```
import nltk
import re
from statistics import mode
inputfile='football_player.txt' #Location of the file
buf=open(inputfile,encoding='utf-8-sig') #file is loaded anc encoded in the specified format
list_of_doc=buf.read().split('\n')
print(list_of_doc)
new_string = []
for i in list_of_doc:
    if len(i) != 0:
        new_string.append(i) #removing empty lines in between the paragraph
list_of_doc = new_string

import nltk
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
def ie_preprocess(document):
  sentence = sent_tokenize(document)
  pos_sentences = []
  for i in sentence:
    text = word_tokenize(i)                #tokenizing the sentence
    pos_sentences.append(nltk.pos_tag(text)) #Part of Speech tag is determined for the each sentence
  return pos_sentences
inputfile='football_player.txt' #Location of the file
buf=open(inputfile,encoding='utf-8-sig') #file is loaded anc encoded in the specified format
list_of_doc=buf.read().split('\n')
print(list_of_doc)



new_string = []
for i in list_of_doc:
    if len(i) != 0:
```

```
        new_string.append(i) #removing empty lines in between the paragraph
list_of_doc = new_string
```

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
def ie_preprocess(document):
  sentence = sent_tokenize(document)
  pos_sentences = []
  for i in sentence:
    text = word_tokenize(i)                #tokenizing the sentence
    pos_sentences.append(nltk.pos_tag(text)) #Part of Speech tag is determined for the each sentence
  return pos_sentences
first_doc=list_of_doc[0]
pos_sent=ie_preprocess(first_doc)
pos_sent
```

Expected output [...[('He', 'PRP'), ('is', 'VBZ'), ('a', 'DT'), ('forward', 'NN'), ('and', 'CC'), ('serves', 'NNS'), ('as', 'IN'), ('captain', 'NN'), ('for', 'IN'), ('Portugal', 'NNP'), ('.', '.')], ...]

```
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
named_entities=[]
def named_entity_finding(pos_sent):
tree = nltk.ne_chunk(pos_sent,binary=True)    #checked only for binary value true
for subtree in tree.subtrees():
      if subtree.label() == 'NE':                #named entity
        entity = ""
        for leaf in subtree.leaves():
          entity = entity + leaf[0] + " "        #seperating enitity before space
        named_entities.append(entity.strip())
   return named_entities
named_entity_finding(pos_sent[0])
```

Expected Output:

['Cristiano Ronaldo',
'Santos Aveiro',
'ComM',
'GOIH',
'Portuguese',
'Spanish',
'Real Madrid',
'Portugal']

```
import itertools


def NE_flat_list_fn(pos_sent):
    NE=[]
    for i in pos_sent:
      new_pos = ie_preprocess(i)                    #pos tag each sentence
    for pos in new_pos:
      entity = named_entity_finding(pos)
      NE.append(entity)
    NE_flat_list = list(itertools.chain.from_iterable(NE)) #determine the named enitity of the sentence after flattening the list



    return NE_flat_list


new_list=NE_flat_list_fn(list_of_doc)
```

```
import re,nltk
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
def name_of_the_player(doc):
  sentence = sent_tokenize(doc)[0]
  name=re.compile(r'^(\w.+)(?=\s+.born)')              #regex for print those string that is before "born"
  regex_name=name.match(sentence)[0].split(",")[0]


  return regex_name
def country_of_origin(doc):
  country=[]
  origin=re.compile(r'((?:[\S,]+\s+){0,1})national team')  #regex for the string before "national team"
```

```
  for i,sent in enumerate(sent_tokenize(doc)):
    team = origin.findall(sent)                    #findall function for each word
    if len(team)!=0:
      country.append(team[0])                      #if not empty then append in country
  return country
```

def date_of_birth(doc):

```
  born=[]
  sentence = sent_tokenize(doc)[0]
  DOB = re.compile(r'born\b\s*((?:\S+\s+){0,3})')          #regex for string after born and of size 3
date,month,year]
  birth = DOB.findall(sentence)[0]
  birth = re.sub('\W+',' ', birth )


  return birth
```

def team_of_the_player(doc):

```
  sentence = sent_tokenize(doc)
  new = []
  named_team = []
  national = []
  named=[]


  pos_sent = ie_preprocess(doc)
  for i, sent in enumerate(sentence):
    if i == 2:
      break


    match = re.compile(r'((?:[\S,]+\s+){0,1})national team')       #string before national team
    team_player = re.compile(r'club\s+((?:[\S,]+\s*){0,2})')       #after club
    team = match.findall(sent)
    if len(team_player .findall(sent)) != 0:
      for f in team_player .findall(sent):
        national.append(f)                                  #append in national
```

```
    tree = nltk.ne_chunk(pos_sent[i],binary=False)                #there is also string after organization that should be used to determine the desired output
    for subtree in tree.subtrees():
      if subtree.label() == 'ORGANIZATION':
        entity = ""
        for leaf in subtree.leaves():
          entity = entity + leaf[0] + " "
        named.append(entity.strip())
    named_team.append(named)                                #append in national team
    if len(team) != 0:
      l = team[0]+"national team"
      new.append(l)

  if len(list(set(new))) != 0:
    national_team = list(set(new))[0]
  else:
    national_team = country_of_origin(doc)                    #to determine the country

  named_team = list(itertools.chain.from_iterable(named_team))

  new_club = []
  for i in national:
    new_club.extend(nltk.word_tokenize(i))  l
  for i, s in enumerate(national):
    national[i] = national[i].rstrip()
  if len(list(set(named_team).intersection(national))) != 0:
    named_team = list(set(named_team).intersection(national))   #common words are stored in named_team

  if len(list(set(new))) == 0:
    named_team.append(national_team[0]+ " national team")
  else:
    named_team.append(national_team)
```

```
  named_team = list(set(named_team).difference(nltk.word_tokenize(sentence[0])[0:6]))


  return named_team
```

def position_of_the_player(doc):

```
  position = ["forward", "captain", "attacking midfielder", "striker", "winger", "central midfielder",
"defensive tackle", "defensive end"]
  player_position=[]    #store the various positions in variable position
  player=[]
  sent = sent_tokenize(doc)
  for i, sent in enumerate(sent):
    for j in position:
      player_position = re.compile(r'\b({0})\b'.format(j)) #regex used to determine
      value = bool(player_position.search(sent))          #boolean is true then they are appended from the
string
      if value == True:
        player.append(j)
  return list(set(player))
print(name_of_the_player(list_of_doc[0]))
```

Expected Output: Cristiano Ronaldo dos Santos Aveiro

```
import json
def generate_jsonld(arg):
  soccerld = { "@id": "http://my-soccer-ontology.com/footballer/"+arg[0],


          "name": arg[0],
          "born": arg[1],
          "country": arg[2],
          "position": [
             { "@id": "http://my-soccer-ontology.com/position/",
                "type": arg[3]
             }
           ],
           "team": [
             { "@id": "http://my-soccer-ontology.com/team/",
                "name": arg[4]
```

```
            }
          ]
      }

  return json.dumps(soccerld)

   #Code goes here

   #Hint: arg1,arg2,..... are the arguments you will be passing to the function

def arg_function(doc):

  arg = [name_of_the_player(doc), date_of_birth(doc), country_of_origin(doc),
position_of_the_player(doc), team_of_the_player(doc)]

  return arg

arg=arg_function(list_of_doc[0])

generate_jsonld(arg)
```

Expected Output: '{"@id": "http://my-soccer-ontology.com/footballer/Cristiano Ronaldo dos Santos Aveiro", "name": "Cristiano Ronaldo dos Santos Aveiro", "born": "5 February 1985 ", "country": ["Portugal "], "position": [{"@id": "http://my-soccer-ontology.com/position/", "type": ["captain", "forward"]}], "team": [{"@id": "http://my-soccer-ontology.com/team/", "name": ["Real Madrid", "Portugal national team"]}]}'