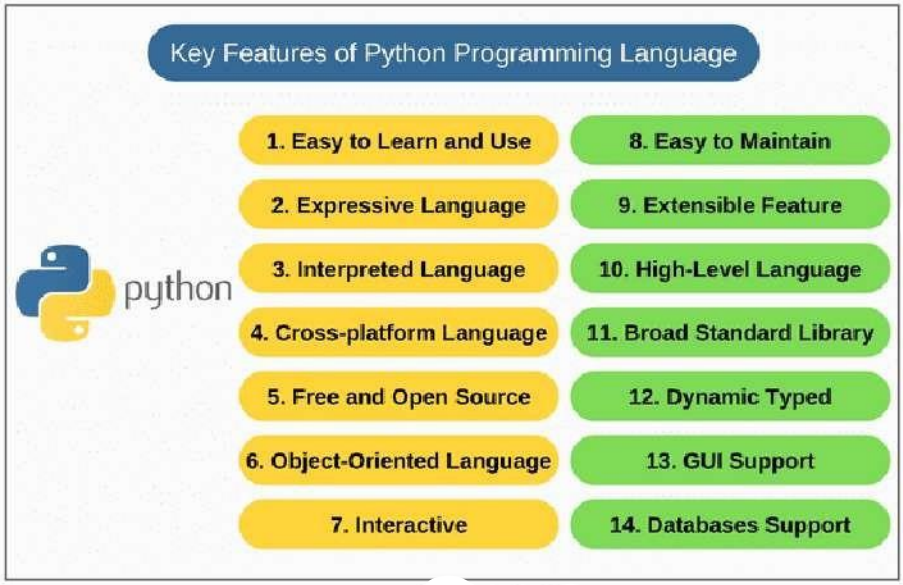
Introduction to Python:

Python is a high-level, interpreted, object-oriented programming language that is widely used for developing various types of applications including web development, data science, artificial intelligence, and automation.

It was created by **Guido van Rossum** in the late 1980s.

Features of Python:



1. **Simple and easy-to-learn syntax:** Python has a simple and readable syntax, making it easy to learn and use, especially for beginners.

2. **Interpreted language:** Python is an interpreted language, which means that it does not require compilation before execution, making it faster and easier to debug.

3. **Object-oriented programming:** Python supports object-oriented programming, allowing developers to create reusable and modular code.

4. **Dynamic typing:** Python is dynamically typed, which means that variable types are determined at runtime, allowing for more flexible and adaptable code.

5. **Extensive standard library:** Python comes with a large standard library that provides a wide range of tools and modules for various tasks, from handling regular expressions to network programming.

6. **Cross-platform:** Python code can be run on multiple operating systems, including Windows, Linux, and macOS, making it a versatile language for developing applications.

7. **Large and active community:** Python has a large and active community of developers who contribute to open-source projects and provide support to beginners.

8. **Third-party packages: Python** has a vast collection of third-party packages available on the Python Package Index (PyPI) that can be easily installed using the pip package manager, making it easy to add functionality to your code.

9. **High-level language:** Python is a high-level language, meaning it provides abstractions that simplify complex tasks and make it easier to write code.

10. **Integration capabilities:** Python can easily integrate with other languages and technologies, including C/C++, Java, and .NET, making it a popular choice for developing applications with diverse technology stacks.

Applications of Python:

1. **Web development:** Python is widely used for web development, with popular web frameworks such as Django and Flask that allow developers to create dynamic and scalable web applications.

2. **Data science**: Python is a popular language for data science and machine learning, with libraries such as NumPy, Pandas, and Scikit-learn that provide powerful tools for data analysis and machine learning.

3. **Scientific computing**: Python is widely used in scientific computing, with libraries such as SciPy, Matplotlib, and IPython that provide tools for scientific and engineering calculations.

4. **Artificial intelligence and robotics:** Python is used in artificial intelligence and robotics, with libraries such as TensorFlow, Keras, and PyTorch that provide powerful tools for deep learning and artificial intelligence.

5. **Desktop applications:** Python can be used to create desktop applications using tools such as PyQt, wxPython, and PyGTK.

6. **Game development**: Python can be used in game development, with libraries such as Pygame that provide tools for creating 2D games.

7. **Automation:** Python can be used for automation tasks, with tools such as Selenium and Beautiful Soup that provide web scraping and automation capabilities.

8. **Scripting:** Python is widely used for scripting tasks, with tools such as Fabric and Ansible that provide automation capabilities for system administration and network management.

Python Versions:

The current stable version of Python as of my knowledge cutoff date (September 2021) is Python 3.11.2, which was released on October 4, 2021. Some of the major versions of Python include:

1. **Python 1.x:** The first version of Python, which was released in 1991.

2. **Python 2.x:** The second major version of Python, which was released in 2000. It was widely used for many years, but its development was discontinued on January 1, 2020.

3. **Python 3.x:** The third major version of Python, which was released in 2008. It introduced many new features and improvements, but also introduced some backward-incompatible changes. Python 3.x is the current and recommended version of Python for new development.

Each major version of Python has multiple minor versions, which include bug fixes, improvements, and new features. For example, Python 3.7, Python 3.8, Python 3.9, and Python 3.10 are all minor versions of Python 3.x, with each version introducing new features and improvements.

Installation of Python:

To install Python on your computer, you can follow these general steps:

1. Go to the official Python website at python.org and download the latest version of Python that is compatible with your operating system (Windows, Mac, or Linux).
   Link : https://www.python.org/downloads/

2. Run the downloaded installation file and follow the installation wizard. Make sure to select the options you want, such as adding Python to your system path or installing third-party libraries.

3. Once the installation is complete, open a command prompt (Windows) or terminal (Mac/Linux) and type python to start the Python interpreter. If Python is installed correctly, you should see the Python prompt (>>>) in the command prompt/terminal window.

You can also use a text editor or integrated development environment (IDE) to write and run Python code. Popular text editors and IDEs for Python include Visual Studio Code, PyCharm, Thonny, Python IDLE, Atom, and Sublime Text.

IDE: Python Command Line mode and Python IDEs;

Default IDEs:

Open cmd -> Type py or Python
Open IDLE(Python 3.11)

Other IDEs ->

Thonny (Link: https://github.com/thonny/thonny/releases/tag/v4.0.2)
Pycharm
Visual Studio Code
SublimeText

Simple Python Program.

An example of a simple Python program that you can create and run using the Thonny IDE:

1. Open Thonny IDE and create a new file by clicking on 'File' > 'New' or using the keyboard shortcut Ctrl+N (Windows/Linux) or Command+N (Mac).

2. Type the following code into the new file:

```
# This is a simple Python program
print('Hello, world!')
```

3. Save the file by clicking on 'File' > 'Save' or using the keyboard shortcut Ctrl+S (Windows/Linux) or Command+S (Mac). Choose a filename and location on your computer to save the file.

4. Run the program by clicking on 'Run' > 'Run current script' or using the keyboard shortcut F5. This will open a new window called 'Shell' at the bottom of the IDE and display the output of the program, which should be 'Hello, world!'.

Python Basics: some of the basic concepts we must practice are,

1. Identifiers
2. Keywords
3. Statements and Expressions
4. Variables
5. Operators
6. Precedence and Association
7. Data Types
8. Indentation
9. Comments

1. Identifiers: Identifiers in Python are names given to variables, functions, classes, modules, and other objects in the code. An identifier is a sequence of letters, digits, and underscores (_) that begins with a letter or underscore.

Rules for naming identifiers in Python:

1. An identifier must start with a letter (a to z, A to Z) or underscore (_) character.
2. The second character onwards, it can have letters (a to z. A to Z), digits (0 to 9), or underscore (_) character.
3. Python identifiers are case-sensitive. For example, my_var and My_Var are two different identifiers.
4. There are some reserved words that cannot be used as identifiers, such as if, else, while, for, and, or, not, class, def, import, from, in, is, try, except, raise, with, and yield.
5. Identifiers can be of any length, but it is recommended to keep them short and meaningful.
6. Avoid using special characters such as !, @, #, $, %, ^, &, *, (,), -, +, =, {, }, [,], |, \, :, ;, ', ", <, >, ,, ., ?, / in identifiers.

Validation ->

Keywords: Keywords in Python are reserved words that have a special meaning, functionality and purpose in the language.

They are used to define the syntax and structure of the code and cannot be used as variable names or identifiers. Python has 35 keywords, as of version 3.10:

```
and       as        assert    async     await
break     class     continue  def       del
elif      else      except    False     finally
for       from      global    if        import
in        is        lambda    None      nonlocal
not       or        pass      raise     return
True      try       while     with      yield
```

Statements: a statement is a unit of code that performs a specific action. A program is made up of one or more statements. A statement can be a single line of code or a group of lines of code that perform a specific task.

1. **Assignment statements:** Assign a value to a variable. For example, x = 5.
2. **Conditional statements:** Control the flow of execution based on a condition. For example,

```
if x > 0:
    print("x is positive").
```

3. **Looping statements:** Repeat a block of code while a condition is true. For example,

```
while x > 0:
    print(x);
    x -= 1.
```

4. **Function and method calls:** Call a function or method to perform a specific task. For example,

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

5. **Import statements:** Import modules or specific functions from modules. For example,

```
import math or from random import randint.
```

Expressions: an expression is a combination of values, variables, operators, and function calls that produces a result.

An expression can be as simple as a single value or as complex as a multi-level function call that uses variables and operators to manipulate data.

Here are some examples of expressions in Python:

1. Numeric expressions: 2 + 3, 5 * 6, 10 / 2, etc.
2. String expressions: "Hello, " + "World!", "Python" * 3, etc.
3. Boolean expressions: 5 > 3, x == 10, etc.
4. Variable expressions: x, y + 5, z * (x + y), etc.
5. Function call expressions: len("Hello"), math.sqrt(25), etc.

In Python, expressions are evaluated to produce a value. The value of an expression can be stored in a variable or used in other expressions or statements.

Variables: a variable is a named location in memory that stores a value. The value of a variable can be changed during the execution of a program. Variables can be used to store different types of data, including numbers, strings, and boolean values.

To create a variable in Python, you simply choose a name for the variable and use the assignment operator (=) to assign a value to it. Here are some examples:

```
x = 5       # x is assigned the value 5
y = "Hello"  # y is assigned the string value "Hello"
z = True     # z is assigned the boolean value True
```

Operators: Operators in Python are symbols or special keywords that perform operations on values or variables. Python supports a wide range of operators, including

1. Arithmetic operators
2. Comparison operators
3. Logical operators
4. Bitwise operators
5. Assignment operators.

1. **Arithmetic operators**: + (addition), - (subtraction), * (multiplication), / (division), % (modulus), ** (exponentiation), // (floor division)
2. **Comparison operators**: == (equal to), != (not equal to), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to)
3. **Logical operators: and, or, not**
4. **Bitwise operators: & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), ~ (bitwise NOT), << (left shift), >> (right shift)**
5. **Assignment operators:**
   = (simple assignment),
   += (addition assignment),
   -= (subtraction assignment),
   *= (multiplication assignment),
   /= (division assignment),
   %= (modulus assignment),
   **= (exponentiation assignment),
   //= (floor division assignment)

Example code:

```
x = 10
y = 3

# Arithmetic operators
print(x + y)  # outputs 13
print(x - y)  # outputs 7
print(x * y)  # outputs 30
print(x / y)  # outputs 3.3333333333333335
print(x % y)  # outputs 1
print(x ** y) # outputs 1000
print(x // y) # outputs 3

# Comparison operators
print(x == y)  # outputs False
print(x != y)  # outputs True
print(x < y)   # outputs False
print(x > y)   # outputs True
print(x <= y)  # outputs False
print(x >= y)  # outputs True

# Logical operators
a = True
b = False
print(a and b)  # outputs False
print(a or b)   # outputs True
print(not a)    # outputs False

# Bitwise operators
m = 60     # 60 = 0011 1100
n = 13     # 13 = 0000 1101
print(m & n)  # outputs 12  (12 = 0000 1100)
print(m | n)  # outputs 61  (61 = 0011 1101)
print(m ^ n)  # outputs 49  (49 = 0011 0001)
print(~m)     # outputs -61 (-61 = 1100 0011)
print(m << 2) # outputs 240 (240 = 1111 0000)
print(m >> 2) # outputs 15  (15  = 0000 1111)

# Assignment operators
x += 5
```

```
y *= 2
print(x)  # outputs 15
print(y)  # outputs 6
```

Precedence and Association:

Precedence refers to the order in which operators are evaluated. Operators with higher precedence are evaluated first, while operators with lower precedence are evaluated later.

Ex:
multiplication (*) has a higher precedence than addition (+), so in the expression 3 + 4 * 5, the multiplication is performed first, giving 3 + 20, which evaluates to 23.

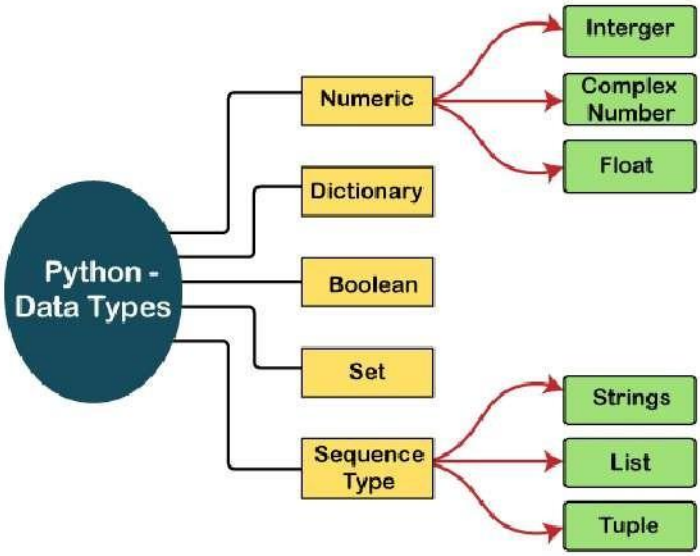
Here's a table of the operator precedence in Python, from highest to lowest:

Operator	Description
`**`	Exponentiation
`~`	Bitwise NOT
`+`, `-`	Positive, negative
`*`, `/`, `%`, `//`	Multiplication, division, modulus, floor division
`+`, `-`	Addition, subtraction
`>>`, `<<`	Bitwise right shift, bitwise left shift
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	`|`
`==`, `!=`, `<`, `>`, `<=`, `>=`, `in`, `not in`, `is`, `is not`	Comparison, membership, identity
`not`	Logical NOT

Association refers to the order in which operators with the same precedence are evaluated. Operators can be left-associative (evaluated from left to right) or right-associative (evaluated from right to left).

For example, the addition operator (+) is left-associative, so in the expression 3 + 4 + 5, the leftmost + is evaluated first, giving (3 + 4) + 5, which evaluates to 12.

Data Types: Python has several built-in data types that are used to represent different kinds of data.



1. Numeric : Numeric data types:

int: integers, for example 42 or -3.
float: floating-point numbers, for example 3.14 or -0.5.
complex: complex numbers, for example 1 + 2j or -3j.

2. Dictionary : Mapping data type: dict
dict: unordered collection of key-value pairs, for example {'name': 'John', 'age': 30}.

3. Boolean : bool
bool: represents truth values, either True or False.

4. Set: set
set: unordered collection of unique items, for example {1, 2, 3} or {'apple', 'banana', 'cherry'}.

5. Sequence Type:

list: ordered and mutable collection of items, for example [1, 2, 3] or ['apple', 'banana', 'cherry'].
tuple: ordered and immutable collection of items, for example (1, 2, 3) or ('apple', 'banana', 'cherry').
range: immutable sequence of numbers, for example range(0, 10, 2) represents the sequence 0, 2, 4, 6, 8.
String:
str: a string of characters, for example "Hello, world!" or 'Python is fun'.

Indentation: In Python, indentation is used to indicate a block of code. Python does not use curly braces or other symbols to indicate the beginning and end of a block of code, instead it uses indentation to group statements into a block.

The standard indentation in Python is four spaces.

Comments: In Python, comments are used to add explanatory or informative notes to your code. Comments in Python are not executed by the interpreter, and tney are ignored when the program runs.

To add a comment in Python, you can use **the # symbol**. Everything that comes after the # symbol on a line is considered a comment and will be ignored by the interpreter. For example:

```
# This is a comment in Python
print("Hello, world!") # This is also a comment
```

single-line comments start with the # symbol and are used to provide brief explanations of the code. They are often used to make the code more readable and understandable to other programmers who may be working on the code. Single-line comments can appear on their own line or at the end of a line or code.

Multi-line comments in Python are also known as **docstrings.** They are used to provide more detailed explanations of the code and are typically used for documenting functions, classes, and modules. Multi-line comments start and end with three double quotes (""") and can span multiple lines.

Ex:

```
"""
This is a multi-line comment.
It can span multiple lines and is often used to
provide detailed explanations of functions, classes, and modules.
"""
```

Functions: A function is a block of reusable code that performs a specific task. Functions help to make your code more modular, reusable, and easier to read and maintain.

To define a function in Python, you use the **def** keyword followed by the name of the function, the input parameters (if any), and a colon (:). The body of the function is then indented, and can contain any valid Python code.

Ex:

```
def add_numbers(a, b):
    result = a + b
    return result
```

Built-in Functions:

Python comes with several built-in functions that are available for use without the need to import any external libraries. These functions are built into the Python interpreter and can be used to perform a wide range of common tasks. Here are some examples of built-in functions in Python:

print(): This function is used to display output to the console.

input(): This function is used to get input from the user through the console.

len(): This function is used to get the length of a sequence such as a string, list, or tuple.

type(): This function is used to get the type of a Python object.

int(), float(), and str(): These functions are used to convert objects to integers, floating-point numbers, and strings, respectively.

range(): This function is used to generate a sequence of numbers.

max() and min(): These functions are used to get the maximum and minimum values in a sequence, respectively.

sum(): This function is used to get the sum of all the elements in a sequence.

sorted(): This function is used to sort a sequence.

abs(): This function is used to get the absolute value of a number.

Console Input and Console Output:

Console input and output are essential aspects of many Python programs. They allow users to interact with the program and provide input, and they allow the program to display output to the user. Here's how to perform console input and output in Python:

Console Output:

To display output to the console in Python, you can use the print() function. The syntax for print() is as follows:

print(value1, value2, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Here, value1, value2, etc. are the values that you want to display. You can pass any number of values to the print() function, separated by commas. The sep argument is used to specify the separator between the values (default is a single space). The end argument is used to specify what to append at the end of the output (default is a newline character).

Examples:

```
# Multiple values example
a = 10
b = 20
print("a =", a, "b =", b, sep=' | ', end='\n\n')
print("The sum of a and b is:", a + b)
```

Console Input:

To get input from the console in Python, you can use the input() function. The syntax for input() is as follows:

input([prompt])

Here, prompt is an optional string that is displayed to the user before input is requested. When the input() function is called, the user will be prompted to enter a value, and that value will be returned by the function.

Example:

```
# Simple console input example
name = input("Enter your name: ")
print("Hello, " + name + "!")
```

Type Conversions: Conversion of one data type to another using type conversion functions. Here are some common type conversion functions in Python:

int(): This function is used to convert a value to an integer.

float(): This function is used to convert a value to a floating-point number.

str(): This function is used to convert a value to a string.

bool(): This function is used to convert a value to a boolean (True or False) value.

Example:

```
# converting to integer
num_str = "10"
num_int = int(num_str)
print(type(num_int))  # Output: <class 'int'>

# converting to float
num_str = "10.5"
num_float = float(num_str)
print(type(num_float))  # Output: <class 'float'>

# converting to string
num_int = 10
num_str = str(num_int)
print(type(num_str))  # Output: <class 'str'>

# converting to boolean
num = 10
bool_val = bool(num)
print(bool_val)  # Output: True
```

Python Libraries: Python libraries are collections of pre-written code that provide a set of functions and tools that you can use in your Python programs. Python has a large and active community of developers, which has led to the creation of a vast number of libraries covering a wide range of applications. Here are some of the most popular Python libraries:

NumPy: NumPy is a library for working with large, multi-dimensional arrays and matrices, and provides a wide range of mathematical functions to operate on them.

Pandas: Pandas is a library for data analysis and manipulation, providing tools for handling tabular data, time series, and more.

Matplotlib: Matplotlib is a library for creating static, animated, and interactive visualizations in Python, and provides a wide range of customizable plot types.

SQLIte: for interaction with SQLite databases.

Importing Libraries with Examples.

In Python, libraries are collections of pre-existing code that allow you to perform specific tasks without having to write the code from scratch. Here are some examples of how to import commonly used libraries in Python:

NumPy: NumPy is a library used for numerical operations and scientific computing. You can import it with the following command:

Example:

```
import numpy as np
# Creating an array
my_array = np.array([1,2,3,4])
print("Arra Using numpy : ",my_array)
```

Python Control Flow: Python control flow allows you to control the order of execution of the code based on certain conditions or events. There are several ways to implement control flow in Python:

Types of Control Flow;

Branching statements/Control Flow Statements/conditional statements

1. if: The if statement is used to check if a condition is true, and execute a block of code if the condition is true. Here's an example:

```
x = 5
if x > 0:
   print("x is positive")   #output x is positive
```

2. if else: The if else statement is used to execute one block of code if a condition is true, and another block of code if the condition is false. Here's an example:

Ex:

```
x = 10
if x > 5:
   print("x is greater than 5")
else:
   print("x is less than or equal to 5")
```

3. if elif else: elif ladder:

In Python, the elif ladder is a construct that allows you to chain multiple if and else statements together. The elif statement is short for "else if," and it allows you to check for additional conditions after the initial if statement.

Using an elif ladder can be a more efficient way of testing multiple conditions than using separate if statements for each condition.

Ex:

```
x = 10
if x > 20:
   print("x is greater than 20")
elif x > 15:
   print("x is greater than 15 but less than or equal to 20")
elif x > 10:
   print("x is greater than 10 but less than or equal to 15")
else:
```

```
print("x is less than or equal to 10")
```

Iterations/Looping: iterations or loops are used to execute a block of code repeatedly. There are two main types of loops in Python: for loops and while loops.

4.while loop: A while loop is used to execute a block of code repeatedly as long as a certain condition is true.

Syntax:

```
while condition:
    # code block to be executed
```

Ex:

```
count = 0
while count < 10:
    print(count)
    count += 1
```

5. for loop: The for loop in Python is used to iterate over a sequence (such as a list, tuple, or string) and execute a block of code for each item in the sequence. The basic syntax of a for loop is as follows:

Syntax:

```
for variable in sequence/range():
    # code block to be executed
```

Ex:

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

6.break: the break statement is used to immediately exit a loop, regardless of whether the loop's condition has been satisfied or not. When the break statement is executed inside a loop, the loop is terminated and the program continues executing the code after the loop. **Syntax:**

```
while condition:
    # code block
    if some_condition:
        break
    # more code
```

Ex:

```
i = 0
while i < 10:
  print(i)
  i += 1
  if i == 5:
    break
```

Output:

7. continue statements: the continue statement is used to skip over the current iteration of a loop and move on to the next iteration. When the continue statement is executed inside a loop, the code in the loop's body that follows the continue statement is skipped, and the loop proceeds with the next iteration.

syntax of the continue statement:

```
while condition:
  # code block
  if some_condition:
    continue
  # more code
```

Ex:

```
i = 0
while i < 10:
  i += 1
  if i % 2 == 0:
    continue
  print(i)
```

Output:

range () and exit () functions:

range() : The range() function is used to generate a sequence of numbers. It returns a sequence of numbers starting from 0 (by default) and increments by 1 (by default), and stops before a specified number.

Syntax:

```
range(stop)
range(start, stop, step)
```

Example:

```
for i in range(5):
    print(i)
```

Output:

exit() : The exit() function is used to immediately terminate the Python interpreter. It takes an optional integer argument that can be used to specify an exit status.

Ex:

```
print("Before exit")
exit()
print("After exit")
```