Chapter Two

Basics of Python Programming

Collage of Computing

Department of Software Engineering

Compiled by Desalegn A

Introduction to python

- **Python** is a widely used programming language that offers several unique features and advantages compared to languages like **Java and C++.**
- Python is an **object-oriented programming language** created by **Guido Rossum in 1989**. It is ideally designed for rapid prototyping of **complex applications**.
- **Python is one of the most popular programming languages. Although it is a general-purpose language, it is used in various areas of applications such as Machine Learning, Artificial Intelligence, web development, IoT, and more.**
- Many large companies use the Python programming language, including NASA, Google, YouTube, BitTorrent, etc.

Why learn Python?

Python provides many useful features to the programmer:-

- **Easy to use and Read** - Python's syntax is clear and easy to read, making it an ideal language for both beginners and experienced programmers
- **Dynamically Typed** - The data types of variables are determined during run-time. We do not need to specify the data type of a variable during codes
- **High-level** - High-level language means human readable code
- **Garbage Collected** - Memory allocation and de-allocation are automatically managed. Programmers do not specifically need to manage the memory

Why learn Python?

- **Cross-platform Compatibility - Python can be easily installed on Windows, macOS, and various Linux distributions.**
- **Rich Standard Library - Python comes with several standard libraries that provide ready-to-use modules and functions.**
- **Open Source - Python is an open-source, cost-free programming language.**
- **GUI Programming Support:** Python provides several GUI frameworks, such as Tkinter to allowing developers to create desktop applications easily.
- **Wide Range of Libraries and Frameworks:** Python has a vast collection of libraries and frameworks, such as NumPy, Pandas, Django, and Flask, that can be used to solve a wide range of problems.

Application area of python

- **Data Science**: Data Science is a vast field, and Python is an important language for this field because of its simplicity, ease of use, and availability of powerful data analysis and visualization libraries like NumPy, Pandas, and Matplotlib.
- **Desktop Applications:** Tkinter are useful libraries that can be used in GUI - Graphical User Interface-based Desktop Applications. There are better languages for this field, but it can be used with other languages for making Applications.
- **Mobile Applications:** While Python is not commonly used for creating mobile applications, it can still be combined with frameworks like Kivy or BeeWare to create cross-platform mobile applications.

Application area of python

- **Web Applications:** Python is commonly used in web development on the backend with frameworks like Django and Flask and on the front end with tools like JavaScript HTML and CSS.

- **Machine Learning:** Python is widely used for machine learning due to its simplicity, ease of use, and availability of powerful machine learning libraries

- **Computer Vision or Image Processing Applications:** Python can be used for computer vision and image processing applications through powerful libraries such as OpenCV and Scikit-image.

- **Gaming:** Python has libraries like Pygame, which provide a platform for developing games using Python.

DevOps: Python is widely used in DevOps for automation and scripting of infrastructure management, configuration management, and deployment processes

Python Popular Frameworks and Libraries

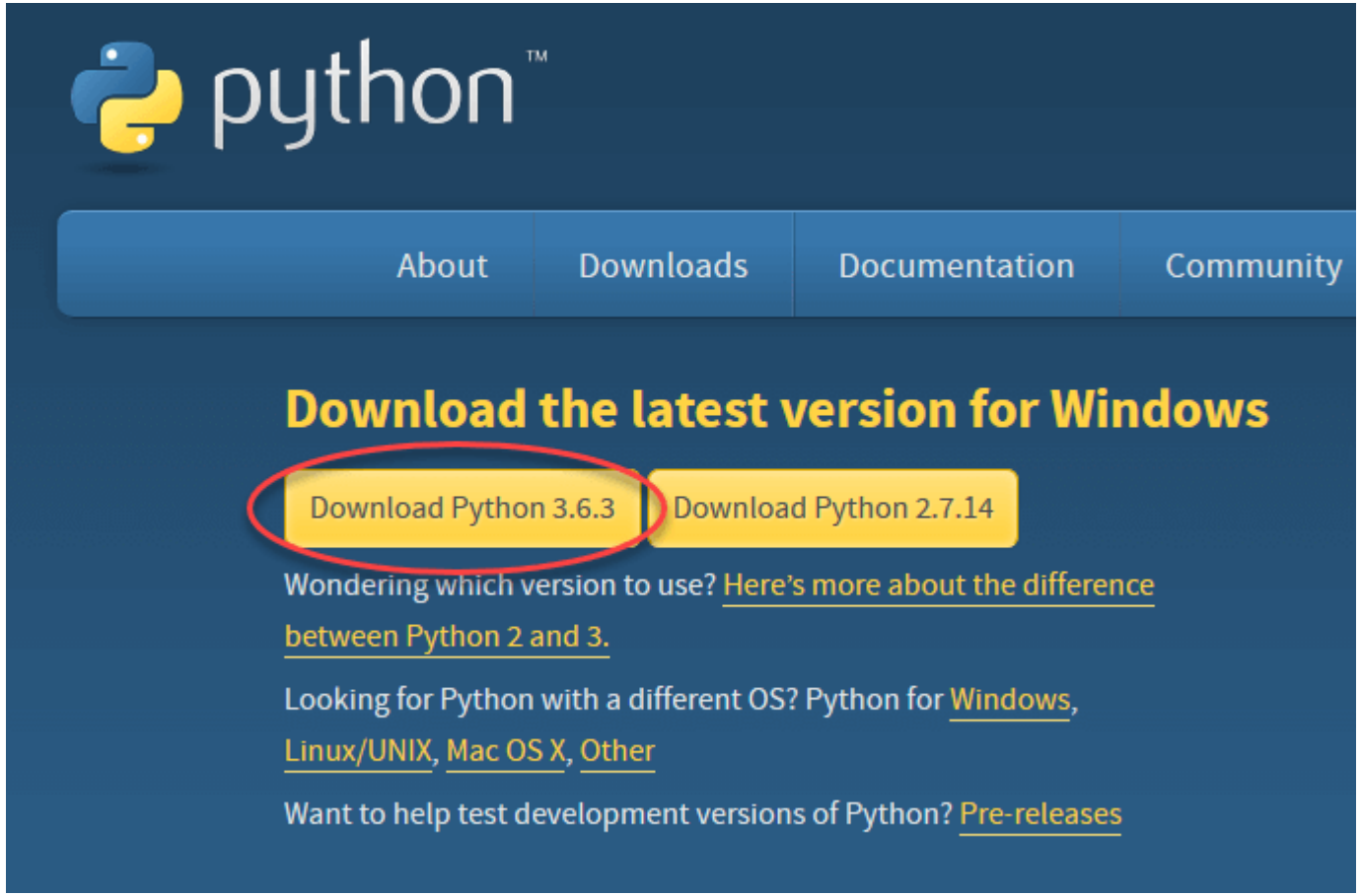
- Python has wide range of libraries and frameworks widely used in various fields such
- **Web development (Server-side)** - Django Flask, Pyramid, CherryPy
- **GUIs based applications** - Tkinter, PyGTK, PyQt, PyJs, etc.
- **Machine Learning** - TensorFlow, PyTorch, Scikit-learn, Matplotlib, Scipy, etc.
- **SQLAlchemy:** a library for working with SQL databases
- **NLTK:** a library for natural language processing
- **Pygame**: a library for game development
- **Pytest:** a testing framework for Python Django
- **REST framework**: a toolkit for building RESTful API

Advantages of Python language

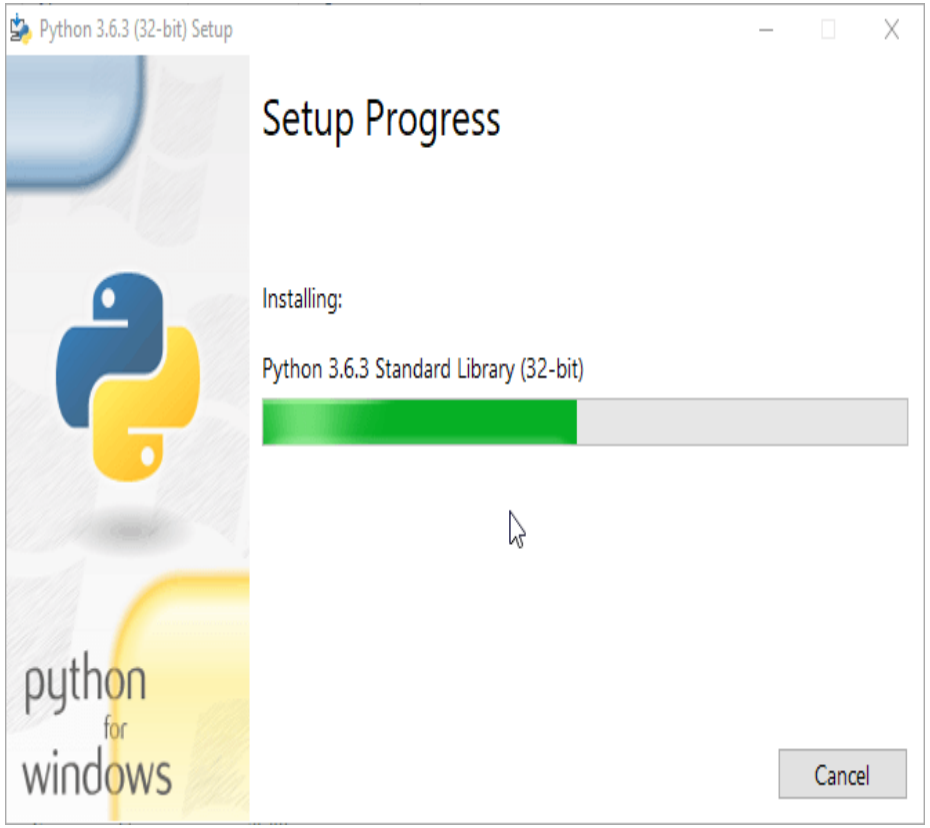
- Cross-Platform Compatibility
- Strong Community Support
- Integration and Extensibility
- Scalability and Performance
- Versatility and Flexibility

How to install Python

➢1. Visit the official website of Python https://www.python.org/downloads/ and choose your version.

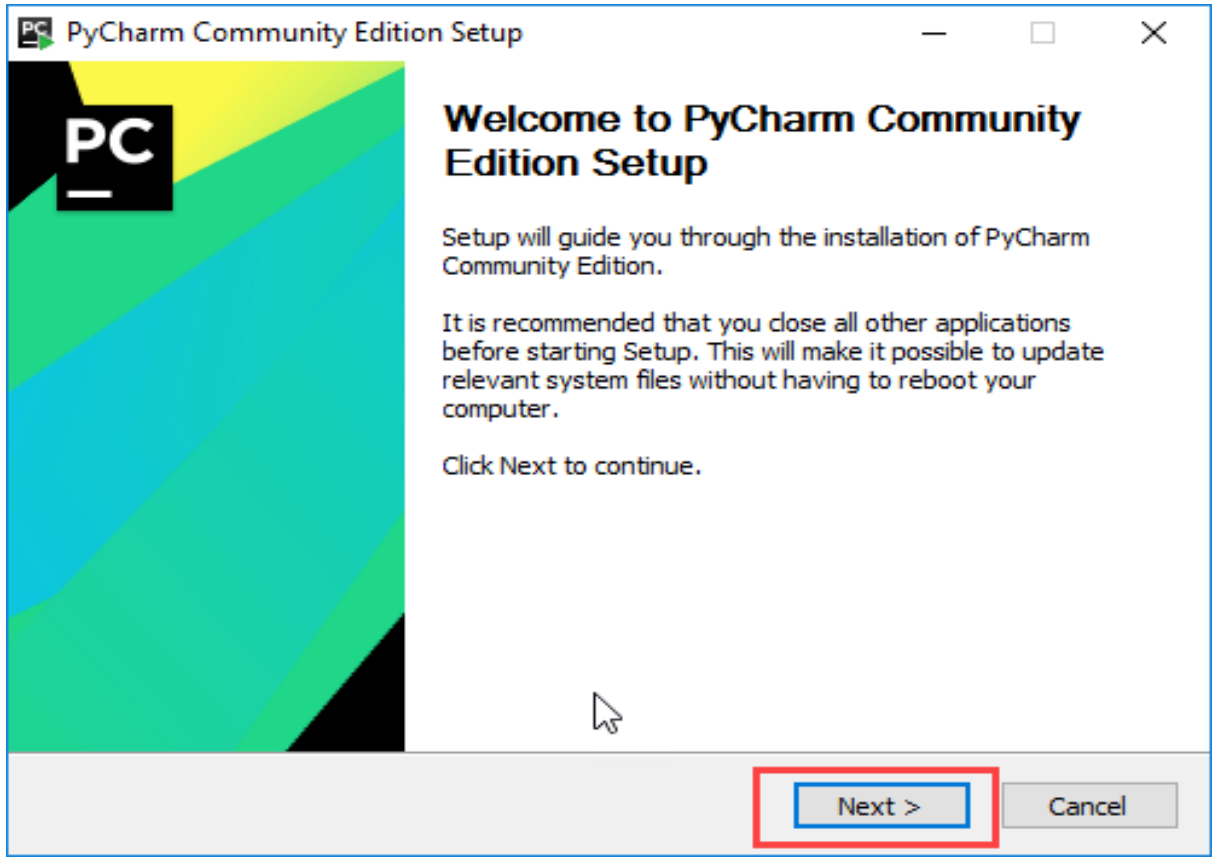


Once the download is completed, run the .exe file to install Python. Now click on Install Now.



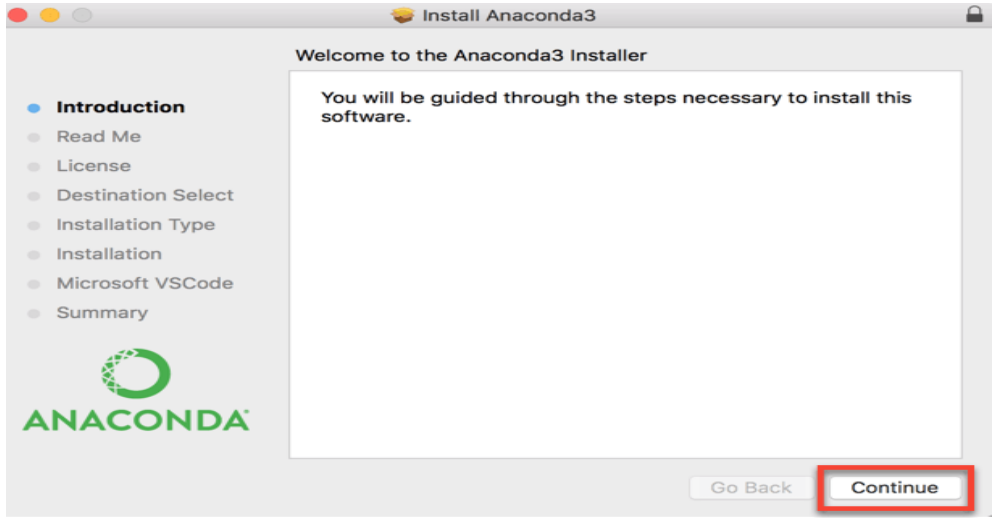
How to install Python IDE

- **2. PyCharm is a cross-platform editor developed by JetBrains. Pycharm provides all the tools you need for productive Python development.**
- visit the website https://www.jetbrains.com/pycharm/download/ and Click the "DOWNLOAD" link under the Community Section.



How to install Python IDE

- 2. **Anaconda** is a free and open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing.
- An anaconda is an open-source free path that allows users to write programming in Python language. The **anaconda** is termed by navigator as it includes various applications of Python such as **Spyder, Vs code, Jupiter notebook, PyCharm**
- How to install anaconda: Go to https://www.anaconda.com/download/ and download Anaconda



Basic elements of Python:

- **Python Basic Syntax:-**
- **There is no use of curly braces or semicolons in Python programming language. It is an English-like language.**
- **But Python uses indentation to define a block of code. Indentation is nothing but adding whitespace before the statement.**
- Python is a case-sensitive language, which means that uppercase and lowercase letters are treated differently.
- In Python, comments can be added using the '#' symbol. Any text written after the '#' symbol is considered a comment.
- Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments: example ''' This is a multiline comment. '''

Basic elements of Python:

Python print() Function

- Python print() function is used to display output to the console or terminal.
- It allows us to display text, variables and other data in a human readable format.

```
# Displaying a string
print("Hello, World!")
# Displaying multiple values
name = "Aman"
age = 21
print("Name:", name, "Age:", age)
```

Output

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\Desu> python -u "c:\Users\Desu\Desktop\gg\firstprogram python.py"
Hello, World!
Name: Aman Age: 21
```

Python accepts single (‘), double (”) and triple (‘‘‘ or ”””) quotes to denote string literals, as long as the same type of quote starts and ends the string.

Python Variables:

Python Variables

- Python variables are the reserved memory locations used to store values with in a Python Program.
- When you create a variable you reserve some space in the memory
- Based on the data type of a variable, Python interpreter allocates memory and decides what can be stored in the reserved memory
- Python variables do not need explicit declaration to reserve memory space or you can say to create a variable.
- A Python variable is created automatically when you assign a value to it. The equal sign (=) is used to assign values to variables.

Example

```
counter = 100          # Creates an integer variable
miles   = 1000.0       # Creates a floating point variable
name    = "Zara Ali"   # Creates a string variable
```

Print (counter)
Print(miles)
Print(name)
or print(coutter, miles,name)

Deleting Python Variables:

- You can delete the reference to a number object by using the del statement. The syntax of the del statement is

del var name

Example

```
#Declare a variable and initialize it
f = 11;
print(f)



del f
print(f)
```

```
Run  Python5.4
"C:\Users\DK\Desktop\Python code\Python
 5/PythonCode5/Python5.4.py"
11
Traceback (most recent call last):
  File "C:/Users/DK/Desktop/Python code/
    print(f)
NameError: name 'f' is not defined
```

Once you delete variable f and print f it will show this comment, which means your variable is now deleted

- You can get the data type of a Python variable using the python built-in function **type()** as follows.

Example: Printing Variables Type

```
x = "Zara"
y =  10
z =  10.10
print(type(x))
print(type(y))
print(type(z))
```

Output

Python Variables - Multiple Assignment:

- Python allows to initialize more than one variables in a single statement.
- In example A , three variables have same value.

Example A

```
a=b=c=10
print (a,b,c)
```

Output

```
a,b,c = 10,20,30
print (a,b,c)
```

Output

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number or any special character like $, (, * % etc.)
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Python variable names are case-sensitive which means Name and NAME are two different variables in Python.
- Python reserved keywords cannot be used naming the variable.

Constants in Python

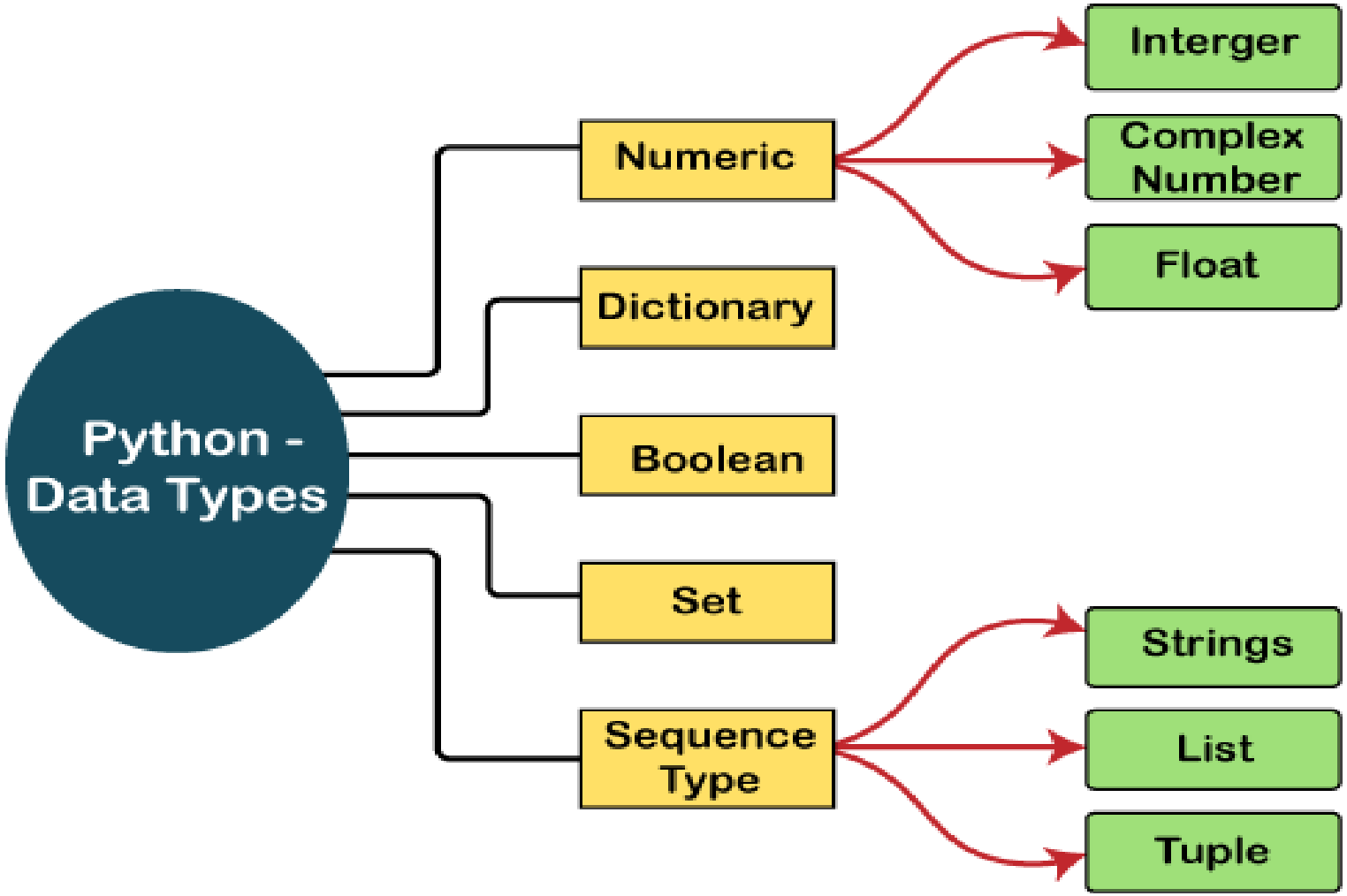
Python constant

Python doesn't have any formally defined constants, However you can indicate a variable to be treated as a constant by using all-caps names with underscores

For example, the name PI_VALUE indicates that you don't want the variable redefined or changed in any way.

Data Types in Python:

- A data type represents a kind of value and determines what operations can be done on it. It defines what type of data we are going to store in a variable.



Python Numeric Data Type

Python supports four different numerical types and each of them have built-in classes in Python library, called int, bool, float and complex

firstprogram python.py

C: > Users > Desu > Desktop > gg > firstprogram python.py > ...

```python
# integer variable.
a=100
print("The type of variable having value", a, " is ", type(a))
# boolean variable.
b=True
print("The type of variable having value", b, " is ", type(b))
# float variable.
c=20.345
print("The type of variable having value", c, " is ", type(c))
# complex variable.
d=10+3j
print("The type of variable having value", d, " is ", type(d))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
 made up of multiple lines and sentences.
PS C:\Users\Desu> python -u "c:\Users\Desu\Desktop\gg\firstprogram python.py"
The type of variable having value 100  is  <class 'int'>
The type of variable having value True  is  <class 'bool'>
The type of variable having value 20.345  is  <class 'float'>
The type of variable having value (10+3j)  is  <class 'complex'>
```

Python Sequence Data Type

- Sequence is a collection data type. It is an ordered collection of items. Items in the sequence have a positional index starting with 0. It is conceptually similar to an array in C++.

1. Python String Data Type

- Python string is a sequence of one or more Unicode characters, enclosed in single, double or triple quotation marks.
- Python strings are immutable which means when you perform an operation on strings, you always produce a new string object of the same type, rather than mutating an existing string

Python Sequence Data Type

- A string is a non-numeric data type. Obviously, we cannot perform arithmetic operations on it. However, operations such as slicing and concatenation can be done.

- Python's str class defines a number of useful methods for string processing. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator in Python.

Python Sequence Data Type

Example of String Data Type

firstprogram python.py ×

C: > Users > Desu > Desktop > gg > firstprogram python.py > ...

```python
str = 'Hello World!'
print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])      # Prints string starting from 3rd character
print (str * 2)      # Prints string two times
print (str + "TEST") # Prints concatenated string
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Desu> python -u "c:\Users\Desu\Desktop\gg\firstprogram python.py"
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
PS C:\Users\Desu>
```

Output

Python Sequence Data Type

2. Python List Data Type

- A Python list contains items separated by commas and enclosed within square brackets ([]).
- Python lists are similar to arrays in C++. One difference between them is that all the items belonging to a Python list can be of different data type. List declaration

Example

```
x= [2023, "Python", 3.11, 5+6j, 1.23E-4]
print(type(x))
<class 'list'>
```

```
x= [['One', 'Two', 'Three'], [1,2,3], [1.0, 2.0, 3.0]]
print(type(x))
<class 'list'>
```

Python Sequence Data Type

Python List Data Type

➢The values stored in a Python list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']
print (list)          # Prints complete list
print (list[0])       # Prints first element of the list
print (list[1:3])     # Prints elements starting from 2nd till 3rd
print (list[2:])      # Prints elements starting from 3rd element
print (tinylist * 2)  # Prints list two times
print (list + tinylist) # Prints concatenated lists
```

Output

Python Sequence Data Type

3. Python Tuple Data Type

- A Python tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses (...)
- A tuple is also a sequence, hence each item in the tuple has an index referring to its position in the collection. The index starts from 0. Tuple declaration

Example

```
x=(2023, "Python", 3.11, 5+6j, 1.23E-4)
print(type(x))
<class 'tuple'>
```

```
x=(['One', 'Two', 'Three'], 1,2.0,3, (1.0, 2.0, 3.0))
print(type(x))
<class 'tuple'>
```

Data Types in Python:

Python Tuples Data Type :- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed that means lists are mutable, while tuples are enclosed in parentheses (( )) and cannot be updated (immutable).

Tuples can be thought of as read-only lists

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')
print (tuple)       # Prints the complete tuple
print (tuple[0])    # Prints first element of the tuple
print (tuple[1:3])  # Prints elements of the tuple starting from 2nd till 3rd
print (tuple[2:])   # Prints elements of the tuple starting from 3rd element
print (tinytuple * 2)  # Prints the contents of the tuple twice
print (tuple + tinytuple) # Prints concatenated tuples
```

Output

Python Boolean Data Types

- Python **Boolean** type is one of built-in data types which represents one of the two values either **True** or **False**.
- Python **bool()** function allows you to evaluate the value of any expression and returns either True or False based on the expression.

firstprogram python.py

C: > Users > Desu > Desktop > gg > firstprogram python.py > ...

```python
a = True
# display the value of a
print(a)
# display the data type of a
print(type(a))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Desu> python -u "c:\Users\Desu\Desktop\gg\firstprogram python.py"
True
<class 'bool'>
PS C:\Users\Desu>
```

firstprogram python.py

C: > Users > Desu > Desktop > gg > firstprogram python.py > ...

```python
# Returns false as a is not equal to b
a = 2
b = 4
print(bool(a==b))
# Following also prints the same
print(a==b)
# Returns False as a is None
a = None
print(bool(a))
# Returns false as a is an empty sequence
a = ()
print(bool(a))
# Returns false as a is 0
a = 0.0
print(bool(a))
# Returns false as a is 10
a = 10
print(bool(a))
```

Python Input/output operations

- Python provides us with two built-in functions to read the input from the keyboard. The input () Function The raw_input () Function

- When the interpreter encounters input() function, it waits for the user to enter data from the standard input stream (keyboard)

firstprogram python.py

C: > Users > Desu > Desktop > gg > firstprogram python.py > ...

```python
name = input("Enter your name : ")
city = input("Enter your city : ")
print ("Hello My name is", name)
print ("I am from ", city)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Desu> python -u "c:\Users\Desu\Desktop\gg\firstprogram python.py"
Enter your name : abebe
Enter your city : Dbr
Hello My name is abebe
I am from  Dbr
PS C:\Users\Desu>
```

Taking Numeric Input in Python

➢Let us write a Python code that inputs width and height of a rectangle from the user and computes the area.)

firstprogram python.py ×

C: > Users > Desu > Desktop > gg > firstprogram python.py > ...

```python
width = input("Enter width : ")
height = input("Enter height : ")
area = width*height
print ("Area of rectangle = ", area)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Desu> python -u "c:\Users\Desu\Desktop\gg\firstprogram python.py"
Enter width : 45
Enter height : 70
Traceback (most recent call last):
  File "c:\Users\Desu\Desktop\gg\firstprogram python.py", line 3, in <module>
    area = width*height
           ~~~~~~^~~~~~~
TypeError: can't multiply sequence by non-int of type 'str'
PS C:\Users\Desu>
```

The reason is, Python always read the user input as a string.
Hence, width="45" and height="70" are the strings and obviously you cannot perform multiplication of two strings.

- ➢ To overcome this problem, we shall use **int()**, another built-in function from Python's standard library.
- ➢ It converts a string object to an integer.
- ➢ To accept an integer input from the user, read the input in a string, and type cast it to integer with int() function

Taking Numeric Input in Python

Let us write a Python code that accepts integer value of width and height of a rectangle from the user and computes the area.)

firstprogram python.py

C: > Users > Desu > Desktop > gg > firstprogram python.py > ...

```python
width = int(input("Enter width : "))
height = int(input("Enter height : "))
area = width*height
print ("Area of rectangle = ", area)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Desu> python -u "c:\Users\Desu\Desktop\gg\firstprogram python.py"
Enter width : 45
Enter height : 80
Area of rectangle =  3600
PS C:\Users\Desu>
```

- We can use **float()** function converts a string into a float object.
- Write a python program accepts the user input and parses it to a float variable

Operators in Python

- The operator is a symbol that performs a specific operation between two operands. Python also has some operators

1. Arithmetic operators

- Arithmetic operators used between two operands for a particular operation. There are many arithmetic operators. It includes the exponent (**) operator as well as the + (addition), - (subtraction), * (multiplication), / (divide), % (reminder), and // (floor division) operators.
- **** (Exponent):-** As it calculates the first operand's power to the second operand, it is an exponent operator.
- **// (Floor division):-It** provides the quotient's floor value, by dividing the two operands.

Operators in Python

2. Comparison operator

Comparison operators compare the values of the two operands and return a true or false Boolean value in accordance. The example of comparison operators are ==, !=, <=, >=, >, <.

a = 32 # Initialize the value of a.py

C: > Users > Desu > a = 32 # Initialize the value of a.py > ...

```python
a = 32        # Initialize the value of a
b = 6         # Initialize the value of b
print('Two numbers are equal or not:',a==b)
print('Two numbers are not equal or not:',a!=b)
print('a is less than or equal to b:',a<=b)
print('a is greater than or equal to b:',a>=b)
print('a is greater b:',a>b)
print('a is less than b:',a<b)
```

Output

Operators in Python

3. Assignment Operators

- Using the assignment operators, the right expression's value is assigned to the left operand. There are some examples of assignment operators like =, +=, -=, *=, %=, **=,

C: > Users > Desu > a = 32 # Initialize the value of a.py > ...

```python
a = 32          # Initialize the value of a
b = 6           # Initialize the value of b
print(a==b)     # Compare values of a and b
a += b          # Perform addition and update a
print(a)        # Print the updated value of a
a -= b          # Perform subtraction and update a
print(a)        # Print the updated value of a
print(a==b)
a %= b          # Perform modulus operation and update a
print(a)        # Print the updated value of a
print(a==b)
a //= b         # Perform integer division and update a
print(a)        # Print the updated value of a
```

Output

- The error occurs in the subsequent lines because you cannot use assignment operators directly within a print() statement. Example :- A=2,b=3

print(a+=b)----------→ error

Operators in Python

4. Bitwise Operators

The two operands' values are processed bit by bit by the bitwise operators. The examples of Bitwise operators are bitwise OR (|), bitwise AND (&), bitwise XOR (^), negation (~), Left shift (<<), and Right shift (>>).

```
a=5
b=2
print(a&b)
print(a|b)
print(a^b)
print(~b)
print(a>>2)
print(a<<2)
```

- The result of **AND** is 1 only if both bits are 1.
- The result of **OR** is 1 if any of the two bits is 1.
- The result of **XOR** is 1 if the two bits are different.
- **shift (<<)** takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.
- **Right shift (>>)** takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.

Output

- **Bitwise NOT (~)** is the only unary bitwise operator. Since take only one operand. **X=-X-1**

Operators in Python

5. Logical Operators

The assessment of expressions to make decisions typically uses logical operators

and	The condition will also be true if the expression is true. If the two expressions a and b are the same, then a and b must both be true.
or	The condition will be true if one of the phrases is true. If a and b are the two expressions, then an or b must be true if and is true and b is false.
not	If an expression **a** is true, then not (a) will be false and vice versa.

Example

firstprogram python.py × desu.py

C: > Users > Desu > Desktop > gg > firstprogram python.py > ...

```
a = 5            # initialize the value of a
print('Is this statement true?:',a > 3 and a < 5)
print('Any one statement is true?:',a > 3 or a < 5)
print(not(a > 3 and a < 5))
```

Output

Operators in Python

6. Membership Operators

- The membership of a value inside a Python data structure can be verified using Python membership operators.
- The result is true if the value is in the data structure; otherwise, it returns false.

Operator	Description
in	If the first operand cannot be found in the second operand, it is evaluated to be true (list, tuple, or dictionary).
not in	If the first operand is not present in the second operand, the evaluation is true (list, tuple, or dictionary).

Example

```
x = ["Rose", "Lotus"]
print(' Is value Present?', "Rose" in x)
print(' Is value not Present?', "Riya" not in x)
```

Output

Operators in Python

7. Identity Operators

Operator	Description
is	If the references on both sides point to the same object, it is determined to be true.
is not	If the references on both sides do not point at the same object, it is determined to be true.

Example

```
a=6
b=5
c=a
print(a is c)
print(a is not c)
print(a is b)
print(a is not b)
print(a==b)
print(a!=b)
```

```
a = ["Rose", "Lotus"]
b = ["Rose", "Lotus"]
c = a
print(a is c)
print(a is not c)
print(a is b)
print(a is not b)
print(a == b)
print(a != b)
```

Output

Precedence and Associativity of Operators in Python

1	*,, /, // , %	Multiplication, matrix, division, floor division, remainder	Left to right
2	+,-	Addition and subtraction	Left to right
3	<<, >>	Shifts	Left to right
4	&	Bitwise AND	Left to right
5	^	Bitwise XOR	Left to right
6	|	Bitwise OR	Left to right

Example:- 100 + 200 / 10 - 3 * 10

Errors in Python

- In any programming language errors is common. If we miss grammar mistakes of the programming, when your code doesn't produce the expected output.

- Errors in Python can be broadly classified into three categories:- **Syntax Errors, Runtime Errors,** and **Logical Errors.**

1. **Syntax errors :-** are the grammar mistakes of the programming world. They occur when you write code that doesn't conform to Python's syntactical rules.

- Forgetting a colon at the end of an if statement.
- Mismatched parentheses, like print("Hello world".
- Mixing single and double quotes improperly in a string.
- Using Python keywords as variable names.

Example

```
print("Hello, Python!"
```

Errors in Python

2. Runtime Errors

- While syntax errors occur during the parsing or interpretation phase of your code, runtime errors happen during the execution phase. Errors can be more challenging because we don't prevent our program from running
- Occurs when you try to divide a number by zero.
- Occurs when you try to access an index in a list, tuple, or string that doesn't exist.
- Occurs when you use a variable or function that hasn't been defined.
- Occurs when you perform an operation on a data type that doesn't support it.
- Occurs when you try to open or manipulate a file that doesn't exist.

Example

```
num1 = 10
num2 = 0
result = num1 / num2
print(result)
```

Errors in Python

3. Logical Errors

- Logical errors, also known as semantic errors, occur when your code doesn't produce the expected output because of a flaw in the algorithm or the overall logic of your program.
- Unlike syntax or runtime errors, logical errors don't generate error messages or exceptions. Instead, your code runs without issues, but it doesn't achieve the intended results.

Example

```
def calculate_average(numbers):
    total = 0
    for num in numbers:
        total += num
    average = total / len(numbers) - 1
    return averag
```

Control Statements

- Python's conditional statements carry out various calculations or operations according to whether a particular Boolean condition is evaluated as true or false.
- Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions.

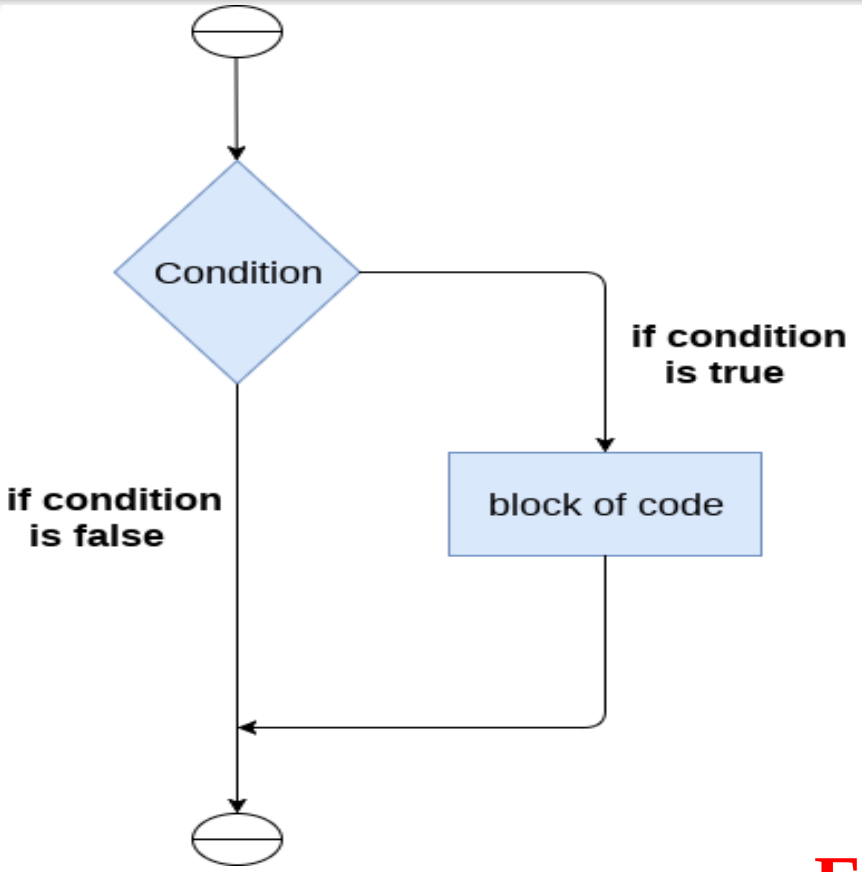
Statement	Description
If	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else	The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked.
Nested if	Nested if statements enable us to use if ? else statement inside an outer if statement.
elif	The elif statement enables us to check multiple conditions and execute the specific

Indentation in python

Indentation in Python

- For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block.
- If two statements are at the same indentation level, then they are the part of the same block.
- Generally, four spaces are given to indent the statements which are a typical amount of indentation in python.
- Indentation is the most used part of the python language since it declares the block of code.
- All the statements of one block are intended at the same level indentation.

The If statement



The syntax of the if-statement is given below.

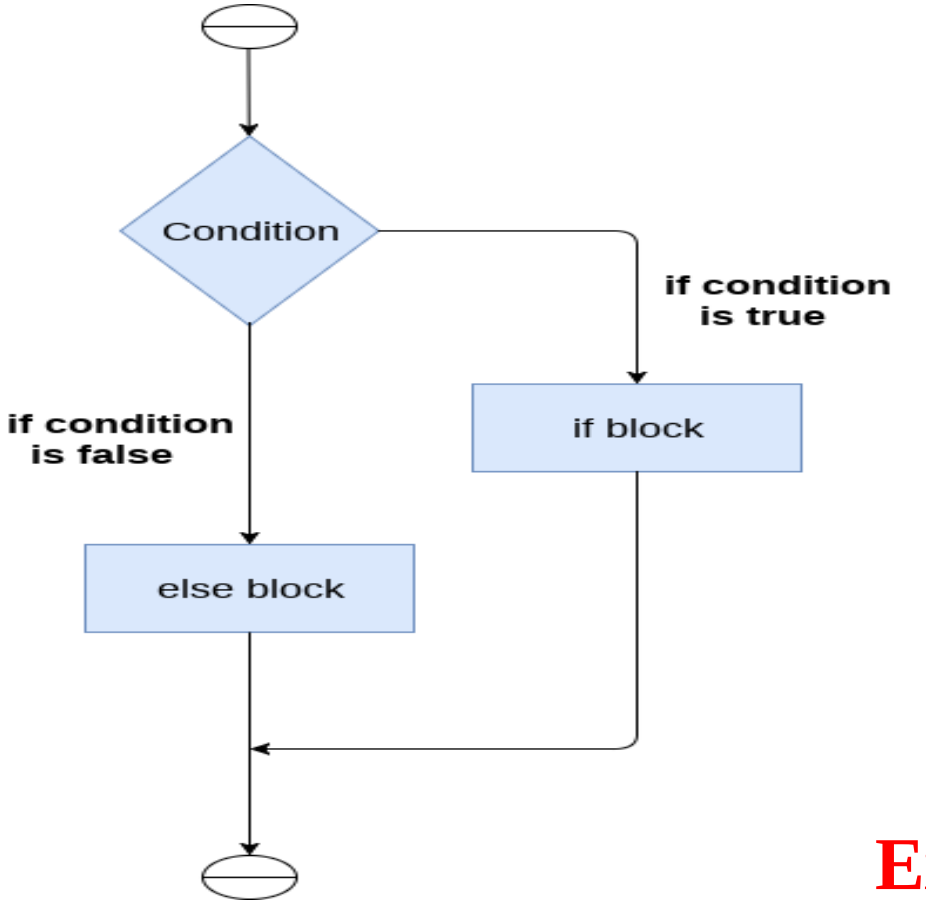
```
if expression:
    statement
if expression:
    statement
```

Exercise

1. Write a program to check the given number is positive using python?

2. Write a Simple Python Program to print the largest of the three numbers?

The if-else statement



The syntax of the if-else statement is given below.

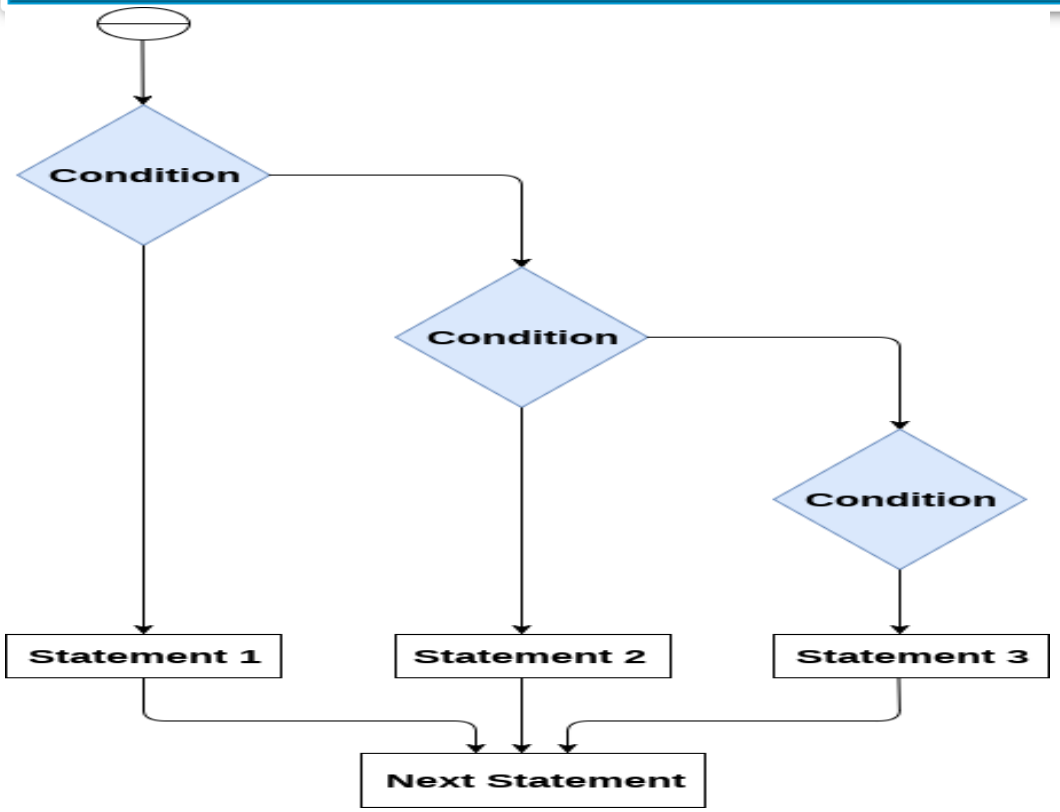
```
if condition:
    block of statements
else:
    else-block
```

Exercise

1. Write a program to check the given number is positive or negative using python?

2. Write a Program to check whether a number is even or not using python.

The elif statement



The syntax of the if-else statement is given below.

```
if expression 1:
    block of statements
elif expression 2:
    block of statements
elif expression 3:
    block of statements
    else:
    block of statements
```

Exercise

1. Write a python program to calculate the Grade letter of a given course by adding mid exam and final exam result based DBU grade scale using **elif** ?
2. Write a python program to find the largest number from three number using **nested if ?**

Looping in Python

- Python loops allow us to execute a statement or group of statements multiple times
- Python programming language provides the following types of loops to handle looping requirements.

Sr.No.	Name of the loop	Loop Type & Description
1	**While loop**	➢ Repeats a statement or group of statements while a given condition is TRUE. ➢ It tests the condition before executing the loop body.
2	**For loop**	➢ This type of loop executes a code block multiple times and abbreviates the code that manages the loop variable.
3	**Nested loops**	➢ We can iterate a loop inside another loop

Python for loop

- Python frequently uses the Loop to iterate over iterable objects like **lists, tuples, and strings.**
- **for loops** are used when a section of code needs to be repeated a certain number of times.

syntax

```
for value in range():
    statement
```

syntax

```
for value in sequence:
    statement
```

- Passing the whitespace to the end parameter (end=' ') indicates that the end character has to be identified by **whitespace and not a newline.**

Example:- write a python program to print a number from 1to 10 using for loop

```
for i in range(11):
    print(i,end=" ")
```

```
for i in range(1,11):
    print(i,end=" ")
```

Or

```
for i in range(11):
    print(i)
```

Or

```
for i in range(1,11):
    print(i)
```

Python for loop

1. Write the output of the following python program? **Output**

```
for i in range(1,10,2):
    print(i,end=" ")
```

```
for i in range(2,10,2):
    print(i,end=" ")
```

2. Write a python program to display the sum of the first 100 number?

```
sum=0
for i in range(1,101,1):
    sum=sum+i
print("the sum is",sum)
```

3. Write a program a python program to calculate the factorial of the given number ?

For loop using list

- Write a python program to find the sum of the square each element of the list using for loop? numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]

syntax

```
for value in sequence:
    statement
```

Since instead of range just replace numbers

Output

Solution:-

```
numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]
sum = 0
for num in numbers:
    sum = sum + num ** 2
print("The sum of squares is: ", sum)
```

Python while loop

➢The Python while loop iteration of a code block is executed as long as the given Condition, i.e., conditional expression, is true.

The syntax
Statement
while Condition:
Statement

1. Write a python program to display a number from 1 to 10?

```
i=1
while i<=10:
    print(i)
    i=i+1
```

2. Write a python program to find the sum of n natural number ?

```
i=1
sum=0
while (i<=10):
    sum=sum+i
    i=i+1
print(" the sum is",sum)
```

Python while loop

Exercise

1. Write a Python for checking a number is Prime number or not
2. Write a python program to find the sum of n natural number ?
3. Write a python program to calculate factorial of a number
4. Write the output of the following program

```
i=1
while i<51:
    if i%5 == 0 or i%7==0 :
        print(i, end=' ')
    i+=1
```

Output

Python while loop using List

Example:- Write a python program to find the sum of the given list using for loop?
numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]

Len() function

- It's important to note that the indexing of elements in a list starts from 0, so the length of a list is always greater than or equal to 0.
- The len() function is efficient and commonly used for iterating over lists or performing operations based on the size of a list

```
numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]
sum = 0
i=0
while i < len(numbers):
    sum =sum+ numbers[i]
    i += 1
print("Sum  of each element:", sum)
```

Looping in Python

Python Loop Control Statements

- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- Python supports the following control statements

Sr.No.	Control Statement & Description
1	**Break statement:-** Terminates the loop statement and transfers execution to the statement immediately following the loop.
2	**Continue statement :-** Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	**Pass statement :-** The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Python break statement

Break is commonly used to break the loop for a given condition. Syntax

```
Loop statements
  break;
```

❑Write the output of the following program??

```
for i in range(1,10,2):
    while i==8:
        break
    print(i)
```

```
my_str = "python"
for char in my_str:
    if char == 'o':
        break
    print(char)
```

```
for i in range(1,10,2):
    while i==7:
     print(i)
     break
```

```
numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]
for char in numbers:
    if char == 1:
        break
    print(char)
```

Python continue statement

- We can use the pass statement as a placeholder when unsure of the code to provide. Therefore, the pass only needs to be placed on that line.

- Write the output of the following program??

Loop statements
continue;

```
for i in range(1,10,2):
    while i==8:
        continue
    print(i)
```

```
my_str = "python"
for char in my_str:
    if char == 'o':
        continue
    print(char)
```

```
for i in range(1,10,2):
    while i==7:
     print(i)
     continue
```

```
numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]
for char in numbers:
    if char == 1:
        continue
    print(char)
```

Python pass statement

Python continue keyword is used to skip the remaining statements of the current loop and go to the next iteration

```
Loop statements
  pass;
```

❑Write the output of the following program??

```
sequence = {"Python", "program",
"Statement", 22,10,"Placeholder“,40}
for value in sequence:
    if value == "program":
        pass
    else:
        print( value)
```

```
sequence =
[4,7,21,7,90,54]
for value in sequence:
    if value == 7:
        pass
    else:
        print(value)
```

Python nested loop

➢In python nested loop is a loop inside another loop. Syntax

```
Outer loop
    inner loop
        statement of inner loop
    statement of outer loop
```

```
for loop:
        for loop:
                statements of for loop
        statements of for loop
```

```
while loop:
        while loop:
                statements of while loop
        statements of while loop
```

```
for loop:
        while loop:
                statements of while loop
        statements of for loop
```

```
while loop:
        for loop:
                statements of for loop
        statements of while loop
```

Python nested loop

Example:- Write the output of the following python program

```
for i in range (1,8):
    for j in range(i):
        print("*",end=" ")
    print(" ")
```

```
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
```

```
for i in range (1,8):
    for j in range(8-i):
        print("*",end=" ")
    print(" ")
```

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
```

Python Function

Function:- A collection of related assertions that carry out a mathematical, analytical, or evaluative operation is known as **a function**

- Python functions are necessary for intermediate-level programming and are easy to define.
- Function names meet the same standards as variable names do.
- The objective is to define a function and group-specific frequently performed actions.
- Instead of repeatedly creating the same code block for various input variables, We can call the function and reuse the code it contains with different variables.

Python Function

Advantages of Python Functions

- We can stop a program from repeatedly using the same code block by including functions.
- Once defined, Python functions can be called multiple times and from any location in a program.
- Our Python program can be broken up into numerous, easy-to-follow functions if it is significant.
- The ability to return as many outputs as we want using a variety of arguments is one of Python's most significant achievements.
- However, Python programs have always incurred overhead when calling functions.

Python Function

Python Functions Declaration

```
1.def function_name( parameters ):
    # code block
```

- The start of a capability header is shown by a catchphrase called **def**.
- function_name is the function's name, which we can use to distinguish it from other functions.
- We will utilize this name to call the capability later in the program. Name functions in Python must adhere to the same guidelines as naming variables.
- Using parameters, we provide the defined function with arguments.
- A colon (:) marks the function header's end.

Python Function

1. write a python program to calculate the sum of two number using function

```
def sum(num, num1):
    return num+num1
x=int(input("Enter any number"))
y=int(input("Enter any number"))
addition = sum(x,y)
print( "The square of the given number is: ", addition )
```

2. Write a python program to square the given number?

3. Write a program to perform arithmetic operation using function?

4. Write a python program to calculate the Grade letter of a given course by adding mid exam and final exam result based DBU grade scale using **function?**

Python Function

1. What is the output of the following program?

```
# Example Python Code for calling a function
# Defining a function
def a_function( string ):
    "This prints the value of length of string"
    return len(string)
# Calling the function we defined
print( "Length of the string Functions is: ", a_function( "Functions" ) )
print( "Length of the string Python is: ", a_function( "Python" ) )
```

Output

Variable Scopes

- When you want to use the same variable for rest of your program or module you declare it as a global variable,
- while if you want to use the variable in a specific function or method, you use a local variable.

❑ Let's understand this Python variable types with the difference between local and global variables in the below program.

1. Let us define variable in Python where the variable "f" is **global** in scope and is assigned value 101 which is printed in output
2. Variable f is again declared in function and assumes **local** scope.

- It is assigned value "I am learning Python." which is printed out as an output. This Python declare variable is different from the global variable "f" defined earlier

Local & Global Variable

3. Once the function call is over, the local variable f is destroyed. At line 12, when we again, print the value of "f" is it displays the value of global variable f=101

Python5.2.py

```
1  # Declare a variable and initialize it
2  f = 101
3  print(f)
4
5  # Global vs. local variables in functions
6  def someFunction():
7  # global f
8      f = 'I am learning Python'
9      print(f)
10
11 someFunction()
12 print(f)
13
```

f is a local variable declared inside the function.

Run Python5.2

```
"C:\Users\DK\Desktop\Python code\Python Test\Python 5\PythonCode
 5/PythonCode5/Python5.2.py"
101
I am learning Python
101
```

Local & Global Variable

- While Python variable declaration using the keyword **global,** you can reference the global variable inside a function.

1. Variable "f" is **global** in scope and is assigned value 101 which is printed
2. Variable f is declared using the keyword **global**. This is **NOT** a **local variable**, but the same global variable declared earlier. Hence when we print its value, the output is 101
3. We changed the value of "f" inside the function. Once the function call is over, the changed value of the variable "f" persists

Local & Global Variable

- At line 12, when we again, print the value of "f" is it displays the value "changing global variable"

```
1   f = 101;
2   print(f)
3
4   # Global vs.local variables in functions
5   def someFunction():
6       global f
7       print(f)
8       f = "changing global variable"
9
10  someFunction()
11  print(f)
12
13
14
```

1 2 3

We are now accessing and changing the global variable f.

someFunction()

Run Python5.3

```
"C:\Users\DK\Desktop\Python code\Python Test\Python 5\Pytho
 5/PythonCode5/Python5.3.py"
101
101
changing global variable
```

Parameter Passing