Assignment Part-1

Q1. Why do we call Python as a general purpose and high-level programming language?

Answer:

Python is a general-purpose, high-level programming language that is used for developing desktop GUI applications, websites and web applications. It is object-oriented, meaning it is based around objects rather than functions. Python was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python supports an Object-Oriented programming approach to develop applications. It is also interpreted, meaning it has to be run in the given system using another program instead of its local processor.

Q2. Why is Python called a dynamically typed language?

Answer:

Python is called a dynamically typed language because it doesn't know about the type of the variable until the code is run. So declaration is of no use. What it does is, It stores that value at some memory location and then binds that variable name to that memory container

Q3. List some pros and cons of Python programming language?

Answer:

Python is a popular programming language that has many advantages and disadvantages. Here are some of the pros and cons of Python programming language:

Pros:

Beginner-friendly

Large Community

Flexible and Extensible

Extensive Libraries

Cons:

Issues with design

Slower than compiled languages

Security

Work Environment

Python has a simple, concise, and straightforward syntax. A Python program looks a lot like plain English and is highly readable.

Q4. In what all domains can we use Python?

Answer:

Python is a versatile language that can be used in many domains. Here are some of the domains where Python is used:

Web development: Python has powerful back-end frameworks like Django and light-weight micro frameworks like Flask, making it easy to make small, easily deployable web apps.

Networking and Internet Development: Python provides two levels of access to network services.

Machine Learning: Python has a clear dominance in machine learning.

Desktop Application Development: Python is extensively used in developing cross-platform GUI applications.

Web Application Development: Python is used in web application development.

Web Scraping: Python is used in web scraping.

Natural Language Processing: Python is used in natural language processing.

Q5. What are variable and how can we declare them?

Answer:

Python has no command for declaring a variable. A variable is created when some value is assigned to it. The value assigned to a variable determines the data type of that variable. Thus, declaring a variable in Python is very simple. Just name the variable and assign the required value to it.

Here’s an example:

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

Copy

This will output:

```
5
```

John

Q6. How can we take an input from the user in Python?

Answer:

You can use the input() function to take input from the user in Python. The input() function takes an optional string argument which is shown as a prompt to the user. After taking input, the input() function returns the value entered by the user as a string.

Here's an example:

```
name = input("What is your name? ")
print("Hello, " + name + "!")
```

Copy

This will output:

```
What is your name? John
Hello, John!
```

Q7. What is the default datatype of the value that has been taken as an input using input() function?

Answer:

The default datatype of the value that has been taken as an input using the input() function is string .

Q8. What is type casting?

Answer:

Type casting is the process of converting one data type to another data type. In Python, you can use the following built-in functions to perform type casting:

int() - converts a value to an integer

float() - converts a value to a floating-point number

str() - converts a value to a string

Here’s an example:

```
x = "5"
print(type(x)) # Output: <class 'str'>

y = int(x)
print(type(y)) # Output: <class 'int'>
```

Copy

This will output:

```
<class 'str'>
<class 'int'>
```

Q9. Can we take more than one input from the user using single input() function? If yes, how? If no, why?

Answer:

Yes, we can take more than one input from the user using a single input() function. We can use the split() method to split the input string into multiple values. Here’s an example:

```
x, y = input("Enter two numbers separated by a space: ").split()
print("x =", x)
print("y =", y)
```

Copy

This will output:

```
Enter two numbers separated by a space: 5 10
x = 5
```

y = 10

Q10. What are keywords?

Answer:

Keywords are reserved words in Python that have special meanings and cannot be used as variable names. Examples of keywords in Python include if, else, while, for, def, class, etc.

Q11. Can we use keywords as a variable? Support your answer with reason.

Answer:

No, we cannot use keywords as a variable because they are reserved words in Python that have special meanings and cannot be used as variable names.

Q12. What is indentation? What's the use of indentaion in Python?

Answer:

Indentation is the process of adding whitespace (spaces or tabs) at the beginning of a line to indicate a block of code. In Python, indentation is used to group statements into blocks of code. The use of indentation in Python is important because it determines the scope of variables and functions.

Q13. How can we throw some output in Python?

Answer:

We can use the print() function to throw some output in Python. The print() function takes one or more arguments and prints them to the console. Here’s an example:

print("Hello, World!")

Copy

This will output:

Hello, World!

Q14. What are operators in Python?

Answer:

Operators in Python are symbols that perform operations on operands (variables or values). Examples of operators in Python include arithmetic operators (+, -, *, /, %, //, **), comparison operators (<, >, <=, >=, ==, !=), logical operators (and, or, not), bitwise operators (&, |, ^, ~, <<, >>), etc.

Q15. What is difference between / and // operators?

Answer:

The / operator performs floating-point division, while the // operator performs integer division (floor division). For example:

```
x = 5 / 2
print(x) # Output: 2.5

y = 5 // 2
print(y) # Output: 2
```

Q16. Write a code that gives following as an output.

```
iNeuroniNeuroniNeuroniNeuron
```

Answer:

```
s = "iNeuron"
```

```
print(s * 4)
```

Copy

This will output:

```
iNeuroniNeuroniNeuroniNeuron
```

Q17. Write a code to take a number as an input from the user and check if the number is odd or even.

Answer:

Here's a code that takes a number as an input from the user and checks if the number is odd or even:

```
num = int(input("Enter a number: "))
if num % 2 == 0:
    print(num, "is even")
else:
    print(num, "is odd")
```

Copy

Here's how it works:

The input() function is used to take a number as input from the user.

The int() function is used to convert the input string to an integer.

The % operator is used to check if the number is divisible by 2 (even) or not (odd).

If the number is even, the code prints a message saying that the number is even. Otherwise, it prints a message saying that the number is odd.

Q18. What are boolean operator?

Boolean operators are operators that operate on boolean values (True or False). Examples of boolean operators in Python include and, or, and not.

Q19. What will the output of the following?

```

1 or 0


0 and 0


True and False and True


1 or 0 or 0

```

Answer:

The output of the code will be:

1

0

False

1

Q20. What are conditional statements in Python?

Conditional statements in Python are used to execute different code blocks based on different conditions. The two main types of conditional statements in Python are if statements and if-else statements.

Q21. What is use of 'if', 'elif' and 'else' keywords?

The if, elif, and else keywords are used in conditional statements in Python to execute different code blocks based on different conditions. The if keyword is used to check a condition, and if the condition is true, the code block following the if statement is executed. The elif keyword is used to check an

additional condition if the previous condition(s) were false. The else keyword is used to execute a code block if none of the previous conditions were true.

Q22. Write a code to take the age of person as an input and if age >= 18 display "I can vote". If age is < 18 display "I can't vote".

Here's a code that takes the age of a person as input and displays "I can vote" if age >= 18, and "I can't vote" otherwise:

```
age = int(input("Enter your age: "))
if age >= 18:
    print("I can vote")
else:
    print("I can't vote")
```

Q23. Write a code that displays the sum of all the even numbers from the given list.

```
numbers = [12, 75, 150, 180, 145, 525, 50]
```

Here's a code that displays the sum of all the even numbers from the given list:

```
numbers = [12, 75, 150, 180, 145, 525, 50]
sum = 0
for num in numbers:
    if num % 2 == 0:
        sum += num
print("Sum of even numbers:", sum)
```

Copy

This will output:

Sum of even numbers: 392

Q24. Write a code to take 3 numbers as an input from the user and display the greatest no as output.

Here's a code that takes 3 numbers as input from the user and displays the greatest number as output:

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
c = int(input("Enter third number: "))
if a > b and a > c:
    print(a, "is the greatest number")
elif b > a and b > c:
    print(b, "is the greatest number")
else:
    print(c, "is the greatest number")
```

Q25. Write a program to display only those numbers from a list that satisfy the following conditions

- The number must be divisible by five

- If the number is greater than 150, then skip it and move to the next number

- If the number is greater than 500, then stop the loop

```
numbers = [12, 75, 150, 180, 145, 525, 50]
```

```

Here’s a code that displays only those numbers from a list that satisfy the following conditions:

numbers = [12, 75, 150, 180, 145, 525, 50]

for num in numbers:

  if num % 5 == 0 and num <= 150:

    print(num)

  elif num > 500:

    break

Copy

This will output:

75

150

50

Q26. What is a string? How can we declare string in Python?

A string is a sequence of characters. In Python, you can declare a string by enclosing a sequence of characters in single quotes (') or double quotes (").

Here’s an example:

s1 = 'Hello'

s2 = "World"

Q27. How can we access the string using its index?

You can access the characters in a string using their index. In Python, string indices start at 0. Here’s an example:
```

```
s = "Hello"
print(s[0]) # Output: H
print(s[1]) # Output: e
print(s[2]) # Output: l
print(s[3]) # Output: l
print(s[4]) # Output: o
```

Q28. Write a code to get the desired output of the following

```
string = "Big Data iNeuron"
desired_output = "iNeuron"
```

Here’s a code that gets the desired output of the following:

```
string = "Big Data iNeuron"
desired_output = string[-7:]
print(desired_output)
```

Copy

This will output:

```
iNeuron
```

Q29. Write a code to get the desired output of the following

```
string = "Big Data iNeuron"
desired_output = "norueNi"
```

```

Here’s a code that gets the desired output of the following:

string = "Big Data iNeuron"

desired_output = string[-2:-8:-1]

print(desired_output)

Copy

This will output:

norueN

Q30. Resverse the string given in the above question.

Here’s a code that reverses the string given in the above question:

string = "Big Data iNeuron"

reversed_string = string[::-1]

print(reversed_string)

Copy

This will output:

norueNi ataD giB

Q31. How can you delete entire string at once?

there are different ways to delete a string in Python. One way is to assign an empty string to the variable that holds the string. For example:

s = "Hello World"
```

s = "" # delete the string

print(s) # prints nothing

Copy

Another way is to use the del keyword to delete the variable that holds the string. For example:

s = "Hello World"

del s # delete the variable

print(s) # raises NameError

Copy

A third way is to use the replace() or translate() methods to remove all the characters from the string. For example:

s = "Hello World"

s = s.replace("","") # replace every character with nothing

print(s) # prints nothing

Copy

s = "Hello World"

s = s.translate({ord(c): None for c in s}) # translate every character to None

print(s) # prints nothing

Q32. What is escape sequence?

an escape sequence is a special character or combination of characters that has a different meaning than the literal characters. In Python, escape sequences are prefixed with a backslash (\) and are used to represent certain special characters within string literals. For example, you can use escape sequences to include characters that can't be directly typed in a string, such as a newline (\n), a tab (\t), or a backslash (\\).

Some common escape sequences in Python are:

\n – represents a new line

\t – represents a tab

\\ – represents a backslash

\’ – represents a single quote

\" – represents a double quote

\a – represents an ASCII bell

\b – represents an ASCII backspace

\f – represents an ASCII form feed

\r – represents an ASCII carriage return

\v – represents an ASCII vertical tab

Q33. How can you print the below string?

```
'iNeuron's Big Data Course'
```

you can print the below string by using either double quotes or an escape sequence. For example:

```
# using double quotes
print("iNeuron's Big Data Course")

# using escape sequence
print('iNeuron\'s Big Data Course')
```

Copy

Both of these will print:

```
iNeuron's Big Data Course
```

Q34. What is a list in Python?

a list in Python is a collection of items that can be of different types, such as integers, strings, or objects. A list is created by putting comma-separated values between square brackets [ ]. A list is

ordered, meaning that the items have a fixed position or index, starting from 0. A list is also mutable, meaning that the items can be changed, added, or removed after the list is created.

For example, you can create a list of fruits like this:

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

Copy

You can access the items in the list by using the index inside square brackets [ ]. For example, you can print the first item like this:

```
print(fruits[0]) # prints apple
```

Copy

You can modify the items in the list by assigning a new value to the index. For example, you can change the second item like this:

```
fruits[1] = "orange" # changes banana to orange
```

Copy

You can add new items to the list by using methods such as append() or insert(). For example, you can add a new item at the end of the list like this:

```
fruits.append("mango") # adds mango to the end of the list
```

Copy

You can remove items from the list by using methods such as remove() or pop(). For example, you can remove the last item from the list like this:

```
fruits.pop() # removes and returns mango from the list
```

Q35. How can you create a list in Python?

a list in Python is a collection of items that can be of different types, such as integers, strings, or objects. A list is created by putting comma-separated values between square brackets [ ]. A list is ordered, meaning that the items have a fixed position or index, starting from 0. A list is also mutable, meaning that the items can be changed, added, or removed after the list is created.

For example, you can create a list of fruits like this:

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

Copy

You can access the items in the list by using the index inside square brackets [ ]. For example, you can print the first item like this:

```
print(fruits[0]) # prints apple
```

Copy

You can modify the items in the list by assigning a new value to the index. For example, you can change the second item like this:

```
fruits[1] = "orange" # changes banana to orange
```

Copy

You can add new items to the list by using methods such as append() or insert(). For example, you can add a new item at the end of the list like this:

```
fruits.append("mango") # adds mango to the end of the list
```

Copy

You can remove items from the list by using methods such as remove() or pop(). For example, you can remove the last item from the list like this:

```
fruits.pop() # removes and returns mango from the list
```

Q36. How can we access the elements in a list?

In Python, you can access elements in a list by their index, which is a number that represents the position of the element in the list. The index of the first element is 0, the second element is 1, and so on. You can also use negative indexes to access elements from the end of the list, such as -1 for the last element, -2 for the second last element, and so on. You can also access a slice of the list by

specifying a range of indexes, such as [1:4] for the elements from index 1 to 3 (excluding 4). Here are some examples123:

```
my_list = [1, 2, 3, 4, 5]
# Access the first element in the list
print(my_list[0]) # Prints 1
# Access the last element in the list
print(my_list[-1]) # Prints 5
# Access a slice of the list
print(my_list[1:4]) # Prints [2, 3, 4]
```

Q37. Write a code to access the word "iNeuron" from the given list.

```
lst = [1,2,3,"Hi",[45,54, "iNeuron"], "Big Data"]
```

you can write a code to access the word “iNeuron” from the given list by using the index of the sublist that contains it, and then the index of the word within that sublist. For example, you can write:

```
lst = [1,2,3,"Hi",[45,54, "iNeuron"], "Big Data"]
print(lst[4][2]) # prints iNeuron
```

Copy

This will print:

```
iNeuron
```

Q38. Take a list as an input from the user and find the length of the list.

you can take a list as an input from the user and find the length of the list by using the input() function to get the user input as a string, and then the eval() function to convert the string to a list. Then, you can use the len() function to get the length of the list. For example, you can write:

user_input = input("Enter a list: ") # get user input as a string

user_list = eval(user_input) # convert user input to a list

list_length = len(user_list) # get the length of the list

print("The length of the list is:", list_length) # print the length of the list

Copy

This will prompt the user to enter a list, and then print the length of the list. For example, if the user enters:

[1, 2, 3, 4, 5]

Copy

The output will be:

The length of the list is: 5

Q39. Add the word "Big" in the 3rd index of the given list.

```
lst = ["Welcome", "to", "Data", "course"]
```

To add an element to a list at a specific index, you can use the insert() method, which takes two arguments: the index and the element. For example, to add the word “Big” in the 3rd index of the given list, you can write:

lst = ["Welcome", "to", "Data", "course"]

lst.insert(3, "Big")

print(lst) # Prints ["Welcome", "to", "Data", "Big", "course"]

Q40. What is a tuple? How is it different from list?

a tuple in Python is a collection of items that can be of different types, such as integers, strings, or objects. A tuple is created by putting comma-separated values between parentheses ( ). A tuple is ordered, meaning that the items have a fixed position or index, starting from 0. A tuple is also immutable, meaning that the items cannot be changed, added, or removed after the tuple is created.

For example, you can create a tuple of colors like this:

colors = ("red", "green", "blue")

Copy

You can access the items in the tuple by using the index inside parentheses ( ). For example, you can print the second item like this:

print(colors(1)) # prints green

Copy

You cannot modify the items in the tuple by assigning a new value to the index. For example, this will raise an error:

colors(1) = "yellow" # TypeError: 'tuple' object does not support item assignment

Copy

You cannot add new items to the tuple or remove items from the tuple. For example, these will raise errors:

colors.append("purple") # AttributeError: 'tuple' object has no attribute 'append'

colors.remove("red") # AttributeError: 'tuple' object has no attribute 'remove'

Copy

The main difference between a tuple and a list is that a tuple is immutable and a list is mutable. This means that a tuple can be used as a key in a dictionary or as an element of a set, while a list cannot. A tuple also consumes less memory than a list.

Q41. How can you create a tuple in Python?

A tuple is a collection of ordered and immutable elements in Python. You can create a tuple by placing all the elements inside parentheses () and separating them by commas. For example12:

t1 = (1, 2, 3, 4) # A tuple of integers

t2 = ("Make", "Use", "Of") # A tuple of strings

t3 = (1.2, 5.9, 5.4, 9.3) # A tuple of floats

Copy

If you want to create a tuple with only one element, you have to add a comma after the element, otherwise Python will not recognize it as a tuple. For example3:

t4 = ("apple",) # A tuple with one element

Copy

You can also use the tuple() built-in function to create a tuple from an iterable object, such as a list or a string. For example4:

t5 = tuple([1, 2, 3]) # A tuple from a list

t6 = tuple("Hello") # A tuple from a string

Q42. Create a tuple and try to add your name in the tuple. Are you able to do it? Support your answer with reason.

No, I'm not able to add my name in the tuple. This is because tuples are immutable, which means they cannot be changed or modified after they are created. Therefore, we cannot use methods like append() or insert() to add elements to a tuple. Here is an example of what happens if I try to add my name to a tuple:

t = (1, 2, 3) # A tuple of numbers

t.append("Himanshu") # Try to add my name to the tuple

Copy

This will raise a TypeError saying that the tuple object has no attribute append. The same error will occur if I try to use insert() or any other method that tries to change the tuple.

The only way to add an element to a tuple is to create a new tuple that combines the original tuple and the element. For example:

t = (1, 2, 3) # A tuple of numbers

t = t + ("Himanshu",) # Create a new tuple that adds my name

print(t) # Prints (1, 2, 3, "Himanshu")

Copy

However, this is not really adding an element to the existing tuple, but creating a new one with a different memory address.

Q43. Can two tuple be appended. If yes, write a code for it. If not, why?

Yes, two tuples can be appended by using the + operator, which creates a new tuple that contains the elements of both tuples. For example:

t1 = (1, 2, 3) # A tuple of numbers

t2 = ("a", "b", "c") # A tuple of letters

t3 = t1 + t2 # A new tuple that appends t1 and t2

print(t3) # Prints (1, 2, 3, "a", "b", "c")

Copy

However, this is not really appending the tuples in place, but creating a new one with a different memory address. The original tuples are not changed or modified.

Q44. Take a tuple as an input and print the count of elements in it.

To take a tuple as an input, we can use the input() function and the eval() function, which evaluates the input as a Python expression. For example:

t = eval(input("Enter a tuple: ")) # Take a tuple as an input

Copy

To print the count of elements in a tuple, we can use the len() function, which returns the number of items in a sequence. For example:

print(len(t)) # Print the count of elements in the tuple

Copy

Here is the complete code:

t = eval(input("Enter a tuple: ")) # Take a tuple as an input

print(len(t)) # Print the count of elements in the tuple

Copy

Here is an example of running the code:

Enter a tuple: (1, 2, 3, 4, 5)

5

Q45. What are sets in Python?

Sets are a built-in data type in Python that store unordered, mutable and unique items. You can create a set by using curly braces { } or the set() function. For example, s = {1, 2, 3} or s = set([1, 2, 3])

Q46. How can you create a set?

A set is a collection of unordered, unchangeable and unique elements in Python. You can create a set by placing all the elements inside curly braces {} and separating them by commas. For example12:

s1 = {1, 2, 3, 4} # A set of integers

s2 = {"apple", "banana", "cherry"} # A set of strings

s3 = {True, False, None} # A set of booleans

Copy

You can also use the set() built-in function to create a set from an iterable object, such as a list or a string. For example31:

```
s4 = set([1, 2, 3]) # A set from a list

s5 = set("Hello") # A set from a string
```

Copy

However, if you want to create an empty set, you have to use the set() function, because using empty curly braces {} will create an empty dictionary instead. For example1:

```
s6 = set() # An empty set

s7 = {} # An empty dictionary
```

Q47. Create a set and add "iNeuron" in your set.

Here is the code to create a set and add “iNeuron” in it:

```
# Create an empty set

s = set()

# Add "iNeuron" to the set

s.add("iNeuron")

# Print the set

print(s)
```

Copy

The output is:

```
{'iNeuron'}
```

Q48. Try to add multiple values using add() function.

The add() function can only add one element at a time to a set. If you want to add multiple values, you have to use the update() function, which takes an iterable object as a parameter and adds all its elements to the set. For example:

```
s = {1, 2, 3} # A set of numbers
s.add(4) # Add one element using add()
s.update([5, 6, 7]) # Add multiple elements using update()
print(s) # Prints {1, 2, 3, 4, 5, 6, 7}
```

Q49. How is update() different from add()?

The main difference between update() and add() is that update() can add multiple elements to a set, while add() can only add one element. Also, update() can accept iterable sequences as parameters, while add() can only accept immutable values123.

For example, if you have a set s = {1, 2, 3}, you can use s.update([4, 5, 6]) to add three elements at once, but you cannot use s.add([4, 5, 6]) because a list is not immutable. However, you can use s.add(4) to add one element.

Q50. What is clear() in sets?

The clear() method is used to remove all elements from a set, leaving it empty. It does not take any parameters or return any value. It simply modifies the original set. For example123:

```
s = {"apple", "banana", "cherry"} # A set of fruits
s.clear() # Remove all elements from the set
print(s) # Prints set()
```

Copy

The time complexity of the clear() method on a set with n elements is O(n), which means it takes linear time to clear the set4.

Q51. What is frozen set?

A frozen set is a special kind of set that is immutable and unique. It means that you cannot add, remove or change the elements of a frozen set once it is created. You can use a frozen set to perform mathematical set operations such as union, intersection, difference, etc1234.

You can create a frozen set by using the frozenset() function on an iterable object, such as a list, a set, a tuple or a dictionary. For example, fs = frozenset([1, 2, 3]) or fs = frozenset({1: "a", 2: "b", 3: "c"}).

Q52. How is frozen set different from set?

A frozen set is a special kind of set that is immutable, meaning it cannot be changed after it is created. It can be created by using the frozenset() function on an iterable object, such as a list or a set. For example1:

fs1 = frozenset([1, 2, 3]) # A frozen set from a list

fs2 = frozenset({"apple", "banana", "cherry"}) # A frozen set from a set

Copy

A regular set is mutable, meaning it can be modified by adding or removing elements. It can be created by using the set() function or the curly braces {} syntax. For example:

s1 = set([1, 2, 3]) # A set from a list

s2 = {"apple", "banana", "cherry"} # A set from curly braces

Copy

The main difference between a frozen set and a regular set is that a frozen set can be used as a key in a dictionary or an element in another set, because it is hashable and unchangeable. A regular set cannot be used in such cases, because it is unhashable and changeable2345. For example:

d = {fs1: "frozen", s1: "regular"} # This will raise an error because s1 is not hashable

s3 = {fs1, s1} # This will also raise an error because s1 is not hashable

Q53. What is union() in sets? Explain via code.

The union() method in sets is used to combine two or more sets and return a new set that contains all the distinct elements from the original sets123456. You can also use the | operator to perform the union operation.

For example, if you have two sets s1 = {1, 2, 3} and s2 = {3, 4, 5}, you can use s1.union(s2) or s1 | s2 to get a new set {1, 2, 3, 4, 5} that contains all the elements from both sets without any duplicates.

Here is the code to demonstrate how to use union() in sets:

```
# Create two sets
s1 = {1, 2, 3}
s2 = {3, 4, 5}

# Use union() method to combine them
s3 = s1.union(s2)
print(s3)

# Use | operator to combine them
s4 = s1 | s2
print(s4)
```

Copy

The output is:

```
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5}
```

Q54. What is intersection() in sets? Explain via code.

The intersection() method is used to find the common elements between two or more sets. It returns a new set that contains only the elements that are present in all the sets. It does not modify the original sets. For example1234:

```
s1 = {1, 2, 3, 4} # A set of numbers

s2 = {"apple", "banana", "cherry", 4} # A set of fruits and a number

s3 = {4, 5, 6, 7} # Another set of numbers

s4 = s1.intersection(s2, s3) # Find the common element between s1, s2 and s3

print(s4) # Prints {4}
```

Copy

You can also use the & operator to perform the intersection operation. For example:

```
s4 = s1 & s2 & s3 # Equivalent to s1.intersection(s2, s3)

print(s4) # Prints {4}
```

Copy

If you have a list of sets, you can use the * operator to unpack them and pass them as arguments to the intersection() method. For example5:

```
setlist = [s1, s2, s3] # A list of sets

s4 = set.intersection(*setlist) # Unpack the list and find the intersection

print(s4) # Prints {4}
```

Q55. What is dictionary ibn Python?

A dictionary in Python is a data structure that stores key-value pairs. A key is a unique identifier for a value, and a value can be any data type, such as a number, a string, a list, or another dictionary1234.

You can create a dictionary by using curly braces {} and separating the key-value pairs by commas. For example, d = {"name": "Alice", "age": 25, "hobbies": ["reading", "coding", "cooking"]}.

You can access the value of a key by using square brackets [] and the key name. For example, d["name"] will return "Alice".

You can also modify, add or delete the key-value pairs in a dictionary by using assignment statements. For example, d["age"] = 26 will change the value of "age" to 26, d["gender"] = "female" will add a new key-value pair to the dictionary, and del d["hobbies"] will remove the key-value pair from the dictionary.

Q56. How is dictionary different from all other data structures.

A dictionary is a data structure that stores data as key-value pairs, where each key is unique and associated with a value. A dictionary is unordered, meaning the order of the elements is not preserved or guaranteed. A dictionary is mutable, meaning it can be changed after it is created. A dictionary is created by using the curly braces {} syntax or the dict() function. For example1:

d1 = {"name": "Alice", "age": 25} # A dictionary with two key-value pairs

d2 = dict(name="Bob", age=30) # Another way to create a dictionary

Copy

A dictionary is different from other data structures in Python in several ways:

A dictionary is different from a list, which is an ordered collection of elements that can be accessed by index. A list can have duplicate elements and can store any type of data. A list is created by using the square brackets [] syntax or the list() function. For example23:

l1 = [1, 2, 3, 4] # A list of numbers

l2 = list("hello") # Another way to create a list

Copy

A dictionary is different from a tuple, which is an ordered collection of elements that can be accessed by index. A tuple can have duplicate elements and can store any type of data. A tuple is immutable, meaning it cannot be changed after it is created. A tuple is created by using the parentheses () syntax or the tuple() function. For example34:

t1 = (1, 2, 3, 4) # A tuple of numbers

t2 = tuple("hello") # Another way to create a tuple

Copy

A dictionary is different from a set, which is an unordered collection of elements that cannot have duplicates. A set can only store hashable data types, such as numbers, strings, tuples, etc. A set is mutable, meaning it can be changed after it is created. A set is created by using the curly braces {} syntax or the set() function. For example534:

s1 = {1, 2, 3, 4} # A set of numbers

s2 = set("hello") # Another way to create a set

Q57. How can we delare a dictionary in Python?

You can declare a dictionary in Python by using curly braces {} and separating the key-value pairs by commas. For example, d = {"name": "Alice", "age": 25, "hobbies": ["reading", "coding", "cooking"]}.

Here is the code to demonstrate how to declare a dictionary in Python:

Create a dictionary

d = {"name": "Alice", "age": 25, "hobbies": ["reading", "coding", "cooking"]}

Print the dictionary

print(d)

Copy

The output is:

{'name': 'Alice', 'age': 25, 'hobbies': ['reading', 'coding', 'cooking']}

Q58. What will the output of the following?

```

var = {}

print(type(var))

```

The output of the following code will be:

<class 'dict'>

Copy

This is because the curly braces {} syntax creates an empty dictionary, which is a data structure that stores key-value pairs. The type() function returns the type of the object passed as an argument. Therefore, type(var) returns <class 'dict'>, which indicates that var is a dictionary.

Q59. How can we add an element in a dictionary?

You can add an element in a dictionary by using an assignment statement with a new key and a value. For example, d["gender"] = "female" will add a new key-value pair to the dictionary.

Here is the code to demonstrate how to add an element in a dictionary:

```
# Create a dictionary
d = {"name": "Alice", "age": 25, "hobbies": ["reading", "coding", "cooking"]}

# Print the dictionary
print(d)

# Add a new element
d["gender"] = "female"

# Print the updated dictionary
print(d)
```

Copy

The output is:

```
{'name': 'Alice', 'age': 25, 'hobbies': ['reading', 'coding', 'cooking']}
{'name': 'Alice', 'age': 25, 'hobbies': ['reading', 'coding', 'cooking'], 'gender': 'female'}
```

Q60. Create a dictionary and access all the values in that dictionary.

To create a dictionary, you can use the curly braces {} syntax or the dict() function. For example, you can create a dictionary that stores the names and ages of some people:

```
d = {"Alice": 25, "Bob": 30, "Charlie": 35} # Using curly braces
# or
d = dict(Alice=25, Bob=30, Charlie=35) # Using dict() function
```

Copy

To access all the values in a dictionary, you can use the values() method, which returns an iterable object that contains the values of the dictionary. You can loop over this object using a for loop or convert it to a list using the list() function. For example:

```
# Using a for loop
for value in d.values():
    print(value)

# Output:
# 25
# 30
# 35

# Using list() function
values = list(d.values())
print(values)

# Output:
# [25, 30, 35]
```

Q61. Create a nested dictionary and access all the element in the inner dictionary.

You can create a nested dictionary by using another dictionary as a value in a key-value pair. For example, d = {"name": "Alice", "age": 25, "address": {"city": "Bangalore", "state": "Karnataka", "country": "India"}}.

You can access the elements in the inner dictionary by using the key name of the outer dictionary and the key name of the inner dictionary separated by square brackets []. For example, d["address"]["city"] will return "Bangalore".

Here is the code to demonstrate how to create a nested dictionary and access all the elements in the inner dictionary:

```
# Create a nested dictionary
d = {"name": "Alice", "age": 25, "address": {"city": "Bangalore", "state": "Karnataka", "country": "India"}}

# Print the nested dictionary
print(d)

# Access the elements in the inner dictionary
print(d["address"]["city"])
print(d["address"]["state"])
print(d["address"]["country"])
```

Copy

The output is:

```
{'name': 'Alice', 'age': 25, 'address': {'city': 'Bangalore', 'state': 'Karnataka', 'country': 'India'}}
Bangalore
Karnataka
India
```

Q62. What is the use of get() function?

The get() function is a method of the dictionary class that allows you to access the value of a key in a dictionary without raising an error if the key is not found. The syntax of the get() function is:

dict.get(key, default=None)

Copy

The key parameter is the key that you want to look up in the dictionary. The default parameter is an optional value that you can specify to return if the key is not found. By default, it is None, which means that the get() function will return None if the key is not found.

For example, suppose you have a dictionary called d that stores some names and ages:

```
d = {"Alice": 25, "Bob": 30, "Charlie": 35}
```

Copy

You can use the get() function to access the value of a key in the dictionary, such as "Alice":

```
d.get("Alice")
# Output: 25
```

Copy

If you try to access a key that is not in the dictionary, such as "David", the get() function will not raise a KeyError, but will return None:

```
d.get("David")
# Output: None
```

Copy

You can also specify a default value to return if the key is not found, such as "Not found":

```
d.get("David", "Not found")
# Output: Not found
```

Copy

The get() function is useful when you want to avoid errors or handle missing keys gracefully.

Q63. What is the use of items() function?

The items() function in Python dictionary is used to get a view object that displays a list of the dictionary's (key, value) tuple pairs. The view object reflects any changes done to the dictionary.

You can use the items() function to iterate over the key-value pairs of a dictionary, or to convert them into a list or another data structure.

For example, if you have a dictionary d = {"name": "Alice", "age": 25, "gender": "female"}, you can use d.items() to get a view object that looks like this: dict_items([("name", "Alice"), ("age", 25), ("gender", "female")]).

Here is the code to demonstrate how to use the items() function in Python dictionary:

```
# Create a dictionary
d = {"name": "Alice", "age": 25, "gender": "female"}

# Use items() function to get a view object
v = d.items()
print(v)

# Iterate over the key-value pairs
for k, v in d.items():
    print(k, v)

# Convert the view object into a list
l = list(d.items())
print(l)
```

Copy

The output is:

```
dict_items([('name', 'Alice'), ('age', 25), ('gender', 'female')])
name Alice
age 25
```

gender female

[('name', 'Alice'), ('age', 25), ('gender', 'female')]

Q64. What is the use of pop() function?

The pop() function is a method of the dictionary class that allows you to remove and return the value of a key in a dictionary. The syntax of the pop() function is:

dict.pop(key, default)

Copy

The key parameter is the key that you want to remove and return from the dictionary. The default parameter is an optional value that you can specify to return if the key is not found. If you do not specify a default value and the key is not found, the pop() function will raise a KeyError.

For example, suppose you have a dictionary called d that stores some names and ages:

d = {"Alice": 25, "Bob": 30, "Charlie": 35}

Copy

You can use the pop() function to remove and return the value of a key in the dictionary, such as "Alice":

d.pop("Alice")

Output: 25

d is now {"Bob": 30, "Charlie": 35}

Copy

If you try to remove and return a key that is not in the dictionary, such as "David", and you do not specify a default value, the pop() function will raise a KeyError:

d.pop("David")

Output: KeyError: 'David'

Copy

You can also specify a default value to return if the key is not found, such as "Not found":

```
d.pop("David", "Not found")

# Output: Not found

# d is unchanged
```

Copy

The pop() function is useful when you want to delete a key-value pair from a dictionary and use its value. You can learn more about it from these sources123.

Q65. What is the use of popitems() function?

The popitem() function in Python dictionary is used to remove and return the item that was last inserted into the dictionary as a tuple. It follows the Last In First Out (LIFO) order. In versions before 3.7, the popitem() function removes a random item.

You can use the popitem() function to delete items from a dictionary while also getting their key-value pairs.

For example, if you have a dictionary d = {"name": "Alice", "age": 25, "gender": "female"}, you can use d.popitem() to remove and return the last inserted item, which is ("gender", "female").

Here is the code to demonstrate how to use the popitem() function in Python dictionary:

```
# Create a dictionary

d = {"name": "Alice", "age": 25, "gender": "female"}

# Use popitem() function to remove and return the last inserted item

t = d.popitem()

print(t)

# Print the updated dictionary

print(d)
```

Copy

The output is:

('gender', 'female')

{'name': 'Alice', 'age': 25}

Q66. What is the use of keys() function?

The keys() function is a method of the dictionary class that allows you to get a view object that contains the keys of the dictionary. The syntax of the keys() function is:

dict.keys()

Copy

The keys() function does not take any parameters. It returns a view object that is an iterable and dynamic collection of the dictionary's keys. You can loop over this object using a for loop or convert it to a list using the list() function.

For example, suppose you have a dictionary called d that stores some names and ages:

d = {"Alice": 25, "Bob": 30, "Charlie": 35}

Copy

You can use the keys() function to get a view object that contains the keys of the dictionary:

d.keys()

Output: dict_keys(['Alice', 'Bob', 'Charlie'])

Copy

You can iterate over this object using a for loop:

for key in d.keys():

    print(key)

```
# Output:
# Alice
# Bob
# Charlie
```

Copy

You can also convert this object to a list using the list() function:

```
keys = list(d.keys())
print(keys)

# Output:
# ['Alice', 'Bob', 'Charlie']
```

Copy

The keys() function is useful when you want to access or manipulate the keys of a dictionary. You can learn more about it from these sources123.

Q67. What is the use of values() function?

The values() function in Python dictionary is used to get a view object that displays a list of all values in a given dictionary1234. The view object reflects any changes done to the dictionary.

You can use the values() function to iterate over only the values in the dictionary, or to convert them into a list or another data structure.

For example, if you have a dictionary d = {"name": "Alice", "age": 25, "gender": "female"}, you can use d.values() to get a view object that looks like this: dict_values(["Alice", 25, "female"]).

Here is the code to demonstrate how to use the values() function in Python dictionary:

```
# Create a dictionary
d = {"name": "Alice", "age": 25, "gender": "female"}
```

```
# Use values() function to get a view object
v = d.values()
print(v)

# Iterate over the values
for val in d.values():
    print(val)

# Convert the view object into a list
l = list(d.values())
print(l)
```

Copy

The output is:

```
dict_values(['Alice', 25, 'female'])
Alice
25
female
['Alice', 25, 'female']
```

Q68. What are loops in Python?

Loops in Python are a way of repeating a block of code multiple times based on a condition or a sequence. There are two main types of loops in Python: for loops and while loops123.

A for loop is used for iterating over a sequence, such as a list, a tuple, a dictionary, a set, or a string. You can use a for loop to perform an action on each element of the sequence. The syntax of a for loop is:

```
for element in sequence:
```

```
    # do something with element
```

Copy

For example, you can use a for loop to print each character of a string:

```
s = "Hello"
for c in s:
    print(c)

# Output:
# H
# e
# l
# l
# o
```

Copy

A while loop is used for executing a block of code as long as a condition is true. You can use a while loop to repeat an action until the condition becomes false. The syntax of a while loop is:

```
while condition:
    # do something
```

Copy

For example, you can use a while loop to print the numbers from 1 to 10:

```
n = 1
while n <= 10:
    print(n)
    n = n + 1

# Output:
# 1
# 2
```

```
# 3
# 4
# 5
# 6
# 7
# 8
# 9
# 10
```

Copy

You can also use nested loops, which are loops inside other loops. You can use nested loops to create more complex patterns or operations. For example, you can use nested loops to print a multiplication table:

```
for i in range(1, 11):
    for j in range(1, 11):
        print(i * j, end=" ")
    print()

# Output:
# 1 2 3 4 5 6 7 8 9 10
# 2 4 6 8 10 12 14 16 18 20
# 3 6 9 12 15 18 21 24 27 30
# ...
```

Copy

Loops are very useful when you want to automate repetitive tasks or process data in Python. You can learn more about them from these sources45.

Q69. How many type of loop are there in Python?

There are three types of loops in Python123:

while loop: This loop is used to execute a block of statements repeatedly until a given condition is satisfied. The loop stops when the condition becomes False.

for loop: This loop is used to iterate over a sequence (such as a list, tuple, string, or range) or any other iterable object. The loop stops when there are no more items in the sequence.

nested loop: This loop is used to place one loop inside another loop. The inner loop executes for each iteration of the outer loop.

Here is an example of each type of loop in Python:

```
# while loop example
n = 5
while n > 0:
    print(n)
    n = n - 1

# for loop example
fruits = ["apple", "banana", "orange"]
for fruit in fruits:
    print(fruit)

# nested loop example
for i in range(1, 4):
    for j in range(1, 3):
        print(i, j)
```

Copy

The output is:

```
5
4
3
2
1
apple
```

banana

orange

1 1

1 2

2 1

2 2

3 1

3 2

Q70. What is the difference between for and while loops?

The main difference between for and while loops in Python is that for loops are used to iterate over a sequence of elements, while while loops are used to execute a block of code until a condition is false123.

For example, if you want to print the numbers from 1 to 10, you can use a for loop with the range() function:

```
for i in range(1, 11):
    print(i)

# Output:
# 1
# 2
# 3
# ...
```

Copy

The range() function returns a sequence of numbers from the start value to the end value (excluding the end value). The for loop iterates over each number in the sequence and prints it.

However, if you want to print the numbers from 1 to 10 until the user enters “stop”, you can use a while loop with an input function:

```
n = 1
while n <= 10:
   print(n)
   n = n + 1
   if input("Enter 'stop' to end: ") == "stop":
      break

# Output:
# 1
# Enter 'stop' to end:
# 2
# Enter 'stop' to end: stop
```

Copy

The input() function returns a string that the user enters. The while loop checks if the number is less than or equal to 10 and prints it. Then it asks the user to enter “stop” to end the loop. If the user enters “stop”, the break statement exits the loop.

You can learn more about the difference between for and while loops from these sources45.

Q71. What is the use of continue statement?

The continue statement in Python is used to skip the current iteration of the loop and continue with the next iteration12345. The continue statement can be used in both while and for loops.

You can use the continue statement to avoid executing some statements in the loop for certain conditions, or to optimize the performance of the loop.

For example, if you want to print only the odd numbers from 1 to 10, you can use a continue statement to skip the even numbers:

```
# for loop with continue statement
for i in range(1, 11):
    # if i is even, skip the rest of the loop
    if i % 2 == 0:
        continue
    # otherwise, print i
    print(i)
```

Copy

The output is:

```
1
3
5
7
9
```

Q72. What is the use of break statement?

The break statement in Python is used to terminate the loop immediately when it is encountered. It is usually placed inside an if condition to check for some external or internal factor that should stop the loop123.

For example, if you want to search for an element in a list and stop the loop when you find it, you can use a break statement:

```
lst = [1, 2, 3, 4, 5]
target = 3
for i in lst:
    if i == target:
        print("Found", target)
```

```
        break
    else:
        print("Not found", i)


# Output:
# Not found 1
# Not found 2
# Found 3
```

Copy

The break statement exits the for loop when the element is equal to the target. Otherwise, it prints “Not found” and continues the loop.

Q73. What is the use of pass statement?

The pass statement in Python is a null statement that is used as a placeholder for future code1234. The pass statement does nothing when it is executed, but it avoids getting an error when empty code is not allowed.

You can use the pass statement when you have a loop, function, class, or conditional statement that is not implemented yet, or that you want to ignore for some reason.

For example, if you want to define a function that does nothing, you can use a pass statement:

```
# define a function that does nothing
def do_nothing():
    pass

# call the function
do_nothing()
```

Copy

The output is:

(no output)

Q74. What is the use of range() function?

The range() function in Python is used to create a sequence of numbers from a given start value to a given end value (excluding the end value). It can also take a step value to specify the increment or decrement of the sequence123.

For example, if you want to create a sequence of numbers from 0 to 9, you can use range(10):

```
print(list(range(10)))
```

```
# Output:
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Copy

The range() function returns a range object that can be converted to a list using the list() function. By default, the start value is 0 and the step value is 1.

If you want to create a sequence of numbers from 1 to 10 with a step of 2, you can use range(1, 11, 2):

```
print(list(range(1, 11, 2)))
```

```
# Output:
# [1, 3, 5, 7, 9]
```

Copy

The range() function takes three arguments: start, stop and step. The stop value is not included in the sequence.

You can learn more about the range() function from these sources45.

Q75. How can you loop over a dictionary?

You can loop over a dictionary in Python by using a for loop123456. There are different methods to loop over the dictionary depending on what you want to access:

If you want to access only the keys of the dictionary, you can use the dict.keys() method or simply loop over the dictionary itself.

If you want to access only the values of the dictionary, you can use the dict.values() method.

If you want to access both the keys and values of the dictionary, you can use the dict.items() method or unpack the key-value pairs in the loop.

Here is an example of each method:

```
# create a dictionary
my_dict = {"name": "Alice", "age": 25, "city": "New York"}

# loop over the keys
for key in my_dict.keys():
    print(key)

# loop over the values
for value in my_dict.values():
    print(value)

# loop over the items
for item in my_dict.items():
    print(item)

# unpack the key-value pairs
for key, value in my_dict.items():
    print(key, value)
```

Copy

The output is:

name

age

city

Alice

25

New York

('name', 'Alice')

('age', 25)

('city', 'New York')

name Alice

age 25

city New York

Coding problems

Q76. Write a Python program to find the factorial of a given number.

A factorial of a number n is the product of all positive integers from 1 to n. For example, the factorial of 5 is 5 * 4 * 3 * 2 * 1 = 120.

There are different ways to write a Python program to find the factorial of a given number, but one of the simplest ways is to use a for loop with the range() function. Here is an example:

```
# Define a function to find the factorial of a given number
def factorial(n):
    # Initialize the result variable to 1
    result = 1
    # Loop from 1 to n using the range() function
    for i in range(1, n + 1):
```

```
        # Multiply the result by i
        result = result * i
    # Return the result
    return result

# Ask the user to enter a number
n = int(input("Enter a number: "))
# Call the factorial function and print the result
print("The factorial of", n, "is", factorial(n))
```

Copy

This program defines a function called factorial that takes a parameter n and returns the factorial of n. It uses a for loop to iterate from 1 to n and multiply the result by each number. Then it asks the user to enter a number and calls the factorial function with that number. It prints the result using the print() function.

Here is an example output:

```
Enter a number: 5
The factorial of 5 is 120
```

Q77. Write a Python program to calculate the simple interest. Formula to calculate simple interest is SI = (P*R*T)/100

Here is a possible Python program to calculate the simple interest:

```
# define a function to calculate the simple interest
def simple_interest(p, r, t):
    # use the formula SI = (P*R*T)/100
    si = (p * r * t) / 100
    # return the result
    return si
```

```
# ask the user to enter the principal amount, rate of interest and time period
p = float(input("Enter the principal amount: "))
r = float(input("Enter the rate of interest: "))
t = float(input("Enter the time period: "))

# call the function and print the result
result = simple_interest(p, r, t)
print(f"The simple interest is {result}")
```

Q78. Write a Python program to calculate the compound interest. Formula of compound interest is A = P(1+ R/100)^t.

Compound interest is the interest that is added to the principal amount of a loan or deposit, so that the interest also earns interest. The formula of compound interest is A = P(1 + R/100)^t, where A is the final amount, P is the principal amount, R is the annual interest rate, and t is the time in years.

Here is an example program:

```
# Define a function to calculate the compound interest
def compound_interest(P, R, t):
    # Calculate the final amount using the formula
    A = P * (1 + R / 100) ** t
    # Return the final amount
    return A

# Ask the user to enter the principal amount, interest rate and time
P = float(input("Enter the principal amount: "))
R = float(input("Enter the annual interest rate: "))
t = float(input("Enter the time in years: "))
# Call the compound_interest function and print the result
```

```
print("The final amount is", compound_interest(P, R, t))
```

Copy

This program defines a function called compound_interest that takes three parameters: P, R and t. It calculates the final amount using the formula and returns it. Then it asks the user to enter the principal amount, interest rate and time and calls the compound_interest function with those values. It prints the result using the print() function.

Here is an example output:

```
Enter the principal amount: 1000
Enter the annual interest rate: 10
Enter the time in years: 5
The final amount is 1610.5100000000002
```

Q79. Write a Python program to check if a number is prime or not.

Here is a possible Python program to check if a number is prime or not:

```
# define a function to check if a number is prime or not
def is_prime(n):
    # if n is less than 2, it is not prime
    if n < 2:
        return False
    # loop from 2 to the square root of n
    for i in range(2, int(n**0.5) + 1):
        # if n is divisible by i, it is not prime
        if n % i == 0:
            return False
    # otherwise, it is prime
    return True
```

```
# ask the user to enter a number
n = int(input("Enter a number: "))

# call the function and print the result
result = is_prime(n)
if result:
    print(f"{n} is a prime number")
else:
    print(f"{n} is not a prime number")
```

Q80. Write a Python program to check Armstrong Number.

An Armstrong number is a number that is equal to the sum of its digits raised to the power of the number of digits123456. For example, 153 is an Armstrong number because it has 3 digits and 153 = 1^3 + 5^3 + 3^3.

Here is a possible Python program to check if a number is an Armstrong number:

```
# define a function to check if a number is an Armstrong number
def is_armstrong(n):
    # convert the number to a string
    s = str(n)
    # get the number of digits
    d = len(s)
    # initialize the sum
    sum = 0
    # loop over each digit
    for c in s:
        # convert the digit to an integer
        i = int(c)
        # add the power of the digit to the sum
```

```
        sum += i**d
    # compare the sum with the original number
    return sum == n

# ask the user to enter a number
n = int(input("Enter a number: "))

# call the function and print the result
result = is_armstrong(n)
if result:
    print(f"{n} is an Armstrong number")
else:
    print(f"{n} is not an Armstrong number")
```

Q81. Write a Python program to find the n-th Fibonacci Number.

Here is a possible Python program to find the n-th Fibonacci Number:

```
# Define a function to calculate the n-th Fibonacci Number
def fibonacci(n):
  # Check if n is a positive integer
  if n < 0 or not isinstance(n, int):
    return "Invalid input"
  # Base cases: F(0) = 0, F(1) = 1
  if n == 0:
    return 0
  if n == 1:
    return 1
  # Recursive case: F(n) = F(n-1) + F(n-2)
  return fibonacci(n-1) + fibonacci(n-2)
```

```
# Ask the user for the value of n
n = int(input("Enter the value of n: "))

# Print the result
print(f"The {n}-th Fibonacci Number is {fibonacci(n)}")
```

Q82. Write a Python program to interchange the first and last element in a list.

Here is a possible Python program to interchange the first and last element in a list:

```
# define a function to interchange the first and last element in a list
def swap_first_last(lst):
    # if the list is empty or has only one element, return the list as it is
    if len(lst) <= 1:
        return lst
    # otherwise, store the first and last element in temporary variables
    first = lst[0]
    last = lst[-1]
    # swap the first and last element using indexing
    lst[0] = last
    lst[-1] = first
    # return the modified list
    return lst

# create a sample list
my_list = [1, 2, 3, 4, 5]

# call the function and print the result
result = swap_first_last(my_list)
```

```
print(f"The original list is {my_list}")
print(f"The swapped list is {result}")
```

Copy

The output is:

```
The original list is [5, 2, 3, 4, 1]
The swapped list is [5, 2, 3, 4, 1]
```

Q83. Write a Python program to swap two elements in a list.

Here is a possible Python program to swap two elements in a list:

```
# Define a function to swap two elements in a list
def swap(lst, i, j):
  # Check if lst is a list and i and j are valid indices
  if not isinstance(lst, list) or i < 0 or i >= len(lst) or j < 0 or j >= len(lst):
    return "Invalid input"
  # Swap the elements at index i and j using a temporary variable
  temp = lst[i]
  lst[i] = lst[j]
  lst[j] = temp
  # Return the modified list
  return lst

# Ask the user for the list and the indices to swap
lst = eval(input("Enter the list: "))
i = int(input("Enter the first index: "))
j = int(input("Enter the second index: "))

# Print the result
```

```
print(f"The list after swapping {i}-th and {j}-th elements is {swap(lst, i, j)}")
```

Q84. Write a Python program to find N largest element from a list.

There are different ways to find N largest elements from a list in Python12345. One of them is to use the built-in sort() method to sort the list in descending order and then slice the first N elements23. For example:

```
# define a function to find N largest elements from a list
def find_n_largest(lst, n):
    # sort the list in descending order
    lst.sort(reverse=True)
    # slice the first N elements
    return lst[:n]

# create a sample list
my_list = [4, 5, 1, 2, 9]

# call the function and print the result
result = find_n_largest(my_list, 2)
print(f"The original list is {my_list}")
print(f"The 2 largest elements are {result}")
```

Copy

The output is:

```
The original list is [9, 5, 4, 2, 1]
The 2 largest elements are [9, 5]
```

Q85. Write a Python program to find cumulative sum of a list.

Here is a possible Python program to find cumulative sum of a list:

```
# Define a function to find cumulative sum of a list
def cumulative_sum(lst):
  # Check if lst is a list
  if not isinstance(lst, list):
    return "Invalid input"
  # Initialize an empty list to store the cumulative sum
  cum_sum = []
  # Initialize a variable to keep track of the current sum
  curr_sum = 0
  # Loop through the elements of lst
  for x in lst:
    # Add the element to the current sum
    curr_sum += x
    # Append the current sum to the cumulative sum list
    cum_sum.append(curr_sum)
  # Return the cumulative sum list
  return cum_sum

# Ask the user for the list
lst = eval(input("Enter the list: "))

# Print the result
print(f"The cumulative sum of the list is {cumulative_sum(lst)}")
```

Q86. Write a Python program to check if a string is palindrome or not.

Here is a possible Python program to check if a string is palindrome or not:

```
# define a function to check if a string is palindrome or not
def is_palindrome(s):
    # convert the string to lowercase
    s = s.lower()
    # reverse the string using slicing
    r = s[::-1]
    # compare the original and reversed string
    return s == r

# ask the user to enter a string
s = input("Enter a string: ")

# call the function and print the result
result = is_palindrome(s)
if result:
    print(f"{s} is a palindrome")
else:
    print(f"{s} is not a palindrome")
```

Q87. Write a Python program to remove i'th element from a string.

Here is a possible Python program to remove i’th element from a string:

```
# Define a function to remove i'th element from a string
def remove_char(s, i):
  # Check if s is a string and i is a valid index
  if not isinstance(s, str) or i < 0 or i >= len(s):
    return "Invalid input"
  # Return the string without the i'th character using slicing
  return s[:i] + s[i+1:]
```

```
# Ask the user for the string and the index to remove
s = input("Enter the string: ")
i = int(input("Enter the index to remove: "))

# Print the result
print(f"The string after removing {i}-th character is {remove_char(s, i)}")
```

Q88. Write a Python program to check if a substring is present in a given string.

Here is a possible Python program to check if a substring is present in a given string:

```
# define a function to check if a substring is present in a given string
def is_substring(sub, s):
    # use the in operator to check if the substring is in the string
    return sub in s

# ask the user to enter a string and a substring
s = input("Enter a string: ")
sub = input("Enter a substring: ")

# call the function and print the result
result = is_substring(sub, s)
if result:
    print(f"{sub} is present in {s}")
else:
    print(f"{sub} is not present in {s}")
```

Q89. Write a Python program to find words which are greater than given length k.

Here is a possible Python program to find words which are greater than given length k:

```
# Define a function to find words which are greater than given length k
def long_words(s, k):
  # Check if s is a string and k is a positive integer
  if not isinstance(s, str) or k < 0 or not isinstance(k, int):
    return "Invalid input"
  # Split the string into a list of words
  words = s.split()
  # Initialize an empty list to store the long words
  long_words = []
  # Loop through the words
  for word in words:
    # Check if the word length is greater than k
    if len(word) > k:
      # Append the word to the long words list
      long_words.append(word)
  # Return the long words list
  return long_words

# Ask the user for the string and the value of k
s = input("Enter the string: ")
k = int(input("Enter the value of k: "))

# Print the result
print(f"The words which are greater than {k} characters are {long_words(s, k)}")
```

Q90. Write a Python program to extract unquire dictionary values.

There are different ways to extract unique dictionary values in Python12345. One of them is to use the set() function along with a loop to iterate over the values in the dictionary3. This works well if the values are not lists, as sets only contain unique elements. For example:

```
# define a dictionary with some values
my_dict = {"a": 1, "b": 2, "c": 1, "d": 3}

# create an empty set to store the unique values
unique_values = set()

# loop through the values in the dictionary
for value in my_dict.values():
    # add the value to the set
    unique_values.add(value)

# print the unique values
print(f"The unique values are {unique_values}")
```

Copy

The output is:

```
The unique values are {1, 2, 3}
```

Q91. Write a Python program to merge two dictionary.

Here is a possible Python program to merge two dictionary:

```
# Define a function to merge two dictionary
def merge_dict(d1, d2):
  # Check if d1 and d2 are dictionary
  if not isinstance(d1, dict) or not isinstance(d2, dict):
    return "Invalid input"
```

```
 # Create a new dictionary to store the merged result
 merged = {}
 # Loop through the keys and values of d1
 for key, value in d1.items():
   # Add the key and value to the merged dictionary
   merged[key] = value
 # Loop through the keys and values of d2
 for key, value in d2.items():
   # Add the key and value to the merged dictionary, overwriting any existing value
   merged[key] = value
 # Return the merged dictionary
 return merged

# Ask the user for the two dictionary
d1 = eval(input("Enter the first dictionary: "))
d2 = eval(input("Enter the second dictionary: "))

# Print the result
print(f"The merged dictionary is {merge_dict(d1, d2)}")
```

Q92. Write a Python program to convert a list of tuples into dictionary.

```
Input : [('Sachin', 10), ('MSD', 7), ('Kohli', 18), ('Rohit', 45)]
Output : {'Sachin': 10, 'MSD': 7, 'Kohli': 18, 'Rohit': 45}
```

Here is a possible Python program to convert a list of tuples into dictionary:

```
# Define a function to convert a list of tuples into dictionary
def tuple_to_dict(lst):
```

```
 # Check if lst is a list of tuples
 if not isinstance(lst, list) or not all(isinstance(x, tuple) for x in lst):
  return "Invalid input"
 # Create an empty dictionary to store the result
 d = {}
 # Loop through the tuples in the list
 for t in lst:
  # Check if the tuple has exactly two elements
  if len(t) == 2:
   # Assign the first element as the key and the second element as the value in the dictionary
   d[t[0]] = t[1]
  else:
   # Return an error message if the tuple has more or less than two elements
   return "Invalid tuple"
 # Return the dictionary
 return d

# Ask the user for the list of tuples
lst = eval(input("Enter the list of tuples: "))

# Print the result
print(f"The dictionary is {tuple_to_dict(lst)}")
```

Q93. Write a Python program to create a list of tuples from given list having number and its cube in each tuple.

```
Input: list = [9, 5, 6]
Output: [(9, 729), (5, 125), (6, 216)]
```

Here is a possible Python program to create a list of tuples from given list having number and its cube in each tuple:

```
# Define a function to create a list of tuples from given list having number and its cube in each tuple
def cube_tuple(lst):
  # Check if lst is a list of numbers
  if not isinstance(lst, list) or not all(isinstance(x, (int, float)) for x in lst):
    return "Invalid input"
  # Create an empty list to store the result
  result = []
  # Loop through the numbers in the list
  for x in lst:
    # Calculate the cube of the number
    cube = x ** 3
    # Create a tuple of the number and its cube
    t = (x, cube)
    # Append the tuple to the result list
    result.append(t)
  # Return the result list
  return result

# Ask the user for the list of numbers
lst = eval(input("Enter the list of numbers: "))

# Print the result
print(f"The list of tuples is {cube_tuple(lst)}")
```

Q94. Write a Python program to get all combinations of 2 tuples.

```
```

Input : test_tuple1 = (7, 2), test_tuple2 = (7, 8)

Output : [(7, 7), (7, 8), (2, 7), (2, 8), (7, 7), (7, 2), (8, 7), (8, 2)]

```

Here is a possible Python program to get all combinations of 2 tuples:

```
# define two tuples
test_tuple1 = (7, 2)
test_tuple2 = (7, 8)

# create an empty list to store the combinations
combinations = []

# loop through the first tuple
for x in test_tuple1:
    # loop through the second tuple
    for y in test_tuple2:
        # append the pair (x, y) to the list
        combinations.append((x, y))

# loop through the second tuple
for x in test_tuple2:
    # loop through the first tuple
    for y in test_tuple1:
        # append the pair (x, y) to the list
        combinations.append((x, y))

# print the combinations
print(f"The combinations are {combinations}")
```

Copy

The output is:
```

The combinations are [(7, 7), (7, 8), (2, 7), (2, 8), (7, 7), (7, 2), (8, 7), (8, 2)]

Q95. Write a Python program to sort a list of tuples by second item.

```
Input : [('for', 24), ('Geeks', 8), ('Geeks', 30)]
Output : [('Geeks', 8), ('for', 24), ('Geeks', 30)]
```

Here is a possible Python program to sort a list of tuples by second item:

```
# Define a function to sort a list of tuples by second item
def sort_by_second(lst):
  # Check if lst is a list of tuples
  if not isinstance(lst, list) or not all(isinstance(x, tuple) for x in lst):
    return "Invalid input"
  # Use the sorted function with a key function that returns the second item of each tuple
  result = sorted(lst, key=lambda x: x[1])
  # Return the sorted list
  return result

# Ask the user for the list of tuples
lst = eval(input("Enter the list of tuples: "))

# Print the result
print(f"The sorted list of tuples is {sort_by_second(lst)}")
```

Q96. Write a python program to print below pattern.

```
*
```

```
* *
* * *
* * * *
* * * * *
```

Here is a possible python program to print below pattern:

```
# define the number of rows
n = 5

# loop from 1 to n
for i in range(1, n + 1):
    # print i stars followed by a newline
    print("* " * i)
```

Copy

The output is:

```
*
* *
* * *
* * * *
* * * * *
```

Q97. Write a python program to print below pattern.

```
    *
   **
  ***
 ****
```

*****

```

Here is a possible python program to print below pattern:

```
# Define a function to print the pattern
def print_pattern(n):
  # Check if n is a positive integer
  if n < 0 or not isinstance(n, int):
    return "Invalid input"
  # Loop from 1 to n
  for i in range(1, n+1):
    # Print n-i spaces followed by i stars in each line
    print(" " * (n-i) + "*" * i)

# Ask the user for the number of lines
n = int(input("Enter the number of lines: "))

# Print the pattern
print_pattern(n)
```

Q98. Write a python program to print below pattern.

```
    *
   * *
  * * *
 * * * *
* * * * *
```

Here is a possible python program to print below pattern:
```

```
# define the number of rows
n = 5

# loop from 1 to n
for i in range(1, n + 1):
    # print n - i spaces followed by i stars and a newline
    print(" " * (n - i) + "* " * i)
```

Copy

The output is:

```
    *
   * *
  * * *
 * * * *
* * * * *
```

Q99. Write a python program to print below pattern.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Here is a possible python program to print below pattern:

```
# Define a function to print the pattern
def print_pattern(n):
```

```
 # Check if n is a positive integer
 if n < 0 or not isinstance(n, int):
  return "Invalid input"
 # Loop from 1 to n
 for i in range(1, n+1):
  # Initialize an empty string to store the numbers in each line
  line = ""
  # Loop from 1 to i
  for j in range(1, i+1):
   # Append j to the line with a space
   line += str(j) + " "
  # Print the line
  print(line)

# Ask the user for the number of lines
n = int(input("Enter the number of lines: "))

# Print the pattern
print_pattern(n)
```

Q100. Write a python program to print below pattern.

```
A
B B
C C C
D D D D
E E E E E
```

Here is a possible python program to print below pattern:

```
# define the number of rows
n = 5

# define the starting letter
letter = "A"

# loop from 1 to n
for i in range(1, n + 1):
    # print i times the letter followed by a newline
    print((letter + " ") * i)
    # increment the letter by one using chr() and ord() functions
    letter = chr(ord(letter) + 1)
```

Copy

The output is:

```
A
B B
C C C
D D D D
E E E E E
```