**PROGRAM NO:12** **DATE :**

**AIM:** Implement Web scraping using python

## SOURCE CODE

```
!pip install autoscraper
from autoscraper import AutoScraper
url="https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/"
wanted_list=['Self-built Web Scrapers']
Scraper=AutoScraper()
result=Scraper.build(url,wanted_list)
print(result)
```

## OUTPUT:

['Web Scraping', 'crawler', 'Self-built Web Scrapers', 'Browser extensions Web Scrapers', 'Cloud Web Scrapers']

**PROGRAM NO:13** **DATE :**

**AIM:** Implement problem on Natural Language Processing-part of speech, tagging Ngram using NLTK.

## SOURCE CODE

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize,sent_tokenize
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
stop_words = set(stopwords.words('english'))
txt ="Hello. MCA S3 is fantastic. We learn many new concepts and implement them in
our practical exams. "
"Ist of all the data science is a new paper. "
tokenized= sent_tokenize(txt)
for i in tokenized:
  wordsList= nltk.word_tokenize(i)
  wordsList= [w for w in wordsList if not w in stop_words]
  tagged = nltk.pos_tag(wordsList)
  print(tagged)

def generate_N_grams(text,ngram=1):
  words=[word for word in text.split(" ") if word not in set(stopwords.words('english'))]
  print("Sentence after removing stopwords:",words)
  temp=zip(*[words[i:] for i in range(0,ngram)])
  ans=[".join(ngram) for ngram in temp]
  return ans
generate_N_grams("The sun rises in the east",2)
generate_N_grams("The sun rises in the east",3)
generate_N_grams("The sun rises in the east",4)
```

## OUTPUT:

```
[('Hello', 'NNP'), ('.', '.')]
```

[('MCA', 'NNP'), ('S3', 'NNP'), ('fantastic', 'JJ'), ('.', '.')]

[('We', 'PRP'), ('learn', 'VBP'), ('many', 'JJ'), ('new', 'JJ'), ('concepts', 'NNS'), ('implement', 'JJ'), ('practical', 'JJ'), ('exams', 'NN'), ('.', '.')]

[('Ist', 'NNP'), ('data', 'NNS'), ('science', 'NN'), ('new', 'JJ'), ('paper', 'NN'), ('.', '.')]

Sentence after removing stopwords: ['The', 'sun', 'rises', 'east']

['Thesun', 'sunrises', 'riseseast']

Sentence after removing stopwords: ['The', 'sun', 'rises', 'east']

['Thesunrises', 'sunriseseast']

Sentence after removing stopwords: ['The', 'sun', 'rises', 'east']

['Thesunriseseast']

**PROGRAM NO:11** **DATE :**

**AIM:** Program on CNN to classify images from any standard sataset in the public domain using the keras framework

## SOURCE CODE

```
from keras. datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
(X_train, y_train), (X_test, y_test)= mnist.load_data()
print("X_train shape",X_train. shape)
print("y_train shape", y_train.shape)
print("X_test shape", X_test. shape)
print("y_test shape", y_test.shape)
import matplotlib.pyplot as plt
plt.imshow(X_train[5], cmap=plt.cm.binary)
print(y_train[5])
X_train= X_train.reshape(60000, 784)
X_test= X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test= X_test.astype('float32')
X_train/= 255
X_test/=255
X_train.shape
n_classes = 10
Y_train= np_utils.to_categorical(y_train, n_classes)
Y_test=np_utils.to_categorical(y_test, n_classes)
model = Sequential()
model.add(Dense(100,input_shape=(784,), activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
model.compile(loss='categorical_crossentropy',metrics=['accuracy'], optimizer='adam')
model. fit(X_train, Y_train, batch_size=100, epochs=10)
test_loss, test_acc= model.evaluate(X_test, Y_test)
print("TEST ACCURACY",round(test_acc,3))
print("TEST LOSS",round(test_loss,3))
```

**OUTPUT:**

```
X_train shape (60000, 28, 28)
y_train shape (60000,)
X_test shape (10000, 28, 28)
y_test shape (10000,)
2
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 100)               78500

 dense_3 (Dense)             (None, 10)                1010

=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
600/600 [==============================] - 3s 4ms/step - loss: 0.3592 - accuracy:
0.9001
Epoch 2/10
600/600 [==============================] - 2s 4ms/step - loss: 0.1646 - accuracy:
0.9534
Epoch 3/10
600/600 [==============================] - 2s 4ms/step - loss: 0.1170 - accuracy:
0.9672
Epoch 4/10
600/600 [==============================] - 2s 4ms/step - loss: 0.0915 - accuracy:
0.9736
Epoch 5/10
600/600 [==============================] - 2s 4ms/step - loss: 0.0759 - accuracy:
0.9777
Epoch 6/10
```

600/600 [==============================] - 2s 4ms/step - loss: 0.0628 - accuracy: 0.9818
Epoch 7/10
600/600 [==============================] - 3s 5ms/step - loss: 0.0527 - accuracy: 0.9846
Epoch 8/10
600/600 [==============================] - 2s 4ms/step - loss: 0.0453 - accuracy: 0.9870
Epoch 9/10
600/600 [==============================] - 2s 4ms/step - loss: 0.0394 - accuracy: 0.9883
Epoch 10/10
600/600 [==============================] - 2s 4ms/step - loss: 0.0339 - accuracy: 0.9904
313/313 [==============================] - 1s 2ms/step - loss: 0.0828 - accuracy: 0.9748
TEST ACCURACY 0.975
TEST LOSS 0.083