100 Days of code: The complete Python pro bootcamp for 2023

# Strings:

Each character is encoded in the ASCII or Unicode symbol. So we can say that Python strings are also called the collection of Unicode characters.

1. str3 = ''''' Triple quotes are generally used for
2. represent the multiline or
3. docstring'''
4. **print**(str3)

**Output:**

Hello Python

Hello Python

Triple quotes are generally used for

represent the multiline or

docstring

1. str = "JAVATPOINT"
2. **print**(str[0:])
3. **print**(str[1:5])
4. **print**(str[:3])

**Output:**

JAVATPOINT

AVAT

JAV

1. **print**(str[-2:])
2. **print**(str[-4:-1])
3. **print**(str[-7:-2])
4. # Reversing the given string
5. **print**(str[::-1])

**Output:**

NT

OIN

ATPOI

TNIOPTAVAJ

1. str1 = " world"
2. str = "Hello"
3. **print**(str\*3) # prints HelloHelloHello
4. **print**(str+str1)# prints Hello world
5. **print**('w' **in** str) # prints false as w is not present in str
6. **print**('wo' **not** **in** str1) # prints false as wo is present in str1.
7. **print**(r'C://python37') # prints C://python37 as it is written
8. **print**("The string str : %s"%(str)) # prints The string str : Hello

### Regex Module

A set of characters with highly specialized syntax that we can use to find or match other characters or groups of characters. In short, regular expressions, or Regex, are widely used in the UNIX world.

Here's the list of the **metacharacters**;

“””. ^ $ \* + ? { } [ ] \ | ( )”””

#### Metacharacters

Metacharacters are characters with a special meaning:

|  |  |  |  |
| --- | --- | --- | --- |
| **Character** | **Description** | **Example** | **Try it** |
| [] | A set of characters  x = re.findall("[a-m]", txt)  (searches all the characters inbetween) | "[a-m]" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta1) |
| \ | Signals a special sequence (can also be used to escape special characters)  x = re.findall("\d", txt)  (searches the digits in a string) | "\d" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta2) |
| . | Any character (except newline character)  x = re.findall("he..o", txt)  we need to enter the exact number of dots | "he..o" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta3) |
| ^ | Starts with  x = re.findall("^hello", txt) | "^hello" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta4) |
| $ | Ends with  x = re.findall("planet$", txt) | "planet$" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta5) |
| \* | Zero or more occurrences  x = re.findall("he.\*o", txt) | "he.\*o" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta6) |
| + | One or more occurrences  Search for a sequence that starts with "he", followed by 1 or more (any) characters, and an "o":  x = re.findall("he.+o", txt) | "he.+o" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta7) |
| ? | Zero or one occurrences  x = re.findall("he.?o", txt) | "he.?o" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta10) |
| {} | Exactly the specified number of occurrences  x = re.findall("he.{2}o", txt) | "he.{2}o" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta8) |
| | | Either or  x = re.findall("falls|stays", txt) | "falls|stays" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_meta9) |
| () | Capture and group |  |  |

#### Special Sequence:

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

|  |  |  |  |
| --- | --- | --- | --- |
| **Character** | **Description** | **Example** | **Try it** |
| \A | Returns a match if the specified characters are at the beginning of the string  x = re.findall("\AThe", txt) #sentence | "\AThe" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_seq1) |
| \b | Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")  x = re.findall(r"\bain", txt) #word  x = re.findall(r"ain\b", txt) #word  Here r character (r’portal’) stands for raw, not RegEx. The raw string is slightly different from a regular string, it won’t interpret the \ character as an escape character. This is because the regular expression engine uses \ character for its own escaping purpose. | r"\bain" r"ain\b" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_seq2) [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_seq2-2) |
| \B | Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")   1. #Check if "ain" is present, but NOT at the beginning of a word: 2. x = re.findall(r"\Bain", txt) 3. #Check if "ain" is present, but NOT at the end of a word: 4. x = re.findall(r"ain\B", txt) | r"\Bain" r"ain\B" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_seq3) [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_seq3-2) |
| \d | Returns a match where the string contains digits (numbers from 0-9)  x = re.findall("\d", txt) | "\d" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_seq4) |
| \D | Returns a match where the string DOES NOT contain digits  x = re.findall("\D", txt) | "\D" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_seq5) |
| \s | Returns a match where the string contains a white space character  x = re.findall("\s", txt) | "\s" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_seq6) |
| \S | Returns a match where the string DOES NOT contain a white space character  x = re.findall("\S", txt) | "\S" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_seq7) |
| \w | Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore \_ character)  #Return a match at every word character (characters from a to Z, digits from 0-9, and the underscore \_ character):  x = re.findall("\w", txt) | "\w" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_seq8) |
| \W | Returns a match where the string DOES NOT contain any word characters  x = re.findall("\W", txt) | "\W" | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_seq9) |
| \Z | Returns a match if the specified characters are at the end of the string  x = re.findall("Spain\Z", txt) | "Spain\Z" |  |

#### Sets:

A set is a set of characters inside a pair of square brackets [] with a special meaning:

|  |  |  |
| --- | --- | --- |
| **Set** | **Description** | **Try it** |
| [arn] | Returns a match where one of the specified characters (a, r, or n) is present  x = re.findall("[arn]", txt) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_set1) |
| [a-n] | Returns a match for any lower case character, alphabetically between a and n  x = re.findall("[a-n]", txt) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_set2) |
| [^arn] | Returns a match for any character ***EXCEPT*** a, r, and n  x = re.findall("[^arn]", txt) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_set3) |
| [0123] | Returns a match where any of the specified digits (0, 1, 2, or 3) are present  x = re.findall("[01238]", txt) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_set4) |
| [0-9] | Returns a match for any digit between 0 and 9  x = re.findall("[0-9]", txt) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_set5) |
| [0-5][0-9] | Returns a match for any two-digit numbers from 00 and 59  x = re.findall("[0-5][0-9]", txt) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_set6) |
| [a-zA-Z] | Returns a match for any character alphabetically between a and z, lower case OR upper case  x = re.findall("[a-zA-Z]", txt) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_regex_set7) |
| [+] | In sets, +, \*, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string  x = re.findall("[+]", txt) |  |

#### The findall() Function:

import re  
  
txt = "The rain in Spain"  
x = re.findall("Portugal", txt)  
print(x)

output:

[]  
No match

#### The search() Function:

import re  
txt = "The rain in Spain"  
x = re.search("\s", txt)  
  
print("The first white-space character is located in position:", x.start())

output:

The first white-space character is located in position: 3

2.import re

#Split the string at the first white-space character:

txt = "The rain in Spain"

x = re.split("\s", txt, 1)

print(x)

ouput:

[‘The’, ‘rain in Spain’]

#### The sub() Function:

The sub() function replaces the matches with the text of your choice:

import re

#Replace all white-space characters with the digit "9":

txt = "The rain in Spain"

x = re.sub("\s", "9", txt)

print(x)

output:

The9rain9in9Spain

#### Match Object:

A Match Object is an object containing information about the search and the result.

The Match object has properties and methods used to retrieve information about the search, and the result:

.span() returns a tuple containing the start-, and end positions of the match.  
.string returns the string passed into the function  
.group() returns the part of the string where there was a match

Import re

Txt = “The rain in Spain”

X = re.search(r”\bS\w+, Txt)

Print(x.span())

**OutPut:**

(12,17)

Import re

Txt = “The rain in Spain”

X = re.search(r”\bS\w+”, Txt)

Print(x.string)

**Output:**

The rain in Spain

3.Import re

Txt = “The rain in Spain”

X = re.search(r”\bS\w+”, Txt)

Print(x.group())

**Output:**

Spain

Example:

### Functools Module:

#### Reduce() function:

Reduce(function, iterable, initializer = None)

**1.**

**from** functools **import** reduce  
mylist = [1, 2, 3, 4, 5]  
result = reduce(**lambda** x, y: x + y, mylist)  
print(result)

Output:

15

2.

**from** functools **import** reduce  
mylist = [1, 3, 5, 2, 4]  
result = reduce(**lambda** x, y: x **if** x > y **else** y, mylist)  
print(result)

**Output:**

5

3.

**from** functools **import** reduce  
my\_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
result = reduce(**lambda** x, y: x + y, my\_list)  
print(result)

**Output:**

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

#### **@lru\_cache** decorator

lru cache() is a decorator that wraps a function in a memoizing callable that saves up to maximize the results of a function call and returns the stored value when the function is called again with the same inputs.

#### Partial()

The partial() method in Functool is used to construct partial functions/objects, which is an important feature because it allows for:

* Replication of existing functions with some parameters already set.
* In a well-documented manner, create a newer version of an existing function.

#### Partialmethod()

The partialmethod() method provides a new partial method descriptor, which is similar to partial but is intended to be used as a method specification rather than a callable method. You can think of it as a method's partial().

#### Single dispatch()

The first is a generic function, which is a function made up of numerous functions that all perform the same task for different types. The dispatch algorithm determines which implementation will be utilized during a call.

The second is the Single dispatch, which is a type of generic function dispatch in which the implementation is determined by a single argument's type.

#### Singledispatchmethod():

It's a decorator that works in the same way as @singledispatch, but for methods instead of functions.

#### Catched\_property()

The cached property() decorator changes a class method into a property whose value is calculated only once and then cached as a normal attribute throughout the life of the instance, as the name suggests

#### Total\_ordering()

Given a class that defines one or more rich comparison ordering methods (equivalent to, =, >, >=, and ==), such as \_\_lt\_\_(), \_\_le\_\_(), \_\_gt\_\_(), \_\_ge\_\_(), or \_\_eq\_\_()).

#### Update\_wrapper()

It makes a wrapper function's metadata look like the wrapped function. In the case of partial functions, update wrapper(partial, parent) will update the partial function's documentation( doc ) and name( name ) to match the parent function's.

Wraps()

It's just a shortcut for calling update wrapper() on the decorated function. It's the same as calling partial(update wrapper, wrapped=wrapped, assigned=assigned, updated=updated, wrapped=wrapped).

#### Cmp\_to\_key()

It converts a key function from an old-style comparison function. Any callable that accepts two parameters, compares them and returns a negative number for less-than, zero for equality, or a positive number for greater-than is referred to as a comparison function. The operator.itemgetter() key function is an example of a callable that accepts one argument and returns another value to be used as the sort key. Tools like sorted(), min(), max(), and itertools.groupby use key functions ().

### Collections Module:

***Table of Content:***

* [*Counters*](https://www.geeksforgeeks.org/python-collections-module/#counters)
* [*OrderedDict*](https://www.geeksforgeeks.org/python-collections-module/#ordereddict)
* [*DefaultDict*](https://www.geeksforgeeks.org/python-collections-module/#defaultdict)
* [*ChainMap*](https://www.geeksforgeeks.org/python-collections-module/#chainmap)
* [*NamedTuple*](https://www.geeksforgeeks.org/python-collections-module/#namedtuple)
* [*DeQue*](https://www.geeksforgeeks.org/python-collections-module/#deque)
* [*UserDict*](https://www.geeksforgeeks.org/python-collections-module/#userdict)
* [*UserList*](https://www.geeksforgeeks.org/python-collections-module/#userlist)
* [*UserString*](https://www.geeksforgeeks.org/python-collections-module/#userstring)

#### Counters:

Counter is a container included in the in the collections module. Containers are objects that hold objects.

from collections import Counter

print(Counter(['B', 'B', 'A', 'B', 'C', 'A', 'B', 'B', 'A', 'C']))

print(Counter({'A': 3, 'B': 5, 'C': 2}))

print(Counter(A = 3, B = 5, C = 2))

coun = Counter()

coun.update([1, 2, 3, 1, 2, 1, 1, 2])

print(coun)

coun.update([1, 2, 4])

print(coun)

c1 = Counter(A = 4, B = 3, C = 10)

c2 = Counter(A = 10, B = 3, C = 4)

c1.subtract(c2)

print(c1)

counn = Counter(a = 1, b = 2, c = 3)

print(list(coun.elements()))

for letter, count in coun.most\_common(2):

print('%s: %d' % (letter, count))

**Output:**

Counter({'B': 5, 'A': 3, 'C': 2})

Counter({'B': 5, 'A': 3, 'C': 2})

Counter({'B': 5, 'A': 3, 'C': 2})

Counter({1: 4, 2: 3, 3: 1})

Counter({1: 5, 2: 4, 3: 1, 4: 1})

Counter({'C': 6, 'B': 0, 'A': -6})

[1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 4]

1: 5

2: 4

#### Ordereddict

from collections import OrderedDict

# Create an ordered dictionary of key-value pairs

my\_dict = OrderedDict([('a', 1), ('b', 2), ('c', 3)])

# Add a new item to the end of the dictionary

my\_dict['d'] = 4

# Add a new item at a specific position in the dictionary

my\_dict.update({'e': 5}, [('f', 6)])

my\_dict.move\_to\_end('e', last=False)

# Iterate over the dictionary in the order in which items were added

for key, value in my\_dict.items():

print(key, value)

**Output:**

e 5

a 1

b 2

c 3

d 4

f 6

#### DefaultDict

Dictionary in Python is an unordered collection of data values that are used to store data values like a map

When the int class is passed as the default\_factory argument, then a defaultdict is created with default value as zero.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

from collections import defaultdict

d = defaultdict(int)

L = [1, 2, 3, 4, 2, 4, 1, 2]

for i in L:

d[i] += 1

print(d)

defaultdict(<class 'int'>, {1: 2, 2: 3, 3: 1, 4: 2})

#### ChainMap

**A ChainMap encapsulates many dictionaries into a single unit and returns a list of dictionaries.**

import collections

dic1 = { 'a' : 1, 'b' : 2 }

dic2 = { 'b' : 3, 'c' : 4 }

dic3 = { 'f' : 5 }

chain = collections.ChainMap(dic1, dic2)

print ("All the ChainMap contents are : ")

print (chain.maps)

chain1 = chain.new\_child(dic3)

print ("Displaying new ChainMap : ")

print (chain1.maps)

print ("Value associated with b before reversing is : ",end="")

print (chain1['b'])

chain1.maps = reversed(chain1.maps)

print ("Value associated with b after reversing is : ",end="")

print (chain1['b'])

**Output:**

All the ChainMap contents are :

[{'a': 1, 'b': 2}, {'b': 3, 'c': 4}]

Displaying new ChainMap :

[{'f': 5}, {'a': 1, 'b': 2}, {'b': 3, 'c': 4}]

Value associated with b before reversing is : 2

Value associated with b after reversing is : 3

#### NamedTuple

import collections

# Declaring namedtuple()

Student = collections.namedtuple('Student', ['name', 'age', 'DOB'])

# Adding values

S = Student('Nandini', '19', '2541997')

# using \_fields to display all the keynames of namedtuple()

print("All the fields of students are : ")

print(S.\_fields)

# .\_replace returns a new namedtuple, it does not modify the original

print("returns a new namedtuple : ")

print(S.\_replace(name='Manjeet'))

# original namedtuple

print(S)

**Output:**

All the fields of students are :

('name', 'age', 'DOB')

returns a new namedtuple :

Student(name='Manjeet', age='19', DOB='2541997')

Student(name='Nandini', age='19', DOB='2541997')

### Difflib module:

The **difflib** module in Python provides functionality for comparing sequences, particularly strings. It provides several classes and functions to find the differences between two sequences and present them in a human-readable format.

Here are some of the key features of the **difflib** module:

1. **SequenceMatcher**: This class provides a way to compare two sequences and get the differences between them. It works by finding the longest common subsequence between the sequences and then identifying the differences before and after it.

Here's an example of how to use **SequenceMatcher**:

makefileCopy code

import difflib s1 = "Hello world" s2 = "Hello there" matcher = difflib.SequenceMatcher(None, s1, s2) print(matcher.get\_opcodes())

Output:

cssCopy code

[('equal', 0, 5, 0, 5), ('delete', 5, 6, 5, 5), ('equal', 6, 11, 5, 10), ('insert', 11, 11, 10, 11)]

The **get\_opcodes()** method returns a list of tuples that represent the differences between the two sequences. In this case, the output shows that the first five characters are equal, then there is a deletion of one character in s1, then five more characters are equal, and finally there is an insertion of one character in s2.

1. **get\_close\_matches**: This function takes a word and a list of possible matches and returns a list of the best matches based on a similarity score. It is particularly useful for finding the closest match to a misspelled word.

Here's an example of how to use **get\_close\_matches**:

makefileCopy code

import difflib words = ["apple", "banana", "cherry", "durian"] misspelled\_word = "dureon" best\_matches = difflib.get\_close\_matches(misspelled\_word, words) print(best\_matches)

Output:

cssCopy code

['durian']

The **get\_close\_matches** function returns a list of the best matches for the misspelled word, based on their similarity score. In this case, it correctly identifies "durian" as the closest match to "dureon".

### Escape Sequence

1. **print**("They said, \"What's going on?\"")

**Output:**

They said, "What's going on?"

|  |  |
| --- | --- |
| ASCII Carriege Return(CR) | print("Hello \r World!")  **Output:**  World! |
| ASCII Horizontal Tab | print("Hello \t World!")  Output:  Hello World! |
| ASCII Vertical Tab | print("Hello \v World!")  Output:  Hello  World! |

### The format() method

1. **print**("{} and {} both are the best friend".format("Devansh","Abhishek"))
2. **print**("{1} and {0} best players ".format("Virat","Rohit"))
3. **print**("{a},{b},{c}".format(a = "James", b = "Peter", c = "Ricky"))

**Output:**

Devansh and Abhishek both are the best friend

Rohit and Virat best players

James, Peter,Ricky

### Python String join() Method

1. str = ""
2. list = ['J','a','v','a','t','p','o','i','n','t']
3. str2 = str.join(list)
4. print(f’{“ ".join(list)}’)
5. print(str2)

**Output:**

J a v a t p o i n t

Javatpoint

### Python String split() Method

split(sep=None, maxsplit=-1)

1. str = "Java is a programming language"
2. str2 = str.split()
3. **print**(str2)

**Output:**

['Java', 'is', 'a', 'programming', 'language']

1. str = "Java is a programming language"
2. str2 = str.split('a',1)
3. **print**(str2)
4. str2 = str.split('a',3)
5. **print**(str2)

**Output:**

['J', 'va is a programming language]

['J', 'v', ' is ', ' programming language'

### Use F-Strings for String Formatting

1. price = 12.3456
2. formatted\_price = f"The price is ${price:.2f}"
3. **print**(formatted\_price)
4. **Output:**
5. The price is $12.35

### Strip Function:

1. text = "!!!Hello, World!!!"
2. clean\_text = text.strip("!")
3. **print**(clean\_text)

**Output:**

Hello, World

### Python Docstrings

def my\_function():

''' Demonstrates triple double quotes

docstrings and does nothing really.'''

return None

print("Using \_\_doc\_\_:")

print(my\_function.\_\_doc\_\_)

print("Using help:")

help(my\_function)

|  |
| --- |
|  |

**Output:**

Using \_\_doc\_\_:

Demonstrates triple double quotes

docstrings and does nothing really.

Using help:

Help on function my\_function in module \_\_main\_\_:

my\_function()

Demonstrates triple double quotes

docstrings and does nothing really.

### Python – Find all duplicate characters in a string

def duplicate\_characters(string):

# Create a set to store characters seen before

seen = set()

# Create a set to store duplicate characters

duplicates = set()

# Iterate through each character in the string

for char in string:

# Check if the character has been seen before

if char in seen:

duplicates.add(char)

else:

seen.add(char)

# Convert the duplicates set to a list and return it

return list(duplicates)

# Test cases

print(duplicate\_characters("geeksforgeeks"))

**Output**

['g', 'e', 'k', 's']

### .items() method

**.items()** is a **built**-in method that can be used on a dictionary object. This method returns a view object that contains the key-value pairs of the dictionary as tuples.

Examples of this method:

1.person = {'name': 'John', 'age': 25, 'gender': 'male'}

for key, value in person.items():

print(key, value)

Output:

name John

age 25

gender male

2. person = {'name': 'John', 'age': 25, 'gender': 'male'}

person\_list = list(person.items())

print(person\_list)

Output:

[('name', 'John'), ('age', 25), ('gender', 'male')]

3.person = {'name': 'John', 'age': 25, 'gender': 'male'}

filtered\_person = {key: value for key, value in person.items() if key != 'age'}

print(filtered\_person)

Output:

{'name': 'John', 'gender': 'male'}

2.def find\_dup\_char(input):

x=[]

for i in input:

if i not in x and input.count(i)>1:

x.append(i)

print(" ".join(x))

# Driver program

if \_\_name\_\_ == "\_\_main\_\_":

input = 'geeksforgeeks'

find\_dup\_char(input)

**Output**

G e k s

3. from collections import Counter

def count\_and\_display\_vowels(string):

vowels = 'aeiouAEIOU'

vowels\_list = filter(lambda c: c in vowels, string)

count = Counter(vowels\_list)

return count

string = "Geeks for Geeks"

print(count\_and\_display\_vowels(string))

**Output**:

Counter({'e': 4, 'o': 1})

### Program for testing whether given input is valid password or not.

import re

print(Password := str(input("Enter your password:")))

flag = 0

while True:

if (len(Password) <= 8):

flag = -1

break

elif not re.search("[a-z]", Password):

flag = -1

break

elif not re.search("[A-Z]", Password):

flag = -1

break

elif not re.search("[0-9]", Password):

flag =- 1

break

elif not re.search("[\_@$]", Password):

flag = -1

break

elif re.search("\s", Password):

flag = -1

break

else:

flag = 0

print("Valid Password")

break

if flag == -1:

print("Not a valid password")

**Output**:

Enter your password:Prasad@222

Prasad@222

Valid Password

### Using list comprehension to know the number of vowels in the given string:

def vowel\_count(str):

# Creating a list of vowels

vowels = "aeiouAEIOU"

# Using list comprehension to count the number of vowels in the string

count = len([char for char in str if char in vowels])

# Printing the count of vowels in the string

print("No. of vowels :", count)

# Driver code

str = "GeeksforGeeks"

# Function Call

vowel\_count(str)

**Output**:

No. of vowels : 5

### Program to check for URL in a string

‘(?i)’ – This is a flag that sets the regular expression to be case-insensitive.

# Python code to find the URL from an input string

# Using the regular expression

import re

def Find(string):

# findall() has been used

# with valid conditions for URLs in string

regex = r"(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))\*\))+(?:\(([^\s()<>]+|(\([^\s()<>]+\)))\*\)|[^\s`!()\[\]{};:'\".,<>?«»“”‘’]))"

url = re.findall(regex, string)

return [x[0] for x in url]

# Driver Code

string = 'My Profile: https://auth.geeksforgeeks.org/user/Chinmoy%20Lenka/articles in the portal of https://www.geeksforgeeks.org/'

print("Urls: ", Find(string))

**Output:**

[‘https://auth.geeksforgeeks.org/user/Chinmoy%20Lenka/articles’, ‘https://www.geeksforgeeks.org/’]

### Check if a Substring is Present in a given String:

Note that **index()** method is case-sensitive, meaning that it will only find substrings that match the case of the characters in the string. If you want to perform a case-insensitive search, you can use the **find()** method instead, which works in a similar way but ignores the case of the characters in the string.

1.any\_string = "Geeks for Geeks substring "

start = 0

end = 1000

print(any\_string.index('substring', start, end))

**Output**:

16

\*\*\*\*

a = ['Geeks-13', 'for-56', 'Geeks-78', 'xyz-46']

for i in a:

if i.\_\_contains\_\_("Geeks"):

print(f"Yes! {i} is containing.")

**Output:**

Yes! Geeks-13 is containing.

Yes! Geeks-78 is containing.

**2**.s="geeks for geeks"

s2="geeks"

x=list(filter(lambda x: (s2 in s),s.split()))

print(["yes" if x else "no"])

**Output:**

[‘Yes’]

**3.**import operator as op

s="geeks for geeks"

s2="geeks"

print(["yes" if op.countOf(s.split(),s2)>0 else "no"])

**Output:**

[‘Yes’]

As similar to contains method………….

if(op.contains(s,s2)):

**#Sorting a list in ascending order**

Numbers = [5, 2, 8, 1, 3]

Sorted\_numbers = sorted(Numbers)

Numbers.sort()

Print(Numbers)

Print(Sorted\_numbers)

**Output:**

**[1, 2, 3, 5, 8]**

**[1, 2, 3, 5, 8]**

**#Sorting a list in descending order**

Numbers = [5,3,6,7,2,3,8]

Sorted\_numbers = sorted(Numbers, reverse = True)

Print(Sorted\_numbers)

**Output:**

**[8, 7, 6, 5, 3, 3, 2]**

### Python program to check if two strings are anagram:

**Def** check(s1, s2):

If (**sorted**(s1) == **sorted**(s2)):

Print(“Yes, anagram.”)

Else:

Print(“No, anagram.”)

S1 = “listen”

S2 = “silent”

Check(s1, s2)

**Output:**

Yes, anagram

**‘Counter()’** function from the ‘**Collections’** module to count the occurrences of each character in both strings. **‘Counter()’**  returns a dictionary containing the count of each character in the string.

**# Python3 program for the above approach**

from collections import Counter

def check(s1, s2):

if(Counter(s1) == Counter(s2)):

print("The strings are anagrams.")

else:

print("The strings aren't anagrams.")

s1 = "listen"

s2 = "silent"

check(s1, s2)

**Output:**

The strings are anagrams.

**\*\*\*\***

inp1 = "listen"

inp2 = "silenti"

#Sort Elements

x = [inp1[i] for i in range(0,len(inp1))]

x.sort()

y = [inp2[i] for i in range(0,len(inp2))]

y.sort()

# the sorted strings are checked

if (x == y):print("The strings are anagrams.")

else: print("The strings aren't anagrams.")

**Output:**

The strings aren’t anargams.

### Lambda Expressions:

Lamda arguments : expressions

Python Lambda Functions are anonymous function means that the function is without a name.

1.

Calc = lambda num : “Even Number” if num % 2 == 0 else “Odd number”

Print(calc(20))

**Output**:

Even Number

2.

filter\_nums = lambda s: ''.join([ch for ch in s if not ch.isdigit()])

print("filter\_nums():", filter\_nums("Geeks101"))

`

do\_exclaim = lambda s: s + '!'

print("do\_exclaim():", do\_exclaim("I am tired"))

find\_sum = lambda n: sum([int(x) for x in str(n)])

print("find\_sum():", find\_sum(101))

**Output:**

Filter\_nums() : Geeks

Do\_exclaim(): I am tired!

Find\_sum() : 2

3.

My\_list = [1, ,2, 3, 4, 5]

New\_list = list(filter(lambda x: x % 2 ! = 0, my\_list))

Print(new\_list)

**Output:**

[1, 3, 5]

4.

my\_list = [1, 2, 3, 4, 5]  
transformed\_list = list(map(**lambda** x: x \*\* 2, filter(**lambda** x: x % 2 == 0, my\_list)))  
print(transformed\_list) *# Output: [4, 16]*

*5.*

mylist = [**'apple'**, **'banana'**, **'cherry'**, **'data'**, **'elderberry'**]  
sortedlist = sorted(mylist, key = **lambda** x: x[1] + x[0])  
print(sortedlist)

**Output:**

['banana', 'data', 'cherry', 'elderberry', 'apple']

### Map() function:

Map(function, iterable)

**1.**

Def myfunc(a, b):

Return a+b

X = map(mufunc, (‘apple’, ‘banana’, ‘cherry’), (‘orange’, ‘lemon’ ‘pineapple’))

Print(x)

Print(list(x)

**Output:**

<map object at 0x034244F0>

[‘appleorange’, ‘bananalemon’, ‘cherrypineapple’]

2.

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

Squares = map(lambda x: x\*\*2, numbers)

Print(list(squares))

**Output:**

[1, 4, 9, 16, 25]

3.

words = [**'apple'**, **'banana'**, **'cherry'**, **'date'**, **'elderberry'**]  
vowels = [**'a'**, **'e'**, **'i'**, **'o'**, **'u'**]  
no\_vowels = map(**lambda** x: **''**.join(filter(**lambda** y: y.lower() **not in** vowels, x)), words)  
print(list(no\_vowels))

**Output:**

['ppl', 'bnn', 'chrry', 'dt', 'ldrbrry']

### Filter() function:

Filter(function, iterable)

1.

my\_list = [**'apple'**, **''**, **'banana'**, **''**, **'cherry'**]  
filtered\_list = list(filter(**lambda** x: x != **''**, my\_list))  
print(filtered\_list)

**Output:**

[‘apple’, ‘banana’, ‘cherry’]

2.

ages = [5, 12, 17, 18, 24, 32]  
  
def myFunc(x):  
  if x < 18:  
    return False  
  else:  
    return True  
adults = filter(myFunc, ages)  
for x in adults:  
  print(x)

**Output:**

18

24

32

### Program for Replacing characters:

**def** replaceChars(input, c1, c2):  
 newChars = map(**lambda** x: x **if** (x != c1 **and** x != c2) **else** c1 **if** (x == c2) **else** c2, input)  
 print(**''**.join(newChars))  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 input = **'grrksfoegrrks'** c1 = **'e'** c2 = **'r'** replaceChars(input, c1, c2)

**Output:**

Geeksforgeeks

**Alternative**

def replaceChars(input, c1, c2):

newChars = ''

for x in input:

if x == c1:

newChars += c2

elif x == c2:

newChars += c1

else:

newChars += x

print(newChars)

if \_\_name\_\_ == "\_\_main\_\_":

input = 'grrksfoegrrks'

c1 = 'e'

c2 = 'r'

replaceChars(input, c1, c2)

### Map function and dictionary in python to sum ASCII values:

**def** asciiSums(sentence):  
 words = sentence.split(**' '**)  
 result = {}  
 **for** word **in** words:  
 currentSum = sum(map(ord, word))  
 result[word] = currentSum  
 totalSum = 0  
 sumsOfAscii = [result[word] **for** word **in** words]  
 print(**'Sum of ASCII values:'**)  
 print(**' '**.join(map(str, sumsOfAscii)))  
 print(**'Total Sum -> '**, sum(sumsOfAscii))  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 sentence = **'I am a geek'** asciiSums(sentence)

**Output:**

Sum of ASCII values:

73 206 97 412

Total Sum -> 788

### Program to find the longest common word in two strings

**def** largest\_common\_word(str1, str2):words1 = str1.split()  
 words2 = str2.split()longest\_word = **''****for** w1 **in** words1:  
 **for** w2 **in** words2:**if** w1 == w2 **and** len(w2) > len(longest\_word):  
 longest\_word = w1  
  
 **return** longest\_word  
str1 = **"The quick brown fox jumps over the lazy dog"**str2 = **"The lazy dog is jumped over by the quick brown fox"**print(largest\_common\_word(str1, str2))