**INTRODUCTION**

* **PROGRAMMING LANGUAGES**

A programming language is a vocabulary and set of grammatical rules for instructing a [computer](https://www.webopedia.com/TERM/C/computer.html) or computing device to perform specific tasks. The term programming language usually refers to [high-level languages](https://www.webopedia.com/TERM/H/high_level_language.html), such as [BASIC](https://www.webopedia.com/TERM/B/BASIC.html), [C](https://www.webopedia.com/TERM/C/C.html), [C++](https://www.webopedia.com/TERM/C/C_plus_plus.html), [COBOL](https://www.webopedia.com/TERM/C/COBOL.html), [Java](https://www.webopedia.com/TERM/J/Java.html), [FORTRAN](https://www.webopedia.com/TERM/F/FORTRAN.html), [Ada](https://www.webopedia.com/TERM/A/Ada.html), and [Pascal](https://www.webopedia.com/TERM/P/Pascal.html).

Each programming language has a unique set of keywords (words that it understands) and a special [syntax](https://www.webopedia.com/TERM/S/syntax.html) for organizing program [instructions](https://www.webopedia.com/TERM/I/instruction.html). A computer is a computational device which is used to process the data under the control of a computer program. Program is a sequence of instruction along with data. While executing the program, raw data is processed into a desired output format. These computer programs are written in a programming language which are high level languages. High level languages are nearly human languages which are more complex than the computer understandable language which are called machine language, or low-level language.



## HIGH-LEVEL PROGRAMMING LANGUAGES

High-level programming languages, while simple compared to human languages, are more complex than the languages the computer actually understands, called [machine languages](https://www.webopedia.com/TERM/M/machine_language.html). Each different type of [CPU](https://www.webopedia.com/TERM/C/CPU.html) has its own unique machine language.

Lying between machine languages and high-level languages are languages called [assembly languages](https://www.webopedia.com/TERM/A/assembly_language.html). Assembly languages are similar to machine languages, but they are much easier to program in because they allow a [programmer](https://www.webopedia.com/TERM/P/programmer.html) to substitute [names](https://www.webopedia.com/TERM/N/name.html) for numbers. Machine languages consist of numbers only.

Lying above high-level languages are languages called [fourth-generation languages](https://www.webopedia.com/TERM/F/fourth_generation_language.html)(usually abbreviated 4GL). 4GLs are far removed from machine languages and represent the class of computer languages closest to human languages.

High-level languages are designed to be used by the human operator or the programmer. They are referred to as "closer to humans." In other words, their programming style and context is easier to learn and implement than low-level languages, and the entire code generally focuses on the specific program to be created.

A high-level language does not require addressing hardware constraints when developing a program. However, every single program written in a high-level language must be interpreted into machine language before being executed by the computer

"High-level language" refers to the higher level of abstraction from [machine language](https://en.wikipedia.org/wiki/Machine_language). Rather than dealing with registers, memory addresses and call stacks, high-level languages deal with variables, arrays, [objects](https://en.wikipedia.org/wiki/Object_(computer_science)), complex arithmetic or Boolean expressions, subroutines and functions, loops, [threads](https://en.wikipedia.org/wiki/Thread_(computer_science)), locks, and other abstract computer science concepts, with a focus on [usability](https://en.wikipedia.org/wiki/Usability) over optimal program efficiency.

Unlike low-level [assembly languages](https://en.wikipedia.org/wiki/Assembly_language), high-level languages have few, if any, language elements that translate directly into a machine's native [opcodes](https://en.wikipedia.org/wiki/Opcode). One thing to note about high-level programming languages is that these languages allow the programmer to be detached and separated from the machine. That is, unlike low-level languages like assembly or machine language, high-level programming can amplify the programmer's instructions and trigger a lot of data movements in the background without their knowledge. The responsibility and power of executing instructions have been handed over to the machine from the programmer.

## ASSEMBLY LANGUAGES

A [programming language](https://www.webopedia.com/TERM/P/programming_language.html) that is once removed from a computer’s machine. Machine languages consist entirely of numbers and are almost impossible for humans to read and write. Assembly languages have the same structure and set of [commands](https://www.webopedia.com/TERM/C/command.html) as machine languages, but they enable a [programmer](https://www.webopedia.com/TERM/P/programmer.html) to use names instead of numbers.

An assembly language is a low-level programming language for microprocessors and other programmable devices. It is not just a single language, but rather a group of languages. An assembly language implements a symbolic representation of the machine code needed to program a given CPU architecture.

Assembly language is also known as assembly code. The term is often also used synonymously with 2GL.

An assembly language is the most basic programming language available for any processor. With assembly language, a programmer works only with operations that are implemented directly on the physical CPU. Assembly languages generally lack high-level conveniences such as variables and functions, and they are not portable between various families of processors. They have the same structures and set of commands as machine language but allow a programmer to use names instead of numbers. This language is still useful for programmers when speed is necessary or when they need to carry out an operation that is not possible in high-level languages. An assembly language is a low-level [programming language](https://techterms.com/definition/programming_language) designed for a specific type of [processor](https://techterms.com/definition/processor). It may be produced by [compiling](https://techterms.com/definition/compile) source code from a high-level programming language (such as [C/C++](https://techterms.com/definition/cplusplus)) but can also be written from scratch. Assembly code can be converted to machine code using an [assembler](https://techterms.com/definition/assembler).

Since most [compilers](https://techterms.com/definition/compiler) convert [source code](https://techterms.com/definition/sourcecode) directly to machine code, software developers often create [programs](https://techterms.com/definition/program) without using assembly language. However, in some cases, assembly code can be used to fine-tune a program. For example, a programmer may write a specific [process](https://techterms.com/definition/process) in assembly language to make sure it functions as efficiently as possible.

While assembly languages differ between processor [architectures](https://techterms.com/definition/architecture), they often include similar instructions and operators. Below are some examples of instructions supported by [x86](https://techterms.com/definition/x86) processors.

## MACHINE LANGUAGES

Machine language is the lowest-level [programming language](https://www.webopedia.com/TERM/P/programming_language.html)(except for computers that utilize programmable [microcode](https://www.webopedia.com/TERM/M/microcode.html)). Machine languages are the only [languages](https://www.webopedia.com/TERM/L/language.html) understood by computers. Computer programs are written in one or more [programming languages](https://www.computerhope.com/jargon/p/programming-language.htm), like [C++](https://www.computerhope.com/jargon/c/cplus.htm), [Java](https://www.computerhope.com/jargon/j/java.htm), or [Visual Basic](https://www.computerhope.com/jargon/v/vb.htm). A computer cannot directly understand the programming languages used to create computer programs, so the program code must be [compiled](https://www.computerhope.com/jargon/c/compile.htm). Once a program's code is compiled, the computer can understand it because the program's code is turned into machine language.

Below is an example of machine language (binary) for the text "Hello World".

01001000 01100101 01101100 01101100 01101111 00100000 01010111 01101111 01110010 01101100 01100100

## HISTORY OF PYTHON

Python was developed in 1980 by **Guido van Rossum** at the National Research Institute for Mathematics and Computer Science in the Netherlands as a successor of ABC language capable of exception handling and interfacing. Python features a dynamic type system and automatic memory management. It supports multiple programming

paradigms, including object-oriented, imperative, functional and

procedural, and has a large and comprehensive standard library.

Van Rossum picked the name Python for the new language from a TV

show, Monty Python's Flying Circus.

In December 1989 the creator developed the 1st python interpreter as

a hobby and then on 16 October 2000, Python 2.0 was released with

many new features.

...In December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of

ABC that would appeal to Unix/C hackers. I chose Python as a working

title for the project, being in a slightly irreverent mood (and a big fan of

Monty Python's Flying Circus)

—**Guido van Rossum**

**Python Version Releases Date**

Python 1.0 - January 1994

Python 1.5 - December 31, 1997

Python 1.6 - September 5, 2000

Python 2.0 - October 16, 2000

Python 2.1 - April 17, 2001

Python 2.2 - December 21, 2001

Python 2.3 - July 29, 2003

Python 2.4 - November 30, 2004

Python 2.5 - September 19, 2006

Python 2.6 - October 1, 2008

Python 2.7 - July 3, 2010

Python 3.0 - December 3, 2008

Python 3.1 - June 27, 2009

Python 3.2 - February 20, 2011

Python 3.3 - September 29, 2012

Python 3.4 - March 16, 2014

Python 3.5 - September 13, 2015

Python 3.6 - December 23, 2016

Python 3.7 - June 27, 2018

* **WHY PYTHON**

**1) Readable and Maintainable Code**

While writing a software application, you must focus on the quality of its source code to simplify maintenance and updates.

**2) Multiple Programming Paradigms**

Like other modern programming languages, Python also supports several programming paradigms. It supports object oriented and structured programming fully.

**3) Compatible with Major Platforms and Systems**

At present, Python is supports many operating systems. You can even use Python interpreters to run the code on specific platforms and tools. Also, Python is an interpreted programming language. It allows you to you to run the same code on multiple platforms without recompilation

**4) Robust Standard Library**

Its large and robust standard library makes Python score over other programming languages. The standard library allows you to choose from a wide range of modules according to your precise needs. Each module further enables you to add functionality to the Python application without writing additional code.

**5) Many Open Source Frameworks and Tools**

As an open source programming language, Python helps you to curtail software development cost significantly. You can even use several open source Python frameworks, libraries and development tools to curtail development time without increasing development cost.

* **CHARACTERISTICS OF PYTHON**
* **Easy to read**: Python source-code is clearly defined and visible to the eyes.
* **Portable**: Python codes can be run on a wide variety of hardware platforms

having the same interface.

* **Extendable**: Users can add low level-modules to Python interpreter.
* **Scalable**: Python provides an improved structure for supporting large

programs than shell-scripts.

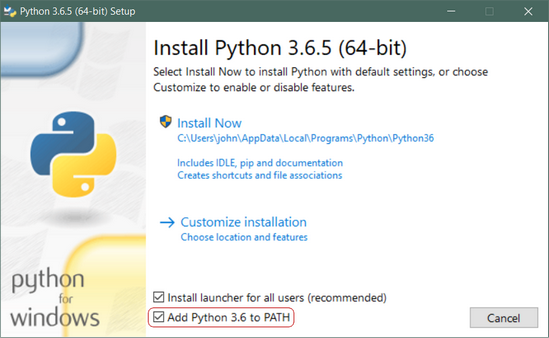
* **Object-Oriented Language**: It supports object-oriented features

and techniques of programming.

* **Interactive Programming Language**: Users can interact with the python

interpreter directly for writing programs.

* **Easy language**: Python is easy to learn language especially for beginners.
* **Straight forward Syntax**: The formation of python syntax is simple and straightforward which also makes it popular.
* **DOWNLOADING & INSTALLING PYTHON**

1. Open a browser window and navigate to the [Download page for Windows](https://www.python.org/downloads/windows/) at [python.org](https://www.python.org/).
2. Underneath the heading at the top that says **Python Releases for Windows**, click on the link for the **Latest Python 3 Release - Python 3.x.x**. (As of this writing, the latest is Python 3.7.1.)
3. Scroll to the bottom and select either **Windows x86-64 executable installer** for 64-bit or **Windows x86 executable installer** for 32-bit.
4. [](https://files.realpython.com/media/win-install-dialog.40e3ded144b0.png)Once you have chosen and downloaded an installer, simply run it by double-clicking on the downloaded file. A dialog should appear that looks something like this:

* **PRACTICAL WORKING WITH PYTHON**

**IDLE**

IDLE is an integrated development environment for Python, which has been bundled with the default implementation of the language since 1.5.2b1. It is packaged as an optional part of the Python packaging with many Linux distributions. It is completely written in Python and the Tkinter GUI toolkit.

**PYTHON 3.7 SHELL**

Python provides a Python Shell (also known as Python Interactive Shell) which is used to execute a single Python command and get the result. Python Shell waits for the input command from the user.

## Variables

Variables are containers for storing data values.

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

E.g. x = 5  
 y = "John"  
 print(x)  
 print(y)

## String Literals

String literals in python are surrounded by either single quotation marks, or double quotation marks.

‘hello’ is the same as “hello”

You can display a string literal with the print() function

E.g. a = "Hello Python"  
 print(a)

**Data Structures in Python**

1. **LISTS**

* Ordered collection of data.
* Supports similar slicing and indexing functionalities as in the case of Strings.
* They are mutable.
* Advantage of a list over a conventional array
  + Lists have no size or type constraints(no setting restrictions beforehand).
  + They can contain different object types.
  + We can delete elements from a list by using Del list\_name[index\_val]

Example-

* + my\_list = ['one', 'two','three',4,5]
  + len(my\_list) would output 5.

1. **Dictionary**

* Lists are sequences but the dictionaries are mappings.
* They are mappings between a unique key and a value pair.
* These mappings may not retain order.
* Constructing a dictionary.
* Accessing object from a dictionary**.**
* Nesting Dictionaries.
* Basic Dictionary Methods.
* Basic Syntax
* d={} empty dictionary will be generated and assign keys and values to
* it, like d[‘animal’] = ‘Dog’
* d = {'K1':'V1', 'K2’:’V2'}
* d['K1'] outputs 'V1‘

1. **Tuples**

* Immutable in nature, i.e they cannot be changed.
* No type restriction
* Indexing and slicing, everything's same like that in strings and lists.
* Constructing tuples.
* Basic tuple methods.
* Immutability.
* When to use tuples?
* We can use tuples to present things that shouldn’t change, such as days of
* the week, or dates on a calendar, etc.

1. **Sets**

* A set contains unique and unordered elements and we can construct them
* by using a set() function.
* Convert a list into Set-
* l=[1,2,3,4,1,1,2,3,6,7]
* k = set(l)
* k becomes {1,2,3,4,6,7}
* Basic Syntax-
* x=set()
* x.add(1)
* x = {1}
* x.add(1)
* This would make no change in x now

# **LOOPS IN PYTHON**

1. [**While Loop**](https://www.tutorialspoint.com/python/python_while_loop.htm)

Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

Syntax 🡪 **While(Expression):**

E.g. i = 1  
 while(i < 6):  
   print(“ Hello Python…”)

## For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a

set, or a string).

Syntax 🡪 **for x in range(expression):**

E.g. fruits = ["A", "B", "C"]  
  x in fruits:  
   print(x)

## 3. Break Statement

With the break statement we can stop the loop even if the while condition is true.

Syntax 🡪 **break;**

E.g. i = 1  
 while i < 6:  
   print(i)  
   if i == 3:  
     break  
   i += 1

## 4. Continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

Syntax 🡪 **continue;**

E.g. i = 0  
while i < 6:  
  i += 1   
  if i == 3:  
    continue  
  print(i)

* **CONDITIONAL STATEMENTS**

1. **if statement** 🡪 These conditions can be used in several ways, most commonly in "if statements" and loops. An "if statement" is written by using the **if** keyword.

Syntax 🡪 **if(condition):**

**Statements**

E.g. a = 33  
 b = 200  
 if(a<b):

   print("b is greater than a")

## elif statement🡪 The elif keyword is python’s way of saying "if the previous conditions were not true, then try this condition".

Syntax 🡪 **if(condition):**

**statements**

**elif:**

**statements**

**E.g.** a = 33

b = 33

if(b > a):

print("b is greater than a")

elif a == b:

print("a and b are equal")

1. **else statement 🡪**The else keyword catches anything which isn't caught by the preceding conditions.

Syntax 🡪 **if(condition):**

**statements**

**else:**

**statements**

E.g. a = 200  
 b = 33  
 if(b > a):  
   print("b is greater than a")  
 elif(a == b):  
   print("a and b are equal")  
 else:  
   print("a is greater than b")

1. **Switch statements 🡪** The Pythonic way to implement switch statement is to use the powerful dictionary mappings, also known as associative arrays, that provide simple one-to-one key-value mappings

Syntax 🡪**def function\_name(argument):**

**switcher = {**

**case1: statement,**

**case2: statement,**

**}**

**E.g.** def switch\_demo(argument):

    switcher = {

        1: "January",

        2: "February",

        3: "March",

        4: "April",

        5: "May",

        6: "June",

        7: "July",

        8: "August",

        9: "September",

        10: "October",

        11: "November",

        12: "December"

    }

    print switcher.get(argument, "Invalid month")

# **PYTHON STRING METHODS**

# **capitalize( )**

Upper case the first letter in this sentence:

Syntax 🡪 **string.capitalize( )**

E.g. txt = "hello, and welcome to my world."  
  
x = txt.capitalize()  
  
print (x)

1. **center( )**

Pads string with specified character

**Syntax 🡪 string.center(width[, fillchar])**

E.g. string = "Python is awesome"

new\_string = string.center(24)

print("Centered String: ", new\_string)

1. **casefold( )**

Converts to casefolded strings

**Syntax 🡪 string.casefold()**

E.g. string = "PYTHON IS AWESOME"

print("Lowercase string:", string.casefold())

1. **count( )**

Returns occurrences of substring in string

**Syntax 🡪 string.count(substring, start=..., end=...)**

**E.g.** string = "Python is awesome, isn't it?"

substring = "is"

count = string.count(substring)

# print count

print("The count is:", count)

1. **islower( )**

Checks if all Alphabets in a String are Lowercase

**Syntax 🡪** string.islower( )

**E.g.** s = 'this is good'

print(s.islower())

s = 'this is a1so g00d'

print(s.islower())

s = 'this is Not good'

print(s.islower())

1. **isupper()**

Returns if all characters are uppercase characters

**Syntax 🡪** string.isupper()

**E.g.** string = "THIS IS GOOD!"

print(string.isupper());

# numbers in place of alphabets

string = "THIS IS ALSO G00D!"

print(string.isupper());

1. **join( )**

Returns a Concatenated String

**Syntax 🡪 string.join(iterable)**

**E.g.** numList = ['1', '2', '3', '4']

seperator = ', '

print(seperator.join(numList))

numTuple = ('1', '2', '3', '4')

print(seperator.join(numTuple))

s1 = 'abc'

s2 = '123'

""" Each character of s2 is concatenated to the front of s1"""

print('s1.join(s2):', s1.join(s2))

""" Each character of s1 is concatenated to the front of s2"""

print('s2.join(s1):', s2.join(s1))

1. **swapcase( )**

Swap uppercase characters to lowercase; vice versa

**Syntax 🡪 string.swapcase( )**

**E.g.** string = "THIS SHOULD ALL BE LOWERCASE."

print(string.swapcase())

string = "this should all be uppercase."

print(string.swapcase())

string = "ThIs ShOuLd Be MiXeD cAsEd."

print(string.swapcase())

1. **split( )**

Splits String from Left

**Syntax 🡪 str.split([separator [, maxsplit]])**

**E.g.** text= 'Love thy neighbour'

# splits at space

print(text.split())

grocery = 'Milk, Chicken, Bread'

# splits at ','

print(grocery.split(', '))

# Splitting at ':'

print(grocery.split(':'))

1. **replace( )**

The replace( ) method returns a copy of the string where all occurrences of a substring is replaced with another substring.

**Syntax 🡪 str.replace(old, new [, count])**

**E.g**. song = 'cold, cold heart'

print (song.replace('cold', 'hurt'))

song = 'Let it be, let it be, let it be, let it be'

'''only two occurrences of 'let' is replaced'''

print(song.replace('let', "don't let", 2))

# **PYTHON INTEGERS METHOD**

1. **int( )**

The int( ) method returns an integer object from any number or string.

**Syntax 🡪 int(x=0, base=10)**

## int( ) Parameters

## The int( ) method takes two arguments:

* **x** - Number or string to be converted to integer object.
* Default argument is **zero**.
* **base** - Base of the number in **x**.  
  Can be 0 (code literal) or 2-36.

## Return value from int( )

## The int( ) method returns:

* an integer object from the given number or string, treats default base as 10
* (No parameters) returns 0
* (If base given) treats the string in the given base (0, 2, 8, 10, 16)

**E.g.** print("int(123) is:", int(123))

# float

print("int(123.23) is:", int(123.23))

# string

print("int('123') is:", int('123'))

* **PYTHON MODULES INTRODUCTION**

**Use of NumPy**

NumPy is a Python package. It stands for 'Numerical Python'. It is a library

consisting of multidimensional array objects and a collection of routines for

processing of array.

**Numeric**, the ancestor of NumPy, was developed by Jim Hugunin. Another

package Numarray was also developed, having some additional functionalities. In

2005, Travis Oliphant created NumPy package by incorporating the features of

Numarray into Numeric package. There are many contributors to this open source

project.

**Operations using NumPy**

Using NumPy, a developer can perform the following operations −

* Mathematical and logical operations on arrays.
* Fourier transforms and routines for shape manipulation.
* Operations related to linear algebra. NumPy has in-built functions for linear

algebra and random number generation.

**E.g.** import numpy as np

# Creating array object

arr = np.array( [[ 1, 2, 3],

                 [ 4, 2, 5]] )

# Printing type of arr object

print("Array is of type: ", type(arr))

# Printing array dimensions (axes)

print("No. of dimensions: ", arr.ndim)

# Printing shape of array

print("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array

print("Size of array: ", arr.size)

# Printing type of elements in array

print("Array stores elements of type: ", arr.dtype)

**Output :**

Array is of type:

No. of dimensions: 2

Shape of array: (2, 3)

Size of array: 6

Array stores elements of type: int64

**Use of Pandas**

Pandas is an open-source, BSD-licensed Python library providing high-performance,

easy-to-use data structures and data analysis tools for the Python

programming language. Python with Pandas is used in a wide range of fields

including academic and commercial domains including finance, economics,

Statistics, analytics, etc.

Pandas is an open-source Python Library providing high-performance data

manipulation and analysis tool using its powerful data structures. The name

Pandas is derived from the word Panel Data – an Econometrics from

Multidimensional data.

**Key Features of Pandas**

• Fast and efficient Data Frame object with default and customized indexing**.**

• Tools for loading data into in-memory data objects from different file

formats.

• Data alignment and integrated handling of missing data.

• Reshaping and pivoting of date sets.

• Label-based slicing, indexing and subletting of large data sets.

• Columns from a data structure can be deleted or inserted.

• Group by data for aggregation and transformations.

Pandas deals with the following three data structures −

* Series
* DataFrame
* Panel

# **Python GUI – tkinter**

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter outputs the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.  
**To create a tkinter:**

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.

There are two main methods used you the user need to remember while creating the Python application with GUI.

1. **Tk(screenName=None,  baseName=None,  className=’Tk’,  useTk=1):** To create a main window, tkinter offers a method ‘Tk(screenName=None,  baseName=None,  className=’Tk’,  useTk=1)’. To change the name of the window, you can change the className to the desired one. The basic code used to create the main window of the application is:

**m=tkinter.Tk() where m is the name of the main window object**

1. **mainloop():** There is a method known by the name mainloop() is used when you are ready for the application to run. mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event till the window is not closed.

**m.mainloop()**

**E.g.** import tkinter

m = tkinter.Tk()

'''

widgets are added here

'''

m.mainloop()

tkinter also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows. There are mainly three geometry manager classes class.

1. **pack() method:**It organizes the widgets in blocks before placing in the parent widget.
2. **grid() method:**It organizes the widgets in grid (table-like structure) before placing in the parent widget.
3. **place() method:**It organizes the widgets by placing them on specific positions directed by the programmer.

**Creating a Cmd Base Calculator Using Looping Statement**

**Code:** print("1. Addition")

print("\n2.Subtration")

print("\n3.Multiplication")

print("\n4.Divison")

print("\n5.stop and exit")

choice = int(input("Enter your choice: "))

if(choice>=1 and choice<=4):

print("\nEnter two numbers")

num = int(input("1--> "))

num2 = int(input("2--> "))

#add block

if (choice == 1):

print(num+num2)

#sub block

elif(choice == 2):

print(num-num2)

#mull block

elif(choice == 3):

print(num\*num2)

#div block

elif(choice == 4):

print(num/num2)

#5th block exit

else:

print("exit")

**OUTPUT:** 1. Addition

2.Subtration

3.Multiplication

4.Divison

5.stop and exit

Enter your choice: 1

Enter two numbers

1--> 10

2--> 20

30

* **PROJECT**

**S.E.O INTRODUCTION**

What is SEO/Search Engine Optimization ?

**🡪** SEO stands for **“search engine optimization.”** It is the process of getting traffic from the “free,” “organic,” “editorial” or “natural” search results on search engines.

SEO is an acronym that stands for search engine optimization, which is the process of optimizing your website to get organic, or un-paid, traffic from the search engine results page. In other words, SEO involves making certain changes to your website design and content that make your site more attractive to a search engine. You do this in hopes that the search engine will display your website as a top result on the search engine results page.

Though search engine optimization can get quite complex when it comes to all the different factors that impact your ranking, the basic process is not as difficult to understand. Search engines want to provide the best service for their users. This means delivering results on the search engine pages that are not only high quality but also relevant to what the searcher is looking for.

When it comes to SEO, there are two different approaches that organizations take to optimizing their sites for the search engines – black hat vs. white hat SEO.

Some organizations are only interested in SEO so that they can rank their content quickly and make some money in the short-term. Black hat SEO involves tactics that focus on optimizing content only for the search engines. This means that organizations are not considering the human visitors that will read and navigate their site content. These organizations will bend or break the rules in order to improve their site rankings to make a quick buck.

In the end, this approach to SEO produces pages that are often difficult for people to read and look a whole lot like spam. Though the sites may rank more quickly than those that are optimized properly, these sites are often penalized or banned by search engines rather quickly. Overall, this get-rich-quick approach to SEO ruins the organization’s chance of building a site that is sustainable and able to bring in new leads for years to come.

**OBJECTIVES OF SEO**

##### **• Generating Traffic**

##### **• Branding**

##### **• Marketing**

##### **• Ideological Impact**

##### **• E- Commerce Purpose**

##### **• Target Customers**

##### **• Competition**

##### **• Your Potential Customers Search for Your Products**

**SALIENT FEATURES**

* SEO will help the website to achieve a higher rank in the SERP.
* SEO helps the owner of the website boost the number of visitors by 400%
* It will help the customer conversions to increase by 100 %.
* SEO will ultimately help in increasing the profit of the website’s owner.

**CODING PHASE OF PROJECT**

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

from bs4 import BeautifulSoup

import re

import pandas as pd

import os

import csv

import urllib

from urllib.request import urlopen

import lxml

from selenium.webdriver.firefox.firefox\_binary import FirefoxBinary

from selenium.webdriver.common.desired\_capabilities import DesiredCapabilities

import nltk

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

from nltk.tokenize import RegexpTokenizer, word\_tokenize, sent\_tokenize

from nltk.stem.wordnet import WordNetLemmatizer

import termcolor

from termcolor import colored

import requests

import collections

from collections import Counter

import re

import string

from string import punctuation

import os

import random

nltk.download('wordnet')

nltk.download('punkt')

nltk.download('stopwords')

url=str(input("Enter a url you want to work with:" ))

# create a new Firefox session

binary = FirefoxBinary(r"C:\\Program Files\\Mozilla Firefox\\firefox.exe")

caps = DesiredCapabilities.FIREFOX.copy()

caps['marionette'] = True

driver=webdriver.Firefox(firefox\_binary=binary,capabilities=caps,executable\_path="C:\\Users\\vshiv\\AppData\\Local\\Programs\\Python\\Python37-32\\geckodriver-v0.24.0-win64\\geckodriver.exe")

driver.implicitly\_wait(30)

driver.get(url)

#dataset = pd.read\_csv('C:\\Users\\HP\\AppData\\Local\\Programs\\Python\\Python37-32\\papers1.csv','wb',error\_bad\_lines=False,delimiter = '\t , \n',engine='python')

#dataset.head()

#finding keywords

bsurl=urlopen(url)

soup=BeautifulSoup(bsurl, "lxml")

keywords= soup.find("meta",property="og:keywords")

print(colored("Keywords from the web page are:") +keywords["content"] if keywords else "No keywords found")

#telling driver to click a link

python\_button = driver.find\_element\_by\_link\_text(input("Enter the link text you want to find:")) #FHSU

python\_button.click()

print("Link Text found and clicked")

#stopwords counter

word = re.sub("[^a-zA-Z]"," ",soup.getText())

#extract words

words = word\_tokenize(word)

#remove stop words

stop\_words = set(stopwords.words('english'))

filtered\_words = [w for w in words if not w in stop\_words]

a = Counter([x.lower() for y in filtered\_words for x in y.split()])

b = (a.most\_common())

makeaframe = pd.DataFrame(b)

makeaframe.columns = ['Words', 'Frequency']

makeaframe.head()

print("Stopwords found and removed:")

print(makeaframe)

#word counter

# We get the words within paragrphs

text\_p = (''.join(s.findAll(text=True))for s in soup.findAll('p'))

c\_p = Counter((x.rstrip(punctuation).lower() for y in text\_p for x in y.split()))

# We get the words within divs

text\_div = (''.join(s.findAll(text=True))for s in soup.findAll('div'))

c\_div = Counter((x.rstrip(punctuation).lower() for y in text\_div for x in y.split()))

# We sum the two countesr and get a list with words count from most to less common

total = c\_div + c\_p

print("10 most common words are: ")

print(total.most\_common(10))

#Selenium hands the page source to Beautiful Soup

soup\_level1=BeautifulSoup(driver.page\_source, 'lxml')

x = 0 #counter

#data=[driver.find\_element\_by\_link\_text("Older Posts"),'driver.find\_element\_by\_link\_text("Newer Posts")]

while x>=0:

x+=1

#Beautiful Soup finds Title links on the agency page and the loop begins

if x%2==0 and x%3!=0:

for link in soup\_level1.find\_all('a',id=re.compile("Blog1\_blog-pager-older-link")):

python\_button =driver.find\_element\_by\_link\_text("Older Posts") #FHSU

python\_button.click()

print("Link to Older Posts clicked")

if python\_button==False:

python\_button=driver.find\_element\_by\_link\_text("Newer Posts")

python\_button.click()

print("Link to Newer Posts clicked because start of the file reached")

continue;

elif x%3==0 and x%2!=0:

for link in soup\_level1.find\_all('a',id=re.compile("Blog1\_blog-pager-newer-link")):

python\_button =driver.find\_element\_by\_link\_text("Newer Posts") #FHSU

python\_button.click()

print("Link to Newer Posts clicked")

continue;

#increment the counter variable before starting the loop over

elif x%2==0 and x%3==0:

driver.execute\_script("window.history.go(-1)")

print("Previous Page")

continue;

else:

python\_button=driver.find\_element\_by\_link\_text(input("Enter link text: "))

python\_button.click()

print("Entred link clicked")

continue;

#end loop block

#loop has completed

#combine all pandas dataframes in the list into one big dataframe

"""result = pd.concat([pd.DataFrame(datalist[i]) for i in range(len(datalist))],ignore\_index=True)

#convert the pandas dataframe to JSON

json\_records = result.to\_json(orient='records')

#pretty print to CLI with tabulate

#converts to an ascii table

print(tabulate(result, headers=['words','frequncy'],tablefmt='psql'))

#get current working directory

path = os.getcwd()

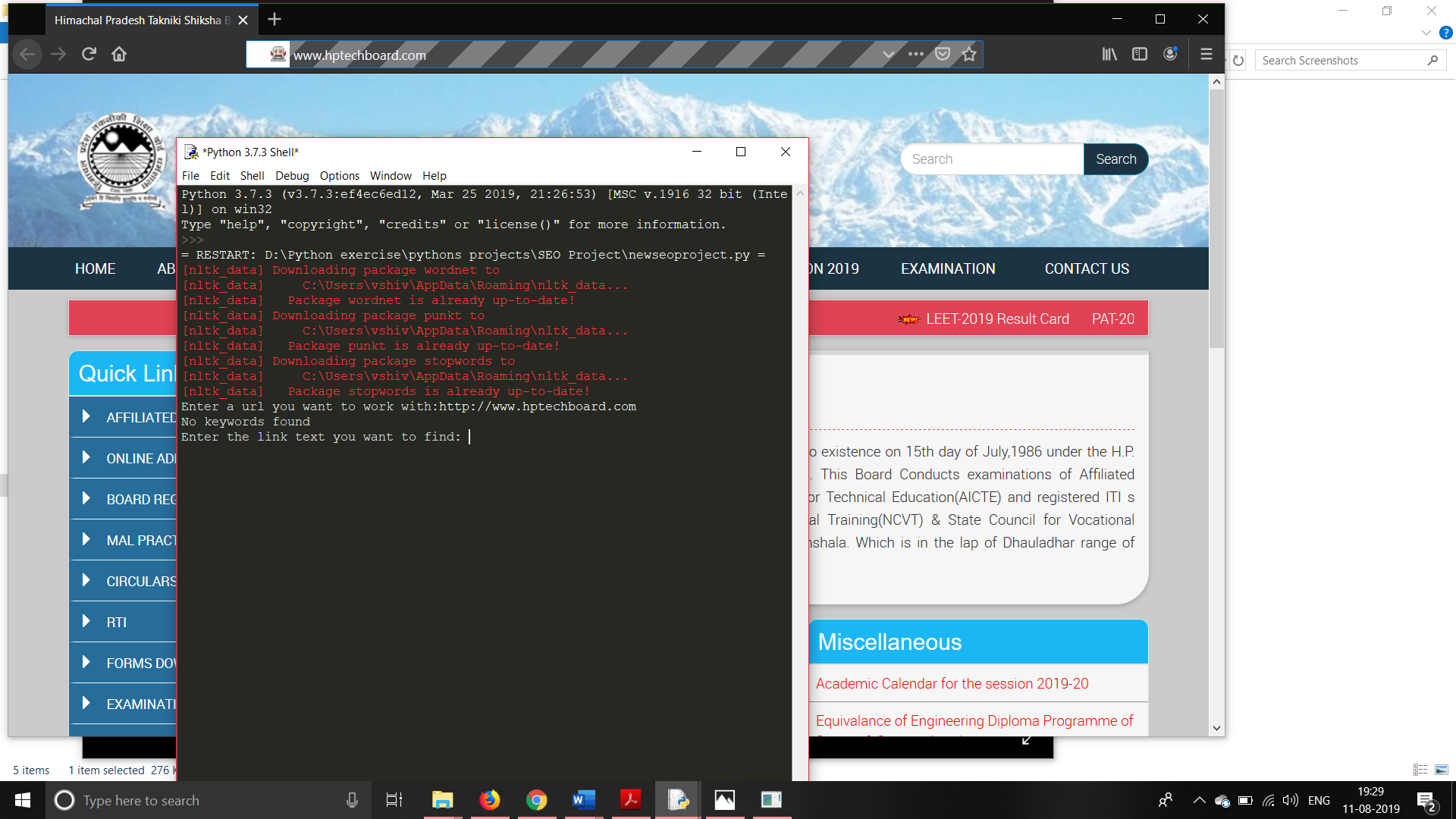
#open, write, and close the file

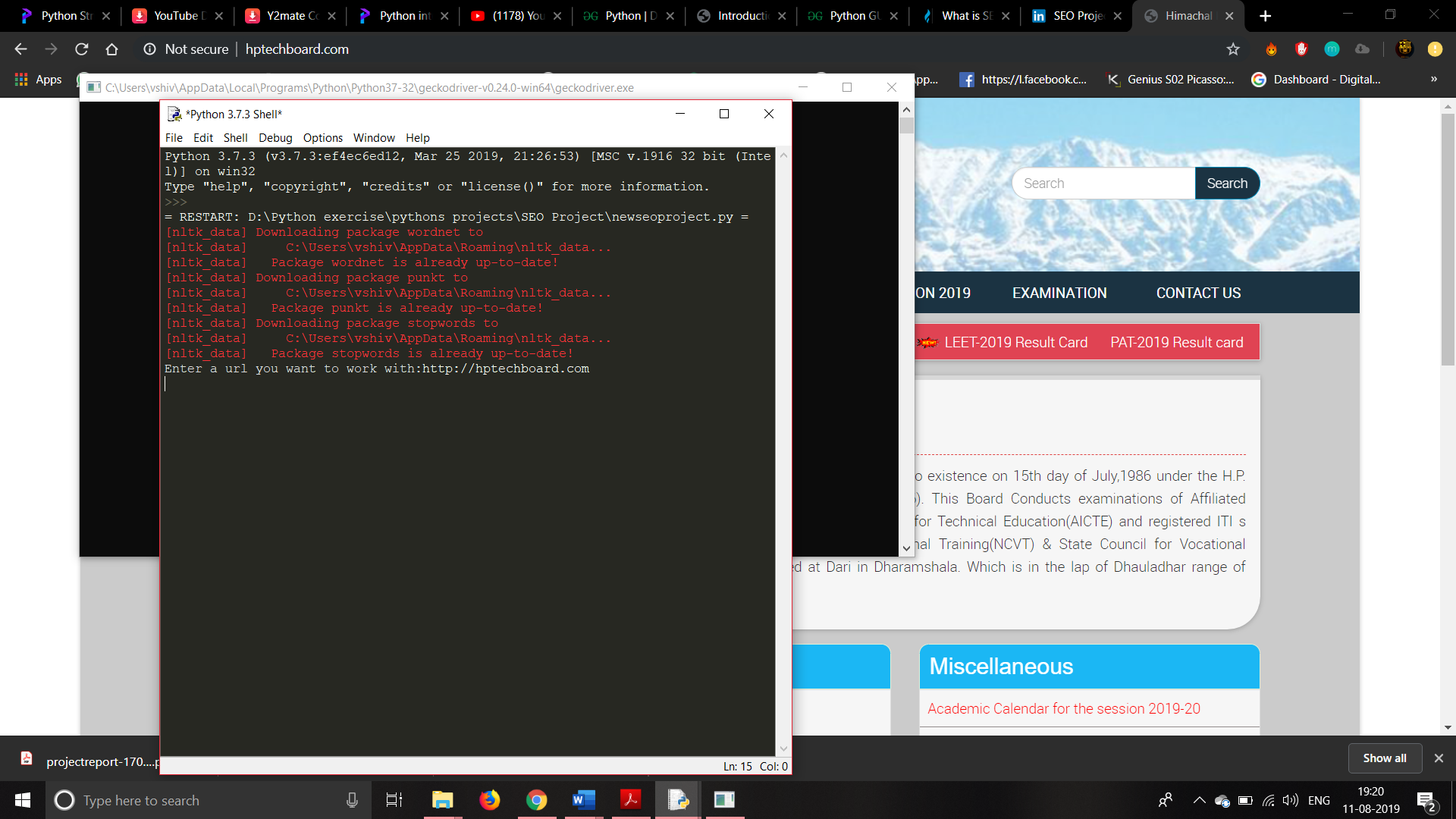
f = open(path + "\\fhsu\_payroll\_data.json","w") #FHSU

f.write(json\_records)

f.close()"""

**PROJECT OUT COMES**

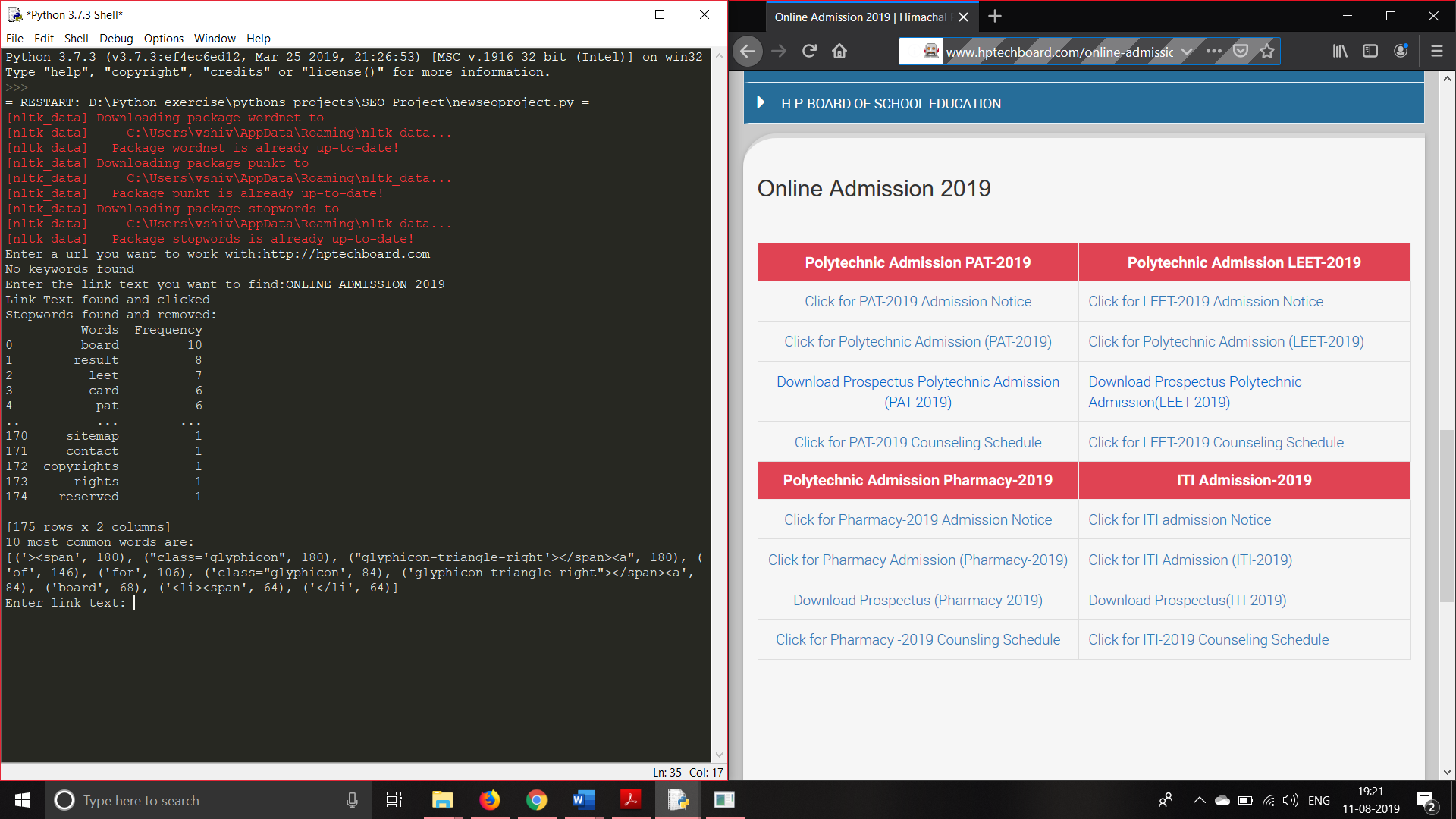
Enter the URL you want to be Optimize:

E.g. 🡪 http://www.hptechboard.com

Enter the link you want to know the frequency:

E.g. 🡪 ONLINE ADMISSION 2019

The Frequency of Entered Link :



* **CONCLUSION**

The project revolves around the specific domain of search engine optimization which comes under the umbrella term of Digital Marketing. The purpose of the project is to be very efficient and an effective so as to attain a substantial rank. And hence discovering and implementing ways to increase the probability of finding a good listing. The report explores all aspects of the system. This will enhance the overall understanding of the technique and its usage in a very practical manner.

**Summary**

This project report aims to acquaint the reader with the novel methods of search engine optimisation for driving more and more users to the website. We would use the aforementioned methods continuously and discover more and more users accessing our project website. In addition to it we aim to get sub links that are assigned by Google which is assigned to a website which has more number of users. Although the technique of SEO isn’t that new but it’s utility in the present times has increased a lot with the emergence and widespread acceptance of e- commerce.

**Limitations**

Everything in the technical world is transitory. Each day a new technology is known to the world. With this increases the competition. Unpaid SEO is all about constant effort to get the competition going, to indefinitely monitor and work as long as the site exists and to get the results. Hence, the biggest limitation is the all-time effort to maintain the rank, to be aware of all the new techniques in the market and to have a constant check on the competition.

**Future Scope**

As a future work we would to understand and develop tools which can add the site to a search

engine whenever user wants and can remove the sites which are not good. We would try to have the page rank of the site being improved. It would be a great task to have the site listed immediately after the paid-SEO sites.