Types of AI:

The three kinds of AI based on capabilities

1. Artificial Narrow AI

Artificial Narrow Intelligence, also known as Weak AI (what we refer to as Narrow AI), is the only type of AI that exists today. Any other form of AI is theoretical. It can be trained to perform a single or narrow task, often far faster and better than a human mind can.

However, it can't perform outside of its defined task. Instead, it targets a single subset of cognitive abilities and advances in that spectrum. Siri, Amazon's Alexa and IBM Watson® are examples of Narrow AI. Even OpenAI's ChatGPT is considered a form of Narrow AI because it's limited to the single task of text-based chat.

2. General AI

Artificial General Intelligence (AGI), also known as Strong AI, is today nothing more than a theoretical concept. AGI can use previous learnings and skills to accomplish new tasks in a different context without the need for human beings to train the underlying models. This ability allows AGI to learn and perform any intellectual task that a human being can.

3. Super AI

Super AI is commonly referred to as artificial superintelligence and, like AGI, is strictly theoretical. If ever realized, Super AI would think, reason, learn, make judgements and possess cognitive abilities that surpass those of human beings.

The applications possessing Super AI capabilities will have evolved beyond the point of understanding human sentiments and experiences to feel emotions, have needs and possess beliefs and desires of their own.

https://www.ibm.com/think/topics/artificial-intelligence-types

File Handling :

The key function for working with files in Python is the open() function.

The open() function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file.

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

To open a file for reading it is enough to specify the name of the file.

```
f = open("demofile.txt")
```

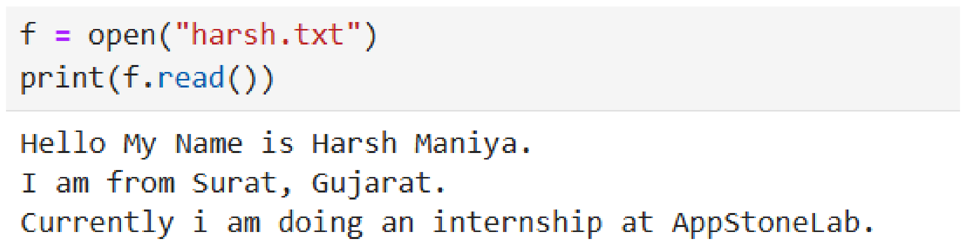
```
f = open("demofile.txt", "rt")
```

Both perform the same task.

Open a File :

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file.



```
f = open("harsh.txt")
print(f.read())
```

```
Hello My Name is Harsh Maniya.
I am from Surat, Gujarat.
Currently i am doing an internship at AppStoneLab.
```

Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return.

```
f = open("harsh.txt")
print(f.read(19))
```

```
Hello My Name is Ha
```

Read Lines

You can return one line by using the readline() method.

```
h = open('harsh.txt')
print(h.readline())
```

```
Hello My Name is Harsh Maniya.
```

```
h = open('harsh.txt')
print(h.readline())
print(h.readline())
```

```
Hello My Name is Harsh Maniya.

I am from Surat, Gujarat.
```

```
h = open('harsh.txt')
for i in h:
    print(i)
```

```
Hello My Name is Harsh Maniya.

I am from Surat, Gujarat.

Currently i am doing an internship at AppStoneLab.
```

Close Files

It is a good practice to always close the file when you are done with it.

```
m = open('harsh.txt','r')
print(m.readline())
m.close()
```

```
Hello My Name is Harsh Maniya.
```

Python File Write

Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

- "a" - Append - will append to the end of the file
- "w" - Write - will overwrite any existing content

```
a = open('harsh.txt','a')
a.write(' It will last 3 months')
a.close()

a = open('harsh.txt','r')
print(a.read())
```

```
Hello My Name is Harsh Maniya.
I am from Surat, Gujarat.
Currently i am doing an internship at AppStoneLab. It will last 3 months
```

```
b = open('harsh.txt','w')
b.write('Overwritten.')
b.close()

b = open('harsh.txt','r')
print(b.read())
```

```
Overwritten.
```

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

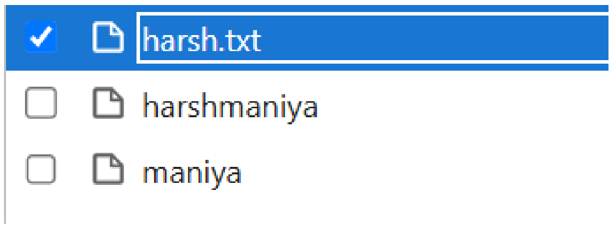
"x" - Create - will create a file, returns an error if the file exists

"a" - Append - will create a file if the specified file does not exists

"w" - Write - will create a file if the specified file does not exists

```
f = open('maniya','x')
```

```
f = open('harshmaniya','w')
```



harsh.txt
harshmaniya
maniya

Delete a File

To delete a file, you must import the OS module, and run its os.remove() function.

```
import os
os.remove('maniya')
```

```
import os
if os.path.exists('maniya'):
    os.remove('maniya')
else:
    print('File does not exists.')
```

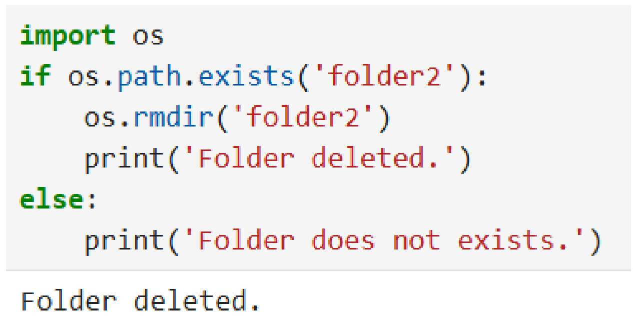
```
File does not exists.
```

```
import os
if os.path.exists('harshmaniya'):
    os.remove('harshmaniya')
    print('File deleted.')
else:
    print('File does not exists.')
```

```
File deleted.
```

Delete Folder

To delete an entire folder, use the os.rmdir() method.



```
import os
if os.path.exists('folder2'):
    os.rmdir('folder2')
    print('Folder deleted.')
else:
    print('Folder does not exists.')
```

```
Folder deleted.
```

Natural Language ToolKit (NLTK)

The **Natural Language Toolkit (NLTK)** is a powerful Python programming environment for statistical natural language processing (NLP). It provides libraries for tasks like tokenization, parsing, classification, stemming, labeling, and semantic reasoning. NLTK also includes educational resources, such as a curriculum and a companion book, which outline common NLP tasks with visual demonstrations and experimental datasets.

NLTK is one of the most versatile NLP libraries, offering tools that enable computers to understand and respond to natural language. It supports multiple languages, including Arabic, Chinese, Dutch, French, German, Hindi, Italian, Japanese, Portuguese, Russian, and Spanish, among others. Features like tokenization, stemming, and morphological analysis are available for these languages.

In addition to standard NLP tasks, NLTK provides tools for sentiment analysis, making it useful for applications like social media monitoring and product review analysis. While powerful on its own, NLTK can also integrate with machine learning libraries like scikit-learn and TensorFlow, allowing for sophisticated NLP applications, such as deep learning-based language modeling.

NLTK benefits from a large and active community of users and contributors, offering abundant resources like tutorials, example code, and online forums. This makes it an excellent choice for both beginners and experienced practitioners in the NLP domain.

Tokenization

- **Purpose**: Splitting text into smaller units like words or sentences for further processing.
- **Methods**:
  - `word_tokenize()`: Tokenizes a string into words.
  - `sent_tokenize()`: Tokenizes a string into sentences.
- **Real-World Use**:
  - Preprocessing text for sentiment analysis or chatbots.
  - Segmenting text in search engine pipelines.

```
from nltk.tokenize import word_tokenize, sent_tokenize
```

```
text = "My name is Harsh Maniya. I live in Surat. I am pursuing my B.Tech in AI-DS. Currently i am doing an internship at AppStoneLab. "
```

```
print(sent_tokenize(text))
print(word_tokenize(text))
```

```
['My name is Harsh Maniya.', 'I live in Surat.', 'I am pursuing my B.Tech in AI-DS.', 'Currently i am doing an internship at AppStoneLab.']
['My', 'name', 'is', 'Harsh', 'Maniya', '.', 'I', 'live', 'in', 'Surat', '.', 'I', 'am', 'pursuing', 'my', 'B.Tech', 'in', 'AI-DS', '.', 'Currently',
'i', 'am', 'doing', 'an', 'internship', 'at', 'AppStoneLab', '.']
```

Stopword Removal

- **Purpose**: Removes common, less meaningful words (e.g., "is", "the") to focus on important words.
- **Method**:
  - `nltk.corpus.stopwords.words(language)`
- **Real-World Use**:
  - Text summarization.
  - Improving the accuracy of classification algorithms by removing noise.

```
text = """My name is Harsh Maniya.
          I live in Surat. I am pursuing my B.Tech in AI-DS.
          Currently i am doing an internship at AppStoneLab."""
```

```
tokenized = word_tokenize(text)
stop_words = set(stopwords.words('english'))
```

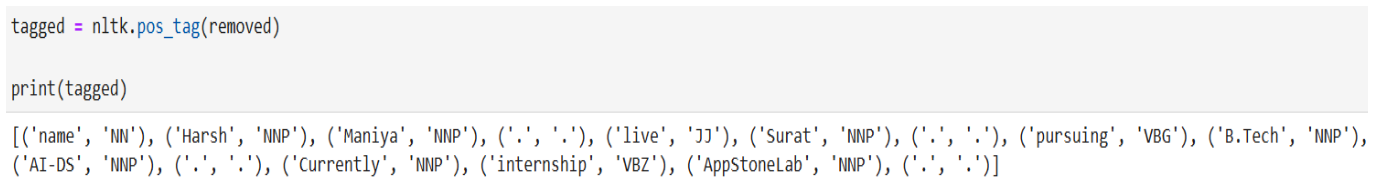
```
def remove_stopwords(tokenized):
    removed = []
    for word in tokenized:
        if word.lower() not in stop_words:
            removed.append(word)
    return removed
```

```
remove_stopwords(tokenized)
```

```
['name',
 'Harsh',
 'Maniya',
 '.',
 'live',
 'Surat',
 '.',
 'pursuing',
 'B.Tech',
 'AI-DS',
 '.',
 'Currently',
 'internship',
 'AppStoneLab',
 '.']
```

Part-of-Speech (POS) Tagging

- **Purpose**: Assigns grammatical tags (e.g., noun, verb) to each word in a sentence.
- **Method**:
  - `pos_tag()`
- **Real-World Use**:
  - Analyzing text for grammatical errors.
  - Improving keyword extraction and named entity recognition systems.



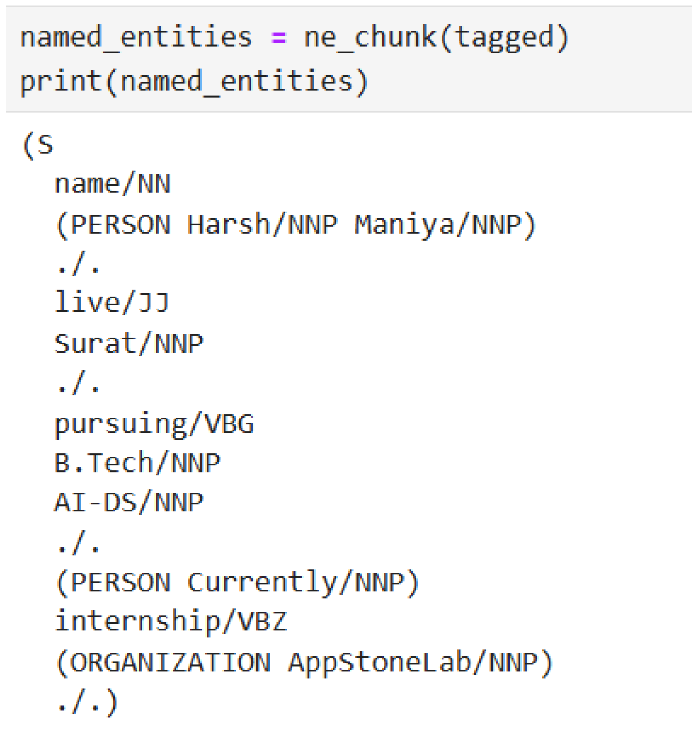
```
tagged = nltk.pos_tag(removed)

print(tagged)
```

```
[('name', 'NN'), ('Harsh', 'NNP'), ('Maniya', 'NNP'), ('.', '.'), ('live', 'JJ'), ('Surat', 'NNP'), ('.', '.'), ('pursuing', 'VBG'), ('B.Tech', 'NNP'),
('AI-DS', 'NNP'), ('.', '.'), ('Currently', 'NNP'), ('internship', 'VBZ'), ('AppStoneLab', 'NNP'), ('.', '.')]
```

Named Entity Recognition (NER)

- **Purpose**: Identifies entities like names, organizations, and locations in text.
- **Method**:
  - `nltk.ne_chunk()`
- **Real-World Use**:
  - Extracting names or organizations from news articles.
  - Building information extraction systems for resumes or legal documents.



```
named_entities = ne_chunk(tagged)
print(named_entities)
```

```
(S
  name/NN
  (PERSON Harsh/NNP Maniya/NNP)
  ./.
  live/JJ
  Surat/NNP
  ./.
  pursuing/VBG
  B.Tech/NNP
  AI-DS/NNP
  ./.
  (PERSON Currently/NNP)
  internship/VBZ
  (ORGANIZATION AppStoneLab/NNP)
  ./.)
```

Stemming

- **Purpose**: Reduces words to their root forms by removing prefixes or suffixes.
- **Methods**:
  - `PorterStemmer().stem()`
  - `LancasterStemmer().stem()`
- **Real-World Use**:
  - Search engines match related words like "run" and "running."
  - Keyword indexing systems.

```
from nltk.stem import PorterStemmer
```

```
ps = PorterStemmer()
```

```
removedddd = ["program", "programming", "programmer", "programs", "programmed"]
```

```
stemmed = [ps.stem(word) for word in removedddd]
```

```
print(stemmed)
```

```
['program', 'program', 'programm', 'program', 'program']
```

Lemmatization

- **Purpose**: Converts words to their base or dictionary form while considering the word's context.
- **Method**:
  - `WordNetLemmatizer().lemmatize()`
- **Real-World Use**:
  - Applications where preserving context is crucial, like grammar checking or semantic analysis.

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
```

```
words = 'foxes jumping dogs corpora rides caring'

tokenizedddd = word_tokenize(words)
```

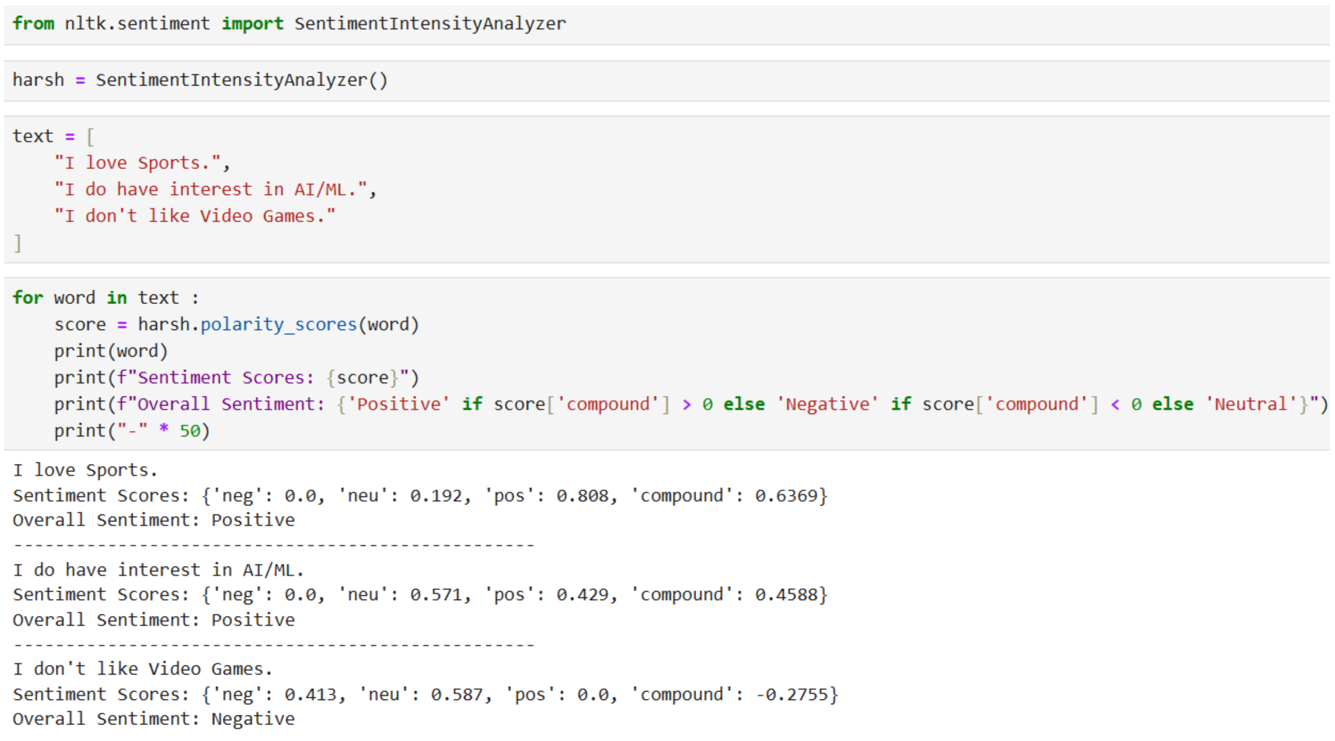
```
lemmatized = [lemmatizer.lemmatize(word) for word in tokenizedddd]
```

```
print(lemmatized)
```

```
['fox', 'jumping', 'dog', 'corpus', 'ride', 'caring']
```

Sentiment Analysis

- **Purpose**: Analyzes the sentiment (positive, negative, or neutral) of text.
- **Tools**:
  - Tokenization combined with classifiers (like Naive Bayes) or pretrained models.
- **Real-World Use**:
  - Social media monitoring.
  - Customer feedback and review analysis.



```python
from nltk.sentiment import SentimentIntensityAnalyzer
```

```python
harsh = SentimentIntensityAnalyzer()
```

```python
text = [
    "I love Sports.",
    "I do have interest in AI/ML.",
    "I don't like Video Games."
]
```

```python
for word in text :
    score = harsh.polarity_scores(word)
    print(word)
    print(f"Sentiment Scores: {score}")
    print(f"Overall Sentiment: {'Positive' if score['compound'] > 0 else 'Negative' if score['compound'] < 0 else 'Neutral'}")
    print("-" * 50)
```

```
I love Sports.
Sentiment Scores: {'neg': 0.0, 'neu': 0.192, 'pos': 0.808, 'compound': 0.6369}
Overall Sentiment: Positive
--------------------------------------------------
I do have interest in AI/ML.
Sentiment Scores: {'neg': 0.0, 'neu': 0.571, 'pos': 0.429, 'compound': 0.4588}
Overall Sentiment: Positive
--------------------------------------------------
I don't like Video Games.
Sentiment Scores: {'neg': 0.413, 'neu': 0.587, 'pos': 0.0, 'compound': -0.2755}
Overall Sentiment: Negative
```

Numpy :

What is Numpy ?

NumPy stands for Numerical Python. It is an Open Source Python library used for working with arrays. NumPy was created in 2005 by Travis Oliphant. It has functions for working in the domain of linear algebra, Fourier transform, and matrices.

Advantages of Numpy over Python List

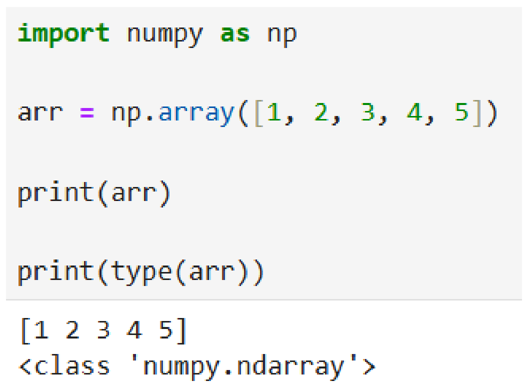
NumPy provides an array object (ndarray) that is up to 70x faster than traditional Python lists. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.This behavior is called locality of reference.This is the main reason why NumPy is faster than lists.

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() function.



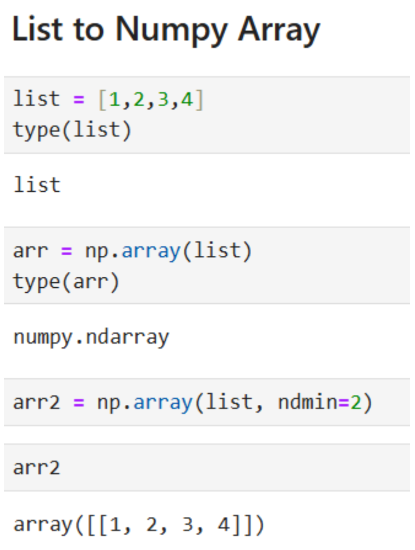
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```



List to Numpy Array

```
list = [1,2,3,4]
type(list)
```

```
list
```

```
arr = np.array(list)
type(arr)
```

```
numpy.ndarray
```

```
arr2 = np.array(list, ndmin=2)
```

```
arr2
```

```
array([[1, 2, 3, 4]])
```

Arrange Function

```
np.arange(1,10)
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

MultiDimensional Array

```
arr = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
```

```
arr
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

Check Number of Dimensions?

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

Ndim Function

```
arr.ndim
```

```
2
```

```
arr2.ndim
```

```
1
```

Higher Dimensional Arrays

An array can have any number of dimensions.

When the array is created, you can define the number of dimensions by using the ndmin argument.

```
arr2 = np.array(list, ndmin=5)
arr2
```

```
array([[[[[1, 2, 3, 4]]]]])
```

Access Array Elements

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

Use negative indexing to access an array from the end.

```
arr = np.array([1, 2, 3, 4])
print(arr[1])
```

```
2
```

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st row: ', arr[0, 1])
```

```
2nd element on 1st row:  2
```

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

```
6
```

```
print('Last element from 2nd dim: ', arr[1, -1])
```

```
Last element from 2nd dim:  10
```

Slicing arrays

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: [start:end].

We can also define the step, like this: [start:end:step].

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1.

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
print(arr[4:])
print(arr[:4])
```

```
[2 3 4 5]
[5 6 7]
[1 2 3 4]
```

Negative Slicing

Use the minus operator to refer to an index from the end.

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])
print(arr[1:5:2])
print(arr[::2])
```

```
[5 6]
[2 4]
[1 3 5 7]
```

Checking the Data Type of an Array

The NumPy array object has a property called dtype that returns the data type of the array.

dtype function

```
arr.dtype
```

```
dtype('int32')
```

```
arr2 = np.array([1.3,4.7,34.1,67.1])
```

```
arr2.dtype
```

```
dtype('float64')
```

Converting Data Type on Existing Arrays

The best way to change the data type of an existing array, is to make a copy of the array with the astype() method.

The astype() function creates a copy of the array, and allows you to specify the data type as a parameter.

The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or you can use the data type directly like float for float and int for integer.

```
arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')

print(newarr)
print(newarr.dtype)
```

```
[1 2 3]
int32
```

Get the Shape of an Array

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

```
arr.shape
```

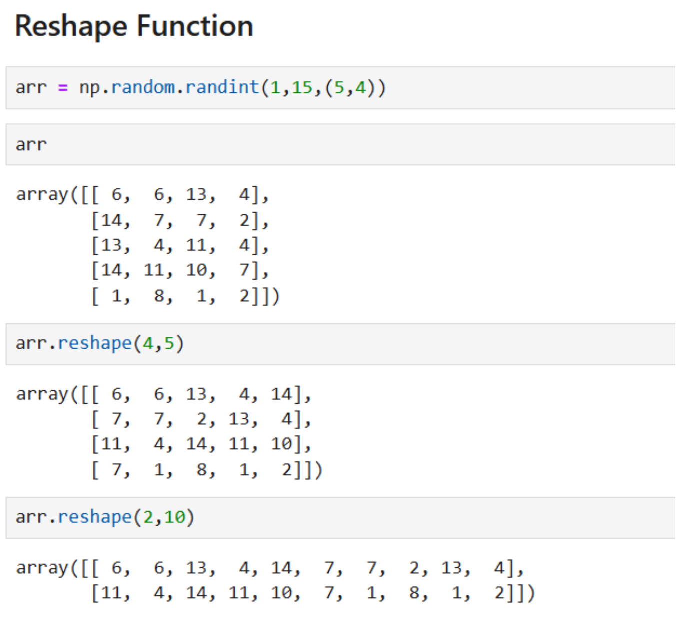
```
(4, 3)
```

Reshaping arrays

Reshaping means changing the shape of an array.

The shape of an array is the number of elements in each dimension.

By reshaping we can add or remove dimensions or change number of elements in each dimension.



Reshape Function

```
arr = np.random.randint(1,15,(5,4))
```

```
arr
```

```
array([[ 6,  6, 13,  4],
       [14,  7,  7,  2],
       [13,  4, 11,  4],
       [14, 11, 10,  7],
       [ 1,  8,  1,  2]])
```

```
arr.reshape(4,5)
```

```
array([[ 6,  6, 13,  4, 14],
       [ 7,  7,  2, 13,  4],
       [11,  4, 14, 11, 10],
       [ 7,  1,  8,  1,  2]])
```

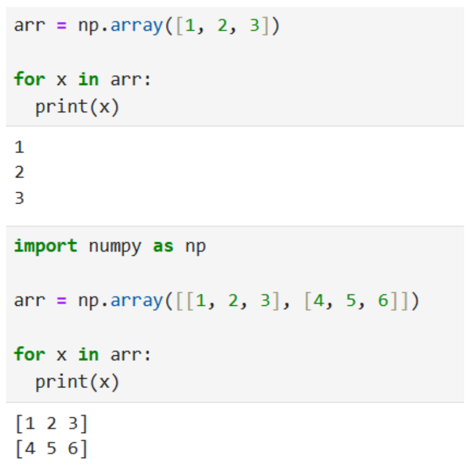
```
arr.reshape(2,10)
```

```
array([[ 6,  6, 13,  4, 14,  7,  7,  2, 13,  4],
       [11,  4, 14, 11, 10,  7,  1,  8,  1,  2]])
```

Iterating Arrays

Iterating means going through elements one by one.

As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python.



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

for x in arr:
  print(x)
```

```
1
2
3
```

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
  print(x)
```

```
[1 2 3]
[4 5 6]
```

Enumerated Iteration Using ndenumerate()

Enumeration means mentioning sequence number of somethings one by one.

Sometimes we require corresponding index of the element while iterating, the ndenumerate() method can be used for those usecases.

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

for idx, x in np.ndenumerate(arr):
  print(idx, x)
```

```
(0,) 1
(1,) 2
(2,) 3
```

Joining NumPy Arrays

Joining means putting contents of two or more arrays in a single array.

In SQL we join tables based on a key, whereas in NumPy we join arrays by axes.

We pass a sequence of arrays that we want to join to the concatenate() function, along with the axis. If axis is not explicitly passed, it is taken as 0.

```
arr = np.random.randint(1,50,(3,3))
arr2 = np.random.randint(1,100,(3,3))

ar5 = np.concatenate((arr,arr2),0)
```

```
ar5
```

```
array([[24, 32,  3],
       [45, 49, 36],
       [43, 49, 44],
       [39, 84, 78],
       [37, 26, 31],
       [63, 31, 82]])
```

Sorting Arrays

Sorting means putting elements in an ordered sequence.

Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a specified array.

```
np.sort(arr2,0)
```

```
array([[ 2,  1,  5,  1],
       [10,  9,  7,  1],
       [11,  9, 10,  2],
       [11, 12, 13,  5],
       [14, 14, 14, 10]])
```

```
ar5 = arr2
```

```
np.sort(ar5,kind='mergesort')
```

```
array([[ 1,  5, 10, 14],
       [ 2, 10, 12, 13],
       [ 1,  2, 14, 14],
       [ 1,  9, 10, 11],
       [ 5,  7,  9, 11]])
```

Filtering Arrays

Getting some elements out of an existing array and creating a new array out of them is called filtering.

In NumPy, you filter an array using a boolean index list.

```
arr = np.random.randint(1,50,10)
```

```
arr
```

```
array([16, 26, 12, 46, 22, 36, 45, 47,  6, 22])
```

```
print(arr[arr>20])
```

```
[26 46 22 36 45 47 22]
```

Generate Random Number

NumPy offers the random module to work with random numbers.

The random module's rand() method returns a random float between 0 and 1.

The randint() method takes a size parameter where you can specify the shape of an array.

```
np.random.rand(3,5)
```

```
array([[0.6020039 , 0.28131643, 0.38828615, 0.34578559, 0.84516989],
       [0.12075078, 0.95843193, 0.01710649, 0.58053048, 0.84111537],
       [0.86015426, 0.36213258, 0.07581099, 0.46654113, 0.79665434]])
```

```
np.random.randint(1,1000,5)
```

```
array([218,  18,  48,  27, 299])
```

```
np.random.randn(10)
```

```
array([-0.49756012,  0.01953276, -0.46028472,  0.83638821,  1.10319109,
        0.97988431,  0.14999249,  1.03148145,  0.8291002 ,  1.24573555])
```

NumPy ufuncs

What are ufuncs?

ufuncs stands for "Universal Functions" and they are NumPy functions that operate on the ndarray object.

Why use ufuncs?

ufuncs are used to implement vectorization in NumPy which is way faster than iterating over elements.

They also provide broadcasting and additional methods like reduce, accumulate etc. that are very helpful for computation.

ufuncs also take additional arguments, like:

where boolean array or condition defining where the operations should take place.

dtype defining the return type of elements.

out output array where the return value should be copied.

How To Create Your Own ufunc

To create your own ufunc, you have to define a function, like you do with normal functions in Python, then you add it to your NumPy ufunc library with the frompyfunc() method.

The frompyfunc() method takes the following arguments:

- function - the name of the function.
- inputs - the number of input arguments (arrays).
- outputs - the number of output arrays.

```
def myadd(x, y):
  return x+y

myadd = np.frompyfunc(myadd, 2, 1)

print(myadd([1, 2, 3, 4], [5, 6, 7, 8]))
```

```
[6 8 10 12]
```

Check if a Function is a ufunc

Check the type of a function to check if it is a ufunc or not.

A ufunc should return <class 'numpy.ufunc'>.

```
print(type(np.add))
```

```
<class 'numpy.ufunc'>
```