

Aim and Objectives

The aim of this lab is to provide students with a solid understanding of control flow in Python programming, enabling them to effectively design, implement, and interpret decision-making processes within their code. By mastering control flow, students will develop the skills necessary to create robust, logical programs that can make decisions based on different conditions.

After successful completion of this lab, students should be able to:

- Demonstrate understanding of the fundamental concepts of control flow in Python programming.
- Create and interpret flowcharts representing control flow structures.
- Construct boolean expressions and logical operators to construct conditions.
- Implement decision-making structures using **if-else**, **if-elif-else**, and nested **if-else** conditionals.
- Apply control flow knowledge to solve practical coding problems.


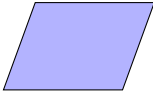
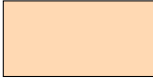
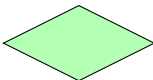


1 Representing algorithms using flowcharts

An algorithm is a sequence of logical steps to solve a problem. **Flowcharts** use graphical symbols to represent different parts of an algorithm.

It consists of a set of blocks in which each block is connected using flow lines representing the flow of the algorithm. When an algorithm is represented as a flowchart, a non-programmer will be able to understand it easily.

Table 1 shows the most commonly used flowchart symbols and their functionality.

Table 1: Flowchart Symbols

Symbol	Name	Function
	Start/End	Represents the beginning or end of a program
	Input/Output	Inputs or outputs of data/information
	Process	Represents calculations or data manipulations
	Decision	Represents a comparison, question or decision
	Link	Represents the flow between flowchart elements
	Connector	Represents a break in the path and continuation from another place

There are three fundamental control structures that are used in complex problems. They are, **sequence**, **selection**, and **repetition**.

1.1 Sequence

The sequence control structure represents the straightforward execution of statements in a linear order. Each statement is executed one after another, from top to bottom as shown in Figure 1

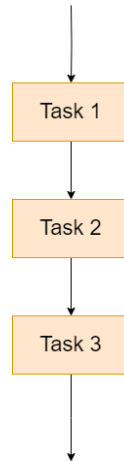


Figure 1: Sequence flow control

1.2 Selection

The selection control structure allows the program to choose between different paths based on a condition. It typically involves making decisions and executing different blocks of code depending on whether a condition is true or false as shown in Figures 2 and 3.

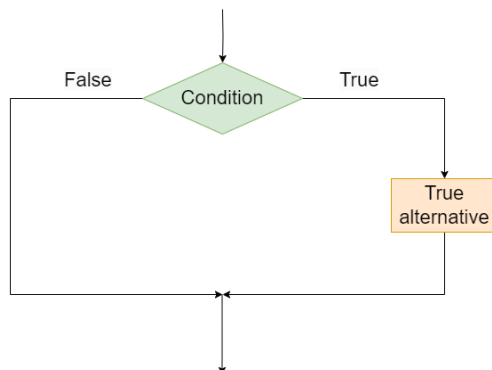


Figure 2: Selection (True alternative only)

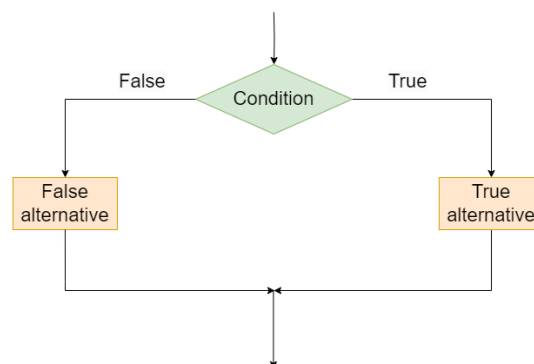


Figure 3: Selection (Both alternatives)

1.3 Repetition

The repetition control structure allows a block of code to be executed repeatedly based on a condition. This can be implemented through loops as shown in Figure 4

2 Boolean expressions

1. A **boolean expression** is an expression that is either **true** or **false**. One way to write a boolean expression is to use the operator `==`, which compares two values and produces a boolean value:

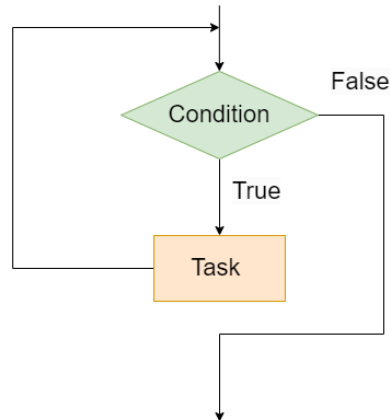


Figure 4: Repetition flow control

```

1 >>> 5 == 5
2 True
3 >>> 5 == 6
4 False

```

True and **False** are special values that are built into Python.

- The `==` operator is one of the **comparison operators**; the others are:
`!=` (not equal to):

```

1 >>> 5 != 6
2 True

```

`<` (less than) and `>` (greater than):

```

1 >>> 3 < 5
2 True
3 >>> 5 > 3
4 True

```

`<=` (less than or equal to) and `>=` (greater than or equal to):

```

1 >>> 3 <= 5
2 True
3 >>> 5 >= 5
4 True

```

Pause and Reflect:

- What is the difference between a single equal sign (`=`) and double equal sign (`==`)?
- Are `=<` and `=>` valid in Python?

3 Logical operators

- There are three logical operators: **and**, **or**, **not**

The meaning of these operators is similar to their meaning in English:

```

1 >>> 5 > 3 and 8 > 6 # Both conditions must be true
2 True
3 >>> 5 > 3 and 8 < 6 # One is false
4 False
5 >>> 5 > 3 or 8 < 6 # Atleast one condition must be true
6 True
7 >>> not 5 > 3 # Inverts the boolean value
8 False

```

2. In Python, any nonzero value is interpreted as **True**:

```
1 >>> 5 and 1
2 1
```

When you evaluate **5 and 1** in Python, it doesn't directly return True or False. Instead, the **and** operator returns the first value if it is False-like (zero or an empty value). Otherwise, it returns the second value. This property of Python is known as 'short-circuiting'.

```
1 >>> 0 or 5
2 5
```

Similarly, the **or** operator returns the first value if it is True-like (non-zero). If the first value is False-like, it returns the second value.

Pause and Reflect: What are the outputs of the following expressions?

```
1 >>> 1 and 5
2 >>> 0 or -3
3 >>> 27 and 'hello'
4 >>> 0 or []
5 >>> not 'hello'
6 >>> not ''
```

4 Conditional execution

1. In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly. **Conditional** statements give us this ability. The simplest form is the **if** statement:

```
1 # Simple IF statement
2 # CO1010 Lab 02
3 x = 12
4 if x > 0:
5     print("x is positive")
```

The boolean expression after the **if** statement is called the **condition**. If it is true, then the indented statement gets executed. If not, nothing happens.

2. A second form of the **if** statement is alternative execution, in which there are two possibilities and the condition determines which one gets executed. The syntax looks like this:

```
1 # Simple IF-ELSE statement
2 # CO1010 Lab 02
3 y = 13
4 if y%2 == 0:
5     print(y, " is even")
6 else:
7     print(y, " is odd")
```

Since the condition must be true or false, exactly one of the alternatives will be executed. The alternatives are called **branches**, because they are branches in the flow of execution.

3. Sometimes there are more than two possibilities and we need more than two branches. One way to express a computation like that is a **chained conditional** and includes the use of **elif** keyword. **elif** is an abbreviation of **else if**. Again, exactly one branch will be executed. There is no limit of the number of **elif** statements, but the last branch has to be an **else** statement:

```
1 # Simple IF-ELSE statement with chained conditions
2 # CO1010 Lab 02
3 x = 10
4 y = -1
5 if x < y:
6     print(x, " is less than ", y)
```

```

7 elif x > y:
8     print(x, " is greater than ",y)
9 else:
10    print(x, " and ",y, " are equal")

```

Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes.

4. One conditional can also be nested within another:

```

1 # Simple IF-ELSE statement with nested conditions
2 # CO1010 Lab 02
3 if x == y:
4     print(x, " and ",y, " are equal")
5 else:
6     if x < y:
7         print(x, " is less than ",y)
8     else:
9         print(x, " is greater than ",y)

```

Lab Tasks

Complete the exercises below, show them to an instructor and get them marked within the lab hours.

*While peer-learning is highly encouraged in our labs, **copying someone else's codes will earn you zero marks for all lab exercises.***

- First draw a flowchart and then write a Python script named Lab02A.py (use **if-else**) to do the following task: Get a temperature value in degrees Celsius (T_c) from the user. Then the user can select whether to convert the temperature value to Fahrenheit (T_f) by entering **F** or Kelvin (T_k) by entering **K**. You should output the corresponding temperature value requested by the user. You may use the following equations for the conversion.

$$T_f = \frac{9}{5}T_c + 32 \quad (1)$$

$$T_k = T_c + 273.15 \quad (2)$$

Your script should behave like the following:

```

1 Enter temperature in Celsius: 0
2 Do you want to convert K or F?: F
3 The temperature value in Fahrenheit is 32.00.

```

- First draw a flowchart and then a Python script named Lab02B.py (use **if-elif-else**) to perform the following task: The Reynolds number (Re) is used in fluid dynamics to determine the nature of a fluid flow. For an internal flow (e.g. water flow through a pipe), the flow can be categorized as given in the following table. The flow region changes with the Reynolds number.

$Re \leq 2300$	Laminar Region
$2300 < Re \leq 4000$	Transition Region
$Re > 4000$	Turbulent Region

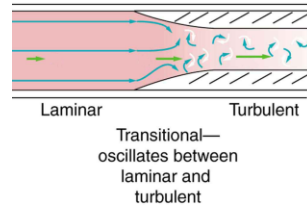
Your script should behave like the following:

```

1 Enter the Reynolds number (Re): 4000
2 The fluid is in: Transition Region

```

- Write a Python script named Lab02C.py (using **nested if-else**) to perform the following task:



- (a) Get a numeric grade (0-100) from the user.
- (b) The user can then select the grading scheme they want to use:
 - Enter **'SG'** for "Standard Grading" (A, B, C, D, F).
 - Enter **'PMG'** for "Plus-Minus Grading" (A+, A, A-, B+, B, B-, C+, C, C-, D+, D, D-, F).
- (c) Based on the user's input, the program should calculate and display the corresponding letter grade.

You may use the following rules for the conversion:

Grading Schemes

Standard Grading Scheme (A):

- 90-100: A
- 80-89: B
- 70-79: C
- 60-69: D
- 0-59: F

Plus-Minus Grading Scheme (B):

- 97-100: A+
- 93-96: A
- 90-92: A-
- 87-89: B+
- 83-86: B
- 80-82: B-
- 77-79: C+
- 73-76: C
- 70-72: C-
- 67-69: D+
- 63-66: D
- 60-62: D-
- 0-59: F

Submission

Please follow these steps for submission:

1. Complete the tasks and save your Python scripts with the following names:
 - Lab02A.py

- Lab02B.py
 - Lab02C.py
2. Compress all three scripts into a single ZIP folder.
 3. Rename the ZIP folder to E22xxx.ZIP, where xxx represents the last three digits of your student registration number.
 4. Submit the ZIP folder to the link provided on the FEeLS course page.

Steps for Creating a ZIP File

If you are unfamiliar with how to create a ZIP file, please follow the steps below:

1. Select the three Python scripts (Lab02A.py, Lab02B.py, Lab02C.py).
2. Right-click on one of the selected files.
3. Choose **Send to** → **Compressed (zipped) folder**.
4. A new ZIP folder will appear. Rename this folder to E22xxx.ZIP, replacing xxx with the last three digits of your registration number.