Section-1: Decision Structures, Boolean Logic, and Control Statements: if, if-else, if-elif-else statements, Nested Decision Structures, Comparing Strings, Logical Operators, Boolean Variables. P1-20
Section-2: Repetition Structures: Introduction, while loop, for loop, Calculating a Running Total, Input Validation Loops, Nested Loops.
Section-3: Strings: Accessing characters and Substring in Strings, Data Encryption, Strings and Number Systems.

Decision Structures

A **decision structure** is a set of program statements that makes a decision and changes the flow of the program based on that decision. These are also called **Control Flow Statements**. The decisions are made based on **True** or **False** of a **Boolean Logic test.**

The **control flow statements** are classified as follows:

A. **Selection or Decision or Conditional Statements**
    a. **if, if-else, elif**
B. **Loop or Repetition or Iterative Statements**
    a. **for, while**
C. **Jump Statements**
    a. **break, continue, pass**

A. Selection or Decision or Conditional Statements

In decision statements, the conditional expressions are evaluated with an outcome of either True or False.

a. The selection statements are 3 types:

Type	Single Selection	Two-Way Selection	Multi-Way Selection
Command	**if** statement	**if - else** statement	Nested **if - else** statements **elif** Ladder statements

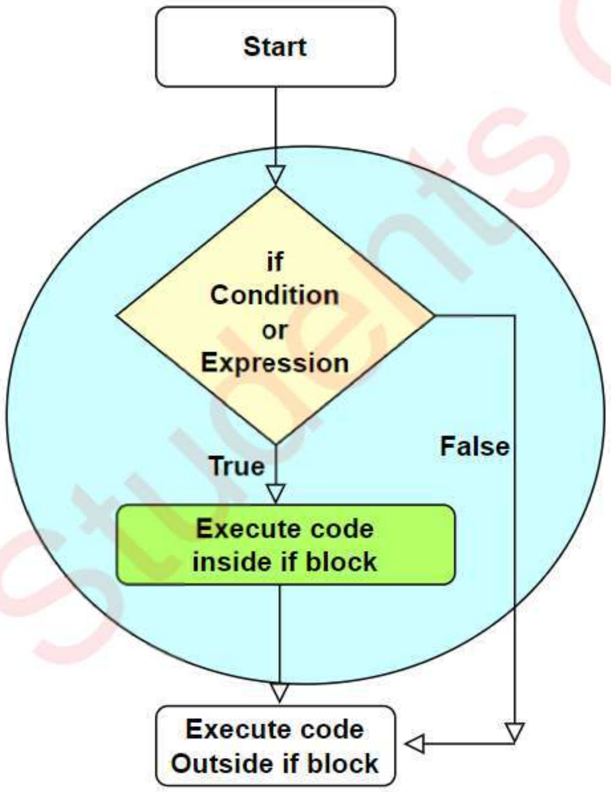
Single Selection in Python ("if" statement)

- "if" is a simple selection statement in Python. It is used to modify the flow of execution of a program.
- "if" consists of a **condition (boolean expression), colon :, and a block of statements with the same indentation**,
    - When the condition is **True**, the **'if' block of statements** will be executed,
    - Otherwise, the first statement outside the 'if' block will be executed.
- All the statements inside the 'if' block must have the same indentation of spaces

Syntax:

```
if condition:
    True block of statements
    statement 1
    statement 2
    ...
    statement n

Statements outside if block
```


Start
if Condition or Expression
True
False
Execute code inside if block
Execute code Outside if block

Application:

```
#if statement example
m, n = 77, 87
if(m < n):
    result = "m is smaller than n"
    print(result)
```

Output:

```
m is smaller than n
```

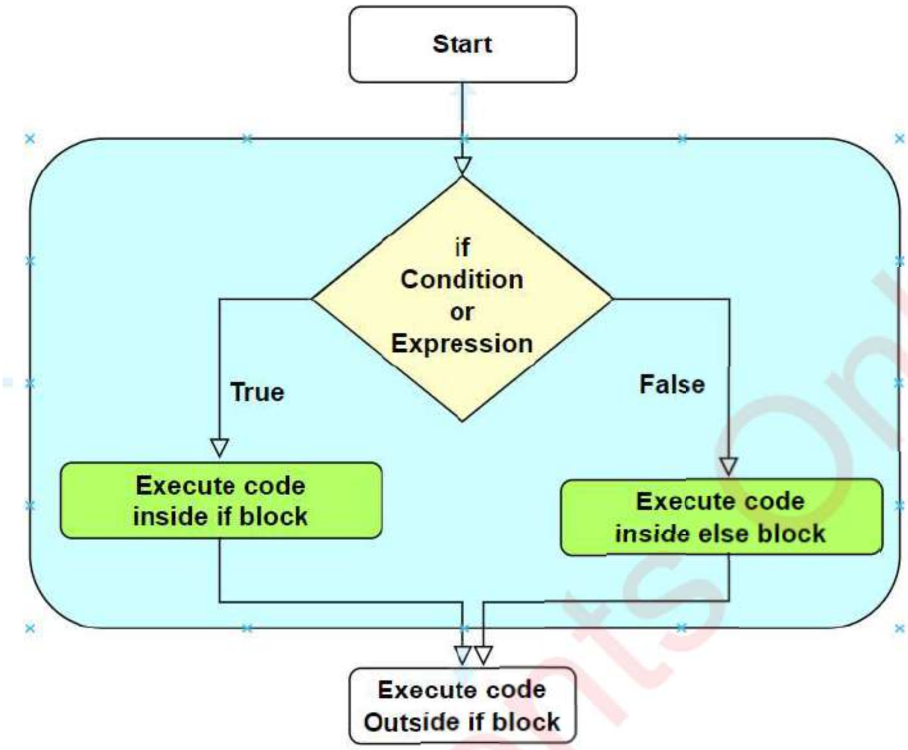
Two-Way Selection in Python ("if-else" statement)

- An **if** statement can also be followed by an optional **else** statement. **if-else** is a two-way decision statement which means, we have only two alternative choices.
- **if** statement will have a Boolean_Expression; **else** statement has NO Boolean_Expression.
- **if-else** consists of a condition (Boolean_Expression), a block of statements for '**if**', and another block of statements for '**else**'.
    - When the Boolean_Expression is **True**, the '**if**' block of statements will be executed
    - When the Boolean_Expression is **False**, the '**else**' block of statements will be executed
- **Indentation:**
    - All the statements inside the '**if**' block **must have the same indentation** with spaces
    - All the statements inside the '**else'** block **must have the same indentation** with spaces
    - **However, a different indentation can be used for 'if' block and 'else' block.**

Syntax:

```
if condition:
    True block of statements
    statement 1
    statement 2
    ...
    statement n
else:
    False block of statements
    statement 1
    statement 2
    ...
    statement n

Statements outside if-else block
```


Start
if Condition or Expression
True
False
Execute code inside if block
Execute code inside else block
Execute code Outside if block

Application-1:

```
'''
if-else construct using Membership operator 'in' and 'set' of names
Strings are case sensitive in Python;
Upper case strings are different from Lower case strings
'''
cse = {"Saida", "Ajay", "Sai", "Veda"}
sname = input("Enter a name to search : ")
if sname in cse:
    print("Yes, {} is in CSE branch!".format(sname))
else:
    print("No, {} is Not in CSE branch!".format(sname))
```

Output:

```
Enter a name to search : Veda
Yes, Veda is in CSE branch!

Enter a name to search : veda
No, veda is Not in CSE branch!
```

Application-2:

```
'''
Aim: Program to Check whether the given number is Even or Odd.
'''
num = int(input("Enter an integer : "));
# true if num is perfectly divisible by 2
if(num % 2 == 0):
    print("{} is even.".format(num))
else:
  print("{} is odd.".format(num))
# Notice that if block has 4 space indentation and else block has 2 space indentation
```

Output:

```
Enter an integer : 7
7 is odd.
Enter an integer : 4
4 is even.
```

Application-3:

```
'''
lab-7: Write a program that asks the user for two numbers and prints
Close if the numbers are within .001 of each other and not close
otherwise.
'''
a = float(input("Enter first number : "))
b = float(input("Enter second number : "))
c = abs(a - b)
if c > 0.0009 and c <= 0.001 :
        print("Close")
else :
    print("Not Close")
```

Output:

```
Enter first number : 4.001
Enter second number : 4.002
Close
```

Multi-Way Selection - Nested “if-else”

When a series of decisions is required, the multi-way selection statements are used. There are 2 types of multi-way selection statements

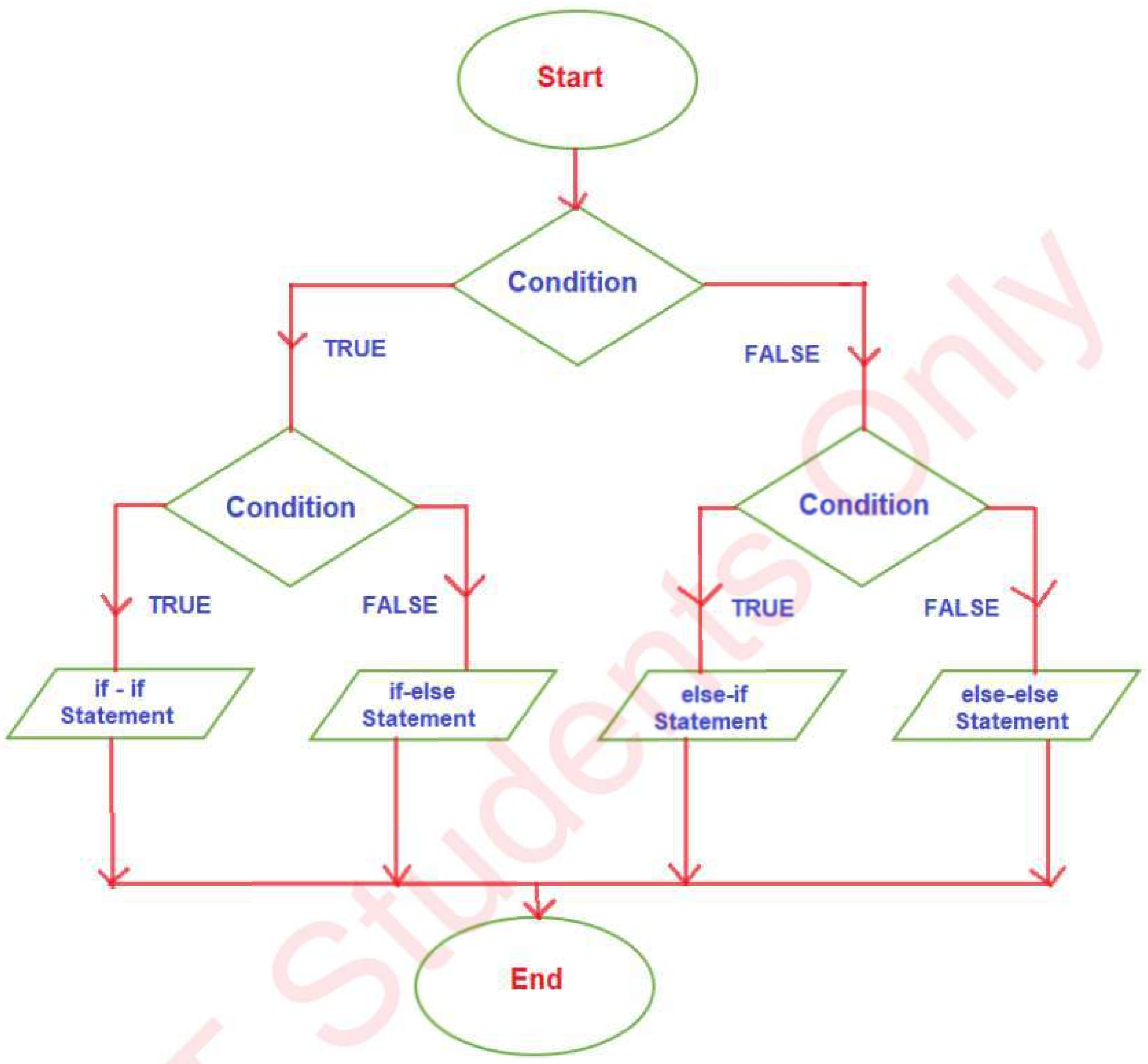
1. **Nested " if-else " statements**
2. **" elif " Ladder statements**

1. Nested if-else statements

Nesting means using one "**if-else**" construct within another "**if-else**" construct. Use nested "if-else" when you need to decide more within the parent "if" condition or parent "else" condition. **The nested "if-else" is used when multiple paths of decisions are required.**

Syntax:

```
if(Condition/Expression):          # Outer if block
    if(Condition/Expression):      # Inner if block
        Statements
    else:                          # Inner else block
        Statements
else:                              # Outer else block
    if(Condition/Expression):      # Inner if block
        Statements
    else:                          # Inner else block
        Statements
```



Application-1:

```
#Checks whether input marks are pass or fail
# JustPass=40, pass>40, fail<40
marks = int(input("Enter marks 0-100 : "))
if marks >= 40:
    if marks==40:
        print("Just Passed with ", marks)
    else:
        print("Passed with ", marks);
else:
    print("Failed with ", marks)
```

Output:

```
Enter marks 0-100 : 75
Passed with  75

Enter marks 0-100 : 40
Just Passed with  40

Enter marks 0-100 : 30
Failed with  30
```

Application-2: Write a program to find whether the given year is a Leap Year?

```python
'''
Aim: Find whether the given year is a Leap Year.
Note: A Century year ends with 00 or divisible by 100
Conditions:
Non-Centuary years that are exactly divisible by 4 are leap years.
A century year divisible by 4, 100, and 400 is a leap year.
A century year divisible by 4, 100, but not divisible by 400 is not a
leap year.

For example,
1900 is not a leap year (Century year, Divisible by 4 & 100; but not
by 400)
1999 is not a leap year (Non-Centuary, Not divisible by 4)
2000 is the leap year   (Century year, Divisible by 4, 100, and 400)
2004 is the leap year   (Non-Centuary, Divisible by 4)
2024 is the leap year   (Non-Centuary, Divisible by 4)
'''
year=int(input('Enter a year:'))
if year%4==0:
    if year%100==0:      # Centuary year
        if year%400==0:  # Century year divisible by 400
            print(f'{year} is a leap year')
        else:            # Century year Not divisible by 400
            print(f'{year} is not leap year')
    else:                # Non-Centuary year, Divisible by 4
        print(f'{year} is a leap year')
else:                    # not divisible by 4
        print(f'{year} is not a leap year')
```

```
Output:
Enter a year:2022
2022 is not a leap year

Enter a year:2024
2024 is a leap year

Enter a year:3000
3000 is not leap year
```

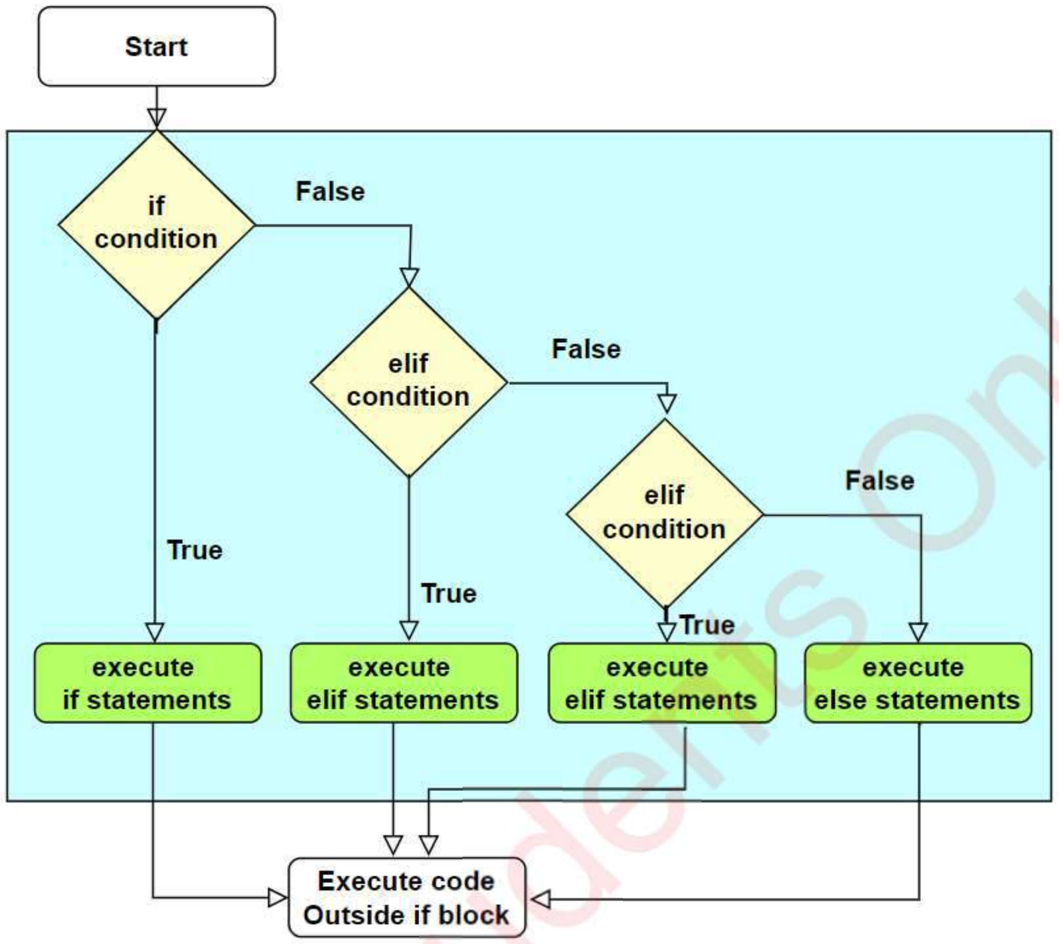
2. "elif" Ladder statements

In Python, "elif" keyword is a short form of "else if". The "elif" is useful **when you need to decide a series of decisions** after each of the previous "if" conditions.

- The series of conditions are evaluated from **top to bottom**.
- When one condition becomes **true**, the statements of that condition will be executed and the control comes out of the whole "if" block.
- When all the conditions are **false**, then the last default **"else"** statement is executed and the control comes out of the whole "if" block.

Syntax:

```
if boolean_expression1:
    statement(s)
elif boolean_expression2:
    statement(s)
elif boolean_expression3:
    statement(s)
else:
    statement(s)
```



Application-1:

```python
# elif conditional statements
x = 20
y = 70
if x > y:
    print("x is greater than y")
elif y > x:
    print ("y is greater than x")
else:
    print("x and y are equal")
```

Output:

```
y is greater than x
```

Application-2: Grades

Write a Program to Prompt for Marks between 0 and 100. If the Marks Is Out of Range, Print an Error. If the Marks are between 0 and 100, Print a Grade Using the Following Table.

Score	>=90	>=80	>=70	>=50	>=40	<40
Grade	A+	A	B	C	D	F

```
# elif conditional statements
marks=int(input("Enter marks 0-100 : "))
if(marks,0 or marks>100):
    print("Marks out of range.")
elif(marks>=90):
    print("Grade A+")
elif(marks>=80):
    print("Grade A")
elif(marks>=70):
    print("Grade B")
elif(marks>=50):
    print("Grade C")
elif(marks>=40):
    print("Grade D")
else:
    print("Grade F")
```

Output:

```
Enter marks 0-100 : 70
Grade B
```

Application-3:

```
'''
Lab-16. Write a program that asks the user to enter a length in feet.
The program should then give the user the option to convert from feet
into inches, yards, miles, millimeters, centimeters, meters, or
kilometers. Say if the user enters a 1, then the program converts to
inches, if they enter a 2, then the program converts to yards, etc.
While this can be done with if statements, it is much shorter with
lists and it is also easier to add new conversions if you use lists.
'''
```

```
feet=int(input("Input distance in feet: "))
print("Choose your option: ")
print("1. inches")
print("2. yards")
print("3. miles")
print("4. millimeters")
print("5. centimeters")
print("6. meters")
print("7. kilometers")
option=int(input("Enter the option : "))
if option==1:
    dist=round(feet*12,2)
    units="inches"
    print("The distance in {} is {} inches.".format(units,dist))
elif option==2:
    dist=feet/3
    units="yards"
    print("The distance in %s is %.2f yards."%(units,dist))
elif option==3:
    #dist=round(feet*0.000189394,3)
    dist=feet/5280
    units="miles"
    print("The distance in %s is %.2f miles."%(units,dist))
elif option==4:
    dist=feet*304.8
    units="millimeters"
    print("The distance in %s is %.2f millimeters."%(units,dist))
elif option==5:
    dist=feet*30.8
    units="centimeters"
    print("The distance in %s is %.2f centimeters."%(units,dist))
elif option==6:
    #dist=round(feet*0.3048,3)
    dist=feet*0.3048
    units="meters"
    print(f"The distance in %s is %.2f meters."%(units,dist))
elif option==7:
    dist=feet/3280.8
    units="kilometers"
    print("The distance in %s is %.2f kilometers."%(units,dist))
else:
    print("Invalid choice!!!")
```

Output:

```
Input distance in feet: 456
Choose your option:
1. inches
2. yards
3. miles
4. millimeters
5. centimeters
6. meters
7. kilometers
Enter the option : 7
The distance in kilometers is 0.14 kilometers.
```

Comparing Strings

In Python, string comparison is the process of comparing two strings to determine whether they are equal or not.

Strings in Python are stored as objects with an ID (memory address).
Python reuses the same memory for two equal strings to save memory, and run faster & easier.

Reference-only

Objects in Python consist of 3 properties:

1. **Identity** - address of the memory where the string is stored
2. **Type** - data type of the string ‘str’
3. **Value** - content stored in the object

Commonly used string comparison methods in Python are,

A. using Built-in **Operators**
B. using Built-in **Functions**

A. Using Built-in Operators for String Comparison:

- Equality/Inequality Operators (**==, !=**) compare similarity
- Identity Operators (**is, is not**) compare address (use **id()** function to find the address of an object)
- Comparision/Relational Operators (**<, <=, >, >=**) compare alphabetical order

Equality/Inequality Operators (==, !=):

The **"=="** and **"!="** operators checks if two strings are equal or not. The strings are case-sensitive. So, upper-case strings are different from lower-case strings.

Syntax: string1 == string2 (returns True if 2 strings are Equal)
Syntax: string1 != string2 (returns True if 2 strings are Not Equal)

Identity Operators (is, is not):

The "**is**" and **'is not'** operators compare the address of two strings and find they are of the same object or different object.
Python considers equal strings as the same object and stores them in the same memory location. So, the '**is**' and '**is not**' operators compare their address locations.

Syntax: string1 is string2
Syntax: string1 is not string2

ASCII	83	**111**	102	116	119	97	114	101
String1	**S**	**o**	**f**	**t**	**w**	**a**	**r**	**e**
String2	**S**	**O**	**F**	**T**	**W**	**A**	**R**	**E**
ASCII	83	**79**	70	84	87	65	82	69

ASCII Values - A-Z : 97-122, a-z : 65-90, 0-9 : 48-57

Application:

```
# Equality/In-Equality operators: ==, !=
# Identity Operators: is, is not   # compare memory address
s1 = "Software"
s2 = "Software"
s3 = "SOFTWARE"
print(s1 == s2)      # True
print(s1 == s3)      # False
print(s1 != s3)      # True

print(s1 is s2)      # True
print(s1 is s3)      # False
print(s1 is not s3) # True
# Notice the address of s1 and s2 are same
print("Address of s1 : ", id(s1))
print("Address of s2 : ", id(s2))
print("Address of s3 : ", id(s3))
```

Output:

```
True
False
True

True
False
True

Address of s1 :  2295811751536
Address of s2 :  2295811751536
Address of s3 :  2295811751344
```

Comparision Operators (<, <=, >, >=) :

The comparison operators check two strings lexicographically, that is based on their alphabetical order. The alphabetical order is determined by comparing the ASCII values of the characters in the strings.

Syntax: string1 > string2
Syntax: string1 < string2

ASCII	67	83	69
String1	C	S	E
String2	A	I	
ASCII	65	73	

Application:

```python
dept1 = "CSE"
dept2 = "AI"
if dept1 < dept2:
    print(f"{dept1} comes before {dept2}")
else:
    print(f"{dept2} comes before {dept1}")
```

Output:

```
AI comes before CSE
```

B. Using Built-in Functions for String Comparison:

starstwith() and **endswith()** functions return True or False depending on whether the given substring is found at the beginning, end, or anywhere in the string.
find() function will return the position number (index) of the searched substring in the main string. It returns -1 if the searched string is not found.
count() function will return a number of times the given string has occurred in the main string.

Function	Definition & Syntax	Example s1="Hi CIT Engineers"
startswith()	Returns True if a string1 starts with a prefix (substring / tuple - True if any one tuple member matches) `string.startswith(prefix, start, end)`	**s1.startswith("Hi")** t1=("Hi", "Hello") s1.startwith(t1)
endswith()	Returns True if a string2 ends with a suffix (substring / tuple - True if any one tuple member matches) `string.endswith(suffix,start, end)`	**s1.endswith("eers")** t2=("fine", "Engineers") s1.endswith(t2)
find()	Returns position# (index#) of substring in string1; Returns -1 if not found. `string.find("substring", start, end)` (optional) start=0, end=last-index	**s1.find("CIT")** Output: 3
count()	Returns no.of times given value occurs in a string `string.count("substring", start, end)` (optional) start=0, end=last-index	**s1.count("i")** (Default & optional, start=0, end=last)

Application:

```
#String built-in functions startswith(), endswith(), find(), count()
s1 = "Hi CIT Engineers"
if s1.startswith("Hi"):
    print("The string starts with 'Hi'")
tpl=("Hi","Hello")
result = s1.startswith(tpl):
    print("Start word in tuple?",result)   # True
if s1.endswith("eers"):
    print("The string ends with 'Engineers'")
if s1.find("CIT") != -1:
    print("The string contains 'CIT'")
n = s1.count("i")
    print("Number of i letters in the string: ",n)
```

Output:

```
The string starts with 'Hi'
Start word in tuple? True
The string ends with 'Engineers'
The string contains 'CIT'
Number of i letters in the string: 2
```

Logical Operators (and, or, not)

The logical operators are the keywords that **combine multiple conditions into a single condition**. The following table provides information about logical operators.

Operator	Meaning	Example
and	Returns True if all conditions are True otherwise returns False	10 < 5 and 12 > 10 is False
or	Returns False if all conditions are False otherwise returns True	10 < 5 or 12 > 10 is True
not	Returns True if condition is False and returns False if the condition is True	not(10 < 5 and 12 > 10) is True

- **Logical and -** Returns True only if ALL conditions are True, if any one of those conditions is False then whole condition becomes False.
- **Logical or -** Returns True if ANY condition is True, if all conditions are False then the whole condition becomes False.

Application-1: Basic Logical Operators

```
#Logical Opertaors
a = True
b = False
print(a and b) #output: False
print(a or b) #output: True
print(not a) #output: False
a=10
b=5
la = (a<b) and (b<c)   # Combined two conditions
lo = (a<b) or (b<c)    # Combined two conditions
ln = not(a<b)
print("Logical AND = ",la)  #False
print("Logical OR  = ",lo)  #True
print("Logical NOT = ",ln)  #True
```

Application-2: if-else using Logical Operators

```
# Find smallest of three numbers using elif statement
a=10
b=5
c=12
if((a<b) and (a<c)):
    print("a is smallest")
elif ((b<a) and (b<c)):
    print("b is smallest")
else:
    print("c is smallest")
```

Output:

```
b is smallest
```

Boolean Variables

A boolean variable can have only two values: **True** or **False**
The variables with the boolean values **True** or **False** are called Boolean type variables.
These boolean values are case sensitive; hence, the T and F of True and False must be capital letters.

Syntax:

Variable = Boolean value
Variable = Boolean expression

We can define a boolean variable by simply assigning a True or False value or even an expression that gets evaluated to one of these values.

Application:

```
# Boolean variables & assignment
a = False   # assigned boolean value False
b = True    # assigned boolean value True
print(type(a))
print(type(b))
c = (5>2)   # assigned boolean expression
print("c value : ", c)
print("c data type : ", type(c))
```

Output:

```
<class 'bool'>
<class 'bool'>
c value :  True
c data type :  <class 'bool'>
```

bool() built-in function:

bool() method evaluates any **value or a variable or any expression** and returns a Boolean value either True or False.

bool() method

- returns True for one argument of any value or expression; and
- returns False for `0, None, False, empty values "", (), {}, []`

Syntax:

```
bool()              # False
bool(value)         # True
bool(variable)      # True
bool(expression)    # True
```

Application:

```
# bool() function will always return True for any value
# except 0, None, False, empty values such as "", (), {}, [].
print("Returns True for any value")
print(bool("CSE AI ML"))
print(bool('''Guntur'''))
print(bool(75))
print(bool([10, 20, 40]))
print(bool(-11))
print(bool(3.14))
print(bool(25>(50/3)))

print("Returns False for 0, None, False, empty values \"\", (), {}, []")
print(bool())
print(bool(0))
print(bool(None))
print(bool(False))
print(bool([]))
print(bool(""))
print(bool({}))
print(bool(()))
```

Output:

```
Returns True for any value
True
True
True
True
True
True
True
Returns False for 0, None, False, empty values "", (), {}, []
False
False
False
False
False
False
False
False
```