Strat Of the Day 10:42 Monday 12 May 2025

Python

Python - What is Python and Its Feature with Setup and Installation

Python is a high-level, general-purpose programming language known for its readability and versatility. It's widely used in web development, data science, machine learning, and various other fields. Python is appreciated for its ease of learning and ability to run on different platforms.

Key characteristics of Python:

* **High-level:**

Python is designed to be easy to read and understand, with a clear syntax that resembles natural language.

* **General-purpose:**

Python is not specialized for any particular task and can be used for a wide range of applications, including web development, data analysis, and software development.

* **Interpreted:**

Python code is executed line by line by an interpreter, which means that code can be executed as soon as it is written.

* **Object-oriented:**

Python supports object-oriented programming principles, allowing developers to create reusable and modular code.

* **Dynamic typing:**

Python does not require developers to declare the data type of variables, making it a flexible and dynamic language.

* **Large standard library:**

Python comes with a vast collection of built-in modules and functions that can be used to perform a wide variety of tasks.

* **Extensible:**

Python can be extended with code written in other languages like C or C++, making it suitable for performance-critical applications.

* **Open source:**

Python is free to use and distribute under a permissive license, which encourages community involvement and innovation.

Why Python is popular:

* **Readability:**

Python's syntax is designed to be clear and concise, making it easier to read and understand than other languages like C++ or Java.

* **Ease of learning:**

Python's simple syntax and clear structure make it a popular choice for beginners.

* **Large community:**

The Python community is large and active, providing extensive documentation, tutorials, and support.

* **Versatility:**

Python can be used for a wide range of applications, from web development to data science.

* **Rapid prototyping:**

Python's interpreted nature and dynamic typing make it ideal for quickly prototyping and experimenting with code.

Examples of Python's use:

* **Web development:** Python frameworks like Django and Flask are used to build web applications.
* **Data science:** Python libraries like NumPy, Pandas, and Scikit-learn are used for data analysis and machine learning.
* **Software development:** Python is used to develop a variety of software applications, from simple scripts to complex systems.
* **Automation:** Python can be used to automate tasks, such as running tests, building software, and managing infrastructure.
* **Scientific computing:** Python is used for scientific research, simulations, and modeling.
* **Game development:** Python can be used to script game logic and create game applications.

Python – Variables

Variables

Variables are containers for storing data values.

Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Example

x = 5  
y = "John"  
print(x)   
print(y)

Variables do not need to be declared with any particular type, and can even change type after they have been set.

### Example

x = 4       # x is of type int  
x = "Sally" # x is now of type str  
print(x) // Sally

## Casting

If you want to specify the data type of a variable, this can be done with casting.

### Example

x = str(3)    # x will be '3'  
y = int(3)    # y will be 3  
z = float(3)  # z will be 3.0

## Get the Type

You can get the data type of a variable with the type() function.

### Example

x = 5  
y = "John"  
print(type(x))  
print(type(y))

## Single or Double Quotes?

String variables can be declared either by using single or double quotes:

### Example

x = "John"  
# is the same as  
x = 'John'

## Case-Sensitive

Variable names are case-sensitive.

### Example

This will create two variables:

a = 4  
A = "Sally"  
#A will not overwrite a

### ✅ 1. ****Variable Naming Rules (Identifiers)****

* **Start only with a letter (a-z, A-Z) or underscore (\_)**
* **Cannot start with a digit**
* **Only letters, digits, and underscores allowed**
* **No special characters like @, $, %, etc.**
* **Case-sensitive** (e.g., name, Name, and NAME are different)

### ✅ 2. ****Multiple Assignment****

Ek hi line mein multiple variables assign kar sakte ho:

x, y, z = 1, 2, 3

Ya same value sabko:

a = b = c = "Python"

### ✅ 3. ****Unpacking a Collection****

List ya tuple ke elements ko directly variables mein assign kar sakte ho:

fruits = ["apple", "banana", "cherry"]

x, y, z = fruits

### ✅ 4. ****Global vs Local Variables****

Agar variable function ke bahar banaya hai = **global**,  
function ke andar banaya = **local**

x = "global"

def func():

x = "local"

print(x)

func() # local

print(x) # global

### ✅ 5. ****Global Keyword****

Function ke andar global variable ko modify karne ke liye:

x = "awesome"

def myfunc():

global x

x = "fantastic"

myfunc()

print(x) # fantastic

### ✅ 6. ****Dynamic Typing****

Python mein variable ka type change ho sakta hai:

x = 10 # int

x = "hi" # ab str

# Python Data Types

**Last Updated :**12 Mar, 2025

Python Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, Python data types are classes and variables are instances (objects) of these classes. The following are the standard or built-in data types in Python:

* **Numeric –**[int](https://www.geeksforgeeks.org/python-numbers/" \t "_blank), [float](https://www.geeksforgeeks.org/python-float-type-and-its-methods/), [complex](https://www.geeksforgeeks.org/python-complex-function/)
* **Sequence Type –**[string](https://www.geeksforgeeks.org/python-string/), [list](https://www.geeksforgeeks.org/python-lists/), [tuple](https://www.geeksforgeeks.org/python-tuples/)
* **Mapping Type –**[dict](https://www.geeksforgeeks.org/python-dictionary/" \t "_blank)
* **Boolean –**[bool](https://www.geeksforgeeks.org/boolean-data-type-in-python/" \t "_blank)
* **Set Type –**[set](https://www.geeksforgeeks.org/python-sets/), [frozenset](https://www.geeksforgeeks.org/frozenset-in-python/" \t "_blank)
* **Binary Types –**[bytes](https://www.geeksforgeeks.org/python-bytes-method/), [bytearray](https://www.geeksforgeeks.org/python-bytearray-function/), [memoryview](https://www.geeksforgeeks.org/memoryview-in-python/" \t "_blank)

### ****Numeric Data Types in Python****

1. **int** – Integer (whole numbers, positive/negative, no decimals)  
   Example: 5, -10, 1000
2. **float** – Decimal numbers (real numbers with fractions)  
    Example: 3.14, -2.0, 1e3 (means 1000.0)
3. **complex** – Numbers with real and imaginary parts  
    Example: 2 + 3j, 5j, 1.5 - 2j

a = 5

print(type(a))

b = 5.0

print(type(b))

c = 2 + 4j

print(type(c))

<class 'int'>

<class 'float'>

<class 'complex'>

**2. Sequence Data Types in Python**

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion. There are several sequence data types of Python:

* [Python String](https://www.geeksforgeeks.org/python-string/)
* [Python List](https://www.geeksforgeeks.org/python-lists/)
* [Python Tuple](https://www.geeksforgeeks.org/tuples-in-python/)

### ****Sequence Data Types in Python****

Sequence data types are **ordered collections** of data. Each element in a sequence has a defined position (index), which allows accessing elements by their position.

Python has the following main sequence data types:

### 1. ****String (****str****)****

### String Data Type

Python[Strings](https://www.geeksforgeeks.org/python-strings/)are arrays of bytes representing Unicode characters. In Python, there is no character data type Python, a character is a string of length one. It is represented by str class.

Strings in Python can be created using single quotes, double quotes or even triple quotes. We can access individual characters of a String using index.

* A string is a sequence of characters enclosed in single, double, or triple quotes.
* Strings are **immutable** (cannot be changed after creation).

**Example:**

python

CopyEdit

s = "Hello"

print(s[0]) # Output: H (indexing)

print(s[1:4]) # Output: ell (slicing)

In Python, **negative indexing** ka matlab hota hai **right se** access karna.

s = 'Welcome to the Geeks World'

print(s)

# check data type

print(type(s))

# access string with index

print(s[1])

print(s[2])

print(s[-1]) // d

### List Data Type

[Lists](https://www.geeksforgeeks.org/python-list/)are just like arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

**Creating a List in Python**

Lists in Python can be created by just placing the sequence inside the square brackets[].

* A list is an ordered, **mutable** collection of items.
* Items can be of **different data types**.
* Declared using square brackets [].

**Example:**

# Empty list

a = []

# list with int values

a = [1, 2, 3]

print(a)

# list with mixed int and string

b = ["Geeks", "For", "Geeks", 4, 5]

print(b)

my\_list = [10, "Python", 3.14]

print(my\_list[1]) # Output: Python

my\_list[0] = 20 # Modifying list element

print(my\_list) # Output: [20, 'Python', 3.14]

**Access List Items**

In order to access the list items refer to the index number. In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

### Tuple Data Type

Just like a list, a [tuple](https://www.geeksforgeeks.org/python-tuples/) is also an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable. Tuples cannot be modified after it is created.

#### ****Creating a Tuple in Python****

In Python Data Types,[tuples](https://www.geeksforgeeks.org/python-tuples/)are created by placing a sequence of values separated by a ‘comma’ with or without the use of parentheses for grouping the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, lists, etc.).

***Note:*** *Tuples can also be created with a single element, but it is a bit tricky. Having one element in the parentheses is not sufficient, there must be a trailing* ***‘comma’*** *to make it a tuple.*

*# initiate empty tuple*

*tup1 = ()*

*tup2 = ('Geeks', 'For')*

*print("\nTuple with the use of String: ", t2)*

* A tuple is an ordered, **immutable** collection of items.
* Declared using parentheses ().

**Example:**

python

CopyEdit

my\_tuple = (1, 2, "Python")

print(my\_tuple[2]) # Output: Python

# my\_tuple[0] = 5 # Error: Tuples are immutable

### ****Tuple Packing in Python****

**Tuple packing** means putting multiple values together into a single tuple **without using parentheses explicitly** (although you can use them).

### ****Syntax & Example:****

# Tuple Packing

my\_tuple = 10, 20, "Python"

print(my\_tuple) # Output: (10, 20, 'Python')

print(type(my\_tuple)) # Output: <class 'tuple'>

Even though we didn’t use parentheses, Python **automatically packs** the values into a tuple

### ****You can also do unpacking:****

a, b, c = my\_tuple

print(a) # Output: 10

print(b) # Output: 20

print(c) # Output: Python

#### Access Tuple Items

In order to access the tuple items refer to the index number. Use the index operator [ ] to access an item in a tuple.

tup1 = tuple([1, 2, 3, 4, 5])

# access tuple items

print(tup1[0])

print(tup1[-1])

print(tup1[-3])

## Boolean Data Type in Python

Python Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). However non-Boolean objects can be evaluated in a Boolean context as well and determined to be true or false. It is denoted by the class bool.

**Example:**The first two lines will print the type of the boolean values True and False, which is **<class ‘bool’>.**The third line will cause an error, because true is not a valid keyword in Python. Python is case-sensitive, which means it distinguishes between uppercase and lowercase letters.

print(type(True))

print(type(False))

print(type(true))

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

Traceback (most recent call last):  
 File "/home/7e8862763fb66153d70824099d4f5fb7.py", line 8, in   
 print(type(true))  
NameError: name 'true' is not defined

## 4. Set Data Type in Python

In Python Data Types, [Set](https://www.geeksforgeeks.org/python-sets/)is an unordered collection of data types that is iterable, mutable, and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

### Create a Set in Python

Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by a **‘comma’.**The type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

**Example:**The code is an example of how to create sets using different types of values, such as **strings**, **lists**, and mixed values

# initializing empty set

s1 = set()

s1 = set("GeeksForGeeks")

print("Set with the use of String: ", s1)

s2 = set(["Geeks", "For", "Geeks"])

print("Set with the use of List: ", s2)

### ****Access Set Items****

Set items cannot be accessed by referring to an index, since sets are unordered the items have no index. But we can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in the keyword.

set1 = set(["Geeks", "For", "Geeks"])

print(set1)

# loop through set

for i in set1:

print(i, end=" ")

# check if item exist in set

print("Geeks" in set1)

#### ****Explanation:****

* You are creating a set named set1 using a list ["Geeks", "For", "Geeks"].
* Sets in Python **automatically remove duplicate values**.
* So "Geeks" is written twice in the list, but the set will only keep **one "Geeks"**.

### ****Summary of Your Code:****

1. You created a **set from a list** that had duplicates → Python removed duplicates.
2. You **printed the set** → elements came in random order.
3. You **looped through the set** → printed each element.
4. You **checked membership** → "Geeks" is in the set → result is True.

## Dictionary Data Type

A dictionary in Python is a collection of data values, used to store data values like a map, unlike other Python Data Types that hold only a single value as an element, a Dictionary holds a key: value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon : , whereas each key is separated by a ‘comma’.

### Create a Dictionary in Python

Values in a dictionary can be of any datatype and can be duplicated, whereas keys can’t be repeated and must be immutable. The dictionary can also be created by the built-in function **dict().**

***Note*** *– Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.*

# initialize empty dictionary

d = {}

d = {1: 'Geeks', 2: 'For', 3: 'Geeks'}

print(d)

# creating dictionary using dict() constructor

d1 = dict({1: 'Geeks', 2: 'For', 3: 'Geeks'})

print(d1)

{1: 'Geeks', 2: 'For', 3: 'Geeks'}

{1: 'Geeks', 2: 'For', 3: 'Geeks'}

### ****Accessing Key-value in Dictionary****

In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets. Using **get() method**we can access the dictionary elements.

d = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# Accessing an element using key

print(d['name'])

# Accessing a element using get

print(d.get(3))

## Python Data Type Exercise Questions

Below are two exercise questions on Python Data Types. We have covered list operation and tuple operation in these exercise questions. For more exercises on Python data types visit the page mentioned below.

**Q1. Code to implement basic list operation**

fruits = ["apple", "banana", "orange"]

print(fruits)

fruits.append("grape")

print(fruits)

fruits.remove("orange")

print(fruits)

**Output**

['apple', 'banana', 'orange']

['apple', 'banana', 'orange', 'grape']

['apple', 'banana', 'grape']

**Q2. Code to implement basic tuple operation**

coordinates = (3, 5)

print(coordinates)

print("X-coordinate:", coordinates[0])

print("Y-coordinate:", coordinates[1])

**Output**

(3, 5)

X-coordinate: 3

Y-coordinate: 5

Python Exception Handling

**Last Updated :**02 Apr, 2025

Python Exception Handling handles errors that occur during the execution of a program. Exception handling allows to respond to the error, instead of crashing the running program. It enables you to catch and manage errors, making your code more robust and user-friendly. Let’s look at an example:

### Handling a Simple Exception in Python

Exception handling helps in preventing crashes due to errors. Here’s a basic example demonstrating how to catch an exception and handle it gracefully:

# Simple Exception Handling Example

n = 10

try:

res = n / 0 # This will raise a ZeroDivisionError

except ZeroDivisionError:

print("Can't be divided by zero!")

**Output**

Can't be divided by zero!

**Explanation:** In this example, dividing number by 0 raises a **[ZeroDivisionError](https://www.geeksforgeeks.org/zerodivisionerror-float-division-by-zero-in-python/" \t "_blank)**. The try block contains the code that might cause an exception and the except block handles the exception, printing an error message instead of stopping the program.

**Difference Between Exception and Error**

* **Error**: Errors are serious issues that a program should not try to handle. They are usually problems in the code’s logic or configuration and need to be fixed by the programmer. Examples include syntax errors and memory errors.
* **Exception**: Exceptions are less severe than errors and can be handled by the program. They occur due to situations like invalid input, missing files or network issues.

# Syntax Error (Error)

print("Hello world" # Missing closing parenthesis

# ZeroDivisionError (Exception)

n = 10

res = n / 0

**Explanation:** A syntax error is a coding mistake that prevents the code from running. In contrast, an exception like ZeroDivisionError can be managed during the program’s execution using exception handling.

### Syntax and Usage

Exception handling in Python is done using the try, except, else and finally blocks.

*try:  
# Code that might raise an exception  
except SomeException:  
# Code to handle the exception  
else:  
# Code to run if no exception occurs  
finally:  
# Code to run regardless of whether an exception occurs*

**try, except, else and finally Blocks**

* **try Block**: [try block](https://www.geeksforgeeks.org/python-try-except/) lets us test a block of code for errors. Python will “try” to execute the code in this block. If an exception occurs, execution will immediately jump to the except block.
* **except Block:** [except block](https://www.geeksforgeeks.org/python-try-except/) enables us to handle the error or exception. If the code inside the try block throws an error, Python jumps to the except block and executes it. We can handle specific exceptions or use a general except to catch all exceptions.
* **else Block:** [else block](https://www.geeksforgeeks.org/try-except-else-and-finally-in-python/) is optional and if included, must follow all except blocks. The else block runs only if no exceptions are raised in the try block. This is useful for code that should execute if the try block succeeds.
* **finally Block:** [finally block](https://www.geeksforgeeks.org/finally-keyword-in-python/)always runs, regardless of whether an exception occurred or not. It is typically used for cleanup operations (closing files, releasing resources).

## Common Exceptions in Python

Python has many [built-in exceptions](https://www.geeksforgeeks.org/built-exceptions-python/), each representing a specific error condition. Some common ones include:

| **Exception Name** | **Description** |
| --- | --- |
| **BaseException** | The base class for all built-in exceptions. |
| [Exception](https://www.geeksforgeeks.org/python-exception-handling/) | The base class for all non-exit exceptions. |
| **ArithmeticError** | Base class for all errors related to arithmetic operations. |
| [ZeroDivisionError](https://www.geeksforgeeks.org/zerodivisionerror-float-division-by-zero-in-python/) | Raised when a division or modulo operation is performed with zero as the divisor. |
| [OverflowError](https://www.geeksforgeeks.org/python-overflowerror-math-range-error/) | Raised when a numerical operation exceeds the maximum limit of a data type. |
| [FloatingPointError](https://www.geeksforgeeks.org/floating-point-error-in-python/) | Raised when a floating-point operation fails. |
| [AssertionError](https://www.geeksforgeeks.org/python-assertion-error/) | Raised when an assert statement fails. |
| [AttributeError](https://www.geeksforgeeks.org/python-attributeerror/) | Raised when an attribute reference or assignment fails. |
| [IndexError](https://www.geeksforgeeks.org/python-list-index-out-of-range-indexerror/) | Raised when a sequence subscript is out of range. |
| [KeyError](https://www.geeksforgeeks.org/how-to-handle-keyerror-exception-in-python/) | Raised when a dictionary key is not found. |
| [MemoryError](https://www.geeksforgeeks.org/how-to-handle-the-memoryerror-in-python/) | Raised when an operation runs out of memory. |
| [NameError](https://www.geeksforgeeks.org/handling-nameerror-exception-in-python/) | Raised when a local or global name is not found. |
| [OSError](https://www.geeksforgeeks.org/handling-oserror-exception-in-python/) | Raised when a system-related operation (like file I/O) fails. |
| [TypeError](https://www.geeksforgeeks.org/handling-typeerror-exception-in-python/) | Raised when an operation or function is applied to an object of inappropriate type. |
| [ValueError](https://www.geeksforgeeks.org/how-to-fix-valueerror-exceptions-in-python/) | Raised when a function receives an argument of the right type but inappropriate value. |
| [ImportError](https://www.geeksforgeeks.org/importerror-unknown-location-in-python/) | Raised when an import statement has issues. |
| [ModuleNotFoundError](https://www.geeksforgeeks.org/how-to-fix-the-module-not-found-error/) | Raised when a module cannot be found. |

### 1. ****Is it necessary to mention the exception name to handle it?****

**No**, it's **not necessary** to mention the exception name while handling exceptions, but **it's not a good practice** to omit it.

### 1. Without specifying exception name:

try:

# some risky code

x = 10 / 0

except:

print("An error occurred!")

* This **will work**, and it will catch the **ZeroDivisionError**.

### ⚠️ But what's the problem?

* It will **catch all kinds of errors** — whether it's ZeroDivisionError, ValueError, or even something unexpected.
* This makes **debugging difficult**, because you won't know what actually went wrong.
* Example: It will even catch serious issues like FileNotFoundError, TypeError, or KeyboardInterrupt.

### 2. Better approach (Recommended):

try:

x = 10 / 0

except ZeroDivisionError:

print("Can't divide by zero.")

* This will handle **only** the specific error you're expecting.

### 3. Handling multiple exceptions:

try:

# risky code

except (ZeroDivisionError, ValueError):

print("Handled common math errors.")

### 4. Catching all errors with details (Best for debugging):

try:

x = 10 / 0

except Exception as e:

print("Error occurred:", e)

* This will show **what error actually happened**, and is very useful during development.

## Python Catching Exceptions

When working with exceptions in Python, we can handle errors more efficiently by specifying the types of exceptions we expect. This can make code both safer and easier to debug.

### Catching Specific Exceptions

Catching specific exceptions makes code to respond to different exception types differently.

**Example:**

try:

x = int("str") # This will cause ValueError

#inverse

inv = 1 / x

except ValueError:

print("Not Valid!")

except ZeroDivisionError:

print("Zero has no inverse!")

**Output**

Not Valid!

**Explanation:**

* The ValueError is caught because the string “str” cannot be converted to an integer.
* If x were 0 and conversion successful, the ZeroDivisionError would be caught when attempting to calculate its inverse.

### Catching Multiple Exceptions

We can catch multiple exceptions in a single block if we need to handle them in the same way or we can separate them if different types of exceptions require different handling.

**Example:**

a = ["10", "twenty", 30] # Mixed list of integers and strings

try:

total = int(a[0]) + int(a[1]) # 'twenty' cannot be converted to int

except (ValueError, TypeError) as e:

print("Error", e)

except IndexError:

print("Index out of range.")

**Output**

Error invalid literal for int() with base 10: 'twenty'

**Explanation:**

* The ValueError is caught when trying to convert “twenty” to an integer.
* TypeError might occur if the operation was incorrectly applied to non-integer types, but it’s not triggered in this specific setup.
* IndexError would be caught if an index outside the range of the list was accessed, but in this scenario, it’s under control.

### Catch-All Handlers and Their Risks

Here’s a simple calculation that may fail due to various reasons.

**Example:**

try:

# Simulate risky calculation: incorrect type operation

res = "100" / 20

except ArithmeticError:

print("Arithmetic problem.")

except:

print("Something went wrong!")

**Output**

Something went wrong!

**Explanation:**

* An ArithmeticError (more specific like ZeroDivisionError) might be caught if this were a number-to-number division error. However, TypeError is actually triggered here due to attempting to divide a string by a number.
* **catch-all except:** is used to catch the TypeError, demonstrating the risk that the programmer might not realize the actual cause of the error (type mismatch) without more detailed error logging.

**Raise an Exception**

We [raise](https://www.geeksforgeeks.org/python-raise-keyword/) an exception in Python using the raise keyword followed by an instance of the exception class that we want to trigger. We can choose from built-in exceptions or define our own custom exceptions by inheriting from Python’s built-in Exception class.

**Basic Syntax:**

*raise ExceptionType(“Error message”)*

def set(age):

if age < 0:

raise ValueError("Age cannot be negative.")

print(f"Age set to {age}")

try:

set(-5)

except ValueError as e:

print(e)

**Output**

Age cannot be negative.

**Explanation:**

* The function set checks if the age is negative. If so, it raises a ValueError with a message explaining the issue.
* This ensures that the age attribute cannot be set to an invalid state, thus maintaining the integrity of the data.

### Advantages of Exception Handling:

* **Improved program reliability**: By handling exceptions properly, you can prevent your program from crashing or producing incorrect results due to unexpected errors or input.
* **Simplified error handling**: Exception handling allows you to separate error handling code from the main program logic, making it easier to read and maintain your code.
* **Cleaner code:** With exception handling, you can avoid using complex conditional statements to check for errors, leading to cleaner and more readable code.
* **Easier debugging**: When an exception is raised, the Python interpreter prints a traceback that shows the exact location where the exception occurred, making it easier to debug your code.

### Disadvantages of Exception Handling:

* **Performance overhead:** Exception handling can be slower than using conditional statements to check for errors, as the interpreter has to perform additional work to catch and handle the exception.
* **Increased code complexity**: Exception handling can make your code more complex, especially if you have to handle multiple types of exceptions or implement complex error handling logic.
* **Possible security risks:** Improperly handled exceptions can potentially reveal sensitive information or create security vulnerabilities in your code, so it’s important to handle exceptions carefully and avoid exposing too much information about your program.

logging module

The logging module in Python is a built-in library that provides a flexible framework for emitting log messages from Python programs. It allows developers to record events that occur during the execution of a program, which can be useful for debugging, monitoring, and understanding the behavior of the application.

Basic Usage

To use the logging module, you first need to import it:

import logging

Then, you can create a logger instance using logging.getLogger():

logger = logging.getLogger(\_\_name\_\_)

After that, you can log messages at different severity levels using the logger instance:

logger.debug('This is a debug message')

logger.info('This is an info message')

logger.warning('This is a warning message')

logger.error('This is an error message')

logger.critical('This is a critical message')

Log Levels

The logging module defines several standard log levels, each representing a different level of severity:

* DEBUG: Detailed information, typically used for debugging purposes.
* INFO: General information about the execution of the program.
* WARNING: An indication that something unexpected happened, or indicative of some problem in the near future.
* ERROR: A more serious problem that the program was unable to handle.
* CRITICAL: A critical error that may lead to the termination of the program.

Configuration

The logging module can be configured in various ways, including: Basic configuration using logging.basicConfig(), Configuration using a configuration file, and Programmatic configuration.

Basic configuration using logging.basicConfig() is the simplest way to configure the logging module:

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')

This will set the logging level to INFO and define a format for the log messages.

Handlers and Formatters

The logging module uses handlers to send log messages to different destinations, such as the console, a file, or a network socket. Formatters are used to format the log messages before they are sent to the destination.

You can create custom handlers and formatters to customize the behavior of the logging module.