Python Cheat Sheet (2023)

Python is one of the most widely used and popular programming languages, was developed by Guido van Rossum and released first in 1991. Python is a free and open-source language with a very simple syntax and easy-to-learn features. It is easy for developers to **learn Python**. It supports object-oriented programming and is most commonly used to perform general-purpose programming. Python is used in several domains like Data Science, Machine Learning, Deep Learning, Artificial Intelligence, Scientific Computing Scripting, Networking, Game Development Web Development, Web Scraping, and various other domains.

Python Cheat Sheet

To give a comprehensive overview of Python programming, we made a **Python Cheat Sheet** for Python programmers. In this **Cheat Sheet of Python**, you'll learn all the basics to advanced topics and concepts of Python, like *Python data types, Python for loop, Python slice, python map function, python dictionary, Python file Handling, etc.*

Python Cheat sheet

- What is Python?
- Python Input
- Operators in Python
- Python Indentation
- Data Types in Python
- Python String
- Conditional Statements
- Python Functions
- Python Built-in Functions
- Python Regex
- File Handling in Python
- Try and Except Statement
- Python OOPs Concepts
- Python Modules
- Python Interview Questions Answers
- Python Interview MCQs

What is Python?

Python is one of the most widely used and popular programming languages, was developed by Guido van Rossum and released first on February 20, 1991. Python is a free and open-source language with a very simple and clean syntax which makes it easy for developers to learn Python. It supports object-oriented programming and is most commonly used to perform general-purpose programming. Python is used in several domains like Data Science, Machine Learning, Deep Learning, Artificial Intelligence, Scientific Computing Scripting, Networking, Game Development Web Development, Web Scraping, and various other domains.

Python Program to Print Hello world

The print() function in Python is used to print Python objects as strings as standard output.

Python3

```
# Python program to print "Hello world"
print("Hello world")
```

Output:

```
Hello World
```

Python end parameter in print()

The end parameter can be used to avoid the new line after the output or end the output with a different string.

Python3

```
# Python program to print "Hello world"
# with the cursor on same line
print("Hello world", end = ' ')
print("Geeksforforgeeks", end = ' ')
```

Output:

```
Hello World Geeksforforgeeks
```

Python sep parameter in print()

The separator between the inputs to the print() method in Python is by default a space, however, this can be changed to any character, integer, or string of our choice. The sep parameter is used to do the same thing.

Python3

```
# Python program showing
# use of sep parameter
print("Geeks", "for", "Geeks", sep = ' ')
print("Geeks",
print("Geeks", "geeksforgeeks", sep = '@')
```

Output:

```
Geeks for Geeks
Geeks@geeksforgeeks
```

Python Input

The input() method in Python is used to accept user input. By default, it returns the user input as a string. By default, the input() function accepts user input as a string.

Python3

```
# Python program showing
# use of input()
s = input("Enter your value: ")
print("Hello ", s)
```

Output:

```
Enter your value: Hello Geeks
Hello Geeks
```

Operators in Python

In general, Operators are used to execute operations on values and variables. These are standard symbols used in logical and mathematical processes.

Arithmetic Operators

Python Arithmetic Operators are used to perform mathematical operations like addition, subtraction, multiplication, and division.

Python3

```
# Examples of arithmetic operator
a = 9
b = 4

# Addition of numbers
add = a + b

# Subtraction of numbers
sub = a - b

# Multiplication of number
mul = a * b

# Division of both number
mul = a / b

# Modulo of both
mod = a % b

# Print results
print(add)
print(sub)
print(mul)
print(mod)
print(a/b)
```

Output:

```
13
5
36
1
2.25
```

Comparison Operators

When comparing values, relational operators are utilized. Depending on the criteria, it returns True or False. Relational Operators are another name for these operators.

Python3

```
# Examples of Relational Operators
a = 13
b = 33

# a is greater than b or not
print(a > b)

# a is lesser than b or not
print(a < b)

# a is b or not
print(a == b)

# a is not b or not
print(a != b)

# a is greater than
# equal to b or not
print(a >= b)

# a is lesser than
# equal to b or not
print(a <= b)
```

Output:

```
False
True
False
True
False
True
```

Logical Operators in Python

Logical operators are used on conditional statements in Python (either True or False). They conduct the logical AND, OR, and NOT operations.

Python3

```
# Examples of Logical Operator
a = True
b = False

# Print a and b is true
print(a and b)

# Print a or b is true
print(a or b)

# Print not a is false
print(not a)
```

Output:

```
False
True
False
```

Bitwise Operators in Python

Bitwise operators are used in Python to do bitwise operations on integers. After converting the numbers to binary, operations are done on each bit or corresponding pair of bits, hence the name bitwise operators.

Python3

```
# Examples of Bitwise operators
a = 25
b = 20

# Print bitwise and operation
print(a & b)

# Print bitwise or operation
print(a | b)

# Print bitwise NOT operation
print(~ a)

# Print bitwise xor operation
print(a ^ b)

# Print bitwise right shift operation
print(a >> 2)

# Print bitwise left shift operation
print(a << 2)
```

Output:

```
8
29
-26
5
100
```

Python Comment

Comments in Python are the lines in the code that are ignored by the interpreter during the execution of the program. There are three types of comments in Python:

- Single line Comments
- Multiline Comments
- Docstring Comments

Python3

```
# This is the comment
# Python program to demonstrate
# multiline comment

"""This Python program to demonstrate
Multiline Comment"""

name = "geeksforgeeks"
print(name)
```

Output:

```
geeksforgeeks
```

DataType in Python

The type() function can be used to define the values of various data types and to check their data types.

Python3

```
# Integer datatype - int
x = "Hello World"
# Integer datatype - hex
x = 20
# Integer datatype - float
x = 20.5
# Integer datatype - complex
x = 1j
# Integer datatype - list
x = ["geeks", "for", "geeks"]
# Integer datatype - tuple
x = ("geeks", "for", "geeks")
# Integer datatype - range
x = range(10)
# Integer datatype - dict
x = {"name": "Harsh", "age": 20}
# Integer datatype - set
x = {"geeks", "for", "geeks"}
# Integer datatype - frozenset
x = frozenset({"geeks", "for", "geeks"})
# Integer datatype - bool
x = True
# Integer datatype - bytes
x = b"Hello"
# Integer datatype - bytearray
x = bytearray(5)
# Integer datatype - memoryview
x = memoryview(bytes(5))
# Integer datatype - NoneType
x = None
```

List in Python

The Python list is a sequence data type that is used to store the collection of data. Tuples and String are other types of sequence data types.

Python3

```
lst = ['Geeks', 'for', 'Geeks']
print(lst)
```

Output:

```
['Geeks', 'for', 'Geeks']
```

List comprehension

A Python list comprehension is made up of brackets carrying the expression, which is run for each element, as well as the for loop, which is used to iterate over the Python list's elements.

Also, Read - Python Arrays

Python3

```
# Creating list comprehension to iterate through list
lst = [character for character in [1, 2, 3]]
print(lst)
# printing list
print(lst[0])
```

Output:

```
[1, 2, 3]
```

Dictionary in Python

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

Python3

```
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print(Dict)
# printing dictionary
print(Dict[1])
```

Output:

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

Python Dictionary Comprehension

Like List Comprehension, Python allows dictionary comprehensions. We can create dictionaries using simple expressions. A dictionary comprehension takes the form {key: value for (key, value) in iterable}

Python3

```
# Dictionary comprehension to square each value
Dict = {x: x**2 for x in [1, 2, 3, 4, 5]}
print(Dict)

# Adding a key value pair in dictionary comprehension
Dict = {x: x**2 for x in [1, 2, 3, 4, 5]}
print(Dict)

# Access one of the values
# using the corresponding key
print(Dict[3])
```

Output:

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Tuples in Python

Tuple is a list-like collection of Python objects. A tuple stores a succession of values of any kind, which are indexed by integers.

Python3

```
lst = ['Geeks', 'for', 'Geeks']
print(lst)
```

Output:

```
['Geeks', 'for', 'Geeks']
```

Set in Python

A Python set is an unordered collection of data types that can be iterated, mutated and contains no duplicate elements. The order of the elements in a set is unknown, yet it may contain several.

Python3

```
lst = ['Geeks', 'for', 'Geeks']
print(lst)
```

Output:

```
['Geeks', 'for', 'Geeks']
```

Trending in News

- 10 Best Tutorial Alternatives to LEAN Stack
- How to Get Quality Promises Free Courses on GeeksforGeeks
- Nikita Kalyuzhna Instagram to Foundation: Ukrainian Model Fundraiser
- Sample Interview Intro & Other Interview
- Top 20 Project Ideas For Beginners in DSA

Python3

```
var = ("Tuples", "for", "beds")
print(var)
```

Output:

```
('For', 'beds')
```

Python String

In Python, a string is a data structure that represents a collection of characters. A string cannot be changed once it has been formed because it is an immutable data type.

Creating and Accessing String

Strings in Python can be created using single quotes or double quotes or even triple quotes. In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String.

Python3

```
string1 = 'GeeksforGeeks'
print("Initial String: ")
print(string1)

# printing first character
print("\nFirst character of String is: ")
print(string1[0])

# printing last character
print("\nLast character of String is: ")
print(string1[-1])
```

Output:

```
Initial String:
GeeksforGeeks

First character of String is:
G

Last char racter of String is:
s
```

String Slicing

Strings in Python can be constructed with single, double, or even triple quotes. The slicing method is used to access a single character or a range of characters in a String. A String operator (colon) is used to slice a String

Python3

```
# creating a string
string = 'GeeksforGeeks'
print("Initial String: ")
print(string)

# printing 3rd character
print("\nSlicing characters from 3-12: ")
print(string[3:12])

# printing 3rd character to 12th character
# and skip one character
print("\nSlicing characters between 3-12: ")
print(string[3:12:2])
```

Output:

```
Initial String:
GeeksforGeeks

Slicing characters from 3-12:
ksforGe

Slicing characters between 3rd and 2nd last character:
ksforGe
```

Conditional Statements

Decision-making statements in programming languages decide the direction(Control Flow) of the flow of program execution.

Python If-Else

In a conditional Statement, the additional block of code is merged as an else statement which is performed when if condition is false.

Python3

```
# Python program to illustrate if else statement
i = 10
if (i < 15):
    print("i is smaller than 15")
    print("i'm also one digit")
else:
    print("i is greater than or equal to 15")
```

Output:

```
i is greater than 15
i'm not in if and not in else Block
```

Python For Loop

The Python For loop is used for sequential traversal, that is, iterating over an iterable such as a String, Tuple, List, Set, or Dictionary. For loops in Python only support collection-based iteration.

Python3

```
# Python program to illustrate
# for loop with string
for i in "geeks":
    print(i)
```

Output:

```
g
e
e
k
s
```

Python While Loop

The Python while loop is used to execute a set of statements repeatedly until a condition is met. When the condition is met, the loop immediately following the loop in the program is run.

Python3

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

Output:

```
Hello Geek
Hello Geek
Hello Geek
```

For more read, refer this break, continue, and pass in Python

Python Functions

Python functions are a collection of statements that serve a specific purpose. The idea is to bring together some often or repeatedly performed actions and construct a function so that we can more the code included in it rather than writing the same code for different inputs over and over.

Python3

```
# Defining function
def add():
    num = 2
    num = num + 2
    print(num)
# Driver code to call a function
add()
```

Output:

```
welcome to GFG
```

Function Arguments

Arguments are the values given between the function's parenthesis. A function can take as many parameters as it wants, separated by commas.

Python3

```
# A simple Python function to check
# whether x is even or odd
def evenOdd(x):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")

# Driver code to call the function
evenOdd(2)
evenOdd(3)
```

Output:

```
even
odd
```

Return Statement in Python Function

The function return statement is used to terminate a function and return to the function caller with the provided value or data item.

Python3

```
# Python program to
# illustrate return statement
def cube(x, n):
    # A returning one if x and n
    # return a 1 if
    if x == 1:
        return 1
    # Else return the product of x
    # cube and n
    else:
        return x * cube(x, n - 1)

# A driver code function
# to test the above
print("5 raised to 3 is", cube(5, 3))
print("1 raised to 3 is", cube(1, 3))
```

Output:

```
5 raised to 3 is 125
1 raised to 3 is 1
```

The range() function

The Python range() function returns a sequence of numbers, in a given range.

Python3

```
# Print first 5 integers
# using range() function
for i in range(5):
    print("Hello, GeE")
print("Bye")
```

Output:

```
0 1 2 3 4
```

*args and **kwargs in Python

The Python with *args and **kwargs keywords allow functions to take variable-length parameters. The number of non-keyworded arguments and the actual values that can be performed on the tuple are specified by the "*args"Rwargs, on the other hand, pass a variable number of keyword arguments dictionary to function, which can then do dictionary operations.

Python3

```
def myFun(arg1, arg2, arg3):
    print("arg1: " + arg1)
    print("arg2: " + arg2)
    print("arg3: " + arg3)

# Now we can use *args or **kwargs to
# pass a variable number of
# arguments to a function
args = ("Geeks", "for", "Geeks")
myFun(*args)

kwargs = {"arg3": "Geeks", "arg2": "for", "arg1": "Geeks"}
myFun(**kwargs)
```

Output:

```
arg1: Geeks
arg2: for
arg3: Geeks
arg1: Geeks
arg2: for
arg3: Geeks
```

Python Builtin Function

There are numerous built-in functions in Python that make creating code easier. Learn about the built-in functions of Python in this post as we examine their numerous uses and highlight a few of the most popular ones.

Python Map Function

The Python function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable.

Python3

```
# Square value of a
def square(x):
    return x * x

# Driver code (without using map)
lst = [1, 2, 3, 4, 5]
result = map(square, lst)
print(list(result))
```

Output:

```
[1, 4, 9, 16]
```

Python Filter Function

The Filter method filters the given sequence using a function that examines each element in the sequence to see if it is true or false.

Python3

```
# Function that filters vowels
def fun(variable):
    letters = ['a', 'e', 'i', 'o', 'u']
    if (variable in letters):
        return True
    else:
        return False

# sequence
sequence = ['g', 'e', 'e', 'k', 's', 'f', 'o', 'r', 'g', 'e', 'e', 'k']

# using filter function
filtered = filter(fun, sequence)

print('The filtered letters are:')
for s in filtered:
    print(s)
```

Output:

```
The filtered letters are:
e
e
e
```

Python Reduce Function

The reduce function is used to apply a certain function to all of the list components indicated in the sequence sent along.

Python3

```
# Import reduce from functools module
from functools import reduce

lst = [1, 2, 3, 4, 5]
print(reduce(lambda x, y: x + y, lst))
```

Output:

```
15
```

Python Lambda

Python lambda functions are anonymous, which means they have no name. As we already know, the def keyword is used to define a normal function in Python. The lambda keyword in Python is used to declare an anonymous function.

Python3

```
# Lambda name "cube" takes one argument
cube = lambda num: "cube numbers" if num > 0 else "not number"

print(cube(5))
```

Output:

```
cube number
```

For More Read, refer Python Built in Functions

Python RegEx

We define a pattern using a Regular Expression to match email addresses. The pattern r"\b[A-Za-z0-9. %+-]+@[A-Za-z0-9.-]+.[A-Za-z]{2,}\b" is a common pattern for matching email addresses. Using the re.search() function, the pattern is then found in the given text. If a match is found, we use the .group() method to extract and print the matched email. Otherwise, a message indicating that no email was found is displayed.

Python3

```
import re

# Define the pattern for email addresses
pattern = r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+.[A-Za-z]{2,}\b"

# Example text containing email addresses
text = "Contact us at support@example.com or sales@example.com"

# Search for the pattern in the text
match = re.search(pattern, text)

# If a match is found, print the matched email
if match:
    matched_email = match.group()
    print("Found email:", matched_email)
else:
    print("No email found.")
```

Output:

```
Found email: support@example.com
```

MetaCharacters are helpful, significant, and will be used in module in functions, which helps us comprehend the analogy with RE. The list of metacharacters is shown below.

1. Used to drop the special meaning of character following it.
2. Represent a character class
3. * Matches the beginning.
4. $ Matches the end.
5. Matches any character except newline.
6. | Means OR (Matches with any of the characters separated by it.
7. * Matches zero or one occurrence.
8. * Any number of occurrences (including 0 occurrences).
9. + One or more occurrences.
10. {} Indicate the number of occurrences of a preceding RegEx to match.
11. () Enclose a group of RegEx.

For More RegEx example, refer to Python RegEx.

File Handling in Python

Python has supports File handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files.

Python3

```
import os

def create_file(filename):
    try:
        with open(filename, 'w') as f:
            print("File '{}' created successfully.".
                  format(filename))
    except IsADirectoryError:
        print("Error: '{}' is a directory, not a file.".
              format(filename))
    except PermissionError:
        print("Error: Permission denied to create file '{}'.
              format(filename))
    except Exception as e:
        print("Error: Unexpected error occurred while creating file '{}': {}".
              format(filename, str(e)))

def read_file(filename):
    try:
        with open(filename, 'r') as f:
            content = f.read()
            print("File '{}' contents:".
                  format(filename))
            print(content)
    except FileNotFoundError:
        print("Error: File '{}' not found.".
              format(filename))
    except PermissionError:
        print("Error: Permission denied to read file '{}'.
              format(filename))
    except Exception as e:
        print("Error: Unexpected error occurred while reading file '{}': {}".
              format(filename, str(e)))

def write_file(filename, content):
    try:
        with open(filename, 'w') as f:
            f.write(content)
            print("Content written to file '{}'.
                  format(filename))
    except PermissionError:
        print("Error: Permission denied to write to file '{}'.
              format(filename))
    except Exception as e:
        print("Error: Unexpected error occurred while writing to file '{}': {}".
              format(filename, str(e)))

def delete_file(filename):
    try:
        os.remove(filename)
        print("File '{}' deleted successfully.".
              format(filename))
    except FileNotFoundError:
        print("Error: File '{}' not found.".
              format(filename))
    except PermissionError:
        print("Error: Permission denied to delete file '{}'.
              format(filename))
    except Exception as e:
        print("Error: Unexpected error occurred while deleting file '{}': {}".
              format(filename, str(e)))

# Example usage
filename = "example.txt"
content = "This is some content to write to the file.\n"
create_file(filename)
read_file(filename)
write_file(filename, content)
read_file(filename)
delete_file(filename)
```

Output:

```
File 'example.txt' created successfully.
Error: File 'example.txt' not found.
Content written to file 'example.txt'.
File 'example.txt' contents:
This is some content to write to the file.

File 'example.txt' deleted successfully.
```

```
try:
    with open("example.txt", mode_file_read) ?
        print("File: ", filename, ") removed to ")
    with open("example.txt", mode_file_read) ?
    except IOerror:
        print("Error: could not rename file: ", filename)

def delete_file(filename):
    try:
        os.remove(filename)
        print("File: ", filename, ") deleted successfully ")
    except IOerror:
        print("Error: could not delete file: ", filename)

if __name__ == "__main__":
    create_file("example.txt")
    rename_file("example.txt", "example_new.txt")

    create_file("filename")
    write_file("filename", "This is some additional text.\n")
    read_file("filename")
    rename_file("filename", new_filename)
    read_file("new_filename")
    delete_file("new_filename")
```

Output:

```
File example.txt created successfully.
Hello, world!

Text appended to file example.txt successfully.
Hello, world!
This is some additional text.

File example.txt renamed to new_example.txt successfu...
```

Try and Except Statement

In Python, Exceptions and Errors are used to catch and manage exceptions. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause.

Python3

```
# [1, [2, [3, 4]]]
# [1, 2, 3, 4]
print("Nested element = %d" %a[1][1])

# Error because cannot access 2nd element in array
print("an error occurred", %a[1][1])

except:
    print("An error occurred")
```

Output:

```
Nested element = 2
an error occurred
```

Python OOPs Concepts

Object Oriented Programming (OOPs) is a programming paradigm in Python that employs objects and classes. It seeks to include real-world entities such as inheritance, polymorphism, encapsulation, and so on into programming. The primary idea behind OOPs is to join the data and the functions that act on it as a single unit so that no other portion of the code can access it.

- Class Introduction
- Polymorphism
- Encapsulation
- Inheritance
- Data Abstraction

In this example, we have a Car class with characteristics that represent the car's make, model, and year. The _make attribute is protected with a single underscore , The __model attribute is marked as private with two underscores __. The year attribute is open for the public.

We can use the getter function get_make() to retrieve the protected attribute _make. We can use the setter method set_model() to edit the private attribute __model. Using the getter method get_model(), we may retrieve the changed private attribute __model. There are no restrictions on accessing the public attribute year. We leverage the visibility and accessibility of class members by using encapsulation with private and protected properties, offering a level of data hiding and abstraction.

Python3

```
class car:
    def __init__(self, make, model, year):
        self._make = make  # protected attribute
        self.__model = model  # private attribute
        self.year = year  # public attribute

    def get_make(self):
        return self._make

    def get_model(self, model):
        self.__model = model

    def get_model(self):
        return self.__model

my_car = car("Honda", "Civic", 2013)

print(my_car.get_make())  # accessing protected attribute
my_car.set_model("Jazz")  # modifying private attribute
print(my_car.get_model())  # accessing modified private attribute
print(my_car.year)  # accessing public attribute
```

Output:

```
Toyota
Camry
2012
```

Python Modules

A library is a group of modules that collectively address a certain set of requirements or applications. A module is a file (py file) that includes functions, class defining statements, and variables linked to a certain activity. The term "standard library modules" refers to the pre-installed Python modules.

- Calendar
- Numpy
- Math
- Os
- Pandas
- Random
- Shelve
- Sqlite

Python Interview Questions Answers

- Python Interview Questions Answers

Python Cheat Sheet - FAQs

Q. What are the key features of Python?

Features of Python includes readability, simplicity, extensive library support, cross-platform compatibility, dynamic typing, a strong community, and object-oriented programming (OOP).

Q. What type of language is Python? Programming or scripting?

Python is capable of scripting, but in a general sense, it is considered a general-purpose programming language.

Q. What is PEP 8?

*PEP stands for **Python Enhancement Proposal**. It is a set of rules that specify how to format Python code for maximum readability.*

Q. What is the difference between .py and .pyc files?

The .py files are the python source code files and the .pyc files contain the bytecode of the python files. .pyc files are generated when the code is imported from some other source. The interpreter converts the source .py files to .pyc files which helps by saving time.

Q. What are the main data structures in Python?

Python supports lists, tuples, sets, and dictionaries as its primary data structures, each serving different purposes.

Don't miss your chance to ride the wave of the data revolution! Every industry is scaling new heights by tapping into the power of data. Sharpen your skills and become a part of the hottest trend in the 21st century.

Dive into the future of technology - explore the Complete Machine Learning and Data Science Program by GeeksforGeeks and stay ahead of the curve.

Last Updated : 12 Mar, 2024

Previous
HTML Cheat Sheet - A Basic Guide to HTML

Share your thoughts in the comments
Add Your Comment

Similar Reads

Completely Cheat Sheet for Python Operations
Python Open() Cheat Sheet
Regex Cheat Sheet - Python
jQuery Cheat Sheet - A Basic Guide to jQuery
Tkinter Cheat Sheet
CSS Cheat Sheet - A Basic Guide to CSS
Linux Commands Cheat Sheet
Git Cheat Sheet
pip(pip3) Cheat Sheet
Subnet Mask Cheat Sheet

Suggest a Topic
+ Topic

Article Tags : GFG Theory, python basics, Python-Functions, Python-Operators, Python
Practice Tags : Python

GeeksforGeeks
A 24th, 25th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305
Company
About Us
Legal
Careers
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
Explore
Job & Internships
Hack-A-Thons
GFG Weekly Contest
DSA to JAVA/C++
Mobile System Design
Master CP
GeeksforGeeks Videos
Geeks Community
Languages
Python
Java
C++
PHP
Golang
SQL
Scala
Kotlin
Web Development
Tutorials Archive
DSA
Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets
Data Science & BI
Data Science With Python
Data Science For Beginner
Machine Learning Tutorial
Pandas Tutorial
NumPy Tutorial
NLP Tutorial
Deep Learning Tutorial
HTML & CSS
HTML
CSS
Web Templates
CSS Frameworks
Bootstrap
Tailwind CSS
SASS
LESS
Web Design
Design Tutorial
Python Tutorial
Python Programming Examples
Python Projects
Python Packages
Flask Tutorial
Pandas Tutorial
Python Interview Question
Computer Science
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
World GK
DevOps
Git
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap
Competitive Programming
Top 50 Array Sol For CP
Top 50 Tree
Top 50 Graph
Top 50 String
Top 50 DP
Top 10 Algorithms For CP
Management & Finance
Management
HR Management
Finance
Income Tax
Organizational Behaviour
Marketing
Free Online Tools
Typing Test
Image Editor
Code Formatters
Code Converters
Currency Converter
Random Number Generator
Random Password Generator
System Design
High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions
More Tutorials
Software Development
Software Testing
Product Management
SAP
SEO - Search Engine Optimization
Linux
Excel
JavaScript
JavaScript Examples
TypeScript
ReactJS
NextJS
AngularJS
NodeJS
Lodash
Web Browser
GeeksforGeeks Videos
DSA
Python
Java
C++
Data Science
CS Subjects
geeksforGeeks, Sanchhaya Education Private Limited, All rights reserved