Loops in Python are used to repeatedly execute a block of code until a certain condition is met. Python provides two main types of loops: for and while.

Loops- write once and execute any number of times

Do while is not included in python- In Python, there is no explicit "do-while" loop construct as found in some other programming languages like C or Java. However, the same behavior can be achieved using a while loop with a conditional check at the end.

while True:

# Code to be executed

user\_input = input("Do you want to continue? (yes/no): ")

# Check the condition to continue or break the loop

if user\_input.lower() != 'yes':

break

In this example, the loop will continue executing the code block until the user enters something other than "yes" when prompted. The loop uses **break** to exit when the condition is not met.

**While Loop in Python**

In Python, a [while loop](https://www.geeksforgeeks.org/python-while-loop/) is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

**While Loop Syntax:**

while expression:  
 statement(s)

### ****Components of a While Loop:****

1. The first line starts with the while keyword, indicating this is a while loop.
2. Following that is a condition to be checked. In this example, that's sum(hand) <= 17.
3. The while loop heading always ends with a colon :.
4. Indented after this heading is the body of the while loop. If the condition for the while loop is true, the code lines in the loop's body will be executed.
5. We then go back to the while heading line, and the condition is evaluated again. This process of checking the condition and then executing the loop repeats until the condition becomes false.
6. When the condition becomes false, we move on to the line following the body of the loop, which will be unindented.

Infinite loop example

## ****For Loops:****

 A for loop is used to "iterate", or do something repeatedly, over an iterable.

An iterable is an object that can return one of its elements at a time.

For loop is of 2 types:

1. With range function – lower end, higher end, increment/decrement
2. Without range functions- list, dictionaries etc.,.

# Operators

* **Operators:** Arithmetic, Assignment, Comparison, Logical

An arithmetic operator is a mathematical function that takes 2 operands and performs a calculation on them.

1. **Addition (+):**
   * Combines two values.
2. **Subtraction (-):**
   * Finds the difference between two values.
3. **Multiplication (\*):**
   * Multiplies two values.
4. **Division (/):**
   * Divides the left value by the right value.
5. **Floor Division (//):**
   * Returns the quotient, discarding the remainder.
6. **Modulus (%):**
   * Returns the remainder of the division.
7. **Exponentiation (**):\*\*
   * Raises the left value to the power of the right value.

Now Arithmetic operators in python follow the **BEDMAS** order of operations.  
Look at this mathematical equation (2+3\*5) what do you think will be the answer 25 or 17. The answer is 17 because in maths we follow a convention called as the order of operations BEDMAS ie.

* Brackets
* Exponents
* Division and Multiplication (left to right)
* Addition and Substraction (left to right)

Preference given to the operator that comes first in BEDMAS.

**Assignment operators** in Python are used to assign values to variables. They combine the assignment (=) operation with other arithmetic or bitwise operations.

1. **Assignment (=):**
   * Assigns the value on the right to the variable on the left.
2. **Addition Assignment (+=):**
   * Adds the value on the right to the variable on the left and assigns the result to the left variable.
3. **Subtraction Assignment (-=):**
   * Subtracts the value on the right from the variable on the left and assigns the result to the left variable.
4. **Multiplication Assignment (\*=):**
   * Multiplies the variable on the left by the value on the right and assigns the result to the left variable.
5. **Division Assignment (/=):**
   * Divides the variable on the left by the value on the right and assigns the result to the left variable.
6. **Floor Division Assignment (//=):**
   * Performs floor division on the variable on the left by the value on the right and assigns the result to the left variable.
7. **Modulus Assignment (%=):**
   * Computes the modulus of the variable on the left with the value on the right and assigns the result to the left variable.
8. **Exponentiation Assignment (**=):\*\*
   * Raises the variable on the left to the power of the value on the right and assigns the result to the left variable.

These operators provide a concise way to perform arithmetic operations while updating the value of a variable.

**Comparison operators** in Python are used to compare two values, and the result is a Boolean value (**True** or **False**). Here are the main comparison operators along with brief explanations:

1. **Equal to (==):**
   * Returns **True** if the values on both sides are equal.
2. **Not equal to (!=):**
   * Returns **True** if the values on both sides are not equal.
3. **Greater than (>):**
   * Returns **True** if the value on the left is greater than the value on the right.
4. **Less than (<):**
   * Returns **True** if the value on the left is less than the value on the right.
5. **Greater than or equal to (>=):**
   * Returns **True** if the value on the left is greater than or equal to the value on the right.
6. **Less than or equal to (<=):**
   * Returns **True** if the value on the left is less than or equal to the value on the right.

These operators are commonly used in conditional statements and expressions to make decisions based on the comparison of values.

**Logical operators** in Python are used to perform logical operations on Boolean values. Here are the main logical operators along with brief explanations:

1. **Logical AND (and):**
   * Returns **True** if both the operands are true; otherwise, it returns **False**.
2. **Logical OR (or):**
   * Returns **True** if at least one of the operands is true; if both are false, it returns **False**.
3. **Logical NOT (not):**
   * Returns **True** if the operand is false; returns **False** if the operand is true.

Logical operators are commonly used in conditional statements and expressions to combine and evaluate multiple conditions. They are crucial for creating complex decision-making structures in programs.

**Decision making** in programming is the process of controlling the flow of a program by evaluating conditions and executing specific code blocks based on whether those conditions are true or false.

1. **if statement:**
   * The **if** statement is used for conditional execution of a block of code. If the specified condition is true, the indented code block under the **if** statement is executed; otherwise, it is skipped.
2. **if-else statement:**
   * The **if-else** statement extends the **if** statement by providing an alternative block of code to be executed when the condition specified in the **if** statement is false. Either the code under the **if** block or the code under the **else** block is executed, but not both.
3. **if-elif-else statement:**
   * The **if-elif-else** statement is an extension of the **if-else** statement. It allows you to check multiple conditions in sequence. If the first condition is true, the corresponding block of code is executed; otherwise, the next condition is checked. If none of the conditions is true, the code under the **else** block is executed.

A **function** is a block of which when called perform a specific task. In python programming or any programming language you use, a function is used:

 *When a specific block of code is required to be performed again and again.*

 *To make the code easier to understand and more user friendly.*

### IMPLEMENTING A FUNCTION

##### *For implementing a function we need to follow three steps:*

* *Define the function*
* *Write function body*
* *Return Statement*

python

def function\_name(parameters):

# Code block

return result

Components:

Function Name: Identifies the function and is used to call it.

Parameters: Inputs passed to the function (optional).

Code Block: Set of statements that define the function's behavior.

Return Statement: Specifies the value the function returns (optional).

**Types of Functions in Python:**

1. **Built-in Functions:**
   * Functions that are already defined in Python, such as **print()**, **len()**, **max()**, etc.

Python print("Hello, World!")

1. **User-Defined Functions:**
   * Functions created by the user to perform a specific task.

Python def greet(name): print(f"Hello, {name}!") greet("Alice")

1. **Anonymous Functions (Lambda Functions):**
   * Small, unnamed functions defined using the **lambda** keyword.

Python

square = lambda x: x\*\*2 print(square(5))

**fie handling**

File handling in Python involves various operations related to reading from and writing to files. Python provides built-in functions and methods for these operations. Here are the key concepts and operations for file handling in Python:

**Opening a File:**

To interact with a file, you need to open it first. The **open()** function is used for this purpose.

**Opening a File:**

To interact with a file, you need to open it first. The **open()** function is used for this purpose.

**Writing to a File:**

**Closing a File:**

It is important to close a file using the **close()** method when you are done with it.

**File Modes:**

* **Read Mode ("r"):** Used for reading from the file.
* **Write Mode ("w"):** Used for writing to the file. Creates a new file or truncates an existing file.
* **Append Mode ("a"):** Used for appending data to the end of the file.
* **Binary Mode ("b"):** Used for handling binary files.
* **Exclusive Creation Mode ("x"):** Creates a new file but fails if the file already exists.

Top of Form