Python Interview Questions:

What is diffrence between modules and package?

**Script :** A script is a python file that is intended to run directly. If you run it, it should do something. This means the script contains code written outside the scope of classes and functions.

Def add(x,y):

Return x +y

Result = add(5,3)

Print(result)

**Module:** A module is a python file that is intentended to be imported into other scripts. It consists of classes, functions and variables to be used in other files.

Def add(x+y):

Return x+y

**Packages:** Collection of related modules that work together to provide certain functionallity.

[Article for Packages](https://www.scaler.com/topics/python/python-packages/)

**Library :** It is a collections of packages. It contains 100’s of modules in it. These module provide wide range of functionalities.

2. How python is interpretator? / Is python is compile or interpreted language? / why python is interpreted language?

**Python is an interpreted language, but it also involves some compilation.** This means when we run a python code, it is first compiled and then interpreted line by line. The compilation part is mostly hidden from the user. While running the code, Python generates a byte code internally, this byte code is then converted using a python virtual machine (p.v.m) to generate the output.

Interpreted in simple terms means running code **line by line**. It also means that the instruction is executed without earlier compiling the whole program into machine language.

Now, let us try to prove the fact python is both compiled as well interpreted.

Take a Sample Code

[Sample Code:](file:///H:\Python%20Practice)

To run above python code simply type Python3 “Name of the file”

Your python file will be executed after running above commond and you will get output in terminal itself.

Now, In order to get bytecode containing \_\_pycache\_\_ file you have to run following commonds.

Firstly run python in order to check is there any error or not.

Import py\_compile (py\_compile is module)

py\_compile.compile("Python\_Compiled\_Interpreted.py")

**Advantages of Interpreted Languages**

An interpreted language gives some extra benefits and flexibility over compiled language.

* Since the interpreter reads instructions line by line and generates output till the point the code is correct, the ease of debugging increases as it is easier to get information about the source point of error.
* The size of programs written in Python is less as compared to other languages.
* As Python generates byte code before interpretation, this byte code can be used by any other platform to generate output.

**Disadvantages of Interpreted Languages**

* A program that is executed in an interpreted language is slower as compared to a language that is directly compiled.
* It happens because the line of codes passes through an interpretation run-time.
* The code has to be compiled, and after compilation, a byte code file is generated before interpretation which makes the execution time high. Therefore due to this problem, the run-time complexity of the program increases.

3. What is byte code?

After running source code by python commond, internally byte code is generated, which is not visible to us.

[Byte Code:](https://prepinsta.com/python/what-is-bytecode-in-python/)

4. What are the benefits of using python language as a tool in the present scenario?

Python is a programming language with objects, modules, exception and automatic memory management. The benefits of pythons are that it is simple and easy, poratable, extensible built in data structures and it is an open source.

5. What is Pickling and Unpickling ?

Pickling:

Also known as serialization, this process converts a Python object into a byte stream that can be saved or transmitted. This is useful for saving and restoring objects, storing large amounts of data, and more.

Unpickling

Also known as deserialization, this process converts a byte stream back into a Python object.

6. Is python case sensitive?

Yes, Python is case-sensitive. This means that identifiers such as variable names, function names, and other names are distinguished by their case.

7. What is pandas?

Pandas is an open-source, high-performance, and easy-to-use library for data manipulation and analysis in Python.

Here's a technical explanation:

**Key Features:**

1. Data Structures: Pandas introduces two primary data structures:

- Series (1-dimensional labeled array): A sequence of values with an index.

- DataFrame (2-dimensional labeled data structure): A table with rows and columns.

2. Data Manipulation: Pandas provides efficient data merging, joining, reshaping, and aggregation.

3. Data Analysis: Pandas integrates well with other popular data analysis libraries in Python, such as NumPy, SciPy, and Matplotlib.

4. Input/Output: Pandas supports reading and writing data from various file formats, including CSV, Excel, JSON, and SQL databases.

**Technical Components:**

1. Indexing: Pandas uses label-based indexing, allowing for intuitive data selection and manipulation.

2. Data Alignment: Pandas automatically aligns data based on indexes, simplifying operations.

3. GroupBy: Pandas enables grouping data by one or more columns and performing aggregation operations.

4. Reshaping: Pandas provides functions for pivoting, melting, and stacking data.

[Examples:](file:///H:\Python%20Practice)

Advantages:

1. Efficient Memory Usage: Pandas optimizes memory usage, handling large datasets.

2. High-Performance Computing: Pandas leverages vectorized operations and caching for speed.

3. Flexible Data Handling: Pandas supports missing data, data merging, and data alignment.

Examples of vectorized operations in pandas:

1. Element-wise operations: df['A'] + df['B'] adds two columns element-wise.

2. Aggregations: df['A'].sum() calculates the sum of an entire column.

3. Conditional operations: df['A'] > 0 creates a boolean mask for values greater than 0.

4. Data alignment: df1 + df2 aligns data based on indexes and performs operations.

**Use Cases:**

1. Data cleaning and preprocessing

2. Data analysis and visualization

3. Data science and machine learning

4. Web scraping and data extraction

5. Business intelligence and reporting

**Integration:**

1. NumPy and SciPy for numerical computations

2. Matplotlib and Seaborn for data visualization

3. Scikit-learn for machine learning

4. SQL databases for data storage

By leveraging pandas, developers and data scientists can efficiently handle structured data, perform complex analysis, and extract valuable insights.

**8. Differnce between loc and iloc**

loc

- Label-based indexing

- Accesses rows and columns by their labels (index values)

- Includes the end index in the range

Example:

import pandas as pd

df = pd.DataFrame({

'A': [1, 2, 3],

'B': [4, 5, 6]

}, index=['row1', 'row2', 'row3'])

print(df.loc['row1', 'A']) # Output: 1

print(df.loc['row1':'row2', 'A']) # Output: [1, 2]

iloc

- Integer-based indexing

- Accesses rows and columns by their integer positions

- Excludes the end index in the range

Example:

import pandas as pd

df = pd.DataFrame({

'A': [1, 2, 3],

'B': [4, 5, 6]

})

print(df.iloc[0, 0]) # Output: 1

print(df.iloc[0:2, 0]) # Output: [1, 2]

Key differences:

| | loc | iloc |

| --- | --- | --- |

| Indexing type | Label-based | Integer-based |

| Row access | df.loc['row\_label'] | df.iloc[row\_number] |

| Column access | df.loc[:, 'column\_label'] | df.iloc[:, column\_number] |

| Range inclusion | Includes end index | Excludes end index |

When to use each:

- Use loc when:

- Working with labeled data

- Need to access specific rows or columns by name

- Use iloc when:

- Working with numerical data

- Need to access rows or columns by position

**9. How is exceptional handling done in python?**

*(Try except usecase)*

Exception handling in Python is done using try-except blocks. Here's a basic example:

try:

# Code that might raise an exception

except ExceptionType:

# Handle the exception

Here's a breakdown:

1. Try: Wrap the code that might raise an exception in a try block.

2. Except: Catch the exception with an except block, specifying the exception type (e.g., ExceptionType).

3. Handle: Write code to handle the exception in the except block.

**Example:**

try:

x = 1 / 0

except ZeroDivisionError:

print("Cannot divide by zero!")

This code attempts to divide by zero, which raises a ZeroDivisionError. The except block catches this exception and prints a friendly error message.

**You can also:**

- Catch multiple exceptions with multiple except blocks

- Use except Exception as e to access the exception object

- Use finally to execute code regardless of whether an exception was raised

- Raise custom exceptions with raise

**Best practices:**

- Keep try blocks small to minimize the scope of exceptions

- Be specific with exception types to avoid catching unrelated exceptions

- Provide informative error messages or logging

- Use finally for cleanup code, like closing files or connections

The main difference between a for loop and a while loop is the way they handle iteration:

**9. What is diffrence between for loop and while loop?**

For Loop:

1. Iterates over a sequence (e.g., list, tuple, string) or an iterator.

2. Automatically updates the loop variable to the next value in the sequence.

3. Executes the loop body for each value in the sequence.

4. Terminates when the end of the sequence is reached.

While Loop:

1. Executes the loop body as long as a specified condition is true.

2. Requires manual updating of the loop variable (if needed).

3. Continues executing until the condition becomes false.

Key differences:

1. Iteration style: For loops iterate over a sequence, while loops iterate based on a condition.

2. Loop variable management: For loops automatically update the loop variable, while loops require manual updates.

3. Termination: For loops terminate when the sequence ends, while loops terminate when the condition becomes false.

Choose a for loop when:

- You need to iterate over a sequence.

- You want automatic loop variable updates.

Choose a while loop when:

- You need to repeat a task based on a condition.

- You need more control over the loop variable.

[Example:](file:///H:\Python%20Practice)

Indentation is required in Python to denote block-level structure, such as:

1. Code blocks within control structures (if, for, while, etc.)

2. Function definitions

3. Class definitions

Python uses indentation (spaces or tabs) to determine the grouping of statements, making the code more readable and avoiding the need for explicit block delimiters

if True:

print("Indented code block")

print("Part of the same block")

print("Outside the block")

In this example, the two print statements are part of the same code block because they're indented equally. The third print statement is outside the block because it's not indented.

Python's indentation requirements:

- Use 4 spaces for indentation (PEP 8 recommendation)

- Be consistent in your indentation (spaces or tabs, but not both)

- Indent code blocks, function definitions, and class definitions

Proper indentation is crucial in Python, as incorrect indentation can lead to syntax errors or unexpected behavior.

**10. How to delete file in python?**

To delete a file in Python, you can use the os module or the pathlib module.

Using os module:

1. Import the os module: import os

2. Use os.remove() to delete the file: os.remove('file\_name.txt')

Example:

import os

os.remove('example.txt')

Using pathlib module:

1. Import the pathlib module: import pathlib

2. Create a Path object for the file: file\_path = pathlib.Path('file\_name.txt')

3. Use file\_path.unlink() to delete the file: file\_path.unlink()

Example:

import pathlib

file\_path = pathlib.Path('example.txt')

file\_path.unlink()

Note:

- Make sure the file exists before trying to delete it.

- Be cautious when deleting files, as it's a permanent operation.

- If you want to delete a directory, use os.rmdir() or file\_path.rmdir() instead.

Also, you can use try-except block to handle errors, like file not found:

try:

os.remove('example.txt')

except FileNotFoundError:

print("File not found")

**11.Which sorting technic used by sort() and sorted() functions in python?**

i. Sorted will return list object where as .sort will not return anything

ii. sort is a method of list class

Difference - 01

Sorted is a built-in function used for sorting iterable (list,set,tuple,dictionary,string etc)

Differnce – 02

Sort() the original list (modifies original list)

Return None

Sorted() returns new modified sorted iterable

Returns new object

In Python, lists, tuples, and arrays are data structures used to store collections of items. Each has its own characteristics and use cases. Here’s a breakdown of the differences between them:

### 1. Lists

**Characteristics**:

* **Mutable**: Lists can be modified after creation. You can change, add, or remove elements.
* **Ordered**: Elements have a defined order, and this order is maintained.
* **Dynamic**: Lists can grow or shrink in size as needed.
* **Homogeneous and Heterogeneous**: Lists can hold elements of different data types, including other lists.

Syntax:

my\_list = [1, 2, 3, 'hello', [4, 5]]

Operations:

* Access: my\_list[2] returns 3
* Modification: my\_list[1] = 'world'
* Addition: my\_list.append('new')
* Deletion: del my\_list[3]

### 2. Tuples

**Characteristics**:

* **Immutable**: Tuples cannot be modified after creation. Once they are created, you cannot change, add, or remove elements.
* **Ordered**: Elements have a defined order, which is maintained.
* **Fixed Size**: The size of a tuple is fixed once created.
* **Homogeneous and Heterogeneous**: Tuples can also hold elements of different data types.

Syntax:

my\_tuple = (1, 2, 3, 'hello', (4, 5))

Operations:

* Access: my\_tuple[2] returns 3
* Modification: Not allowed (tuples are immutable)
* Addition: Not allowed
* Deletion: Not allowed

3. Arrays (from the array module)

**Characteristics**:

* **Mutable**: Arrays can be modified after creation.
* **Ordered**: Elements have a defined order, which is maintained.
* **Fixed Type**: Arrays are type-restricted; all elements must be of the same type (e.g., integers, floats).
* **Less Flexible**: Arrays are less flexible than lists but more efficient in terms of memory and performance for numerical data.

**Syntax:**

import array

my\_array = array.array('i', [1, 2, 3, 4])

Here, 'i' indicates that the array will hold integers.

Operations:

* Access: my\_array[2] returns 3
* Modification: my\_array[1] = 10
* Addition: my\_array.append(5)
* Deletion: my\_array.remove(4)

Lists:

Mutable, ordered, and flexible in size and data type. Ideal for general-purpose use.

Tuples

Immutable, ordered, and fixed in size. Useful for fixed collections of items and as keys for dictionaries.

Arrays:

Mutable, ordered, but type-restricted. More memory-efficient for numerical data compared to lists.

Each data structure serves different needs based on the specific requirements of the task at hand.

12. What is slicing in python?

Slicing in Python refers to accessing a subset of elements from a sequence (like a list, tuple, or string) by specifying a range of indices. It provides a way to extract parts of sequences without modifying the original sequence.

Syntax

The basic syntax for slicing is:

sequence[start:stop:step]

 start: The index where the slice begins (inclusive). If omitted, it defaults to the beginning of the sequence.

* stop: The index where the slice ends (exclusive). If omitted, it defaults to the end of the sequence.
* step: The step size or stride between elements. If omitted, it defaults to 1.

Examples:

Key Points:

**Start Index**: If omitted, defaults to the beginning of the sequence.

**Stop Index**: If omitted, defaults to the end of the sequence.

**Step**: If omitted, defaults to 1. A negative step value can be used to reverse the sequence.

**Immutability**: For immutable sequences like tuples and strings, slicing returns a new sequence. Lists, which are mutable, are also not modified by slicing, though slices can be assigned back to modify the list.

Slicing is a powerful feature in Python that allows for concise and efficient manipulation of sequences.