A scripting language is a “wrapper” language that integrates OS functions.
The interpreter is a layer of software logic between your code and the computer hardware on your machine.
scripting languages doesn’t require any compilation and are directly interpreted

Scripting Language Features:
Automation of the required process into a program.
Fetching information from the provided data sets.
Requires less code than modern programming languages

Feature	Programming Language	Scripting Language
Purpose	Create standalone applications and software	Automate tasks, integrate systems, add dynamic behavior to applications
Execution	Compiled (translated to machine code before execution)	Interpreted (executed line-by-line at runtime)
Complexity	More complex, stricter syntax, fine-grained control	Simpler, easier syntax, built-in functions for common tasks
Use Cases	Operating systems, large-scale software, performance-critical applications	Web development, system administration, data analysis, automation, prototyping
Dependencies	Self-contained, can run independently	May require a host environment or runtime
Examples	C, C++, Java, C#, Rust, Go	Python, JavaScript, Perl, Ruby, PHP, Bash

Python was conceptualized by Guido Van Rossum in the late 1980s

Python is a high-level programming language which is:

- Interpreted: Python is processed at runtime by the interpreter.
- Interactive: You can use a Python prompt and interact with the interpreter directly to write your programs.
- Object-Oriented: Python supports Object-Oriented technique of programming.
- Beginner’s Language: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications.
- Python has support for an interactive mode of testing and debugging.
- Python has a broad standard library cross-platform.
- Everything in Python is an Object: variables, functions, even code. Every object has an ID, a type, and a value.
- Python supports GUI applications
- Python supports automatic garbage collection.
- Python can be easily integrated with C, C++, and Java.

Applications:

1. Embedded scripting language: Python is used as an embedded scripting language for various testing/ building/ deployment/ monitoring frameworks, scientific apps, and quick scripts.

2. 3D Software:3D software like Maya uses Python for automating small user tasks, or for doing more complex integration such as talking to databases and asset management systems.

3. Web development: Python is an easily extensible language that provides good integration with database and other web standards.

Difference between Compiler and Interpreter

No	Compiler	Interpreter
1	Compiler Takes **Entire** program as input	Interpreter Takes **Single** instruction as input .
2	Intermediate Object Code is **Generated**	**No** Intermediate Object Code is **Generated**
3	Conditional Control Statements are Executes **faster**	Conditional Control Statements are Executes **slower**
4	**Memory Requirement : More** (Since Object Code is Generated)	**Memory Requirement** is **Less**
5	Program need not be **compiled** every time	Every time higher level program is converted into lower level program
6	**Errors** are displayed after **entire program** is checked	**Errors** are displayed for **every instruction** interpreted (if any)
7	**Example** : C Compiler	**Example** : BASIC

Python is often referred to as an "interpreted" language, but in reality, it is both compiled and interpreted.

When you run a Python script or program, the source code is first translated by the Python interpreter into an intermediate form known as bytecode.

This process is called compilation.

The bytecode is a lower-level representation of the original Python code, and it is platform-independent.

—----------

Functions
To take input from the user we make use of a built-in function input().

We can also typecast this input to integer, float, or string by specifying the input() function inside the type.
num1 = int(input())

The + operator can be used to concatenate (join) strings Together.

' ')

```
msg1="Hello Friends"
msg2="Python is fun"
print(msg1,msg2)
```

```
print(msg1,msg2,sep=',',end='!')
```

```
Hello Friends,Python is fun!
```

print(day,mon,yr,sep="-")
Output:
12-12-2022

Triple quotes ("' ‘’ or “”“ “““) are used for multi-line strings.

f-Strings (Formatted String Literals)
Introduced in Python 3.6, f-strings allow you to embed expressions inside string literals, using curly braces {}.OP: John is 30 years old.

\r: Carriage return-
Moves the cursor to the beginning of the line. Often used in conjunction with \n for platform-specific newlines (e.g.,Windows uses \r\n)
print("Hello, World!\rPython")

Output:
Python World!

Python indentation is a way of telling a Python interpreter that the group of statements belongs to a particular block of code.

String Methods

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters

lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string

The Python List is an ordered collection (also known as a sequence) of elements. List elements can be accessed, iterated, and removed according to the order they inserted at the creation time.

Basic List Operations

Method	Description	Syntax	Example	O
append()	Appends an element to the list. In insert(), if the index is 0, then element is inserted as the first element and if we write, list.insert(len(list), obj), then it inserts obj as the last element in the list. That is, if index= len(list) then insert() method behaves exactly same as append() method.	list. append(obj)	num_list = [6,3,7,0,1,2,4,9] num_list. append(10) print (num_list)	[6 0, 4,
count()	Counts the number of times an element appears in the list.	list.count (obj)	print(num_list. count(4))	1
index()	Returns the lowest index of obj in the list. Gives a ValueError if obj is not present in the list.	list. index(obj)	>>> num_list = [6,3,7,0,3,7,6,0] >>> print(num_ list.index(7))	2
insert()	Inserts obj at the specified index in the list.	list. insert(index, obj)	>>> num_list = [6,3,7,0,3,7,6,0] >>> num_list. insert(3, 100) >>> print(num_list)	[1 3 0
pop()	Removes the element at the specified index from the list. Index is an optional parameter. If no index is specified, then removes the last object (or element) from the list.	list. pop([index])	num_list = [6,3,7,0,1,2,4,9] print(num_list. pop()) print(num_list)	9 [7 2

List Methods

remove()	Removes or deletes obj from the list. ValueError is generated if obj is not present in the list. If multiple copies of obj exists in the list, then the first value is deleted.	list. remove(obj)	>>> num_list = [6,3,7,0,1,2,4,9] >>> num_list. remove(0) >>> print(num_list)	[6,3,7, 1,2,4, 9]
reverse()	Reverse the elements in the list.	list. reverse()	>>> num_list = [6,3,7,0,1,2,4,9] >>> num_list. reverse() >>>print(num_list)	[9, 4, 2, 1, 7, 3, 6]
sort()	Sorts the elements in the list.	list.sort()	>>> num_list = [6,3,7,0,1,2,4,9] >>> num_list. sort() >>> print(num_list)	[9, 4, 2, 1, 0, 7, 3, 6]
extend()	Adds the elements in a list to the end of another list. Using + or += on a list is similar to using extend().	list1. extend(list2)	>>> num_list1 = [1,2,3,4,5] >>> num_list2 = [6,7,8,9,10] >>> num_list1. extend(num_list2) >>>print(num_ list1)	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Tuples are ordered collections of elements that are unchangeable. The tuple is the same as the list, except the tuple is immutable means we can't modify the tuple once created.

Basic Tuple Operations

Operation	Expression	Output
Length	`len((1,2,3,4,5,6))`	`6`
Concatenation	`(1,2,3) + (4,5,6)`	`(1, 2, 3, 4, 5, 6)`
Repetition	`('Good..')*3`	`'Good..Good..Good..'`
Membership	`5 in (1,2,3,4,5,6,7,8,9)`	`True`
Iteration	`for i in (1,2,3,4,5,6,7,8,9,10):` `  print(i,end=' ')`	`1,2,3,4,5,6,7,8,9,10`

Comparison (Use >, <, ==)	`Tup1 = (1,2,3,4,5)`
	`Tup2 = (1,2,3,4,5)`
	`print(Tup1>Tup2)`
		`False`

Maximum	`max(1,0,3,8,2,9)`	`9`
Minimum	`min(1,0,3,8,2,9)`	`0`
Convert to tuple (converts a sequence into a tuple)	`tuple("Hello")` `tuple([1,2,3,4,5])`	`('H', 'e', 'l', 'l', 'o')` `(1, 2, 3, 4, 5)`

Since tuples are immutable, iterating through tuples is faster than iterating over a list. This means that a tuple performs better than a list.
In Python, dictionaries are unordered collections of unique values stored in (Key-Value) pairs. Use a dictionary data type to store data as a key-value pair.

First, a list is an ordered set of items. But, a dictionary is a data structure that is used for matching one item (key) with another (value).

In Python, a set is an unordered collection of data items that are unique. In other words, Python Set is a collection of elements (Or objects) that contains no duplicate elements.

```
# define an empty set
a_set = set()

# define a set
a_set = {'one', 2}

# adding item to a set
a_set.add('c')

# removing an item from a set, raise keyError if item does not exist
a_set.remove('one')

# removing an item from a set, no keyError if item does not exist
a_set.discard('one')

# remove the first item from a set and return it
```

a_set.pop()

# remove all items from a set
a_set.clear()

The union() function combines the data present in both sets.

The intersection() function finds the data present in both sets only.

The difference() function deletes the data present in both and outputs data present only in the set passed.

The symmetric_difference() does the same as the difference() function but outputs the data which is remaining in both sets.

Arrays and lists are the same structure with one difference, Lists allow heterogeneous data element storage whereas Array allow only homogenous elements to be stored within them.
import array as arr

Use lists to store a collection of data that does not need random access.

Use lists if the data has to be modified frequently.

Use a set if you want to ensure that every element in the data structure must be unique.

Use tuples when you want that your data should not be altered.

—----------
A lambda function is a small anonymous function (defined without a name).

Lambda functions can take any number of arguments:
Errors are problems in a program that causes the program to stop its execution.
The finally block runs whether or not an exception occurred. It's often used for cleanup actions.