**Assignment Part-1**

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

**Answer:** A general-purpose programming language is one that is usable (not just theoretically, but practically) for solving a very wide range of different kinds of problems.

In Python, you can write detailed numerical simulations, web microservices, file format parsers, and so on. So, Python is a general-purpose programming language. You would not want to use it to write a kernel extension, but general-purpose doesn’t mean all possible purposes, just a wide variety of them.

There is an advantage to general-purpose languages: For one thing, it means we do not have to learn a thousand languages, only a handful. For another, by giving us more generic problem-solving capabilities, they can often let us see abstract connections between things that weren’t obvious when we first glanced at the problem, and then use those abstract connections to organize your code.

Also python is considered a high-level programming language because of various features:

* **Clear syntax**
  + Python has a clear and clean syntax which is easily readable. It allows even beginners to work with complex software development projects as the team can coordinate easily on the coding front.
  + The simple coding syntax facilitates test-driven development for all applications of Python.
* **Scalable**
  + Companies love Python for its scalability. Some of the companies implementing the [uses of Python](https://www.techrepublic.com/article/python-5-use-cases-for-programmers/) language include Google, Spotify, Netflix, Instagram, and many more that want scalable applications.
  + It allows handling a massive amount of traffic with ease.
* **Versatile**
  + Unlike most programming languages, the [practical uses for Python](https://www.botreetechnologies.com/blog/python-in-healthcare-application/) are not limited to just web or mobile development.
  + It is a popular choice for building web apps, gaming applications, enterprise-grade apps, e-commerce applications, ML and AI applications, and much more.

If you want to scale your application and expand its customer base, Python programming is an excellent choice for you. It comes with a vast collection of libraries, which allow companies to add a lot of features without reducing the load time.

Q2. Why is Python called a dynamically typed language?

**Answer:** Python is a **dynamically typed** language. 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. And makes the contents of the container accessible through that variable name. So, the data type does not matter. As it will get to know the type of the value at run-time. For e.g:

Example1

# This will store 6 in the memory and binds the

# name x to it. After it runs, type of x will

# be int.

x = 6

print(type(x))

# This will store 'hello' at some location int

# the memory and binds name x to it. After it

# runs type of x will be str.

x = 'hello'

print(type(x))

**Output:**

<class 'int'>

<class 'str'>

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

**Answer:** Python Programming Language: Pros & Cons

**Advantages of using Python:**

* Simple and Easy

Python is simple and easy to learn and code. Its code is also easily understandable. The simplicity of language is also one major reason people are learning Python today. Reading Python is much like reading English language.

* Extensive Libraries

Python has a vast collection of libraries. These libraries contain code for CGI, web-browsers, image manipulation, database, threading, unit-testing and many more. Because of these libraries, it makes easier to code. This also helps to limit the length of code to write in Python. Some popular Python libraries are: TensorFlow, Numpy, Keras, PyTorch, Eli5 etc.

* Versatility

Python is probably the most very versatile programming language today. It offers tools for almost every application. This includes web development, GUI, data mining, machine learning, etc. This versatility is a massive advantage for people trying to learn any language.

* Improved Productivity

As we have already mentioned earlier, Python is simple to learn and code. Because of this simplicity, developers can focus more on solving the actual problem. They don’t need to spend long hours to understand the code and its behaviour.

Also, the availability of extensive libraries means developers need to spend less time to code. This aids to the productivity of developers.

* Portability

In Python, you can write your code once, and it will run smoothly on different platforms with ease. Whereas in programming languages like C/C++, you need to change the code to make them run smoothly in different platforms.

However, you have to be careful not to include any system-dependent features while writing your code.

* Interpreted Language

Python is an interpreted programming language. This means Python directly executes your code line by line. In case of any error, it will stop further execution and show you the error. This makes debugging easier and faster for developers.

Python has varied advantageous features and is a popular language to get started with today. However, this language has still some limitations or shortcomings which you should be aware of, as mentioned below.

* Speed and Performance

Programmers sometimes worry about the speed of Python programs. According to some study, it shows that Python is slower than other languages like Java and C++.

However, there is no need to worry. There are tons of ways to improve the performance of programs written in Python run faster. You can learn further on improving performance [here](https://wiki.python.org/moin/PythonSpeed).

* Mobile Computing

Python is a favorable language in many server and desktop platform, but it still has a long way to go in mobile computing.

Often developers have to write mobile apps in a particular language depending on the mobile platform targeted. For example, you need to write iOS app in either Objective-C or Swift. Likewise, if you need to develop mobile apps for Android you need to be written in Java.

Python developers use frameworks like Kivy to write cross-platform mobile apps.

* Database Access

Python lacks behind when it comes to database interaction. Python’s database access layer is relatively underdeveloped in comparison to popular technologies like JDBC and ODBC.

* Runtime Errors

A run time error is a type of error that is raised when we run the application. Because Python language is dynamically typed , it has many design restrictions that might lead to runtime error.

As we can understand from above, Python offers a lot of advantage for developers today. Hence, it is currently used in top world-class companies like Google, Facebook, and Uber as a part of their applications.

Q4. In what all domains can we use Python?

**Answer**: Python can handle almost all types of requests, which makes it highly useful for all kinds of development activities. From enterprise apps to gaming, the application of Python now ranges to a wide variety of applications.

**Here are the top 10 uses of Python in the real world:**

1. **Web application development**
   * Unarguably, one of the top practical uses for Python is [web application development](https://www.botreetechnologies.com/web-application-development). Python is now easily the go-to programming language for web applications.
   * Web development has several uses of Python in the real world. It provides security, convenience, and scalability to applications.
   * Python has a lot of [web development frameworks](https://www.botreetechnologies.com/blog/top-10-web-development-frameworks/) like Django and Flask, which enable rapid app development. Django’s dynamic development capabilities have made Python a useful tool for web applications. The framework is packed with standard libraries, reducing the development time and providing more time-to-market for the web application.
2. **Data Science**
   * As a highly-demanded skill, Data science is now reaching the top. It is becoming one of the most important areas with applications of Python programming.
   * [Python libraries](https://www.botreetechnologies.com/blog/top-python-libraries/) like Pandas, NumPy, SciPy, and several others help you to work with data and extract valuable information and insights.
   * Data scientists have to know the uses of Python for extracting and processing data. It allows them to visualize the data through graphs. Matplotlib and Seaborn, both are used for data visualization.
   * With increasing popularity, Python is the first thing that data scientists have to learn. It is preliminary to working with research and data-based companies.
3. **Artificial Intelligence**
   * Probably the most interesting practical uses for Python is in Artificial Intelligence and Machine Learning. Python is a stable and secure language that can handle the computations required for developing Machine Learning models.
   * Machine Learning algorithms are one of the important real life uses of Python. Developers can write algorithms easily using the programming language.
   * Python has an extensive collection of libraries for [Machine Learning applications](https://www.botreetechnologies.com/machine-learning-solutions). These include SciPy, Pandas, Keras, TensorFlow, NumPy and many more.
   * The uses of Python language in [AI solutions](https://www.botreetechnologies.com/artificial-intelligence-solutions) include advanced computing, data analytics, image recognition,  text & data processing and much more that businesses can profit from. If you want to learn more about AI and Python, click here.
4. **Game development**
   * Gaming app development is now a prominent industry, and it has many applications of Python programming. There are libraries which are widely used for interactive game development.
   * Some of the real world Python projects in the gaming industry include Battlefield 2, Frets on Fire, World of Tanks, etc. These games use Python libraries like PySoy and PyGame for development.
   * Python allows game developers to build tree-based algorithms which are useful in designing different levels in a game. Games require handling multiple requests at once, and Python is extremely fantastic at that.
   * Python game app development is one of the top 10 uses of Python in the real world. It offers developers the opportunity to install a 3D game engine that helps in building powerful games and interfaces.
5. **Internet of Things**
   * Another one of the real life uses of Python is in the internet of things. Python programming language enables developers to turn any object into an electronic gadget with the help of Raspberry Pi.
   * Python is used to create embedded software, allowing high-performance application of Python on smaller objects which can work with the programming language.
   * With the help of Raspberry Pi, developers can do high-level computations using Python applications. By embedding it, developers can turn normal objects into smart electronics.
   * In large scale industries, IoT is widely used to track inventory, move machines, and track order processing along with the status of shipment.
6. **Web Scraping**
   * Web scraping of massive amounts of data is becoming useful for companies for extraction valuable customer information and making smart decisions.
   * This real life application of Python includes scraping large amounts of websites and webpages to extract data for a particular purpose. It could be job listing, price comparison, detailed information and much more.
   * Selenium, PythonRequest, MechanicalSoup are some of the tools which are used to build [web scraping](https://www.botreetechnologies.com/blog/web-scraping-using-mechanize-in-ruby-on-rails/) applications of Python programming.
   * Python has simple code, so it doesn’t involve any complexity in writing software that can provide large amounts of data.
7. **Desktop GUI**
   * Python programming language can work with multiple operating systems and has a powerful architecture for building applications.
   * It has rich text processing tools and a clear syntax, allowing developers to code Desktop GUI applications without any hassle.
   * PyQT, Kivy, PyGUI are a few toolkits and frameworks offered to get you started with the practical uses of Python for GUI development.
   * Developers can create highly functional GUIs with Python and reduce the turnaround time for development.
8. **Enterprise applications**
   * Enterprise applications are highly different from regular web applications. They are designed to serve the needs of an organization rather than individual users.
   * The applications of Python programming in building enterprise-grade applications vary from enterprise to enterprise. It is used mostly for scalability, readability, and its powerful functionality.
   * Enterprise applications can be complicated as they require a lot of security and database handling capabilities. Python is a robust language that can handle multiple database requests at once.
   * Odoo and Tryton are some of the enterprise application development tools that enable building apps with Python. Enterprise apps are one of the most significant uses of Python language.
   * Improve your efficiency with an enterprise application built with Python at an affordable price.
9. **Image recognition and text processing**
   * Applications built with Python can also enable companies to identify images from a database of images and also helps in text processing.
   * With its unique image processing and graphic ensign capabilities, Python allows developers to design 2D and 3D images through different tools.
   * Inkscape, GIMP, Paint Shop are a few examples that showcase the real life applications of Python for designing graphics and images.
   * Some of the top 3D animation packages use Python in their programming stack, which includes Blender, Houdini, 3ds Max, Lightwave, and many more.
10. **Education programs**
    * One of the popular Python programming uses is in the development of education programs and online courses. Python is a really beginner-friendly programming language with a simple learning curve and a wide variety of resources.
    * The syntax of Python is similar to English, which makes it the preferred programming language for beginners. Because of this, education program development at the basic and advanced level is done using Python.
    * Professionals all around the world use Python for building education programs and training courses based on levels. That is why it is one of the best [use cases of Python Development](https://www.botreetechnologies.com/python-development).

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

**Answer:** Python Variable is containers which store values. [Python](https://www.geeksforgeeks.org/python-programming-language/) is not “statically typed”. We do not need to declare variables before using them or declare their type. A variable is created the moment we first assign a value to it. A Python variable is a name given to a memory location. It is the basic unit of storage in a program.

**Example of Python Variables**

* Python3

|  |
| --- |
| Var **=** "iNeuron"  print(Var) |

**Output:**

iNeuron

***Notes:***

* *The value stored in a variable can be changed during program execution.*
* *A Python Variables is only a name given to a memory location, all the operations done on the variable effects that memory location.*

**Rules for creating variables in Python**

* A variable name must start with a letter or the underscore character.
* A variable name cannot start with a number.
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ ).
* Variable names are case-sensitive (name, Name and NAME are three different variables).
* The reserved words(keywords) cannot be used naming the variable.

**Let’s see the simple variable creation:**

* Python3

|  |
| --- |
| # An integer assignment  age **=** 45    # A floating point  salary **=** 1456.8    # A string  name **=** "John"    print(age)  print(salary)  print(name) |

**Output:**

45

1456.8

John

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

**Answer:**

The method to take input is different for different datatypes.

## String Input

The input() method is used to take string input from the user.The user can enter numeric value as well but it will be treated as a string. The program can contain any logic or operation to be performed on the string entered by the user ,but in example, we’ll simply print the string which the user enters.

### Example

print("Enter a string")

a=input()

print("The string entered by user is",a)

### Output

Enter a string

TutorialsPoint

The string entered by user is TutorialsPoint

The above example upon execution, prints the message “Enter a string” on the output screen and lets the user enter something. When input() function executes, the program flow will be stopped until the user gives some input. After entering the string, the second print statement executes.

## Integer Input

The integer input can be taken by just type casting the input received into input(). Thus, for taking integer input, we use int(input()) . Only numerical values can be entered by the user, else it throws an error.

### Example

print("Enter a number")

a=int(input())

print("The number entered by user is",a)

### Output

Enter a number

10

The number entered by user is 10

## Float Input

The float input can be taken by type casting input received in input() .We’ll use float(input()) to take float input. The user can enter integer or float values but the value will be treated as float.

### Example

print("Enter a number")

a=float(input())

print("The number entered by user is",a)

### Output

Enter a number

2.5

The number entered by user is 2.5

## Take Input as Array of Integers

We may at times, need to take an array as input from the user. There is no separate syntax for taking array input.

### Example

print("Enter no. of elements")

a=int(input())

print("Enter",a,"integer elements")

array=[]

for i in range(a):

   array.append(int(input()))

print("Array entered by user is",array)

### Output

Enter no. of elements

5

Enter 5 integer elements

1

2

3

4

5

Array entered by user is [1, 2, 3, 4, 5]

In the above example, the size of the array is taken as input from the user. Then the array is declared and using for loop, we take further elements input from the user and append those in the array.

For taking string array input, we can use input() instead of int(input()) inside for loop.

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

**Answer:** By default input() function takes the user’s input in a string. So, to take the input in the form of int you need to use int() along with the input function.

Q8. **What is type casting?**

**Answer:** If we have a specified value or an object and we want to convert it into a string object, we use the str() method in python. Similarly, we use the float() method to convert a specified value or an object into a floating-point number or a floating object, and the [int() method](https://www.scaler.com/topics/int-python/) to convert an object into an integer value or an integer object.

So, to convert one data type into another data type, we use the typecasting in python.

Q10**. What are keywords?**

**Answer:** Python keywords are reserved words. They are used by python interpreters to understand the program. Keywords define the structure of programs. We can’t use keywords to name program entities such as variables, classes, and functions.

Python has a lot of keywords. The number keeps on growing with the new features coming in python. Python 3.10.5 is the current stable version as of writing this tutorial. There are 35 keywords in Python 3.10.5 release

|  |
| --- |
| Here **is** a list of the Python keywords.    False               **class**               **from**                **or**  None                **continue**            **global**              **pass**  True                **def**                 **if**                  **raise**  **and**                 **del**                 **import**              **return**  as                  **elif**                **in**                  **try**  **assert**              **else**                **is**                  **while**  **async**               **except**              **lambda**              with  **await**               **finally**             nonlocal            **yield**  **break**               **for**                 **not** |

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

**Answer**: We cannot use a keyword as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language. In Python, keywords are case sensitive.

## There is difference between keyword and variable?

KEYWORD: The reserved words of python which have a special fixed meaning for the interpreter are called keywords. No keywords can be used as an identifier. VARIABLE: It is like a container that stores values that can be access or change.

All variable names must begin with a letter of the alphabet or an underscore(\_). After the first initial letter, variable names can also contain letters and numbers. Variable names are case sensitive. No spaces or special characters are allowed.

For e.g: Keywords reports that the most “key” words are squatter, police, breakage, council, sued, Timson, resisted, community. These “key” words are not the most frequent words (which are those like the) but the words which are most unusually frequent in the 1,000-word article.

Keywords in a programming language are the vocabulary of the language. They have very specific meanings. Identifiers (including variable names) are proper names of things.

Words similar to a keyword cannot be used as a variable name. Ex: – The data types like int, char, float etc. These are actually keywords. So you can’t declare a variable with names int, char or float.

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

**Answer:** Python Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Example

if 5 > 2:  
  print("Five is greater than two!")

Python will give you an error if you skip the indentation:

Example

Syntax Error:

if 5 > 2:   
print("Five is greater than two!")

The number of spaces is up to you as a programmer, but it has to be at least one.

Q13. How can we throw some output in Python?

**Answer: Python print() function**prints the message to the screen or any other standard output device.

**Syntax:**

print(value(s), sep= ' ', end = '\n', file=file, flush=flush)

**Parameters:**

* **value(s):**Any value, and as many as you like. Will be converted to a string before printed
* **sep=’separator’ :**(Optional) Specify how to separate the objects, if there is more than one. Default :’ ‘
* **end=’end’:**(Optional) Specify what to print at the end.Default : ‘\n’
* **file :**(Optional) An object with a write method. Default :sys.stdout
* **flush :**(Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

**Return Type:**It returns output to the screen.

Though it is not necessary to pass arguments in the print() function, it requires an empty parenthesis at the end that tells python to execute the function rather calling it by name.

Q14. What are operators in Python?

**Answer:** Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:

Example

print(10 + 5)

Python divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

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

**Answer** :

In Python 3.0 and above

a) / operator, aka classic division, output will be a float

>>> 5/2

2.5

b) // operator, aka floor division, output will be an integer

>>> 5//2

2

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

iNeuroniNeuroniNeuroniNeuron

**Answer:**

# Multiply a string 4 times

Str\_Academy = "iNeuron"

print (Str\_Academy \* 4)

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

**Answer:**

#Taking user input

num = int(input("Enter a number: "))

if (num % 2) == 0:

print("{0} is Even".format(num))

else:

print("{0} is Odd".format(num))

Q18. What are boolean operator?

**Answer :**

**Boolean operators** are words and symbols, such as AND or NOT, that let you expand or narrow your search parameters when using a database or search engine. When you search using these operators, it is known as a Boolean search.

You can use Boolean operators such as **AND**, **OR**, and **NOT** alongside keywords to create a Boolean string that will refine your search to find the most relevant results and [sources](https://www.scribbr.com/working-with-sources/finding-sources/).

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**:

0

0

False

0

Q20. What are conditional statements in Python?

**Answer**:

## Python Conditional Statements

A conditional statement in python, also called a condition constructs, is a statement that accommodates a condition inside itself. This condition is constructed using the bitwise, boolean, and comparison operators in Python. A conditional statement always generates a boolean output that is either true or false. (Note that true is represented as True in Python and false as False). After generating the output, the statement decides whether to move inside the conditional block or leave it as it is. The code inside the conditional block is executed if and only if the conditional statement returns true. (from expressions inside the construct).

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

**Answer**:

## ****If, Elif and Else Statements****

The if/elif/else structure is a common way to control the flow of a program, allowing us to execute specific blocks of code depending on the value of some data.

### if statement

If the condition following the keyword if evaluates as true, the block of code will execute. Note that parentheses are not used before and after the condition check as in other languages.

if True:

print('If block will execute!')

x = 5

if x > 4:

print("The condition was true!") #this statement executes

### else statement

You can optionally add an else response that will execute if the condition is false:

if not True:

print('If statement will execute!')

else:

print('Else statement will execute!')

Or you can also see this example:

y = 3

if y > 4:

print("I won't print!") #this statement does not execute

else:

print("The condition wasn't true!") #this statement executes

Note that there is no condition following the *else* keyword - it catches all situations where the condition was *false*

### elif statement

Multiple conditions can be checked by including one or more elif checks after your initial if statement. Just keep in mind that only one condition will execute:

z = 7

if z > 8:

print("I won't print!") #this statement does not execute

elif z > 5:

print("I will!") #this statement will execute

elif z > 6:

print("I also won't print!") #this statement does not execute

else:

print("Neither will I!") #this statement does not execute

Note: only the first condition that evaluates as *true* will execute. Even though z > 6 is true, the if/elif/else block terminates after the first true condition. This means that an else will only execute if none of the conditions were true.

### Nested if statements

We can also create nested if’s for decision making.

Let’s take an example of finding a number which is even and also greater than 10

python

x = 34

if x % 2 == 0: # this is how you create a comment and now, checking for even.

if x > 10:

print("This number is even and is greater than 10")

else:

print("This number is even, but not greater 10")

else:

print ("The number is not even. So point checking further.")

This was just a simple example for nested if’s. Please feel free to explore more online.

While the examples above are simple, you can create complex conditions using [boolean comparisons](https://guide.freecodecamp.org/python/comparisons) and [boolean operators](https://guide.freecodecamp.org/python/boolean-operations).

### Inline python if-else statement

We can also use if-else statements inline python functions. The following example should check if the number is greater or equal than 50, if yes return True:

python

x = 89

is\_greater = True if x >= 50 else False

print(is\_greater)

Output

>

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".

**Answer:**

# input age

age = int(input("Enter Age : "))

# condition to check voting eligibility

**if** age>=18:

status="Eligible"

**else**:

status="Not Eligible"

**print**("You are ",status," for 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]

**Answer:**

# initializing list

test\_list = [12, 75, 150, 180, 145, 525, 50]

 # printing original list

print("The original list is : " + str(test\_list))

even\_sum = 0

for num in test\_list:

         # adding in particular summation according to value

        if int(num) % 2 == 0:

            even\_sum += int(num)

# printing result

print("Even digit sum : " + str(even\_sum))

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

**Answer:**

#input first,Second and third number

num1=int(input("Enter First Number "))

num2=int(input("Enter Second Number "))

num3=int(input("Enter Third Number "))

#Check if first number is greater than rest of the two numbers.

if (num1> num2 and num1> num3):

    print("The Largest number is", num1)

#Check if Second number is greater than rest of the two numbers.

elif (num2 > num1 and num2> num3):

    print ("The Largest number is", num2)

else:

    print ("The Largest number is", num3)

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]

**Answer:**

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

b = []

for i in a:

    if i > 150:

        if i > 500:

            break

        continue

    if i % 5 == 0:

        b.append(i)

print(b)

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

**Answer:** A string (str) in Python is a single character or a collection of characters. When we create a string, this internally converts into a combination of 1s and 0s for the computer to handle. This conversion of string to binary numbers is called encoding and is done using Unicode in Python.

In Python, a string can be created by enclosing it with either single or double quotes (‘’ or “”). For example,

**Examples of creating a String in Python:**

name1='Python'

name2="PythonGeeks"

You cannot use a single quote at one end and a double quote at the other end. This will result in an error.

**Example of getting an error on using single and a double quote:**

name3='abcd"

**Output:**

SyntaxError: EOL while scanning string literal

And the type of these variables can be checked by using the built-in function type() as shown below.

**Example of checking the data type of the variable holding text:**

print("The type of ",name1,"is:",type(name1))

print("The type of ",name2,"is:",type(name2))

**Output**

The type of Python is: <class ‘str’>

The type of PythonGeeks is: <class ‘str’>

Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

Example

print("Hello")  
print('Hello')

Output:

Hello  
Hello

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

**Answer:** Accessing Strings

You can access the characters in a string by referring to its index number inside square brackets [].

This example prints the **first character** in **myString**:

Example

string\_myString = "Hello";

print( string\_myString[0]);

Outputs:

H

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

string = "Big Data iNeuron"

desired\_output = "iNeuron"

**Answer:**

string = "Big Data iNeuron"

print(string[9:])

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

string = "Big Data iNeuron"

desired\_output = "norueNi"

**Answer:**

string = "Big Data iNeuron"

desired\_output = string.split(" ")[2]

print(desired\_output[-1: :-1])

Q30. Reverse the string given in the above question.

**Answer:**

string = "Big Data iNeuron"

print(string[-1: :-1])

Q31. How can you delete entire string at once?

**Answer:** Given string **str**. Whole string can be deleted using del statement.

String = "123"

del String

print(String)

## Q32. What is escape sequence?

## Answer:

* An escape sequence is a special character used in the form of**backslash(\)** followed by a **character** that is required.
* These characters are used to represent whitespace.
* Whitespace gives characters like space, tab, formfeed, vertical tab.

|  |  |
| --- | --- |
| **Escape sequence** | **Meaning** |
| \’ | This represents a single quote |
| \n | This represents a newline |
| \r | This represents a carriage return |
| \t | This represents a tab |
| \b | This represents a backspace |
| \f | This represents a formfeed |
| \ooo | This represents an octal value |
| \xhh | This represents a hex value |
| \\ | This represents a backslash |
| \uxxxx | This represents 16 bits hex value |
| \uxxxxxxxx | This represents 32 bits hex value |

Q33. How can you print the below string?

'iNeuron's Big Data Course'

**Answer**:

String = "'iNeuron's Big Data Course'"

print(String)

Q34. What is a list in Python?

**Answer:**

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are [Tuple](https://www.w3schools.com/python/python_tuples.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

Lists are created using square brackets:

Example

Create a List:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

Output:

['apple', 'banana', 'cherry']

List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

**Note:** There are some [list methods](https://www.w3schools.com/python/python_lists_methods.asp) that will change the order, but in general: the order of the items will not change.

Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

Example

Lists allow duplicate values:

thislist = ["apple", "banana", "cherry", "apple", "cherry"]  
print(thislist)

After Run:

['apple', 'banana', 'cherry', 'apple', 'cherry']

Q35. How can you create a list in Python?

**Answer:**

The Python lists are widely used in python. Lists are one of the most used data structures in Python. It's an unordered data store.

The lists are a sequential data store. Item saved in the list by its index. The list index starts with 0. This mean that a simple list  x = [1, 2, 3]. To get the 1st item you will need the item by index.

## ****Create a List in Python****

To define lists in Python there are two ways. The first is to add your items between two square brackets.

**Example:**

items = [1, 2, 3, 4]

The 2nd method is to call the Python list built-in function by passing the items to it.

**Example:**

Items  = list(1, 2,3,4)

In both cases, the output will be

[1, 2, 3, 4]

The list can accept any data type. You can have a list of integers and strings. List in python doesn't enforce to have a single item type in it. You can have a list of different items.

[1, 'name', {"key" : "value"}, list(1, 2, 3)]

This gives you the flexibility to add multiple data types in the list. You can add a list inside this list. This is called a nested list.

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

**Answer**:

### Accessing a list

To access a list – we can simply print the list object and the complete list prints as an output.

print (list\_object)

**Example:**

In this example, we will declare and assign the list, will print their types, and will print the list. To find the type of an object – we use [type() method](https://www.includehelp.com/python/determine-the-type-of-an-object.aspx).

# declaring lists

list1 = [10, 20, 30, 40, 50, 10, 60, 10]

list2 = ["Hello", "IncludeHelp"]

list3 = ["Hello", 10, 20, "IncludeHelp"]

# printing the list and its elements

**print**("list1: ", list1)

**print**("list2: ", list2)

**print**("list3: ", list3)

# printing the types

**print**("Type of list1 object: ", type(list1))

**print**("Type of list2 object: ", type(list2))

**print**("Type of list3 object: ", type(list3))

**Output**

list1: [10, 20, 30, 40, 50, 10, 60, 10]

list2: ['Hello', 'IncludeHelp']

list3: ['Hello', 10, 20, 'IncludeHelp']

Type of list1 object: <class 'list'>

Type of list2 object: <class 'list'>

Type of list3 object: <class 'list'>

### Accessing list elements based on the index

**To access list elements based on the given index** – we simply pass the index starting from **0 to length-1** to access the particular element and we can also pass the negative index to access the list elements in the reverse order (**-1** to access the last element, **-2** to access the second last element, and so on...)

**Syntax:**

list\_object[index]

**Example:**

# declaring lists

list1 = [10, 20, 30, 40, 50]

# Accessing the elements of a list by its index

**print**("list1[0]: ", list1[0])

**print**("list1[1]: ", list1[1])

**print**("list1[2]: ", list1[2])

**print**("list1[3]: ", list1[3])

**print**("list1[4]: ", list1[4])

**print**() # prints a new line

# Accessing the elements of a list by its index

# in reverse order

**print**("list1[-1]: ", list1[-1])

**print**("list1[-2]: ", list1[-2])

**print**("list1[-3]: ", list1[-3])

**print**("list1[-4]: ", list1[-4])

**print**("list1[-5]: ", list1[-5])

**Output**

list1[0]: 10

list1[1]: 20

list1[2]: 30

list1[3]: 40

list1[4]: 50

list1[-1]: 50

list1[-2]: 40

list1[-3]: 30

list1[-4]: 20

list1[-5]: 10

### Access elements using List slicing

We can also access a set of elements by using list slicing by defining the start\_index and end\_index.

**Syntax:**

list\_object[[start]:[end])

**Note:** One of the values start or end may be optional – consider the below example.

**Example:**

# declaring lists

list1 = [10, 20, 30, 40, 50]

# printing list

**print**("list1: ", list1)

# printing elements using list slicing

# prints 5 elements from starting

**print**("list1[:5]: ", list1[:5])

# prints 3 elements from starting

**print**("list1[:3]: ", list1[:3])

# prints all elements from the index 0

**print**("list1[0:]: ", list1[0:])

# prints all elements from the index 3

**print**("list1[3:]: ", list1[3:])

# prints the elements between index 2 to 3

**print**("list1[2:3]: ", list1[2:3])

# prints the elements between index 0 to 4

**print**("list1[0:4]: ", list1[0:4])

# prints the elements between index 1 to 4

**print**("list1[1:4]: ", list1[1:4])

# prints elements in the reverse order

**print**("list1[ : : -1]: ", list1[ : : -1])

**Output**

list1: [10, 20, 30, 40, 50]

list1[:5]: [10, 20, 30, 40, 50]

list1[:3]: [10, 20, 30]

list1[0:]: [10, 20, 30, 40, 50]

list1[3:]: [40, 50]

list1[2:3]: [30]

list1[0:4]: [10, 20, 30, 40]

list1[1:4]: [20, 30, 40]

list1[ : : -1]: [50, 40, 30, 20, 10]

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

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

**Answer**:

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

print(lst[4][2])

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

**Answer:**

input\_string = input("Enter all family members name separated by space  ")

# Split string into words

family\_list = input\_string.split(" ")

print("\n")

# Iterate a list

print("Printing all family member names")

for name in family\_list:

    print(name)

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

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

**Answer**:

My\_list = ["Welcome", "to", "Data", "course"]

My\_list[2] = "Big " + My\_list[2]

print(My\_list)

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

**Answer** :

List and Tuple in Python are the classes of Python Data Structures. The list is dynamic, whereas the tuple has static characteristics. This means that lists can be modified whereas tuples cannot be modified, the tuple is faster than the list because of static in nature. Lists are denoted by the square brackets but tuples are denoted by parenthesis.

**Important differences between List and Tuple in Python**

| SR.NO. | LIST | TUPLE |
| --- | --- | --- |
| 1 | Lists are mutable | Tuples are immutable |
| 2 | The implication of iterations is Time-consuming | The implication of iterations is comparatively Faster |
| 3 | The list is better for performing operations, such as insertion and deletion. | Tuple data type is appropriate for accessing the elements |
| 4 | Lists consume more memory | Tuple consumes less memory as compared to the list |
| 5 | Lists have several built-in methods | Tuple does not have many built-in methods. |
| 6 | The unexpected changes and errors are more likely to occur | In tuple, it is hard to take place. |

Q41. How can you create a tuple in Python?

**Answer** :

A tuple is a collection of immutable Python objects. It can hold elements of any arbitrary data type (integer, string, float, list, etc.) which makes it a flexible and powerful data structure. It is a part of the Python core language and widely used in Python programs and projects.

**Creating a Tuple**

A tuple in Python can be created by enclosing all the comma-separated elements inside the parenthesis **()**.

Close

t1 = (1, 2, 3, 4)  
t2 = ("Make", "Use", "Of")  
t3 = (1.2, 5.9, 5.4, 9.3)

Elements of the tuple are immutable and ordered. It allows duplicate values and can have any number of elements. You can even create an empty tuple. A tuple's elements can be of any data type (integer, float, strings, tuple, etc.).

**Creating an Empty Tuple**

An empty tuple can be created by using empty opening and closing parentheses.

emptyTuple = ()

**Creating a Tuple With a Single Element**

To create a tuple with only 1 element, you need to add a **comma** after the element to get it recognized as a tuple by Python.

# t1 is a tuple  
t1 = ( 3.14, )  
print( type(t1) )  
# prints  
<**class** '**tuple**'&**gt**;# t2 is not a tuple  
t2 = ( 3.14 )  
print( type(t2) )  
# prints  
<**class** '**float**'&**gt**;

**Note: type()** Function returns the class type of the object passed as a parameter.

By not using a comma after the element results in the class type of t2 as ‘float’, thus it is mandatory to use a comma after the element when creating a tuple with a single value.

**Creating a Tuple With Different Data Types**

Elements of the tuple can be of any data type. This feature makes the tuple versatile.

tup1 = ( 'MUO', **True**, 3.9, 56, [1, 2, 3] )  
**print**( tup1 )  
# prints  
('MUO', **True**, 3.9, 56, [1, 2, 3])

**Creating a Tuple Using tuple() Constructor**

Tuples can also be created using the **tuple()** constructor. Using the tuple() constructor you can convert sequences like list/dictionary into a tuple.

tup1 = tuple( (1, 2, 3) )  
**print**( tup1 )  
# prints  
(1, 2, 3)

**Creating a Nested Tuple**

Tuples can easily be nested inside the other tuples. You can nest the tuple up to any level you want.

tup1 = (1, 2, 3)  
tup2 = ( 'Hello', tup1, 45 )  
**print**( tup2 )  
# prints  
('Hello', (1, 2, 3), 45)

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

**Answer**:

In python, a**tuple is a collection of items** that are ordered and immutable(unchangeable), tuples are written with round brackets **“()”** and it can contain mixed data types. To create a tuple in Python, add items with round brackets “()” like my\_tuple = (“red”, “blue”, “green”) and to create an empty tuple, use empty round brackets ” () ” with no items in it.

**Example:**

my\_tuple = ("red", "blue", "green")

print(my\_tuple)

Now, adding my name in the tuple:

my\_tuple = ("red", "blue", "green")

print(my\_tuple)

my\_tuple.append("Veena")

This operations are not possible as the tuples are immutable.

Error message:

my\_tuple.append("Veena")

AttributeError: 'tuple' object has no attribute 'append'

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

**Answer**: In python, **you cannot add items in the tuple** ones it is created and tuples are unchangeable which means we cannot change.

Here, we cannot add items in tuples and this will raise an error. Therefore, two tuples can not be appended as is as tuples are immutable. But if we convert them to list first and then append then and convert them back to tuple, this can be an alternative way to append tuples.

my\_tuple1 = (34,20,32)

my\_tuple2 = (44,50,62)

my\_tuple3 = tuple(list(my\_tuple1) + list(my\_tuple2))

print(my\_tuple3)

print(type(my\_tuple3))

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

**Answer** :

In python, to get the length of a tuple we will use **len()** method and it will return the number of items in a tuple.

**Example:**

my\_tuple = ("red", "blue", "green")

print(len(my\_tuple))

Output

3

Q45. What are sets in Python?

**Answer** :

Sets in Python are similar to those in math.

A set is an unordered and mutable collection of unique elements. It works based on another data structure called a hash table. Because of this, it is advantageous over the lists while checking for an element in a set.

Q46. How can you create a set?

**Answer:**

#### **Creating a set in Python**

Sets have the elements in the curly brackets with each of them separated by a comma. It can contain values of different data types.

**Example of creating a set:**

set1={1,2,3,4,5} #creating a set with same data type

print(f"The type of {set1} is {type(set1)}")

set2={3.4,'t','PythonGeeks',4} #creating a set with different data type

print(f"The type of {set2} is {type(set2)}")

**Output:**

The type of {1, 2, 3, 4, 5} is <class ‘set’>

The type of {3.4, ‘PythonGeeks’, 4, ‘t’} is <class ‘set’>

##### 1. Does not allow duplicates:

One of the properties of a set is not to allow duplicate values. While creating a set if we give a duplicate value, it will occur only once in the set. For example,

**Example of creating a set with duplicate values:**

set\_1={1,3,'a',5,'a',3,7,3}

print(set\_1)

**Output:**

{1, 3, 7, 5, ‘a’}

##### 2. Unhashable elements:

Though a set is mutable, it cannot have mutable elements like lists, sets, or dictionaries. Therefore, there is no concept of nested sets. So we get an error if we try to create a set with any of these data types as shown below.

**Example of creating a set with mutable elements :**

set1={1,[3,'b',5.0],6.7,'t'}

set2={4.6,{5,6,'t'},7}

set3={3.2,{1:'s',2:'r'},7,'y'}

**Output:**

Traceback (most recent call last):  
File “<pyshell#6>”, line 1, in <module>  
set1={1,[3,’b’,5.0],6.7,’t’}  
TypeError: unhashable type: ‘list’

Traceback (most recent call last):  
File “<pyshell#7>”, line 1, in <module>  
set2={4.6,{5,6,’t’},7}  
TypeError: unhashable type: ‘set’

Traceback (most recent call last):  
File “<pyshell#8>”, line 1, in <module>  
set3={3.2,{1:’s’,2:’r’},7,’y’}  
TypeError: unhashable type: ‘dict’

But we can include a tuple in a set as it is immutable. For example,

**Example of creating a set with a tuple:**

set4={1,3.6,(4,'t',7.8)}

set4

**Output:**

{1, 3.6, (4, ‘t’, 7.8)}

##### 3. Creating an empty set:

Generally, when we try creating an empty list or tuple or a dictionary, we give only the corresponding brackets. But this is not possible in the case of an empty set. Look at the below example to know the reason.

**Example of creating a variable with curly brackets:**

var={}

print(f"The type of {var} is {type(var)}")

**Output:**

The type of {} is <class ‘dict’>

Remember even the dictionaries use curly brackets to enclose their elements. So, when we try to give empty curly brackets, it is considered as an empty dictionary.

Then how do we create an empty set, we create a set using the set() function.

##### 4. Creating set using the set() function:

Another way to create a set is by using the set() function. First let us see the creation of an empty set, which is false in a boolean context.

**Example of creating an empty set using the set() function:**

set1=set()

print(f"The type of {set1} is {type(set1)}")

bool(set1)

**Output:**

The type of set() is <class ‘set’>

False

We can give any iterable as an argument to the set(), to create the corresponding set.

**Example of creating a set using the set() function:**

set1=set("PythonGeeks") #creating a set from a string

print(set1)

set2=set([1,2,3,4,1,2]) #creating a set from a list

set2

set3=set(('a','b',4,5.6))#creating a set from a tuple

set3

**Output:**

{‘P’, ‘h’, ‘n’, ‘o’, ‘y’, ‘e’, ‘k’, ‘t’, ‘G’, ‘s’}

{1, 2, 3, 4}

{4, 5.6, ‘b’, ‘a’}

Observe that the duplicates from the arguments of the set do not occur in the resulting set. Also, see the difference between the creation of a set using curly brackets and using the set() function with a single string.

**Example of creating a set with string:**

set1={'abab'}

set1

set2=set('abab')

set2

**Output:**

{‘abab’}

{‘b’, ‘a’}

Interesting result, isn’t it? This is because when we create using the bracket, the string is considered as a single value. But while using the set() function, it is considering the string as an iterable with a collection of characters.

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

**Answer**:

My\_Set = {"red", "blue", "green"}

print(My\_Set)

My\_Set.add("iNeuron")

print(My\_Set)

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

**Answer:**

We cannot add multiple values using add() function.

For e.g:

My\_Set = {"red", "blue", "green"}

print(My\_Set)

My\_Set.add("iNeuron",1)

print(My\_Set)

Output:

My\_Set.add("iNeuron",1)

TypeError: add() takes exactly one argument (2 given)

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

**Answer:**

**Differences between add() and update()**

1. We use add() function to add a single element. Whereas use update() function to add multiple elements to the set.
2. add() is faster than update().
3. add () accepts immutable parameters only. Whereas accepts iterable sequences.
4. add() accepts a single parameter, whereas update() can accept multiple sequences.

For e.g

We can add multiple values in set using update.

My\_Set = {"red", "blue", "green"}

print(My\_Set)

Multivalue\_Set= ["iNeuron", "Big Data", "Hadoop"]

My\_Set.update(Multivalue\_Set)

print(My\_Set)

Q50. What is clear() in sets?

**Answer:**

Python Set clear() method removes all elements from the set.

**Example 1: Python Set clear() Method Example**

* Python3

|  |
| --- |
| test\_set **=** {1, 2, 3, 4}  test\_set.clear()  print("After clear() on test\_set:", test\_set) |

**Output:**

After clear() on test\_set: set()

**Example 2: Python Set clear() Method on a Set of Strings**

* Python3

|  |
| --- |
| # set of letters  GEEK **=** {"A", "B", "C"}  print('GEEK before clear:', GEEK)    # clearing vowels  GEEK.clear()  print('GEEK after clear:', GEEK) |

**Output:**

GEEK before clear: {'B', 'C', 'A'}

GEEK after clear: set()

Q51. What is frozen set?

**Answer:**

**Python frozenset() Method**creates an immutable Set object from an iterable. It is a built-in Python function. As it is a set object therefore we cannot have duplicate values in the frozenset.

**frozenset() in Python**

***Syntax :****frozenset(iterable\_object\_name)****Parameter :****iterable\_object\_name*

* *This function accepts iterable object as input parameter.*

***Return :****Returns an equivalent frozenset object.*

**Using frozenset() Method on tuple**

If no parameters are passed to frozenset() function, then it returns an empty frozenset type object in [Python](https://www.geeksforgeeks.org/python-programming-language/).

* Python3

|  |
| --- |
| # passing an empty tuple  nu **=** ()    # converting tuple to frozenset  fnum **=** frozenset(nu)    # printing empty frozenset object  print("frozenset Object is : ", fnum) |

**Output:**

frozenset Object is : frozenset()

**Using frozenset() Method on list**

Here as a parameter a [list](https://www.geeksforgeeks.org/python-lists/) is passed and now it’s frozenset object is returned.

* Python3

|  |
| --- |
| l **=** ["Geeks", "for", "Geeks"]    # converting tuple to frozenset  fnum **=** frozenset(l)    # printing empty frozenset object  print("frozenset Object is : ", fnum) |

**Output:**

frozenset Object is : frozenset({'Geeks', 'for'})

**Using frozenset() Method on Dictionary**

Since frozenset objects are immutable, they are mainly used as key in [dictionary](https://www.geeksforgeeks.org/python-dictionary/) or elements of other sets. The below example explains it clearly.

* Python3

|  |
| --- |
| # creating a dictionary  Student **=** {"name": "Ankit", "age": 21, "sex": "Male",             "college": "MNNIT Allahabad", "address": "Allahabad"}    # making keys of dictionary as frozenset  key **=** frozenset(Student)    # printing dict keys as frozenset  print('The frozen set is:', key) |

**Output:**

The frozen set is: frozenset({'address', 'name', 'age', 'sex', 'college'})

**Exceptions while using Python frozenset() method**

If by mistake we want to change the *frozenset*object, then it throws a [TypeError](https://www.geeksforgeeks.org/handling-typeerror-exception-in-python/)

* Python3

|  |
| --- |
| # creating a list  favourite\_subject **=** ["OS", "DBMS", "Algo"]    # creating a frozenset  f\_subject **=** frozenset(favourite\_subject)    # below line will generate error  f\_subject[1] **=** "Networking" |

**Output:**

TypeError Traceback (most recent call last)

Input In [13], in <cell line: 8>()

5 f\_subject = frozenset(favourite\_subject)

7 # below line will generate error

----> 8 f\_subject[1] = "Networking"

TypeError: 'frozenset' object does not support item assignment

Q52. How is frozen set different from set?

**Answer:**

Python provides two built-in functions which are set() and frozenset(). These two functions are used for creating sets but come with a few differences. Let’s see how you can use them.

## Python set()

A set is an unordered and unindexed collection of unique elements. Sets are mutable, you can change the elements using a built-in function like add(), remove(), etc. Since the elements are mutable and not in order, they don’t have hash values. So you can’t access the elements with the help of index numbers.

**Note:** Sets can’t be used as a dictionary key or as elements of another set. They can be used as a dictionary value.

Set is represented by curly braces like this {} or you can use set(). You can’t use only curly braces to create an empty set, this will create a dictionary. You can use set() if you want to create an empty set. Sets can include any immutable data type like string, number, tuple, etc. You can also include mutable data type like list, dictionary, etc.

Let’s go through some examples and see some of the operations you can perform on sets:

fruits = {"Apple", "Banana", "Cherry", "Apple", "Kiwi"}

print('Unique elements:', fruits)

# Add new fruit

fruits.add("Orange")

print('After adding new element:', fruits)

# Size of the set

print('Size of the set:', len(fruits))

# check if the element is present in the set

print('Apple is present in the set:', "Apple" in fruits)

print('Mango is present in the set:', "Mango" in fruits)

# Remove the element from the set

fruits.remove("Mango")

print('After removing element:', fruits)

# Discard the element from the set

fruits.discard("Mango")

print('After discarding element:', fruits)

The output of the above code is as follows:

Unique elements: {'Kiwi', 'Apple', 'Cherry', 'Banana'}

After adding new element: {'Kiwi', 'Orange', 'Banana', 'Apple', 'Cherry'}

Size of the set: 5

Apple is present in the set: True

Mango is present in the set: False

Traceback (most recent call last):

File "c:\Users\hp\Desktop\set() and frozenset().py", line 19, in <module>

Fruits.remove("Mango")

KeyError: 'Mango'

In the above example, some of the built-in functions have been used. There exists two functions remove() and discard() which help to remove the element from the set. They both remove the element from the set if there is an element present in the set but there is a difference between them.  
  
If the element is not in the set which you want to remove then discard() returns none while remove() will raise an error. You can learn more about the operations from their [official documentation](https://docs.python.org/3/library/stdtypes.html#set).

## Python frozenset()

A frozenset is an unordered and unindexed collection of unique elements. It is immutable and it is hashable. It is also called an immutable set. Since the elements are fixed, unlike sets you can't add or remove elements from the set.

Frozensets are hashable, you can use the elements as a dictionary key or as an element from another set. Frozensets are represented by the built-in function which is frozenset(). It returns an empty set if there are no elements present in it. You can use frozenset() if you want to create an empty set.

Let's go through some examples to understand more about frozenset:

fruits = {"Apple", "Banana", "Cherry", "Apple", "Kiwi"}

basket = frozenset(fruits)

print('Unique elements:', basket)

# Add new fruit throws an error!

basket.add("Orange")

print('After adding new element:', basket)

The output of the above code is as follows:

Unique elements: frozenset({'Kiwi', 'Cherry', 'Apple', 'Banana'})

Traceback (most recent call last):

File "c:\Users\hp\Desktop\set() and frozenset().py", line 37, in <module>

Basket.add("Orange")

AttributeError: 'frozenset' object has no attribute 'add'

The above example shows you can't add a new element to the frozenset.

Let's see how can use a dictionary with frozenset:

student = {"Name": "John", "Age": "25", "Gender": "Male"}

key = frozenset(student)

print("The keys are:", key)

Output:

The keys are: frozenset({'Age', 'Name', 'Gender'})

Let's see other operations that you can perform on frozenset, you can also perform these operations on normal sets:

numbers1 = frozenset([1, 2, 3, 4, 5])

numbers2 = frozenset([2, 3, 4, 5])

# Combining both of them using "|" operator

# You can also use union() method

combined = numbers1 | numbers2

print("Combined set:", combined)

# Selecting common elements using "&" operator

# You can also use intersection() method

intersect = numbers1 & numbers2

print("Intersected set:", intersect)

# Selecting elements which are not common using "-" operator

# You can also use difference() method

difference = numbers1 - numbers2

print("Difference set:", difference)

# Membership testing

# It returns True if sets (frozenset) have no common items otherwise False

Disjoint = numbers1.isdisjoint(numbers2)

print("Disjoint:", Disjoint)

# It returns True if all the items of a set (frozenset) are common in another set (frozenset)

Subset = numbers1.issubset(numbers2)

print("Subset:", Subset)

# It returns True if a set (frozenset) has all items present in another set (frozenset)

Superset = numbers1.issuperset(numbers2)

print("Superset:", Superset)

The output of the above code is as follows:

Combined set: frozenset({1, 2, 3, 4, 5})

Intersected set: frozenset({2, 3, 4, 5})

Difference set: frozenset({1})

Disjoint: False

Subset: False

Superset: True

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

**Answer :**

**Python set Union() Method** returns a new set which contains all the items from the original set.

**Union** of two given sets is the set which contains all the elements of both the sets. The union of two given sets A and B is a set which consists of all the elements of A and all the elements of B such that no element is repeated.

Diagram, venn diagram

Description automatically generated

*The symbol for denoting union of sets is****‘U’***

**Python set Union() Method Syntax:**

***Syntax:****set1.union(set2, set3, set4….)*

***Parameters:****zero or more sets*

***Return:****Returns a set, which has the union of all sets(set1, set2, set3…) with set1. It returns a copy of set1 only if no parameter is passed.*

**Python set Union() Method Example:**

* Python3

|  |
| --- |
| A **=** {2, 4, 5, 6}  B **=** {4, 6, 7, 8}    **print**("A U B:", A.union(B)) |

**Output:**

A U B: {2, 4, 5, 6, 7, 8}

**Example 1: Working with Python set Union() methods**

* Python3

|  |
| --- |
| set1 **=** {2, 4, 5, 6}  set2 **=** {4, 6, 7, 8}  set3 **=** {7, 8, 9, 10}    # union of two sets  print("set1 U set2 : ", set1.union(set2))    # union of three sets  print("set1 U set2 U set3 :", set1.union(set2, set3)) |

**Output**

set1 U set2 : {2, 4, 5, 6, 7, 8}

set1 U set2 U set3 : {2, 4, 5, 6, 7, 8, 9, 10}

**Output:**

set1 U set2 : {2, 4, 5, 6, 7, 8}

set1 U set2 U set3 : {2, 4, 5, 6, 7, 8, 9, 10}

**Example 2: Python Set Union Using the | Operator**

We can use “|” operator to find the union of the sets.

* Python3

|  |
| --- |
| set1 **=** {2, 4, 5, 6}  set2 **=** {4, 6, 7, 8}  set3 **=** {7, 8, 9, 10}    # union of two sets  print("set1 U set2 : ", set1 | set2)    # union of three sets  print("set1 U set2 U set3 :", set1 |set2 | set3) |

**Output**

set1 U set2 : {2, 4, 5, 6, 7, 8}

set1 U set2 U set3 : {2, 4, 5, 6, 7, 8, 9, 10}

**Output:**

set1 U set2 : {2, 4, 5, 6, 7, 8}

set1 U set2 U set3 : {2, 4, 5, 6, 7, 8, 9, 10}

**Example 3 : Python Set Union() Method on String**

* Python3

|  |
| --- |
| A **=** {'ab', 'ba', 'cd', 'dz'}  B **=** {'cd', 'ab', 'dd', 'za'}    **print**("A U B:", A.union(B)) |

**Output**

A U B: {'za', 'ab', 'dd', 'dz', 'ba', 'cd'}

**Example 4 : Python Set Union() Method on multiple set (With 3 Set).**

* Python3

|  |
| --- |
| A **=** {2, 4, 5, 6}  B **=** {4, 6, 7, 8}  c **=** {7, 8, 9, 0}    **print**("A U B:", A.union(B).union(c)) |

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

**Python set intersection() method returns** a new set with an element that is common to all set

The intersection of two given sets is the largest set, which contains all the elements that are **common** to both sets. The intersection of two given sets A and B is a set which consists of all the elements which are common to both A and B.

Diagram, venn diagram

Description automatically generated

**Python Set intersection() Method Syntax:**

***Syntax:****set1.intersection(set2, set3, set4….)****Parameters:***

* *any number of sets can  be passed*

***Return:****Returns a set which has the intersection of all sets(set1, set2, set3…) with set1. It returns a copy of set1 only if no parameter is passed.*

**Python Set intersection() Method Example:**

* Python3

|  |
| --- |
| s1 **=** {1, 2, 3}  s2 **=** {2, 3}  print(s1.intersection(s2)) |

**Output:**

{2, 3}

**Example 1: Working of set intersection()**

* Python3

|  |
| --- |
| # Python3 program for intersection() function  set1 **=** {2, 4, 5, 6}  set2 **=** {4, 6, 7, 8}  set3 **=** {4, 6, 8}    # intersection of two sets  print("set1 intersection set2 : ",        set1.intersection(set2))    # intersection of three sets  print("set1 intersection set2 intersection set3 :",        set1.intersection(set2, set3)) |

**Output:**

set1 intersection set2 : {4, 6}

set1 intersection set2 intersection set3 : {4, 6}

**Example 2: Python set intersection operator(&)**

We can also get intersections using ‘&’ operator.

* Python3

|  |
| --- |
| # Python3 program for intersection() function  set1 **=** {2, 4, 5, 6}  set2 **=** {4, 6, 7, 8}  set3 **=** {1, 0, 12}    print(set1 & set2)  print(set1 & set3)    print(set1 & set2 & set3) |

**Output:**

{4, 6}

set()

set()

**Example 3: Python set intersection opposite**

symmetric\_difference() is an opposite to the Python Set intersection() method.

* Python3

|  |
| --- |
| # Python3 program for intersection() function  set1 **=** {2, 4, 5, 6}  set2 **=** {4, 6, 7, 8}  set3 **=** {1, 0, 12}    print(set1.symmetric\_difference(set2))  print(set1.symmetric\_difference(set3))  print(set2.symmetric\_difference(set3)) |

**Output:**

{2, 5, 7, 8}

{0, 1, 2, 4, 5, 6, 12}

{0, 1, 4, 6, 7, 8, 12}

**Example 4: Python set intersection empty**

Intersection of empty sets returns an empty set.

* Python3

|  |
| --- |
| set1 **=** {}  set2 **=** {}    # union of two sets  print("set1 intersection set2 : ",        set(set1).intersection(set(set2))) |

**Output:**

set1 intersection set2 : set()

Q55. What is dictionary in Python?

**Answer :**

**Dictionary in Python** is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.

## Example of Dictionary in Python

Dictionary holds **key:value** pair. Key-Value is provided in the dictionary to make it more optimized.

* Python3

|  |
| --- |
| Dict **=** {1: 'Geeks', 2: 'For', 3: 'Geeks'}  print(Dict) |

**Output:**

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

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

**Answer :**

**Dictionary in Python** is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.

**Example of Dictionary in Python**

Dictionary holds **key:value** pair. Key-Value is provided in the dictionary to make it more optimized.

* Python3

|  |
| --- |
| Dict **=** {1: 'Geeks', 2: 'For', 3: 'Geeks'}  print(Dict) |

**Output:**

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

**Whereas Data Structures** are a way of organizing data so that it can be accessed more efficiently depending upon the situation. Data Structures are fundamentals of any programming language around which a program is built. Python helps to learn the fundamental of these data structures in a simpler way as compared to other programming languages.

So, to put it differently, a python dict is one example of a data structure. So are the set, list, string and unicode types. A python dict is a highly optimized [hash table](http://en.wikipedia.org/wiki/Hash_table), which is a data structure.

Q57. How can we declare a dictionary in Python?

**Answer** :

In [Python](https://www.geeksforgeeks.org/python-programming-language/), a dictionary can be created by placing a sequence of elements within curly **{}** braces, separated by ‘comma’. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its **Key:value**. Values in a dictionary can be of any data type and can be duplicated, whereas keys can’t be repeated and must be *immutable*.

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

* Python3

|  |
| --- |
| # Creating a Dictionary  # with Integer Keys  Dict **=** {1: 'Geeks', 2: 'For', 3: 'Geeks'}  print("\nDictionary with the use of Integer Keys: ")  **print**(Dict)    # Creating a Dictionary  # with Mixed keys  Dict **=** {'Name': 'Geeks', 1: [1, 2, 3, 4]}  print("\nDictionary with the use of Mixed Keys: ")  print(Dict) |

**Output:**

Dictionary with the use of Integer Keys:

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

Dictionary with the use of Mixed Keys:

{'Name': 'Geeks', 1: [1, 2, 3, 4]}

Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing to curly braces{}.

|  |
| --- |
| # Creating an empty Dictionary  Dict **=** {}  print("Empty Dictionary: ")  **print**(Dict)    # Creating a Dictionary  # with dict() method  Dict **=** dict({1: 'Geeks', 2: 'For', 3: 'Geeks'})  print("\nDictionary with the use of dict(): ")  **print**(Dict)    # Creating a Dictionary  # with each item as a Pair  Dict **=** dict([(1, 'Geeks'), (2, 'For')])  print("\nDictionary with each item as a pair: ")  print(Dict) |

**Output:**

Empty Dictionary:

{}

Dictionary with the use of dict():

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

Dictionary with each item as a pair:

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

#### Complexities for Creating a Dictionary:

***Time complexity:****O(len(dict))*

***Space complexity:****O(n)*

## Nested Dictionary

Diagram

Description automatically generated

* Python3

|  |
| --- |
| # Creating a Nested Dictionary  # as shown in the below image  Dict **=** {1: 'Geeks', 2: 'For',          3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}    print(Dict) |

**Output:**

{1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}

Q58. What will the output of the following?

var = {}

print(type(var))

**Answer**:

<class 'dict'>

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

**Answer**:

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["color"] = "red"  
print(thisdict)

Output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

Update Dictionary

The update() method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

The argument must be a dictionary, or an iterable object with key:value pairs.

ExampleO

Add a color item to the dictionary by using the update() method:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.update({"color": "red"})

Output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

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

**Answer :**

**How to Create a Dictionary in Python**

A dictionary in Python is made up of key-value pairs.

In the two sections that follow you will see two ways of creating a dictionary.

The first way is by using a set of curly braces, {}, and the second way is by using the built-in dict() function.

**How to Create An Empty Dictionary in Python**

To create an empty dictionary, first create a variable name which will be the name of the dictionary.

Then, assign the variable to an empty set of curly braces, {}.

#create an empty dictionary

my\_dictionary = {}

print(my\_dictionary)

#to check the data type use the type() function

print(type(my\_dictionary))

#output

#{}

#<class 'dict'>

Another way of creating an empty dictionary is to use the dict() function without passing any arguments.

It acts as a constructor and creates an empty dictionary:

#create an empty dictionary

my\_dictionary = dict()

print(my\_dictionary)

#to check the data type use the type() function

print(type(my\_dictionary))

#output

#{}

#<class 'dict'>

**How to Create A Dictionary With Items in Python**

To create a dictionary with items, you need to include *key-value* pairs inside the curly braces.

The general syntax for this is the following:

dictionary\_name = {key: value}

Let's break it down:

* dictionary\_name is the variable name. This is the name the dictionary will have.
* = is the assignment operator that assigns the key:value pair to the dictionary\_name.
* You declare a dictionary with a set of curly braces, {}.
* Inside the curly braces you have a key-value pair. Keys are separated from their associated values with colon, :.

Let's see an example of creating a dictionary with items:

#create a dictionary

my\_information = {'name': 'Dionysia', 'age': 28, 'location': 'Athens'}

print(my\_information)

#check data type

print(type(my\_information))

#output

#{'name': 'Dionysia', 'age': 28, 'location': 'Athens'}

#<class 'dict'>

In the example above, there is a sequence of elements within the curly braces.

Specifically, there are three key-value pairs: 'name': 'Dionysia', 'age': 28, and 'location': 'Athens'.

The keys are name, age, and location. Their associated values are Dionysia, 28, and Athens, respectively.

When there are multiple key-value pairs in a dictionary, each key-value pair is separated from the next with a comma, ,.

Let's see another example.

Say that you want to create a dictionary with items using the dict() function this time instead.

You would achieve this by using dict() and passing the curly braces with the sequence of key-value pairs enclosed in them as an argument to the function.

#create a dictionary with dict()

my\_information = dict({'name': 'Dionysia' ,'age': 28,'location': 'Athens'})

print(my\_information)

#check data type

print(type(my\_information))

#output

#{'name': 'Dionysia', 'age': 28, 'location': 'Athens'}

#<class 'dict'>

It's worth mentioning the fromkeys() method, which is another way of creating a dictionary.

It takes a predefined sequence of items as an argument and returns a new dictionary with the items in the sequence set as the dictionary's specified keys.

You can *optionally* set a value for all the keys, but by default the value for the keys will be None.

The general syntax for the method is the following:

dictionary\_name = dict.fromkeys(sequence,value)

Let's see an example of creating a dictionary using fromkeys() without setting a value for all the keys:

#create sequence of strings

cities = ('Paris','Athens', 'Madrid')

#create the dictionary, `my\_dictionary`, using the fromkeys() method

my\_dictionary = dict.fromkeys(cities)

print(my\_dictionary)

#{'Paris': None, 'Athens': None, 'Madrid': None}

Now let's see another example that sets a value that will be the same for all the keys in the dictionary:

#create a sequence of strings

cities = ('Paris','Athens', 'Madrid')

#create a single value

continent = 'Europe'

my\_dictionary = dict.fromkeys(cities,continent)

print(my\_dictionary)

#output

#{'Paris': 'Europe', 'Athens': 'Europe', 'Madrid': 'Europe'}

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

**Answer:**

## Accessing an element of a nested dictionary

In order to access the value of any key in the nested dictionary, use indexing [] syntax.

* Python3

|  |
| --- |
| # Creating a Dictionary  Dict **=** {'Dict1': {1: 'Geeks'},          'Dict2': {'Name': 'For'}}    # Accessing element using key  **print**(Dict['Dict1'])  **print**(Dict['Dict1'][1])  print(Dict['Dict2']['Name']) |

**Output:**

{1: 'Geeks'}

Geeks

For

## ****Dictionary methods****

* [**clear()**](https://www.geeksforgeeks.org/python-dictionary-clear/)**–**Remove all the elements from the dictionary
* [**copy()**](https://www.geeksforgeeks.org/python-dictionary-copy/)**–**Returns a copy of the dictionary
* [**get()**](https://www.geeksforgeeks.org/get-method-dictionaries-python/)**–**Returns the value of specified key
* [**items()**](https://www.geeksforgeeks.org/python-dictionary-items-method/)**–**Returns a list containing a tuple for each key value pair
* [**keys()**](https://www.geeksforgeeks.org/python-dictionary-keys-method/)**–**Returns a list containing dictionary’s keys
* [**pop()**](https://www.geeksforgeeks.org/python-dictionary-pop-method/)**–** Remove the element with specified key
* [**popitem()**](https://www.geeksforgeeks.org/python-dictionary-popitem-method/)**–**Removes the last inserted key-value pair
* [**update()**](https://www.geeksforgeeks.org/python-dictionary-update-method/)**–**Updates dictionary with specified key-value pairs
* [**values()**](https://www.geeksforgeeks.org/python-dictionary-values/)**–** Returns a list of all the values of dictionary
* Python3

|  |
| --- |
| # demo for all dictionary methods  dict1 **=** {1: "Python", 2: "Java", 3: "Ruby", 4: "Scala"}    # copy() method  dict2 **=** dict1.copy()  **print**(dict2)    # clear() method  dict1.clear()  **print**(dict1)    # get() method  **print**(dict2.get(1))    # items() method  **print**(dict2.items())    # keys() method  print(dict2.keys())    # pop() method  dict2.pop(4)  print(dict2)    # popitem() method  dict2.popitem()  print(dict2)    # update() method  dict2.update({3: "Scala"})  **print**(dict2)    # values() method  print(dict2.values()) |

**Output:**

{1: 'Python', 2: 'Java', 3: 'Ruby', 4: 'Scala'}

{}

Python

dict\_items([(1, 'Python'), (2, 'Java'), (3, 'Ruby'), (4, 'Scala')])

dict\_keys([1, 2, 3, 4])

{1: 'Python', 2: 'Java', 3: 'Ruby'}

{1: 'Python', 2: 'Java'}

{1: 'Python', 2: 'Java', 3: 'Scala'}

dict\_values(['Python', 'Java', 'Scala'])

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

**Answer** :

The get() method returns the value of the item with the specified key.

Syntax

*dictionary*.get(*keyname, value*)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *Keyname* | Required. The keyname of the item you want to return the value from |
| *Value* | Optional. A value to return if the specified key does not exist. Default value None |

Example

Try to return the value of an item that do not exist:

car = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
  
x = car.get("price", 15000)  
  
print(x)

**After Run:**

**15000**

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

**Answer**: In Python Dictionary, **items()** method is used to return the list with all dictionary keys with values.

***Syntax:****dictionary.items()****Parameters:****This method takes no parameters.****Returns:****A view object that displays a list of a given dictionary’s (key, value) tuple pair.*

**Example #1:** 

* Python3

|  |
| --- |
| # Python program to show working  # of items() method in Dictionary    # Dictionary with three items  Dictionary1 **=** { 'A': 'Geeks', 'B': 4, 'C': 'Geeks' }    print("Dictionary items:")    # Printing all the items of the Dictionary  print(Dictionary1.items()) |

**Output:** 

Dictionary items:

dict\_items([('A', 'Geeks'), ('B', 4), ('C', 'Geeks')])

Order of these items in the list may not always be same.   
    
**Example #2:** To show working of items() after modification of Dictionary. 

* Python3

|  |
| --- |
| # Python program to show working  # of items() method in Dictionary    # Dictionary with three items  Dictionary1 **=** { 'A': 'Geeks', 'B': 4, 'C': 'Geeks' }    print("Original Dictionary items:")    items **=** Dictionary1.items()    # Printing all the items of the Dictionary  print(items)    # Delete an item from dictionary  **del**[Dictionary1['C']]  print('Updated Dictionary:')  print(items) |

**Output:** 

Original Dictionary items:

dict\_items([('A', 'Geeks'), ('C', 'Geeks'), ('B', 4)])

Updated Dictionary:

dict\_items([('A', 'Geeks'), ('B', 4)])

If the Dictionary is updated anytime, the changes are reflected in the view object automatically.

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

**Answer**: The pop() method removes the specified item from the dictionary.

The value of the removed item is the return value of the pop() method, see example below.

Syntax

*dictionary*.pop(*keyname, defaultvalue*)

Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *keyname* | Required. The keyname of the item you want to remove |
| *defaultvalue* | Optional. A value to return if the specified key do not exist.  If this parameter is not specified, and the no item with the specified key is found, an error is raised |

More Examples

Example

The value of the removed item is the return value of the pop() method:

car = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
  
x = car.pop("model")  
  
print(x)

After run:

Mustang

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

**Answer:**

**Python dictionary popitem() method** removes the last inserted key-value pair from the dictionary and returns it as a tuple.

**Python Dictionary popitem() Method Syntax:**

***Syntax :****dict.popitem()*

***Parameters :****None*

***Returns :****A tuple containing the arbitrary key-value pair from dictionary. That pair is removed from dictionary.*

**Note:**popitem() method return keyError if dictionary is empty.

**Python Dictionary popitem() Method Example:**

* Python3

|  |
| --- |
| d **=** {1: '001', 2: '010', 3: '011'}  print(d.popitem()) |

**Output:**

(3, '011')

**Example  1: Demonstrating the use of popitem()**

Here we are going to use[**Python**](https://www.geeksforgeeks.org/python-programming-language/)**dict popitem() method**to pop the last element.

* Python3

|  |
| --- |
| test\_dict **=** {"Nikhil": 7, "Akshat": 1, "Akash": 2}    # Printing initial dict  **print**("Before using popitem(), test\_dict: ", test\_dict)    # using popitem() to return+  # remove the last keym value pair  res **=** test\_dict.popitem()    # Printing the pair returned  print('The key, value pair returned is : ', res)    # Printing dict after deletion  print("After using popitem(), test\_dict: ", test\_dict) |

**Output :**

Before using popitem(), test\_dict: {'Nikhil': 7, 'Akshat': 1, 'Akash': 2}

The key, value pair returned is : ('Akash', 2)

After using popitem(), test\_dict: {'Nikhil': 7, 'Akshat': 1}

**Practical Application:**This particular function can be used to remove items and it’s details one by one.

**Example 2: Demonstrating the application of popitem()**

* Python3

|  |
| --- |
| # Python 3 code to demonstrate  # application of popitem()    # initializing dictionary  test\_dict **=** {"Nikhil": 7, "Akshat": 1, "Akash": 2}    # Printing initial dict  print("The dictionary before deletion : " **+** str(test\_dict))    n **=** len(test\_dict)    # using popitem to assign ranks  **for** i **in** range(0, n):      print("Rank " **+** str(i **+** 1) **+** " " **+** str(test\_dict.popitem()))    # Printing end dict  print("The dictionary after deletion : " **+** str(test\_dict)) |

**Output :**

The dictionary before deletion : {'Nikhil': 7, 'Akshat': 1, 'Akash': 2}

Rank 1 ('Akash', 2)

Rank 2 ('Akshat', 1)

Rank 3 ('Nikhil', 7)

The dictionary after deletion : {}

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

**Answer**: The **keys()** method in [Python Dictionary](https://www.geeksforgeeks.org/python-dictionary/), returns a view object that displays a list of all the keys in the dictionary in order of insertion using [Python](https://www.geeksforgeeks.org/python-programming-language/).

***Syntax:****dict.keys()*

***Parameters:****There are no parameters.*

***Returns:****A view object is returned that displays all the keys. This view object changes according to the changes in the dictionary.*

**Method 1: Accessing the key using the keys() function**

A simple example to show how the keys() function works in the dictionary.

* Python3

|  |
| --- |
| # Dictionary with three keys  Dictionary1 **=** {'A': 'Geeks', 'B': 'For', 'C': 'Geeks'}    # Printing keys of dictionary  print(Dictionary1.keys()) |

**Output:**

dict\_keys(['A', 'B', 'C'])

**Method 2: Python access dictionary by key**

Demonstrating the practical application of keys() using the [Python loop](https://www.geeksforgeeks.org/loops-in-python/).

* Python3

|  |
| --- |
| # initializing dictionary  test\_dict **=** {"geeks": 7, "for": 1, "geeks": 2}    # accessing 2nd element using naive method  # using loop  j **=** 0  **for** i **in** test\_dict:  **if** (j **==** 1):          print('2nd key using loop : ' **+** i)      j **=** j **+** 1 |

**Output:**

2nd key using loop : for

TypeError: 'dict\_keys' object does not support indexing

**Note:**The second approach would not work because ***dict\_keys***in Python 3 do not support indexing.

**Method 3: Accessing key using keys() indexing**

Here, we first extracted all the keys and then we implicitly converted them into the [Python list](https://www.geeksforgeeks.org/python-list/) to access the element from it.

* Python3

|  |
| --- |
| # initializing dictionary  test\_dict **=** {"geeks": 7, "for": 1, "geeks": 2}    # accessing 2nd element using keys()  print('2nd key using keys() : ', list(test\_dict.keys())[1]) |

**Output:**

2nd key using keys() : for

**Method 4: Python Dictionary update() function**

To show how to update the dictionary keys using the [update() function](https://www.geeksforgeeks.org/python-dictionary-update-method/). Here, when the dictionary is updated, keys are also automatically updated to show the changes.

* Python3

|  |
| --- |
| # Dictionary with two keys  Dictionary1 **=** {'A': 'Geeks', 'B': 'For'}    # Printing keys of dictionary  print("Keys before Dictionary Updation:")  keys **=** Dictionary1.keys()  print(keys)    # adding an element to the dictionary  Dictionary1.update({'C': 'Geeks'})    print('\nAfter dictionary is updated:')  print(keys) |

**Output:**

Keys before Dictionary Updation:

dict\_keys(['B', 'A'])

After dictionary is updated:

dict\_keys(['B', 'A', 'C'])

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

**Answer**: The values() method returns a view object. The view object contains the values of the dictionary, as a list.

The view object will reflect any changes done to the dictionary, see example below.

Syntax

*dictionary*.values()

Parameter Values

No parameters

More Examples

Example

When a values is changed in the dictionary, the view object also gets updated:

car = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
  
x = car.values()  
  
car["year"] = 2018  
  
print(x)

After run:  
dict\_values(['Ford', 'Mustang', 2018])

Q68. What are loops in Python?

**Answer**: A loop is a command that tends to repeat itself to obtain the desired result. In other words, a programming command that repeats itself either the known number of times or the unknown number of times to fulfill certain conditions is defined as a loop.

# **Loops in Python with Examples**

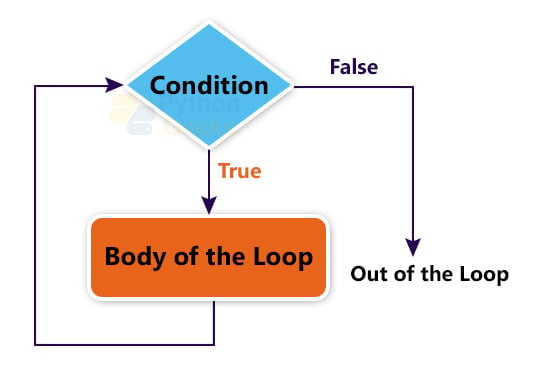
Generally, a loop is something that coils around itself. Loops in the programming context have a similar meaning. In this article, we will learn different types of loops in Python and discuss each of them in detail with examples. So let us begin.

### Introduction to Loops in Python

In programming, the loops are the constructs that repeatedly execute a piece of code based on the conditions. These are useful in many situations like going through every element of a list, doing an operation on a range of values, etc.

There are two types of loops in Python and these are for and while loops. Both of them work by following the below steps:

1. Check the condition  
2. If True, execute the body of the block under it. And update the iterator/ the value on which the condition is checked.  
3. If False, come out of the loop

[](https://pythongeeks.org/wp-content/uploads/2021/07/Working-of-Loops.jpg)

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

**Answer** : Python programming language provides the following types of loops to handle looping requirements. Python provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

**While Loop in Python**

In python, a [while loop](https://www.geeksforgeeks.org/python-while-loop/) is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

**Syntax** :

while expression:

statement(s)

All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.   
Example:

* Python

|  |
| --- |
| # Python program to illustrate  # while loop  count **=** 0  **while** (count < 3):      count **=** count **+** 1      print("Hello Geek") |

**Output:**

Hello Geek

Hello Geek

Hello Geek

**Using else statement with while loops**

As discussed above, while loop executes the block until a condition is satisfied. When the condition becomes false, the statement immediately after the loop is executed.

The else clause is only executed when your while condition becomes false. If you break out of the loop, or if an exception is raised, it won’t be executed.

**If else like this:**

if condition:

# execute these statements

else:

# execute these statements

**and while loop like this are similar**

while condition:

# execute these statements

else:

# execute these statements

**Examples:**

* Python

|  |
| --- |
| # Python program to illustrate  # combining else with while  count **=** 0  **while** (count < 3):      count **=** count **+** 1      print("Hello Geek")  **else**:      print("In Else Block") |

**Output:**

Hello Geek

Hello Geek

Hello Geek

In Else Block

**Single statement while block**

Just like the if block, if the while block consists of a single statement then we can declare the entire loop in a single line as shown below:

* Python

|  |
| --- |
| # Python program to illustrate  # Single statement while block  count **=** 0  **while** (count **==** 0): print("Hello Geek") |

**Note**: It is suggested **not to use** this type of loops as it is a never ending infinite loop where the condition is always true and you have to forcefully terminate the compiler.

**For Loop in Python**

[For loops](https://www.geeksforgeeks.org/python-for-loops/) are used for sequential traversal. For example: traversing a list or string or array etc. In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is “for in” loop which is similar to [for each](https://www.geeksforgeeks.org/g-fact-40-foreach-in-c-and-java/) loop in other languages. Let us learn how to use for in loop for sequential traversals.

**Syntax:**

for iterator\_var in sequence:

statements(s)

It can be used to iterate over a range and iterators.

* Python3

|  |
| --- |
| # Python program to illustrate  # Iterating over range 0 to n-1    n **=** 4  **for** i **in** range(0, n):      print(i) |

**Output :**

0

1

2

3

**Example with List, Tuple, string, and dictionary iteration using For Loops**

* Python

|  |
| --- |
| # Python program to illustrate  # Iterating over a list  **print**("List Iteration")  l **=** ["geeks", "for", "geeks"]  **for** i **in** l:      print(i)    # Iterating over a tuple (immutable)  **print**("\nTuple Iteration")  t **=** ("geeks", "for", "geeks")  **for** i **in** t:      print(i)    # Iterating over a String  **print**("\nString Iteration")  s **=** "Geeks"  **for** i **in** s:  **print**(i)    # Iterating over dictionary  **print**("\nDictionary Iteration")  d **=** dict()  d['xyz'] **=** 123  d['abc'] **=** 345  **for** i **in** d:  **print**("%s  %d" **%** (i, d[i]))    # Iterating over a set  **print**("\nSet Iteration")  set1 **=** {1, 2, 3, 4, 5, 6}  **for** i **in** set1:      print(i), |

**Output:**

List Iteration

geeks

for

geeks

Tuple Iteration

geeks

for

geeks

String Iteration

G

e

e

k

s

Dictionary Iteration

xyz 123

abc 345

**Iterating by the index of sequences:**

We can also use the index of elements in the sequence to iterate. The key idea is to first calculate the length of the list and in iterate over the sequence within the range of this length.   
See the below example:

* Python

|  |
| --- |
| # Python program to illustrate  # Iterating by index    list **=** ["geeks", "for", "geeks"]  **for** index **in** range(len(list)):      print list[index] |

**Output:**

geeks

for

geeks

**Using else statement with for loops:**

We can also combine else statement with for loop like in while loop. But as there is no condition in for loop based on which the execution will terminate so the else block will be executed immediately after for block finishes execution.   
Below example explains how to do this:

* Python

|  |
| --- |
| # Python program to illustrate  # combining else with for    list **=** ["geeks", "for", "geeks"]  **for** index **in** range(len(list)):  **print** (list[index])  **else**:      print ("Inside Else Block") |

**Output:**

geeks

for

geeks

Inside Else Block

**Nested Loops**

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

**Syntax:**

for iterator\_var in sequence:

for iterator\_var in sequence:

statements(s)

statements(s)

The syntax for a nested while loop statement in the Python programming language is as follows:

while expression:

while expression:

statement(s)

statement(s)

A final note on loop nesting is that we can put any type of loop inside of any other type of loop. For example, a for loop can be inside a while loop or vice versa.

* Python

|  |
| --- |
| # Python program to illustrate  # nested for loops in Python  **from** \_\_future\_\_ **import** print\_function  **for** i **in** range(1, 5):  **for** j **in** range(i):  **print**(i, end**=**' ')      print() |

**Output:**

1

2 2

3 3 3

4 4 4 4

**Loop Control Statements**

Loop control statements change execution from their normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

**Continue Statement:**

It returns the control to the beginning of the loop.

* Python

|  |
| --- |
| # Prints all letters except 'e' and 's'  **for** letter **in** 'geeksforgeeks':  **if** letter **==** 'e' **or** letter **==** 's':  **continue**      print ('Current Letter :', letter)      var **=** 10 |

**Output:**

Current Letter : g

Current Letter : k

Current Letter : f

Current Letter : o

Current Letter : r

Current Letter : g

Current Letter : k

**Break Statement:**

It brings control out of the loop

* Python

|  |
| --- |
| **for** letter **in** 'geeksforgeeks':        # break the loop as soon it sees 'e'      # or 's'  **if** letter **==** 'e' **or** letter **==** 's':  **break**    print 'Current Letter :', letter |

**Output:**

Current Letter : e

**Pass Statement:**

We use pass statement to write empty loops. Pass is also used for empty control statements, functions and classes.

* Python

|  |
| --- |
| # An empty loop  **for** letter **in** 'geeksforgeeks':  **pass**  print 'Last Letter :', letter |

**Output:**

Last Letter : s

**How for loop in Python works internally?**

Before proceeding to this section, you should have a prior understanding of Python Iterators.

Firstly, lets see how a simple for loop looks like.

* Python3

|  |
| --- |
| # A simple for loop example    fruits **=** ["apple", "orange", "kiwi"]    **for** fruit **in** fruits:     print(fruit) |

**Output**

apple

orange

kiwi

Here we can see the for loops iterates over iterable object fruit which is a list. Lists, sets, dictionaries are few iterable objects while an integer object is not an iterable object.

For loops can iterate over any iterable object (example: List, Set, Dictionary, Tuple or String).

Now with the help of the above example, let’s dive deep and see what happens internally here.

1. Make the list (iterable) an iterable object with help of the iter() function.
2. Run an infinite while loop and break only if the StopIteration is raised.
3. In the try block, we fetch the next element of fruits with the next() function.
4. After fetching the element we did the operation to be performed with the element. (i.e print(fruit))

* Python3

|  |
| --- |
| fruits **=** ["apple", "orange", "kiwi"]    # Creating an iterator object  # from that iterable i.e fruits  iter\_obj **=** iter(fruits)    # Infinite while loop  **while** True:  **try**:      # getting the next item      fruit **=** next(iter\_obj)  **print**(fruit)  **except** StopIteration:        # if StopIteration is raised,      # break from loop  **Break** |

**Output**

apple

orange

kiwi

We can see that under the hood we are calling iter() and next() method.

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

**Answer** : There are various kinds of loops such as for loop, while loop, if loop, if-else loop, if-else-if loop, etc. But the most commonly used loops are for and while loops.

## ****For loop vs While loop****

The difference between for loop and while loop is that in for loop the number of iterations to be done is already known and is used to obtain a certain result whereas in while loop the command runs until a certain condition is reached and the statement is proved to be false.

Q71. What is the use of continue statement?

**Answer : Python Continue Statement**skips the execution of the program block from after the continue statement and forces the control to start the next iteration.

**Python Continue statement** is a loop control statement that forces to execute the next iteration of the loop while skipping the rest of the code inside the loop for the current iteration only, i.e. when the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped for the current iteration and the next iteration of the loop will begin.

**Python continue Statement Syntax**

while True:

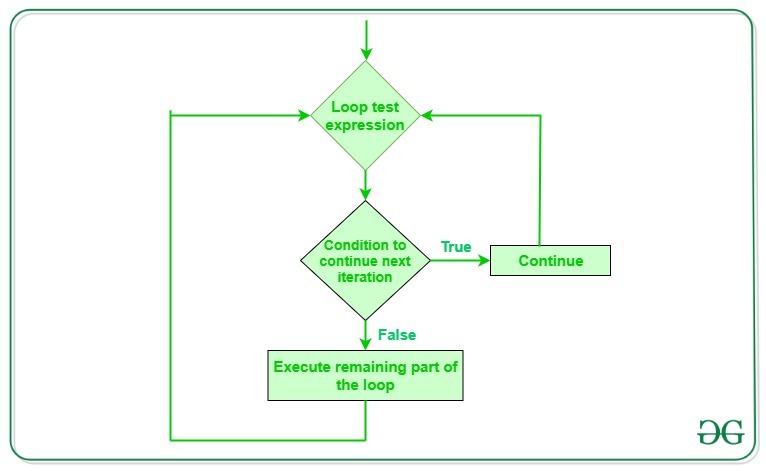
...

if x == 10:

continue

print(x)

**Flowchart of Continue Statement**



*flowchart of Python continue statement*

**Continue statement in Python Example**

**Example 1: Demonstration of Continue statement in Python**

In this example, we will use continue inside some condition within a loop.

* Python3

|  |
| --- |
| **for** var **in** "Geeksforgeeks":  **if** var **==** "e":  **continue**      print(var) |

**Output:**

G

k

s

f

o

r

g

k

s

Explanation: Here we are skipping the print of character ‘e’ using[if-condition](https://www.geeksforgeeks.org/if-else-statement-in-javascript/) checking and continue statement.

**Example 2: Printing range with Python Continue Statement**

Consider the situation when you need to write a program which prints the number from 1 to 10, but not 6.

It is specified that you have to do this using loop and only one loop is allowed to use. Here comes the usage of the continue statement. What we can do here is we can run a loop from 1 to 10 and every time we have to compare the value of the loop variable with 6. If it is equal to 6 we will use the continue statement to continue to the next iteration without printing anything, otherwise we will print the value.

* Python3

|  |
| --- |
| # loop from 1 to 10  **for** i **in** range(1, 11):        # If i is equals to 6,      # continue to next iteration      # without printing  **if** i **==** 6:  **continue**  **else**:          # otherwise print the value          # of i          print(i, end**=**" ") |

**Output:**

1 2 3 4 5 7 8 9 10

**Note:** The continue statement can be used with any other loop also like [while loop,](https://www.geeksforgeeks.org/python-while-loop/) similarly as it is used with [for loop](https://www.geeksforgeeks.org/python-for-loops/) above.

**Usage of Continue Statement**

[Loops](https://www.geeksforgeeks.org/loops-in-python/) in Python automate and repeat the tasks efficiently. But sometimes, there may arise a condition where you want to exit the loop completely, skip an iteration or ignore that condition. These can be done by [loop control statements](https://www.geeksforgeeks.org/break-continue-and-pass-in-python/). Continue is a type of loop control statement that can alter the flow of the loop.

Q72. What is the use of break statement?

**Answer : Python break**is used to terminate the execution of the loop.

**Python break statement Syntax:**

Loop{

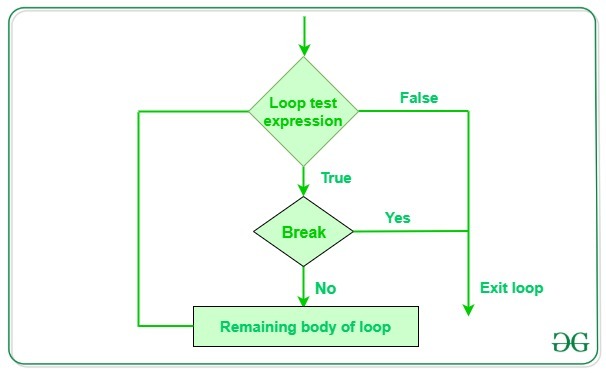
Condition:

break

}

**Python break statement**

**break statement in**[**Python**](https://www.geeksforgeeks.org/python-programming-language/) is used to bring the control out of the loop when some external condition is triggered. break statement is put inside the loop body (generally after if condition).  It terminates the current loop, i.e., the loop in which it appears, and resumes execution at the next statement immediately after the end of that loop. If the break statement is inside a nested loop, the break will terminate the innermost loop.



**Example of Python break statement**

**Example 1:**

* Python3

|  |
| --- |
| **for** i **in** range(10):      print(i)  **if** i **==** 2:  **break** |

**Output:**

0

1

2

**Example 2:**

* Python3

|  |
| --- |
| # Python program to  # demonstrate break statement    s **=** 'geeksforgeeks'  # Using for loop  **for** letter **in** s:    **print**(letter)      # break the loop as soon it sees 'e'      # or 's'  **if** letter **==** 'e' **or** letter **==** 's':  **break**    print("Out of for loop"    )  **print**()    i **=** 0    # Using while loop  **while** True:      print(s[i])        # break the loop as soon it sees 'e'      # or 's'  **if** s[i] **==** 'e' **or** s[i] **==** 's':  **break**      i **+=** 1    print("Out of while loop ") |

**Output:**

g

e

Out of for loop

g

e

Out of while loop

In the above example, both the loops are iterating the string ‘geeksforgeeks’ and as soon as they encounter the character ‘e’ or ‘s’, if the condition becomes true and the flow of execution is brought out of the loop.

**Example 3:**

* Python3

|  |
| --- |
| num **=** 0  **for** i **in** range(10):      num **+=** 1  **if** num **==** 8:  **break**  **print**("The num has value:", num)  print("Out of loop") |

**Output**

The num has value: 1

The num has value: 2

The num has value: 3

The num has value: 4

The num has value: 5

The num has value: 6

The num has value: 7

Out of loop

In the above example, after iterating till num=7, the value of num will be 8 and the break is encountered so the flow of the execution is brought out of the loop.

Q73. What is the use of pass statement?

**Answer** : The **pass** statement is a null statement. But the difference between pass and comment is that comment is ignored by the interpreter whereas pass is not ignored.

**Python pass Statement Syntax:**

pass

**Python pass Statement Example**

When the user does not know what code to write, So user simply places **pass**at that line. Sometimes, **pass** is used when the user doesn’t want any code to execute. So user can simply place **pass**where empty code is not allowed, like in loops, function definitions, class definitions, or in if statements. So using pass statement user avoids this error.

**Example 1:**Pass statement can be used in empty functions

* Python3

|  |
| --- |
| **def** function:  **pass** |

**Example 2:**pass statement can also be used in empty class

* Python3

|  |
| --- |
| **class** geekClass:  **pass** |

**Example 3:**pass statement can be used in for loop when user doesn’t know what to code inside the loop

* Python3

|  |
| --- |
| n **=** 10  **for** i **in** range(n):      # pass can be used as placeholder    # when code is to added later  **Pass** |

**Example 4:**pass statement can be used with conditional statements

* Python3

|  |
| --- |
| a **=** 10  b **=** 20    **if**(a<b):  **pass**  **else**:    print("b<a") |

**Example 5:**lets take another example in which the pass statement get executed when the condition is true

* Python3

|  |
| --- |
| li **=**['a', 'b', 'c', 'd']    **for** i **in** li:  **if**(i **==**'a'):  **pass**  **else**:          print(i) |

**Output:**

b

c

d

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

**Answer** : The Python**range() function**returns a sequence of numbers, in a given range. The most common use of it is to iterate sequence on a sequence of numbers using [Python](https://www.geeksforgeeks.org/python-programming-language/) loops.

**Syntax of Python range() function**

***Syntax:****range(start, stop, step)*

***Parameter:***

* ***start****: [ optional ] start value of the sequence*
* ***stop:****next value after the end value of the sequence*
* ***step:****[ optional ]**integer value, denoting the difference between any two numbers in the sequence.*

***Return:****Returns a range type object.*

**Example of Python range() function**

* Python3

|  |
| --- |
| # print first 5 integers  # using python range() function  **for** i **in** range(5):      print(i, end**=**" ")  print() |

**Output:**

0 1 2 3 4

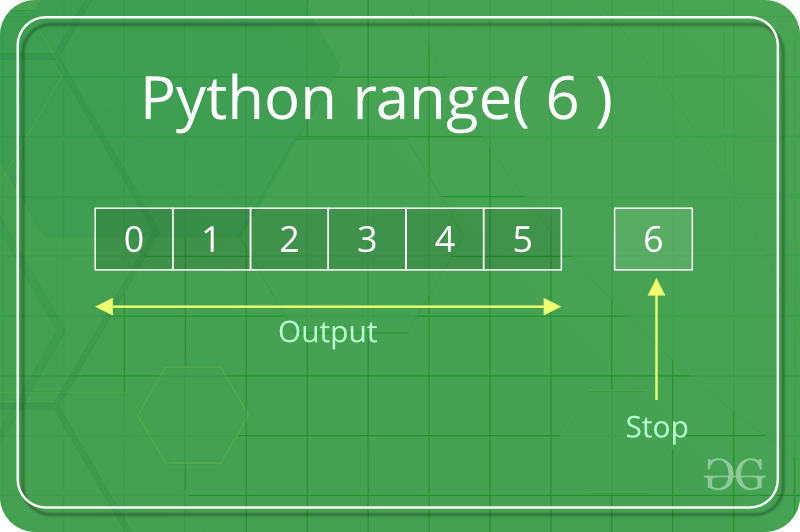
**What is the use of the range function in Python**

In simple terms, range() allows the user to generate a series of numbers within a given range. Depending on how many arguments the user is passing to the function, the user can decide where that series of numbers will begin and end, as well as how big the difference will be between one number and the next. Python range() function takes can be initialized in 3 ways.

* range (stop) takes one argument.
* range (start, stop) takes two arguments.
* range (start, stop, step) takes three arguments.

**Python range (stop)**

When the user call range() with one argument, the user will get a series of numbers that starts at 0 and includes every whole number up to, but not including, the number that the user has provided as the stop.



*Python range visualization*

**Example: Demonstration of Python range (stop)**

* Python3

|  |
| --- |
| # printing first 6  # whole number  **for** i **in** range(6):      print(i, end**=**" ")  print() |

**Output:**

0 1 2 3 4 5

**Python range (start, stop)**

When the user call **range()** with two arguments, the user gets to decide not only where the series of numbers stops but also where it starts, so the user don’t have to start at 0 all the time. Users can use range() to generate a series of numbers from X to Y using range(X, Y).

A picture containing timeline

Description automatically generated

*Python range visualization*

**Example:  Demonstration of Python range (start, stop)**

* Python3

|  |
| --- |
| # printing a natural  # number from 5 to 20  **for** i **in** range(5, 20):      print(i, end**=**" ") |

**Output:**

5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

**Python range (start, stop, step)**

When the user call range() with three arguments, the user can choose not only where the series of numbers will start and stop, but also how big the difference will be between one number and the next. If the user doesn’t provide a step, then range() will automatically behave as if the step is 1. In this example, we are printing even numbers between 0 and 10, so we choose our starting point from 0(start = 0) and stop the series at 10(stop = 10). For printing an even number the difference between one number and the next must be 2 (step = 2) after providing a step we get the following output (0, 2, 4, 8).

A picture containing diagram

Description automatically generated

*Python range visualization*

**Example:  Demonstration of Python range (start, stop, step)**

* Python3

|  |
| --- |
| **for** i **in** range(0, 10, 2):      print(i, end**=**" ")  print() |

**Output:**

0 2 4 6 8

**Python range() with Examples**

**Example 1: Incrementing the range using a positive step**

If a user wants to increment, then the user needs steps to be a positive number.

* Python3

|  |
| --- |
| # incremented by 4  **for** i **in** range(0, 30, 4):      print(i, end**=**" ")  print() |

**Output :**

0 4 8 12 16 20 24 28

**Example 2: Python range() using negative step**

If a user wants to decrement, then the user needs steps to be a negative number.

* Python3

|  |
| --- |
| # incremented by -2  **for** i **in** range(25, 2, **-**2):      print(i, end**=**" ")  print() |

**Output :**

25 23 21 19 17 15 13 11 9 7 5 3

**Example 3: Python range() with float**

Python range() function doesn’t support the float numbers. i.e. user cannot use floating-point or non-integer numbers in any of its argument. Users can use only integer numbers.

* Python3

|  |
| --- |
| # using a float number  **for** i **in** range(3.3):      print(i) |

**Output :**

for i in range(3.3):

TypeError: 'float' object cannot be interpreted as an integer

**Example 4: Concatenation of two range() functions using itertools chain() method**

The result from two range() functions can be concatenated by using the chain() method of [itertools module](https://www.geeksforgeeks.org/python-itertools/). The chain() method is used to print all the values in iterable targets one after another mentioned in its arguments.

* Python3

|  |
| --- |
| **from** itertools **import** chain    # Using chain method  print("Concatenating the result")  res **=** chain(range(5), range(10, 20, 2))    **for** i **in** res:  **print**(i, end**=**" ") |

**Output:**

Concatenating the result

0 1 2 3 4 10 12 14 16 18

**Example 5: Accessing range() with an index value**

A sequence of numbers is returned by the range() function as its object that can be accessed by its index value. Both positive and negative indexing is supported by its object.

* Python3

|  |
| --- |
| ele **=** range(10)[0]  print("First element:", ele)    ele **=** range(10)[**-**1]  **print**("\nLast element:", ele)    ele **=** range(10)[4]  print("\nFifth element:", ele) |

**Output:**

First element: 0

Last element: 9

Fifth element: 4

**Some Important points to remember about the Python range() function:**

* range() function only works with the integers, i.e. whole numbers.
* All arguments must be integers. Users can not pass a string or float number or any other type in a **start**, **stop** and **step** argument of a range().
* All three arguments can be positive or negative.
* The **step** value must not be zero. If a step is zero, python raises a ValueError exception.
* range() is a type in Python
* Users can access items in a range() by index, just as users do with a list:

Q75. How can you loop over a dictionary?

**Answer** : Here are some of the ways you can deal with a Python dictionary using loops.

**Looping Through Keys and Values**

A dictionary in Python contains key-value pairs. You can iterate through its keys using the **keys()** method:

myDict = {"A" : 2, "B" : 5, "C" : 6}  
  
**for** **i** **in** **myDict**.keys():  
print("Key"+" "+i)  
  
<**strong**>Output:  
Key A  
Key B  
Key Cstrong>

The above code is slightly more verbose than you need, though. You can access the keys by calling them directly from **myDict** without using **myDict.keys()**.

That's because a Python for loop picks the keys by default when it sees a dictionary. So you can write the above code like this instead:

for key in myDict:  
print("Key"+ " " +key)  
  
<**strong**>Output:  
Key A  
Key B  
Key Cstrong>

To access the values, use the corresponding **values()** method:

myDict = {"A" : 2, "B" : 5, "C" : 6}  
  
**for** **i** **in** **myDict**.values():  
**print**(i)  
  
<**strong**>Output:  
2  
5  
6strong>

Similarly, you can access the values directly using their keys:

for key in myDict:  
**print**(myDict[key])  
  
<**strong**>Output:  
2  
5  
6strong>

While iterating through a dictionary, you can access its keys and values at the same time. You can do this in two ways.

The first method is to iterate through the dictionary and access the values using the **dict[key]** method. Then print each key-value pair within the loop:

for key in myDict:  
print(key, "|", myDict[key])  
  
<**strong**>Output:  
A | 2  
B | 5  
C | 6strong>

Alternatively, you can access the keys and values simultaneously using the **items()** method:

**for** **key**, **value** **in** **myDict**.items():  
print(key, "|", value)  
  
<**strong**>Output:  
A | 2  
B | 5  
C | 6strong>

Sometimes, you might want to output the result in reverse order. Here's how to do that using the **sorted()** function:

myDict = {"A" : 2, "B" : 5, "C" : 6}  
  
**for** key, value in sorted(myDict.items(), reverse=**True**):  
print(key, "|", value)  
  
<**strong**>Output:  
C | 6  
B | 5  
A | 2strong>

**Converting a Dictionary Into a List**

Converting a dictionary into a list using iteration is as easy as [**transforming a list into a dictionary**](https://www.makeuseof.com/convert-list-into-dictionary-python/).

You can create a list containing an individual tuple for each key-value pair:

myDict = {"A" : "MUO", "B" : "Google", "C" : "Python"}  
myList = []  
**for** **key**, **value** **in** **myDict**.items():  
**myList**.append((**key**, **value**))  
**print**(myList)  
  
<**strong**>Output: [('A', 'MUO'), ('B', 'Google'), ('C', 'Python')]strong>

Or you can convert the dictionary into a nested list of key-value pairs:

myDict = {"A" : "MUO", "B" : "Google", "C" : "Python"}  
myList = []  
**for** **key**, **value** **in** **myDict**.items():  
**myList**.append([key, value])  
**print**(myList)  
  
<**strong**>Output: [['A', 'MUO'], ['B', 'Google'], ['C', 'Python']]strong>

And if you want to transform a dictionary into a stretched, or flattened, list:

myDict = {"A" : "MUO", "B" : "Google", "C" : "Python"}  
myList = []  
**for** **key**, **value** **in** **myDict**.items():  
myList+= key, value  
**print**(myList)  
  
<**strong**>Output: ['A', 'MUO', 'B', 'Google', 'C', 'Python']strong>

**Adding Up the Values in a Dictionary**

It's easy to sum all the values in a dictionary using a **for** loop:

myDict = {"A":6, "B":7, "C":9}  
g = 0 *# initilize a variable to store the running total*  
**for** **i** **in** **myDict**.values():  
g += i *# add each value to the total*  
**print**(g)  
  
<**strong**>Output: 22strong>

This is an iterative equivalent to using the **sum()** function which is an iterator itself. So you can add the values using **sum()** instead of looping as you did above:

summedValues = sum(myDict.values())  
**print**(summedValues)  
  
<**strong**>Output: 22strong>

**Looping Through a Nested Python Dictionary**

A nested dictionary might be a bit confusing to loop through at first. But it's as easy as iterating through a regular one.

The code below, for instance, outputs the content of each list in the dictionary:

myDict = {"A" : [1, 2, 3], "B" : [4, 5, 6]}  
  
**for** **i** **in** **myDict**.keys():  
**print**(myDict[i])  
  
<**strong**>Output:  
[1, 2, 3]  
[4, 5, 6]strong>

As it is in a regular dictionary, looping out the entire items outputs all key-value pairs in individual tuples:

myDict = {"A" : [1, 2, 3], "B" : [4, 5, 6]}  
  
**for** **i** **in** **myDict**.items():  
**print**(i)  
  
<**strong**>Output:  
('A', [1, 2, 3])  
('B', [4, 5, 6])strong>

**RELATED:**[**Python Dictionary: How You Can Use It To Write Better Code**](https://www.makeuseof.com/tag/properly-use-python-dictionary/)

You can also see specific values in a dictionary containing other dictionaries. But keep in mind that the values of a complex dictionary are the items of other dictionaries inside it. In essence, a complex dictionary contains parent and child keys.

Let's output the values in the complex dictionary below to see how this works:

complexArray = {  
"Detail" : {  
"Name" : "Idowu",  
"Logs" : 20,  
"isAdmin" : True  
},  
"Activities" : {  
"Inputs" : 14,  
"Input Type" : "Video"  
}  
}  
  
**for** **value** **in** **complexArray**.values():  
**print**(value)  
  
<**strong**>Output:  
{'Name': 'Idowu', 'Logs': 20, 'isAdmin': True}  
{'Inputs': 14, 'Input Type': 'Video'}strong>

Using this insight, you can print specific values from the dictionary above.

To view the values of **Detail**, for instance:

for value in complexArray["Detail"].values():  
**print**(value)  
  
<**strong**>Output:  
Idowu  
20  
**True**

Using a nested **for** loop, you can see all the values of all the child keys:

**for** **value** **in** **complexArray**.values():  
**for** i in value.values(): *# get the values of each key in the child dictionaries*  
**print**(i)  
  
<**strong**>Output:  
Idowu  
20  
**True**  
14  
Videostrong>

Regardless of their parent dictionary, the above iteration outputs all the child values from the nested dictionary.

**Modifying Dictionary Items**

Since a dictionary is mutable, you can modify its content as you like while iterating through it.

For instance, you can swap **values** for **keys** and insert the output in a new dictionary:

myDict = {"A" : "MUO", "B" : "Google", "C" : "Python"}  
swappedDict = {}  
**for** **key**, **value** **in** **myDict**.items():  
swappedDict[value] = key  
**print**(swappedDict)  
  
<**strong**>Output: {'MUO': 'A', 'Google': 'B', 'Python': 'C'}strong>

You can achieve the above using **for** loop in a dictionary comprehension as well:

swappedDict = {value:key for key, value in myDict.items()}  
**print**(swappedDict)  
  
<**strong**>Output: {'MUO': 'A', 'Google': 'B', 'Python': 'C'}strong>

You can also delete specific items from a dictionary while looping through it. This is a pretty handy way to remove duplicates.

The example code below removes duplicated items and inserts one of them back after iterating through the array:

myDict = {"A" : "MUO", "B" : "Google", "C" : "Python", "C" : "Python"}  
**for** key in **list**(myDict.keys()):  
if key == 'C':  
**del** myDict[key]  
myDict[key]="Python"  
**print**(myDict)  
<**strong**>Output: {'A': 'MUO', 'B': 'Google', 'C': 'Python'}strong>

**Play Around With Python Dictionaries**

A Python dictionary is an essential tool for managing data in memory. So a basic understanding of the dictionary data structure, including how to iterate through it and get what you want, helps you in real-life scenarios.

And because you can customize what happens within a Python loop, it lets you manipulate your output. Nevertheless, iterating through a Python dictionary is easy once you understand the basic concepts of the Python loop.

**Coding problems**

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

**Answer:**

n = int(input("Enter a number: "))

factorial = 1

for i in range(n, 0, -1):

    factorial = factorial \* i

print("factorial of ", n, " is: ", factorial)

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

**Answer:**

# User Input Values

p = int(input("Enter the value for p :"))

r = int(input("Enter the value for r :"))

t = int(input("Enter the value for t :"))

sit = (p\*r\*t)/100

print("simple interest = ", sit)

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

**Answer:**

# User Input Values

p = int(input("Enter the value for p :"))

r = int(input("Enter the value for r :"))

t = int(input("Enter the value for t :"))

cit = p \* ((1+ (r/100)) \*\*t)

print("compound interest = ", cit)

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

**Answer:**

n **=** int(input("enter a number "))

flag **=** 0

**for** i **in** range (2, n):

**if** n **%** i **==** 0:

flag **=** 1

**if** flag **==** 0:

print(n, " is a prime number")

**else**:

print(n, " is NOT a prime number")

Q80. Write a Python program to check Armstrong Number.

**Answer:**

Method 1:

n=int(input("Enter any number: "))

a=list(map(int,str(n)))

b=list(map(**lambda** x:x\*\*3,a))

**if**(sum(b)==n):

**print**("The number is an armstrong number. ")

**else**:

**print**("The number isn't an arsmtrong number. ")

Method 2:

*# Get the input number*

n **=** int(input("enter a number "))

val **=** n

*# get the individual digits from the list*

digit\_list **=** []

**while** n **!=** 0:

digit\_list**.**append(n **%** 10)

n **=** n **//** 10

print(digit\_list)

*# Get the sum of i \*\* len(digit\_list)*

sum **=** 0

**for** i **in** digit\_list:

sum **+=** i**\*\***len(digit\_list)

*# checking the sum*

**if** sum **==** val:

print(val, " is a Armstrong number")

**else**:

print(val, " is NOT a Armstrong number")

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

**Answer:**

n = int(input("enter a number "))

Fibonacci\_list = [0, 1]

for i in range(2, n):

    Fibonacci\_list.append(Fibonacci\_list[len(Fibonacci\_list) - 1] + Fibonacci\_list[len(Fibonacci\_list) - 2])

print(Fibonacci\_list)

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

**Answer:**

Given a list, write a Python program to swap first and last element of the list.

My\_list = [1, 2, 3, 6, 7, 8]

print("List before interchange : ", my\_list)

my\_list[0], my\_list[-1] = my\_list[-1], my\_list[0]

print("List after interchange : ", my\_list)

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

My\_list = list(map(int, input("Enter the list: ").split()))

x, y = input("Enter the indexes to be swapped starting with 0: ").split()

x, y = int(x), int(y)

print("List before interchange : ", My\_list)

My\_list[x], My\_list[y] = My\_list[y], My\_list[x]

print("List after interchange : ", My\_list)

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

**Answer:**

My\_list = list(map(int, input("Enter the list: ").split()))

n = int(input("Enter the nth number: "))

sorted\_list = list(set(My\_list))

print(n, "th largest number is: ", sorted\_list[-n])

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

**Answer:**

My\_list = list(map(int, input("Enter the list: ").split()))

cum\_sum = [My\_list[0]]

for i in range(1, len(My\_list)):

    cum\_sum.append(cum\_sum[i-1] + My\_list[i])

print(My\_list)

print("Cummulative Sum for the list", cum\_sum)

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

**Answer:**

Input\_string = input("Enter a string: ")

reverse\_string = Input\_string[-1: : -1]

if Input\_string == reverse\_string:

    print(Input\_string, "is a palindrome")

else:

    print(Input\_string, "is NOT a palindrome")

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

**Answer:**

Input\_string = input("Enter a string: ")

i = int(input("enter the value for i: "))- 1

rem\_str = Input\_string[:i] + Input\_string[i+1 : ]

print(rem\_str)

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

**Answer:**

Input\_string = input("Enter a string: ")

sub\_string = input("Enter a sub-string: ")

if sub\_string in Input\_string:

    print(sub\_string, " is present in ", Input\_string)

else:

    print(sub\_string, " is NOT present in ", Input\_string)

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

**Answer:**

Input\_string = input("Enter a string: ").split()

k = int(input("enter the value for i: "))

counter = 0

for wrd in Input\_string:

    if len(wrd) > k:

        counter += 1

print(counter, " words are greater than length ", k)

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

**Answer:**

dict2 = {

    "Name" : "Veena",

    "age" : 27,

    "Fav\_language" : "Python",

    "Fav\_time" : "coding",

    "Fav\_Quote" : "Justdoit",

    "Life\_is" : "Justdoit",

}

unique\_val = set(dict2.values())

print("Unique dictionary values are: ", unique\_val)

Q91. Write a Python program to merge two dictionary.

**Answer:**

def Merge(dict1, dict2):

    return(dict2.update(dict1))

# Driver code

dict1 = {'a': 10, 'b': 8}

dict2 = {'d': 6, 'c': 4}

# This returns None

print(Merge(dict1, dict2))

# changes made in dict2

print(dict2)

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

**Answer:**

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

Output : {'Sachin': 10, 'MSD': 7, 'Kohli': 18, 'Rohit': 45}

Input\_list = [('Sachin', 10), ('MSD', 7), ('Kohli', 18), ('Rohit', 45)]

output\_val = {}

for i in Input\_list:

    output\_val[i[0]] = i[1]

print(output\_val)

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)]

**Answer**:

Input\_val = [9, 5, 6]

output = list()

for i in Input\_val:

    output.append((i, i\*\*3))

print(output)

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)]

**Answer:**

test\_tuple1 = (7, 2)

test\_tuple2 = (7, 8)

Output = []

for i in range(0, len(test\_tuple1)):

    for j in range(0, len(test\_tuple2)):

        Output.append((test\_tuple1[i], test\_tuple2[j]))

for i in range(0, len(test\_tuple2)):

    for j in range(0, len(test\_tuple1)):

        Output.append((test\_tuple2[i], test\_tuple1[j]))

print(Output)

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)]

**Answer:**

Input\_val = [('for', 24), ('Geeks', 8), ('Geeks', 30)]

length = len(Input\_val)

for i in range(0, length):

    for j in range(0, length-i-1):

        if(Input\_val[j][1] > Input\_val[j+1][1]):

            Input\_val[j], Input\_val[j+1] = Input\_val[j+1], Input\_val[j]

print(Input\_val)

Q96. Write a python program to print below pattern.

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

**Answer:**

x = 5

for i in range (1, x+1):

    for j in range (1, i+1):

        print("\*", end =" ")

    print("")

Q97. Write a python program to print below pattern.

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

**Answer:**

x = 6

for i in range (x, 0, -1):

    for j in range (1, i+1):

        print(" ", end ="")

    for k in range (i+1, x+1):

        print("\*", end ="")

    print("")

Q98. Write a python program to print below pattern.

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

**Answer:**

x = 6

for i in range (x, 0, -1):

    for j in range (1, i+1):

        print(" ", end ="")

    for k in range (i+1, x+1):

        print("\* ", end ="")

    print("")

Q99. Write a python program to print below pattern.

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

**Answer:**

x = 5

for i in range (1, x+1):

    for j in range (1, i+1):

        print(j, end =" ")

    print("")

Q100. Write a python program to print below pattern.

A

B B

C C C

D D D D

E E E E E

**Answer:**

x = 5

letters = " ABCDEFGHIJKLMNOPQRSTUVWXYZ"

for i in range (0, x+1):

    for j in range (0, i):

        print(letters[i], end =" ")

    print("")