages, including Python, JavaScript, and others, the not operator is used to flip the value of a Boolean expression. If the expression is True, applying not to it makes it False, and if the expression is False, applying not to it makes it True.
- The basic syntax of the not operator:

The not operator

- result = not expression
- expression: This is the Boolean expression you want to negate.
- not: This is the not operator itself.
- result: The result of applying not to the expression. It will be the opposite of the original value of the expression.
- Here are some examples of how the not operator can be used in Python:
- x = True

The not operator

```
y = False
result1 = not x  # result1 will be False because not True is False
result2 = not y  # result2 will be True because not False is True
print(result1)  # Output: False
print(result2)  # Output: True
```

The not operator

- The not operator is commonly used in conditional statements and logical expressions to control program flow and make decisions. For example:

```
1. user_input = input("Do you want to continue? (yes/no): ")
2. if not (user_input == "yes"):
3.     print("Exiting program.")
4. else:
5.     print("Continuing program.")
```

The not operator

- In this example, the not operator is used to check if the user's input is not equal to "yes." If it's not equal to "yes," the program exits; otherwise, it continues.
- The not operator is an essential tool for working with Boolean values and logical conditions, allowing you to easily invert the truth value of expressions to control program behavior.
- **The and Operator**
- The and operator is a logical operator used to combine multiple Boolean expressions or variables.

The and Operator

- It returns True if all of the expressions or variables it operates on are True, and False otherwise. In other words, it performs a logical "AND" operation.
- Basic syntax of the and operator:
- result = expression1 and expression2
- expression1 and expression2: These are the Boolean expressions you want to combine using the and operator.
- and: This is the and operator itself.

The and Operator

- result: The result of applying and to the two expressions. It will be True if both expression1 and expression2 are True, and False otherwise.
- Here are some examples of how the and operator can be used in Python:

```
1. x = True
2. y = False
3. result1 = x and x   # result1 will be True because
  both x and x are True
4. result2 = x and y   # result2 will be False because
  one of them (y) is False
```

The and Operator

- print(result1) # Output: True
- print(result2) # Output: False
- The and operator is commonly used in conditional statements and logical expressions to create more complex conditions. For example:
- age = 25
- is_student = True
- if age >= 18 and is_student:

The and Operator

- In this example, the and operator is used to check if both age is greater than or equal to 18 and is_student is True. If both conditions are met, the program prints "You are an adult student."
- The and operator is useful for combining multiple conditions in a way that ensures all conditions must be True for the combined condition to be True. It's an essential tool for controlling program flow based on complex logical conditions.

The or Operator

- **The or Operator**
- The or operator is another logical operator used in computer programming to combine multiple Boolean expressions or variables. It returns True if at least one of the expressions or variables it operates on is True, and False if all of them are False. In other words, it performs a logical "OR" operation.
- **Basic syntax of the or operator**
- result = expression1 or expression2

The or Operator

- expression1 and expression2: These are the Boolean expressions you want to combine using the or operator.
- or: This is the or operator itself.
- result: The result of applying or to the two expressions. It will be True if at least one of expression1 or expression2 is True, and False if both are False.
- Here are some examples of how the or operator can be used in Python:

The or Operator

```
1. x = True
2. y = False
3. result1 = x or x  # result1 will be True because at least
   one x is True
4. result2 = x or y  # result2 will be True because one of
   them (x) is True
5. result3 = y or y  # result3 will be False because both y's
   are False
6. print(result1)  # Output: True
7. print(result2)  # Output: True
8. print(result3)  # Output: False
```

The or Operator

- The or operator is commonly used in conditional statements and logical expressions to create more complex conditions. For example:

```
1.  is_sunny = True
2.  is_warm = True
3.  if is_sunny or is_warm:
4.      print("It's a nice day to go outside.")
5.  else:
6.      print("Maybe it's better to stay indoors.")
```

The or Operator

- In this example, the or operator is used to check if either is_sunny is True or is_warm is True. If at least one of these conditions is met, the program prints "It's a nice day to go outside.“
- The or operator is useful for creating conditions where at least one of several conditions needs to be satisfied for an action to be taken. It's an important part of controlling program flow based on logical conditions.

USING NUMBERS WITH BOOLEAN OPERATORS

- In computer programming, you can use numbers with Boolean operators to create conditions and comparisons involving numerical values.
- Here's how you can use numbers in conjunction with Boolean operators,
- Equal (==) and Not Equal (!=) Comparisons: You can compare numbers using the equal (==) and not equal (!=) operators to create Boolean expressions. For example

```
1.    y = 10
3.    is_equal = x == y  # This will be False because x is not equal to y
4.    is_not_equal = x != y  # This will be True because x is not equal to y
```

USING NUMBERS WITH BOOLEAN OPERATORS

- Greater Than (>) and Less Than (<) Comparisons: You can compare numbers using the greater than (>) and less than (<) operators to create Boolean expressions. For example:

```
1. a = 15
2. b = 20
3. is_greater = a > b  # This will be False because a
   is not greater than b
4. is_less = a < b  # This will be True because a is
   less than b
```

USING NUMBERS WITH BOOLEAN OPERATORS

- Greater Than or Equal To (>=) and Less Than or Equal To (<=) Comparisons: You can also use the greater than or equal to (>=) and less than or equal to (<=) operators to compare numbers. These operators return True if the condition is met. For example:

1. p = 30
2. q = 30

USING NUMBERS WITH BOOLEAN OPERATORS

3. is_greater_or_equal = p >= q # This will be True because p is equal to q
4. is_less_or_equal = p <= q # This will be True because p is equal to q

- Combining Numerical Conditions with Boolean Operators: You can combine numerical conditions using Boolean operators like and, or, and not to create more complex conditions.

For example:

1. temperature = 28
2. is_summer = True

USING NUMBERS WITH BOOLEAN OPERATORS

- print("It's not too hot.")
- In this example, the code checks whether it's a hot day based on both the temperature being greater than 25 and it being summer, or the temperature being greater than 30.
- Using numbers with Boolean operators allows you to create conditional logic based on numerical data, making it possible to make decisions and control program flow based on numeric comparisons and conditions.

USING STRING WITH BOOLEAN OPERATORS

- In programming, you can use strings with Boolean operators to perform various types of string comparisons and create conditions based on string values. Here are some common ways to use strings with Boolean operators:
- Equality and Inequality (== and!=) Comparisons: You can use the equality (==) and inequality (!=) operators to compare strings for equality or inequality:
- name = "Alice"

USING STRING WITH BOOLEAN OPERATORS

- In programming, you can use strings with Boolean operators to perform various types of string comparisons and create conditions based on string values. Here are some common ways to use strings with Boolean operators:
- Equality and Inequality (== and!=) Comparisons: You can use the equality (==) and inequality (!=) operators to compare strings for equality or inequality:
- name = "Alice"
- is_alice = name == "Alice" # This will be True
- is_bob = name != "Bob" # This will be True

String Concatenation with and

- String Concatenation with and: You can concatenate strings using the + operator and use the and operator to create conditions based on the concatenation of strings:

```
1.  fname="Kim"
2.  lname="Hanun"
3.  full_name=fname +" "+lname
4.  if 'Kim' in full_name and 'Hanun':
5.      print("Welcome on board Prof. "+full_name+" We are excited to have you sir!")
```

String Length Comparisons

- String Length Comparisons: You can use the length of strings (obtained using len()) in combination with numerical comparisons and Boolean operators:
- password = "P@ssw0rd"
- is_long_enough = len(password) >= 8 # Check if the password is at least 8 characters long
- contains_digit = any(char.isdigit() for char in password) # Check if the password contains a digit
- if is_long_enough and contains_digit:
- print("Password is strong.")

Substring Checks with in and Combining String Conditions with or

- Substring Checks with in: You can use the in operator to check if one string is contained within another:

```
1.    strr= “Coding is the sweetest thing ever and you can do nothing about it!"
2.    prog = “Sweetest" in strr
3.    about = “Kumi University" in strr
```

- Combining String Conditions with or: You can use the or operator to create conditions where at least one of multiple string conditions must be satisfied:

```
user_input = input("Enter 'yes' or 'no': ")
```

Combining String Conditions with or

```
1. if user_input == "yes" or user_input == "no":
2.     print("You entered either 'yes' or 'no'.")
3. else:
4.     print("Invalid input.")
```

- These examples demonstrate how you can use strings with Boolean operators to create conditions, perform string comparisons, and make decisions based on string values. String manipulation and comparisons are essential in many programming tasks, including input validation, text processing, and decision-making in your code.

BOOLEAN EXPRESSIONS AND RELATIONAL OPERATORS

- Boolean expressions are expressions that evaluate to either true or false. Relational operators, also known as comparison operators, are used to compare values within these expressions. These operators are fundamental in programming for making decisions and controlling the flow of a program. Here are some common relational operators and how they work in Boolean expressions:
- Equal to (==): This operator checks if two values are equal. If they are equal, it returns true; otherwise, it returns false.

```
1. x = 5
2. y = 5
3. result = x == y  # result will be True
```

BOOLEAN EXPRESSIONS AND RELATIONAL OPERATORS

- Not equal to (!=): This operator checks if two values are not equal. If they are not equal, it returns true; otherwise, it returns false.

```
1.  a = 10
2.  b = 20
3.  result = a != b  # result will be True
```

- Greater than (>): This operator checks if the left operand is greater than the right operand. If it is, it returns true; otherwise, it returns false.

```
1.  p = 15
2.  q = 10
3.  result = p > q  # result will be True
```

BOOLEAN EXPRESSIONS AND RELATIONAL OPERATORS

- Less than (<): This operator checks if the left operand is less than the right operand. If it is, it returns true; otherwise, it returns false.

```
m = 7
n = 12
result = m < n  # result will be True
```

- Greater than or equal to (>=): This operator checks if the left operand is greater than or equal to the right operand. If it is, it returns true; otherwise, it returns false.

```
alpha = 10
beta = 10
```

BOOLEAN EXPRESSIONS AND RELATIONAL OPERATORS

- result = alpha >= beta # result will be True
- Less than or equal to (<=): This operator checks if the left operand is less than or equal to the right operand. If it is, it returns true; otherwise, it returns false.

```
c = 8
d = 9
result = c <= d  # result will be True
```

BOOLEAN EXPRESSIONS AND RELATIONAL OPERATORS

```
1. age = 25
2. is_student = True
3. if age >= 18 and is_student:
4.     print("You are an adult student.")
5. else:
6.     print("You are either not an adult
  or not a student.")
```

BOOLEAN EXPRESSIONS AND RELATIONAL OPERATORS

- In this example, the >= operator is used to check if age is greater than or equal to 18, and the and operator is used to ensure that both conditions (age and being a student) are True for the "You are an adult student" message to be printed.

DECISION MAKING STATEMENTS

- Decision-making statements, also known as conditional statements, are fundamental constructs in programming that allow you to control the flow of your program based on conditions or criteria. These statements enable your program to make decisions and execute different blocks of code depending on whether certain conditions are met. Common decision-making statements include:
- **if Statement:**
- print("y is greater than 6")
- else:
- print("y is not greater than 5")

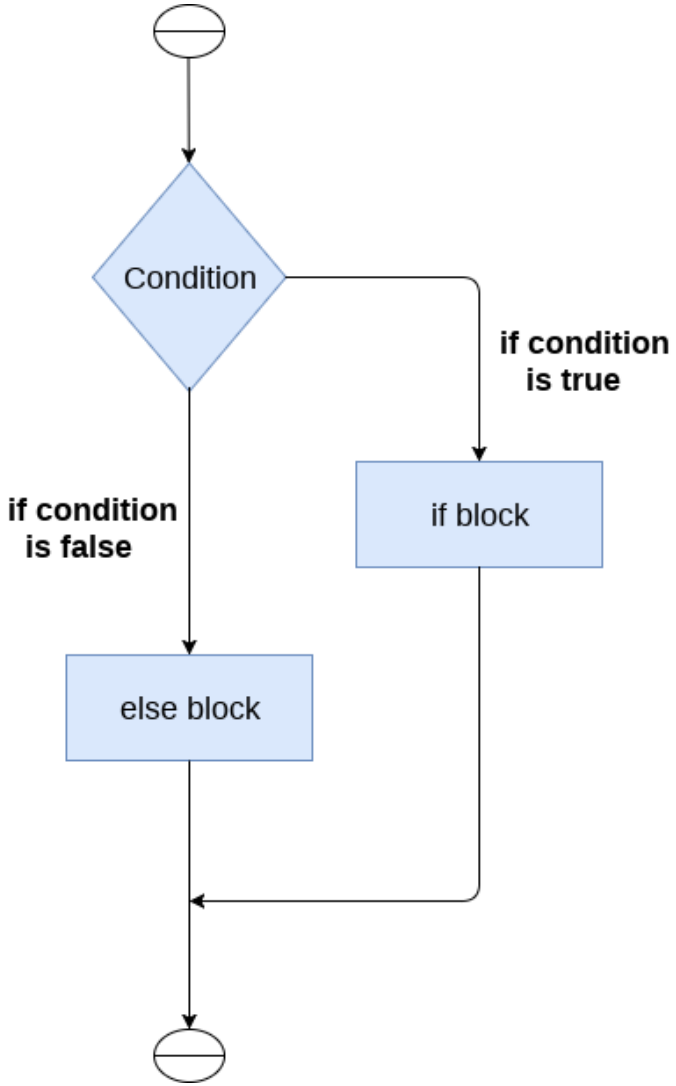
DECISION MAKING STATEMENTS+

- The if statement is used to execute a block of code if a specified condition is True. It can be followed by an optional else statement to specify a block of code to execute if the condition is False.

```
y = 4
if x > 5:
```

- **elif (else if) Statement:**
- The elif statement is used in combination with if to test multiple conditions sequentially. When the first if or elif condition is True, the corresponding block of code is executed, and the rest of the conditions are skipped.

If else statements



Fig, 1 Visual representation of for each [3]

DECISION MAKING STATEMENTS

```
1. score = 85
2. if score >= 90:
3.     print("A grade")
4. elif score >= 80:
5.     print("B grade")
6. elif score >= 70:
7.     print("C grade")
8. else:
9.     print("F grade")
```

Nested if Statements:

- You can nest if, elif, and else statements within each other to create more complex decision-making structures. This is useful when you need to check multiple conditions, and the behavior depends on combinations of these conditions.

```
1.    age = 25
2.    is_student = True
3.    if age >= 18:
4.        if is_student:
5.            print("You are an adult student.")
6.        else:
7.            print("You  are  an  adult  but  not  a
      student.")
8.    else:
9.        print("You are not an adult.")
```

Ternary Operator (Conditional Expression):

- Some programming languages, like Python, allow you to use a ternary operator to write compact conditional expressions. It's often used for simple decisions and can replace a single if-else statement.

```
1.  x = 10
2.  message = "x is greater than 5" if x > 5 else "x is not greater than 5"
3.  print(message)
```

Switch Statement (in some languages):

While not available in all programming languages, some languages provide a switch statement that allows you to evaluate a variable against multiple possible values and execute code based on the matching case.

Python doesn't have a built-in switch statement, but it can be simulated using if-elif-else. [2]

```
day=int(input("Enter the day of week:"))
if day==1:
      print(" Its Monday")
elif day==2:
      print("Its Tuesday")
elif day==3:
      print("Its
Wednesday")
elif day==4:
      print("Its Thursday")
```

```
elif day==5:
      print("Its Friday")
elif day==6:
      print("Its Saturday")
elif day==7:
      print(" Its Sunday")
else:
      print("Sorry!!! Week contains only 7
day")
```

Switch Statement (in some languages):

➢ These decision-making statements are crucial for controlling the flow of your program, enabling it to respond to different situations, user input, and changing data conditions. The choice of which statement to use depends on the complexity of your decision-making logic.

User sign up(authentication) and login program

- Let us close this week's lecture by implementing above project
- First the Algorithm

1. Prompt the user to input their username.
2. Prompt the user to input their email address.
3. Check if the email address contains the '@' and '.' symbols.
   - If it does not contain '@' and '.', display an error message "Invalid email. Please enter a valid one" and exit.
   - If it contains '@' and '.', proceed to the next step.
4. Prompt the user to input their password.

User sign up(authentication) and login program

- If both conditions are met, display a success message "Congratulations! You have signed up successfully."
- If either condition is not met, display an error message "Password is less than 8 characters or password mismatch" and exit.

5. After signing up successfully, prompt the user to enter their username and password (login) to log in.
6. Check if the entered login username and password match the ones entered during sign-up (username and password).
   - If they match, display a success message "You have successfully logged in."
   - If they do not match, display an error message "Login Failed!"
7. End the program.

Code implementation User sign up(authentication) and login program

```
1.  username=input("Please enter your
    username")
2.  email=input("Enter email")
3.  if '@' and '.' in email:
4.      password=input("Enter your
    password")
5.      contains_num=any(char.isdigit() for
    char in password)
6.      if contains_num:
7.          cpassword=input("Confirm
    password")
8.          if len(password)>=8 and
    password == cpassword and contains_num:
9.              print("Congratulation! You
    have sign up successfully")
10.             uname=input("Please enter
    your username\n")
```

```
13. login=input("Please your password\n")
14.             if login == password and
    uname==username:
15.                 print("You have successfuly
    login")
16.             else:
17.                 print("Login Failed!")
18.         else:
19.             print("Password is less than 8
    characters or password mismatch does not contain
    a")
20.
21.     else:
22.         print("Weak password")
23. else:
24.     print("Invalid email. Please enter a valid
    one")
```

```
username=input("Please enter your username")
email=input("Enter email")
if '@' and '.' in email:
    password=input("Enter your password")
    contains_num=any(char.isdigit() for char in password)
    if contains_num:
        cpassword=input("Confirm password")
        if len(password)>=8 and password == cpassword and contains_num:
            print("Congratulation! You have sign up successfully")
            uname=input("Please enter your username\n")
            login=input("Please your password\n")
            if login == password and uname==username:
                print("You have successfuly login")
            else:
                print("Login Failed!")
        else:
            print("Password is less than 8 characters or password mismatch does not contain a")

    else:
        print("Weak password")
else:
    print("Invalid email. Please enter a valid one")
```

Summary

- In this chapter, we discussed control statements, which are programming language constructs that allow us to control the flow of execution of a program. We covered all the major types of control statements, including:
- Conditional statements: These statements allow us to execute different code blocks based on a condition. The most common conditional statements are if, else if, and else.
- Iteration statements: These statements allow us to execute a block of code repeatedly until a certain condition is met. The most common iteration statements are while and for.

Summary

- Jump statements: These statements allow us to jump to a different point in the program's execution. The most common jump statements are break, continue, and goto.
- We also discussed Boolean types, which are data types that can have two values: true or false. Boolean expressions are used to evaluate conditions in control statements.
- To help us understand these concepts, we provided many code examples. We also discussed how control statements and Boolean types can be used to implement common programming patterns, such as input validation, error handling, and data processing.

Reference

[1] Introduction to computing and problem-solving using python, E Balagurusamy, McGraw Hill Education (India) Private Limited, 2016 page 61

[2] Introduction Programming and problem solving with python, Ashok N. Kamthane, Amit A. Kamthane, McGraw Hill Education (India) Private Limited,2018, Page 102

[3]*Python if else - javatpoint*. (n.d.). www.javatpoint.com. https://www.javatpoint.com/python-if-else