Q:1

What is File function in python? What is keywords to create and write file.

Python provides inbuilt functions for creating, writing, and reading files. There are two types of files that can be handled in python, normal text files and binary files (written in binary language, 0s, and 1s).

Text files: In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character (‘\n’) in python by default.

Binary files: In this type of file, there is no terminator for a line, and the data is stored after converting it into machine-understandable binary language.

File Access Modes

Access modes govern the type of operations possible in the opened file. It refers to how the file will be used once its opened. These modes also define the location of the File Handle in the file. File handle is like a cursor, which defines from where the data has to be read or written in the file. There are 6 access modes in python.

Read Only (‘r’) : Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises the I/O error. This is also the default mode in which a file is opened.

Read and Write (‘r+’): Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exist.

Write Only (‘w’) : Open the file for writing. For the existing files, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file.

dWrite and Read (‘w+’) : Open the file for reading and writing. For an existing file, data is truncated and over-written. The handle is positioned at the beginning of the file.Append Only (‘a’): Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file.

Q:2

Explain Exception handling? What is an Error in Python?

In this article, we will discuss how to handle exceptions in Python using try. except, and finally statement with the help of proper examples.

Error in Python can be of two types i.e. Syntax errors and Exceptions. Errors are the problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some internal events occur which changes the normal flow of the program

Difference between Syntax Error and Exceptions:

Syntax Error:

As the name suggests this error is caused by the wrong syntax in the code. It leads to the termination of the program.

Exceptions:

Exceptions are raised when the program is syntactically correct, but the code resulted in an error. This error does not stop the execution of the program, however, it changes the normal flow of the pro

Errors are the problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some

Error Handling

When an error and an exception are raised then we handle that with the help of the Handling method.

Handling Exceptions with Try/Except/Finally

We can handle errors by the Try/Except/Finally method. we write unsafe code in the try, fall back code in except and final code in finally block.

Q:3

How many except statements can a try-except block have? Name Some built-in exception class

There has to be at least one except statement.

Built-in Exceptions:

table below shows built-in exceptions that are usually raised in Python:

Exception	Description
ArithmeticError	Raised when an error occurs in numeric calculations.
AssertionError	Raised when an assert statement fails.
AttributeError	Raised when attribute reference or assignment fails.
Exception	Base class for all exceptions.
EOFError	Raised when the input() method hits an "end of end" condition(EOF).
FloatingPointError	Raised when a floating point calculation fails.
GeneratorExit	Raised when a generator is closed (with the close() Method).
ImportError	Raised when an imported module does not exist.
IndentationError	Raised when indentation is not correct.
IndexError	Raised when an index of a sequence does not exist.
KeyError	Raised when a key does not exist in a dictionary
KeyboardInterrupt	Raised when the user presses Ctrl+c, Ctrl+z or Delete
LookupError	Raised when errors raised cant be found
MemoryError	Raised when a program runs out of memory
NameError	Raised when a variable does not exist
NotImplementedError	Raised when an abstract method requires an inherited the method class to override

Q:4

When will the else part of try-except-else be executed?

The else part is only executed on two conditions :

1. The code in the relevant ‘try’ block has completed
2. The code in the relevant ‘try’ block does not raise an exception.

The situation can seem complex when there are nested try/except block, as it partially depends on what the inner try/except block handles and re-raises.

Imagine this code:

```
try:
        a = function1()
    try:
        b = function2(a)
    except IndexError:
        log.warning('Index Error from function')
    except TypeError:
        raise
except KeyError:
    raise
else:
    print('No exxceptions here')
```

Any exception raised in function1 (regardless of type) will cause the else block to be ignored.

If function2 raises an IndexError then since it has been caught and handled by the inner try/except, it isn’t considered to be raised within the outer block, and the else block will run.

If function 2 raises a TypeError then since the inner try/except re-raises it; it is considered an exception raised by the inner block and the else block won’t run.

Q:5

Can one block of except statements handle multiple exception?

You can also have one except block handle multiple exceptions. To do this, use parentheses. Without that, the interpreter will return a syntax error.

```
>>> try:
    print('10'+10)
    print(1/0)
except (TypeError,ZeroDivisionError):
    print("Invalid input")
```

Output

Invalid input

Q:6

When is the finally block executed?

The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling — it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

The try block of the writeList method that you've been working with here opens a PrintWriter. The program should close that stream before exiting the writeList method. This poses a somewhat complicated problem because writeList's try block can exit in one of three ways.

1. The new FileWriter statement fails and throws an IOException.
2. The list.get(i) statement fails and throws an IndexOutOfBoundsException.
3. Everything succeeds and the try block exits normally.

The runtime system always executes the statements within the finally block regardless of what happens within the try block. So it's the perfect place to perform cleanup.

The following finally block for the writeList method cleans up and then closes the PrintWriter and FileWriter.

```
finally {
    if (out != null) {
        System.out.println("Closing PrintWriter");
        out.close();
    } else {
        System.out.println("PrintWriter not open");
    }
    if (f != null) {
            System.out.println("Closing FileWriter");
            f.close();
         }
}
```

Q:7

What is Instantiation in terms of OOP terminology?

instantiation is the conception of a real-world example or the specific realization of an abstraction or model, e.g. B. a class of objects or a computer process. Instantiation means creating such an instance by, for example, identifying a particular difference of an object within a category, giving it a name, and placing it in a physical location.

The potential confusion for people new to OO is that Instantiation is what happens to create an object, yet this is a class-to-class relationship no objects are involved. A parameterized class cannot have objects instantiated from it unless it is first instantiated itself (hence the confusing name for this type of relationship).

Instantiation is creating an instance of an object in an object-oriented software design (OOP) verbal. An instantiated object is given a name and is create in memory or on disk using the structure described in a class declaration.

What Does Instantiation Mean?

In computer science, instantiation (verb) and instantiation (noun) refer to the creation of an object (or “instance” of a particular class) in an object-oriented programming (OOP) language. An instantiated object is called and created in recollection or on disk by reference to a class declaration. It is also limited or defined by its attributes. Resources are allocated, assigned somewhere in the overall set of codebases, and programmers play a role in its construction.

Object instantiation uses different methods and terminology in other programming environments: for example, in C++ and similar languages, What Is Instantiation In Terms Of Oop Terminology class means creating an object, whereas, in Java, it means creating an object. Initializing a class creates a specific category. The results are the same in both languages (executables), but the way to get there is slightly different.

Technologists may talk about instantiation differently depending on their environments and the protocols and methods they use when working with active classes to create objects. For example, starting a virtual server involves virtualizing each server’s predefined features (disk space, allocated RAM, operating system type, installed software).