# Python Coding Interview Questions

### 1) What is the best way to debug a Python program?

This command can be used to debug a Python program.

1. **Python -m pdb Python-script.py**

### 2) What does the Python keyword imply?

In Python, we can use the <yield> keyword to convert any Python function into a Python generator. Yields function similarly to a conventional return keyword. However, it will always return a generator object. A function can also use the <yield> keyword multiple times.

****Code****

1. **def** creating\_gen(index):
2. months = ['jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec']
3. **yield** months[index]
4. **yield** months[index+2]
5. next\_month = creating\_gen(3)
6. **print**(next(next\_month), next(next\_month))

****Output:****

PauseNext

Mute

Current TimeÂ 0:01

/

DurationÂ 18:10

Loaded: 0.37%

Â

Fullscreen

apr jun

****Explanation:****

The given code characterizes a generator capability named creating\_gen that accepts a record as a contention and yields two sequential months from a predefined list. The generator is then introduced with an underlying file of 3, making a generator object called next\_month. The following() capability is utilized two times to acquire the following two yielded values from the generator, which address two sequential months. At long last, the code prints these two months. Fundamentally, the generator effectively delivers sets of months, and the code exhibits the successive extraction of these month matches utilizing the following() capability.

### 3) How can I make a tuple out of a list?

We can transform a list into a tuple using the Python tuple() method. Since a tuple is immutable, we can't update the list after it has been converted to a tuple.

****Code****

1. month = ['jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec']
2. converting\_list = tuple(month)
3. **print**(converting\_list)
4. **print**(type(converting\_list))

****Output:****

('jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec')

<class 'tuple'>

****Explanation:****

The code scrap instates a rundown called month containing the names of the a year. It then changes over this rundown into a tuple utilizing the tuple() capability and relegates the outcome to the variable converting\_list. The code prints the changed over tuple and its sort utilizing print(converting\_list) and print(type(converting\_list)), individually. In outline, this code exhibits the change of a rundown of months into a tuple, displaying the subsequent tuple and its information type. Tuples, similar to records, are iterable and requested assortments, yet they are changeless, meaning their components can't be altered after creation.

### 4) What exactly is a NumPy array?

NumPy arrays are much more versatile than Python lists. Reading and writing objects are quicker and more efficient using NumPy arrays.

### 5) In Python, in what ways can you make an empty NumPy array?

In Python, there are two ways to build an empty array:

****Code****

1. **import** numpy
2. #method 1
3. array\_1 = numpy.array([])
4. **print**(array\_1)
5. #method 2
6. array\_2 = numpy.empty(shape=(3,3))
7. **print**(array\_2)

****Output:****

[]

[[9.03420200e-308 7.54982336e-308 0.00000000e+000]

[0.00000000e+000 0.00000000e+000 0.00000000e+000]

[0.00000000e+000 0.00000000e+000 1.20953760e-311]]

****Explanation:****

The gave code shows two unique strategies for making NumPy exhibits. In the main strategy, an unfilled cluster called array\_1 is made utilizing numpy.array([]), bringing about an underlying void exhibit. In the subsequent technique, array\_2 is made utilizing numpy.empty(shape=(3,3)), which creates a 3x3 exhibit loaded up with uninitialized values. The code then prints the two exhibits, displaying the result. In synopsis, the code outlines how to make a vacant exhibit and one more with uninitialized values involving NumPy in Python.

ADVERTISEMENT

### 6) In Python, what is a negative index?

In Arrays and Lists, Python contains a unique feature called negative indexing. Python starts indexing from the beginning of an array or list in a positive integer but reads items from the ending of an array or list in a negative index.

### 7) Tell the output of the following code?

****Code****

1. **import** array
2. a = [4, 6, 8, 3, 1, 7]
3. **print**(a[-3])
4. **print**(a[-5])
5. **print**(a[-1])

****Output:****

ADVERTISEMENT

3

6

7

****Explanation:****

The given code utilizes a standard Python list named 'a' containing components [4, 6, 8, 3, 1, 7]. It shows the utilization of negative records to get to components from the finish of the rundown. The articulations a[-3], a[-5], and a[-1] recover the third-to-endure, fifth-to-endlessly last components of the rundown, individually. At the point when printed, these qualities are 3, 6, and 7, exhibiting the capacity to get to components in switch request utilizing negative files. Generally, negative ordering gives a helpful method for getting to components comparative with the finish of the rundown.

### 8) What is the Python data type SET, and how can I use it?

ADVERTISEMENT

"set" is a Python data type which is a sort of collection. Since Python 2.4, it's been a part of the language. A set is a collection of distinct and immutable items that are not in any particular sequence.

### 9) In Python, how do you create random numbers?

We can create random data in Python utilizing several functions. They are as follows:

ADVERTISEMENT

* ****random() -**** This instruction gives a floating-point value ranging from 0 to 1.
* ****uniform(X, Y) -**** This function gives a floating-point value in the X and Y range.
* ****randint(X, Y) -**** This function gives a random integer between X and Y values.

### 10) How do you print the summation of all the numbers from 1 to 101?

Using this program, we can display the summation of all numbers from 1 to 101:

****Code****

1. Print( sum(range(1,102)) )

****Output:****

5151

****Explanation:****

ADVERTISEMENT

The gave code computes the amount of numbers from 1 to 101 utilizing the aggregate() capability and the reach() capability. The articulation sum(range(1, 102)) creates a succession of numbers from 1 to 101 (comprehensive) and afterward computes their total. The outcome is printed utilizing the print() capability. In synopsis, this code proficiently figures and shows the amount of the whole numbers from 1 to 101.

### 11) In a function, how do you create a global variable?

We can create a global variable by designating it as global within every function that assigns to it; we can utilize it in other functions:

****Code****

1. global\_var = 0
2. **def** modify\_global\_var():
3. **global** global\_var # Setting global\_var as a global variable
4. global\_var = 10
5. **def** printing\_global\_var():
6. **print**(global\_var) # There is no need to declare global variable
7. modify\_global\_var()
8. printing\_global\_var() # Prints 10

****Output:****

10

****Explanation:****

The code characterizes a worldwide variable named global\_var introduced to 0. It then characterizes two capabilities: modify\_global\_var() sets the worldwide variable to 10 utilizing the worldwide catchphrase, and printing\_global\_var() prints the worldwide variable. Subsequent to calling modify\_global\_var(), the worldwide variable is changed to 10, and when printing\_global\_var() is executed, it prints the refreshed worth. In outline, the code shows the utilization of the worldwide watchword to alter a worldwide variable inside a capability, permitting changes to engender outside the capability's extension.

### 12) Is it possible to construct a Python program that calculates the mean of numbers in a list?

Calculating the Average of Numbers in Python:

ADVERTISEMENT

****Code****

1. n = int(input("Number of Elements to take average of: "))
2. l=[]
3. **for** i **in** range(1,n+1):
4. element = int(input("Enter the element: "))
5. l.append(element)
6. average = sum(l)/n
7. **print**("Average of the elements in list",round(average,2))

****Output:****

Number of Elements to take average of: 4

Enter the element: 5

Enter the element: 25

Enter the element: 74

Enter the element: 24

Average of the elements in list 32.0

****Explanation:****

This code works out the normal of a rundown of numbers given by the client. It takes client input for the quantity of components (n) and iteratively prompts for every component. The components are put away in a rundown l, and the normal is determined by separating the amount of the components by the complete number of components. The outcome is then printed with an adjusted worth to two decimal spots. In rundown, the code productively processes and shows the normal of a client characterized rundown of numbers.

### 13) Is it possible to build a Python program that reverses a number?

Python program to reverse number:

****Code****

ADVERTISEMENT

1. n = int(input("Enter number: "))
3. reverse = 0
5. **while**(n>0):
6. digit = n%10
7. reverse = reverse\*10+digit
8. n=n//10
9. **print**("The reverse of the number:",reverse)

****Output:****

Enter number: 35257

The reverse of the number: 75253

****Explanation:****

This code piece switches a number entered by the client. It takes a number info (n) and instates a variable converse to 0. The code then, at that point, utilizes some time circle to iteratively separate the last digit of the number (digit) and constructs the switched number (turn around) by duplicating the ongoing worth of opposite by 10 and adding the extricated digit. The circle go on until all digits are handled. At last, the switched number is printed. Basically, the code proficiently works out and shows the opposite of a client input whole number utilizing some time circle.

### 14) In Python, what is the distinction between a list and a tuple?

|  |  |  |
| --- | --- | --- |
| **S. No.** | **List** | **Tuple** |
| 1. | Lists are editable, which means that we can change them. | Tuples (which are just lists that we cannot alter) are immutable. |
| 2. | Lists are comparatively slower | Tuples are more efficient than lists. |
| 3. | Syntax: list1 = [100, 'Itika', 200] | Syntax: tup1 = (100, 'Itika', 200) |

### 15) Is Python programming language or scripting language?

ADVERTISEMENT

Python is a programming language that is frequently utilized for prearranging because of its straightforwardness and coherence. Be that as it may, it is a flexible language fit for taking care of a great many undertakings, from little scripts to huge scope programming improvement.

### 16) Python is an interpreted programming language. Explain.

Python is a deciphered programming language, significance its code is executed line by line by the Python translator without a different gathering step. This prompts more prominent adaptability, simplicity of improvement, and movability, though with possibly somewhat more slow execution contrasted with gathered dialects.

### 17) What is the meaning of pep 8?

PEP 8 is a style guide for Python code, illustrating best practices and shows to further develop code coherence. It covers viewpoints like space, naming shows, whitespace, and that's only the tip of the iceberg. PEP means "Python Improvement Proposition," and Energy 8 explicitly centers around code designing to upgrade consistency and viability across Python projects. Adhering to Enthusiasm 8 rules is prescribed to guarantee spotless, intelligible, and steady Python code.

### 18) What are the advantages of Python?

The advantages of utilizing Python are as follows:

* ****Simple to understand and utilize-**** Python is a powerful language of programming that is simple to learn, read, and write.
* ****Interpreted language-**** Python is an interpreted language, which means it runs the program line by line & pauses if any line contains an error.
* ****Dynamically typed-**** when coding, the programmer does not set data types to variables. During execution, it is automatically assigned.
* Python is free and open-source to use and share. It's free and open source.
* ****Extensive library support-**** Python has a large library of functions that can perform practically any task. It also allows you to use Python Package Manager to import additional packages (pip).
* Python applications are portable and can execute on any system without modification.
* Python's data structures are easy to understand.
* It allows for additional functionality while requiring less coding.

### 19) In Python, what are decorators?

Decorators are just employed to add certain layout patterns to a method without affecting the structure of the function. Typically, decorators are identified before the event they will be improving. We should first define a decorator's function before using it. The function to which We will implement the decorator's function is then written, and the decorator function is simply positioned above it. In this instance, the @ symbol comes preceding the decorator.

### 20) How is a Python Dictionary different from List comprehensions?

Dictionary & list comprehensions are yet another means of defining dictionaries and lists in a simple manner.

This is example of list comprehension

****Code****

1. list\_comp = [i **for** i **in** range(4)]
2. **print**(list\_comp)

****Output:****

[0, 1, 2, 3]

This is example of dictionary

****Explanation:****

The code uses a rundown understanding to make a rundown called list\_comp containing numbers from 0 to 3. The print(list\_comp) proclamation yields the created list. Generally, it shows a succinct method for developing and show a rundown of successive numbers in Python.

****Code****

1. dictt = {i : i+2 **for** i **in** range(10)}
2. **print**(dictt)

****Output:****

{0: 2, 1: 3, 2: 4, 3: 5, 4: 6, 5: 7, 6: 8, 7: 9, 8: 10, 9: 11}

****Explanation:****

This code scrap utilizes a word reference understanding to make a word reference called dictt. It coordinates each number from 0 to 9 with its comparing esteem increased by 2. The print(dictt) proclamation yields the subsequent word reference, outlining an effective method for creating and show a word reference with key-esteem matches in Python. In synopsis, the code features the brief development and printing of a word reference utilizing an understanding.

### 21) What is the most prevalent Python built-in data types?

****Numbers-**** Integers, complex numbers, and floating points are Python's most prevalent built-in data structures. For example, 1, 8.1, 3+6i.

****List-**** A list is a collection of objects that are arranged in a specific order. A list's components could be of multiple data kinds. For example, [[10,'itika',7] .4]

****Tuple-**** It's also a set of items in a specific order. Tuples, not like lists, are immutable, meaning we cannot modify them. For example, (7,'itika',2)

****String-**** A string is a collection of characters. Single or double quotations are used to declare them. "Itika," "She is learning coding through Anupriyan", and so on.

****Set-**** A set is a group of unrelated elements which are not in any particular sequence. (2, 3, 4, 5)

****Dictionary-**** A dictionary is a collection of key and value combinations in which each value may be accessed by its key. The sequence of the items is irrelevant. For example, {3:'ape', 6:'monkey'}

****Boolean-**** True and False is indeed the two possible boolean values.

### 22) What's the distinction between .py and.pyc files?

The Python code we save is contained in the .py files. The .pyc files are created when the program is integrated into the current program from some other source. This file contains the bytecode for the Python files that we imported. The interpreter reduces processing time if we transform the source files having the format of .py files to .pyc files.

### 23) How is a local variable different from a global variable?

****Global Variables:**** Global variables are those that have been declared outside of a function. The scope outside of the function is known as global space. Any program function has access to these variables.

****Local Variables:**** Any variable declared inside a function is referred to as a local variable. This variable does not exist in the global domain; it only exists locally.

****Code****

1. # Python program to show how global variables and local variables are different
2. var = 56
3. # Creating a function
4. **def** addition():
5. var1 = 7
6. c = var + var1
7. **print**("In local scope: ", var1)
8. **print**("Adding a global scope and a local scope variable: ", c)
9. addition()
10. **print**("In global scope: ", var)

****Output:****

In local scope: 7

Adding a global scope and a local scope variable: 63

In global scope: 56

****Explanation:****

The code exhibits the qualification among worldwide and nearby factors in Python. It instates a worldwide variable var with a worth of 56. Inside the function expansion(), a neighborhood variable var1 is made with a worth of 7. The function adds the worldwide variable var and the nearby factor var1, and the outcomes are printed inside the neighborhood scope. Outside the function, the worldwide variable var is printed. The code outlines how factors pronounced inside a function have neighborhood scope, and their qualities don't influence worldwide factors. In outline, it features the detachment among worldwide and neighborhood scopes in Python.

It will generate an error if you attempt to access the local variable exterior of the function addition().

### 24) What is the distinction between Python Arrays and Python Lists?

In Python, arrays and lists both store data similarly. On the other hand, arrays can only have a single data type element, while lists can contain any data type component.

****Code****

1. # Python program to show the difference between a list and an array
3. # Importing array module
4. **import** array as arr
6. # Creating an array and a list
7. array\_1 = arr.array("i", [3, 6, 2, 7, 9, 5])
8. list\_1 = [4, 'Interview', 7.20]
10. **print**(array\_1)
11. **print**(list\_1)
13. # Trying to create an array with multiple data types
14. **try**:
15. array\_2 = arr.array("i", [3, 7, 3, "Interview"])
16. **except** Exception as e:
17. **print**(e)

****Output:****

array('i', [3, 6, 2, 7, 9, 5])

[4, 'Interview', 7.2]

'str' object cannot be interpreted as an integer

****Explanation:****

The code uses the cluster module to make a number exhibit named array\_1 and a rundown named list\_1. It prints both the exhibit and the rundown, displaying the adaptability of Python information structures. Nonetheless, while endeavoring to make another exhibit (array\_2) with blended information types, the code experiences an exemption because of the predefined type code "I" for numbers. The blunder is gotten and printed utilizing an attempt with the exception of block. In synopsis, the code features the creation and printing of a whole number cluster and a rundown, and it exhibits the impediment of making exhibits with blended information types utilizing the cluster module.

### 25) What exactly is \_\_init\_\_?

In Python, \_\_init\_\_ is a function or function Object() { [native code] }. When a new object/instance of a class is created, this function is automatically called to reserve memory. The \_\_init\_\_ method is available in all classes.

Here's an instance of how to put it to good use.

****Code****

1. # Python program to explain \_\_init\_\_
3. **class** Student:
4. **def** \_\_init\_\_(self, st\_name, st\_class, st\_marks):
5. self.st\_name = st\_name
6. self.st\_class = st\_class
7. self.st\_marks = 67
8. S1 = Student("Itika", 10, 67)
9. **print**(S1.st\_name)
10. **print**(S1.st\_class)

****Output:****

Itika

10

****Explanation:****

The code characterizes a Python class named Understudy with a \_\_init\_\_ strategy, which fills in as the constructor. The \_\_init\_\_ technique introduces the qualities st\_name, st\_class, and st\_marks for each example of the class. At the point when an occurrence S1 is made with explicit qualities for these traits, the \_\_init\_\_ strategy is naturally called to set the underlying qualities. The code then, at that point, prints the st\_name and st\_class qualities of the occurrence S1. In synopsis, the code exhibits the utilization of the \_\_init\_\_ technique to instate credits while making cases of a class, displaying the capacity to tweak object instatement in Python.

### 26) What is a lambda function, and how does it work?

A lambda function is a type of nameless function. This method can take as many parameters as you want but a single statement.

****Code****

1. # Python program to show how to use lambda functions
3. # Creating a lambda function for addition
4. sum\_ = **lambda** x, y, z : x + y + z
5. **print**("Sum using lambda function is: ", sum\_(4, 6, 8))

****Output:****

Sum using lambda function is: 18

****Explanation:****

The code defines a lambda function named sum\_ for expansion, taking three boundaries (x, y, and z). The lambda function ascertains the amount of these boundaries and prints the outcome utilizing the print() proclamation. Basically, it shows the succinct creation and utilization of a lambda function for expansion, giving a shorthand method for characterizing little, unknown capabilities in Python. The result of the code is the amount of the qualities 4, 6, and 8, displaying the usefulness of the lambda function.

### 27) In Python, what is the self?

A self is a class instance or object. This is explicitly supplied as the initial argument in Python. However, in Java, in which it is optional, that's not the case. Local variables make it easy to differentiate between a class's methods and attributes.

In the init method of a class, the self variable corresponds to the freshly generated objects, whereas it relates to the entity whose method can be called in the class's other methods.

### 28) How do these commands work: break, pass and continue?

|  |  |
| --- | --- |
| ****Break**** | The loop is terminated when a criterion is fulfilled, and control is passed to the subsequent statement. |
| ****Pass**** | You can use this when you need a code block syntactically correct but don't want to run it. This is a null action in essence. When it is run, nothing takes place. |
| ****Continue**** | When a specified criteria is fulfilled, the control is moved to the start of the loop, allowing some parts of the loop currently in execution to be skipped. |

### 29) In Python, how would you randomise the elements of a list while it's running?

Consider the following scenario:

****Code****

1. # Python program to show to randomise elements of a list
3. # Importing the random module
4. **import** random
5. list\_ = ["Python", "Interview", "Questions", "Randomise", "List"]
6. **print**("Original list: ", list\_)
7. random.shuffle(list\_)
8. **print**("After randomising the list: ", list\_)

****Output:****

Original list: ['Python', 'Interview', 'Questions', 'Randomise', 'List']

After randomising the list: ['List', 'Interview', 'Python', 'Randomise', 'Questions']

****Explanation:****

The code utilizes the random.shuffle() function to haphazardly reorder the components of a rundown named list\_. It first prints the first rundown and afterward applies the random.shuffle() function to haphazardly rearrange the components. The subsequent rundown is printed, exhibiting the successful randomization of the first rundown. In synopsis, the code features how to utilize the random.shuffle() function to rearrange the components of a rundown in an irregular request, giving a basic method for acquainting haphazardness with the grouping of components.

### 30) What is the difference between pickling and unpickling?

The Pickle module takes any Python object and then transforms it into the representation of a string, which it then dumps into a file using the dump method. Unpickling is the procedure of recovering authentic Python items from a saved string representation.

### 31) What method will you use to turn the string's all characters into lowercase letters?

The lower() function could be used to reduce a string to lowercase.

****Code****

1. # Python program to show how to convert a string to lower case
3. string = Anupriyan
4. **print**(string.lower())

****Output:****

Anupriyan

****Explanation:****

The code changes the string 'Anupriyan' over completely to lowercase utilizing the lower() strategy and prints the outcome. This shows a straightforward method for changing the instance of characters in a string in Python.

### 32) How do you comment on multiple lines at once in Python?

Multi-line comments span many lines. A # must precede all lines that we will comment on. You could also use a convenient alternative to comment on several lines. All you have to do is press down the ctrl key, hold it, and click the left mouse key in every area where you need a # symbol to appear, then write a # once. This will add a comment to the lines wherever you insert your cursor.

### 33) In Python, what are docstrings?

Docstrings stands for documentation strings, which are not just comments. We enclose the docstrings in triple quotation marks. They are not allocated to any variable, and, as a result, they can also be used as comments.

****Code****

1. # Python program to show how to write a docstring
3. """
4. This is a docstring.
5. We write docstrings to explain a program.
6. This program will multiply two numbers and then display the output.
7. """
8. a = 39
9. b = 45
10. c = a \* b
11. **print**("Result of multiplication: ", c)

****Output:****

Result of multiplication: 1755

****Explanation:****

The given code duplicates two numbers, 39 and 45, and allocates the outcome to the variable c. The print() proclamation then, at that point, shows the consequence of the augmentation, which is the result of these two numbers. In outline, the code proficiently works out and prints the duplication of 39 and 45, displaying an essential number juggling activity in Python.

### 34) Explain the split(), sub(), and subn() methods of the Python "re" module.

Python's "re" module provides three ways for modifying strings. They are as follows:

split() "splits" a string into a list using a regex pattern.

sub() finds all substrings that match the regex pattern given by us. Then it replaces that substring with the string provided.

subn() is analogous to sub() in that it gives the new string and the number of replacements.

### 35) What is the best way to add items to a Python array?

We can use the append(), extend(), as well as insert (i,x) methods to add items to an array.

****Code****

1. # Python program to show how to add elements to an array
3. array = arr.array('d', [1 , 2 ,3] )
4. array.append(8) # appending will add an element to the end of the array
5. **print**(array)
6. array.extend([4,6,9]) # extending will add elements by looping through the given iterable
7. **print**(array)
8. array.insert(2, 9) # inserting will add the element at the specified index
9. **print**(array)

****Output:****

array('d', [1.0, 2.0, 3.0, 8.0])

array('d', [1.0, 2.0, 3.0, 8.0, 4.0, 6.0, 9.0])

array('d', [1.0, 2.0, 9.0, 3.0, 8.0, 4.0, 6.0, 9.0])

****Explanation:****

The code utilizes the exhibit module to make a twofold accuracy drifting point cluster named 'exhibit' introduced with values [1.0, 2.0, 3.0]. It then, at that point, shows three unique strategies to adjust the cluster:

The append() technique adds the worth 8 to the furthest limit of the cluster.

The extend() technique attaches various components (4, 6, 9) to the cluster.

The insert() technique adds the worth 9 at list 2 in the exhibit.

Every adjustment is trailed by printing the refreshed exhibit, displaying different ways of controlling cluster components in Python.

### 36) What is the best way to remove values from a Python array?

The pop() or remove() methods can be used to remove array elements. The distinction between these 2 methods is that the first returns the removed value, while the second does not.

****Code****

1. # Python program to show how to remove elements from a Python array
3. array = arr.array('d', [1, 3, 8, 1, 4, 8, 2, 4])
4. **print**(array.pop()) # By default it will remove and return the last element of the array
5. **print**(array.pop(5)) # It will return and remove the element present at 5th index
6. array.remove(1) # It will remove only the first occurrence of the element - 1 from the array
7. **print**(array)

****Output:****

4.0

8.0

array('d', [3.0, 8.0, 1.0, 4.0, 2.0])

****Explanation:****

The code introduces a twofold accuracy drifting point exhibit named 'cluster' with values [1.0, 3.0, 8.0, 1.0, 4.0, 8.0, 2.0, 4.0]. It then exhibits three distinct strategies for eliminating components from the cluster:

The pop() technique, without a file indicated, eliminates and returns the last component of the cluster.

The pop(5) strategy eliminates and returns the component at the fifth list.

The remove(1) strategy dispenses with the principal event of the worth 1 from the exhibit.

Every expulsion activity is trailed by printing the refreshed exhibit, showing different ways of erasing components from a Python cluster.

### 37) In Python, what is monkey patching?

The phrase "monkey patch" in Python exclusively references run-time dynamic alterations to a module.

****Code****

1. **class** My\_Class:
2. **def** f(self):
3. **print**("f()")

The monkey-patch testing will then be done as follows:

1. **import** monk
2. **def** monkey\_f(self):
3. **print** ("we  are calling monkey\_f()")
5. # changing address of func
6. monk.My\_Class.func = monkey\_f
7. object\_ = monk.My\_Class()
9. object\_.func()

****Explanation:****

In the given code code, a class named My\_Class is characterized with a strategy f(), which basically prints the string "f()". Following this, a monkey-fix testing situation is illustrated. Monkey fixing includes powerfully changing or broadening code at runtime, commonly to adjust the way of behaving of existing classes or works.

For this situation, an outer module named priest is imported, and another capability named monkey\_f is characterized. This new capability prints the string "we are calling monkey\_f()". The monkey fixing itself is performed by doling out the monkey\_f capability to the func characteristic of the My\_Class class in the priest module. This really replaces the first f() technique with the recently characterized monkey\_f().

A while later, an example of My\_Class is made and doled out to the variable object\_. When the func() technique is approached this article, it currently alludes to the monkey-fixed monkey\_f() capability. Subsequently, the result of the code will be "we are calling monkey\_f()" rather than the first "f()".

### 38) In Python, how do you make an empty class?

An empty class has no statements contained within its blocks. It can be produced by using the pass keyword. But you can make an object outside the class. The PASS statement doesn't do anything in Python.

****Code****

1. # Python program to show how to make an empty class
3. **class** my\_class:
4. **pass**
5. object\_ = my\_class()
6. object\_.name = "Anupriyan"
7. **print**("Name = ", object\_.name)

****Output:****

Name =

Anupriyan

****Explanation:****

The code characterizes a basic class my\_class, makes a case of it, and powerfully adds a property named name with the worth "Anupriyan" to the example. It then, at that point, prints the worth of the name trait, outlining the unique idea of characteristic task in Python classes.

### 39) Write a Python script to implement the Bubble sort algorithm.

****Code****

1. # Python program to show how to implement bubble sort
3. **def** bubble\_Sort(array):
4. n = len(array)
6. **for** i **in** range(n-1):
8. **for** j **in** range(0, n-i-1):
10. **if** array[j] > array[j + 1] :
11. array[j], array[j + 1] = array[j + 1], array[j]
13. **print**(array)
15. #example array
16. arr = [23, 14, 64, 13, 64, 23, 86]
18. bubble\_Sort(arr)

****Output:****

[13, 14, 23, 23, 64, 64, 86]

****Explanation:****

The code characterizes an essential execution of the Air pocket Sort calculation to sort a cluster in climbing request. The bubble\_Sort function accepts an exhibit as information and repeats through its components, looking at nearby coordinates and trading them on the off chance that they are all mismatched. This cycle is rehashed for every component until the whole exhibit is arranged. The code exhibits the arranging of a model cluster [23, 14, 64, 13, 64, 23, 86] utilizing the bubble\_Sort function and prints the arranged exhibit. In synopsis, the code shows a straightforward Air pocket Sort calculation for arranging an exhibit in climbing request.

### 40) In Python, create a program that generates a Fibonacci sequence.

****Code****

1. #taking number of terms to print the series
2. n = 9
3. first = 0 #first value of series
4. second = 1 #second value of series
5. series = [first, second]
6. **if** n == 0:
8. **print**("The required fibonacci series is",first)
9. **else**:
11. **for** i **in** range(0,n-2):
13. num = series[i] + series[i+1]
15. series.append(num)
16. **print**(series)

****Output:****

[0, 1, 1, 2, 3, 5, 8, 13, 21]

****Explanation:****

The given Python code means to produce a Fibonacci series in light of the quantity of terms determined by the variable n. It introduces the initial two upsides of the series (first and second) as 0 and 1, separately, and stores them in a rundown named series. The code then, at that point, checks if the quantity of terms (n) is zero; assuming this is the case, it prints the primary worth of the series. In any case, it enters a circle that repeats n - twice, computing the following term in the series by adding the past two terms and adding it to the series list. At long last, the created Fibonacci series is printed. In any case, there is a slight issue with the circle end condition, as it probably won't create the right number of terms in the event that n is under 2. Generally, the code expects to produce a Fibonacci series in view of the predetermined number of terms, utilizing a rundown to store the series components and a circle to register them.

### 41) Make a Python script that checks if a given number is prime.

****Code****

1. # Python program to check if a number is prime or not
3. # Declaring a variable
4. n = 37
5. **if** n == 2:
6. **print**("2 is a prime number")
8. **if** n != 1:
9. **for** i **in** range(2, n):
10. **if** n % i == 0:
11. **print**("The given number is a composite number")
12. **break**
13. **if** i == n-1:
14. **print**("The given number is a prime number")
15. **else**:
16. **print**("1 is not a prime number")

****Output:****

The given number is a prime number

****Explanation:****

The code decides if a given number 'n' is prime or composite. On the off chance that 'n' is 2, it straightforwardly expresses that it is an indivisible number. For different cases, it actually takes a look at detachability by numbers from 2 to 'n-1'. If 'n' is separable by any number here, it is viewed as a composite number, and the circle breaks. On the off chance that 'n' isn't distinguishable by any number, it is recognized as an indivisible number. On the off chance that 'n' is 1, it expresses that 1 is definitely not an indivisible number. In rundown, the code really arranges the given number as one or the other prime or composite and handles the extraordinary instance of 1.

### 42) Create a Python program that checks if a given sequence is a Palindrome.

Below is the program.

****Code****

1. # Python program to check if the given string is a palindrome
3. # Creating a string
4. sequence = 'abjucujba'
5. # Reversing the string
6. reverse = sequence[::-1]
8. # Checking if the string is a palindrome
9. **if** reverse == sequence:
10. **print**("The sequence is a palindrome")
11. **else**:
12. **print**("The sequence is not a palindrome")

****Output:****

The sequence is a palindrome

****Explanation:****

The code checks if a given string 'succession' is a palindrome. It first switches the string utilizing cutting documentation (sequence[::- 1]) and stores the outcome in the variable 'turn around'. Then, at that point, it contrasts 'opposite' and the first 'arrangement'. Assuming they are equivalent, the string is recognized as a palindrome, and the code prints that it is a palindrome. In any case, it prints that the string isn't a palindrome. In rundown, the code proficiently decides if the info string is a palindrome or not.

### 43) In a NumPy array, how do I extract the indices of N maximum values?

Using the code below.

****Code****

1. **import** numpy
2. array = numpy.array([4, 8, 4, 9, 2])
3. **print**(array.argsort()[-2:][::-1])

****Output:****

[3 1]

****Explanation:****

The code uses the NumPy library to make an exhibit named 'cluster' containing whole numbers. It then, at that point, applies the argsort() function, which returns the records that would sort the cluster. The code chooses the last two records (files of the two biggest components) utilizing [-2:] and turns around the request with [::- 1]. At long last, it prints the files of the two biggest components in sliding request. Basically, the code proficiently distinguishes and shows the records of the two biggest components in the NumPy exhibit.

### 44) Using Python/ NumPy, write code to compute percentiles?

The following method can be used to determine percentiles.

****Code****

1. **import** numpy
2. array = numpy.array([3, 6, 1, 6, 5, 2])
3. percentile = numpy.percentile(array, 45) #Returns 45th percentile
4. **print**(percentile)

****Output:****

3.5

****Explanation:****

The code utilizes the NumPy library to make an exhibit named 'cluster' containing numbers. It then, at that point, utilizes the percentile() function to work out the 45th percentile of the exhibit, addressing the worth beneath which 45% of the information falls. The outcome is printed, exhibiting the way that NumPy works with the calculation of percentiles for mathematical information. In rundown, the code proficiently ascertains and shows the 45th percentile of the given exhibit.

### 45) Write a Python program to determine whether a number is binary.

Following is the code.

****Code****

1. num = 1101001
2. **while**(num > 0):
3. l = num % 10
5. **if** l!=0 **and** l!=1:
7. **print**("given number is not binary")
9. **break**
11. num = num // 10
13. **if** num == 0:
15. **print**("given number is binary")

****Output:****

given number is binary

****Explanation:****

The code checks if a given number, 'num', is in paired design. It emphasizes through every digit of the number utilizing some time circle. Inside the circle, it checks assuming every digit is either 0 or 1. Assuming any digit other than 0 or 1 is experienced, it prints that the given number isn't in twofold configuration and breaks unaware of everything going on. Assuming the circle finishes without experiencing some other digit, it prints that the given number is in twofold configuration. The code effectively decides the double idea of the info number.

### 46) Using the Iterative technique, calculate factorial in Python.

Following is the code.

****Code****

1. num = 12
2. fact = 1
3. **if** num < 0:
5. **print**("Since number is negative factorial cannot be calculated")
6. **elif** num == 0:
8. **print**("Factorial of 0 is 1")
9. **else**:
11. **for** f **in** range(1, num + 1):
13. fact = fact \* i
15. **print**("Factorial of",num ,"is",fact)

****Output:****

Factorial of 12 is 1449225352009601191936

****Explanation:****

The code computes the factorial of a given number 'num'. It instates a variable 'truth' to 1. It first checks in the event that the number is negative and prints a proper message. Assuming the number is zero, it prints that the factorial is 1. For positive numbers, it utilizes a for circle to repeat from 1 to the given number, refreshing the factorial at each step. The end-product is printed. The code productively registers and shows the factorial of a non-negative whole number. Note: There's a grammatical error in the know variable; it ought to be 'for I in range(1, num + 1)' rather than 'for f in range(1, num + 1)'.

### 47) Compute the LCM of two given numbers using Python code.

Following is the code

****Code****

1. num\_1 = 24
2. num\_2 = 92
3. **if** num\_1 > num\_2:
5. greater\_num = num\_1
6. **else**:
8. greater\_num = num\_2
9. **while**(True):
11. **if**((greater\_num % num\_1 == 0) **and** (greater\_num % num\_2 == 0)):
13. lcm = greater\_num
15. **break**
17. greater\_num += 1
18. **print**("LCM of", num\_1, "and", num\_2, "=", greater\_num)

****Output:****

LCM of 24 and 92 = 552

****Explanation:****

The code ascertains the Most un-Normal Various (LCM) of two numbers, 'num\_1' and 'num\_2'. It begins by deciding the more prominent of the two numbers and afterward enters some time circle. Inside the circle, it checks if the current 'greater\_num' is separable by both 'num\_1' and 'num\_2'. If valid, it doles out 'greater\_num' to the LCM and breaks unaware of everything going on. If not, it increases 'greater\_num' and proceeds with the circle. The last LCM is printed. The code proficiently processes and shows the LCM of two given numbers.

### 48) Write a Python program that removes vowels from a string.

Following is the code.

****Code****

1. array = [23, 12, 5, 24, 23, 76, 86, 24, 86, 24, 75]
3. **print**("Reverse order of array is")
4. # Reversing the given array
5. **for** i **in** range(len(array)-1, -1, -1):
7. **print**(array[i], end=' ')

****Output:****

Reverse order of array is

75 24 86 24 86 76 23 24 5 12 23

### 49) Write a Python program that rotates an array by two positions to the right.

Following is the code.

****Code****

1. string = Anupriyan
2. result=''
3. **for** s **in** string:
4. **if** s **in** ('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'):
5. s = ''
6. result += s
7. **print**("Required string without vowels is:", result)

****Output:****

Required string without vowels is: Jvtpnt

****Explanation:****

The code eliminates vowels from a given 'string' and stores the outcome in 'result'. It emphasizes through each person in the string utilizing a for circle, and in the event that the person is a vowel (either lowercase or capitalized), it replaces it with an unfilled string. The adjusted characters are then connected to shape the last string 'result', which is printed as the result. This code exhibits a direct way to deal with kill vowels from a string in Python.

### 50) Write a Python code to reverse a given list.

Following is the code.

1. array = [23, 12, 5, 24, 23, 76, 86, 24, 86, 24, 75]
3. **print**("Reverse order of array is")
4. # Reversing the given array
5. **for** i **in** range(len(array)-1, -1, -1):
7. **print**(array[i], end=' ')

****Output:****

Reverse order of array is

75 24 86 24 86 76 23 24 5 12 23

****Explanation:****

The code prints the converse request of a given cluster. It utilizes a for circle to emphasize through the exhibit backward, beginning from the last component and decrementing the file. The circle prints every component isolated by a space, bringing about the switched request of the first exhibit. In outline, the code effectively shows the converse request of the given cluster.