

Module -1

Introduction To Python

Introduction to Python programming

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side)
- software development
- mathematics
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Features:

There are a lot of features provided by python programming language as follows:

- **Easy to Code:** Python is a very developer-friendly language which means that anyone and everyone can learn to code it in a couple of hours or days. As compared to other object-oriented programming languages like Java, C, C++, and C#, Python is one of the easiest to learn.
- **Open Source and Free:** Python is an open-source programming language which means that anyone can create and contribute to its development. Python has an online forum where thousands of coders gather daily to improve this language further. Along with this Python is free to download and use in any operating system, be it Windows, Mac or Linux.
- **Support for GUI:** GUI or Graphical User Interface is one of the key aspects of any programming language because it has the ability to add flair to code and make the results more visual. Python has support for a wide array of GUIs which can easily be imported to the interpreter, thus making this one of the most favorite languages for developers.

- **Object-Oriented Approach:** One of the key aspects of Python is its object-oriented approach. This basically means that Python recognizes the concept of class and object encapsulation thus allowing programs to be efficient in the long run.
- **Highly Portable:** Suppose you are running Python on Windows and you need to shift the same to either a Mac or a Linux system, then you can easily achieve the same in Python without having to worry about changing the code. This is not possible in other programming languages, thus making Python one of the most portable languages available in the industry.
- **Highly Dynamic:** Python is one of the most dynamic languages available in the industry today. What this basically means is that the type of a variable is decided at the run time and not in advance. Due to the presence of this feature, we do not need to specify the type of the variable during coding, thus saving time and increasing efficiency.
- **Large Standard Library:** Out of the box, Python comes inbuilt with a large number of libraries that can be imported at any instance and be used in a specific program. The presence of libraries also makes sure that you don't need to write all the code yourself and can import the same from those that already exist in the libraries.

Writing a Python Program

When we want to write a program, we use a text editor to write the Python instructions into a file, which is called a script. By convention, Python scripts have names that end with .py

To execute the script, you have to tell the Python interpreter the name of the file.

In a Unix or Windows command window, you would type python hello.py as follows:

```
print('Hello world!')
Hello world!
```

Python Interactive Shell

The **Python Interactive Shell**, also known as the **Python REPL (Read-Eval-Print Loop)**, is an environment where you can enter Python commands and see their results instantly. It is useful for testing code, debugging, and learning Python interactively.

Building blocks

In order to write any program, we must be aware of its structure, available keywords and data types and also required to have some knowledge on Variables, constants and identifiers.

Keywords, Identifiers and variables are the basic building blocks of python

1. **Variables** are used to store and manipulate data in a program. In Python, you can assign a value to a variable using the assignment operator (=).

Example

```
x = 5 # assign the value 5 to the variable x
print(x) # output: 5
```

2. Keywords

Python has 33 keywords (as of Python 3.8+), and these keywords cannot be used as variable names because they have special meanings in the language.

True	False	<u>None</u>	and
or	not	<u>is</u>	if
else	<u>elif</u>	<u>for</u>	<u>while</u>
<u>break</u>	<u>continue</u>	<u>pass</u>	<u>try</u>
<u>except</u>	<u>finally</u>	<u>raise</u>	<u>assert</u>
<u>def</u>	<u>return</u>	<u>lambda</u>	<u>yield</u>
<u>class</u>	<u>import</u>	from	<u>in</u>
<u>as</u>	<u>del</u>	<u>global</u>	<u>with</u>
<u>nonlocal</u>	Async	Await	

3. Data Types

Python has several built-in data types, including:

- Integers (int): whole numbers, e.g. 1, 2, 3, etc.
- Floating Point Numbers (float): decimal numbers, e.g. 3.14, -0.5, etc.
- Strings (str): sequences of characters, e.g. "hello", 'hello', etc.
- Boolean (bool): true or false values
- List (list): ordered collections of items, e.g. [1, 2, 3], ["a", "b", "c"], etc.
- Tuple (tuple): ordered, immutable collections of items, e.g. (1, 2, 3), ("a", "b", "c"), etc.

Example

```
x = 5 # integer
y = 3.14 # float
name = "John" # string
admin = True # boolean
numbers = [1, 2, 3] # list
colors = ("red", "green", "blue") # tuple
```

Values and Variables

Value

A value is one of the basic things a program works with, like a letter or a number.

The values we have seen so far are 1, 2, and "Hello, World!"

These values belong to different types: 2 is an integer, and “Hello, World!” is a string, so called because it contains a “string” of letters. You (and the interpreter) can identify strings because they are enclosed in quotation marks.

The print statement also works for integers. We use the python command to start the interpreter.

```
python
```

```
>>> print(4)
```

```
4
```

If you are not sure what type a value has, the interpreter can tell you.

```
>>> type('Hello, World!')
```

```
<class 'str'>
```

```
>>> type(17)
```

```
<class 'int'>
```

Variable

One of the most powerful features of a programming language is the ability to manipulate variables. A variable is a name that refers to a value.

An assignment statement creates new variables and gives them values:

```
>>> message = 'And now for something completely different'
```

```
>>> n = 17
```

```
>>> pi = 3.1415926535897931
```

This example makes three assignments.

The first assigns a string to a new variable named message;

The second assigns the integer 17 to n;

The third assigns the (approximate) value of π to pi.

To display the value of a variable, you can use a print statement:

```
>>> print(n)
```

```
17
```

```
>>> print(pi)
```

```
3.141592653589793
```

The type of a variable is the type of the value it refers to.

```
>>> type(message)
```

```
<class 'str'>
```

```
>>> type(n)
```

```
<class 'int'>
```

```
>>> type(pi)
```

```
<class 'float'>
```

Variables and Assignment

In Python, a variable is a name given to a value. Variables are used to store and manipulate data in a program. You can think of a variable as a labeled box where you can store a value.

Assignment in Python is the process of assigning a value to a variable. It is a fundamental concept in programming that allows you to store and manipulate data.

Types of Assignment

1. Simple Assignment: Assigns a value to a variable using the assignment operator (=).

Example

```
x = 5
```

2. Multiple Assignment: Assigns multiple values to multiple variables using the assignment operator (=).

Example

```
python  
x, y = 5, 10
```

3. Chained Assignment: Assigns a value to multiple variables using the assignment operator (=).

Example

```
x = y = 5
```

4. Augmented Assignment: Assigns a value to a variable using an augmented assignment operator (e.g., +=, -=, \=, /=).

Example

```
python  
x = 5  
x += 3  
print(x) # output: 8
```

5. Parallel Assignment: Assigns multiple values to multiple variables using the assignment operator (=) and the tuple unpacking syntax.

Example

```
x, y = (5, 10)
```

Identifiers

In Python, an identifier is a name given to a variable, function, class, module, or any other object.

Here are the rules for identifiers in Python:

Rules for Identifiers

1. Letters and Digits: Identifiers can contain letters (a-z, A-Z) and digits (0-9).
2. Underscore: Identifiers can also contain underscores (_).
3. No Special Characters: Identifiers cannot contain special characters like !, @, #, \$, etc.
4. No Keywords: Identifiers cannot be the same as Python keywords like if, else, for, while, etc.
5. Case-Sensitive: Identifiers are case-sensitive, meaning that 'name' and 'Name' are two different identifiers.
6. No Leading Digits: Identifiers cannot start with a digit.

Examples of Valid Identifiers

- name
- age
- total_sum
- averageGrade
- _private_variable

Examples of Invalid Identifiers

- 123name (starts with a digit)
- name! (contains a special character)
- if (is a Python keyword)
- total sum (contains a space)

Expressions

n expression in Python is a combination of values, variables, operators, and functions that evaluates to a single value. Expressions are used to perform calculations, manipulate data, and make decisions.

Types of Expressions

Python has several types of expressions, including:

1. Arithmetic Expressions: These expressions use arithmetic operators (+, -, *, /, etc.) to perform mathematical calculations.

Example

```
x = 5
y = 3
result = x + y
print(result) # output: 8
```

2. Comparison Expressions: These expressions use comparison operators (==, !=, >, <, etc.) to compare values.

Example

```
python
x = 5
y = 3
result = x > y
print(result) # output: True
```

3. Logical Expressions: These expressions use logical operators (and, or, not) to combine comparison expressions.

Example

```
x = 5
y = 3
result = x > y and x == 5
print(result) # output: True
```

4. Assignment Expressions: These expressions use assignment operators (=, +=, -=, etc.) to assign values to variables.

Example

```
python
x = 5
x += 3
print(x) # output: 8
```

5. Function Call Expressions: These expressions call functions with arguments.

Example

```
def greet(name):
    print("Hello, " + name + "!")

greet("John") # output: Hello, John!
```

6. List Expressions: These expressions create new lists using list comprehensions.

Example

```
python
numbers = [1, 2, 3, 4, 5]
double_numbers = [x * 2 for x in numbers]
print(double_numbers) # output: [2, 4, 6, 8, 10]
```

7. Dictionary Expressions: These expressions create new dictionaries using dictionary comprehensions.

Example

```
fruits = ["apple", "banana", "cherry"]
fruit_dict = {fruit: len(fruit) for fruit in fruits}
print(fruit_dict) # output: {'apple': 5, 'banana': 6, 'cherry': 6}
```

Arithmetic Operator

Arithmetic operators are used to perform mathematical operations on numbers. Python supports various arithmetic operators, including addition, subtraction, multiplication, division, modulus, exponentiation, and floor division.

Types of Arithmetic Operators

Here are the different types of arithmetic operators in Python:

Operators	Meaning	Example	Result
+	Addition	$4 + 2$	6
-	Subtraction	$4 - 2$	2
*	Multiplication	$4 * 2$	8
/	Division	$4 / 2$	2
%	Modulus operator to get remainder in integer division	$5 \% 2$	1
**	Exponent	$5 ** 2 = 5^2$	25
//	Integer Division/ Floor Division	$5 // 2$ $-5 // 2$	2 -3

1. Addition Operator (+): The addition operator is used to add two numbers.

Example

```
x = 5
y = 3
result = x + y
print(result) # Output: 8
```

2. *Subtraction Operator (-)*: The subtraction operator is used to subtract one number from another.

Example

```
python
x = 10
y = 4
result = x - y
print(result) # Output: 6
```

3. Multiplication Operator (*): The multiplication operator is used to multiply two numbers.

Example

```
x = 4
y = 5
result = x * y
print(result) # Output: 20
```

4. *Division Operator (/)*: The division operator is used to divide one number by another. It returns a floating-point result.

Example

```
python
```



```
x = 10
y = 2
result = x / y
print(result) # Output: 5.0
```

5. Modulus Operator (%): The modulus operator is used to find the remainder of a division operation.

Example

```
x = 17
y = 5
result = x % y
print(result) # Output: 2
```

6. Exponentiation Operator (\\): The exponentiation operator is used to raise a number to a power.

Example

```
python
x = 2
y = 3
result = x ** y
print(result) # Output: 8
```

7. Floor Division Operator (//): The floor division operator is used to divide one number by another and return the largest possible integer. It rounds down to the nearest whole number.

Example

```
x = 10
y = 3
result = x // y
print(result) # Output: 3
```

Logical Operator

Logical operators are used to evaluate logical expressions and return a boolean value (True or False). They are commonly used in conditional statements, loops, and function calls.

Types of Logical Operators

Python supports three logical operators:

and	Returns True if both statements are true	$x < 5$ and $x < 10$
or	Returns True if one of the statements is true	$x < 5$ or $x < 4$

not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)
-----	---	-----------------------

1. And Operator (and): The and operator returns True if both the operands are true.

Example

```
x = 5
y = 3
result = x > 3 and y < 5
print(result) # Output: True
```

2. *Or Operator (or)*: The or operator returns True if at least one of the operands is true.

Example

```
python
x = 5
y = 3
result = x > 3 or y > 5
print(result) # Output: True
```

3. Not Operator (not): The not operator returns True if the operand is false, and vice versa.

Example

```
x = 5
result = not x > 3
print(result) # Output: False
```

Relational Opeartor

Relational operators are used to compare two values and return a boolean value (True or False). They are commonly used in conditional statements, loops, and function calls.

Types of Relational Operators

Python supports six relational operators:

1. Equal To Operator (==): The equal to operator returns True if both operands are equal.

Example

```
x = 5
y = 5
result = x == y
print(result) # Output: True
```

2. *Not Equal To Operator (!=)*: The not equal to operator returns True if both operands are not equal.

Example

```
python
x = 5
y = 3
result = x != y
print(result) # Output: True
```

3. Greater Than Operator (>): The greater than operator returns True if the left operand is greater than the right operand.

Example

```
x = 5
y = 3
result = x > y
print(result) # Output: True
```

4. Less Than Operator (<): The less than operator returns True if the left operand is less than the right operand.

Example

```
python
x = 3
y = 5
result = x < y
print(result) # Output: True
```

5. Greater Than or Equal To Operator (>=): The greater than or equal to operator returns True if the left operand is greater than or equal to the right operand.

Example

```
x = 5
y = 5
result = x >= y
print(result) # Output: True
```

6. Less Than or Equal To Operator (<=): The less than or equal to operator returns True if the left operand is less than or equal to the right operand.

Example

```
python
```

```
x = 3
y = 5
result = x <= y
print(result) # Output: True
```

Operators and operands

Operators are special symbols that represent computations like addition and multiplication. The values the operator is applied to are called operands.

The operators +, -, *, /, and ** perform addition, subtraction, multiplication, division, and exponentiation, as in the following examples:

```
20+32 hour-1 hour*60+minute minute/60 5**2 (5+9)*(15-7)
```

There has been a change in the division operator between Python 2.x and Python 3.x. In Python 3.x, the result of this division is a floating point result:

```
>>> minute = 59
>>> minute/60
0.9833333333333333
```

The division operator in Python 2.0 would divide two integers and truncate the result to an integer:

```
>>> minute = 59
>>> minute/60
0
```

To obtain the same answer in Python 3.0 use floored (// integer) division.

```
>>> minute = 59
>>> minute//60
0
```

In Python 3.0 integer division functions much more as you would expect if you entered the expression on a calculator.

Order of operations

Python has a specific order of operations, known as operator precedence, which determines the order in which operators are evaluated.

When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence.

For mathematical operators, Python follows mathematical convention. The acronym PEMDAS is a useful way to remember the rules:

- Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, $2 * (3-1)$ is 4, and $(1+1)**(5-2)$ is 8. You can also use parentheses to make an expression easier to read, as in $(minute * 100) / 60$, even if it doesn't change the result.

- Exponentiation has the next highest precedence, so $2**1+1$ is 3, not 4, and $3*1**3$ is 3, not 27.

- Multiplication and Division have the same precedence, which is higher than Addition and Subtraction, which also have the same precedence. So $2*3-1$ is 5, not 4, and $6+4/2$ is 8.0, not 5

- Operators with the same precedence are evaluated from left to right. So the expression 5-3-1 is 1, not 3, because the 5-3 happens first and then 1 is subtracted from 2.

Example

```
python
x = 5
y = 3
result = x + y * 2
print(result)
```

OUTPUT: 11

Control codes within strings

Control codes, also known as escape sequences, are special characters within strings that are used to represent non-printable characters, formatting, or other special instructions.

Types of Control Codes

Python supports several types of control codes within strings:

- 1. Newline (\n):** Represents a new line.

Example

```
print("Hello\nWorld")
```

Output:

```
Hello
World
```

- 2. *Tab (\t)*:** Represents a horizontal tab.

Example

```
python
print("Hello\tWorld")
```

Output:

```
Hello  World
```

- 3. Backslash (\):** Represents a backslash.

Example

```
print("Hello\\World")
```

Output:

Hello\World

4. Single Quote (\'): Represents a single quote within a single-quoted string.

Example

```
python  
print('Hello\'World')
```

Output:
Hello'World

5. Double Quote ("): Represents a double quote within a double-quoted string.

Example

```
print("Hello"World")
```

Output:
Hello"World

Comments

Comments in Python are the lines in the code that are ignored by the interpreter during the execution of the program.

- Comments enhance the readability of the code.
- Comment can be used to identify functionality or structure the code-base.
- Comment can help understanding unusual or tricky scenarios handled by the code to prevent accidental removal or changes.
- Comments can be used to prevent executing any specific part of your code, while making changes or testing.

In Python, **single line comments** starts with hashtag symbol #.

```
# sample comment  
name = "geeksforgeeks"  
print(name)
```

OUTPUT

geeksforgeeks

Multi-Line Comments

Python does not provide the option for multiline comments. However, there are different ways through which we can write multiline comments.

Multiline comments using multiple hashtags (#)

We can multiple hashtags (#) to write multiline comments in Python. Each and every line will be considered as a single-line comment.

```
# Python program to demonstrate  
# multiline comments  
print("Multiline comments")
```

Using String Literals as Comment

Python ignores the string literals that are not assigned to a variable. So, we can use these string literals as Python Comments.

'Single-line comments using string literals'

```
""" Python program to demonstrate
multiline comments """
print("Multiline comments")
```

User input and Output

Python allows for user input. That means we are able to ask the user for input.

Python uses the input() function.

Ex:

- username= input("Enterusername:")
print("Username is: " + username)

The print() function prints the specified message to the screen, or other standard output device.

```
print("hello")
```

Python provides several ways to get user input, including:

1. input() function: The input() function is used to get user input. It returns a string.

Example

```
name = input("Enter your name: ")
print("Hello, " + name)
```

2. raw_input() function: The raw_input() function is used to get user input in Python 2.x. It returns a string.

```
## Example (Python 2.x)
```

```
python
name = raw_input("Enter your name: ")
print "Hello, " + name
```

3. int(input()): To get integer input, use the int() function with input().

Example

```
age = int(input("Enter your age: "))
print("You are " + str(age) + " years old.")
```

4. float(input()): To get floating-point input, use the float() function with input().

Example

```
python
height = float(input("Enter your height: "))
print("You are " + str(height) + " meters tall.")
```

User Output

Python provides several ways to display output to the user, including:

1. print() function: The print() function is used to display output to the user.

Example

```
print("Hello, World!")
```

2. print() with variables: You can use variables with the print() function to display dynamic output.

Example

```
name = "John"
age = 30
print("My name is " + name + " and I am " + str(age) + " years old.")
```

3. print() with formatting: You can use formatting with the print() function to display output in a specific format.

Example

```
name = "John"
age = 30
print("My name is {} and I am {} years old.".format(name, age))
```

Errors in Python

Run-Time Exceptions

These occur during the execution of the program, such as dividing by zero or trying to access an index that is out of range.

Example

```
x = 5 / 0
```

Output:

```
ZeroDivisionError: division by zero
```

Syntax errors

These are the first errors you will make and the easiest to fix. A syntax error means that you have violated the “grammar” rules of Python.

Python does its best to point right at the line and character where it noticed it was confused. The only tricky bit of syntax errors is that sometimes the mistake that needs fixing is actually

earlier in the program than where Python noticed it was confused. So the line and character that Python indicates in a syntax error may just be a starting point for your investigation.

These occur when there is an error in the syntax of the program, such as a missing colon or a mismatched parenthesis.

Example

```
print("Hello World"
```

Output:

SyntaxError: unexpected EOF while parsing

Logic Error

Logic errors A logic error is when your program has good syntax but there is a mistake in the order of the statements or perhaps a mistake in how the statements relate to one another. A good example of a logic error might be, “take a drink from your water bottle, put it in your backpack, walk to the library, and then put the top back on the bottle.”

These occur when the program is syntactically correct but does not produce the expected output, such as an infinite loop or a calculation error.

Example

```
python
x = 5
while x > 0:
    x += 1
```

Output:

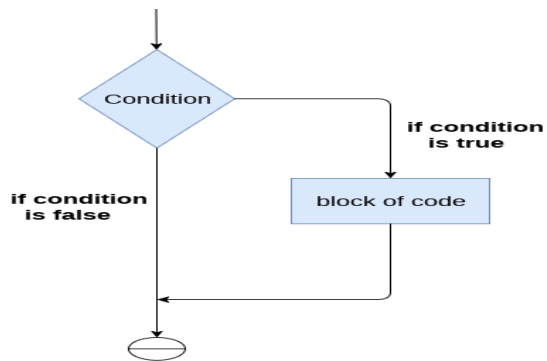
This will cause an infinite loop.

Conditional Statement

Conditional statements let you check if something is true or false and then perform various actions based on that.

Simple IF statement

- The [if statement](#) is the most simple decision-making statement.
- It is used to decide whether a certain statement or block of statements will be executed or not.



Syntax:

if expression:
statement

Ex:

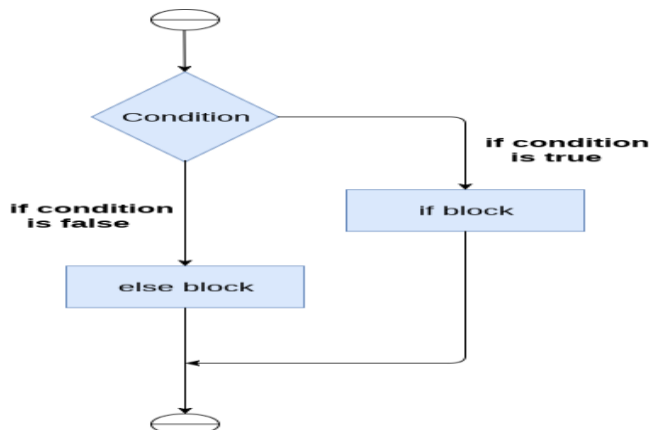
```
num = int(input("enter the number:"))
```

```
if num%2 == 0:
```

```
    print("The Given number is an even number")
```

IF...ELSE statement

- The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.



Syntax:

if condition:
statements

else:
statements (else-block)

Ex1:

```
age = int (input("Enter your age: "))
```

```
if age>=18:
```

```
    print("You are eligible to vote !!");
```

```
else:
```

```
    print("Sorry! you have to wait !!");
```

Ex2:

```
num = int(input("enter the number:"))
if num%2 == 0:
    print("The Given number is an even number")
else:
    print("The Given Number is an odd number")
```

Compound Boolean expressions

A boolean expression is an expression that is either true or false. The following examples use the operator `==`, which compares two operands and produces True if they are equal and False otherwise:

```
>>> 5 == 5
```

True

```
>>> 5 == 6
```

False

```
{}
```

True and False are special values that belong to the class bool; they are not strings:

```
>>> type(True)
```

```
<class 'bool'>
```

```
>>> type(False)
```

```
<class 'bool'>
```

The `==` operator is one of the comparison operators; the others are:

`x != y` # x is not equal to y

`x > y` # x is greater than y

`x < y` # x is less than y

`x >= y` # x is greater than or equal to y

`x <= y` # x is less than or equal to y

`x is y` # x is the same as y

`x is not y` # x is not the same as y

Although these operations are probably familiar to you, the Python symbols are different from the mathematical symbols for the same operations. A common error is to use a single equal sign (`=`) instead of a double equal sign (`==`). Remember that `=` is an assignment operator and `==` is a comparison operator. There is no such thing as `=<` or `=>`.

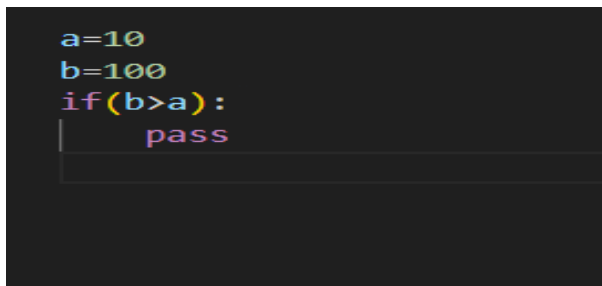
Pass Statement

Pass statement in **Python** is a null operation or a placeholder. It is used when a statement is syntactically required but we don't want to execute any code. It does nothing but allows us to maintain the structure of our program.

Example of using pass in an empty function-1

```
def fun():  
    pass # Placeholder, no functionality yet  
  
# Call the function  
  
fun()
```

Example-2



```
a=10  
b=100  
if(b>a):  
    pass
```

- function fun() is defined but contains the pass statement, meaning it does nothing when called.
- program continues execution without any errors and the message is printed after calling the function.

Nested Conditionals

Nested conditions in Python refer to placing one if statement inside another if, elif, or else block. This is useful when you need to check multiple conditions that depend on each other.

```
if x == y:  
    print('x and y are equal')  
else:  
    if x < y:  
        print('x is less than y')  
    else:  
        print('x is greater than y')
```

The outer conditional contains two branches. The first branch contains a simple statement. The second branch contains another if statement, which has two branches of its own. Those two

branches are both simple statements, although they could have been conditional statements as well.

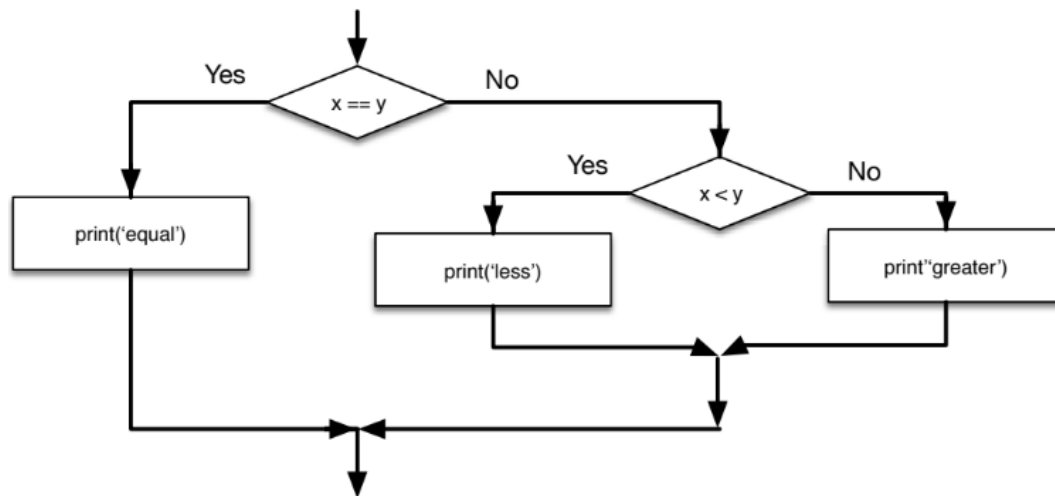


Figure 3.4: Nested If Statements

Although the indentation of the statements makes the structure apparent, nested conditionals become difficult to read very quickly. In general, it is a good idea to avoid them when you can.

Logical operators often provide a way to simplify nested conditional statements.

For example, we can rewrite the following code using a single conditional:

```
if 0 < x:
```

```
    if x < 10:
```

```
        print('x is a positive single-digit number.')
```

The print statement is executed only if we make it past both conditionals, so we can get the same effect with the and operator:

```
if 0 < x and x < 10:
```

```
    print('x is a positive single-digit number.')
```

Iteration Statement

- Loops enable you to perform repetitive tasks efficiently without writing redundant code.
- They iterate over a sequence (like a list, tuple, string, or range) or execute a block of code as long as a specific condition is met.

Types of Loops in Python

1. For Loop

2. While Loop
3. Loop Control Statements (break, continue, pass)

While

- While loop is used to execute a block of statements repeatedly until a given condition is satisfied.
- When the condition becomes false, the line immediately after the loop in the program is executed.

Syntax:

```
while expression:  
    statement(s)
```

Ex:

```
count = 0  
while (count < 3):  
    count = count + 1  
    print("Hello Geek")
```

All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code.

Syntax:

```
while condition:  
    statements  
else:  
    statements
```

Ex:

```
count = 0  
while (count < 3):  
    count = count + 1  
    print("Hello Geek")  
else:  
    print("In else block")
```

for statement

- [For loops](#) are used for sequential traversal
- For example: traversing a [list](#) or [string](#) or [array](#) etc.

Syntax:

```
for iterator_var in sequence:  
    statements(s)
```

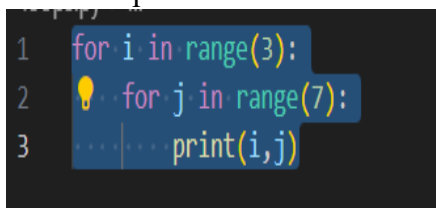
Looping Through a String

Ex:

- ```
for x in "banana":
 print(x)
```
- ```
for each_character in "Blue":  
    print(f'Iterate through character {each_character} in the string 'Blue''')
```

Nested loop

- A nested loop is a loop inside another loop. The inner loop runs completely for each iteration of the outer loop.
- For Example



```
1 for i in range(3):  
2     for j in range(7):  
3         print(i,j)
```

Break Statement

- Whenever the break statement is encountered, the execution control immediately jumps to the first instruction following the loop.
- With the break statement we can stop the loop before it has looped through all the items.


Ex:

```
for x in range(6):  
    if x == 3:  
        break  
    print(x)  
else:  
    print("Finally finished!")
```

If the loop breaks, the else block is not executed.

Working of Python break Statement

```
for val in sequence:
    # code
    if condition:
        break
    # code
```



Continue Statements

- With the continue statement we can stop the current iteration of the loop, and continue with the next.
- To pass control to the next iteration without exiting the loop, use the continue statement.

Ex:

```
for x in range(6):
```


```
    if x == 3:
```

```
        continue
```

```
    print(x)
```

Working of continue Statement in Python

```
for val in sequence:
    # code
    if condition:
        continue
    # code
```



while/else

The while loop is used for iteration. It executes a block of code repeatedly until the condition becomes false and when we add an "else" statement just after the while loop it becomes a "While-Else Loop". This else statement is executed only when the while loops are executed completely and the condition becomes false. If we break the while loop using the "break" statement then the else statement will not be executed.

```
i= 1
```

```
while i< 6:
```



```
print(i)
i+= 1
else:
    print("i is no longer less than 6")
```

Syntax:

while(Condition):

Code to execute while the condition is true

else:

Code to execute after the loop ends naturally

for/else statements

The else keyword in a for loop specifies a block of code to be executed when the loop is finished.

WAPP to print all numbers from 0 to 5, and print a message when the loop has ended

```
for x in range(6):
    print(x)
else:
    print("Finally finished!")
```

Python Program to Check Prime Number

```
# Program to check if a number is prime or not num
= 29

# To take input from the user
#num = int(input("Enter a number: "))

# define a flag variable flag =
False

if num == 1:
    print(num, "is not a prime number")
elif num > 1:
    # check for factors
    for i in range(2, num): if
        (num % i) == 0:
            # if factor is found, set flag to True flag =
            True
            # break out of loop
            break

    # check if flag is True if
    flag:
        print(num, "is not a prime number")
    else:
        print(num, "is a prime number")
```

output:

29 is a prime number

Factorial of a Number using Loop

42

```
# Python program to find the factorial of a number provided by the user.

# change the value for a different result num =
7

# To take input from the user
#num = int(input("Enter a number: "))

factorial = 1

# check if the number is negative, positive or zero if
```

```

num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1") else:
    for i in range(1,num + 1):
        factorial = factorial*i
    print("The factorial of",num,"is",factorial)

```

Output

The factorial of 7 is 5040

Python Program to Check if a Number is Positive, Negative or 0

```

num = float(input("Enter a number: ")) if
num > 0:
    print("Positive number")
elif num == 0:
    print("Zero") else:
    print("Negative number")

```

Output:

Enter a number: -3

Negative number

Python Program to Check if a Number is Odd or Even

Python program to check if the input number is odd or even. #
A number is even if division by 2 gives a remainder of 0.
If the remainder is 1, it is an odd number.

```

num = int(input("Enter a number: ")) if
(num % 2) == 0:

    print("{0} is Even".format(num))
else:
    print("{0} is Odd".format(num))

```

Output:

Enter a number: 43
43 is Odd

Python Program to Find the Largest Among Three Numbers

```
# Python program to find the largest number among the three input numbers #
change the values of num1, num2 and num3
# for a different result num1
= 10
num2 = 14
num3 = 12

# uncomment following lines to take three numbers from user
#num1 = float(input("Enter first number: "))
#num2 = float(input("Enter second number: "))
#num3 = float(input("Enter third number: "))

if (num1 >= num2) and (num1 >= num3):
    largest = num1
elif (num2 >= num1) and (num2 >=
    num3): largest = num2
else:
    largest = num3

print("The largest number is", largest)
```

Output:

The largest number is 14.0