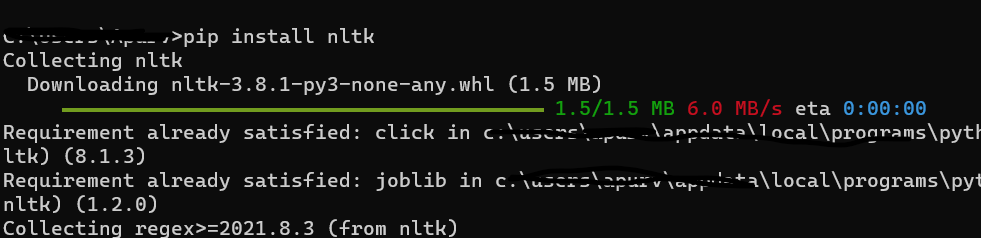
**Natural Language Processing**

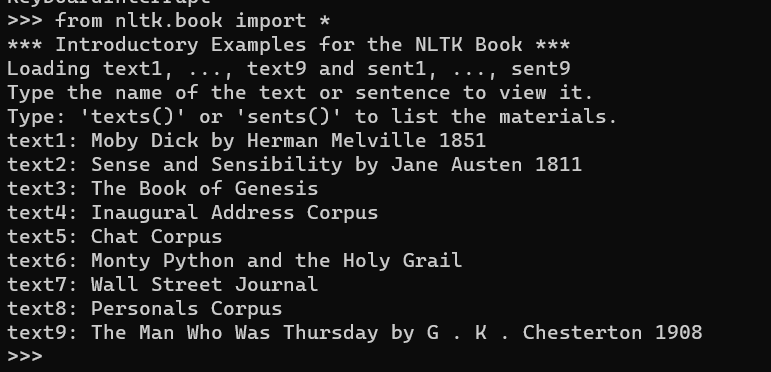
## Chapter 1 : Language Processing with Python

* Getting started with python
* Learn to use python interpreter
* Getting started with NLTK
* Download NLTK – command prompt – “pip install nltk”
* In python interpreter :
* import nltk
* nltk.download() – select book



A screenshot of a computer

Description automatically generated

* as above download some data on the machine and then load some texts for exploration using python interpreter.
* 

**>>> from nltk.book import \***

**\*\*\* Introductory Examples for the NLTK Book \*\*\***

**Loading text1, ..., text9 and sent1, ..., sent9**

**Type the name of the text or sentence to view it.**

**Type: 'texts()' or 'sents()' to list the materials.**

**text1: Moby Dick by Herman Melville 1851**

**text2: Sense and Sensibility by Jane Austen 1811**

**text3: The Book of Genesis**

**text4: Inaugural Address Corpus**

**text5: Chat Corpus**

**text6: Monty Python and the Holy Grail**

**text7: Wall Street Journal**

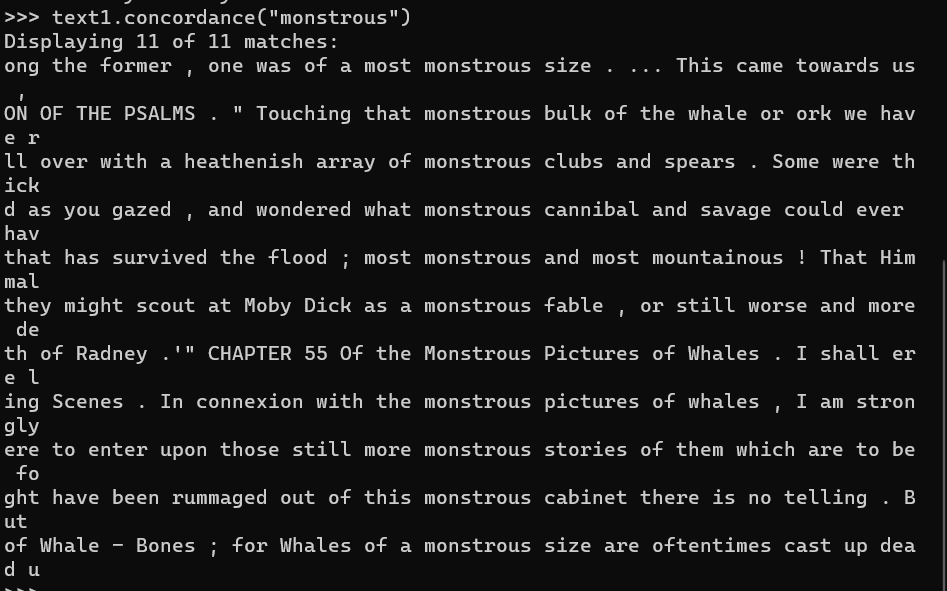
**text8: Personals Corpus**

**text9: The Man Who Was Thursday by G . K . Chesterton 1908**

**>>>**

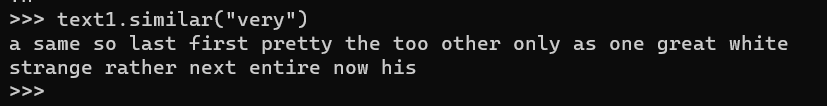
* Text1
* **Searching text**
* A concordance view shows us every occurrence of a given word, together with some context.

**>>> text1.concordance("monstrous")**



A concordance permits us to see words in context.

* We can find out appending the term like the name of the text in question, then inserting relevant word in parentheses

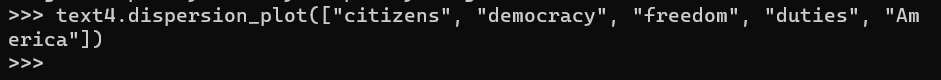


* The term **common\_contexts** allows us to examine just the contexts that are shared by two or more words, such as monstrous and very. We must enclose these words by square brackets as well as parentheses, and separate them with a comma

A black background with white text

Description automatically generated

* We can also determine location of a word in the text, this positional information can be displayed using dispersion plot.



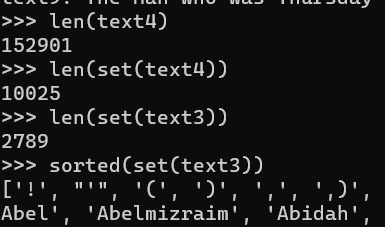
* We can also generate random text

A screenshot of a computer

Description automatically generated

* Counting vocabulary:
* We use len() to get the length of something.
* len(text3)
* token is the technical name for a sequence of characters
* the vocabulary of a text is just the set of tokens that it uses, since in a set, all duplicates are collapsed together.
* Len(set(text1))
* len(set(text2))
* we can also get sorted list of vocabulary items

sorted(set(text3))



* a word type is the form or spelling of the word independently of its specific occurrences in a text- that’s the word considered as unique item of vocabulary.
* To calculate lexical richness of the text, how many times a word is used.
* We can count how often a word occurs in a text, & compute what % of the text is taken up by a specific word.

A computer screen with white text

Description automatically generated

* Define functions to find lexical diversity & percentage for text.
* A computer screen with white text

  Description automatically generated
* Texts as list of words:
* Text = sequence of words and punctuation.
* Sent1 = [‘call’, ‘me’, ‘Ishrael’,’’.’]
* We store a text in list.
* Adding two lists creates a new list with everything from the first list.
* A black background with white text

  Description automatically generated
* This special operation is called concatenation; it combines the lists together into a single list.
* We can add single item to a list using append():

A black background with white text

Description automatically generated

* Indexing Lists:
* We can use indexing to get the required word.
* We can also extract manageable pieces of language from texts, a technique known as slicing.

A computer screen shot of a black background

Description automatically generated

A black background with white text

Description automatically generated

* Variables:
* Variables = expression
* Python evaluates the expression and save its result to the variable. This process is called “assignment”.
* Capitalized words appear before lowercase words in sorted lists.
* Strings:
* Some of the methods we used to access the elements of a list also work with individual words, or strings.
* We can also perform multiplication and addition with strings.

**name = 'Monty'**

**name[:4]**

**name \* 2**

**-'MontyMonty'**

**>>> ' '.join(['Monty', 'Python'])**

**'Monty Python'**

**>>> 'Monty Python'.split()**

**['Monty', 'Python']**

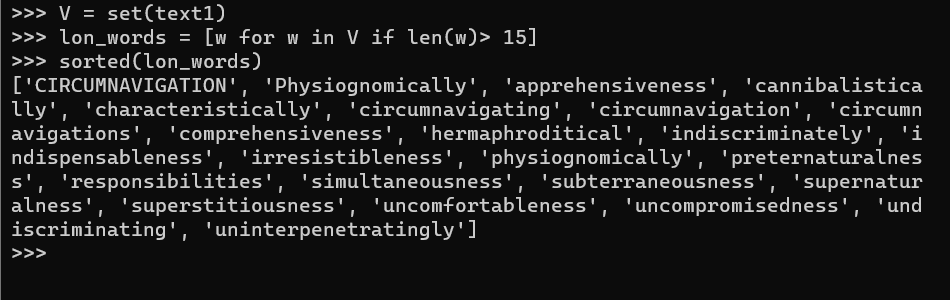
* Computing with language : simple statistics
* Can be explained as below saying example:
* Frequency Distributions:
* To identify the words of a text that are more informative about the topic & genre of the text, we can use frequency distribution for that.
* We use **FreqDist()**
* Vocabulary = fdist1.keys()

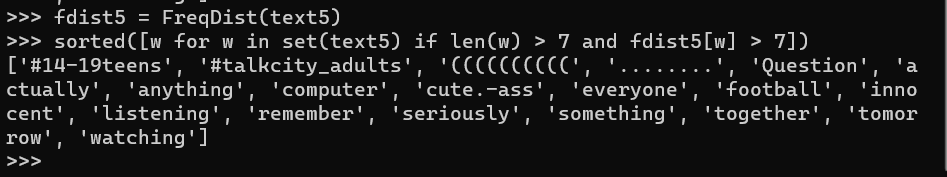
Here **keys()** gives a list of all the distinct types in the text.

A computer screen with white text

Description automatically generated

* Fine-grained selection of words:
* We want to find words from the vocabulary of the text that are more than 15 characters long.
* P(w) is true if and only if w is more than 15 characters long.
* The set of all w such that w is an element of V (vocabulary) and w has property P.



* Hapaxes(unique)
* It is better to find frequently occurring long-words. This seems promising since it eliminates frequent short words & infrequent long words.
* 
* Here above we have use two conditions: len(w) > 7 and fdist5[w]>7
* Collocations and Bigrams:
* A collocation is sequence of words that occur together usually often. Thus “red wine” is a collocation, whereas the wine is not. A characteristic of collocations is that they are resistant to substitution with words that have similar senses.
* 
* A list of word pairs, also known as bigrams.
* Collocations are generally frequent bigrams. The collocations() function can find that.

A screen shot of a computer

Description automatically generated

A black background with white text

Description automatically generated

* We can find the count and the frequency distribution of the words.

