What is Django?

Python-based web framework used for rapid development.

Installing Django + Setup

```
pip install django
```

Creating a project

The below command creates a new project

```
django-admin startproject projectName
```

Starting a server

The below command starts the development server.

```
python manage.py runserver
```

Django MVT

Django follows MVT(Model, View, Template) architecture.

Sample Django Model

The model represents the schema of the database.

```
from django.db import models

class Product(models.Model): //Product is the name of our model
product_id=models.AutoField
```

Sample views.py

View decides what data gets delivered to the template.

```
from django.http import HttpResponse
```

```
def index(request):
return HttpResponse(''Django CodeWithHarry Cheatsheet'')
```

Sample HTML Template

A sample .html file that contains HTML, CSS and Javascript.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>CodeWithHarry Cheatsheet</title>
</head>
<body>
<h1>This is a sample template file.</h1>
</body>
</html>
```

Views in Django

Sample Function-Based Views

A python function that takes a web request and returns a web response.

```
from django.http import HttpResponse

def index(request):
return HttpResponse(''This is a function based view.')
```

Sample Class-Based Views

Django's class-based views provide an object-oriented (OO) way of organizing your view code.

```
from django.views import View

class SimpleClassBasedView(View):
def get(self, request):
... # code to process a GET request
```

URLs in Django

set of URL patterns to be matched against the requested URL.

Sample urls.py file

```
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
path('admin/', admin.site.urls),
path('', views.index, name='index'),
path('about/', views.about, name='about'),

]
```

Forms in Django

Similar to HTML forms but are created by Django using the form field.

Sample Django form

```
from django import forms

# creating a form
class SampleForm(forms.Form):
Name = forms.CharField()
description = forms.CharField()
```

Apps in Django

Apps in Django are like independent modules for different functionalities.

Creating an app

```
python manage.py startapp AppName
```

Listing app in the settings.py

After creating an app, we need to list the app name in INSTALLED_APPS

```
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'AppName'
]
```

Templates in Django

Used to handle dynamic HTML files separately.

Configuring templates in settings.py

```
TEMPLATES = [
{
'BACKEND': 'django.template.backends.django.DjangoTemplates',
'DIRS': ["templates"],
'APP_DIRS': True,
'OPTIONS': {
# ... some options here ...
},
},
]
```

Changing the views.py file

A view is associated with every template file. This view is responsible for displaying the content from the template.

```
def index(request):
return render(request, 'index.html') ; #render is used to return the templat
```

Sample template file

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
<meta charset="UTF-8">
<title>Template is working</title>
</head>
<body>
<h1>This is a sample django template.</h1>
</body>
</html>
```

Migrations in Django

Migrations are Django's way of updating the database schema according to the changes that you make to your models.

Creating a migration

The below command is used to make migration but no changes are made to the actual database.

```
python manage.py makemigrations
```

Applying the migration

The below command is used to apply the changes to the actual database.

```
python manage.py migrate
```

Admin interface in Django

Django comes with a ready-to-use admin interface.

Creating the admin user

```
python manage.py createsuperuser
```

Page Redirection

Redirection is used to redirect the user to a specific page of the application on the occurrence of an event.

Redirect method

```
from django.shortcuts import render, redirect

def redirecting(request):
return redirect("https://www.codewithharry.com")
```

Database

It is defined as a collection of interrelated data stored together to serve multiple applications.

MySQL Elements

MySQL has certain elements that play an important role in querying a database.

Literals

Literals refer to a fixed data value

```
17 #It is a numeric literal
"Harry" #It is a text literal
12.5 #It is a real literal
```

Data Types

Data types are means to identify the type of data.

```
#Numeric

INT -- Integer data type
TINYINT
SMALLINT
MEDIUMINT
BIGINT

FLOAT(M,D) -- Floating point data type
DOUBLE(M,D) -- Double data type also stores decimal values
DECIMAL(M,D) -- Decimal data type


#Data and Time

DATE -- Date data type (YYYY-MM-DD)
DATETIME -- It's a date and time combination (YYYY-MM-DD HH:MM:SS)
TIME -- It stores time (HH:MM:SS)
```

```
#String/Text

CHAR(M) -- Character data type
VARCHAR(M) -- Variable character data type
BLOB or TEXT
```

NULL Values

If a column has no value, then it is said to be NULL

Comments

A comment is a text that is not executed.

```
/* This is a multi-line
comment in MySQL */

# It is a single-line commend

-- It is also a single-line comment
```

MySQL Simple Calculations

You can perform simple calculations in MySQL, just by using the Select command, there's no need to select any particular database to perform these commands.

Addition

It will add two numbers

```
Select 5+8;
```

Subtraction

It will subtract the second number from first

```
Select 15-5;
```

Multiplication

It will give the product of supplied numbers

```
Select 5*5;
```

Division

It will divide the number

```
Select 24/4;
```

```
-- SQL is not a case-sensitive language
```

Accessing Database

These commands allow one to check all the databases and tables

Show command

It will show all the databases in the system

```
Show databases;
```

It will show all the tables in a selected database

```
show tables;
```

Use command

It will start using the specified database i.e. now you can create tables in the selected database

```
use database_name;
```

Creating tables

These commands allow you to create the table in MySQL

Create table command

This query is used to create a table in the selected database

```
Create table <table-name>
(<column_name> <data_type>,
<column_name> <data_type>,
<column_name> <data_type>);
```

Insert command

It will add data into the selected table

```
Insert into <table_name> [<column-list>]
Values (<value1>,<value2>...);
```

Inserting NULL values

This query will add NULL value in the col3 of the selected table

```
Inset into <table-name> (col1, col2,col3)
Values (val1,val2,NULL);
```

Inserting Dates

It will add the following data into the selected column of the table

```
Insert into <table_name> (<col_name>)
Values ('2021-12-10');
```

Select Command

A select query is used to fetch the data from the database

Selecting All Data

It will retrieve all the data of the selected table

```
Select * From <table_name>;
```

Selecting Particular Rows

It will retrieve all the data of the row that will satisfy the condition

```
Select * from <table_name>
Where <condition_to_satisfy>;
```

Selecting Particular Columns

It will retrieve data of selected columns that will satisfy the condition

```
Select column1, column2 from <table_name>
Where <condition_to_satisfy>;
```

DISTINCT Keyword

It will retrieve only distinct data i.e. duplicate data rows will get eliminated

```
Select DISTINCT <column_name> from <table_name>;
```

ALL Keyword

It will retrieve all the data of the selected column

```
Select ALL <column_name> from <table_name>;
```

Column Aliases

It is used to give a temporary name to a table or a column in a table for the purpose of a particular query

```
Select <column1>,<column2> AS <new_name>
From <table_name>;
```

Condition Based on a Range

It will only retrieve data of those columns whose values will fall between value1 and value2 (both inclusive)

```
Select <co11>, <col2>
From <table_name>
Where <value1> Between <value2>;
```

Condition Based on a List

```
Select * from <table_name>
Where <column_name> IN (<val1>,<val2>,<val3>);
```

```
"Select * from <table_name>
Where <column_name> NOT IN (<val1>,<val2>,<val3>);"
```

Condition Based on Pattern Match

```
Select <col1>,<col2>
From <table_name>
Where <column> LIKE 'Ha%';
```

```
Select <col1>,<col2>
From <table_name>
Where <column> LIKE 'Ha__y%';
```

Searching NULL

It returns data that contains a NULL value in them

```
Select <column1>, <column2>
From <table_name> Where <Val> IS NULL;
```

SQL Constraints

SQL constraints are the rules or checks enforced on the data columns of a table

NOT NULL

It will create a table with NOT NULL constraint to its first column

```
Create table <table_name>
( <col1> <data_type> NOT NULL,
<col2> <data_type>,
<col3> <data_type>);
```

DEFAULT

DEFAULT constraint provides a default value to a column

```
Create table <table_name>
( <col1> <data_type> DEFAULT 50,
<col2> <data_type>,
<col3> <data_type>);
```

UNIQUE

UNIQUE constraint ensures that all values in the column are different

```
Create table <table_name>
( <col1> <data_type> UNIQUE,
<col2> <data_type>,
<col3> <data_type>);
```

CHECK

CHECK constraint ensures that all values in a column satisfy certain conditions

```
Create table <table_name>
( <col1> <data_type> CHECK (condition),
<col2> <data_type>,
<col3> <data_type>);
```

Primary Key

Primary key is used to uniquely identify each row in a table

```
Create table <table_name>
( <col1> <data_type> Primary Key,
<col2> <data_type>,
<col3> <data_type>);
```

Foreign Key

```
CREATE TABLE Orders (
OrderID int NOT NULL,
OrderNumber int NOT NULL,
PersonID int,
```

```
PRIMARY KEY (OrderID),
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

Viewing Table Structure

Desc or Describe command

It allows you to see the table structure

```
Desc <table_name>;
```

Modifying Data

Update Command

It will update the values of selected columns

```
Update <table_name>
SET <col1> = <new_value>, <col2> = <new_value>
Where <condition>;
```

Deleting Data

Delete Command

It will delete the entire row that will satisfy the condition

```
Delete From <table_name>
Where <condition>;
```

Ordering Records

Order by clause is used to sort the data in ascending or descending order of specified column

order by clause

It will return records in the ascending order of the specified column name's data

```
Select * from <table_name> order by <column_name>;
```

It will return records in the descending order of the specified column name's data

```
Select * from <table_name> order by <column_name> DESC;
```

Ordering data on multiple columns

It will return records in the ascending order of column1 and descending order of column2

```
Select * From <table_name> order by <column1> ASC, <column2> DESC;
```

Grouping Result

It is used to arrange identical data into groups so that aggregate functions can work on them

Group by clause

It allows you to group two or more columns and then you can perform aggregate function on them

```
Select <column>, Count(*) from <table_name> group by <column>;
```

Having clause

Having clause is used to put conditions on groups

```
Select avg(<column>), sum(<column>) from <table_name> group by <column_name>
```

Altering Table

These commands allow you to change the structure of the table

To Add New Column

It will add a new column in your table

```
Alter Table <table_name>
Add <new_column>;
```

To Modify Old Column

It will update the data type or size of old column

```
Alter Table <table_name>
Modify <old_column_name> [<new_data_type><size>];
```

To Change Name of Column

It will change the name of the old column in the table

```
Alter Table Change <old_column_name> <new_column_name><data_type>;
```

Dropping Table

DROP command

It will delete the complete table from the database

```
Drop table <table_name>;
```

MySQL Functions:

There are many functions in MySQL that perform some task or operation and return a single value

Text/String Functions

Text function work on strings

Char Function

It returns the character for each integer passed

```
Select Char(72,97,114,114,121);
```

Concat Function

It concatenates two strings

```
Select Concat("Harry","Bhai");
```

Lower/Lcase

It converts a string into lowercase

```
Select Lower("Harry Bhai");
```

Upper/Ucase

It converts a string into uppercase

```
Select Upper("CodeWithHarry");
```

Substr

It extracts a substring from a given string

```
Select Substr(string,m,n);
```

Trim

It removes leading and trailing spaces from a given string

```
Select Trim(leading ' ' FROM ' Harry Bhai');
```

Instr

It searches for given second string into the given first string

```
Select Instr(String1,String2);
```

Length

It returns the length of given string in bytes

```
Select Length(String)
```

Numeric Functions

Numeric function works on numerical data and returns a single output

MOD

It returns modulus of two numbers

```
Select MOD(11,4);
```

Power

It returns the number m raised to the nth power

```
Select Power(m,n);
```

Round

It returns a number rounded off number

```
Select Round(15.193,1);
```

Sqrt

It returns the square root of a given number

```
Select Sqrt(144);
```

Truncate

It returns a number with some digits truncated

```
Select Truncate(15.75,1);
```

Date/Time Functions

These are used to fetch the current date and time and allow you to perform several operations on them

Curdate Function

It returns the current date

```
Select Curdate();
```

Date Function

It extracts the date part of the expression

```
Select Date('2021-12-10 12:00:00');
```

Month Function

It returns the month from the date passed

```
Select Month(date);
```

Day Function

It returns the day part of a date

```
Select Day(date);
```

Year Function

It returns the year part of a date

```
Select Year(date);
```

Now Function

It returns the current date and time

```
Select now();
```

Sysdate Function

It returns the time at which function executes

```
Select sysdate();
```

Aggregate Functions

Aggregate functions or multiple row functions work on multiple data and returns a single result

AVG Function

It calculates the average of given data

```
Select AVG(<column_name>) "Alias Name" from <table_name>;
```

COUNT Function

It counts the number of rows in a given column

```
Select Count(<column_name>) "Alias Name" from <table_name>;
```

MAX Function

It returns the maximum value from a given column

```
Select Max(<column_name>) "Alias Name" from <table_name>;
```

MIN Function

It returns the minimum value from a given column

```
Select Min(<column_name>) "Alias Name" from <table_name>;
```

SUM Function

It returns the sum of values in given column

```
Select Sum(<column_name>) "Alias Name" from <table_name>;
```

MySQL Joins

Join clause is used to combine or merge rows from two or more tables based on a related attribute

INNER JOIN

It returns all rows from multiple tables where the join condition is satisfied. It is the most common type of join.

```
SELECT columns FROM table1 INNER JOIN table2 ON table1.column = table2.colum
```

LEFT OUTER JOIN

It returns all rows from the left-hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

```
SELECT columns FROM table1 LEFT [OUTER] JOIN table2 ON table1.column = table
```

RIGHT OUTER JOIN

It returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where the join condition is satisfied

```
SELECT columns FROM table1 RIGHT [OUTER] JOIN table2 ON table1.column = tabl
```

FULL JOIN

It combines the results of both left and right outer joins

```
SELECT column_name FROM table1 FULL OUTER JOIN table2 ON table1.column_name
```

SELF JOIN

In this join, table is joined with itself

```
SELECT column_name FROM table1 T1, table1 T2 WHERE condition;
```

Basics

Hello World

echo function is used to display or print output

```
<?php echo "Hello World!"; ?>
```

Comments

Commets are used to make the code more understandable for programmer, they are not executed by compiler or interpreter.

One Liner

This is a singleline comment

```
// Twinkle Twinkle Little Star
```

Another One Liner

This is a single-line comment

```
# Chocolate dedo mujhe yaar
```

Multiline

This is a multiline comment

```
/* Code With
Harry */
```

Vardump

This function dumps information about one or more variables.

```
<?php var_dump(var1, var2, ...); ?>
```

Variables

Variables are "containers" for storing information.

Defining Variables

```php
<?php
$Title = "PHP Cheat Sheet By CodeWithHarry";
?>
```

Datatypes

Datatype is a type of data

String

A string is a sequence of characters, like "Hello world!".

```php
<?php
$x = "Harry";
echo $x;
?>
```

Integer

An integer is a number without any decimal part.

```php
<?php
$x = 1234;
var_dump($x);
?>
```

Float

A float is a number with a decimal point or a number in exponential form.

```php
<?php
$x = 1.2345;
var_dump($x);
?>
```

Array

An array stores multiple values in one single variable

```php
<?php
$names = array("Harry","Rohan","Shubham");
var_dump($names);
?>
```

Class

A class is a template for objects

```php
<?php
class Harry{
// code goes here...
}
?>
```

Object

An object is an instance of the class.

```php
<?php
class Bike {
public $color;
public $model;
public function __construct($color, $model) {
$this->color = $color;
$this->model = $model;
}
public function message() {
return "My bike is a " . $this->color . " " . $this->model . "!";
}
}

$myBike = new Bike("red", "Honda");
echo $myBike -> message();
?>
```

Escape Characters

Escape sequences are used for escaping a character during string parsing. It is also used for giving special meaning to represent line breaks, tabs, alerts and more.

Line feed

It adds a newline

```
\n
```

Carriage return

It inserts a carriage return in the text at this point.

```
\r
```

Horizontal tab

It gives a horizontal tab space

```
\t
```

Vertical tab

It gives a vertical tab space

```
\v
```

Escape

It is used for escape characters

```
\e
```

Form feed

It is commonly used as page separators but now is also used as section separators.

```
\f
```

Backslash

It adds a backslash

```
\\
```

Dollar sign

Print the next character as a dollar, not as part of a variable

```
\$
```

Single quote

Print the next character as a single quote, not a string closer

```
\'
```

Double quote

Print the next character as a double quote, not a string closer

```
\"
```

Operators

Operators are symbols that tell the compiler or interpreter to perform specific mathematical or logical manipulations. These are of several types.

Arithmetic Operators

Addition

Sum of $x and $y

```
$x + $y
```

Subtraction

Difference of $x and $y

```
$x - $y
```

Multiplication

Product of $x and $y

```
$x * $y
```

Division

Quotient of $x and $y

```
$x / $y
```

Modulus

The remainder of $x divided by $y

```
$x % $y
```

Exponentiation

Result of raising $x to the $y'th power

```
$x ** $y
```

PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.

x = y

The left operand gets set to the value of the expression on the right

```
x = y
```

x += y

Addition

```
x = x + y
```

x -= y

Subtraction

```
x = x - y
```

x *= y

Multiplication

```
x = x * y
```

x /= y

Division

```
x = x / y
```

x %= y

Modulus

```
x = x % y
```

PHP Comparison Operators

Equal

Returns true if $x is equal to $y

```
$x == $y
```

Identical

Returns true if $x is equal to $y, and they are of the same type

```
$x === $y
```

Not equal

Returns true if $x is not equal to $y

```
$x != $y
```

Not equal

Returns true if $x is not equal to $y

```
$x <> $y
```

Not identical

Returns true if $x is not equal to $y, or they are not of the same type

```
$x !== $y
```

Greater than

Returns true if $x is greater than $y

```
$x > $y
```

Less than

Returns true if $x is less than $y

```
$x < $y
```

Greater than or equal to

Returns true if $x is greater than or equal to $y

```
$x >= $y
```

Less than or equal to

Returns true if $x is less than or equal to $y

```
$x <= $y
```

PHP Increment / Decrement Operators

Pre-increment

Increments $x by one, then returns $x

```
=++$x
```

Post-increment

Returns $x, then increments $x by one

```
$x++
```

Pre-decrement

Decrements $x by one, then returns $x

```
--$x
```

Post-decrement

Returns $x, then decrements $x by one

```
$x--
```

PHP Logical Operators

And

True if both $x and $y are true

```
$x and $y
```

Or

True if either $x or $y is true

```
$x or $y
```

Xor

True if either $x or $y is true, but not both

```
$x xor $y
```

And

True if both $x and $y are true

```
$x && $y
```

Or

True if either $x or $y is true

```
$x || $y
```

Not

True if $x is not true

```
!$x
```

PHP String Operators

Concatenation

Concatenation of $txt1 and $txt2

```
$txt1 . $txt2
```

Concatenation assignment

Appends $txt2 to $txt1

```
$txt1 .= $txt2
```

PHP Array Operators

Union

Union of $x and $y

```
$x + $y
```

Equality

Returns true if $x and $y have the same key/value pairs

```
$x == $y
```

Identity

Returns true if $x and $y have the same key/value pairs in the same order and of the same types

```
$x === $y
```

Inequality

Returns true if $x is not equal to $y

```
$x != $y
```

Inequality

Returns true if $x is not equal to $y

```
$x <> $y
```

Non-identity

Returns true if $x is not identical to $y

```
$x !== $y
```

PHP Conditional Assignment Operators

Ternary

Returns the value of $x. The value of $x is expr2 if expr1 = TRUE. The value of $x is expr3 if expr1 = FALSE

```
$x = expr1 ? expr2 : expr3
```

Conditional Statements

Conditional statements are used to perform operations based on some condition.

If Statement

if statement checks the condition and if it is True, then the block of if statement executes; otherwise, control skips that block of code.

```
if (condition) {
// code to execute if condition is met
}
```

If..Else

if the condition of if block evaluates to True, then if block executes otherwise else block executes

```
if (condition) {
// code to execute if condition is met
} else {
// code to execute if condition is not met
}
```

If..Elseif..Else

It executes different codes for more than two conditions

```
if (condition) {
// code to execute if condition is met
} elseif (condition) {
// code to execute if this condition is met
```

```
} else {
// code to execute if none of the conditions are met
}
```

Switch Statement

It allows a variable to be tested for equality against a list of values (cases).

```
switch (n) {
case x:
code to execute if n=x;
break;
case y:
code to execute if n=y;
break;
case z:
code to execute if n=z;
break;
// add more cases as needed
default:
code to execute if n is neither of the above;
}
```

Loops

Iterative statements or Loops facilitate programmers to execute any block of code lines repeatedly.

For Loop

It is used to iterate the statements several times. It is frequently used to traverse the data structures like the array and linked list.

```
for (starting counter value; ending counter value; increment by which
to increase) {
// code to execute goes here
}
```

Foreach Loop

The foreach loop loops through a block of code for each element in an array.

```
foreach ($InsertYourArrayName as $value) {
// code to execute goes here
}
```

While Loop

It iterate the block of code as long as a specified condition is True or vice versa

```
while (condition that must apply) {
// code to execute goes here
}
```

Do-While Loop

This loop is very similar to the while loop with one difference, i.e., the body of the do-while loop is executed at least once even if the condition is False. It is an exit-controlled loop.

```
do {
// code to execute goes here;
} while (condition that must apply);
```

Predefined Variables

PHP provides a large number of predefined variables to all scripts. The variables represent everything from external variables to built-in environment variables, last error messages etc. All this information is defined in some predefined variables.

$GLOBALS

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script.

```
<?php
$a = 10;
$b = 15;

function addition() {
$GLOBALS['c'] = $GLOBALS['a'] + $GLOBALS['b'];
}
addition();
```

```
echo $c;
?>
```

$_SERVER

Returns the filename of the currently executing script. $_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

```
$_SERVER['PHP_SELF']
```

Returns the version of the Common Gateway Interface (CGI) the server is using

```
$_SERVER['GATEWAY_INTERFACE']
```

Returns the IP address of the host server

```
$_SERVER['SERVER_ADDR']
```

Returns the name of the host server (such as www.codewithharry.com)

```
$_SERVER['SERVER_NAME']
```

Returns the server identification string (such as Apache/2.2.24)

```
$_SERVER['SERVER_SOFTWARE']
```

Returns the name and revision of the information protocol (such as HTTP/1.1)

```
$_SERVER['SERVER_PROTOCOL']
```

Returns the request method used to access the page (such as POST)

```
$_SERVER['REQUEST_METHOD']
```

Returns the timestamp of the start of the request (such as 1377687496)

```
$_SERVER['REQUEST_TIME']
```

Returns the query string if the page is accessed via a query string

```
$_SERVER['QUERY_STRING']
```

Returns the Accept header from the current request

```
$_SERVER['HTTP_ACCEPT']
```

Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)

```
$_SERVER['HTTP_ACCEPT_CHARSET']
```

Returns the Host header from the current request

```
$_SERVER['HTTP_HOST']
```

Returns the complete URL of the current page (not reliable because not all user-agents support it)

```
$_SERVER['HTTP_REFERER']
```

Is the script queried through a secure HTTP protocol?

```
$_SERVER['HTTPS']
```

Returns the IP address from where the user is viewing the current page

```
$_SERVER['REMOTE_ADDR']
```

Returns the Hostname from where the user is viewing the current page

```
$_SERVER['REMOTE_HOST']
```

Returns the port being used on the user's machine to communicate with the web server

```
$_SERVER['REMOTE_PORT']
```

Returns the absolute pathname of the currently executing script

```
$_SERVER['SCRIPT_FILENAME']
```

Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@codewithharry.com)

```
$_SERVER['SERVER_ADMIN']
```

Returns the port on the server machine being used by the webserver for communication (such as 80)

```
$_SERVER['SERVER_PORT']
```

Returns the server version and virtual hostname which are added to server-generated pages

```
$_SERVER['SERVER_SIGNATURE']
```

Returns the file system based path to the current script

```
$_SERVER['PATH_TRANSLATED']
```

Returns the path of the current script

```
$_SERVER['SCRIPT_NAME']
```

Returns the URI of the current page

```
$_SERVER['SCRIPT_URI']
```

$_GET

PHP $_GET is a PHP super global variable which is used to collect form data after submitting an HTML form with method="get".

```
<?php
echo "Hello" . $_GET['Example'] . " at " . $_GET['web'];
?>
```

$_POST

PHP $_POST is a PHP super global variable which is used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

```
<html>
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
Name: <input type="text" name="fname">
<input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
$name = $_POST['fname'];
if (empty($name)) {
echo "Please Enter your name";
} else {
echo $name;
}
}
?>
</body>
</html>
```

$_REQUEST

PHP $_REQUEST is a PHP super global variable which is used to collect data after submitting an HTML form.

```
<html>
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
Name: <input type="text" name="fname">
<input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
$name = $_REQUEST['fname'];
if (empty($name)) {
echo "Name is empty";
} else {
echo $name;
}
```

```
}
?>
</body>
</html>
```

Variable-handling Functions

The PHP variable handling functions are part of the PHP core. No installation is required to use these functions.

boolval

Boolval is used to get the boolean value of a variable

```
<?php
echo '0: '.(boolval(0) ? 'true' : 'false')."\n";
echo '42: '.(boolval(42) ? 'true' : 'false')."\n";
echo '0.0: '.(boolval(0.0) ? 'true' : 'false')."\n";
echo '4.2: '.(boolval(4.2) ? 'true' : 'false')."\n";
echo '"": '.(boolval("") ? 'true' : 'false')."\n";
echo '"string": '.(boolval("string") ? 'true' : 'false')."\n";
echo '"0": '.(boolval("0") ? 'true' : 'false')."\n";
echo '"1": '.(boolval("1") ? 'true' : 'false')."\n";
echo '[1, 2]: '.(boolval([1, 2]) ? 'true' : 'false')."\n";
echo '[]: '.(boolval([]) ? 'true' : 'false')."\n";
echo 'stdClass: '.(boolval(new stdClass) ? 'true' : 'false')."\n";
?>
```

isset

It is used to check whether a variable is empty. It also checks whether the variable is set/declared:

```
<?php
$x = 0;
// True because $x is set
if (isset($x)) {
echo "Variable 'x' is set";
}
```

unset

It unsets variables.

```php
<?php
$a = "Namaste world!";
echo "The value of 'a' before unset: " . $a ;
unset($a);
echo "The value of 'a' after unset: " . $a;
?>
```

debug_zval_dump

debug_zval_dump is used to dump a string representation of an internal zval structure to output

```php
<?php
$var1 = 'Hello';
$var1 .= ' World';
$var2 = $var1;

debug_zval_dump($var1);
?>
```

empty

Empty is used to check whether a variable is empty or not.

```php
<?php
$var = 0;

// Evaluates to true because $var is empty
if (empty($var)) {
echo '$var is either 0, empty, or not set at all';
}

// Evaluates as true because $var is set
if (isset($var)) {
echo '$var is set even though it is empty';
}
?>
```

floatval

It returns the float value of different variables:

```php
<?php
$var = '122.34343The';
$float_value_of_var = floatval($var);
echo $float_value_of_var; // 122.34343
?>
```

get_defined_vars

It returns all defined variables, as an array:

```php
<?php
$b = array(1, 1, 2, 3, 5, 8);

$arr = get_defined_vars();

// print $b
print_r($arr["b"]);

/* print path to the PHP interpreter (if used as a CGI)
* e.g. /usr/local/bin/php */
echo $arr["_"];

// print the command-line parameters if any
print_r($arr["argv"]);

// print all the server vars
print_r($arr["_SERVER"]);

// print all the available keys for the arrays of variables
print_r(array_keys(get_defined_vars()));
?>
```

get_resource_type

It returns the resource type:

```php
<?php
// prints: stream
$fp = fopen("foo", "w");
echo get_resource_type($fp) . "\n";

// prints: curl
```

```
$c = curl_init ();
echo get_resource_type($c) . "\n"; // works prior to PHP 8.0.0 as since 8.0
?>
```

gettype

It returns the type of different variables:

```
<?php
$a = 3;
echo gettype($a) ;
?>
```

intval

It returns the integer value of different variables:

```
<?php
echo intval(42); ?>
```

is_array

To check whether a variable is an array or not:

```
<?php
$a = "Hello";
echo "a is " . is_array($a) ;?>
```

Array

An array stores multiple values in one single variable.

Declaring an Array

```
<?php
$cms = array("Harry", "Lovish", "Rohan");
echo "Who needs chocolate? Is it" . $cms[0] . ", " .
$cms[1] . " or " . $cms[2] . "?";
?>
```

Functions

A function is a block of statements that can be used repeatedly in a program

Defining Functions

```
function NameOfTheFunction() {
//place PHP code here
}
```

MySQLi Functions

These functions allow you to access MySQL database server.

mysqli_connect() Function

It opens a non-persistent MySQL connection

```
mysqli_connect()
```

mysqli_affected_rows() Function

It returns the number of affected rows

```
mysqli_affected_rows()
```

mysqli_connect_error() Function

It shows the Error description for the connection error

```
mysqli_connect_error()
```

mysqli_fetch_all() Function

It fetches all result rows as an array

```
mysqli_fetch_all()
```

mysqli_fetch_array() Function

It fetches a result row as an associative, a numeric array, or both

```
mysqli_fetch_array()
```

mysqli_fetch_assoc() Function

It fetches a result row as an associative array

```
mysqli_fetch_assoc()
```

mysqli_fetch_row() Function

It fetches one row from a result set and returns it as an enumerated array

```
mysqli_fetch_row()
```

mysqli_kill() Function

It kills a MySQL thread

```
mysqli_kill()
```

mysqli_close() Function

It closes a database connection

```
mysqli_close()
```

JavaScript Basics

Set of JavaScript basic syntax to add, execute and write basic programming paradigms in Javascript

On Page Script

Adding internal JavaScript to HTML

```
<script type="text/javascript"> //JS code goes here </script>
```

External JS File

Adding external JavaScript to HTML

```
<script src="filename.js"></script>
```

Functions

JavaScript Function syntax

```
function nameOfFunction () {
// function body
}
```

DOM Element

Changing content of a DOM Element

```
document.getElementById("elementID").innerHTML = "Hello World!";
```

Output

This will print the value of a in JavaScript console

```
console.log(a);
```

Conditional Statements

Conditional statements are used to perform operations based on some conditions.

If Statement

The block of code to be executed, when the condition specified is true.

```
if (condition) {
// block of code to be executed if the condition is true
}
```

If-else Statement

If the condition for the if block is false, then the else block will be executed.

```
if (condition) {
// block of code to be executed if the condition is true
} else {
// block of code to be executed if the condition is false
}
```

Else-if Statement

A basic if-else ladder

```
if (condition1) {
// block of code to be executed if condition1 is true
} else if (condition2) {
// block of code to be executed if the condition1 is false and condition2 is
} else {
// block of code to be executed if the condition1 is false and condition2 is
}
```

Switch Statement

Switch case statement in JavaScript

```
switch(expression) {
case x:
// code block
break;
case y:
```

```
// code block
break;
default:
// code block
}
```

Iterative Statements (Loops)

Iterative statement facilitates programmer to execute any block of code lines repeatedly and can be controlled as per conditions added by the programmer.

For Loop

For loop syntax in javascript

```
for (statement 1; statement 2; statement 3) {
// code block to be executed
}
```

While Loop

Runs the code till the specified condition is true

```
while (condition) {
// code block to be executed
}
```

Do While Loop

A do while loop is executed at least once despite the condition being true or false

```
do {
// run this code in block
i++;
} while (condition);
```

Strings

The string is a sequence of characters that is used for storing and managing text data.

charAt method

Returns the character from the specified index.

```
str.charAt(3)
```

concat method

Joins two or more strings together.

```
str1.concat(str2)
```

index of method

Returns the index of the first occurrence of the specified character from the string else -1 if not found.

```
str.indexOf('substr')
```

match method

Searches a string for a match against a regular expression.

```
str.match(/(chapter \d+(\.\d)*)/i;)
```

replace method

Searches a string for a match against a specified string or char and returns a new string by replacing the specified values.

```
str1.replace(str2)
```

search method

Searches a string against a specified value.

```
str.search('term')
```

split method

Splits a string into an array consisting of substrings.

```
str.split('\n')
```

substring method

Returns a substring of a string containing characters from the specified indices.

```
str.substring(0,5)
```

Arrays

The array is a collection of data items of the same type. In simple terms, it is a variable that contains multiple values.

variable

Containers for storing data.

```
var fruit = ["element1", "element2", "element3"];
```

concat method

Joins two or more arrays together.

```
concat()
```

indexOf method

Returns the index of the specified item from the array.

```
indexOf()
```

join method

Converts the array elements to a string.

```
join()
```

pop method

Deletes the last element of the array.

```
pop()
```

reverse method

This method reverses the order of the array elements.

```
reverse()
```

sort method

Sorts the array elements in a specified manner.

```
sort()
```

toString method

Converts the array elements to a string.

```
toString()
```

valueOf method

returns the relevant Number Object holding the value of the argument passed

```
valueOf()
```

Number Methods

JS math and number objects provide several constant and methods to perform mathematical operations.

toExponential method

Converts a number to its exponential form.

```
toExponential()
```

toPrecision method

Formats a number into a specified length.

```
toPrecision()
```

toString method

Converts an object to a string

```
toString()
```

valueOf method

Returns the primitive value of a number.

```
valueOf()
```

Maths Methods

ceil method

Rounds a number upwards to the nearest integer, and returns the result

```
ceil(x)
```

exp method

Returns the value of E^x.

```
exp(x)
```

log method

Returns the logarithmic value of x.

```
log(x)
```

pow method

Returns the value of x to the power y.

```
pow(x,y)
```

random method

Returns a random number between 0 and 1.

```
random()
```

sqrt method

Returns the square root of a number x

```
sqrt(x)
```

Dates

Date object is used to get the year, month and day. It has methods to get and set day, month, year, hour, minute, and seconds.

Pulling Date from the Date object

Returns the date from the date object

```
getDate()
```

Pulling Day from the Date object

Returns the day from the date object

```
getDay()
```

Pulling Hours from the Date object

Returns the hours from the date object

```
getHours()
```

Pulling Minutes from the Date object

Returns the minutes from the date object

```
getMinutes()
```

Pulling Seconds from the Date object

Returns the seconds from the date object

```
getSeconds()
```

Pulling Time from the Date object

Returns the time from the date object

```
getTime()
```

Mouse Events

Any change in the state of an object is referred to as an Event. With the help of JS, you can handle events, i.e., how any specific HTML tag will work when the user does something.

click

Fired when an element is clicked

```
element.addEventListener('click', ()=>{
// Code to be executed when the event is fired
});
```

oncontextmenu

Fired when an element is right-clicked

```
element.addEventListener('contextmenu', ()=>{
// Code to be executed when the event is fired
});
```

dblclick

Fired when an element is double-clicked

```
element.addEventListener('dblclick', ()=>{
// Code to be executed when the event is fired
});
```

mouseenter

Fired when an element is entered by the mouse arrow

```
element.addEventListener('mouseenter', ()=>{
// Code to be executed when the event is fired
});
```

mouseleave

Fired when an element is exited by the mouse arrow

```
element.addEventListener('mouseleave', ()=>{
// Code to be executed when the event is fired
});
```

mousemove

Fired when the mouse is moved inside the element

```
element.addEventListener('mousemove', ()=>{
// Code to be executed when the event is fired
});
```

Keyboard Events

keydown

Fired when the user is pressing a key on the keyboard

```
element.addEventListener('keydown', ()=>{
// Code to be executed when the event is fired
});
```

keypress

Fired when the user presses the key on the keyboard

```
element.addEventListener('keypress', ()=>{
// Code to be executed when the event is fired
});
```

keyup

Fired when the user releases a key on the keyboard

```
element.addEventListener('keyup', ()=>{
// Code to be executed when the event is fired
});
```

Errors

Errors are thrown by the compiler or interpreter whenever they find any fault in the code, and it can be of any type like syntax error, run-time error, logical error, etc. JS provides some functions to handle the errors.

try and catch

Try the code block and execute catch when err is thrown

```
try {
Block of code to try
}
catch(err) {
Block of code to handle errors
}
```

Window Methods

Methods that are available from the window object

alert method

Used to alert something on the screen

```
alert()
```

blur method

The blur() method removes focus from the current window.

```
blur()
```

setInterval

Keeps executing code at a certain interval

```
setInterval(() => {
// Code to be executed
}, 1000);
```

setTimeout

Executes the code after a certain interval of time

```
setTimeout(() => {
// Code to be executed
}, 1000);
```

close

The Window. close() method closes the current window

```
window.close()
```

confirm

The window.confirm() instructs the browser to display a dialog with an optional message, and to wait until the user either confirms or cancels

```
window.confirm('Are you sure?')
```

open

Opens a new window

```
window.open("https://www.codewithharry.com");
```

prompt

Prompts the user with a text and takes a value. Second parameter is the default value

```
var name = prompt("What is your name?", "Harry");
```

scrollBy

```
window.scrollBy(100, 0); // Scroll 100px to the right
```

scrollTo

Scrolls the document to the specified coordinates.

```
window.scrollTo(500, 0); // Scroll to horizontal position 500
```

clearInterval

Clears the setInterval. var is the value returned by setInterval call

```
clearInterval(var)
```

clearTimeout

Clears the setTimeout. var is the value returned by setTimeout call

```
clearTimeout(var)
```

stop

Stops the further resource loading

```
stop()
```

Query/Get Elements

The browser creates a DOM (Document Object Model) whenever a web page is loaded, and with the help of HTML DOM, one can access and modify all the elements of the HTML document.

querySelector

Selector to select first matching element

```
document.querySelector('css-selectors')
```

querySelectorAll

A selector to select all matching elements

```
document.querySelectorAll('css-selectors', ...)
```

getElementsByTagName

Select elements by tag name

```
document.getElementsByTagName('element-name')
```

getElementsByClassName

Select elements by class name

```
document.getElementsByClassName('class-name')
```

Get Element by Id

Select an element by its id

```
document.getElementById('id')
```

Creating Elements

Create new elements in the DOM

createElement

Create a new element

```
document.createElement('div')
```

createTextNode

Create a new text node

```
document.createTextNode('some text here')
```

Importing Flask

```
from flask import Flask
```

Most used import functions

These are some of the most used import functions

```
from flask import Flask, render_template, redirect, url_for, request
```

Boilerplate

This is the basic template or barebone structure of Flask.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
return "<p>Hello, World!</p>"

app.run()
```

route(endpoint)

This is to make different endpoints in our flask app.

```
@app.route("/")
```

Route method

Allowing get and post requests on an endpoint.

```
methods = ['GET', 'POST']
```

Re-run while coding

This is used to automatically rerun the program when the file is saved.

```
app.run(debug=True)
```

Change host

This is used to change the host.

```
app.run(host='0.0.0.0')
```

Change port

This is used to change the port.

```
app.run(port=80)
```

SQLAlchemy

```
from flask_sqlalchemy import SQLAlchemy
```

Database URI

This is the database's address.

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://username:password@localhost
or
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db'
```

Initialization

This is used to initialize SQLAlchemy.

```
db = SQLAlchemy(app)
```

Creating Model

Class to get data from database and to send data to the database.

```
class TableName(db.Model):
column_1 = db.Column(db.Integer, primary_key=True)
```

```
column_2 = db.Column(db.String(80), nullable=False)
column_3 = db.Column(db.String(12), nullable=False)
```

Get all data(.all())

This is used to get all the data from the database.

```
data = ClassName.query.filter_by().all()
```

Filtered data(.first())

This is used to get the first dataset from the list returned by the filter_by function. You can get targetted data by this.

```
data = ClassName.query.filter_by().first()
```

Send/add data to database

This is used to send/add data to the database.

```
data_to_send = ClassName(column_1=dataset1, column_2=dataset2, column_3=data
db.session.add(data_to_send)
db.session.commit()
```

Delete data from the database

This is used to delete data from the database.

```
data_to_send = ClassName(column_1=dataset1, column_2=dataset2, column_3=data
db.session.delete(data_to_send)
db.session.commit()
```

Request method

This is used to know what request is made(get/post).

```
request.method
```

Render Template

This is used to pass whole html file directly.

```
render_template("file.html")
```

FSADeprecationWarning

```
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = True|False
```

Creating Database file

This is used to create a database file

```
from yourapplicationname import db
db.create_all()
exit()
```

Method to return database items

This is used to return database items.

```
def __repr__(self) -> str:
return f"{self.item}"
```

Printing returned content from the method

This is used to print returned database items.

```
data = ClassNameWithMethod.query.all()
print(data)
```

Flask Documentation

```
https://flask.palletsprojects.com/en/latest/
```

Flask SQLAlchemy Documentation

```
https://flask-sqlalchemy.palletsprojects.com/en/2.x/
```

Font

There are many properties related to the font, such as the face, weight, style, etc. These properties allow you to change the style or complete look of your text.

Font-Family

```
font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
```

Font-Style

```
font-style: italic;
```

Font-Variant

```
font-variant: small-caps;
```

Font-Weight

```
font-weight: bold;
```

Font-Size

```
font-size: larger;
```

Font

```
font: style variant weight size family;
```

Text

Text properties allow one to manipulate alignment, spacing, decoration, indentation, etc., in the document.

Text-Align

```
text-align: justify;
```

Letter-Spacing

```
letter-spacing: .15em;
```

Text-Decoration

```
text-decoration: underline;
```

Word-Spacing

```
word-spacing: 0.25em;
```

Text-Transform

```
text-transform: uppercase;
```

Text-Indent

```
text-indent: 0.5cm;
```

Line-Height

```
line-height: normal;
```

Background

As the name suggests, these properties are related to background, i.e., you can change the color, image, position, size, etc., of the document.

Background-Image

```
background-image: url("Path");
```

Background-Position

```
background-position: right top;
```

Background-Size

```
background-size: cover;
```

Background-Repeat

```
background-repeat: no-repeat;
```

Background-Attachment

```
background-attachment: scroll;
```

Background-Color

```
background-color: fuchsia;
```

Background

```
background: color image repeat attachment position;
```

Border

Border properties are used to change the style, radius, color, etc., of buttons or other items of the document.

Border-Width

```
border-width: 5px;
```

Border-Style

```
border-style: solid;
```

Border-Color

```
border-color: aqua;
```

Border-Radius

```
border-radius: 15px;
```

Border

```
border: width style color;
```

Box Model

In laymen's terms, the CSS box model is a container that wraps around every HTML element. It consists of margins, borders, padding, and the actual content. It is used to create the design and layout of web pages.

Float

```
float: none;
```

Clear

```
clear: both;
```

Display

```
display: block;
```

Height

```
height: fit-content;
```

Width

```
width: auto;
```

Margin

```
margin: top right bottom left;
```

Padding

```
padding: top right bottom left;
```

Overflow

```
overflow: hidden;
```

Visibility

```
visibility: visible;
```

Colors

With the help of the color property, one can give color to text, shape, or any other object.

Color

```
color: cornsilk;
```

Opacity

```
opacity: 4;
```

Template Layout

Specifies the visual look of the content inside a template

Box-Align

```
box-align : start;
```

Box-Direction

```
box-direction : normal;
```

Box-Flex

```
box-flex : normal;
```

Box-Flex-Group

```
box-flex-group : 2;
```

Box-Orient

```
box-orient : inline;
```

Box-Pack

```
box-pack : justify;
```

Box-Sizing

```
box-sizing : margin-box;
```

max-width

```
max-width: 800px;
```

min-width

```
min-width: 500px;
```

max-height

```
max-height: 100px;
```

min-height

```
min-height: 80px;
```

Table

Table properties are used to give style to the tables in the document. You can change many things like border spacing, table layout, caption, etc.

Border-Collapse

```
border-collapse: separate;
```

Empty-Cells

```
empty-cells: show;
```

Border-Spacing

```
border-spacing: 2px;
```

Table-Layout

```
table-layout: auto;
```

Caption-Side

```
caption-side: bottom;
```

Columns

These properties are used explicitly with columns of the tables, and they are used to give the table an incredible look.

Column-Count

```
column-count : 10;
```

Column-Gap

```
column-gap : 5px;
```

Column-rule-width

```
column-rule-width : medium;
```

Column-rule-style

```
column-rule-style : dotted ;
```

Column-rule-color

```
column-rule-color : black;
```

Column-width

```
column-width : 10px;
```

Column-span

```
column-span : all;
```

List & Markers

List and marker properties are used to customize lists in the document.

List-style-type

```
list-style-type: square;
```

List-style-position

```
list-style-position : 20px;
```

List-style-image

```
list-style-image : url(image.gif);
```

Marker-offset

```
marker-offset : auto;
```

Animations

CSS animations allow one to animate transitions or other media files on the web page.

Animation-name

```
animation-name : myanimation;
```

Animation-duration

```
animation-duration : 10s;
```

Animation-timing-function

```
animation-timing-function : ease;
```

Animation-delay

```
animation-delay : 5ms;
```

Animation-iteration-count

```
animation-iteration-count : 3;
```

Animation-direction

```
animation-direction : normal;
```

Animation-play-state

```
animation-play-state : running;
```

Animation-fill-mode

```
animation-fill-mode : both;
```

Transitions

Transitions let you define the transition between two states of an element.

Transition-property

```
transition-property: none;
```

Transition-duration

```
transition-duration : 2s;
```

Transition-timing-function

```
transition-timing-function: ease-in-out;
```

Transition-delay

```
transition-delay : 20ms;
```

CSS Flexbox

Flexbox is a layout of CSS that lets you format HTML easily. Flexbox makes it simple to align items vertically and horizontally using rows and columns. Items will "flex" to different sizes to fill the space. And overall, it makes the responsive design more manageable.

Parent Properties (flex container)

display

```
display: flex;
```

flex-direction

```
flex-direction: row | row-reverse | column | column-reverse;
```

flex-wrap

```
flex-wrap: nowrap | wrap | wrap-reverse;
```

flex-flow

```
flex-flow: column wrap;
```

justify-content

```
justify-content: flex-start | flex-end | center | space-between | space-arou
```

align-items

```
align-items: stretch | flex-start | flex-end | center | baseline | first bas
```

align-content

```
align-content: flex-start | flex-end | center | space-between | space-around
```

Child Properties (flex items)

order

```
order: 5; /* default is 0 */
```

flex-grow

```
flex-grow: 4; /* default 0 */
```

flex-shrink

```
flex-shrink: 3; /* default 1 */
```

flex-basis

```
flex-basis: | auto; /* default auto */
```

flex shorthand

```
flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
```

align-self

```
align-self: auto | flex-start | flex-end | center | baseline | stretch;
```

CSS Grid

Grid layout is a 2-Dimensional grid system to CSS that creates complex responsive web design layouts more easily and consistently across browsers.

Parent Properties (Grid container)

display

```
display: grid | inline-grid;
```

grid-template-columns

```
grid-template-columns: 12px 12px 12px;
```

grid-template-rows

```
grid-template-rows: 8px auto 12px;
```

grid-template

```
grid-template: none | <grid-template-rows> / <grid-template-columns>;
```

column-gap

```
column-gap: <line-size>;
```

row-gap

```
row-gap: <line-size>;
```

grid-column-gap

```
grid-column-gap: <line-size>;
```

grid-row-gap

```
grid-row-gap: <line-size>;
```

gap shorthand

```
gap: <grid-row-gap> <grid-column-gap>;
```

grid-gap shorthand

```
grid-gap: <grid-row-gap> <grid-column-gap>;
```

justify-items

```
justify-items: start | end | center | stretch;
```

align-items

```
align-items: start | end | center | stretch;
```

place-items

```
place-items: center;
```

justify-content

```
justify-content: start | end | center | stretch | space-around | space-betwe
```

align-content

```
align-content: start | end | center | stretch | space-around | space-between
```

place-content

```
place-content: <align-content> / <justify-content> ;
```

grid-auto-columns

```
grid-auto-columns: <track-size> ...;
```

grid-auto-rows

```
grid-auto-rows: <track-size> ...;
```

grid-auto-flow

```
grid-auto-flow: row | column | row dense | column dense;
```

Child Properties (Grid items)

grid-column-start

```
grid-column-start: <number> | <name> | span <number> | span <name> | auto;
```

grid-column-end

```
grid-column-end: <number> | <name> | span <number> | span <name> | auto;
```

grid-row-start

```
grid-row-start: <number> | <name> | span <number> | span <name> | auto;
```

grid-row-end

```
grid-row-end: <number> | <name> | span <number> | span <name> | auto;
```

grid-column shorthand

```
grid-column: <start-line> / <end-line> | <start-line> / span <value>;
```

grid-row shorthand

```
grid-row: <start-line> / <end-line> | <start-line> / span <value>;
```

grid-area

```
grid-area: <name> | <row-start> / <column-start> / <row-end> / <column-end>;
```

justify-self

```
justify-self: start | end | center | stretch;
```

align-self

```
align-self: start | end | center | stretch;
```

place-self

```
place-self: center;
```

Structure

This is the basic template or barebone structure of HTML.

Boilerplate

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Document</title>
</head>
<body>

</body>
</html>
```

Headings

There are six headings available in HTML, H1 is the largest among all, and H6 is the smallest.

<h1> Tag

```
<h1>Heading 1</h1>
```

<h2> Tag

```
<h2>Heading 2</h2>
```

<h3> Tag

```
<h3>Heading 3</h3>
```

h4 Tag

```
<h4>Heading 4</h4>
```

h5 Tag

```
<h5>Heading 5</h5>
```

h6 Tag

```
<h6>Heading 6</h6>
```

Container

Container tags are the tags that contain some data such as text, image, etc. There are several container tags in HTML.

div tag

div tag or division tag is used to make blocks or divisions in the document.

```
<div> This is div block </div>
```

span tag

span is a container for inline content

```
<span> This is span block </span>
```

p tag

Paragraph

```
<p> This is a paragraph </p>
```

pre tag

pre tag represents pre-formatted text

```
<pre> Hello World </pre>
```

code tag

code tag is used to represent source codes

```
<code>
import python
</code>
```

Text Formatting

Text formatting tags are used to format text or data of HTML documents. You can do certain things like creating italic, bold, strong text to make your document look more attractive and understandable.

<b> tag

```
<b>I'm bold text</b>
```

tag

```
<strong>I'm important text</strong>
```

<i> tag

```
<i>I'm italic text</i>
```

tag

```
<em>Emphasized text</em>
```

<sub> tag

```
<sub>Subscript</sub>
```

<sup> tag

```
<sup>Superscript</sup>
```

Lists

Lists can be either numerical, alphabetic, bullet, or other symbols. You can specify list type and list items in HTML for the clean document.

 tag

Ordered list starts with <ol> tag and each list item starts with <li> tag

```
<ol>
<li>Data 1</li>
<li>Data 2</li>
<li>Data 3</li>
</ol>
```

 tag

```
<ul>
<li>Your Data</li>
<li>Your Data</li>
</ul>
```

Media

Media is anything that is present in digital form such as image, video, audio, etc.

<audio> tag

It is used to embed sound content in the document.

```
<audio controls>
<source src="demo.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

<img> tag

It is used to embed or import image in a webpage.

```
<img src="Source_of_image" alt="Alternate text">
```

<video> tag

It is used to embed video in the webpage.

```
<video width="480" height="320" controls>
<source src="demo_move.mp4" type="video/mp4">
Your browser does not support the video tag.
</video>
```

Table

A table is a collection of rows and columns. It is used to represent data in tabular form.

Table Structure

```
<table>
<caption>Demo Table</caption>
<thead>
<tr>
<th>Column1</th>
<th colspan="2">Column2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data1</td>
<td>Data2</td>
<td>Data2</td>
</tr>
<tr>
<td>Data1</td>
<td>Data2</td>
<td>Data2</td>
</tr>
</tbody>
<tfoot>
<tr>
<td> </td>
<td>Data</td>
<td>Data</td>
</tr>
```

```
</tfoot>
</table>
```

Links

Links are clickable text that can redirect you to some other page.

<a> tag

<a> or anchor tag defines a hyperlink.

```
<a href="https://www.codewithharry.com/">Visit CodeWithHarry.com!</a>
```

Form

Sample Form

Form is used to collect user's input, generally user's data is sent to server for further processing.

```
<form action="/action.php" method="post">
Name: <input name="name" type="text" /> <br />
Age: <input max="90" min="1" name="age" step="1" type="number" value="18" />
<select name="gender">
<option selected="selected" value="male">Male</option>
<option value="female">Female</option>
</select><br />
<input checked="checked" name="newsletter" type="radio" value="daily" /> Dai
type="radio" value="weekly" /> Weekly<br />
<textarea cols="20" name="comments" rows="5">Comment</textarea><br />
<label><input name="terms" type="checkbox" value="tandc" />Accept terms</lab
<input type="submit" value="Submit" />
</form>
```

Characters & Symbols

Some symbols are not directly present on the keyboard, but there are some ways to use them in HTML documents. We can display them either by entity name, decimal, or hexadecimal value.

Copyright Symbol (©)

```
&copy;
```

Less than (<)

```
&lt
```

Greater than (>)

```
&gt;
```

Ampersand (&)

```
&amp;
```

Dollar ($)

```
&dollar;
```

Random Text

Elon Musk

```
Elon Reeve Musk FRS is an entrepreneur and business magnate. He is the found
```

Semantic Elements

Semantic elements are those elements that are self describable, i.e., from their name itself, you can understand their meaning.

<section> tag

It defines a section in the document

```
<section>This is a section</section>
```

<article> tag

It represents self-contained content

```
<article> Enter your data here </article>
```

<aside> tag

It is used to place content in the sidebar

```
<aside> Your data </aside>
```

Basics

Basic syntax and functions from the Java programming language.

Boilerplate

```
class HelloWorld{
public static void main(String args[]){
System.out.println("Hello World");
}
}
```

Showing Output

It will print something to the output console.

```
System.out.println([text])
```

Taking Input

It will take string input from the user

```
import java.util.Scanner; //import scanner class

// create an object of Scanner class
Scanner input = new Scanner(System.in);

// take input from the user
String varName = input.nextLine();
```

Primitive Type Variables

The eight primitives defined in Java are int, byte, short, long, float, double, boolean, and char those aren't considered objects and represent raw values.

byte

byte is a primitive data type it only takes up 8 bits of memory.

```
age = 18;
```

long

long is another primitive data type related to integers. long takes up 64 bits of memory.

```
viewsCount = 3_123_456L;
```

float

We represent basic fractional numbers in Java using the float type. This is a single-precision decimal number. Which means if we get past six decimal points, this number becomes less precise and more of an estimate.

```
price = 100INR;
```

char

Char is a 16-bit integer representing a Unicode-encoded character.

```
letter = 'A';
```

boolean

The simplest primitive data type is boolean. It can contain only two values: true or false. It stores its value in a single bit.

```
isEligible = true;
```

int

int holds a wide range of non-fractional number values.

```
var1 = 256;
```

short

If we want to save memory and byte is too small, we can use short.

```
short var2 = 786;
```

Comments

A comment is the code that is not executed by the compiler, and the programmer uses it to keep track of the code.

Single line comment

```
// It's a single line comment
```

Multi-line comment

```
/* It's a
multi-line
comment
*/
```

Constants

Constants are like a variable, except that their value never changes during program execution.

```
final float INTEREST_RATE = 0.04;
```

Arithmetic Expressions

These are the collection of literals and arithmetic operators.

Addition

It can be used to add two numbers

```
int x = 10 + 3;
```

Subtraction

It can be used to subtract two numbers

```
int x = 10 - 3;
```

Multiplication

It can be used to multiply add two numbers

```
int x = 10 * 3;
```

Division

It can be used to divide two numbers

```
int x = 10 / 3;
float x = (float)10 / (float)3;
```

Modulo Remainder

It returns the remainder of the two numbers after division

```
int x = 10 % 3;
```

Augmented Operators

Addition assignment

```
var += 10 // var = var + 10
```

Subtraction assignment

```
var -= 10 // var = var - 10
```

Multiplication assignment

```
var *= 10 // var = var * 10
```

Division assignment

```
var /= 10 // var = var / 10
```

Modulus assignment

```
var %= 10 // var = var % 10
```

Escape Sequences

It is a sequence of characters starting with a backslash, and it doesn't represent itself when used inside string literal.

Tab

It gives a tab space

```
\t
```

Backslash

It adds a backslash

```
\\
```

Single quote

It adds a single quotation mark

```
\'
```

Question mark

It adds a question mark

```
\?
```

Carriage return

Inserts a carriage return in the text at this point.

```
\r
```

Double quote

It adds a double quotation mark

```
\"
```

Type Casting

Type Casting is a process of converting one data type into another

Widening Type Casting

It means converting a lower data type into a higher

```
// int x = 45;
double var_name = x;
```

Narrowing Type Casting

It means converting a higher data type into a lower

```
double x = 165.48
int var_name = (int)x;
```

Decision Control Statements

Conditional statements are used to perform operations based on some condition.

if Statement

```
if (condition) {
// block of code to be executed if the condition is true
}
```

if-else Statement

```
if (condition) {
// If condition is True then this block will get executed
} else {
// If condition is False then this block will get executed
}
```

if else-if Statement

```
if (condition1) {
// Codes
}
else if(condition2) {
// Codes
}
else if (condition3) {
// Codes
}
else {
// Codes
}
```

Ternary Operator

It is shorthand of an if-else statement.

```
variable = (condition) ? expressionTrue : expressionFalse;
```

Switch Statements

It allows a variable to be tested for equality against a list of values (cases).

```
switch(expression) {
case a:
// code block
break;
case b:
// code block
break;
default:
// code block
}
```

Iterative Statements

Iterative statements facilitate programmers to execute any block of code lines repeatedly and can be controlled as per conditions added by the coder.

while Loop

It iterates the block of code as long as a specified condition is True

```
while (condition) {
// code block
}
```

for Loop

for loop is used to run a block of code several times

```
for (initialization; termination; increment) {
statement(s)
}
```

for-each Loop

```
for(dataType item : array) {
...
}
```

do-while Loop

It is an exit controlled loop. It is very similar to the while loop with one difference, i.e., the body of the do-while loop is executed at least once even if the condition is False

```
do {
// body of loop
} while(textExpression)
```

Break statement

break keyword inside the loop is used to terminate the loop

```
break;
```

Continue statement

continue keyword skips the rest of the current iteration of the loop and returns to the starting point of the loop

```
continue;
```

Arrays

Arrays are used to store multiple values in a single variable

Declaring an array

Declaration of an array

```
String[] var_name;
```

Defining an array

Defining an array

```
String[] var_name = {''Harry", "Rohan", "Aakash"};
```

Accessing an array

Accessing the elements of an array

```
String[] var_name = {''Harry", "Rohan", "Aakash"};
System.out.println(var_name[index]);
```

Changing an element

Changing any element in an array

```
String[] var_name = {''Harry", "Rohan", "Aakash"};
var_name[2] = "Shubham";
```

Array length

It gives the length of the array

```
System.out.println(var_name.length);
```

Loop through an array

It allows us to iterate through each array element

```
String[] var_name = {''Harry", "Rohan", "Aakash"};
for (int i = 0; i < var_name.length; i++) {
System.out.println(var_name[i]);
}
```

Multi-dimensional Arrays

Arrays can be 1-D, 2-D or multi-dimensional.

```
// Creating a 2x3 array (two rows, three columns)
int[2][3] matrix = new int[2][3];
matrix[0][0] = 10;
// Shortcut
int[2][3] matrix = {
{ 1, 2, 3 },
{ 4, 5, 6 }
};
```

Methods

Methods are used to divide an extensive program into smaller pieces. It can be called multiple times to provide reusability to the program.

Declaration

Declaration of a method

```
returnType methodName(parameters) {
//statements
}
```

Calling a method

Calling a method

```
methodName(arguments);
```

Method Overloading

Method overloading means having multiple methods with the same name, but different parameters.

```
class Calculate
{
void sum (int x, int y)
{
System.out.println("Sum is: "+(a+b)) ;
}
void sum (float x, float y)
{
System.out.println("Sum is: "+(a+b));
}
Public static void main (String[] args)
{
Calculate calc = new Calculate();
calc.sum (5,4); //sum(int x, int y) is method is called.
calc.sum (1.2f, 5.6f); //sum(float x, float y) is called.
}
}
```

Recursion

Recursion is when a function calls a copy of itself to work on a minor problem. And the function that calls itself is known as the Recursive function.

```
void recurse()
{
... .. ...
recurse();
... .. ...
}
```

Strings

It is a collection of characters surrounded by double quotes.

Creating String Variable

```
String var_name = "Hello World";
```

String Length

Returns the length of the string

```
String var_name = "Harry";
System.out.println("The length of the string is: " + var_name.length());
```

String Methods toUpperCase()

Convert the string into uppercase

```
String var_name = "Harry";
System.out.println(var_name.toUpperCase());
```

toLowerCase()

Convert the string into lowercase

```
String var_name = ""Harry"";
System.out.println(var_name.toLowerCase());
```

indexOf()

Returns the index of specified character from the string

```
String var_name = "Harry";
System.out.println(var_name.indexOf("a"));
```

concat()

Used to concatenate two strings

```
String var1 = "Harry";
String var2 = "Bhai";
System.out.println(var1.concat(var2));
```

Math Class

Math class allows you to perform mathematical operations.

Methods max() method

It is used to find the greater number among the two

```
Math.max(25, 45);
```

min() method

It is used to find the smaller number among the two

```
Math.min(8, 7);
```

sqrt() method

It returns the square root of the supplied value

```
Math.sqrt(144);
```

random() method

It is used to generate random numbers

```
Math.random(); //It will produce random number b/w 0.0 and 1.0

int random_num = (int)(Math.random() * 101); //Random num b/w 0 and 100
```

Object-Oriented Programming

It is a programming approach that primarily focuses on using objects and classes. The objects can be any real-world entities.

object

An object is an instance of a Class.

```
className object = new className();
```

class

A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

```
class ClassName {
// Fields
// Methods
// Constructors
// Blocks
}
```

Encapsulation

Encapsulation is a mechanism of wrapping the data and code acting on the data together as a single unit. In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class.

```
public class Person {
private String name; // using private access modifier

// Getter
public String getName() {
return name;
}

// Setter
public void setName(String newName) {
this.name = newName;
}
}
```

Inheritance

Inheritance can be defined as the process where one class acquires the properties of another. With the use of inheritance the information is made manageable in a hierarchical order.

```
class Subclass-name extends Superclass-name
{
//methods and fields
}
```

Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

```
// A class with multiple methods with the same name
public class Adder {
// method 1
public void add(int a, int b) {
System.out.println(a + b);
}

// method 2
public void add(int a, int b, int c) {
System.out.println(a + b + c);
}

// method 3
public void add(String a, String b) {
System.out.println(a + " + " + b);
}
}

// My main class
class MyMainClass {
public static void main(String[] args) {
Adder adder = new Adder(); // create a Adder object
adder.add(5, 4); // invoke method 1
adder.add(5, 4, 3); // invoke method 2
adder.add("5", "4"); // invoke method 3
}
}
```

File Operations

File handling refers to reading or writing data from files. Java provides some functions that allow us to manipulate data in the files.

canRead method

Checks whether the file is readable or not

```
file.canRead()
```

createNewFile method

It creates an empty file

```
file.createNewFile()
```

canWrite method

Checks whether the file is writable or not

```
file.canWrite()
```

exists method

Checks whether the file exists

```
file.exists()
```

delete method

It deletes a file

```
file.delete()
```

getName method

It returns the name of the file

```
file.getName()
```

getAbsolutePath method

It returns the absolute pathname of the file

```
file.getAbsolutePath()
```

length Method

It returns the size of the file in bytes

```
file.length()
```

list Method

It returns an array of the files in the directory

```
file.list()
```

mkdir method

It is used to create a new directory

```
file.mkdir()
```

close method

It is used to close the file

```
file.close()
```

To write something in the file

```
import java.io.FileWriter; // Import the FileWriter class
import java.io.IOException; // Import the IOException class to handle errors

public class WriteToFile {
public static void main(String[] args) {
try {
FileWriter myWriter = new FileWriter("filename.txt");
myWriter.write("Laal Phool Neela Phool, Harry Bhaiya Beautiful");
myWriter.close();
System.out.println("Successfully wrote to the file.");
} catch (IOException e) {
System.out.println("An error occurred.");
e.printStackTrace();
}
}
```

```
}
```

Exception Handling

An exception is an unusual condition that results in an interruption in the flow of the program.

try-catch block

try statement allow you to define a block of code to be tested for errors. catch block is used to handle the exception.

```
try {
// Statements
}
catch(Exception e) {
// Statements
}
```

finally block

finally code is executed whether an exception is handled or not.

```
try {
//Statements
}
catch (ExceptionType1 e1) {
// catch block
}
finally {
// finally block always executes
}
```

Basics

Basic syntax and functions from the C++ programming language.

Boilerplate

```
#include <iostream>
using namespace std;

int main() {
cout << "Welcome To CodeWithHarry";
return 0;
}
```

cout <<

It prints output on the screen

```
cout << "This is C++ Programming";
```

cin >>

It takes input from the user

```
cin >> variable_name
```

Data types

The data type is the type of data

Character type

Typically a single octet(one byte). It is an integer type

```
char variable_name;
```

Integer type

The most natural size of integer for the machine

```
int variable_name;
```

Float type

A single-precision floating-point value

```
float variable_name;
```

Double type

A double-precision floating-point value

```
double variable_name;
```

Void type

Represents the absence of the type

```
void
```

Boolean type

```
bool
```

Escape Sequences

It is a sequence of characters starting with a backslash, and it doesn't represent itself when used inside string literal.

Alarm or Beep

It produces a beep sound

```
\a
```

Backspace

It adds a backspace

```
\b
```

Form feed

```
\f
```

Newline

Newline Character

```
\n
```

Carriage return

```
\r
```

Tab

It gives a tab space

```
\t
```

Backslash

It adds a backslash

```
\\
```

Single quote

It adds a single quotation mark

```
\'
```

Question mark

It adds a question mark

```
\?
```

Octal No.

It represents the value of an octal number

```
\nnn
```

Hexadecimal No.

It represents the value of a hexadecimal number

```
\xhh
```

Null

The null character is usually used to terminate a string

```
\0
```

Comments

A comment is a code that is not executed by the compiler, and the programmer uses it to keep track of the code.

Single line comment

```
// It's a single line comment
```

Multi-line comment

```
/* It's a
multi-line
comment
*/
```

Strings

It is a collection of characters surrounded by double quotes

Declaring String

```
// Include the string library
#include <string>

// String variable
string variable1 = "Hello World";
```

append function

It is used to concatenate two strings

```
string firstName = "Harry ";
string lastName = "Bhai";
string fullName = firstName.append(lastName);
cout << fullName;
```

length function

It returns the length of the string

```
string variable1 = "CodeWithHarry";
cout << "The length of the string is: " << variable1.length();
```

Accessing and changing string characters

```
string variable1 = "Hello World";
variable1[1] = 'i';
cout << variable1;
```

Maths

C++ provides some built-in math functions that help the programmer to perform mathematical operations efficiently.

max function

It returns the larger value among the two

```
cout << max(25, 140);
```

min function

It returns the smaller value among the two

```
cout << min(55, 50);
```

sqrt function

It returns the square root of a supplied number

```
#include <cmath>

cout << sqrt(144);
```

ceil function

It returns the value of x rounded up to its nearest integer

```
ceil(x)
```

floor function

It returns the value of x rounded down to its nearest integer

```
floor(x)
```

pow function

It returns the value of x to the power of y

```
pow(x, y)
```

Decision Making Instructions

Conditional statements are used to perform operations based on some condition.

If Statement

```
if (condition) {
// This block of code will get executed, if the condition is True
}
```

If-else Statement

```
if (condition) {
// If condition is True then this block will get executed
} else {
// If condition is False then this block will get executed
}
```

if else-if Statement

```
if (condition) {
// Statements;
}
else if (condition){
// Statements;
}
else{
// Statements
}
```

Ternary Operator

It is shorthand of an if-else statement.

```
variable = (condition) ? expressionTrue : expressionFalse;
```

Switch Case Statement

It allows a variable to be tested for equality against a list of values (cases).

```
switch (expression)
{
case constant-expression:
statement1;
statement2;
```

```
break;
case constant-expression:
statement;
break;
...
default:
statement;
}
```

Iterative Statements

Iterative statements facilitate programmers to execute any block of code lines repeatedly and can be controlled as per conditions added by the programmer.

while Loop

It iterates the block of code as long as a specified condition is True

```
while (/* condition */)
{
/* code block to be executed */
}
```

do-while loop

It is an exit controlled loop. It is very similar to the while loop with one difference, i.e., the body of the do-while loop is executed at least once even if the condition is False

```
do
{
/* code */
} while (/* condition */);
```

for loop

It is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

```
for (int i = 0; i < count; i++)
{
/* code */
}
```

Break Statement

break keyword inside the loop is used to terminate the loop

```
break;
```

Continue Statement

continue keyword skips the rest of the current iteration of the loop and returns to the starting point of the loop

```
continue;
```

References

Reference is an alias for an already existing variable. Once it is initialized to a variable, it cannot be changed to refer to another variable. So, it's a const pointer.

Creating References

```
string var1 = "Value1"; // var1 variable
string &var2 = var1; // reference to var1
```

Pointers

Pointer is a variable that holds the memory address of another variable

Declaration

```
datatype *var_name;

var_name = &variable2;
```

Functions & Recursion

Functions are used to divide an extensive program into smaller pieces. It can be called multiple times to provide reusability and modularity to the C program.

Function Definition

```
return_type function_name(data_type parameter...){
//code to be executed
}
```

Function Call

```
function_name(arguments);
```

Recursion

Recursion is when a function calls a copy of itself to work on a minor problem. And the function that calls itself is known as the Recursive function.

```
void recurse()
{
... .. ...
recurse();
... .. ...
}
```

Object-Oriented Programming

It is a programming approach that primarily focuses on using objects and classes. The objects can be any real-world entities.

class

```
class Class_name {
public: // Access specifier
// fields
// functions
// blocks
};
```

object

```
Class_name ObjectName;
```

Constructors

It is a special method that is called automatically as soon as the object is created.

```
class className { // The class
public: // Access specifier
className() { // Constructor
cout << "Code With Harry";
}
};

int main() {
className obj_name;
return 0;
}
```

Encapsulation

Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

```
#include<iostream>
using namespace std;
class ExampleEncap{
private:
/* Since we have marked these data members private,
* any entity outside this class cannot access these
* data members directly, they have to use getter and
* setter functions.
*/
int num;
char ch;
public:
/* Getter functions to get the value of data members.
* Since these functions are public, they can be accessed
* outside the class, thus provide the access to data members
* through them
*/
int getNum() const {
return num;
}
char getCh() const {
return ch;
```

```
}
/* Setter functions, they are called for assigning the values
* to the private data members.
*/
void setNum(int num) {
this->num = num;
}
void setCh(char ch) {
this->ch = ch;
}
};
int main(){
ExampleEncap obj;
obj.setNum(100);
obj.setCh('A');
cout<<obj.getNum()<<endl;
cout<<obj.getCh()<<endl;
return 0;
}
```

File Handling

File handling refers to reading or writing data from files. C provides some functions that allow us to manipulate data in the files.

Creating and writing to a text file

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
// Create and open a text file
ofstream MyFile("filename.txt");

// Write to the file
MyFile << "File Handling in C++";

// Close the file
MyFile.close();
}
```

Reading the file

It allows us to read the file line by line

```
getline()
```

Opening a File

It opens a file in the C++ program

```
void open(const char* file_name,ios::openmode mode);
```

OPEN MODES

in

Opens the file to read(default for ifstream)

```
fs.open ("test.txt", std::fstream::in)
```

out

Opens the file to write(default for ofstream)

```
fs.open ("test.txt", std::fstream::out)
```

binary

Opens the file in binary mode

```
fs.open ("test.txt", std::fstream::binary)
```

app

Opens the file and appends all the outputs at the end

```
fs.open ("test.txt", std::fstream::app)
```

ate

Opens the file and moves the control to the end of the file

```
fs.open ("test.txt", std::fstream::ate)
```

trunc

Removes the data in the existing file

```
fs.open ("test.txt", std::fstream::trunc)
```

nocreate

Opens the file only if it already exists

```
fs.open ("test.txt", std::fstream::nocreate)
```

noreplace

Opens the file only if it does not already exist

```
fs.open ("test.txt", std::fstream::noreplace)
```

Closing a file

It closes the file

```
myfile.close()
```

Exception Handling

An exception is an unusual condition that results in an interruption in the flow of the program.

try and catch block

A basic try-catch block in python. When the try block throws an error, the control goes to the except block

```
try {
// code to try
throw exception; // If a problem arises, then throw an exception
}
catch () {
// Block of code to handle errors
}
```

Basics

Basic syntax from the python programming language

Showing Output To User

the print function is used to display or print output

```
print("Content that you wanna print on screen")
```

Taking Input From User

the input function is used to take input from the user

```
var1 = input("Enter your name: ")
```

Empty List

This method allows you to create an empty list

```
my_list = []
```

Empty Dictionary

By putting two curly braces, you can create a blank dictionary

```
my_dict = {}
```

Range Function

range function returns a sequence of numbers, eg, numbers starting from 0 to n-1 for range(0, n)

```
range(int_value)
```

Comments

Comments are used to make the code more understandable for programmers, and they are not executed by compiler or interpreter.

Single line comment

```
#This is a single line comment
```

Multi-line comment

```
'''This is a
multi-line
comment'''
```

Escape Sequence

An escape sequence is a sequence of characters; it doesn't represent itself when used inside string literal or character.

Newline

Newline Character

```
\n
```

Backslash

It adds a backslash

```
\\
```

Single Quote

It adds a single quotation mark

```
\'
```

Tab

It gives a tab space

```
\t
```

Backspace

It adds a backspace

```
\b
```

Octal value

It represents the value of an octal number

```
\ooo
```

Hex value

It represents the value of a hex number

```
\xhh
```

Carriage Return

Carriage return or \r is a unique feature of Python. \r will just work as you have shifted your cursor to the beginning of the string or line.

```
\r
```

Strings

Python string is a sequence of characters, and each character can be individually accessed. Using its index.

String

You can create Strings by enclosing text in both forms of quotes - single quotes or double-quotes.

```
variable_name = "String Data"
```

Slicing

Slicing refers to obtaining a sub-string from the given string.

```
var_name[n : m]
```

String Methods isalnum() method

Returns True if all characters in the string are alphanumeric

```
string_variable.isalnum()
```

isalpha() method

Returns True if all characters in the string are alphabet

```
string_variable.isalpha()
```

isdecimal() method

Returns True if all characters in the string are decimals

```
string_variable.isdecimal()
```

isdigit() method

Returns True if all characters in the string are digits

```
string_variable.isdigit()
```

islower() method

Returns True if all characters in the string are lower case

```
string_variable.islower()
```

isspace() method

Returns True if all characters in the string are whitespaces

```
string_variable.isspace()
```

isupper() method

Returns True if all characters in the string are upper case

```
string_variable.isupper()
```

lower() method

Converts a string into lower case

```
string_variable.lower()
```

upper() method

Converts a string into upper case

```
string_variable.upper()
```

strip() method

It removes leading and trailing spaces in the string

```
string_variable.strip()
```

List

A List in Python represents a list of comma-separated values of any data type between square brackets.

List

```
var_name = [element1, element2, and so on]
```

List Methods index method

Returns the index of the first element with the specified value

```
list.index(element)
```

append method

Adds an element at the end of the list

```
list.append(element)
```

extend method

Add the elements of a list (or any iterable) to the end of the current list

```
list.extend(iterable)
```

insert method

Adds an element at the specified position

```
list.insert(position, element)
```

pop method

Removes the element at the specified position and returns it

```
list.pop(position)
```

remove method

The remove( ) method removes the first occurrence of a given item from the list

```
list.remove(element)
```

clear method

Removes all the elements from the list

```
list.clear()
```

count method

Returns the number of elements with the specified value

```
list.count(value)
```

reverse method

Reverse the order of the list

```
list.reverse()
```

sort method

Sorts the list

```
list.sort(reverse=True|False)
```

Tuples

Tuples are represented as a list of comma-separated values of any data type within parentheses.

Tuple Creation

```
variable_name = (element1, element2, ...)
```

Tuple Methods count method

It returns the number of times a specified value occurs in a tuple

```
tuple.count(value)
```

index method

It searches the tuple for a specified value and returns the position.

```
tuple.index(value)
```

Sets

A set is a collection of multiple values which is both unordered and unindexed. It is written in curly brackets.

Set Creation: Way 1

```
var_name = {element1, element2, ...}
```

Set Creation: Way 2

```
var_name = set([element1, element2, ...])
```

Set Methods: add() method

Adds an element to a set

```
set.add(element)
```

clear() method

Remove all elements from a set

```
set.clear()
```

discard() method

Removes the specified item from the set

```
set.discard(value)
```

intersection() method

Returns intersection of two or more sets

```
set.intersection(set1, set2 ... etc)
```

issubset() method

Checks if a Set is Subset of Another Set

```
set.issubset(set)
```

pop() method

Removes an element from the set

```
set.pop()
```

remove() method

Removes the specified element from the Set

```
set.remove(item)
```

union() method

Returns the union of Sets

```
set.union(set1, set2...)
```

Dictionaries

The dictionary is an unordered set of comma-separated key: value pairs, within {}, with the requirement that within a dictionary, no two keys can be the same.

Dictionary

```
<dictionary-name> = {<key>: value, <key>: value ...}
```

Adding Element to a dictionary

By this method, one can add new elements to the dictionary

```
<dictionary>[<key>] = <value>
```

Updating Element in a dictionary

If the specified key already exists, then its value will get updated

```
<dictionary>[<key>] = <value>
```

Deleting Element from a dictionary

del let to delete specified key: value pair from the dictionary

```
del <dictionary>[<key>]
```

Dictionary Functions & Methods len() method

It returns the length of the dictionary, i.e., the count of elements (key: value pairs) in the dictionary

```
len(dictionary)
```

clear() method

Removes all the elements from the dictionary

```
dictionary.clear()
```

get() method

Returns the value of the specified key

```
dictionary.get(keyname)
```

items() method

Returns a list containing a tuple for each key-value pair

```
dictionary.items()
```

keys() method

Returns a list containing the dictionary's keys

```
dictionary.keys()
```

values() method

Returns a list of all the values in the dictionary

```
dictionary.values()
```

update() method

Updates the dictionary with the specified key-value pairs

```
dictionary.update(iterable)
```

Conditional Statements

The if statements are the conditional statements in Python, and these implement selection constructs (decision constructs).

if Statement

```
if(conditional expression):
statements
```

if-else Statement

```
if(conditional expression):
statements
else:
statements
```

if-elif Statement

```
if (conditional expression) :
statements
elif (conditional expression) :
statements
else :
statements
```

Nested if-else Statement

```
if (conditional expression):
if (conditional expression):
statements
else:
statements
else:
statements
```

Iterative Statements

An iteration statement, or loop, repeatedly executes a statement, known as the loop body, until the controlling expression is false (0).

For Loop

The for loop of Python is designed to process the items of any sequence, such as a list or a string, one by one.

```
for <variable> in <sequence>:
statements_to_repeat
```

While Loop

A while loop is a conditional loop that will repeat the instructions within itself as long as a conditional remains true.

```
while <logical-expression> :
loop-body
```

Break Statement

The break statement enables a program to skip over a part of the code. A break statement terminates the very loop it lies within.

```
for <var> in <sequence> :
statement1
if <condition> :
break
statement2
statement_after_loop
```

Continue Statement

The continue statement skips the rest of the loop statements and causes the next iteration to occur.

```
for <var> in <sequence> :
statement1
if <condition> :
continue
statement2
statement3
statement4
```

Functions

A function is a block of code that performs a specific task. You can pass parameters into a function. It helps us to make our code more organized and manageable.

Function Definition

```
def my_function(parameters):
# Statements
```

File Handling

File handling refers to reading or writing data from files. Python provides some functions that allow us to manipulate data in the files.

open() function

```
var_name = open("file name", "opening mode")
```

close() function

```
var_name.close()
```

Read () function

The read functions contains different methods, read(),readline() and readlines()

```
read() #return one big string
```

It returns a list of lines

```
read-lines
```

It returns one line at a time

```
readline
```

Write () function

This function writes a sequence of strings to the file.

```
write () #Used to write a fixed sequence of characters to a file
```

It is used to write a list of strings

```
writelines()
```

Append () function

The append function is used to append to the file instead of overwriting it. To append to an existing file, simply open the file in append mode (a):

```
file = open("Hello.txt", "a")
```

Exception Handling

An exception is an unusual condition that results in an interruption in the flow of the program.

try and except

A basic try-catch block in python. When the try block throws an error, the control goes to the except block.

```
try:
[Statement body block]
raise Exception()
```

```
except Exception as e:
[Error processing block]
```

OOPS

It is a programming approach that primarily focuses on using objects and classes. The objects can be any real-world entities.

class

The syntax for writing a class in python

```
class class_name:
#Statements
```

class with a constructor

The syntax for writing a class with the constructor in python

```
class CodeWithHarry:

# Default constructor
def __init__(self):
self.name = "CodeWithHarry"

# A method for printing data members
def print_me(self):
print(self.name)
```

object

Instantiating an object

```
<object-name> = <class-name>(<arguments>)
```

filter function

The filter function allows you to process an iterable and extract those items that satisfy a given condition

```
filter(function, iterable)
```

issubclass function

Used to find whether a class is a subclass of a given class (classinfo) or not

```
issubclass(class, classinfo)
```

Iterators and Generators

Here are some of the advanced topics of the Python programming language like iterators and generators

Iterator

Used to create an iterator over an iterable

```
iter_list = iter(['Harry', 'Aakash', 'Rohan'])
print(next(iter_list))
print(next(iter_list))
print(next(iter_list))
```

Generator

Used to generate values on the fly

```
# A simple generator function
def my_gen():
n = 1
print('This is printed first')
# Generator function contains yield statements
yield n
n += 1
print('This is printed second')
yield n
n += 1
print('This is printed at last')
yield n
```

Decorators

Decorators are used to modifying the behavior of function or class. They are usually called before the definition of a function you want to decorate.

property Decorator (getter)

```
@property
def name(self):
return self.__name
```

setter Decorator

It is used to set the property 'name'

```
@name.setter
def name(self, value):
self.__name=value
```

Deletor Decorator

It is used to delete the property 'name'

```
@name.deleter #property-name.deleter decorator
def name(self, value):
print('Deleting..')
del self.__name
```