**NUMPY MODULE**

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy stands for Numerical Python.

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

The source code for NumPy is located at this github repository <https://github.com/numpy/numpy>

POWERFUL N-DIMENSIONAL ARRAYS

Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.

NUMERICAL COMPUTING TOOLS

NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more

INTEROPERABLE

NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.

PERFORMANT

The core of NumPy is well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.

EASY TO USE

NumPy’s high level syntax makes it accessible and productive for programmers from any background or experience level.

OPEN SOURCE

Distributed under a liberal [BSD license](https://github.com/numpy/numpy/blob/main/LICENSE.txt), NumPy is developed and maintained [publicly on GitHub](https://github.com/numpy/numpy) by a vibrant, responsive, and diverse [community](https://numpy.org/community).

Installation of NumPy

If you have [Python](https://www.w3schools.com/python/default.asp) and [PIP](https://www.w3schools.com/python/python_pip.asp) already installed on a system, then installation of NumPy is very easy.

Install it using this command:

C:\Users\ >pip install numpy

## Import NumPy

Once NumPy is installed, import it in your applications by adding the import keyword:

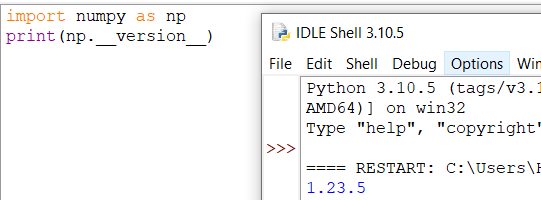
## NumPy as np

NumPy is usually imported under the np alias.

**alias:** In Python alias are an alternate name for referring to the same thing.

## Checking NumPy Version

The version string is stored under \_\_version\_\_ attribute.



# **NumPy Creating Arrays**

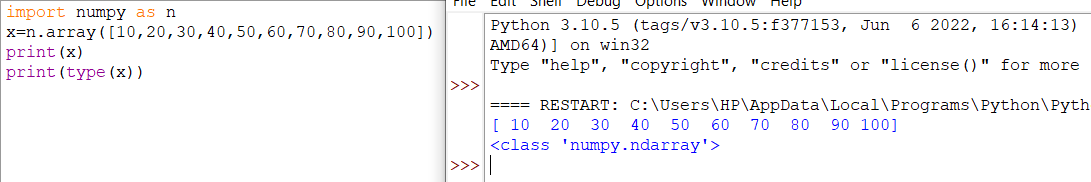
## Create a NumPy ndarray Object

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.

**type():** This built-in Python function tells us the type of the object passed to it. Like in above code it shows that arr is numpy.ndarray type.

To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:



## Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays).

**nested array:** are arrays that have arrays as their elements.

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

## 0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

## 1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

## 2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

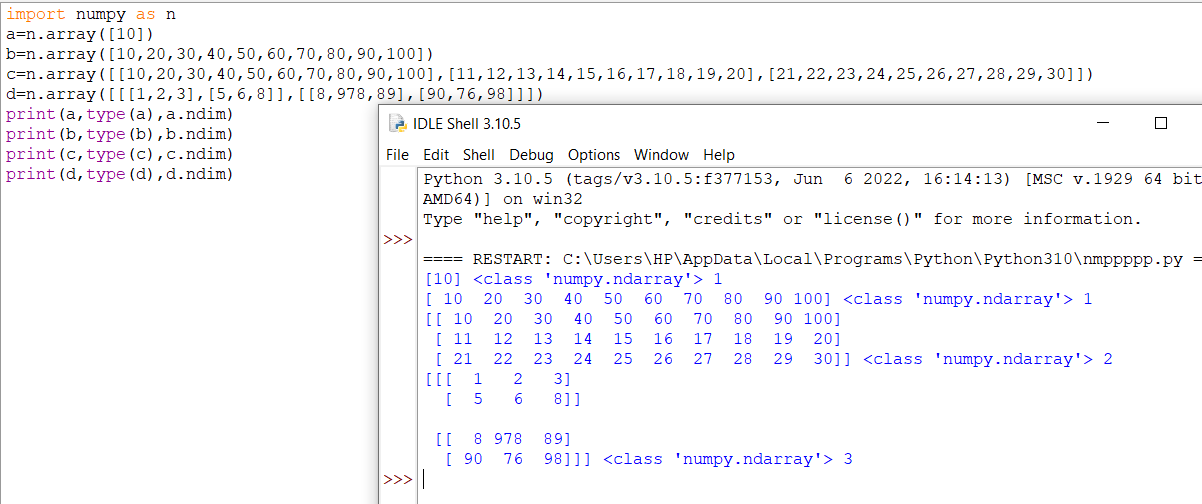
These are often used to represent matrix or 2nd order tensors.

NumPy has a whole sub module dedicated towards matrix operations called numpy.mat

## 3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

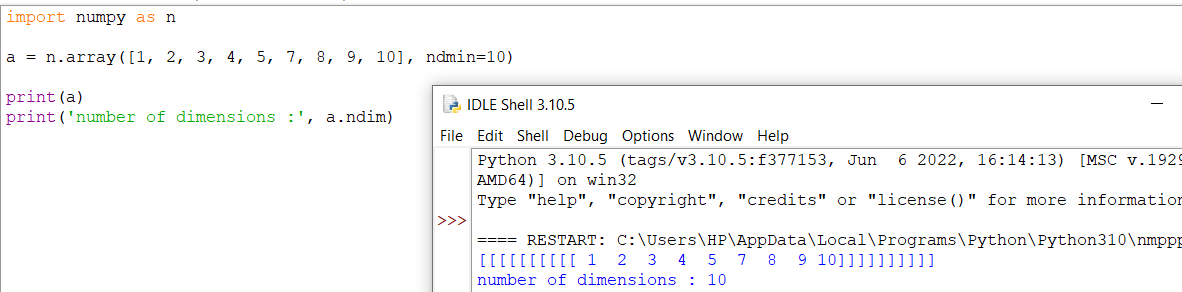
These are often used to represent a 3rd order tensor.



## 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.



# **NumPy Array Indexing**

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.

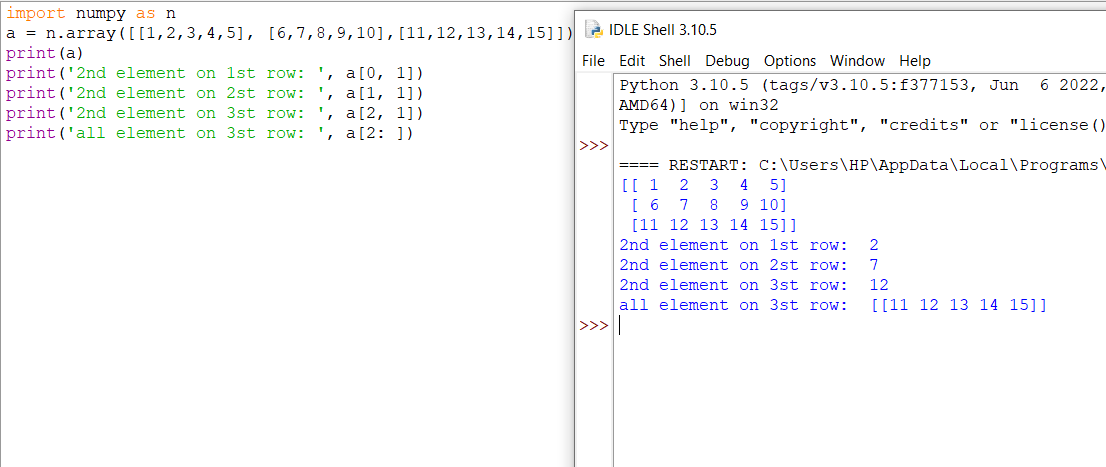
## Access 1-D Arrays

## 

## Access 2-D Arrays

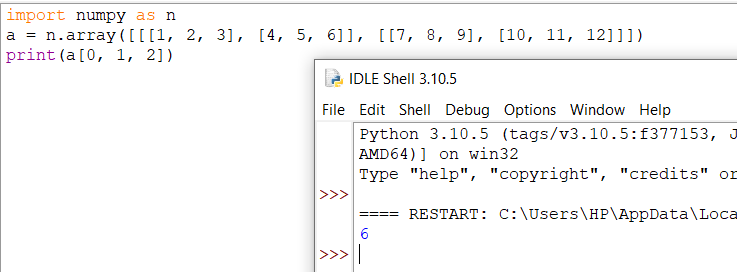
To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

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



## Access 3-D Arrays

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



arr[0, 1, 2] prints the value 6.

And this is why:

The first number represents the first dimension, which contains two arrays:  
[[1, 2, 3], [4, 5, 6]]  
and:  
[[7, 8, 9], [10, 11, 12]]  
Since we selected 0, we are left with the first array:  
[[1, 2, 3], [4, 5, 6]]

The second number represents the second dimension, which also contains two arrays:  
[1, 2, 3]  
and:  
[4, 5, 6]  
Since we selected 1, we are left with the second array:  
[4, 5, 6]

The third number represents the third dimension, which contains three values:  
4  
5  
6  
Since we selected 2, we end up with the third value:  
6

## Negative Indexing

Use negative indexing to access an array from the end.



# **NumPy Array Slicing**

# **Slicing arrays**

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

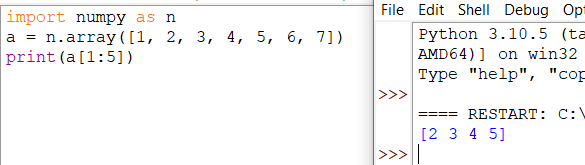
We pass slice instead of index like this: [start:end].

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

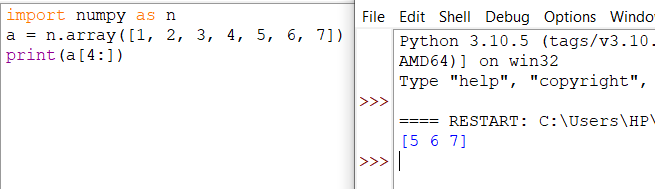
If we don't pass start its considered 0

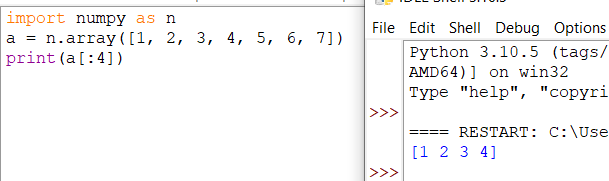
If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1



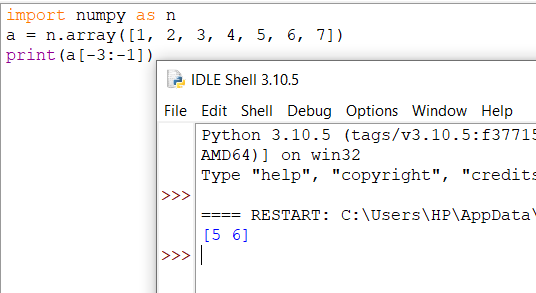
**Note:** The result includes the start index, but excludes the end index.





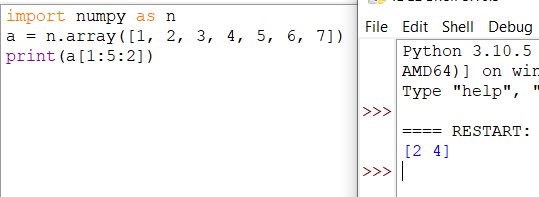
## Negative Slicing

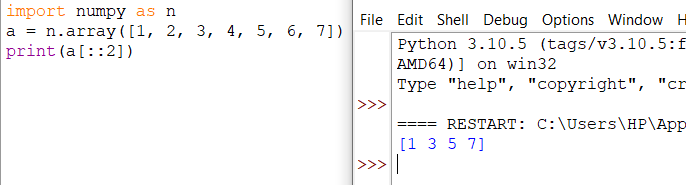
Use the minus operator to refer to an index from the end:



## STEP

Use the step value to determine the step of the slicing:





## Slicing 2-D Arrays

## 

# **NumPy Data Types**

## Data Types in Python

By default Python have these data types:

* strings - used to represent text data, the text is given under quote marks. e.g. "ABCD"
* integer - used to represent integer numbers. e.g. -1, -2, -3
* float - used to represent real numbers. e.g. 1.2, 42.42
* boolean - used to represent True or False.
* complex - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

## Data Types in NumPy

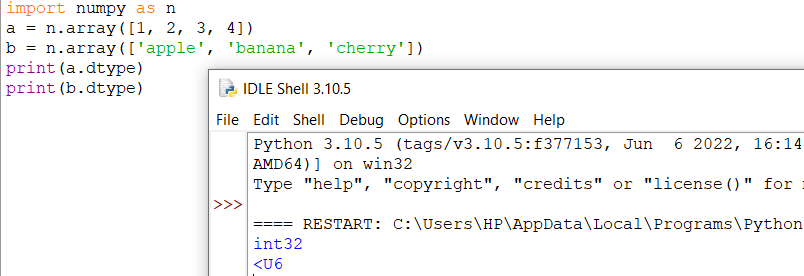
NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

* i - integer
* b - boolean
* u - unsigned integer
* f - float
* c - complex float
* m - timedelta
* M - datetime
* O - object
* S - string
* U - unicode string
* V - fixed chunk of memory for other type ( void )

## Checking the Data Type of an Array

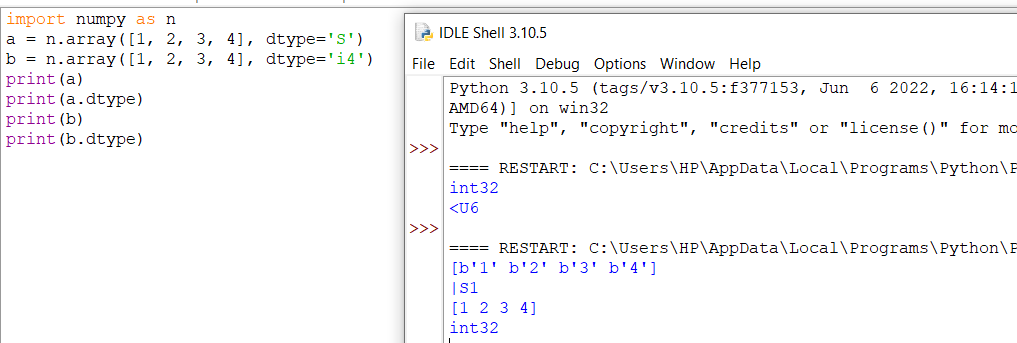
The NumPy array object has a property called dtype that returns the data type of the array:



## Creating Arrays With a Defined Data Type

We use the array() function to create arrays, this function can take an optional

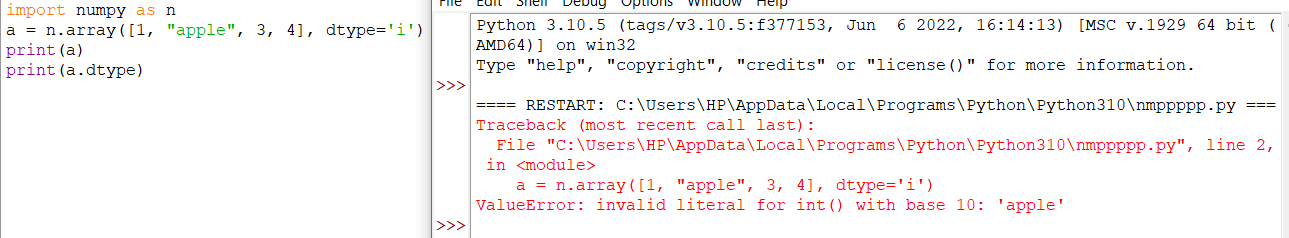
For i, u, f, S and U we can define size as well.



## What if a Value Can Not Be Converted?

If a type is given in which elements can't be casted then NumPy will raise a ValueError.

**ValueError:** In Python ValueError is raised when the type of passed argument to a function is unexpected/incorrect.

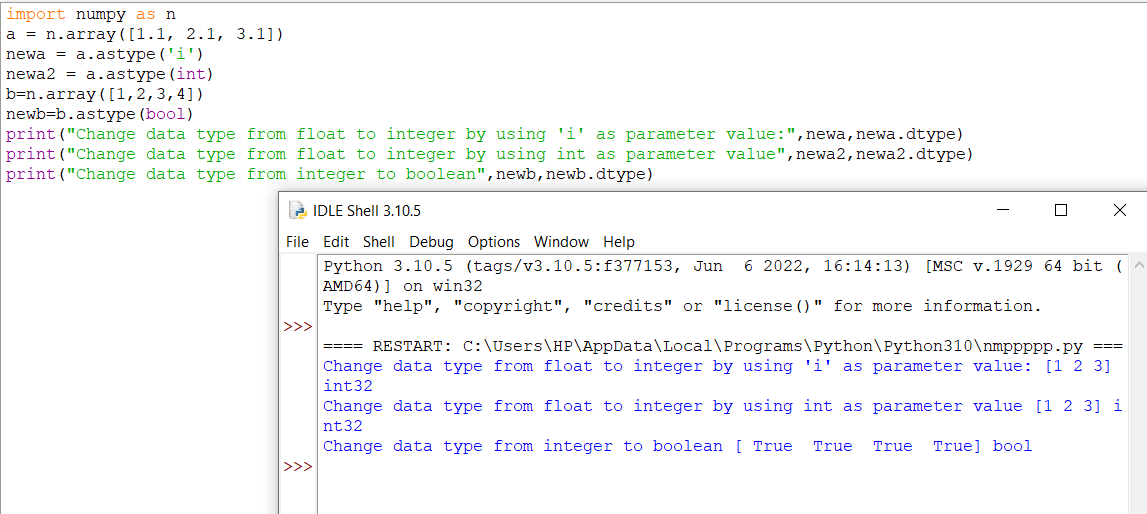


## Converting Data Type on Existing Arrays

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.



# **NumPy Array Copy vs View**

## The Difference Between Copy and View

The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

The copy owns the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.

The view does not own the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

## COPY:

## 

## VIEW:

## 

## The original array SHOULD be affected by the changes made to the view.

## 

## Check if Array Owns its Data

As mentioned above, copies owns the data, and views does not own the data, but how can we check this?

Every NumPy array has the attribute base that returns None if the array owns the data.

Otherwise, the base  attribute refers to the original object.

## 

# **NumPy Array Shape**

## Shape of an Array

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

## 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.

## 

## The example above returns (2, 4), which means that the array has 2 dimensions, where the first dimension has 2 elements and the second has 4

## 

# **NumPy Array Reshaping**

## 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 From 1-D to 2-D

## 

## Reshape From 1-D to 3-D

## 

## Can We Reshape Into any Shape?

Yes, as long as the elements required for reshaping are equal in both shapes.

We can reshape an 8 elements 1D array into 4 elements in 2 rows 2D array but we cannot reshape it into a 3 elements 3 rows 2D array as that would require 3x3 = 9 elements.



## Returns Copy or View?

## 

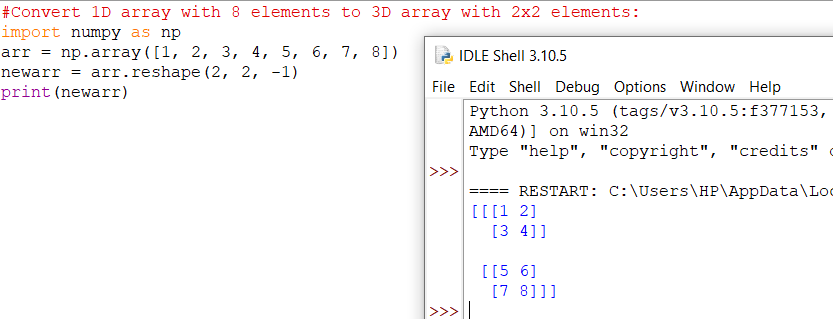
The example above returns the original array, so it is a view.

## Unknown Dimension

You are allowed to have one "unknown" dimension.

Meaning that you do not have to specify an exact number for one of the dimensions in the reshape method.

Pass -1 as the value, and NumPy will calculate this number for you.

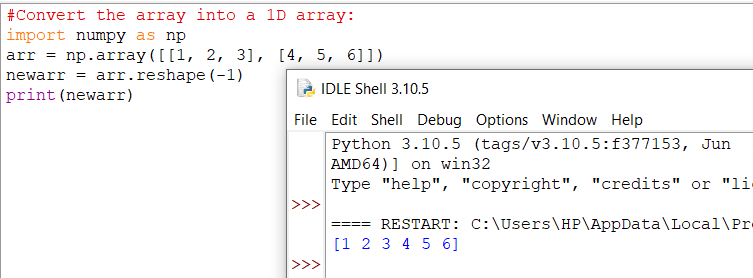


**Note:** We can not pass -1 to more than one dimension.

## Flattening the arrays

Flattening array means converting a multidimensional array into a 1D array.

We can use reshape(-1) to do this.



**Note:** There are a lot of functions for changing the shapes of arrays in numpy flatten, ravel and also for rearranging the elements rot90, flip, fliplr, flipud etc. These fall under Intermediate to Advanced section of numpy.

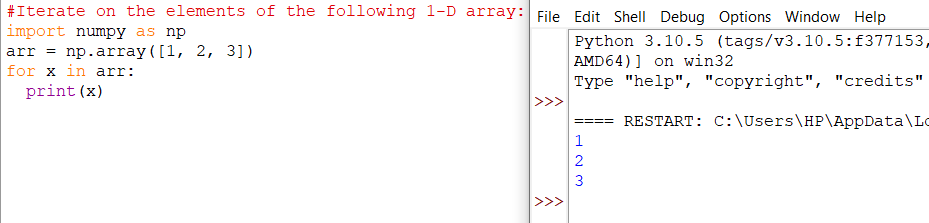
# **NumPy Array Iterating**

## Iterating Arrays

Iterating means going through elements one by one.

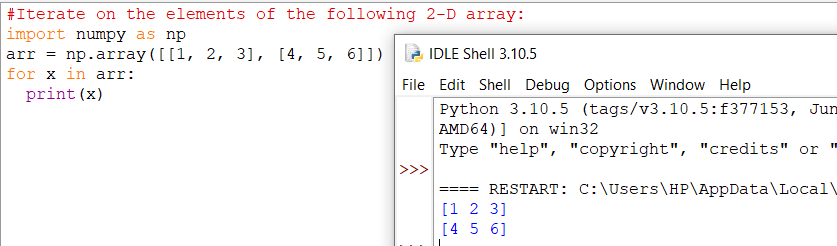
As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python.

If we iterate on a 1-D array it will go through each element one by one.



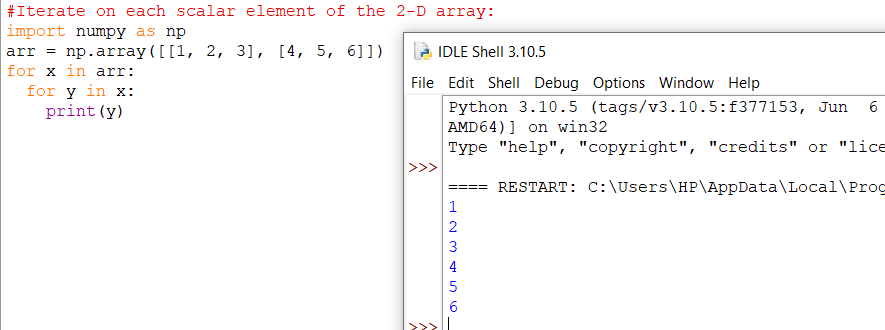
## Iterating 2-D Arrays

In a 2-D array it will go through all the rows.



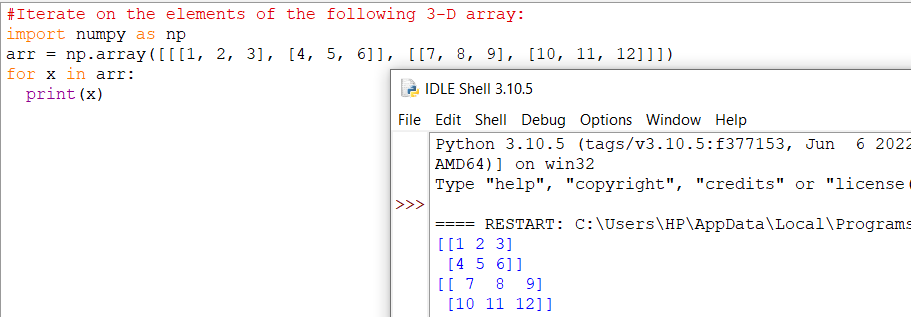
If we iterate on a n-D array it will go through n-1th dimension one by one.

To return the actual values, the scalars, we have to iterate the arrays in each dimension.

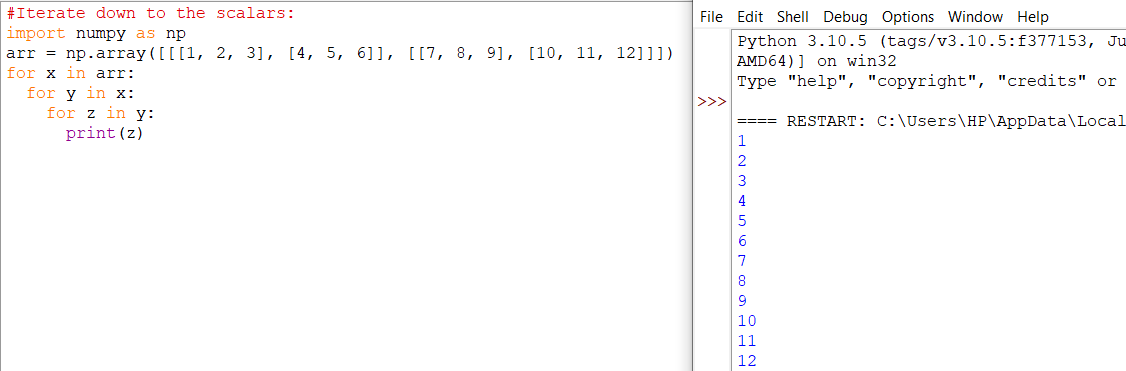


## Iterating 3-D Arrays

In a 3-D array it will go through all the 2-D arrays.



To return the actual values, the scalars, we have to iterate the arrays in each dimension.

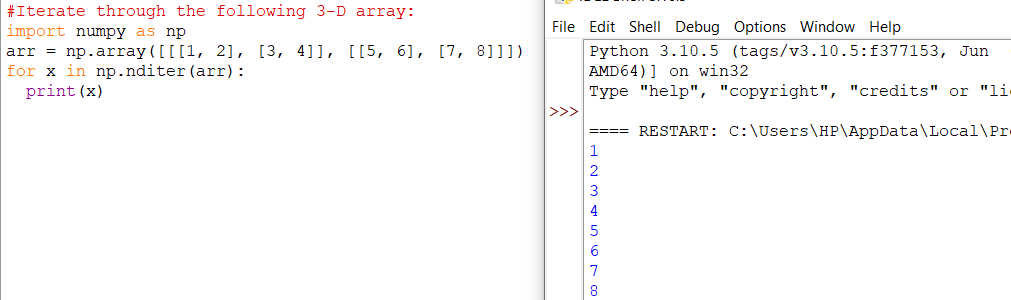


## Iterating Arrays Using nditer()

The function nditer() is a helping function that can be used from very basic to very advanced iterations. It solves some basic issues which we face in iteration, lets go through it with examples.

### **Iterating on Each Scalar Element**

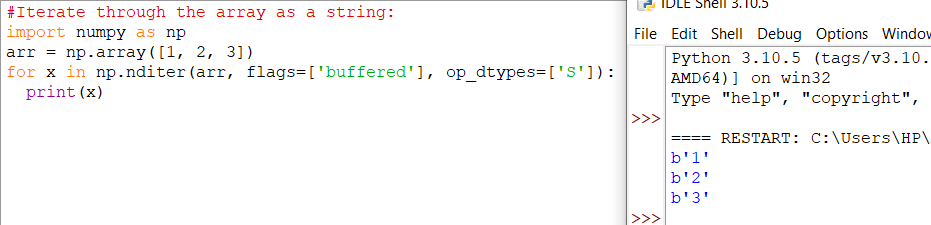
In basic for loops, iterating through each scalar of an array we need to use n for loops which can be difficult to write for arrays with very high dimensionality.



## Iterating Array With Different Data Types

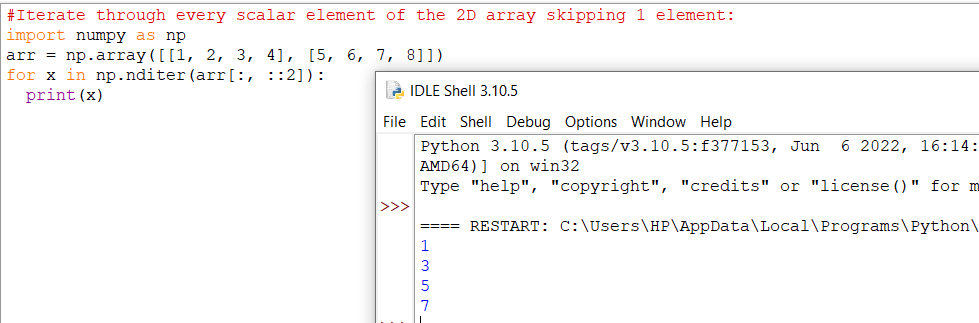
We can use op\_dtypes argument and pass it the expected datatype to change the datatype of elements while iterating.

NumPy does not change the data type of the element in-place (where the element is in array) so it needs some other space to perform this action, that extra space is called buffer, and in order to enable it in nditer() we pass flags=['buffered'].



## Iterating With Different Step Size

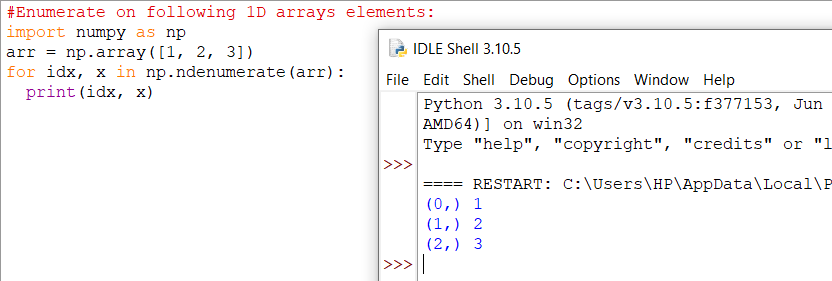
We can use filtering and followed by iteration.



## 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.



## 

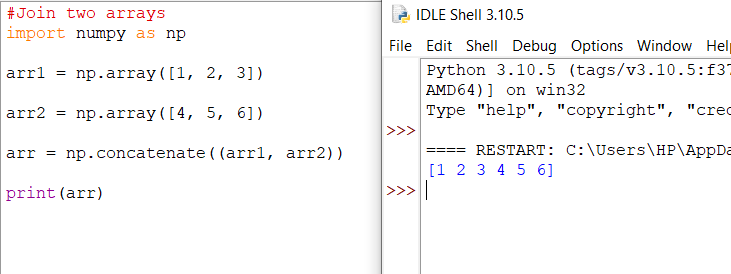
# **NumPy Joining Array**

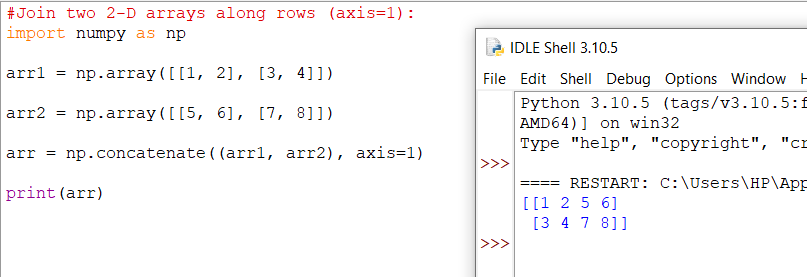
## Joining NumPy Arrays

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.



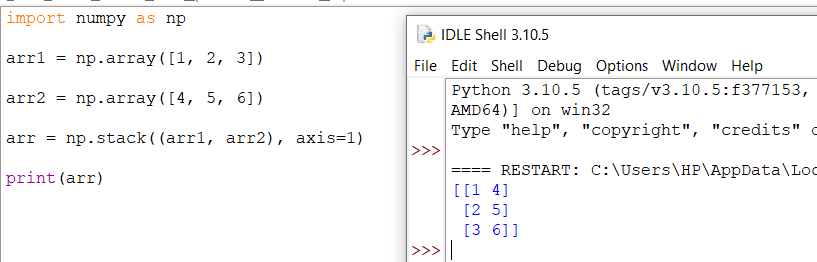


## Joining Arrays Using Stack Functions

Stacking is same as concatenation, the only difference is that stacking is done along a new axis.

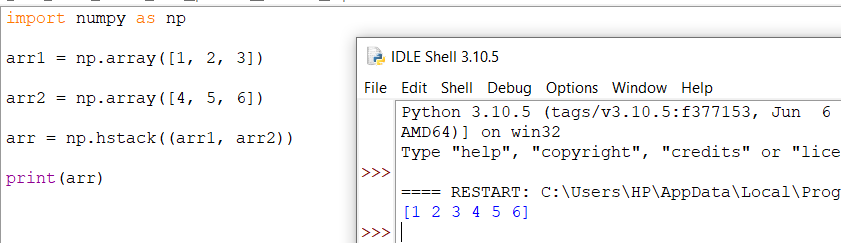
We can concatenate two 1-D arrays along the second axis which would result in putting them one over the other, ie. stacking.

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



## Stacking Along Rows

NumPy provides a helper function: hstack() to stack along rows.



## Stacking Along Columns

NumPy provides a helper function: vstack()  to stack along columns.



## Stacking Along Height (depth)

NumPy provides a helper function: dstack() to stack along height, which is the same as depth.

## 

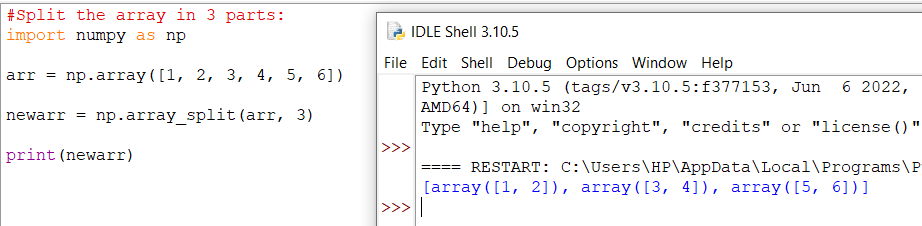
# **NumPy Splitting Array**

## Splitting NumPy Arrays

Splitting is reverse operation of Joining.

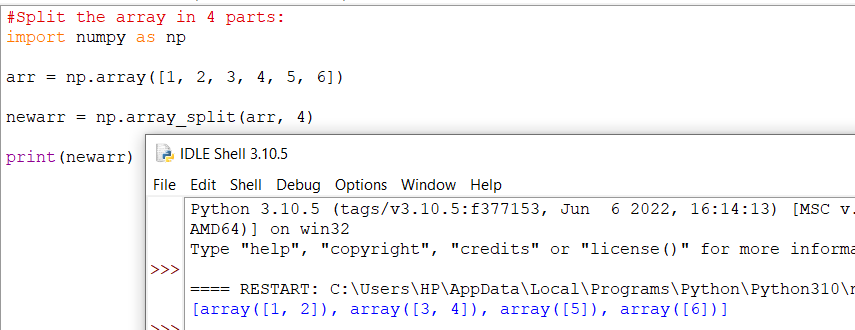
Joining merges multiple arrays into one and Splitting breaks one array into multiple.

We use array\_split() for splitting arrays, we pass it the array we want to split and the number of splits.



**Note:** The return value is an array containing three arrays.

If the array has less elements than required, it will adjust from the end accordingly.

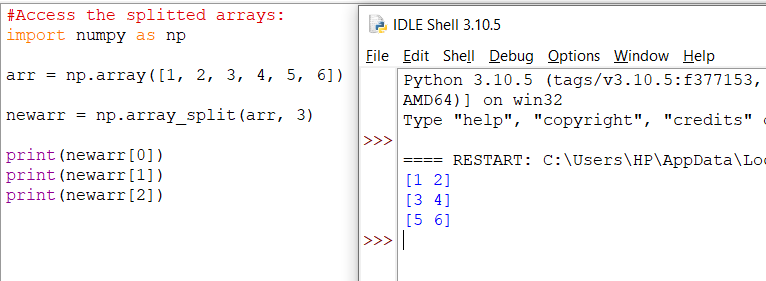


**Note:** We also have the method split() available but it will not adjust the elements when elements are less in source array for splitting like in example above, array\_split() worked properly but split() would fail.

**Split Into Arrays**

The return value of the array\_split() method is an array containing each of the split as an array.

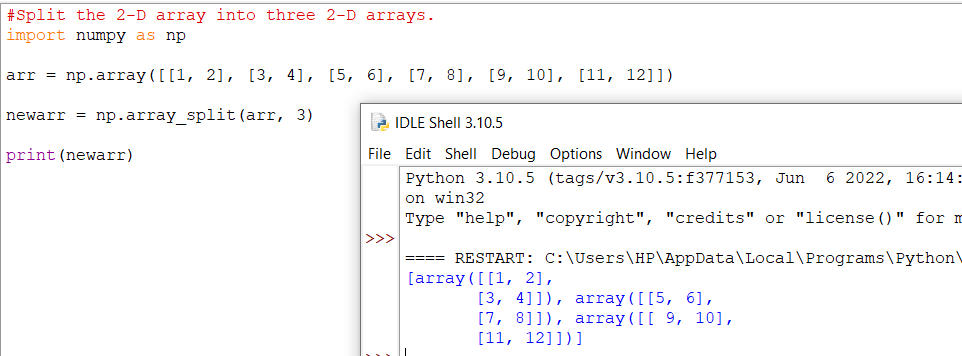
If you split an array into 3 arrays, you can access them from the result just like any array element:



## Splitting 2-D Arrays

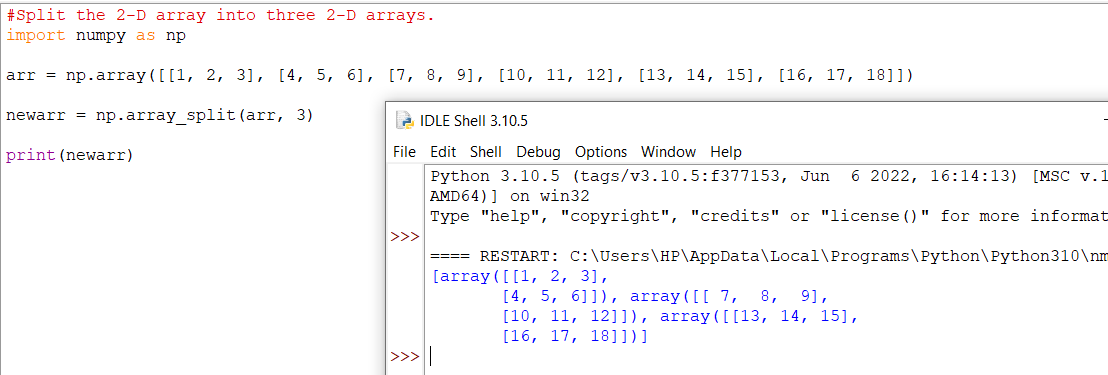
Use the same syntax when splitting 2-D arrays.

Use the array\_split() method, pass in the array you want to split and the number of splits you want to do.



The example above returns three 2-D arrays.

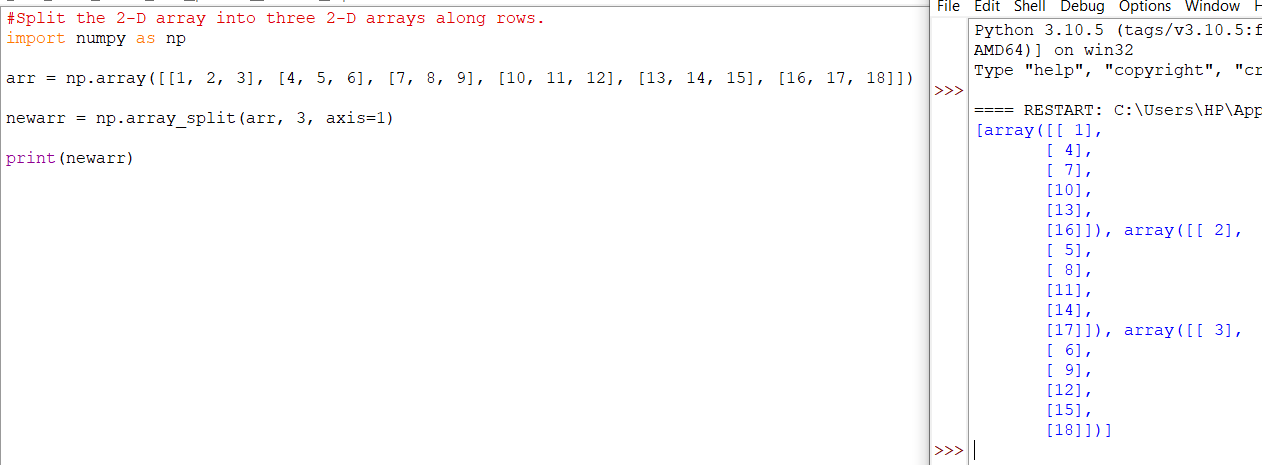
Let's look at another example, this time each element in the 2-D arrays contains 3 elements.



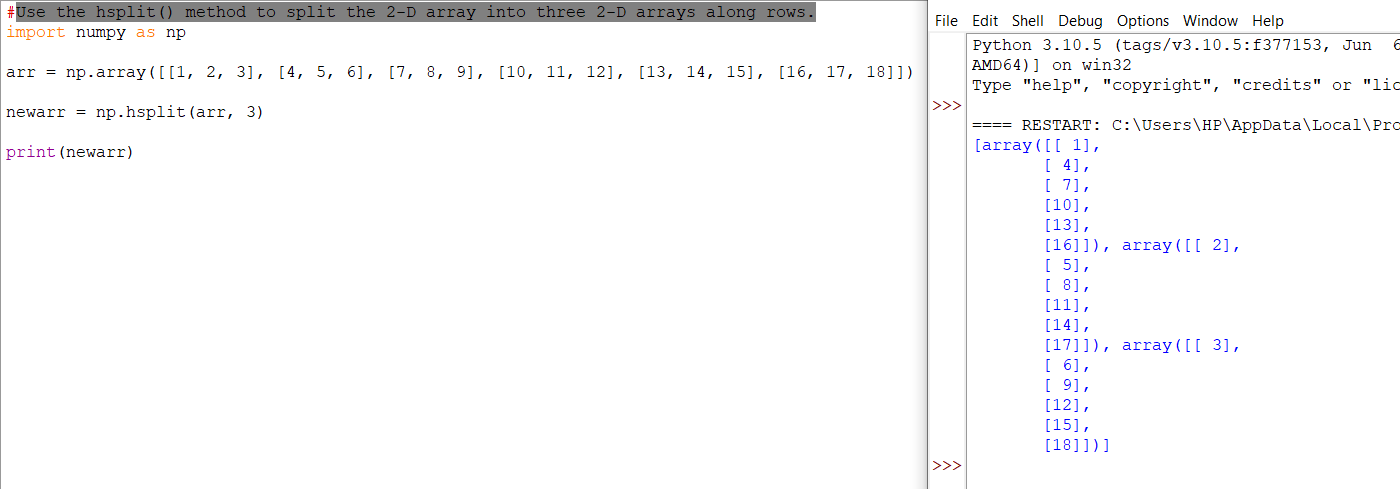
The example above returns three 2-D arrays.

In addition, you can specify which axis you want to do the split around.

The example below also returns three 2-D arrays, but they are split along the row (axis=1).



An alternate solution is using hsplit() opposite of hstack()



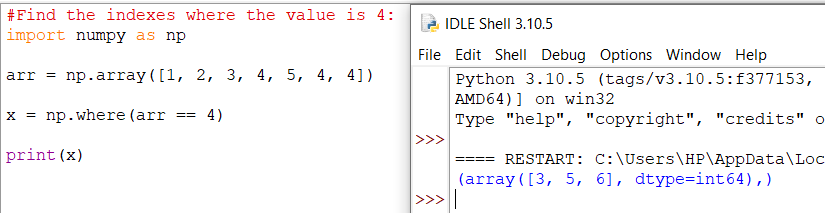
**Note:** Similar alternates to vstack() and dstack() are available as vsplit() and dsplit().

# **NumPy Searching Arrays**

## Searching Arrays

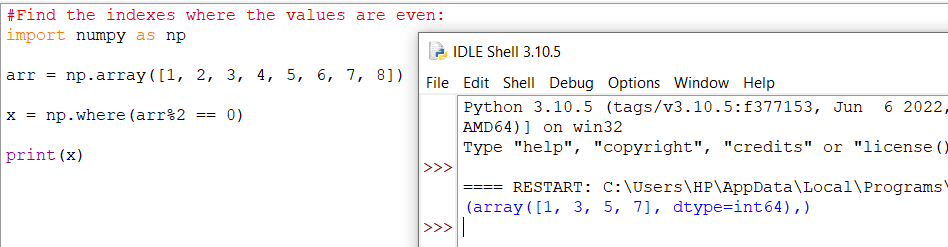
You can search an array for a certain value, and return the indexes that get a match.

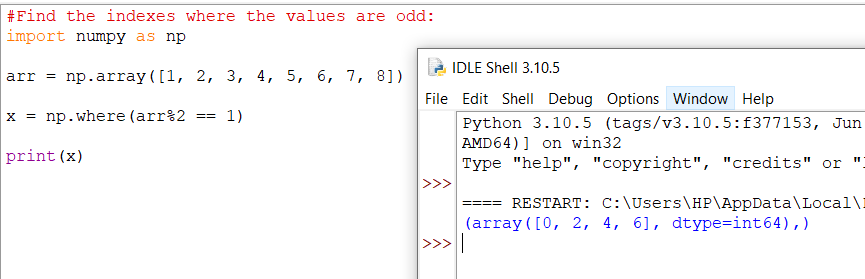
To search an array, use the where() method.



The example above will return a tuple: (array([3, 5, 6],)

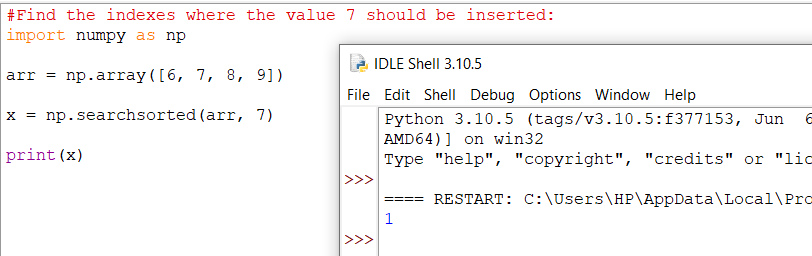
Which means that the value 4 is present at index 3, 5, and 6.





**Search Sorted**

There is a method called searchsorted() which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

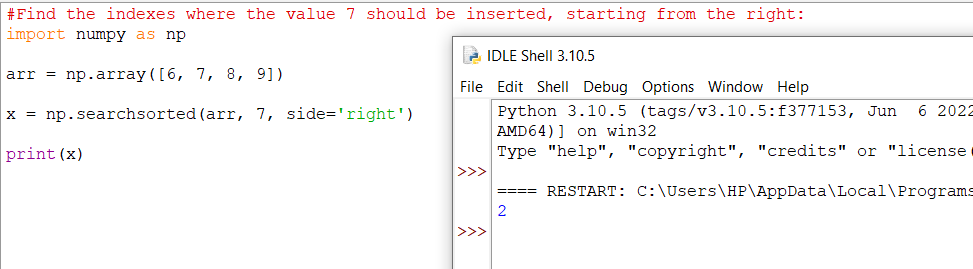
The searchsorted() method is assumed to be used on sorted arrays.

Example explained: The number 7 should be inserted on index 1 to remain the sort order.

The method starts the search from the left and returns the first index where the number 7 is no longer larger than the next value.

### **Search From the Right Side**

By default the left most index is returned, but we can give side='right' to return the right most index instead.



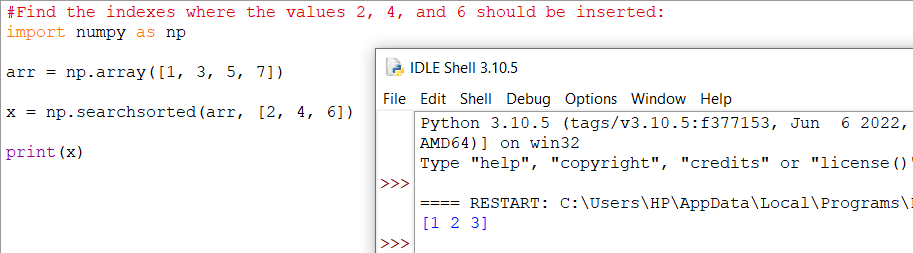
Example explained: The number 7 should be inserted on index 2 to remain the sort order.

The method starts the search from the right and returns the first index where the number 7 is no longer less than the next value.

### **Multiple Values**

To search for more than one value, use an array with the specified values.

The return value is an array: [1 2 3] containing the three indexes where 2, 4, 6 would be inserted in the original array to maintain the order.



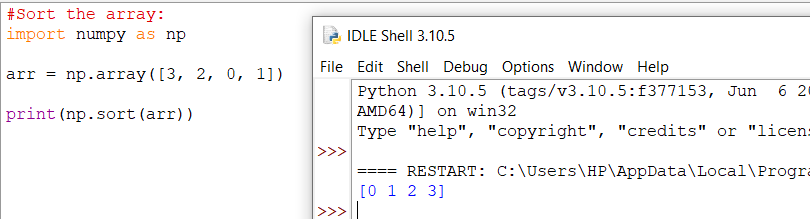
# **NumPy Sorting Arrays**

# **Sorting Arrays**

Sorting means putting elements in an ordered sequence.

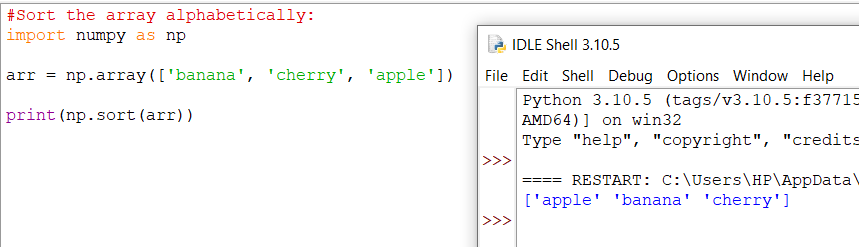
Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

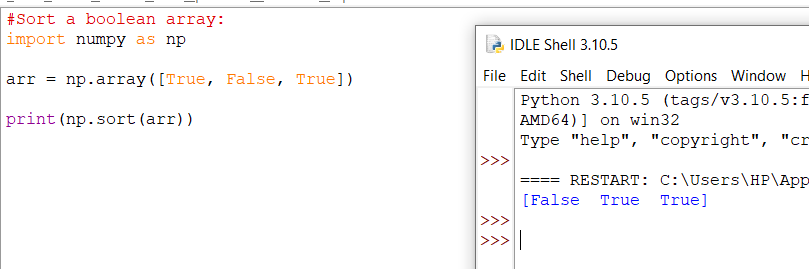
The NumPy ndarray object has a function called sort(), that will sort a specified array.



**Note:** This method returns a copy of the array, leaving the original array unchanged.

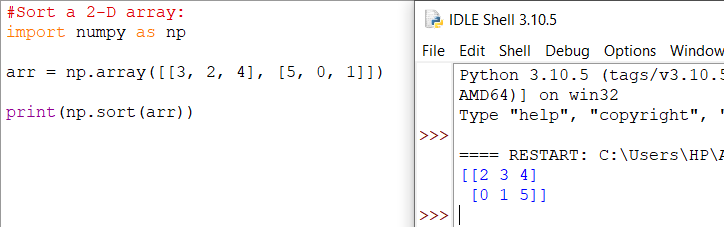
You can also sort arrays of strings, or any other data type:





## Sorting a 2-D Array

If you use the sort() method on a 2-D array, both arrays will be sorted:



# **NumPy Filter Array**

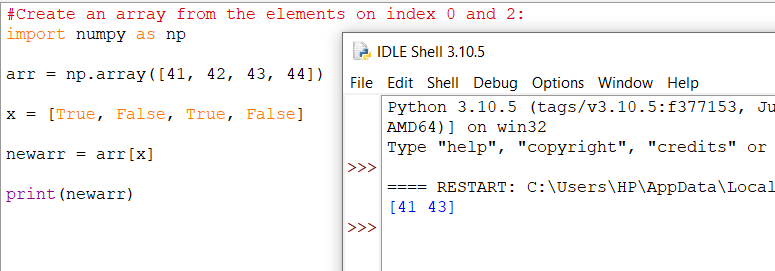
**Filtering Arrays**

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.

A boolean index list is a list of booleans corresponding to indexes in the array.

If the value at an index is True that element is contained in the filtered array, if the value at that index is False that element is excluded from the filtered array.

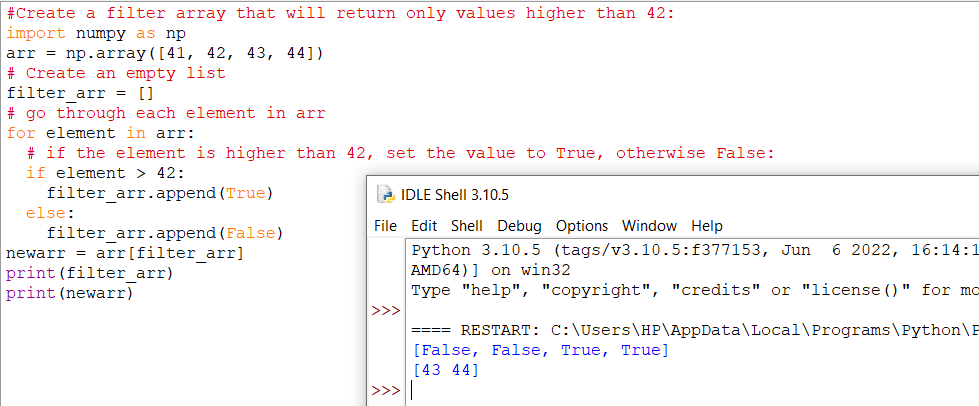


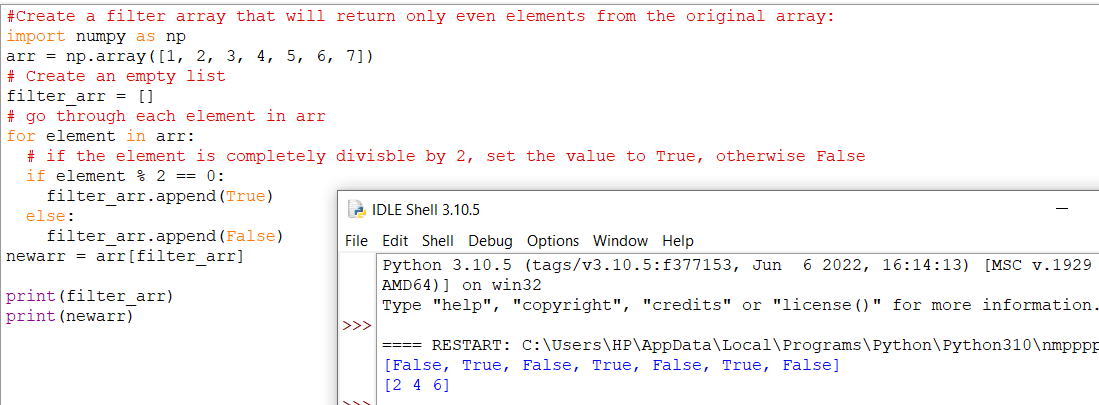
The example above will return [41, 43], why?

Because the new array contains only the values where the filter array had the value True, in this case, index 0 and 2.

## Creating the Filter Array

In the example above we hard-coded the True and False values, but the common use is to create a filter array based on conditions.

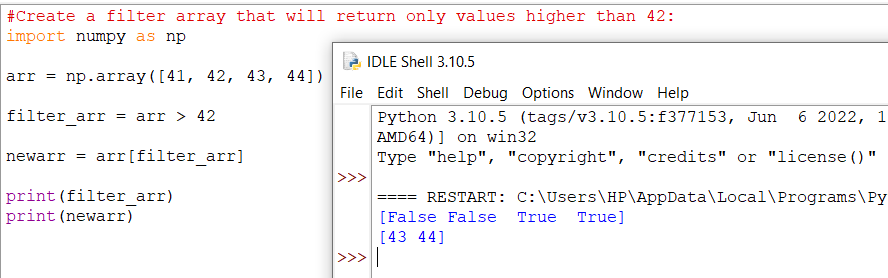


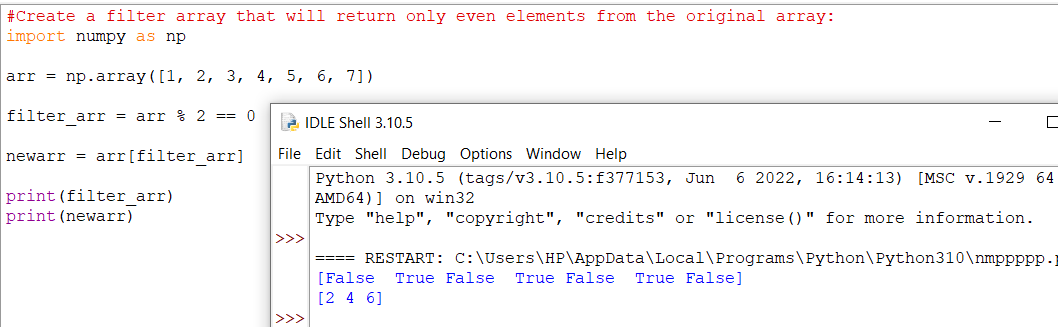


## Creating Filter Directly From Array

The above example is quite a common task in NumPy and NumPy provides a nice way to tackle it.

We can directly substitute the array instead of the iterable variable in our condition and it will work just as we expect it to.





# **Random Numbers in NumPy**

## What is a Random Number?

Random number does NOT mean a different number every time. Random means something that can not be predicted logically.

## Pseudo Random and True Random.

Computers work on programs, and programs are definitive set of instructions. So it means there must be some algorithm to generate a random number as well.

If there is a program to generate random number it can be predicted, thus it is not truly random.

Random numbers generated through a generation algorithm are called pseudo random.

Can we make truly random numbers?

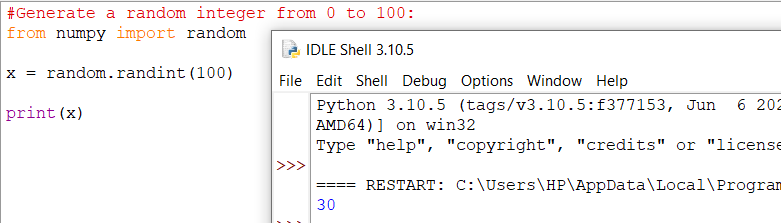
Yes. In order to generate a truly random number on our computers we need to get the random data from some outside source. This outside source is generally our keystrokes, mouse movements, data on network etc.

We do not need truly random numbers, unless its related to security (e.g. encryption keys) or the basis of application is the randomness (e.g. Digital roulette wheels).

In this tutorial we will be using pseudo random numbers.

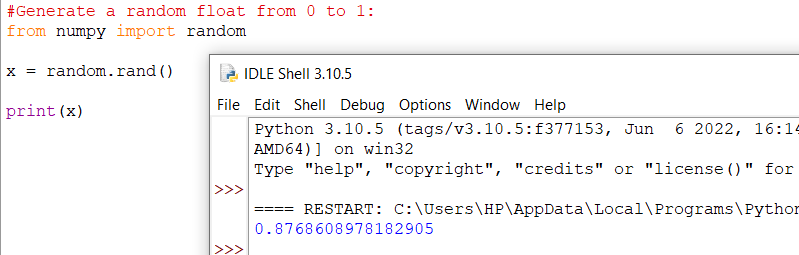
## Generate Random Number

NumPy offers the random module to work with random numbers.



## Generate Random Float

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

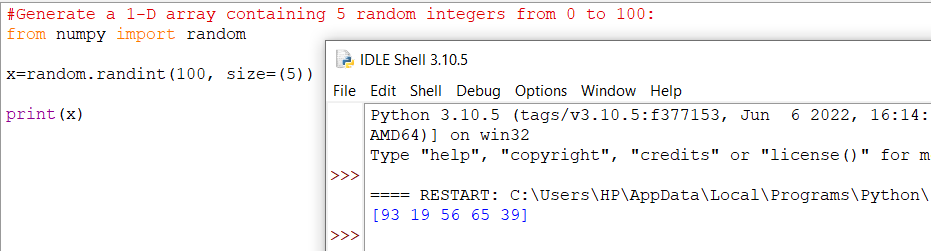


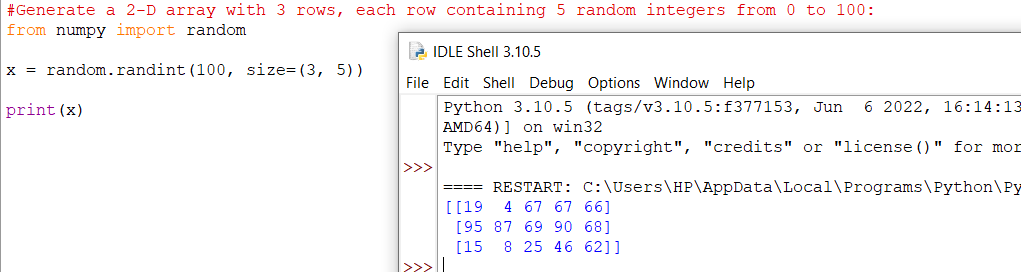
## Generate Random Array

In NumPy we work with arrays, and you can use the two methods from the above examples to make random arrays.

### **Integers**

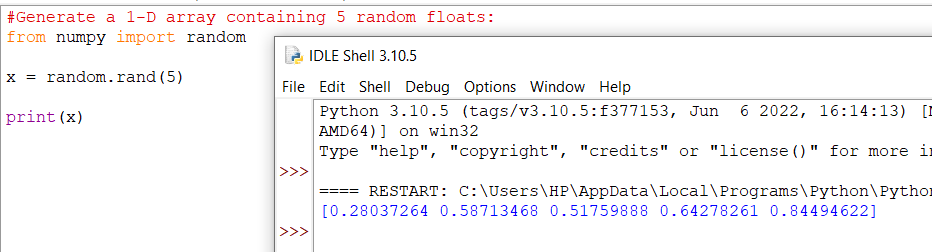
The randint() method takes a size parameter where you can specify the shape of an array.

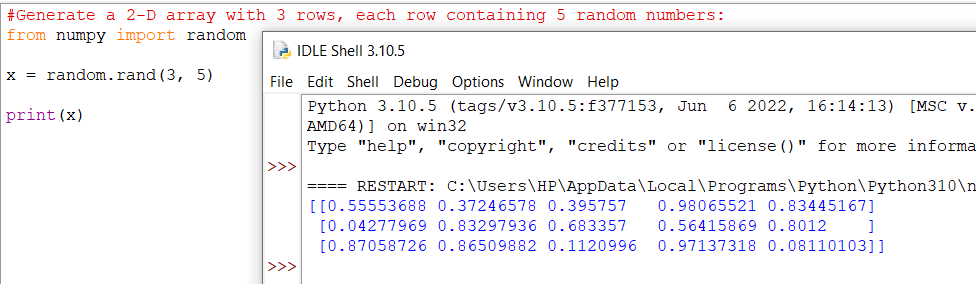




### **Floats**

The rand() method also allows you to specify the shape of the array.

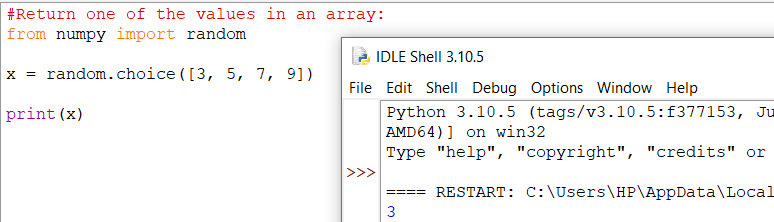




## Generate Random Number From Array

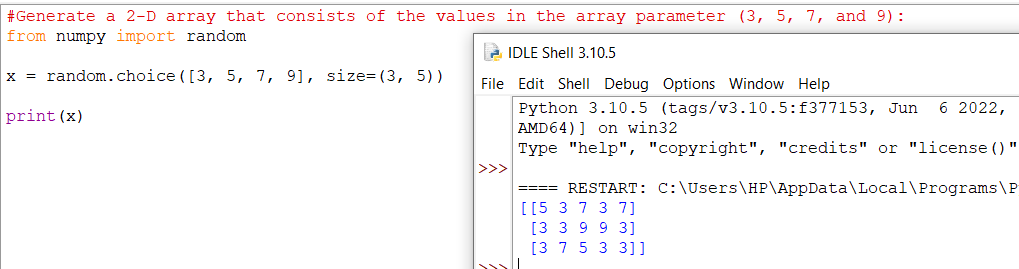
The choice() method allows you to generate a random value based on an array of values.

The choice() method takes an array as a parameter and randomly returns one of the values.



The choice() method also allows you to return an array of values.

Add a size parameter to specify the shape of the array.



# **Random Data Distribution**

## What is Data Distribution?

Data Distribution is a list of all possible values, and how often each value occurs.

Such lists are important when working with statistics and data science.

The random module offer methods that returns randomly generated data distributions.

## Random Distribution

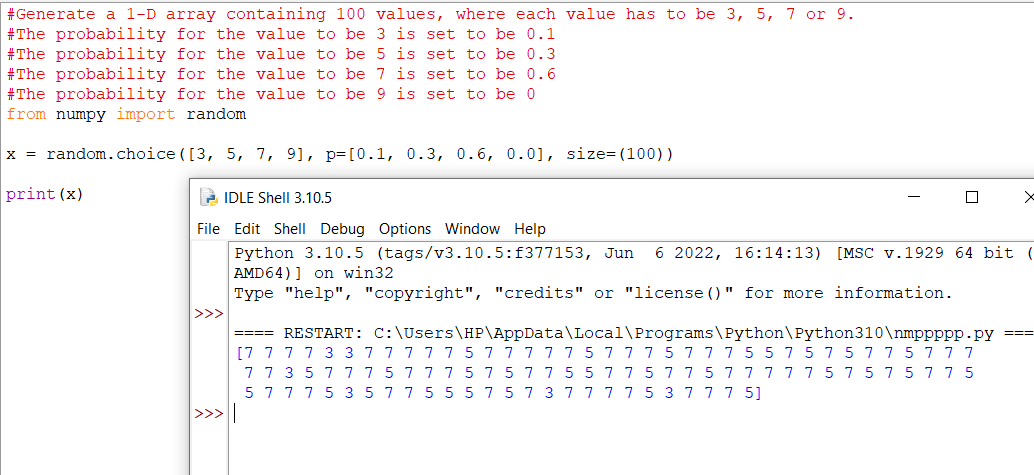
A random distribution is a set of random numbers that follow a certain probability density function.

**Probability Density Function:** A function that describes a continuous probability. i.e. probability of all values in an array.

We can generate random numbers based on defined probabilities using the choice() method of the random module.

The choice() method allows us to specify the probability for each value.

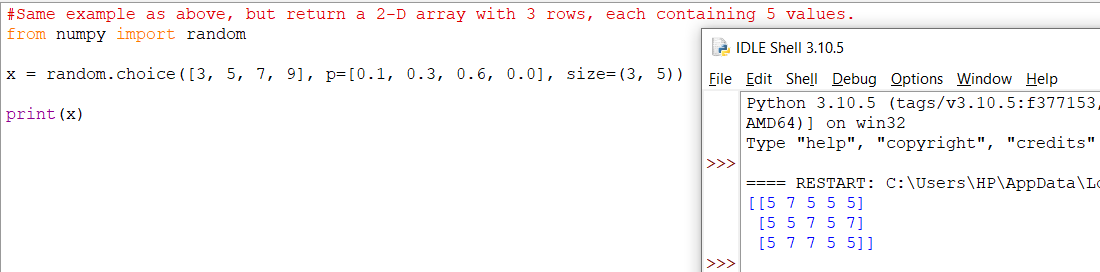
The probability is set by a number between 0 and 1, where 0 means that the value will never occur and 1 means that the value will always occur.



The sum of all probability numbers should be 1.

Even if you run the example above 100 times, the value 9 will never occur.

You can return arrays of any shape and size by specifying the shape in the size parameter.



# **Random Permutations**

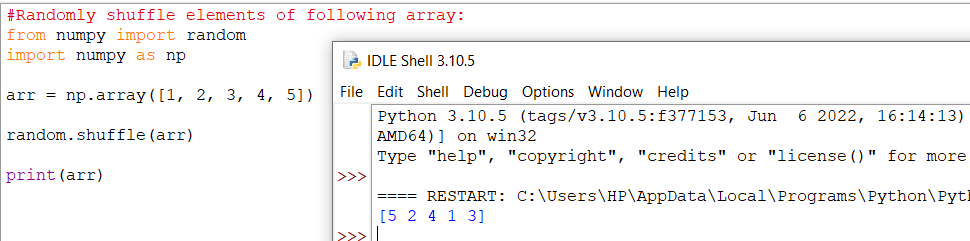
## Random Permutations of Elements

A permutation refers to an arrangement of elements. e.g. [3, 2, 1] is a permutation of [1, 2, 3] and vice-versa.

The NumPy Random module provides two methods for this: shuffle() and permutation().

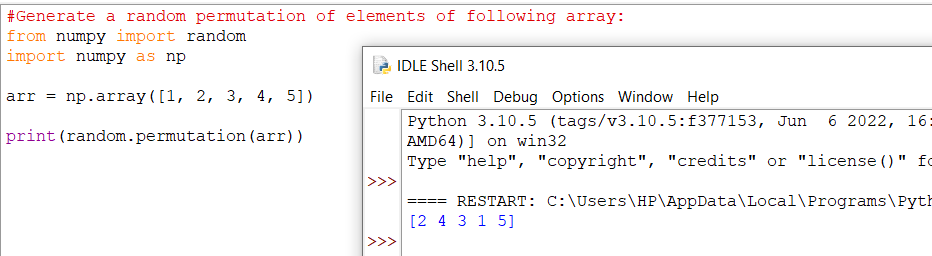
## Shuffling Arrays

Shuffle means changing arrangement of elements in-place. i.e. in the array itself.



The shuffle() method makes changes to the original array.

Generating Permutation of Arrays

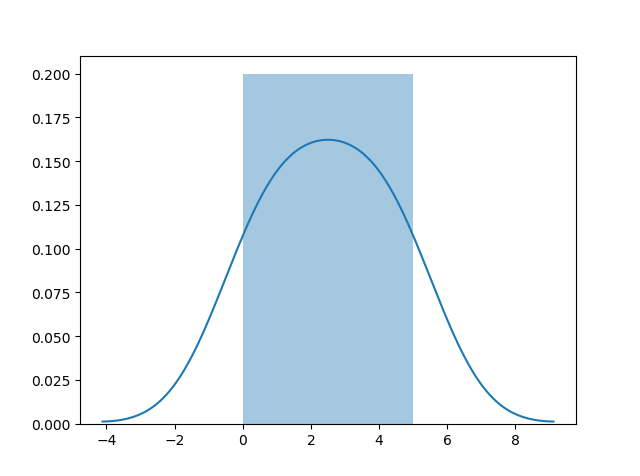


The permutation() method returns a re-arranged array (and leaves the original array un-changed).

# **Seaborn**

## Visualize Distributions With Seaborn

Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.



## Install Seaborn.

If you have [Python](https://www.w3schools.com/python/default.asp) and [PIP](https://www.w3schools.com/python/python_pip.asp) already installed on a system, install it using this command:

C:\Users\Your Name>pip install seaborn

If you use Jupyter, install Seaborn using this command:

C:\Users\Your Name>!pip install seaborn

## Distplots

Distplot stands for distribution plot, it takes as input an array and plots a curve corresponding to the distribution of points in the array.

## Import Matplotlib

Import the pyplot object of the Matplotlib module in your code using the following statement:

import matplotlib.pyplot as plt

You can learn about the Matplotlib module in our [Matplotlib Tutorial](https://www.w3schools.com/python/matplotlib_intro.asp).

## Import Seaborn

Import the Seaborn module in your code using the following statement:

import seaborn as sns

## Plotting a Distplot

## Plotting a Distplot Without the Histogram

## 

**Note:** We will be using: sns.distplot(arr, hist=False) to visualize random distributions in this tutorial.

# **Normal (Gaussian) Distribution**

Normal Distribution

The Normal Distribution is one of the most important distributions.

It is also called the Gaussian Distribution after the German mathematician Carl Friedrich Gauss.

It fits the probability distribution of many events, eg. IQ Scores, Heartbeat etc.

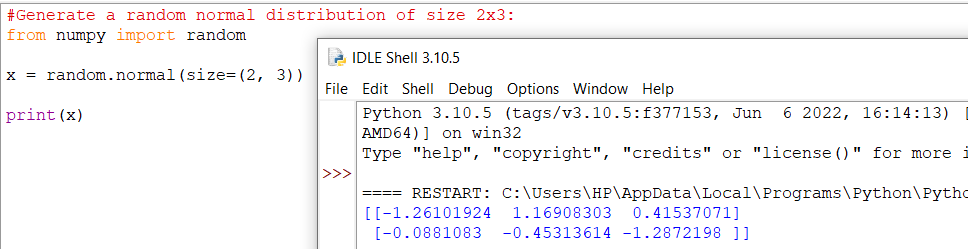
Use the random.normal() method to get a Normal Data Distribution.

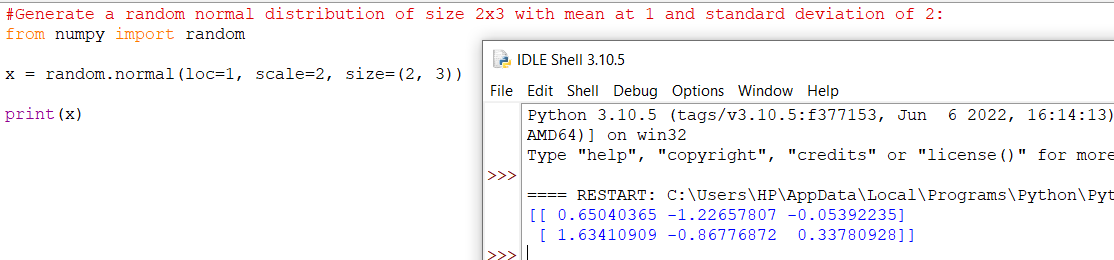
It has three parameters:

loc - (Mean) where the peak of the bell exists.

scale - (Standard Deviation) how flat the graph distribution should be.

size - The shape of the returned array.





## Visualization of Normal Distribution

## 

**Note:** The curve of a Normal Distribution is also known as the Bell Curve because of the bell-shaped curve.

# **Binomial Distribution**

Binomial Distribution

Binomial Distribution is a Discrete Distribution.

It describes the outcome of binary scenarios, e.g. toss of a coin, it will either be head or tails.

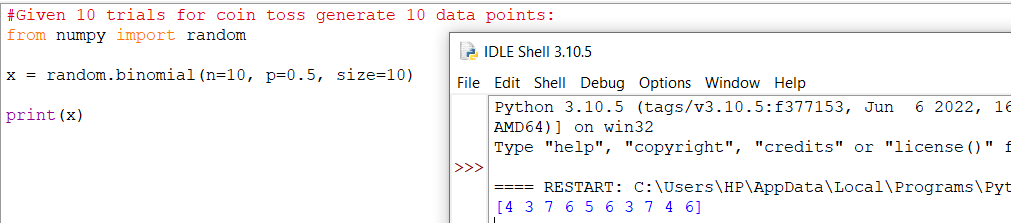
It has three parameters:

n - number of trials.

p - probability of occurence of each trial (e.g. for toss of a coin 0.5 each).

size - The shape of the returned array.

**Discrete Distribution:**The distribution is defined at separate set of events, e.g. a coin toss's result is discrete as it can be only head or tails whereas height of people is continuous as it can be 170, 170.1, 170.11 and so on.

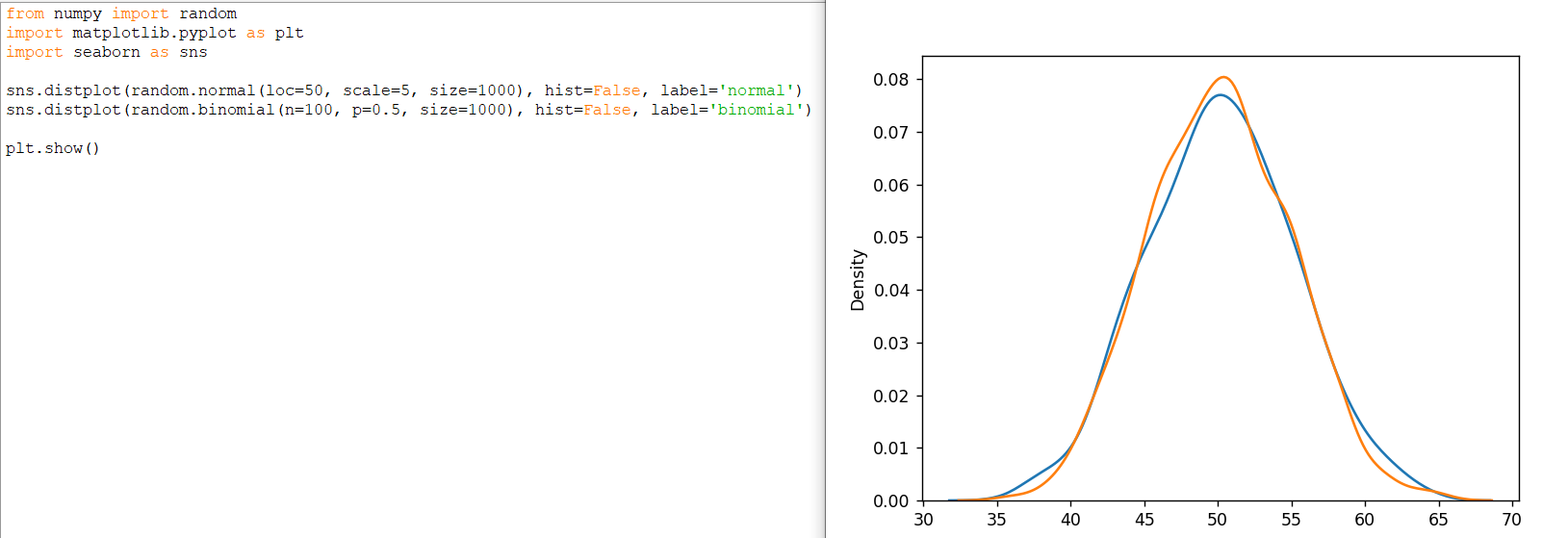


## Visualization of Binomial Distribution

## 

## Difference Between Normal and Binomial Distribution

The main difference is that normal distribution is continous whereas binomial is discrete, but if there are enough data points it will be quite similar to normal distribution with certain loc and scale.



# **Poisson Distribution**

# Poisson Distribution

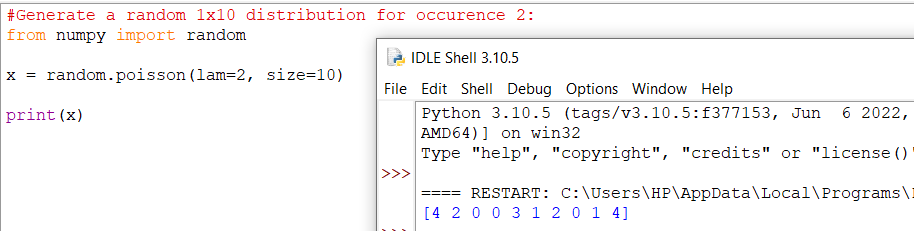
Poisson Distribution is a Discrete Distribution.

It estimates how many times an event can happen in a specified time. e.g. If someone eats twice a day what is probability he will eat thrice?

It has two parameters:

lam - rate or known number of occurences e.g. 2 for above problem.

size - The shape of the returned array.



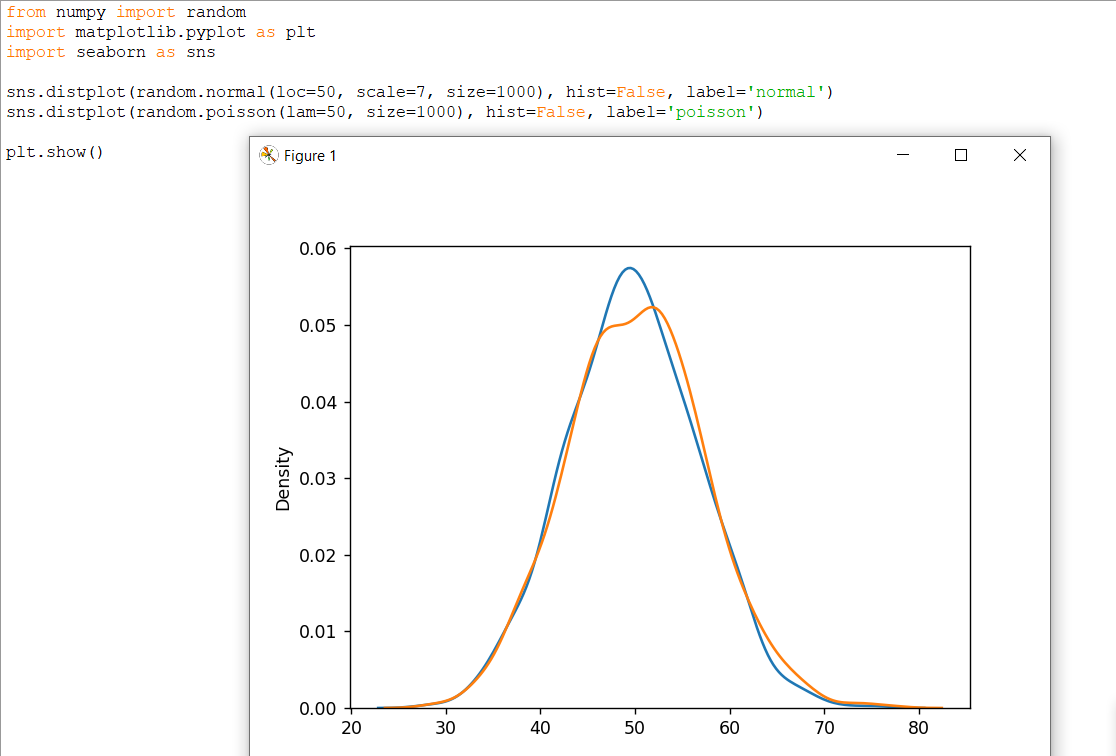
## Visualization of Poisson Distribution

## 

## Difference Between Normal and Poisson Distribution

Normal distribution is continous whereas poisson is discrete.

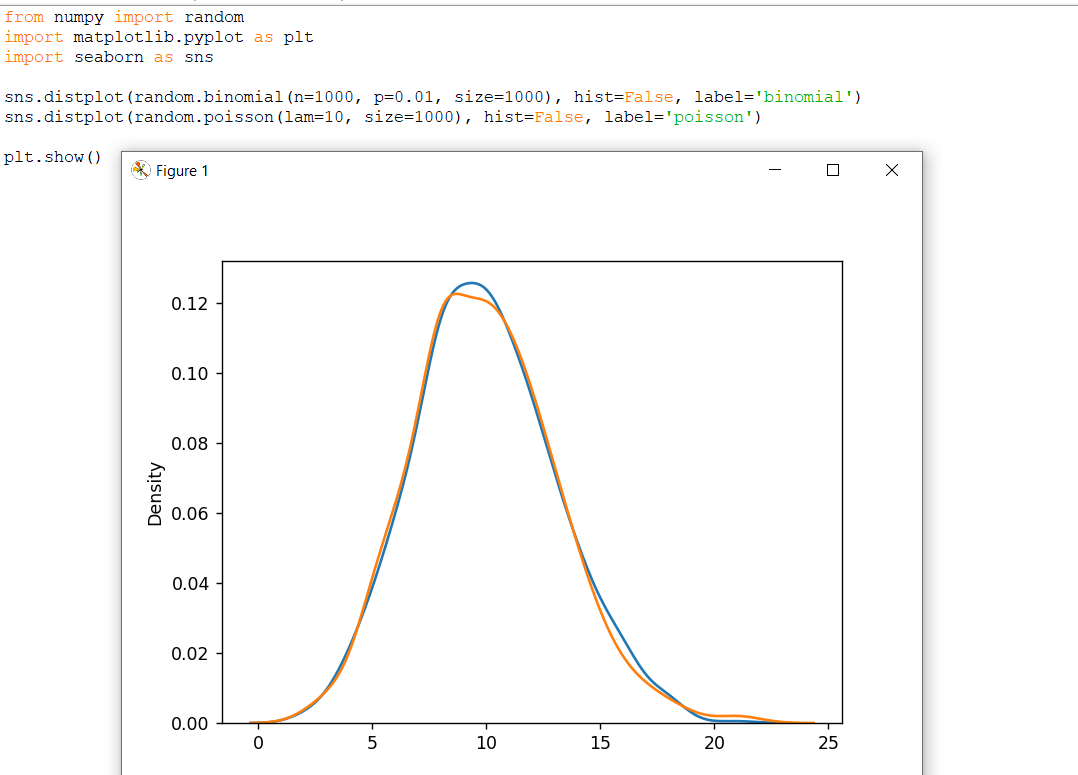
But we can see that similar to binomial for a large enough poisson distribution it will become similar to normal distribution with certain std dev and mean.



## Difference Between Poisson and Binomial Distribution

The difference is very subtle it is that, binomial distribution is for discrete trials, whereas poisson distribution is for continuous trials.

But for very large n and near-zero p binomial distribution is near identical to poisson distribution such that n \* p is nearly equal to lam.



# **Uniform Distribution**

Uniform Distribution

Used to describe probability where every event has equal chances of occuring.

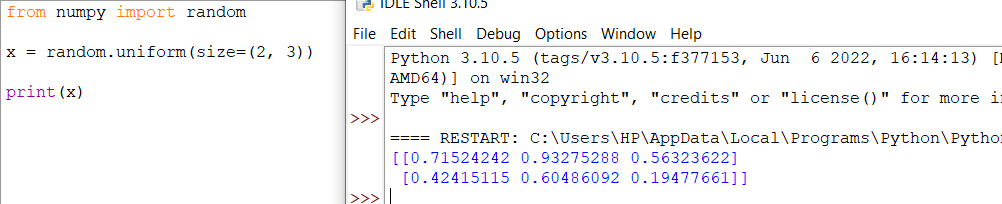
E.g. Generation of random numbers.

It has three parameters:

a - lower bound - default 0 .0.

b - upper bound - default 1.0.

size - The shape of the returned array.



## Visualization of Uniform Distribution

## 

# **Logistic Distribution**

Logistic Distribution

Logistic Distribution is used to describe growth.

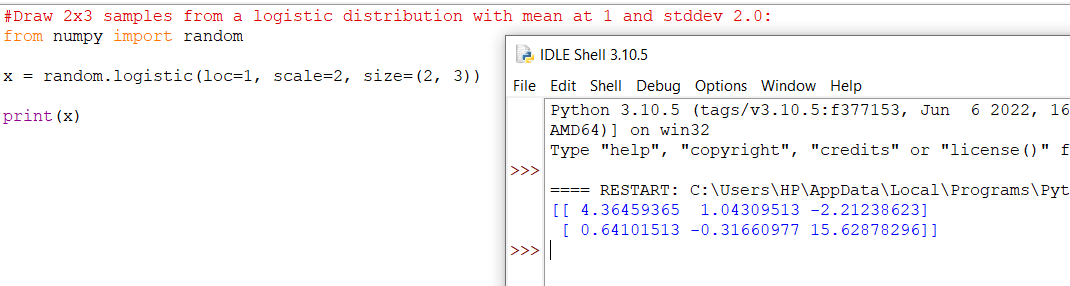
Used extensively in machine learning in logistic regression, neural networks etc.

It has three parameters:

loc - mean, where the peak is. Default 0.

scale - standard deviation, the flatness of distribution. Default 1.

size - The shape of the returned array.



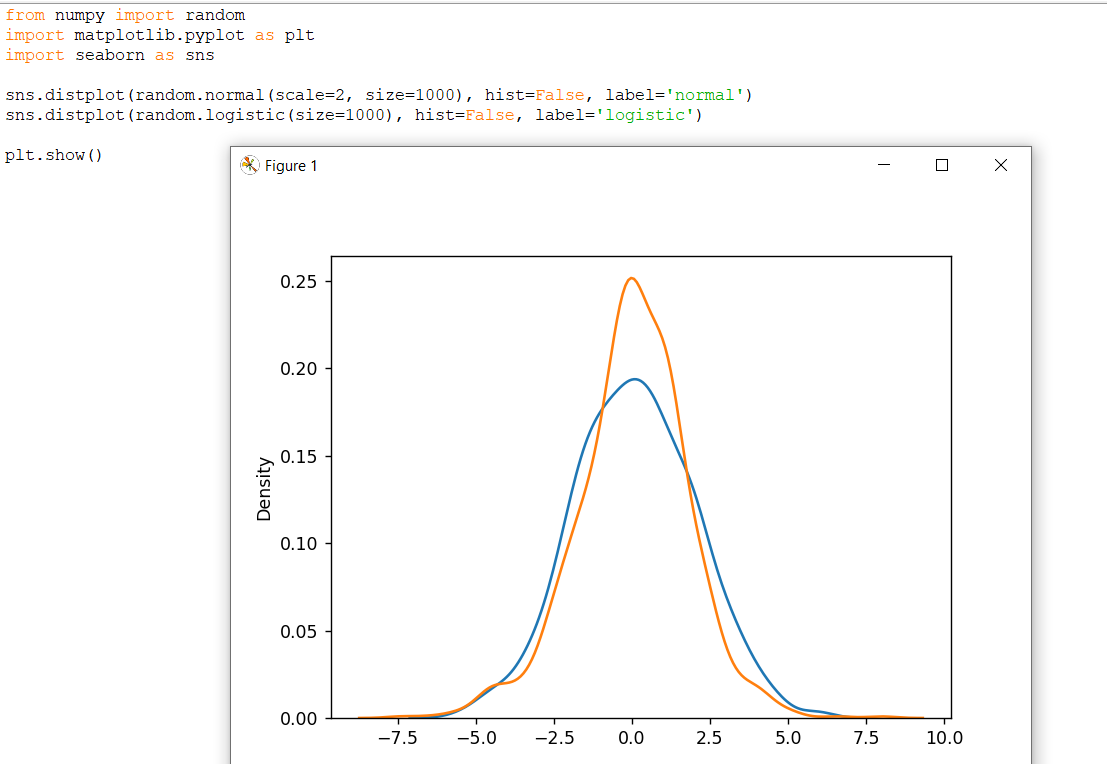
## Visualization of Logistic Distribution

## 

Difference Between Logistic and Normal Distribution

Both distributions are near identical, but logistic distribution has more area under the tails, meaning it represents more possibility of occurrence of an event further away from mean.

For higher value of scale (standard deviation) the normal and logistic distributions are near identical apart from the peak.



# **Multinomial Distribution**

Multinomial Distribution

Multinomial distribution is a generalization of binomial distribution.

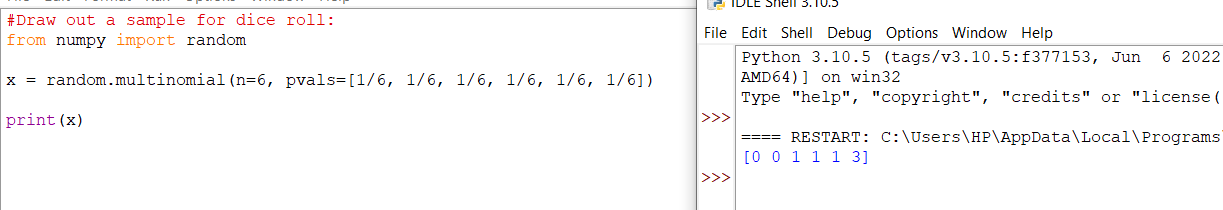
It describes outcomes of multi-nomial scenarios unlike binomial where scenarios must be only one of two. e.g. Blood type of a population, dice roll outcome.

It has three parameters:

n - number of possible outcomes (e.g. 6 for dice roll).

pvals - list of probabilties of outcomes (e.g. [1/6, 1/6, 1/6, 1/6, 1/6, 1/6] for dice roll).

size - The shape of the returned array.



**Note:** Multinomial samples will NOT produce a single value! They will produce one value for each pval.

**Note:** As they are generalization of binomial distribution their visual representation and similarity of normal distribution is same as that of multiple binomial distributions.

# **Exponential Distribution**

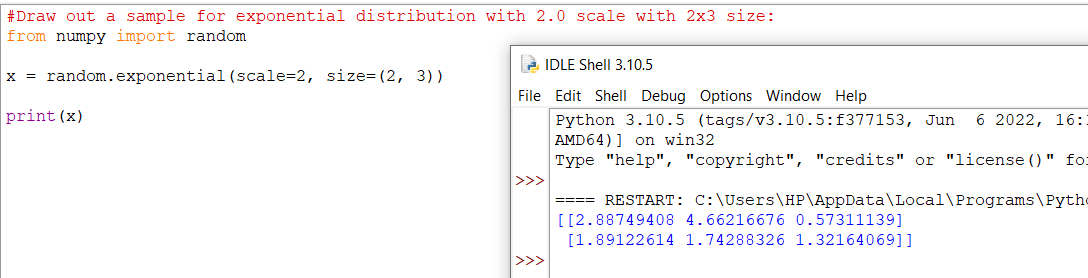
Exponential Distribution

Exponential distribution is used for describing time till next event e.g. failure/success etc.

It has two parameters:

scale - inverse of rate ( see lam in poisson distribution ) defaults to 1.0.

size - The shape of the returned array.



## Visualization of Exponential Distribution

## 

## Relation Between Poisson and Exponential Distribution

Poisson distribution deals with number of occurences of an event in a time period whereas exponential distribution deals with the time between these events.

# **Chi Square Distribution**

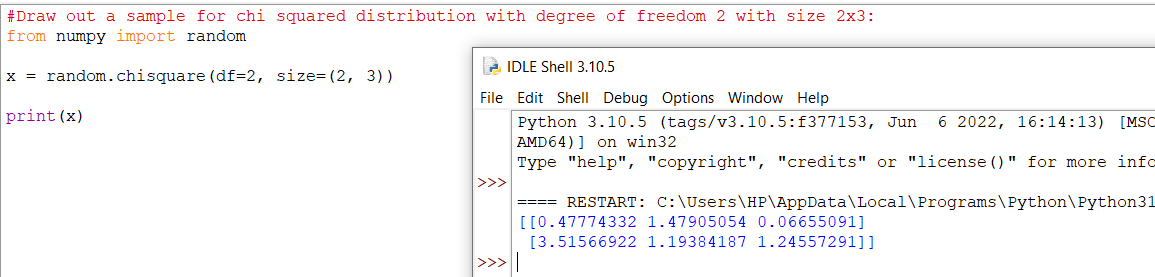
## Chi Square Distribution

Chi Square distribution is used as a basis to verify the hypothesis.

It has two parameters:

df - (degree of freedom).

size - The shape of the returned array.



## Visualization of Chi Square Distribution

## 

# **Rayleigh Distribution**

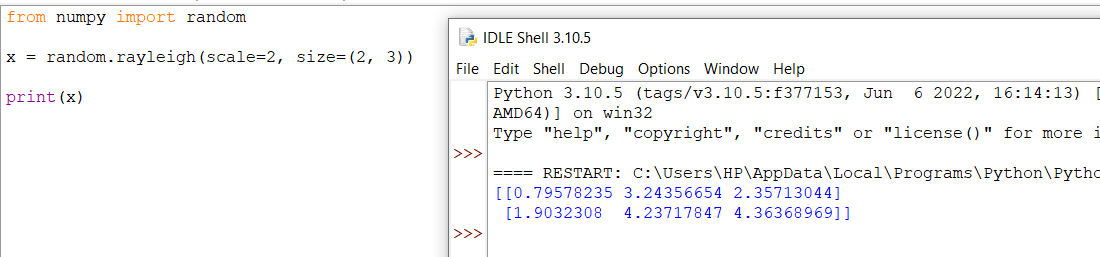
# Rayleigh Distribution

Rayleigh distribution is used in signal processing.

It has two parameters:

scale - (standard deviation) decides how flat the distribution will be default 1.0).

size - The shape of the returned array.



## Visualization of Rayleigh Distribution

## 

## Similarity Between Rayleigh and Chi Square Distribution

At unit stddev and 2 degrees of freedom rayleigh and chi square represent the same distributions.

# **Pareto Distribution**

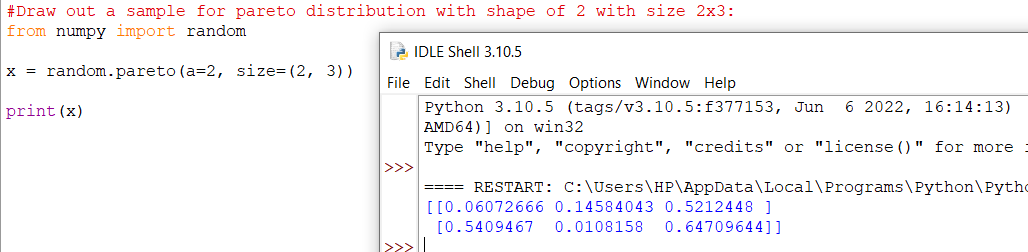
Pareto Distribution

A distribution following Pareto's law i.e. 80-20 distribution (20% factors cause 80% outcome).

It has two parameter:

a - shape parameter.

size - The shape of the returned array.



## Visualization of Pareto Distribution

## 

# **Zipf Distribution**

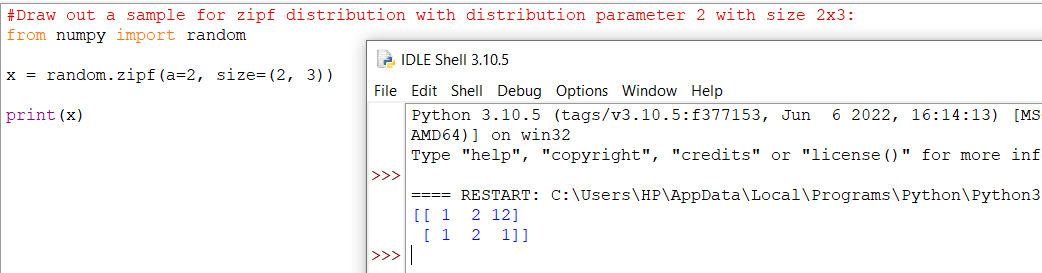
Zipf distritutions are used to sample data based on zipf's law.

**Zipf's Law:** In a collection, the nth common term is 1/n times of the most common term. E.g. the 5th most common word in English occurs nearly 1/5 times as often as the most common word.

It has two parameters:

a - distribution parameter.

size - The shape of the returned array.



## Visualization of Zipf Distribution

Sample 1000 points but plotting only ones with value < 10 for more meaningful chart.



# **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.

## What is Vectorization?

Converting iterative statements into a vector based operation is called vectorization.

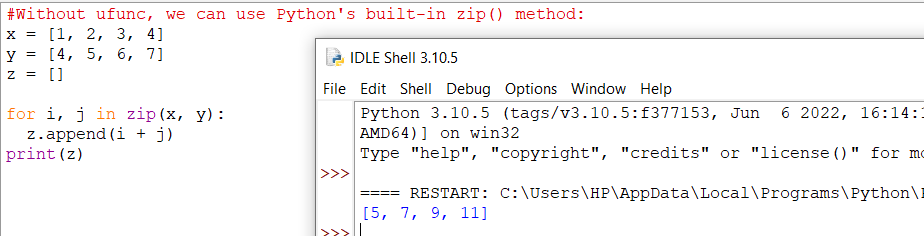
It is faster as modern CPUs are optimized for such operations.

### **Add the Elements of Two Lists**

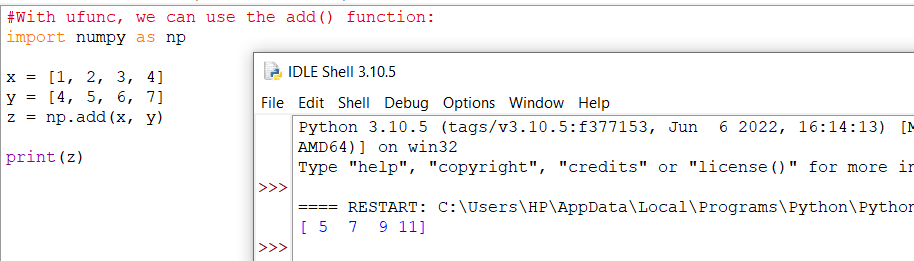
list 1: [1, 2, 3, 4]

list 2: [4, 5, 6, 7]

One way of doing it is to iterate over both of the lists and then sum each elements.



NumPy has a ufunc for this, called add(x, y) that will produce the same result.



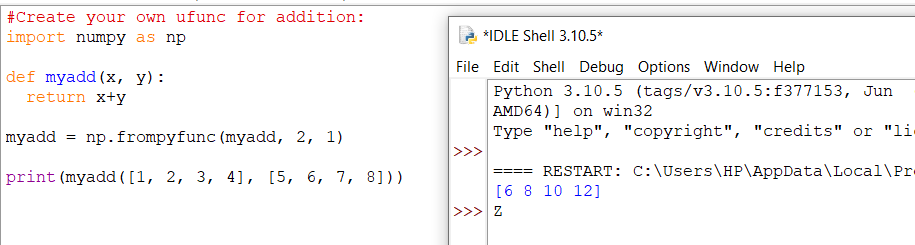
# **Create Your Own ufunc**

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:

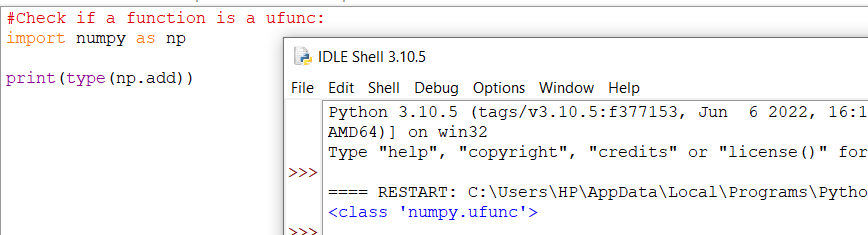
1. function - the name of the function.
2. inputs - the number of input arguments (arrays).
3. outputs - the number of output arrays.



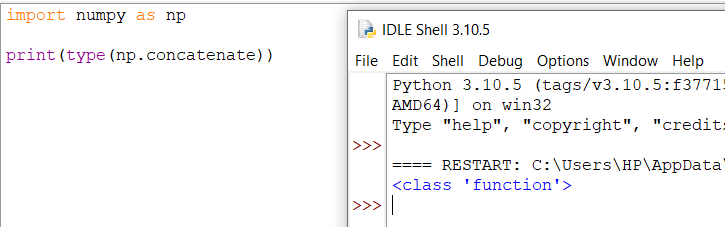
## Check if a Function is a ufunc

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

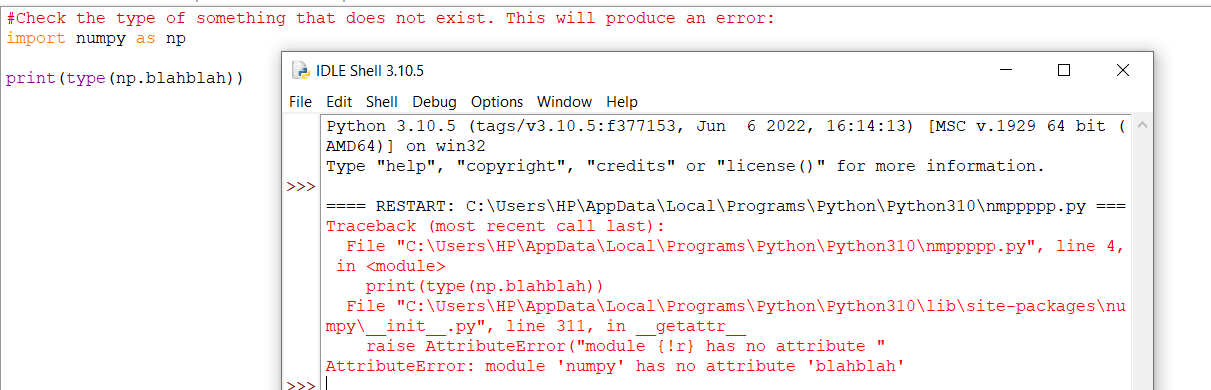
A ufunc should return <class 'numpy.ufunc'>.



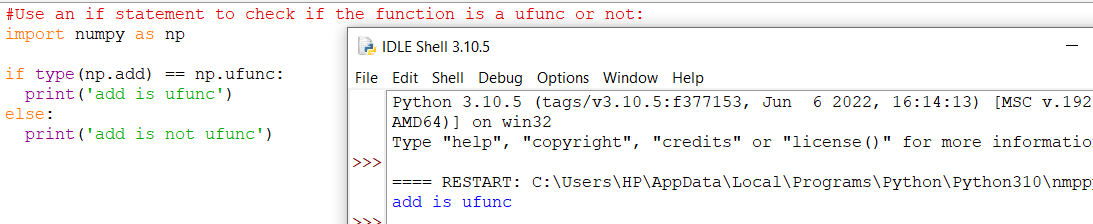
If it is not a ufunc, it will return another type, like this built-in NumPy function for joining two or more arrays:



If the function is not recognized at all, it will return an error:



To test if the function is a ufunc in an if statement, use the numpy.ufunc value (or np.ufunc if you use np as an alias for numpy):



# **Simple Arithmetic**

Simple Arithmetic

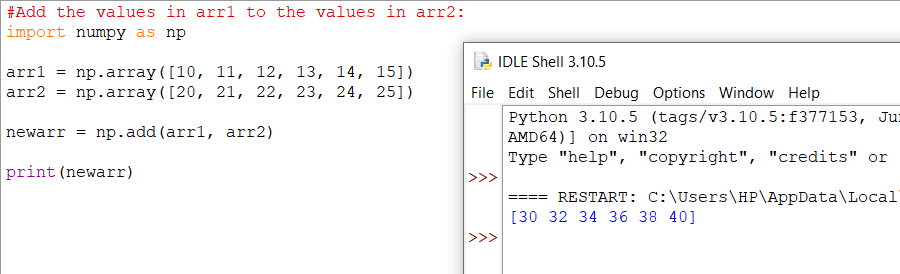
You could use arithmetic operators + - \* / directly between NumPy arrays, but this section discusses an extension of the same where we have functions that can take any array-like objects e.g. lists, tuples etc. and perform arithmetic conditionally.

**Arithmetic Conditionally:** means that we can define conditions where the arithmetic operation should happen.

All of the discussed arithmetic functions take a where parameter in which we can specify that condition.

## Addition

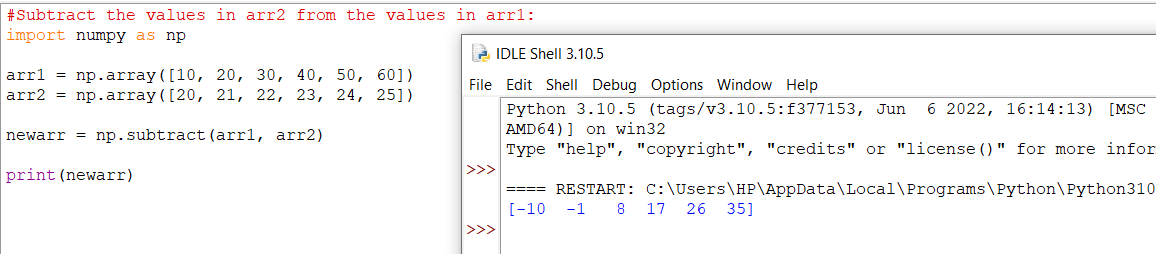
The add() function sums the content of two arrays, and return the results in a new array.



The example above will return [30 32 34 36 38 40] which is the sums of 10+20, 11+21, 12+22 etc.

Subtraction

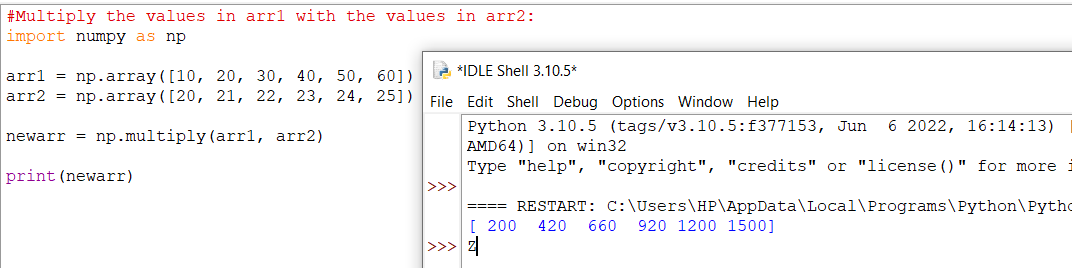
The subtract() function subtracts the values from one array with the values from another array, and return the results in a new array.



The example above will return [-10 -1 8 17 26 35] which is the result of 10-20, 20-21, 30-22 etc.

## Multiplication

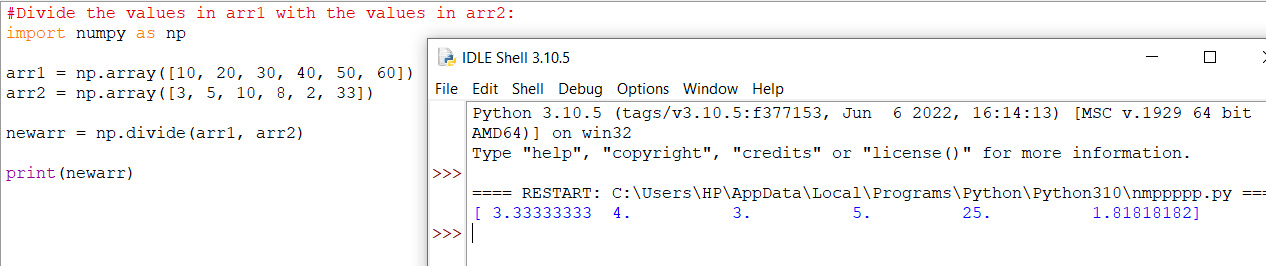
The multiply() function multiplies the values from one array with the values from another array, and return the results in a new array.



The example above will return [200 420 660 920 1200 1500] which is the result of 10\*20, 20\*21, 30\*22 etc.

Division

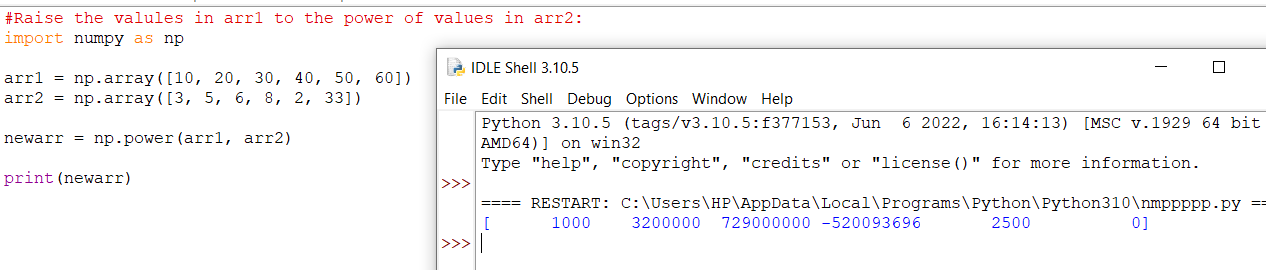
The divide() function divides the values from one array with the values from another array, and return the results in a new array.



The example above will return [3.33333333 4. 3. 5. 25. 1.81818182] which is the result of 10/3, 20/5, 30/10 etc.

## Power

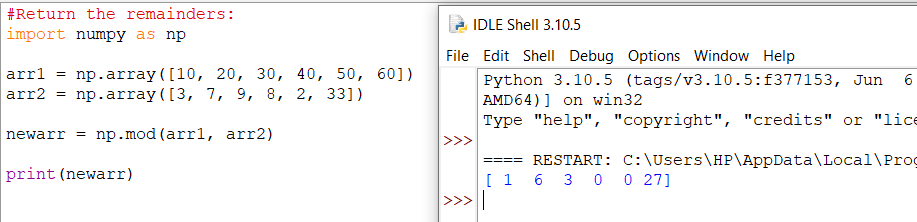
The power() function rises the values from the first array to the power of the values of the second array, and return the results in a new array.



The example above will return [1000 3200000 729000000 6553600000000 2500 0] which is the result of 10\*10\*10, 20\*20\*20\*20\*20, 30\*30\*30\*30\*30\*30 etc.

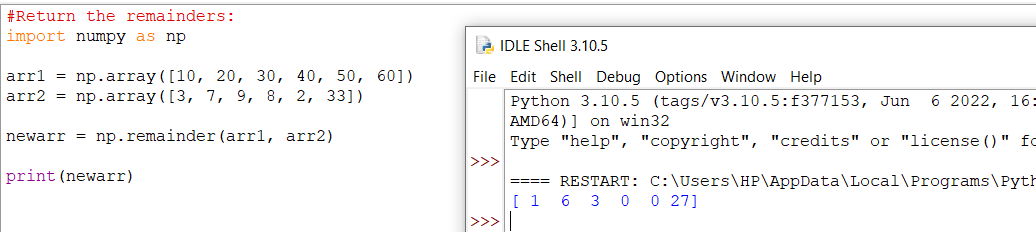
## Remainder

Both the mod() and the remainder() functions return the remainder of the values in the first array corresponding to the values in the second array, and return the results in a new array.



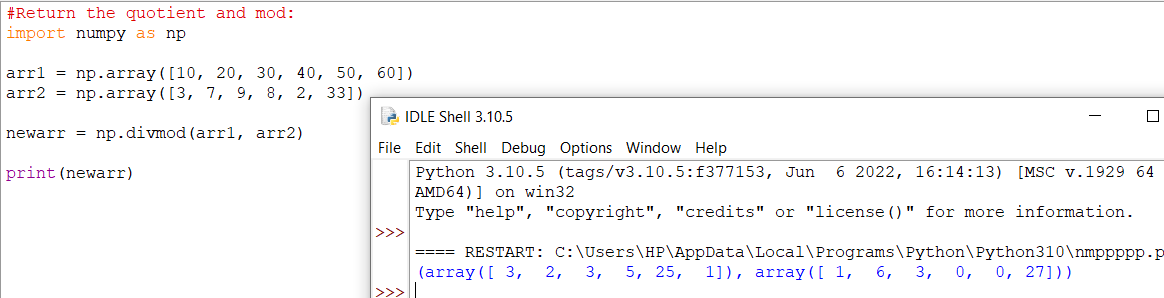
The example above will return [1 6 3 0 0 27] which is the remainders when you divide 10 with 3 (10%3), 20 with 7 (20%7) 30 with 9 (30%9) etc.

You get the same result when using the remainder() function:



## Quotient and Mod

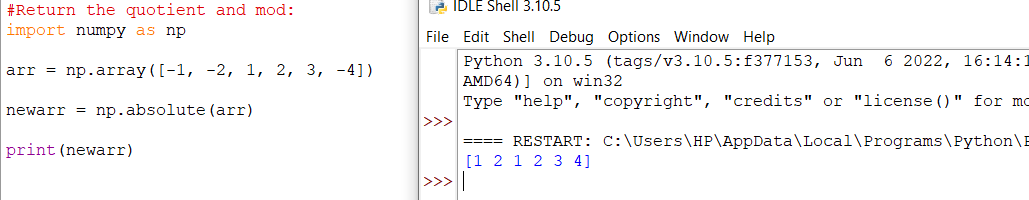
The divmod() function return both the quotient and the the mod. The return value is two arrays, the first array contains the quotient and second array contains the mod.



The example above will return:  
(array([3, 2, 3, 5, 25, 1]), array([1, 6, 3, 0, 0, 27]))  
The first array represents the quotients, (the integer value when you divide 10 with 3, 20 with 7, 30 with 9 etc.  
The second array represents the remainders of the same divisions.

## Absolute Values

Both the absolute() and the abs() functions do the same absolute operation element-wise but we should use absolute() to avoid confusion with python's inbuilt math.abs()



The example above will return [1 2 1 2 3 4].

# **Rounding Decimals**

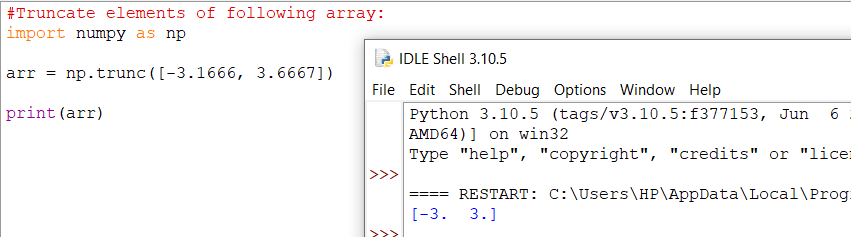
Rounding Decimals

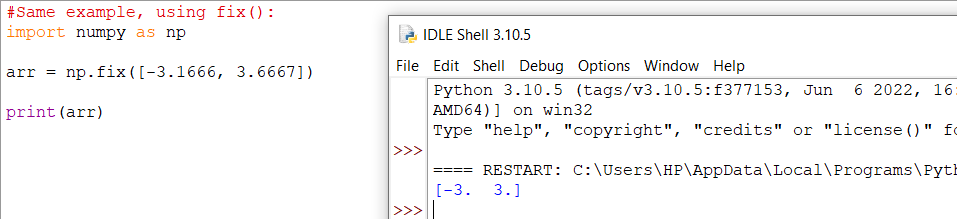
There are primarily five ways of rounding off decimals in NumPy:

* truncation
* fix
* rounding
* floor
* ceil

## Truncation

Remove the decimals, and return the float number closest to zero. Use the trunc() and fix() functions.

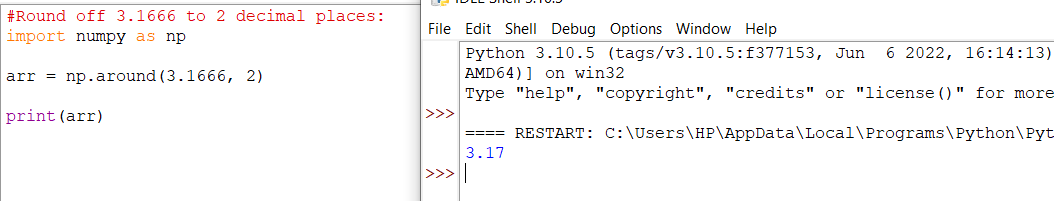




## Rounding

The around() function increments preceding digit or decimal by 1 if >=5 else do nothing.

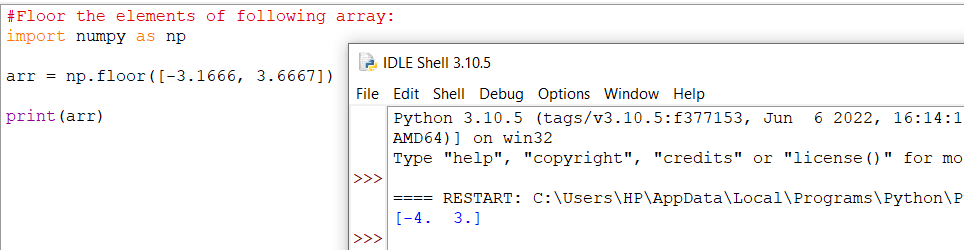
E.g. round off to 1 decimal point, 3.16666 is 3.2



## Floor

The floor() function rounds off decimal to nearest lower integer.

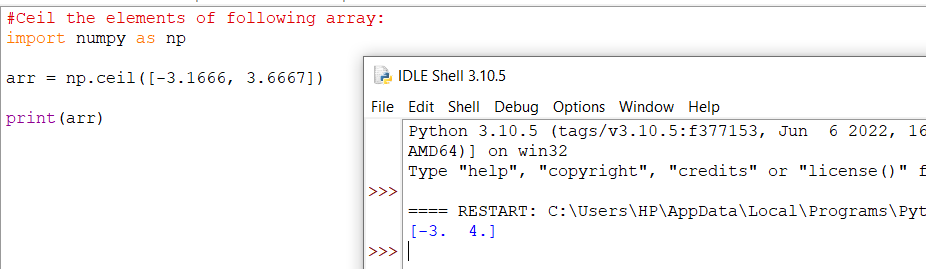
E.g. floor of 3.166 is 3.



## Ceil

The ceil() function rounds off decimal to nearest upper integer.

E.g. ceil of 3.166 is 4.



# **NumPy Logs**

# Logs

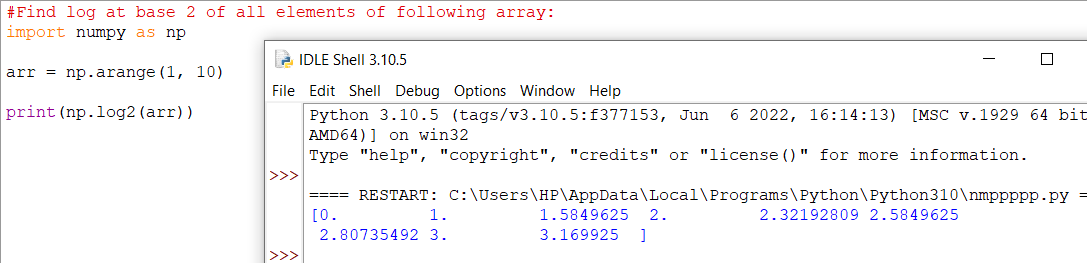
NumPy provides functions to perform log at the base 2, e and 10.

We will also explore how we can take log for any base by creating a custom ufunc.

All of the log functions will place -inf or inf in the elements if the log can not be computed.

## Log at Base 2

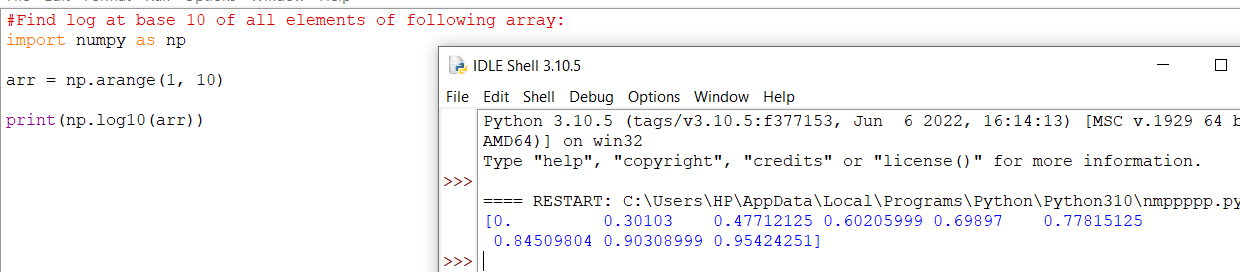
Use the log2() function to perform log at the base 2.



**Note:** The arange(1, 10) function returns an array with integers starting from 1 (included) to 10 (not included).

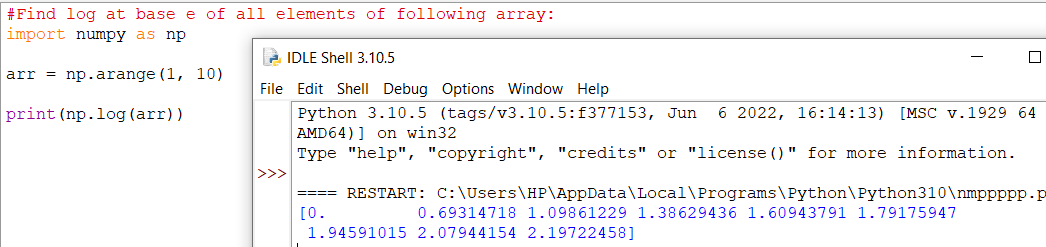
## Log at Base 10

Use the log10() function to perform log at the base 10.



## Natural Log, or Log at Base e

Use the log() function to perform log at the base e.



## Log at Any Base

NumPy does not provide any function to take log at any base, so we can use the frompyfunc() function along with inbuilt function math.log() with two input parameters and one output parameter:

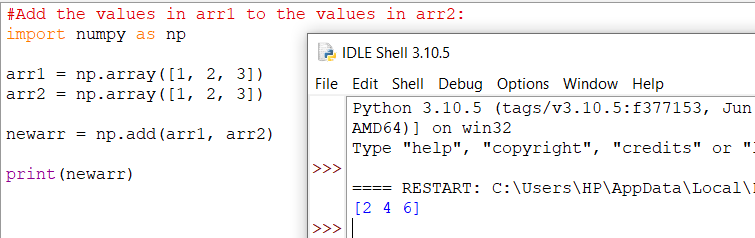
### **Example**

from math import log  
import numpy as np  
  
nplog = np.frompyfunc(log, 2, 1)  
  
print(nplog(100, 15))

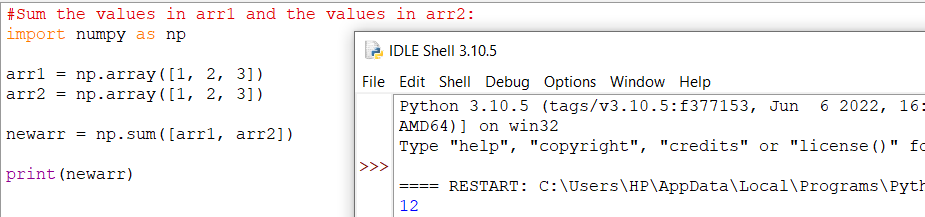
# **NumPy Summations**Summations

What is the difference between summation and addition?

Addition is done between two arguments whereas summation happens over n elements.



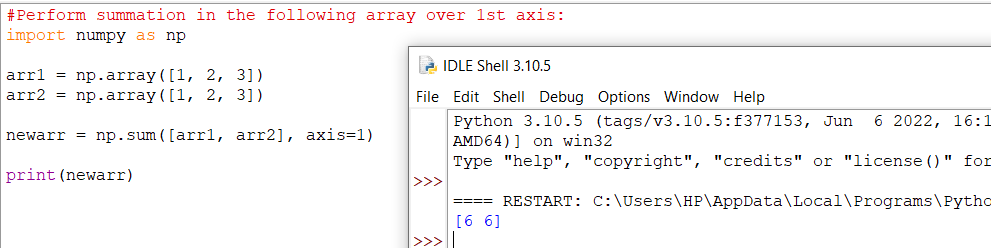
**Returns:** [2 4 6]

****

**Returns:** 12

## Summation Over an Axis

If you specify axis=1, NumPy will sum the numbers in each array.



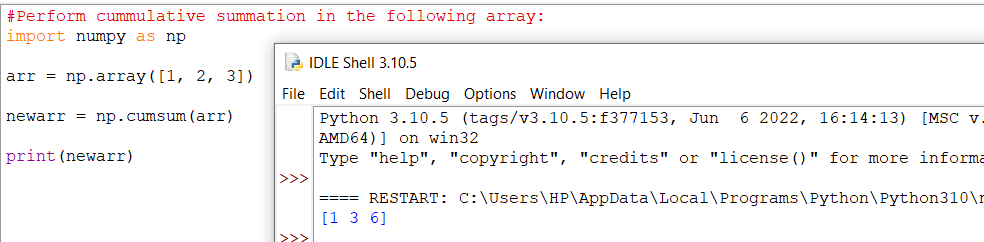
**Returns:** [6 6]

## Cummulative Sum

Cummulative sum means partially adding the elements in array.

E.g. The partial sum of [1, 2, 3, 4] would be [1, 1+2, 1+2+3, 1+2+3+4] = [1, 3, 6, 10].

Perfom partial sum with the cumsum() function.

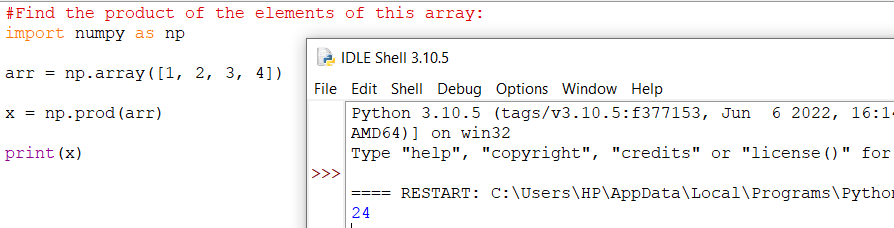


**Returns:** [1 3 6]

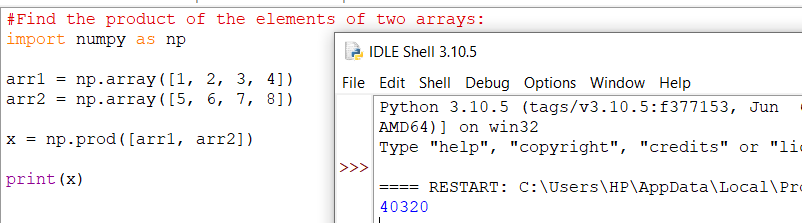
# **NumPy Products**

Products

To find the product of the elements in an array, use the prod() function.

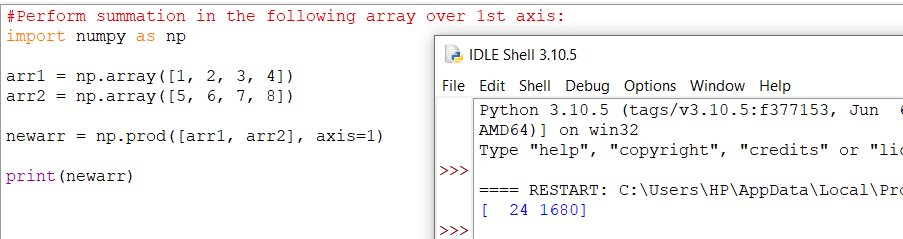


**Returns:** 24 because 1\*2\*3\*4 = 24

**Returns:** 40320 because 1\*2\*3\*4\*5\*6\*7\*8 = 40320

## Product Over an Axis

If you specify axis=1, NumPy will return the product of each array.



**Returns:** [24 1680]

Cummulative Product

Cummulative product means taking the product partially.

E.g. The partial product of [1, 2, 3, 4] is [1, 1\*2, 1\*2\*3, 1\*2\*3\*4] = [1, 2, 6, 24]

Perfom partial sum with the cumprod() function.



**Returns:** [5 30 210 1680]

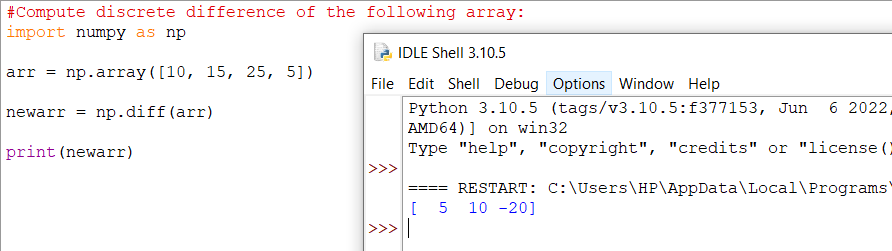
# **NumPy Differences**

Differences

A discrete difference means subtracting two successive elements.

E.g. for [1, 2, 3, 4], the discrete difference would be [2-1, 3-2, 4-3] = [1, 1, 1]

To find the discrete difference, use the diff() function.



**Returns:** [5 10 -20] because 15-10=5, 25-15=10, and 5-25=-20

We can perform this operation repeatedly by giving parameter n.

E.g. for [1, 2, 3, 4], the discrete difference with n = 2 would be [2-1, 3-2, 4-3] = [1, 1, 1] , then, since n=2, we will do it once more, with the new result: [1-1, 1-1] = [0, 0]

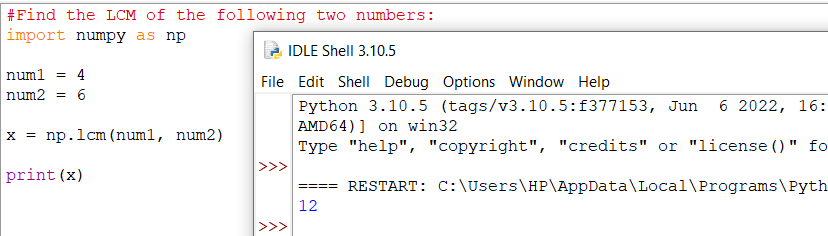


**Returns:** [5 -30] because: 15-10=5, 25-15=10, and 5-25=-20 AND 10-5=5 and -20-10=-30

# **NumPy LCM Lowest Common Multiple**

## Finding LCM (Lowest Common Multiple)

The Lowest Common Multiple is the smallest number that is a common multiple of two numbers.

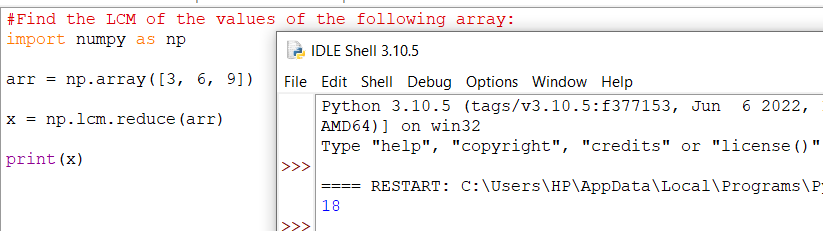


**Returns:** 12 because that is the lowest common multiple of both numbers (4\*3=12 and 6\*2=12).

## Finding LCM in Arrays

To find the Lowest Common Multiple of all values in an array, you can use the reduce() method.

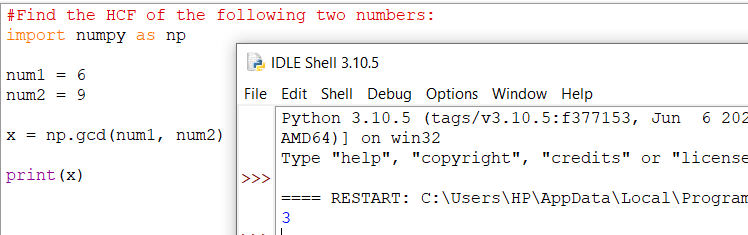
The reduce() method will use the ufunc, in this case the lcm() function, on each element, and reduce the array by one dimension.

**Returns:** 18 because that is the lowest common multiple of all three numbers (3\*6=18, 6\*3=18 and 9\*2=18).

# **NumPy GCD Greatest Common Denominator**

## Finding GCD (Greatest Common Denominator)

The GCD (Greatest Common Denominator), also known as HCF (Highest Common Factor) is the biggest number that is a common factor of both of the numbers.

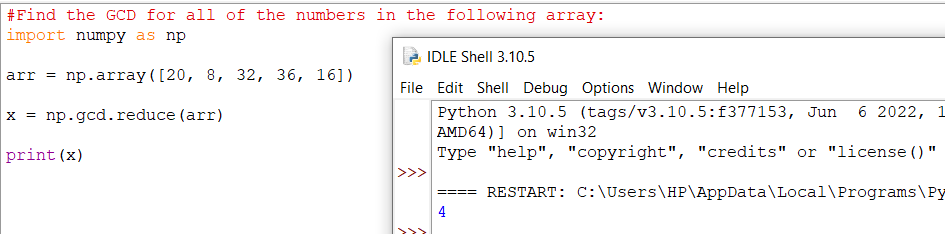


**Returns:** 3 because that is the highest number both numbers can be divided by (6/3=2 and 9/3=3).

## Finding GCD in Arrays

To find the Highest Common Factor of all values in an array, you can use the reduce() method.

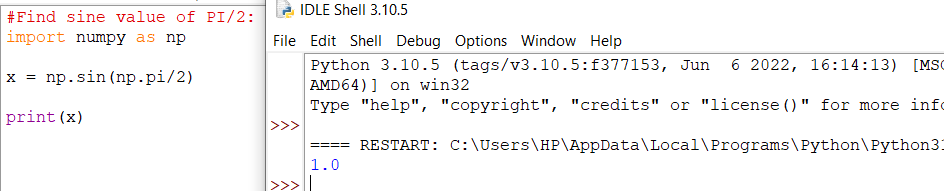
The reduce() method will use the ufunc, in this case the gcd() function, on each element, and reduce the array by one dimension.

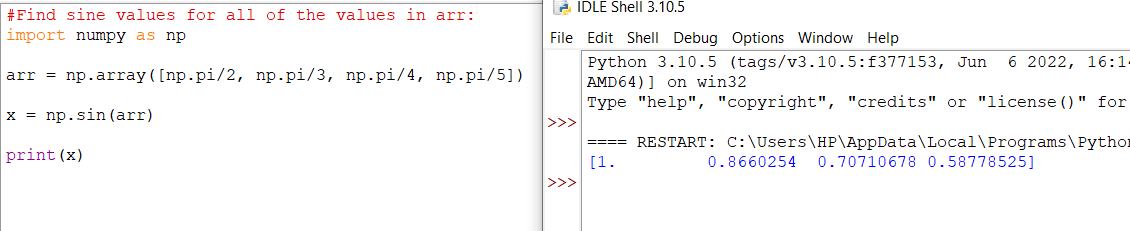
**Returns:** 4 because that is the highest number all values can be divided by.

# **NumPy Trigonometric Functions**

## Trigonometric Functions

NumPy provides the ufuncs sin(), cos() and tan() that take values in radians and produce the corresponding sin, cos and tan values.

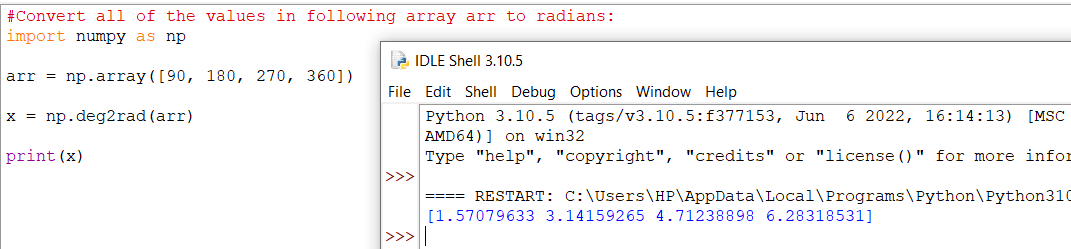




## Convert Degrees Into Radians

By default all of the trigonometric functions take radians as parameters but we can convert radians to degrees and vice versa as well in NumPy.

**Note:** radians values are pi/180 \* degree\_values.



## Radians to Degrees

## 

## Finding Angles

Finding angles from values of sine, cos, tan. E.g. sin, cos and tan inverse (arcsin, arccos, arctan).

NumPy provides ufuncs arcsin(), arccos() and arctan() that produce radian values for corresponding sin, cos and tan values given.



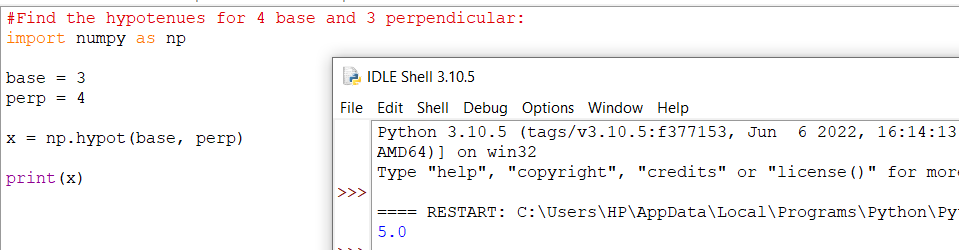
## Angles of Each Value in Arrays

## 

## Hypotenues

Finding hypotenues using pythagoras theorem in NumPy.

NumPy provides the hypot() function that takes the base and perpendicular values and produces hypotenues based on pythagoras theorem.

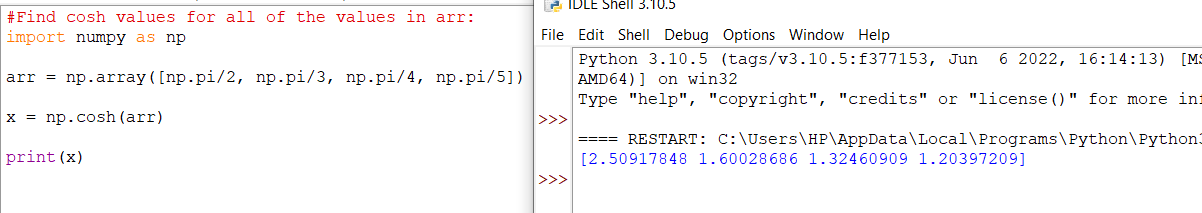


# **NumPy Hyperbolic Functions**

## Hyperbolic Functions

NumPy provides the ufuncs sinh(), cosh() and tanh() that take values in radians and produce the corresponding sinh, cosh and tanh values..

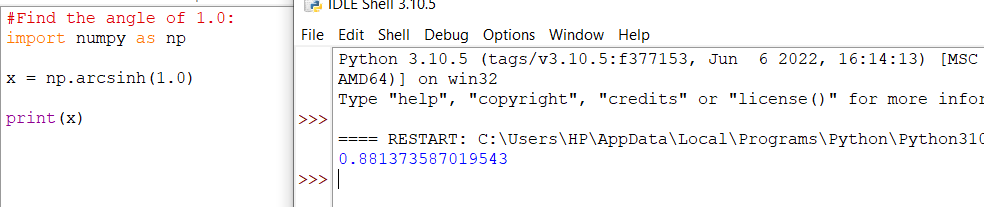
### 



## Finding Angles

Finding angles from values of hyperbolic sine, cos, tan. E.g. sinh, cosh and tanh inverse (arcsinh, arccosh, arctanh).

Numpy provides ufuncs arcsinh(), arccosh() and arctanh() that produce radian values for corresponding sinh, cosh and tanh values given.



## Angles of Each Value in Arrays

## 

# **NumPy Set Operations**

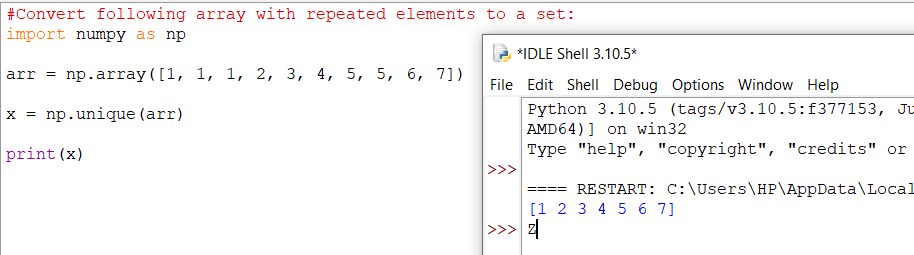
## What is a Set

A set in mathematics is a collection of unique elements.

Sets are used for operations involving frequent intersection, union and difference operations.

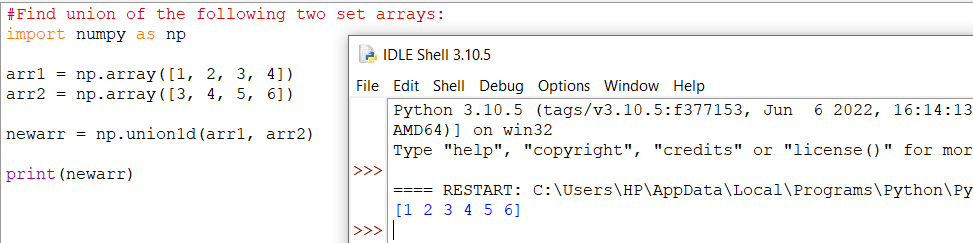
## Create Sets in NumPy

We can use NumPy's unique() method to find unique elements from any array. E.g. create a set array, but remember that the set arrays should only be 1-D arrays.



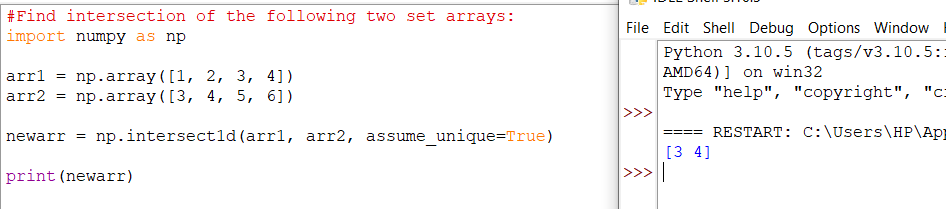
## Finding Union

To find the unique values of two arrays, use the union1d() method.



## Finding Intersection

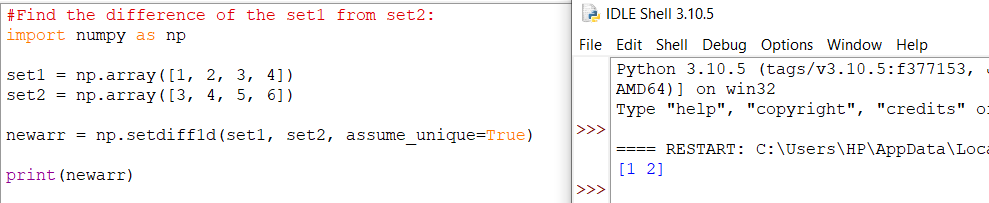
To find only the values that are present in both arrays, use the intersect1d() method.



**Note:** the intersect1d() method takes an optional argument assume\_unique, which if set to True can speed up computation. It should always be set to True when dealing with sets.

## Finding Difference

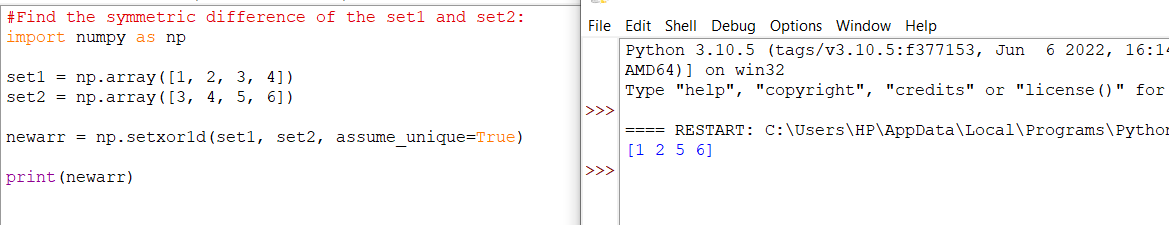
To find only the values in the first set that is NOT present in the seconds set, use the setdiff1d() method.



**Note:** the setdiff1d() method takes an optional argument assume\_unique, which if set to True can speed up computation. It should always be set to True when dealing with sets.

## Finding Symmetric Difference

To find only the values that are NOT present in BOTH sets, use the setxor1d() method.



**Note:** the setxor1d() method takes an optional argument assume\_unique, which if set to True can speed up computation. It should always be set to True when dealing with sets.