**for/ while loop:**

**1.What is difference between for loop and while loop in python?**

* Initialization, conditional checking, and increment or decrement is done while iteration in “for” loop executed. while on the other hand, only initialization and condition checking in the syntax can be done.
* For loop is used when we are aware of the number iterations at the time of execution. on the other hand, in “while” loop, we can execute it even if we are not aware of the number of iterations.
* If you forgot to put the conditional statement in for loop, it will reiterate the loop infinite times but if you forgot to put the conditional statement in while loop, it will show an error to you.
* The syntax in the for loop will be executed only when the initialization statement is on the top of the syntax but in case of while loop, it doesn’t matter whether initialization statement finds anywhere in the syntax.
* The iteration will be executed on if the body of the loop executes. on the contrary, the iteration statement in while loop can be written anywhere in the syntax.

#### **Conclusion**

The for loop and while loop both are iteration statement, but both have their distinct feature. The for loop do have all its declaration (initialization, condition, iteration) at the top of the body of the loop. Adversely, in while loop only initialization and condition are at the top of the body of loop and iteration may be written anywhere in the body of the loop.

**2. How to exit from the loop?**

### **Break**

### In Python, the keyword break causes the program to exit a loop early. break causes the program to jump out of for loops even if the for loop hasn't run the specified number of times.break causes the program to jump out of while loops even if the logical condition that defines the loop is still True.

### **Continue**

### In Python, the keyword continue causes the program to stop running code in a loop and start back at the top of the loop. Remember the keyword break cause the program to exit a loop. continue is similar, but continue causes the program to stop the current iteration of the loop and start the next iteration at the top of the loop.

**3. How to skip some iteration from the loop?**

### **Continue**

### In Python, the keyword continue causes the program to stop running code in a loop and start back at the top of the loop. Remember the keyword break cause the program to exit a loop. continue is similar, but continue causes the program to stop the current iteration of the loop and start the next iteration at the top of the loop.

**4. When else part will be executed in loop?**

* Loop statements may have an else clause. It is executed when the for loop terminates through exhaustion of the iterable — but not when the loop is terminated by a break statement.
* The else clause is also executed in a for loop after the continue statement.

**5. How to iterate backward? E.g to print upto 6.**

**Like 6,5,4,3,2,1.**

for i in range(6, 0, -1):  
 print(i, end=",")

**How we can iterate backward on list?**

* Using for loop

l = [2, 3, 4, 5, 6]  
for i in range(len(l)-1, -1, -1):  
 print(l[i])

* Using reversed() function

for i in reversed(l):  
 print(i)

**Dict:**

**1. How to create dictionary?**

**Dictionary**in Python is an ordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key value is provided in the dictionary to make it more optimized.

Creating a Dictionary

* In Python, a Dictionary can be created by placing sequence of elements within curly **{}** braces, separated by ‘comma’. Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can’t be repeated and must be immutable.
* Dictionary can also be created by the built-in function dict().
* An empty dictionary can be created by just placing to curly braces{}.

**2. How to access data from dictionary? Key, value and both.**

* You can access the items of a dictionary by referring to its key name, inside square brackets
* There is also a method called get() that will give you the same result
* The keys() method will return a list of all the keys in the dictionary.
* The values() method will return a list of all the values in the dictionary.
* The items() method will return each item in a dictionary, as tuples in a list.
* dict = {1: "a", 2: "b"}  
  # Print key names  
  for i in dict:  
   print(i)  
    
  for i in dict.keys():  
   print(i)  
    
  x = dict.keys()  
  print(x)  
    
  # Print values  
  for i in dict:  
   print(dict[i])  
    
  for i in dict.values():  
   print(i)  
    
  y = dict.values()  
  print(y)  
    
  # Print keys and values  
  for x, y in dict.items():  
   print(x, y)  
    
  z = dict.items()  
  print(z)

**3. How to delete element from dictionary?**

* The pop() method removes the item with the specified key name. # dict\_name.pop(“key\_name”)
* The popitem() method removes the last inserted item. # dict\_name.popitem()
* The del keyword removes the item with the specified key name. # del dict\_name[“key\_name”]

**4. What is purpose of get(), update(), setdefault() methods of dictionary?**

* get(): Used to access the items of a dictionary by referring to its key name, inside round brackets. # x = dict.get(“key\_name”)
* update(): Used to insert an item in the dictionary. # dict\_name.update({“key\_name: “value”})
* setdefault(): Returns the value of specified key. If the key does not exist, then it inserts the key with specified value. # dict\_name.setdefault(“key\_name”’, “value”)

**5. Write a program to find number of occurences of each character of each letter present in the**

**given string?**

a = "virendra mali"  
d = {}-  
for i in a:  
 if i in d:  
 d[i] += 1  
 else:  
 d[i] = 1  
print(d)

**File handling:**

**1. Write a program to check whether the given file exist or not. If it is available then print its content?**

def file\_status(file):  
 import os.path  
 if os.path.isfile(file) == True: #os.path.exists(file)  
 print(f"{file} file present")  
 print(f"Content of {file} file is:")  
 a = open(file, "r")  
 print(a.read())  
 else:  
 print(f"{file} file not present")

**2. Write a function in Python to count and display the total number of words in a text file.**

def count\_words(file):  
 a = open(file, "r")  
 b = a.read()  
 c = b.split()

a.close()  
 print(f"Total number of words in {file} file are:", len(c))

**3. Write a function in Python to read lines from a text file. Your function should find and display the occurrence of the word "the"**

def words(file):  
 a = open(file, "r")  
 b = a.read()  
 c = b.split()  
 a.close()  
 count = 0  
 for i in c:  
 if i == "The":  
 count += 1  
 elif i == "the":  
 count += 1  
 else:  
 pass  
 print("Occurrence of word 'the' are:", count)

**4. Write a function in Python to count uppercase character in a text file.**

def uppercase(file):  
 a = open(file, "r")  
 b = a.read()  
 c = b.split()  
 count = 0  
 for i in c:  
 if i == i.upper():  
 count += 1  
 else:  
 pass  
 print("Total uppercase letters in file are:", count)

**5. Write a function in Python to count words in a text file those are ending with alphabet "e".**

def words(file):  
 p = open(file, "r")  
 q = p.read()

p.close()  
 r = q.split()

p.close()  
 count = 0  
 for i in r:  
 if i[-1] == "e":  
 count += 1  
 print("Total number of words ending with 'e' are:", count)

**General:**

**1. What are types of arguments?**

## **1.default arguments:**

* default arguments are values that are provided while defining functions.
* The assignment operator = is used to assign a default value to the argument.
* default a++rguments become optional during the function calls.
* If we provide value to the default arguments during function calls, it overrides the default value.
* The function can have any number of default arguments
* Default arguments should follow non-default arguments.

## 2. Keyword Arguments:

* Functions can also be called using keyword arguments of the form kwarg=value.
* During function call, values passed through arguments need not be in the order of parameters in the function definition. This can be achieved by keyword arguments. But all the keyword arguments should match the parameters in the function definition

## **3. Positional Arguments**

* During function call, values passed through arguments should be in the order of parameters in the function definition. This is called **positional arguments.**
* Keyword arguments should follow positional arguments only.

4. Arbitrary Arguments

If we don’t know the number of arguments needed for the function in advance, we can use arbitary arguments

## **arbitrary positional arguments:**

## For arbitrary positional argument, an **asterisk (\*)** is placed before a parameter in function definition which can hold non-keyword variable-length arguments. These arguments will be wrapped up in a**tuple**. Before the variable number of arguments, zero or more normal arguments may occur.

## **arbitrary keyword arguments:**

## For arbitrary keyword argument, a **double asterisk (\*\*)** is placed before a parameter in function which can hold keyword variable-length arguments.

**2. What is decorator? Write a decorator to display function name and print start and end.**

* [**Decorators**](https://www.geeksforgeeks.org/function-decorators-in-python-set-1-introduction/) are very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class. Decorators allow us to wrap another function in order to extend the behavior of the wrapped function, without permanently modifying it.
* A decorator takes in a function, adds some functionality and returns it.

**3. What is generator? Explain advantages. Write a generator for Fibonacci series.**

* A generator is a special type of function which does not return a single value, instead, it returns an iterator object with a sequence of values. In a generator function, a yield statement is used rather than a return statement.

Advantages of generators:

* Without Generators in Python, producing iterables is extremely difficult and lengthy.
* Generators easy to implement as they automatically implement \_\_iter\_\_(), \_\_next\_\_(), and StopIteration which otherwise, need to be explicitly specified.
* Memory is saved as the items are produced when required, unlike normal Python functions. This fact becomes very important when you need to create a huge number of iterators. This is also considered as the biggest advantage of generators.
* Can be used to produce an infinite number of items.
* They can also be used to pipeline a number of operations

Generator for Fibonacci series:

def fibon(l):

    # Initialize first two Fibonacci Numbers

    a, b = 0, 1

    # One by one yield next Fibonacci Number

    while a < l:

        yield a

        a, b = b, a + b

# Create a generator object

x = fibon(5)

# Iterating over the generator object using next

print(x.\_\_next\_\_())

print(x.\_\_next\_\_())

print(x.\_\_next\_\_())

print(x.\_\_next\_\_())

print(x.\_\_next\_\_())

**4. What is difference between shallow copy and deepcopy?**

|  |  |
| --- | --- |
| **Shallow copy** | **Deep copy** |
| Shallow Copy stores the references of objects to the original memory address. | Deep copy stores copies of the object’s value. |
| Shallow Copy reflects changes made to the new/copied object in the original object. | Deep copy doesn’t reflect changes made to the new/copied object in the original object. |
| Shallow Copy stores the copy of the original object and points the references to the objects. | Deep copy stores the copy of the original object and recursively copies the objects as well. |
| Shallow copy is faster. | Deep copy is comparatively slower. |

**5. How exception handling works in python?**

Python uses try and except keywords to handle exceptions. Both keywords are followed by indented blocks.

Syntax:

try:

#statements in try block

except:

#executed when error in try block

The try: block contains one or more statements which are likely to encounter an exception. If the statements in this block are executed without an exception, the subsequent except: block is skipped.

If the exception does occur, the program flow is transferred to the except: block. The statements in the except: block are meant to handle the cause of the exception appropriately. For example, returning an appropriate error message.

We can specify the type of exception after the except keyword. The subsequent block will be executed only if the specified exception occurs. There may be multiple except clauses with different exception types in a single try block. If the type of exception doesn't match any of the except blocks, it will remain unhandled and the program will terminate.

**Very Important Programs:**

**1. Write a python program for count character from string after each character is changed.**

**e.g. input = 'aaaabahhhhhaaa' output = a4b1a1h5a3**

s = "aaaabahhhhhaaa"  
i = 0  
while i < len(s) - 1:  
 count = 1  
 while s[i] == s[i + 1]:  
 i += 1  
 count += 1  
  
 if i + 1 == len(s):  
 break  
 print(str(s[i]) + str(count), end = '')  
 i += 1

**2. You have a list of N+1 integers between 1 and N. You know there’s at least one duplicate,**

**but there might be more. For example, if N=3, your list might be 3, 1, 1, 3 or it might be 1, 3, 2, 2**

**Print out a number that appears in the list more than once.**

l = [1, 1, 3, 3, 4, 5, 1, 5]  
a = []  
for i in l:  
 if l.count(i) > 1 and i not in a:  
 a.append(i)  
print("Numbers that appears in list more than one are:", a)