**1.Understanding how to create and access elements in a list .**

In [Python](https://www.geeksforgeeks.org/python-programming-language-tutorial/), a list is a built-in dynamic sized array (automatically grows and shrinks). We can store all types of items (including another list) in a list. A list may contain mixed type of items, this is possible because a list mainly stores references at contiguous locations and actual items maybe stored at different locations.

* List can contain duplicate items.
* List in Python are Mutable. Hence, we can modify, replace or delete the items.
* List are ordered. It maintain the order of elements based on how they are added.
* Accessing items in List can be done directly using their position (index), starting from 0.

*# Creating a Python list with different data types*

* + - a = [10, 20, "GfG", 40, True]
    - print(a)
    - *# Accessing elements using indexing*
    - print(a[0]) *# 10*
    - print(a[1]) *# 20*
    - print(a[2]) *# "GfG"*
    - print(a[3]) *# 40*
    - print(a[4]) *# True*
    - *# Checking types of elements*
    - print(type(a[2])) *# str*
    - print(type(a[4])) *# bool*

**2.Indexing in lists(positive and negative indexing).**

Any linear data structure in any programming language is built around indexing. The default index for each machine starts at 0 and continues up to n-1. In this case, n represents the overall number of items in the corresponding data structure. Types include

* Positive indexing − Increases from 0 to 1.
* Negative indexing − each traversal moves from tail to head, starting with the last element.

Similar to how we access elements in lists and strings, we can access elements in tuples. So, indexing and slicing are the only methods we need to access items. Additionally, indexing is straightforward, beginning at index zero, just like in lists. In addition, the figure we put within the square bracket represents the tuple's index. Let's see a few instances of tuple indexing being used to retrieve a tuple's elements.

Example 1.

tup1 = (10, 3, 4, 22, 1)

# for accessing the first element of the tuple

print(tup1[0])

# accessing the third element of the tuple

print(tup1[2])

print(tup1[1+3])

# the expression inside the square brackets results in an integer index 4. Hence, we get the element at the 4th index.

Example 2

myList = [1, 2, 3, 4, 5]

# NORMAL INDEXING

print(myList[0])

print(myList[1])

print(myList[2])

# NEGATIVE INDEXING (STARTS FROM THE LAST ELEMENT IN THE LIST) print(myList[-1])

print(myList[-2])

print(myList[-3])

print(myList[-3:])

**3. slicing a list: access a range of elements.**

Python **list slicing** is fundamental concept that let us easily access specific elements in a list. In this article, we’ll learn the syntax and how to use both positive and negative indexing for slicing with examples.

a = [1, 2, 3, 4, 5, 6, 7, 8, 9]

*# Get all elements in the list*

print(a[::])

print(a[:])

[1, 2, 3, 4, 5, 6, 7, 8, 9]

[1, 2, 3, 4, 5, 6, 7, 8, 9]

Ex:2

a = [1, 2, 3, 4, 5, 6, 7, 8, 9]

*# Get elements starting from index 2*

*# to the end of the list*

b = a[2:]

print(b)

*# Get elements starting from index 0*

*# to index 3 (excluding 3th index)*

c = a[:3]

print(c)

O/p:

[3, 4, 5, 6, 7, 8, 9]

[1, 2, 3]

**4.common list operations: concatenation, repetition, membership.**

List Operations in Python

It is the computation or actions applied to the variable containing the list of data types in an expression.

[List manipulation](https://cloudpunjabi.com/tag/list-manipulation/) in[Python](https://cloudpunjabi.com/what-is-python/)can be done using various operators like concatenation (+), repetition (\*), slicing of the list, and membership operators( in /not in). So, Let’s understand each operator in brief.

[1. Concatenation operator (+)](https://cloudpunjabi.com/list-operations-in-python/#1Concatenation_operator)

[2. Repetition operator (\*)](https://cloudpunjabi.com/list-operations-in-python/#2_Repetition_operator)

[3. List Slicing in Python](https://cloudpunjabi.com/list-operations-in-python/#3_List_Slicing_in_Python)

[4. Comparison Operator](https://cloudpunjabi.com/list-operations-in-python/#4_Comparison_Operator)

[5. Membership Operator (in, not in)](https://cloudpunjabi.com/list-operations-in-python/#5Membership_Operator_in_not_in)

# 1. Concatenation operator (+)

The (+) operator is used to add to two lists.

The syntax of the given operation is: List1+List2

**>>>**lst1=[12, 34, 56]  
**>>>**lst2=[78, 90]  
**>>>**print(lst1+lst2)**#Output**  
[12, 34, 56, 78, 90]

# 2. Repetition operator (\*)

Like [string](https://cloudpunjabi.com/string-operations-in-python/), (\*) operator replicates the string number of specified times.

The syntax of the given operation: List\*n

**>>>**lst1=[12, 34, 56]  
**>>>**print( lst1\*3)**#Output**  
[12, 34, 56, 12, 34, 56, 12, 34, 56]

# 3. List Slicing in Python

List slicing returns a slice or part of the list from the given index range x to y. (x is included but y is not included).

The syntax of the list slicing is: List[ start: stop]

**>>>**lst1= [12, 34, 56, 78, 90]  
**>>>**x= lst1[1: 4]  
**>>>**print(x)**#Output**  
[34, 56, 78]

# 4. Comparison Operator

[Python](https://cloudpunjabi.com/what-is-python/) offers standard[comparison operators](https://cloudpunjabi.com/comparison-operators-in-python/) like <, >, ==, != to compare two lists.

For comparison, two lists must-have elements of comparable [data types](https://cloudpunjabi.com/data-types-in-python/), otherwise, you will get an error.

Python gives the result of [comparison operators](https://cloudpunjabi.com/comparison-operators-in-python/) as True or False and moreover, it compares list element by element it compares the first element if they are the same then will move to the next, and so on.

**Comparison**: [12, 3, 4 , 0] > [9, 12, 34]

**Result:**True

**Reason:** compared the first element of both the list, which is 12 and 9, as 12> 9 hence return True

# 5. Membership Operator (in, not in)

The [membership operator](https://cloudpunjabi.com/membership-operators-in-python/) checks whether an element exists in the given list.

* **in**: Return True if an element exists in the given list; False otherwise
* **not in**: Return True if an element does not exist in the given list; False otherwise.

**>>>**lst1=[12, 34, 56, 78, 90]  
**>>>**56 in lst1  
**>>>**12 not in lst1**#Output**  
True  
False

**5.Understanding list methods like append(),insert(),remove(),pop()**

[Python list](https://www.geeksforgeeks.org/python-lists/)methods are built-in functions that allow us to perform various operations on **lists**, such as a**dding, removing,**or **modifying**elements. In this article, we’ll explore all **Python list methods**with a simple example.

## List Methods

Let’s look at different list methods in Python:

* [append()](https://www.geeksforgeeks.org/python-list-append-method/): Adds an element to the end of the list.
* [copy()](https://www.geeksforgeeks.org/copy-python-deep-copy-shallow-copy/): Returns a shallow copy of the list.
* [clear()](https://www.geeksforgeeks.org/python-list-clear-method/): Removes all elements from the list.
* [count()](https://www.geeksforgeeks.org/python-list-count-method/): Returns the number of times a specified element appears in the list.
* [extend()](https://www.geeksforgeeks.org/python-list-extend-method/): Adds elements from another list to the end of the current list.
* [index()](https://www.geeksforgeeks.org/python-list-index/): Returns the index of the first occurrence of a specified element.
* [insert()](https://www.geeksforgeeks.org/python-list-insert/): Inserts an element at a specified position.
* [pop()](https://www.geeksforgeeks.org/python-list-pop/): Removes and returns the element at the specified position (or the last element if no index is specified).
* [remove()](https://www.geeksforgeeks.org/python-list-remove/): Removes the first occurrence of a specified element.
* [reverse()](https://www.geeksforgeeks.org/python-list-reverse/): Reverses the order of the elements in the list.
* [sort()](https://www.geeksforgeeks.org/python-list-sort-method/): Sorts the list in ascending order (by default).

Example:

a = [1, 2, 3]

*# Add 4 to the end of the list*

a.append(4)

print(a)

*# Insert 5 at index 4*

a.insert(4,5)

print(a)

*# Remove and return the last element in the list*

a.pop()

print(a)

*# Remove the first occurrence of 2*

a.remove(2)

print(a)

**6.Iterationg over a list using loops.**

Python For Loops are used for iterating over a sequence like lists, tuples, strings, and ranges.

* For loop allows you to apply the same operation to every item within loop.
* Using For Loop avoid the need of manually managing the index.
* For loop can iterate over any iterable object, such as dictionary, list or any custom iterators.

a = ["shirt", "sock", "pants", "sock", "towel"]

b = []

for i in a:

if i == "sock":

continue

else:

print(f"Washing {i}")

b.append("socks")

print(f"Washing {b}")

Output

Washing shirt

Washing pants

Washing towel

Washing ['socks']

**7.Sorting and reversing a list using sort(),sorted() and reverse()**

The**sort()** method in Python is a built-in function that allows us to sort the elements of a list in **ascending**or **descending**order and**it modifies the list in place**which means there is no new list created. This method is useful when working with lists where we need to arranged the elements in a specific order, whether numerically or alphabetically.

a = [5, 2, 9, 1, 5, 6]

*# Sort the value in increasing order*

a.sort()

print(a)

[1, 2, 5, 5, 6, 9]

a = [5, 2, 9, 1, 5, 6]

*# Sorting in Descending Order*

a.sort(reverse=True)

print(a)

[9, 6, 5, 5, 2, 1]

[sorted()](https://www.geeksforgeeks.org/python-sorted-function/) method sorts the given sequence as well as set and dictionary(which is not a sequence) either in ascending order or in descending order(does Unicode comparison for string char by char) and always returns a sorted list. This method doesn’t affect the original sequence.

***Syntax:*** *sorted(iterable, key, reverse=False)*

|  |
| --- |
| L = [1, 5, 4, 2, 3]    #Print the sorted list  print("Sorted list:")  print(sorted(L))    #Print the original list  print("\nOriginal list after sorting:")  print(L) |

The reverse() method reverses the sorting order of the elements.

fruits = ['apple', 'banana', 'cherry']

fruits.reverse()

print(fruits)

**8.Basic list manipulations: addition, deletion, updating and slicing**

We can add elements to a list using the following methods:

* [append():](https://www.geeksforgeeks.org/python-list-append-method/) Adds an element at the end of the list.
* [extend():](https://www.geeksforgeeks.org/python-list-extend-method/) Adds multiple elements to the end of the list.
* [insert()](https://www.geeksforgeeks.org/python-list-insert/): Adds an element at a specific position.
* *# Initialize an empty list*
* a = []
* *# Adding 10 to end of list*
* a.append(10)
* print("After append(10):", a)
* *# Inserting 5 at index 0*
* a.insert(0, 5)
* print("After insert(0, 5):", a)
* *# Adding multiple elements [15, 20, 25] at the end*
* a.extend([15, 20, 25])
* print("After extend([15, 20, 25]):", a)
* After append(10): [10]
* After insert(0, 5): [5, 10]
* After extend([15, 20, 25]): [5, 10, 15, 20, 25]

## Removing Elements from List

We can remove elements from a list using:

* [**remove()**](https://www.geeksforgeeks.org/python-list-remove/)**:**Removes the first occurrence of an element.
* [**pop()**](https://www.geeksforgeeks.org/python-list-pop-method/)**:**Removes the element at a specific index or the last element if no index is specified.
* [**del statement**](https://www.geeksforgeeks.org/python-del-to-delete-objects/)**:** Deletes an element at a specified index.
* a = [10, 20, 30, 40, 50]
* *# Removes the first occurrence of 30*
* a.remove(30)
* print("After remove(30):", a)
* *# Removes the element at index 1 (20)*
* popped\_val = a.pop(1)
* print("Popped element:", popped\_val)
* print("After pop(1):", a)
* *# Deletes the first element (10)*
* del a[0]
* print("After del a[0]:", a)
* Output
* After remove(30): [10, 20, 40, 50]
* Popped element: 20
* After pop(1): [10, 40, 50]
* After del a[0]: [40, 50]

## Updating Elements into List

We can change the value of an element by accessing it using its index.

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

*# Change the second element*

a[1] = 25

print(a)

**Output**

[10, 25, 30, 40, 50]

Python **list slicing** is fundamental concept that let us easily access specific elements in a list. In this article, we’ll learn the syntax and how to use both positive and negative indexing for slicing with examples.

a = [1, 2, 3, 4, 5, 6, 7, 8, 9]

*# Get elements from index 1*

*# to index 4 (excluding index 4)*

b = a[1:4]

print(b)

[2, 3, 4]

**9.Introduction to tuples,immutability**

Python Tuple is a collection of objects separated by commas. A tuple is similar to a Python list in terms of indexing, nested objects, and repetition but the main difference between both is Python tuple is immutable, unlike the Python list which is mutable.

## What is Immutable in Tuples?

Unlike [Python lists](https://www.geeksforgeeks.org/python-lists/), tuples are immutable. Some Characteristics of Tuples in Python.

* Like Lists, tuples are ordered and we can access their elements using their index values
* We cannot update items to a tuple once it is created.
* Tuples cannot be appended or extended.
* We cannot remove items from a tuple once it is created.
* t = (1, 2, 3, 4, 5)
* *# tuples are indexed*
* print(t[1])
* print(t[4])
* *# tuples contain duplicate elements*
* t = (1, 2, 3, 4, 2, 3)
* print(t)
* *# updating an element*
* t[1] = 100
* print(t)
* **Output:**
* 2  
  5  
  (1, 2, 3, 4, 2, 3)  
  Traceback (most recent call last):  
   File "Solution.py", line 12, in <module>  
   t[1] = 100  
  TypeError: 'tuple' object does not support item assignment

**10.Creating and accessing elements in tuple**

You can access tuple items by referring to the index number, inside square brackets:

thistuple = ("apple", "banana", "cherry")  
print(thistuple[1])

**11.Basic operatios with tuples: concatenation,repetition,membership**

### 1. Tuple Concatenation (+)

This operation in Python is used to add two tuples together using the (+) operator. Let us take a simple example to understand this operation in Python tuple function.

*Tuple1 + Tuple2*

|  |
| --- |
| t1 = (1, 2)  t2 = (3, 4)  print(t1 + t2) |

(1,2,3,4)

### 2. Tuple Repetition (\*)

This operation is used in Python tuple function to repeat the tuple elements a specific number of times. We use a simple (\*) operator to perform this operation.

|  |
| --- |
| t = (1, 2)  print(t \* 3) |

(1,2,1,2,1,2)

### 3. Membership Check

It is used to check whether a following element is available in a Python tuple or not. It can be easily implemented using “in” or “not in” keywords.

|  |
| --- |
| t = (‘apple’, ‘banana’, ‘cherry’)  print(‘banana’ in t)  print(‘grape’ not in t) |

True

True

**12.Accessing tuple elements using positive and negative indexing**

### Python Access Tuple using a Positive Index

Using square brackets we can get the values from tuples in Python.

t = (10, 5, 20)

print("Value in t[0] = ", t[0])

print("Value in t[1] = ", t[1])

print("Value in t[2] = ", t[2])

Value in t[0] = 10

Value in t[1] = 5

Value in t[2] = 20

### Access Tuple using Negative Index

In the above methods, we use the positive index to access the value in Python, and here we will use the negative index within [].

t = (10, 5, 20)

print("Value in t[-1] = ", t[-1])

print("Value in t[-2] = ", t[-2])

print("Value in t[-3] = ", t[-3])

**Output**

Value in t[-1] = 20

Value in t[-2] = 5

Value in t[-3] = 10

**13.Slicing a tuples to access ranges of elements**

We can use slicing in tuples I’m the same way as we use in strings and lists. Tuple slicing is basically used to obtain a range of items. Furthermore, we perform tuple slicing using the slicing operator. We can represent the slicing operator in the syntax [start:stop:step]. Moreover, it is not necessary to mention the ‘step’ part. The compiler considers it 1 by default if we do not mention the step part.

>>> tup = (22, 3, 45, 4, 2.4, 2, 56, 890, 1)

>>> print(tup[1:4])

# prints 2nd to 4th element

(3, 45, 4)

**14.Introduction to dictionaries: key value pairs**

**A Python dictionary**is a data structure that stores the value in **key: value** pairs. Values in a dictionary can be of any data type and can be duplicated, whereas keys can’t be repeated and must be immutable.

*# create dictionary using { }*

d1 = {1: 'Geeks', 2: 'For', 3: 'Geeks'}

print(d1)

*# create dictionary using dict() constructor*

d2 = dict(a = "Geeks", b = "for", c = "Geeks")

print(d2)

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

{'a': 'Geeks', 'b': 'for', 'c': 'Geeks'}

## Using update()

[update()](https://www.geeksforgeeks.org/python-dictionary-update-method/) is used when we need to add multiple key-value pairs or modify existing ones. We can pass another dictionary or an iterable of key-value pairs to this method and it will add the new pairs or update the existing keys.

d = {'key1': 'geeks', 'key2': 'for'}

d.update({'key3': 'Geeks', 'key4': 'is', 'key5': 'portal', 'key6': 'Computer'})

print(d)

{'key1': 'geeks', 'key2': 'for', 'key3': 'Geeks', 'key4': 'is', 'key5': 'portal', 'key6': 'Computer'}

## Using square brackets []

This is the simplest way to add or update a key-value pair in a dictionary. We access the dictionary by specifying the key inside square brackets and assign the corresponding value. If the key doesn’t exist, it will be added to the dictionary.

d = {'key1': 'geeks', 'key2': 'for'}

d['key3'] = 'Geeks'

d['key4'] = 'is'

d['key5'] = 'portal'

d['key6'] = 'Computer'

print(d)

**Output**

{'key1': 'geeks', 'key2': 'for', 'key3': 'Geeks', 'key4': 'is', 'key5': 'portal', 'key6': 'Computer'}

## Using dictionary unpacking

[dictionary unpacking](https://www.geeksforgeeks.org/python-unpack-dictionary/) allows us to merge dictionaries or add new key-value pairs in a concise way. By using the unpacking operator \*\*, we can merge an existing dictionary with new items in a single statement.

d = {'key1': 'geeks', 'key2': 'for'}

d = {\*\*d, 'key3': 'Geeks', 'key4': 'is', 'key5': 'portal', 'key6': 'Computer'}

print(d)

**Output**

{'key1': 'geeks', 'key2': 'for', 'key3': 'Geeks', 'key4': 'is', 'key5': 'portal', 'key6': 'Computer'}

## Using dict()

[dict()](https://www.geeksforgeeks.org/python-dict-function/)allows us to create a new dictionary by combining existing ones with new key-value pairs. This method is useful when we want to create a new dictionary that includes both the original and new items.

d = {'key1': 'geeks', 'key2': 'for'}

d = dict(d, key3='Geeks', key4='is', key5='portal', key6='Computer')

print(d)

**Output**

{'key1': 'geeks', 'key2': 'for', 'key3': 'Geeks', 'key4': 'is', 'key5': 'portal', 'key6': 'Computer'}

**15.Accessing ,adding and updating dictionary elements**

## Accessing Dictionary Items

We can access a value from a dictionary by using the **key**within square brackets or[**get()**](https://www.geeksforgeeks.org/python-dictionary-get-method/)method.

d = { "name": "Alice", 1: "Python", (1, 2): [1,2,4] }

*# Access using key*

print(d["name"])

*# Access using get()*

print(d.get("name"))

## Adding and Updating Dictionary Items

We can add new key-value pairs or update existing keys by using assignment.

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

*# Adding a new key-value pair*

d["age"] = 22

*# Updating an existing value*

d[1] = "Python dict"

print(d)

## Removing Dictionary Items

We can remove items from dictionary using the following methods:

* del: Removes an item by key.
* [pop()](https://www.geeksforgeeks.org/python-dictionary-pop-method/): Removes an item by key and returns its value.
* [clear()](https://www.geeksforgeeks.org/python-dictionary-clear): Empties the dictionary.
* [popitem()](https://www.geeksforgeeks.org/python-dictionary-popitem-method/): Removes and returns the last key-value pair.

d = {1: 'Geeks', 2: 'For', 3: 'Geeks', 'age':22}

*# Using del to remove an item*

del d["age"]

print(d)

*# Using pop() to remove an item and return the value*

val = d.pop(1)

print(val)

*# Using popitem to removes and returns*

*# the last key-value pair.*

key, val = d.popitem()

print(f"Key: {key}, Value: {val}")

*# Clear all items from the dictionary*

d.clear()

print(d)

**16.Dictionary method like keys(),values() and items()**

### Dictionary keys() Method

The keys() method in Python returns a view object with dictionary keys, allowing efficient access and iteration.

d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}

print(list(d.keys()))

**Output**

['Name', 'Age', 'Country']

### Dictionary values() Method

The values() method in Python returns a view object containing all dictionary values, which can be accessed and iterated through efficiently.

d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}

print(list(d.values()))

**Output**

['Ram', '19', 'India']

### Dictionary items() Method

In Python, the items() method is a built-in dictionary function that retrieves a view object containing a list of tuples. Each tuple represents a key-value pair from the dictionary. This method is a convenient way to access both the keys and values of a dictionary simultaneously, and it is highly efficient.

d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}

print(list(d.items())[1][0])

print(list(d.items())[1][1])

**Output**

Age

19

**17.Iterating over a dictionary using loops**

To iterate through all values of a dictionary in Python using [.values()](https://www.geeksforgeeks.org/python-dictionary-values/), you can employ a for loop, accessing each value sequentially. This method allows you to process or display each individual value in the dictionary without explicitly referencing the corresponding keys.

**Example:** In this example, we are using the values() method to print all the values present in the dictionary.

*# create a python dictionary*

d = {"name": "Geeks", "topic": "dict", "task": "iterate"}

*# loop over dict values*

for val in d.values():

print(val)

### Iterate through keys

In Python, just looping through the dictionary provides you its keys. You can also iterate keys of a dictionary using built-in `[.keys()](https://www.geeksforgeeks.org/python-dictionary-keys-method/)` method.

*# create a python dictionary*

d = {"name": "Geeks", "topic": "dict", "task": "iterate"}

*# default loooping gives keys*

for keys in d:

print(keys)

*# looping through keys*

for keys in d.keys():

print(keys)

### Iterate through both keys and values

You can use the built-in items() method to access both keys and items at the same time. items() method returns the view object that contains the key-value pair as tuples.

*# create a python dictionary*

d = {"name": "Geeks", "topic": "dict", "task": "iterate"}

*# iterating both key and values*

for key, value in d.items():

print(f"{key}: {value}")

**18.Merging two list into a dictionary using loops or zip()**

We are given two lists, we need to convert both of the list into dictionary. **For example** we are given two lists a = [“name”, “age”, “city”], b = [“Geeks”, 30,”Delhi”], we need to convert these two list into a form of dictionary so that the output should be like {‘name’: ‘Geeks’, ‘age’: 30, ‘city’: ‘Delhi’}. We can do this using methods like zip, dictionary comprehension , itertools.starmap.

## Using zip

Use [zip](https://www.geeksforgeeks.org/zip-in-python/) to pair elements from two lists, where the first list provides the keys and second provides the values after that we convert the zipped object into a dictionary using [dict()](https://www.geeksforgeeks.org/python-dict-function/" \t "_blank) which creates key-value pairs.

a = ["name", "age", "city"]

b = ["Alice", 30, "New York"]

*# Create dictionary using zip*

res = dict(zip(a, b))

print(res)

{'name': 'Alice', 'age': 30, 'city': 'New York'}

## Using a Loop

Iterate through both lists simultaneously using zip and for each pair, add the first element as the key and second as the value to the dictionary.

a = ["name", "age", "city"]

b = ["Alice", 30, "New York"]

res = {}

*# Iterate through the lists using zip*

for key, value in zip(a, b):

res[key] = value

print(res)

**Output**

{'name': 'Alice', 'age': 30, 'city': 'New York'}

**19.Counting occurrences of characters in a string using dictionaries**

|  |
| --- |
| # Python3 code to program to find occurrence  # to each character in given string    # initializing string  inp\_str = "GeeksforGeeks"    # using set() + count() to get count  # of each element in string  out = {x : inp\_str.count(x) for x in set(inp\_str )}    # printing result  print ("Occurrence of all characters in GeeksforGeeks is :\n "+ str(out)) |

Occurrence of all characters in GeeksforGeeks is :

{'o': 1, 's': 2, 'e': 4, 'G': 2, 'r': 1, 'f': 1, 'k': 2}

**20.Define Functions in python**

Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Some Benefits of Using Functions

* Increase Code Readability
* Increase Code Reusability
* *# A simple Python function*
* def fun():
* print("Welcome to GFG")
* *# Driver code to call a function*
* fun()

def add(num1: int, num2: int) -> int:

*"""Add two numbers"""*

num3 = num1 + num2

return num3

*# Driver code*

num1, num2 = 5, 15

ans = add(num1, num2)

print(f"The addition of {num1} and {num2} results {ans}.")

The addition of 5 and 15 results 20.

**21.Different type of function :with/without parameters ,with/without return values**

Function Without Arguments

A function without arguments, on the other hand, doesn't take any values as input when it is called. Instead, it performs some computation based on its internal logic and returns a result.

Here's an example of a function without arguments in Python:

def greet():

"""

Prints a greeting message.

"""

print("Hello, how are you?")

Function With Arguments A function with arguments takes one or more values as input when it is called. These values are specified inside the parentheses when you call the function, and they are passed to the function as parameters. The function can then use these parameters to perform some computation and return a result.

Here's an example of a function with arguments in Python:

def area\_of\_rectangle(x,y):

"""

Returns the area of rectangle by length \*breadth (x\*y).

""" The Return Statement

A Python function can either return a value or not. The return statement is used when we want our function to return a value to a function call.

Here's an example of a Python function with a return statement:

def area\_of\_rectangle(x, y):

"""

Returns the area of the rectangle.

"""

return x \* y

def greet(name):

"""

Prints a greeting message.

"""

print(f"Hello, {name}! How are you doing today?")

This function takes one argument, name, and prints a greeting message that includes the name. It doesn't use a return statement to provide a result back to the code that called it. Instead, it performs some computation based on its internal logic and outputs a message to the console. You can call this function like this:

greet("Alice") # Output: "Hello, Alice! How are you doing today?"

**22.Anonymous function (lamda function)**

**Python Lambda Functions** are anonymous functions means that the function is without a name. As we already know the def keyword is used to define a normal function in Python.

s1 = 'GeeksforGeeks'

s2 = lambda func: func.upper()

print(s2(s1))

## lambda with Condition Checking

A lambda function can include conditions using if statements.

**Example:**

*# Example: Check if a number is positive, negative, or zero*

n = lambda x: "Positive" if x > 0 else "Negative" if x < 0 else "Zero"

print(n(5))

print(n(-3))

print(n(0))

**Output**

Positive

Negative

Zero

**23.Introduction to python modules and importing modules**

**Python Module** is a file that contains built-in functions, classes,**its** and variables. There are many **Python modules**, each with its specific work.

## What is Python Module

A [Python](https://www.geeksforgeeks.org/python-programming-language/)module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code.

Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.

## **Create a Python Module**

To create a Python module, write the desired code and save that in a file with**.py**extension. Let’s understand it better with an example:

*# A simple module, calc.py*

def add(x, y):

return (x+y)

def subtract(x, y):

return (x-y)

**24.Standard library modules: math, random**

Python **Random module** generates random numbers in [Python](https://www.geeksforgeeks.org/python-programming-language/). These are pseudo-random numbers means they are not truly random.

import random

list1 = [1, 2, 3, 4, 5, 6]

print(random.choice(list1))

Output

3

Math Module consists of mathematical functions and constants. It is a built-in module made for mathematical tasks.

The math module provides the math functions to deal with basic operations such as addition(+), subtraction(-), multiplication(\*), division(/), and advanced operations like trigonometric, logarithmic, and exponential functions.

|  |
| --- |
| import math  r = 4  pie = math.pi  print(pie \* r \* r) |

Output:

50.26548245743669

**25.Creating custom modules**