

1 Introduction

Object-oriented programming (OOP) is a widely used concept to write powerful applications. Oriented simply means *directed toward*. So object-oriented is functionally directed towards modeling objects. This is one of the many techniques used for modeling complex systems by describing a collection of interacting objects via their data and behaviour [1]. It has some advantages over other design patterns. Development is faster and cheaper, with better software maintainability. This, in turn, leads to higher-quality software, which is also extensible with new methods and attributes. OOP is a programming paradigm that provides a means of structuring programs so that properties and behaviors are bundled into individual objects. It uses the concept of objects and classes. A class can be thought of as a 'blueprint' for objects. An object has two characteristics: attributes (characteristics they possess) and behavior (actions they perform). An example of a class is the class Dog. Don't think of it as a specific dog, or your own dog. We're describing what a dog *is* and can *do*, in general. Dogs usually have a name and age; these are instance attributes. Dogs can also bark; this is a method.

A very important example of a programming language that supports OOP is Python. Python is a general-purpose, versatile, and powerful programming language. It's concise and easy to read. Python offers a number of benefits compared to other programming languages like Java, C++ or R. It's a dynamic language, with high-level data types. This means that development happens much faster than with Java or C++. It does not require the programmer to declare types of variables and arguments. This also makes Python easier to understand and learn for beginners, its code being more readable and intuitive.

These workshops are designed to provide introductory knowledge to the basics of Python programming language. You will start by doing basic arithmetic and using variables in Python. You will learn how to handle data structures such as Python lists and Numpy arrays. You will also learn about loops, conditions, and functions. To top it off, you will learn about visualizing your data using Python.

1.1 Why Python?

Python is a general-purpose and in-demand programming language that has become one of the most popular languages in the world. In recent years, as shown in Fig(1), Python has demonstrated rapid growth as the introductory language in computer science courses and is becoming a preferable programming language in industry and academia [2], [3].

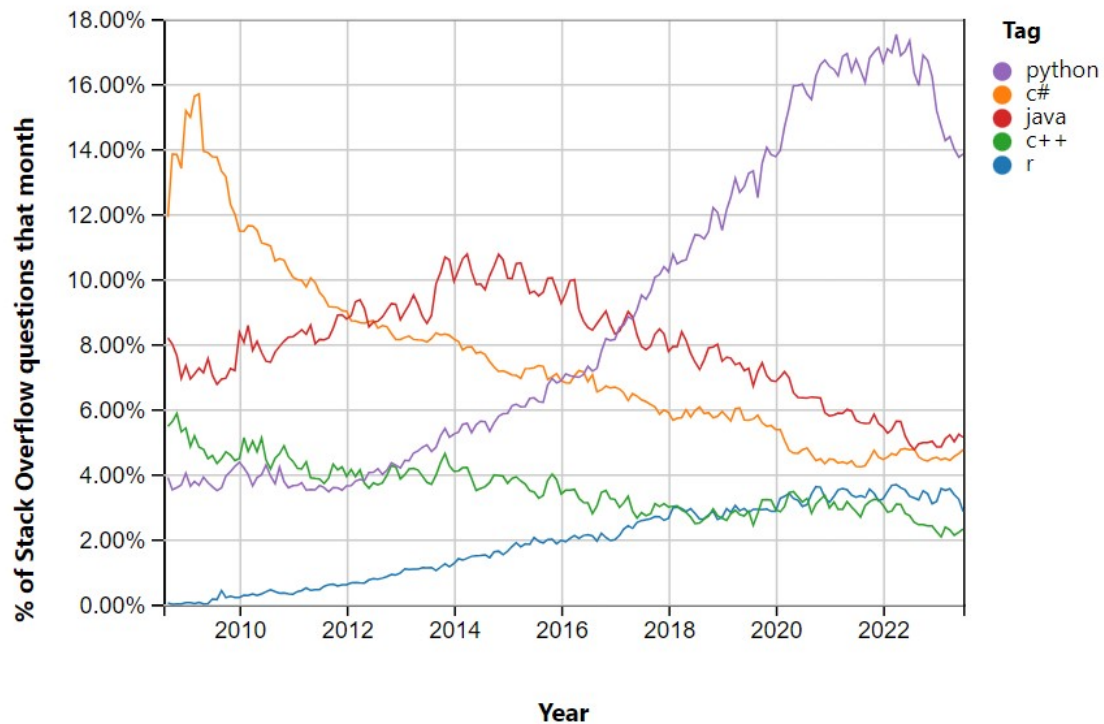


Figure 1: A snapshot of how programming languages have trended over time based on their tags on Stack Overflow since 2008.

Python has libraries for data loading, visualisation, statistics, natural language processing, image processing, and more. This vast ecosystem of tools, packages, and libraries addresses a wide-ranging number of programming scenarios and provides scientists and engineers with a large array of general- and special-purpose functionality. In addition, Python has an easy-to-learn structure, runs on multiple operating systems, is license-fee-free, and has excellent online documentation. Moreover, Python programs tend to be much shorter than equivalent programs in C or C++, or Java. This is because Python is a high-level language. It has data types that allow you to express complex operations in a very concise manner. This surely makes Python stand out from all the other programming languages.

1.2 Python 2 Vs Python 3

There are two major versions of Python that are widely used at the moment: Python 2 (more precisely, 2.7) and Python 3 (with the latest release being 3.9 at the time of writing). This sometimes leads to some confusion about which version to use or learn. However, when looking at the current state of both versions, the

answer is relatively easy. Python 2 is outdated and no longer actively developed, whereas Python 3 contains major changes and is constantly in demand [4]. While Python 2 is still in use for legacy reasons, Python 3 is the current standard. It is worth mentioning that these two versions are not compatible. This means that if you write your code in Python 3, somebody running Python 2 will not be able to run that code.

In these workshops, we will be using Python 3 due to the fact that all new standard library improvements are only available by default in Python 3. Python 3 is also easier for newcomers to learn, and several aspects of the core language are more consistent than those in Python 2.

1.3 Python Essential Libraries

Python is an ocean of libraries that serve various purposes. This means there's a good chance that whatever you're trying to build, there's already a package (or a library) that can make the development easier for you. Normally, as we shall see later, Python requires you to import this library into your project first before you start coding. Importing a library is like getting a piece of lab equipment out of a storage locker and setting it up on the bench for use in a project [4]. As a Python beginner, you should familiarise yourself with the most important, useful, and ubiquitous Python packages since they will come up more often. In the following, we will outline the main and most occurring and used libraries in Python that you should know [5].

1.3.1 Data Processing and Modeling

numpy It is one of the fundamental packages for scientific computing in Python. **numpy** provides tools to help build multi-dimensional arrays and perform calculations on the data stored in them. You can solve algebraic formulae, perform common statistical operations, and much more.

scipy As the name suggests, **scipy** is mainly used for its scientific functions and mathematical functions derived from **numpy**. **scipy** contains modules for optimization, linear algebra, integration, interpolation, special functions, Fast Fourier Transform (FFT), signal and image processing, ordinary differential equations (ODE) solvers and other scientific computing tasks common in science and engineering.

pandas A Major part of an engineer's job is to understand and clean data, i.e., data exploration and manipulation. **pandas** is primarily used for data analysis, and it is one of the most commonly used Python libraries. It provides you with

some of the most useful sets of tools to explore, clean, and analyse your data. **pandas** is a library created to help developers work with “labeled” and “relational” data intuitively. It’s based on two main data structures: “Series” (one-dimensional, like a list of items) and “DataFrames” (two-dimensional, like a table with multiple columns). **pandas** allows converting data structures to DataFrame objects, handling missing data, adding/deleting columns from DataFrame, imputing missing files, and plotting data.

1.3.2 Data Visualisation

matplotlib This is the primary scientific plotting library in Python. It provides functions to generate data visualisations such as line charts, histograms, scatter plots, and so on. Data visualization gives us a clear idea of what the information means by giving it visual context through maps and graphs. This makes it easier to identify trends, patterns, and outliers within large data sets.

2 Anaconda Distribution & Jupiter Notebook

As mentioned before, Python has a sea of libraries to perform and tackle several scientific computing scenarios. But instead of installing hundreds of packages manually one at a time, we will be using a Python distribution. A distribution consists of the core Python packages and several hundred modules, all working seamlessly together. All of this is available through a single download. There are currently several Python-integrated development environments (IDEs) that allow one to write, test, and debug Python software, e.g., IDLE, Microsoft Visual Studio, PyCharm, PyDev, Thonny, etc. The one that we will be using in these workshops is called Anaconda distribution. One of Anaconda's very useful development environments is Jupyter Notebook. Jupyter Notebook is a free, open-source, interactive web tool known as a computational notebook, which researchers can use to combine software code, computational output, explanatory text, and multimedia resources in a single document. Computational notebooks have been around for decades, but Jupyter in particular has exploded in popularity over the past couple of years due to its flexibility and easy-to-use interface. You can use Jupyter Notebooks for nearly all sorts of science and engineering tasks including data cleaning and transformation, numerical simulation, exploratory data analysis, data visualization, statistical modeling, machine learning, deep learning, and much more [4].

2.1 Download Anaconda

Anaconda can be easily downloaded onto your computer/laptop using the UCL software database following the next steps, as shown in Fig(2):

1. Open UCL Software Database using this link: <https://swdb.ucl.ac.uk/>, and then enter your UCL username and password.
2. In the "Search" box, type "Anaconda" and then hit "enter" to look for the package in the dataset.
3. The search should bring back one result: "Anaconda Python". Click on it to view its details.
4. The main page of the package details has several horizontal tabs that provide different tools, e.g., general, licenses, availability, etc. Scroll down and click on "Downloads".
5. The "Downloads" tab has a link that directs the user to the Anaconda website.

6. Locate your download and install Anaconda.

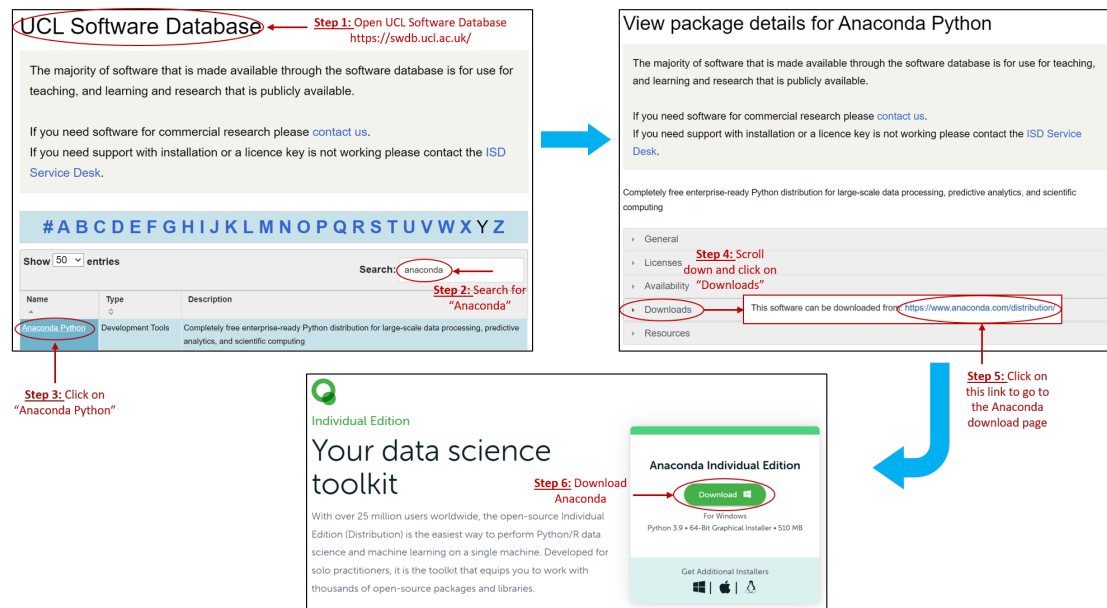


Figure 2: Download Anaconda Distribution.

2.2 Launch Jupyter Notebook

You can launch Jupyter Notebook easily in a straightforward manner by double-clicking on the Anaconda icon and then scrolling down to find the Jupyter Notebook application, and then hitting "Launch" to open it.

Once you are in the Jupyter Notebook interface, you can see all the files in your current directory. All Jupyter Notebooks are identifiable by the notebook icon next to their name. If you already have a Jupyter Notebook in your current directory that you want to view, find it in your files list and click it to open. Notebooks currently running will have a green icon, while non-running ones will be grey. To find all currently running notebooks, click on the Running tab to see a list.

2.3 Create, Understand and Run a Jupyter Notebook

To create a new notebook, go to "New" and select Notebook - Python 3. If you have other Jupyter Notebooks on your system that you want to use, you can click "Upload" and navigate to that particular file.

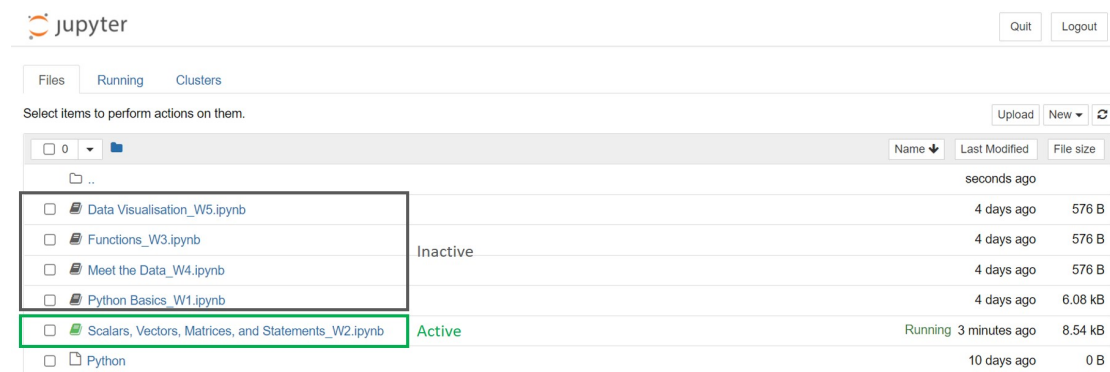


Figure 3: Jupyter Notebook Interface



Figure 4: Create a new Jupyter Notebook file

When you open a new Jupyter notebook, you will notice that it contains a rectangular cell. Cells are how notebooks are structured and are the areas where you write your code. The browser then passes the code to a back-end “kernel” which runs the code and returns the results. This rectangular-cell structure has major flexibility when it comes to managing your code. Rather than writing and re-writing an entire program, you can write lines of code and run them one at a time. Then, if you need to make a change, you can go back and make your edit and rerun the program again, all in the same window. To run a piece of code, click on the cell to select it, then press **SHIFT+ENTER**. After you run a cell, the output of the cell’s code will appear in the space below. The toolbar has several shortcut buttons for popular actions. From left to right: save, add a new cell, cut selected cells, copy selected cells, paste cells below, move selected cells up, move selected cells down, run, interrupt the kernel, restart the kernel, a dropdown that allows you to change the cell type, and a shortcut to open the command palette.

The navigation bar also contains several tabs with multiple options. From left to right: File, Edit, View, Insert, Cell, Kernel, Widgets, Help.

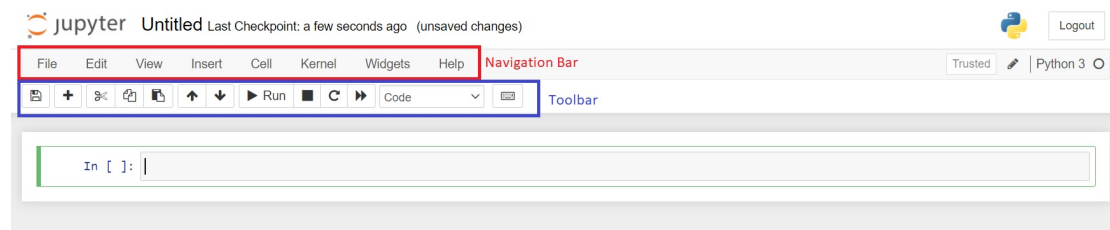


Figure 5: Jupyter Notebook Toolbar and Navigation Bar

To create new cells, use the plus (+) button in the toolbar or from the “Cell” tab in the navigation bar. To cut, copy, delete, or just generally edit cells - select the cell you want to modify and go to the “Edit” tab in the navigation bar to see your options.

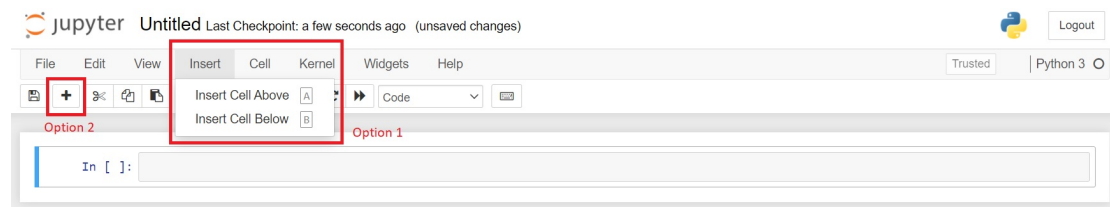


Figure 6: Two options on how to create a new cell.

In addition to running lines of code, you can also include text-only cells that use Markdown format to organize your notebooks. Cells in Jupyter are set up to be in code mode by default. To change its type to a Markdown, there are two options to follow: either choose the “Markdown” options from the toolbar or go to the “Cell” tab in the navigation bar, choose “Cell Type” and then “Markdown” as its type.

Jupyter Notebook files are saved as you go. They will exist in your directory as a JSON file with the extension `.ipynb`. You can also export Jupyter Notebooks in other formats, such as HTML. To do so, go to the “File” menu, scroll down to “Download as” and select the type of file you are looking for as shown in Fig(8).

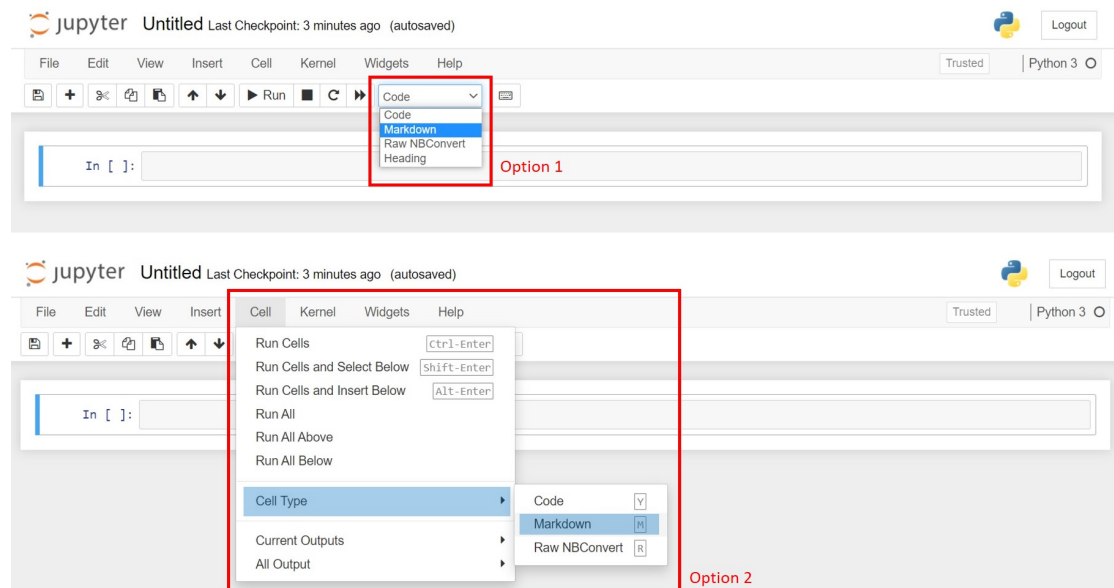


Figure 7: Two options on how to change the cell type to a “Markdown” option.

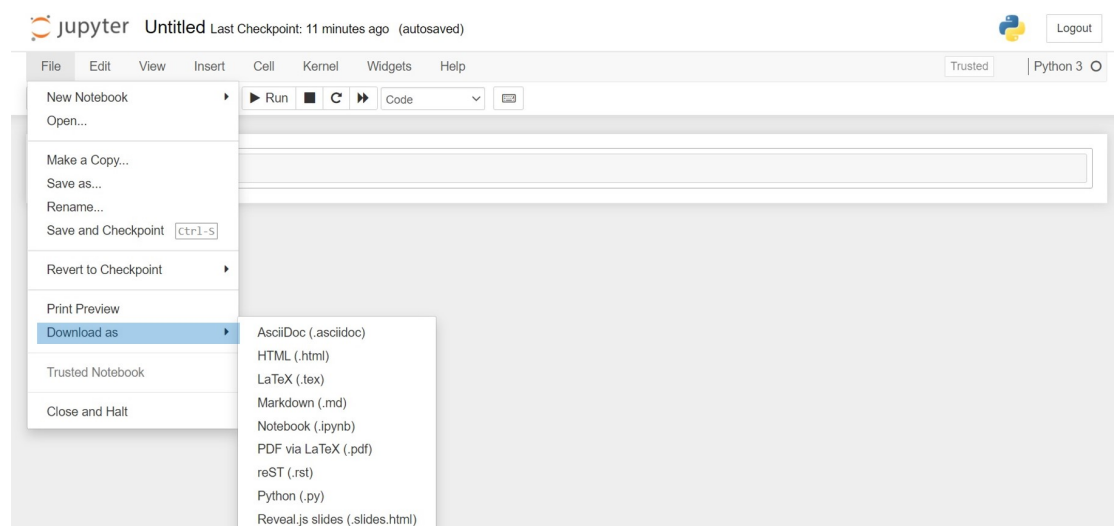


Figure 8: Downloading a Jupyter file as in another format.

3 Python Basics

3.1 Objects

Python contains many data types as part of the core language. As mentioned previously, Python is an object-oriented programming language, so it's important to understand that all data in a Python program is represented by objects and by relationships between objects. Objects whose value can change in the course of program execution are said to be mutable objects, whereas objects whose value is unchangeable after they've been created are called immutable. Each object in Python has three characteristics. These characteristics are:

- **object type:** Object type tells Python what kind of an object it is dealing with. A type could be a number, a string, a list, or something else.
- **object value:** Object value is the data value that is contained by the object. This could be a specific number, for example.
- **object identity:** you can think of object identity as an identity number for the object. Each distinct object in the computer's memory will have its own identity number.

Most Python objects have either data or functions or both associated with them. These are known as attributes. The name of the attribute follows the name of the object. These two are separated by a dot in between them. The two types of attributes are called either data attributes or methods. A data attribute is a value that is attached to a specific object. In contrast, a method is a function that is attached to an object. And typically a method performs some function or some operation on that object. Object type always determines the kind of operations that it supports. In other words, depending on the type of object, different methods may be available to you as a programmer.

3.2 Modules and Methods

Python modules are libraries of code that can be implemented using the `import` statements. Each module comes with several functions (or methods) associated with it. Sometimes, we don't want to use the entire module. Perhaps we just want to choose one function from that module. Let's think about a situation where I just need to be able to have the value of π available to me in my program. So, I don't need anything else from the imported library, just the value of π . To do that, I can tell Python from `math` library to import just π , as illustrated in Fig(9).

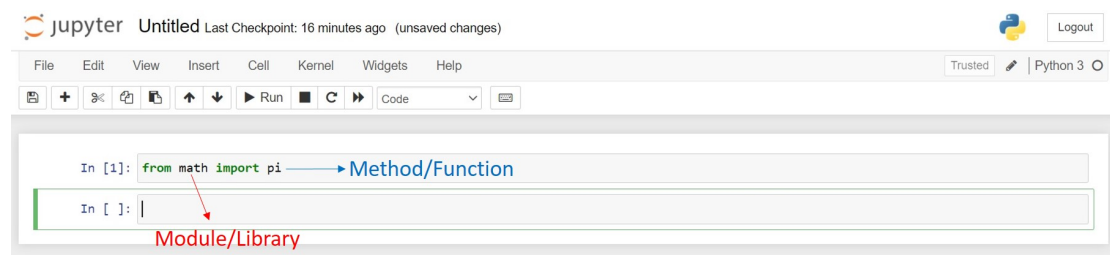


Figure 9: An Example of importing a method from a module.

It's usually helpful to know what the available methods are included in a given module. To list all the module's functions, we can use the `dir(module)` command to show the most relevant methods available.

4 Scalars, Vectors, and Matrices

Linear algebra is one of the fundamental mathematical topics that provide a solid understanding of many science and engineering systems. Scalars, vectors, and matrices are the basic elements used in linear algebra, that form the backbone of various methods, equations, and algorithms. Hence, it is crucial for scientists and engineers to understand these core ideas. In this section, we will go through the definitions of these key elements and how to define them in Python through examples.

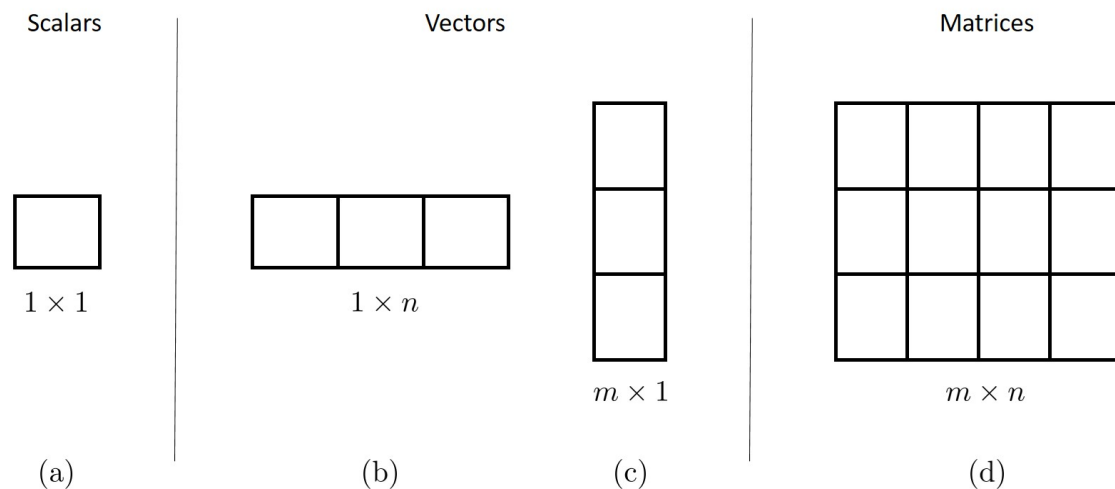


Figure 10: Simplified sketch explaining the difference between scalars, vectors, and matrices.

4.1 Scalars

A scalar is a single number. Normally, a scalar represents the magnitude of a quantity. For example, temperature, distance, speed, or mass – all these quantities have a magnitude and can be defined as a single number. Numbers are one type of object in Python. In fact, Python provides three different numeric types: integers, floating point numbers, and complex numbers. Basic arithmetic operations can be applied to all of these scalar types, e.g., addition, subtraction, division, multiplication, .. etc. Fig(11) shows how to define different types of scalar numbers, and the main operations applied to them.

```
In [32]: #Integers and float numbers
x = 2 #Define an integer
y = 2.5 #Define a float
#Basic Operations
display(x+y) #Addition
display(y-x) #Subtraction
display(x*y) #Multiplication
display(x/y) #Division
display(x//y) #Floor division
display(x**y) #Exponentiation

#Importing "cmath" for complex number operations
import cmath
# Initializing real numbers
x = 5
y = 3
# converting x and y into complex number
z = complex(x,y);
# display the real and imaginary part of complex number
display(z.real)
display(z.imag)
```

Figure 11: Defining different types of scalars and the main arithmetic operations.

4.2 Vectors

A vector is an array of numbers. The numbers are arranged in order and we can identify each individual number by its index in that ordering. In simple terms, a vector is an arrow representing a quantity that has both magnitude and direction wherein the length of the arrow represents the magnitude and the orientation tells you the direction. For example, wind has a direction and magnitude. Vectors can be structured either as a row vector, as shown in Fig(10,b), or a column vector, Fig(10,c). Row vectors have n columns with only one single row, and this can be written as $1 \times n$. Whereas, column vectors structure the data vertically where they have m rows with only one column, and this can be written as $m \times 1$. Whether to use row or column vectors depends mainly on the context and the studied problem.

In Python, there are a number of ways to create a row and column vector, as shown in Fig(12) and Fig(13). Notice, that when defining a column vector, each row is included in square brackets separated by a comma from the other rows.

```
In [1]: #Import numpy package
import numpy as np

In [12]: #Equivalent ways to create a row vector
x1 = np.array([1,2,3])
x2 = np.matrix([1,2,3])

display(x1) #show vector x1
display(x2) #show vector x2
```

Figure 12: Different ways to create a row vector.

There are other ways to generate vectors. Let's say, you want to generate a vector within a given interval. This can be easily done using the `arange()` method,

```
In [13]: #Equivalent ways to create a column vector
y1 = np.array([[1],[2],[3]]) # notice that each row is in square brackets
y2 = np.matrix([[1],[2],[3]])

display(y1) #show vector y2
display(y2) #show vector y3
```

Figure 13: Different ways to create a column vector.

as shown in Fig(14). Very important here to mention is that vectors in Python are zero-indexed, meaning that Python starts indexing (or counting) rows' and columns' elements from 0, i.e., 0th element, 1st element, 2nd element, ...onwards. Therefore, when running the code in Fig(14), the last displayed element of the vector will be before the one specified in the code.

```
In [14]: #Create a vector within a range [a,b], where a is the start of the range, and b is the end+1.
z1 = np.arange(0,10) #0 is the start, and 10 is the end+1
z2 = np.arange(0,10,1) #0 is the start, 10 is the end+1, 1 is the step taken between each value.
z3 = np.arange(0,9,2) #0 is the start, 9 is the end+1, 2 is the step taken between each value.

display(z1) #show vector z1
display(z2) #show vector z2
display(z3) #show vector z3
```

Figure 14: Define a vector within a given range.

4.3 Matrices

A matrix is a 2D array. In other words, a matrix is a collection of m rows (its height) and n columns (its width) of data, as shown in Fig(10,d). The matrix dimension is usually written as $m \times n$, so each element is identified by two indices instead of just one. You can notice that vectors can be described as special cases of matrices. For example, a row vector is a matrix with only one row. Similarly, a column vector is a matrix with only one single column.

To generate a matrix in Python, the `array` method from **numpy** library is used, as shown in Fig(15).

```
In [17]: #Create a 3x3 matrix: 3 rows and 3 columns
A = np.array([[1,2,3],[4,5,6],[7,8,9]])

display(A) #show matrix A
```

Figure 15: Create a matrix.

Once the matrix is created, there are various ways to manipulate it and pull out information from it that could be useful for the computation that you need to do. This can be done by accessing its main components, i.e., rows, columns, and

elements, as illustrated in Fig(16). As mentioned previously, always remember that Python is a zero-indexed language, meaning that it starts indexing (or counting) rows and columns of a matrix from 0.

```
In [19]: #Accessing and manipulating matrix rows, columns and elements
a1 = A[1,2] #access the element in the 2nd row and 3rd column
a2 = A[1,:] #access 2nd row with all its elements/columns (the colon :) means access everything in that row)
a3 = A[:,2] #access 3rd column with all its elements/rows (the colon :) means access everything in that column)

display(a1) #show a1
display(a2) #show a2
display(a3) #show a3
```

Figure 16: Accessing matrix elements.

When dealing with matrices, it is essential to first describe their shape, size, and dimension to understand their main features, as Fig(17) shows.

```
In [29]: #Describing matrices
display(A.shape) #show the number of rows and columns

display(A.size) #show the number of elements (rows*columns)

display(A.ndim) #show the number of dimensions
```

Figure 17: Different ways to describe a matrix.

Matrices have basic operations that are necessary to apply in many problems, e.g., addition, subtraction, and transpose. Transpose is a common operation in linear algebra where the column and row indices of each element are swapped. These operations have straightforward commands in Python to implement, as shown in Fig(18).

```
In [25]: #Matrix operations
A = np.array([[1,2,3],[4,5,6],[7,8,9]]) #Define matrix A
B = np.array([[10,11,12],[13,14,15],[16,17,28]]) #Define matrix B

#Addition
Add1 = A+B
Add2 = np.add(A,B)

display(Add1)
display(Add2)

#Subtraction
Sub1 = A-B
Sub2 = np.subtract(A,B)

display(Sub1)
display(Sub2)

#Transpose
AT = A.T #Take the transpose of matrix A
BT = B.T #Take the transpose of matrix B

display(AT)
display(BT)
```

Figure 18: Matrix Operations.

4.4 Problem: Work

Work (W (J)) is the transfer of energy by a force (\vec{F} (N)) acting on an object as it is moved from one place to another- that is, undergoes a displacement (\vec{s} (m)). You do more work if the force or displacement is greater. Mathematically, we define the work done by a constant force as the dot product between the force and displacement vectors:

$$W = \vec{F} \cdot \vec{s} \quad (1)$$

An essential point to mention here is that work is a scalar quantity, even though it's calculated from two vector quantities. Notice that this equation has a form of the scalar product of two vectors:

$$\vec{A} \cdot \vec{B} = |\vec{A}||\vec{B}| \cos \phi \quad \text{and} \quad \vec{A} \cdot \vec{B} = A_x B_x + A_y B_y + A_z B_z$$

where: $\vec{A} = A_x \hat{i} + A_y \hat{j} + A_z \hat{k}$ and $\vec{B} = B_x \hat{i} + B_y \hat{j} + B_z \hat{k}$. Therefore, it can be re-written as:

$$\vec{F} \cdot \vec{s} = |\vec{F}||\vec{s}| \cos \phi \quad \text{and} \quad \vec{F} \cdot \vec{s} = F_x s_x + F_y s_y + F_z s_z \quad (2)$$

where $|\vec{F}|$ and $|\vec{s}|$ are the magnitudes of the force and displacement vectors, respectively, and Φ is the angle between the force vector and displacement direction.

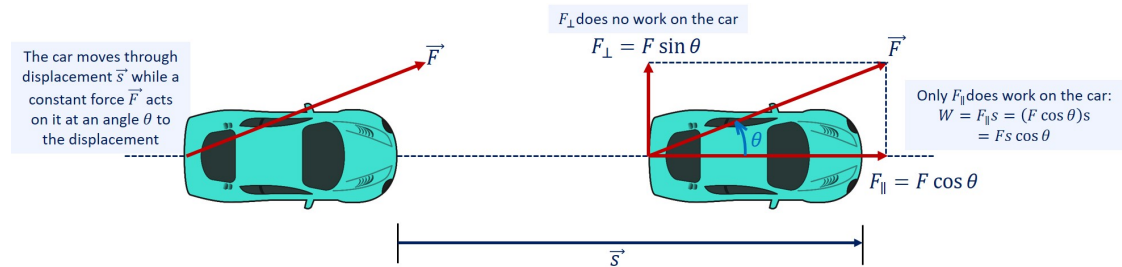


Figure 19: The work done by a constant force acting at an angle to the displacement [6].

As an illustration of this equation [6], think of a person exerting a force on a stalled car of magnitude 210 N in Fig(19), as he pushes it in a distance of 18 m. The car has also a flat tire, so to make the car track straight, the person needs to push at an angle of 30° to the direction of the motion. To calculate the work here, and since we have information on the magnitudes of the force and displacement with the angle between them, we need to use Eq(2) directly, as follows:

$$\begin{aligned}
 W &= \vec{\mathbf{F}} \cdot \vec{\mathbf{s}} = |\vec{\mathbf{F}}| |\vec{\mathbf{s}}| \cos \phi \\
 &= (210\text{N})(18\text{m}) \cos(30^\circ) \\
 &= 3.3 \times 10^3 \text{J}
 \end{aligned}$$

In Python, this can be calculated as shown in Fig(20). Notice the use of the **numpy** library to call both methods: π and \cos .

```
In [52]: #import numpy library
import numpy as np

#Define input values
F = 210 # force magnitude in N
s = 18 # displacement magnitude in m
#phi = (np.pi)/6 #angle in radians between the force and the displacement vectors (30 deg)
phi = 30*(np.pi/180) #or transform the angle from degrees to radians

#Calculate work
W = F*s*(np.cos(phi)) #in J
```

Figure 20: Calculating work using the magnitudes of force $|\vec{\mathbf{F}}|$ and displacement $|\vec{\mathbf{s}}|$ with the angle between their vectors ϕ .

This person needs to push a second stalled car with a steady force $\vec{\mathbf{F}} = (160\text{N})\hat{i} - (40\text{N})\hat{j}$. The displacement of the car is $\vec{\mathbf{s}} = (14\text{m})\hat{i} + (11\text{m})\hat{j}$. To compute the work here, we notice that we have been provided with the components of the force and displacement vectors. Using Eq(2), we can write the following:

$$\begin{aligned}
 W &= \vec{\mathbf{F}} \cdot \vec{\mathbf{s}} = F_x s_x + F_y s_y \\
 &= (160\text{N})(14\text{m}) + (-40\text{mN})(11\text{m}) \\
 &= 1.8 \times 10^3 \text{J}
 \end{aligned}$$

In Python, this can be computed as shown in Fig(21). Notice how arrays (row vectors) are defined using **numpy** library and how their elements are called.

```
In [55]: #import numpy library
import numpy as np

#Define the vectors
F = np.array([160, -40]) #Force vector
s = np.array([14, 11]) #displacement vector

#Calculate the sum of the products of their respective components
W = F[0]*s[0] + F[1]*s[1] #in J
```

Figure 21: Calculating work using the components of the force $\vec{\mathbf{F}}$ and displacement $\vec{\mathbf{s}}$ vectors.

Now, suppose we have the following two vectors:

$$\vec{\mathbf{F}} = 2\hat{i} + 3\hat{j} + 1\hat{k} \quad \text{and} \quad \vec{\mathbf{s}} = -4\hat{i} + 2\hat{j} - 1\hat{k}$$

and we need to find the angle between both of them. To do that, we solve Eq(2) for $\cos \phi$:

$$\begin{aligned} \cos \phi &= \frac{\vec{\mathbf{F}} \cdot \vec{\mathbf{s}}}{|\vec{\mathbf{F}}||\vec{\mathbf{s}}|} = \frac{F_x s_x + F_y s_y + F_z s_z}{\sqrt{F_x^2 + F_y^2 + F_z^2} \sqrt{s_x^2 + s_y^2 + s_z^2}} \\ &= \frac{(2)(-4) + (3)(2) + (1)(-1)}{\sqrt{(2)^2 + (3)^2 + (1)^2} \sqrt{(-4)^2 + (2)^2 + (-1)^2}} \\ &= \frac{-3}{\sqrt{14}\sqrt{21}} \\ &= -0.175 \end{aligned}$$

Taking the inverse of the cosine gives $\phi = 100^\circ$. Notice here that the computed work is negative and the angle is between 90° and 180° . What does that mean? Fig(22) shows the code to compute this using Python.

```
In [58]: #import numpy
import numpy as np

#Define the vectors
F = np.array([2,3,1])
s = np.array([-4,2,-1])

#Calculate the sum of the products of their respective components
FTimes = F[0]*s[0] + F[1]*s[1] + F[2]*s[2]

#Calculate their magnitudes
#Vector F
Fsquare = np.square(F) #the square of each component of vector F
Fsum = np.sum(Fsquare) #the sum of the squared components of F
Fmag = np.sqrt(Fsum) #square root of the sum
#Vector s
ssquare = np.square(s) #the square of each component of vector s
ssum = np.sum(ssquare) #the sum of the squared components of s
smag = np.sqrt(ssum) #square root of the sum

#Calculate the cosine of the angle
CosinePhi = FTimes / (Fmag * smag)

#Calculate the angle (inverse of the cosine)
Phi_rad = np.arccos(CosinePhi) #in radians
Phi_deg = Phi_rad*(180/np.pi) #in degrees

#We notice here that the work is negative and the angle is between 90 and 180. This means ...
```

Figure 22: Calculating the scalar product of two vectors: $\vec{\mathbf{A}}$ and $\vec{\mathbf{B}}$.

4.5 Problem: Rocket Propulsion

Momentum conservation is particularly important and useful for analysing a system in which the masses of parts of the system change with time. In such cases

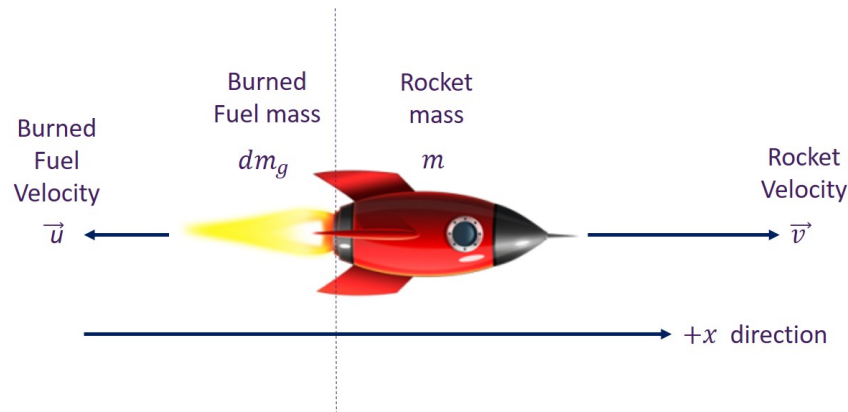


Figure 23: The rocket accelerates to the right due to the expulsion of some of its fuel mass to the left. Conservation of momentum enables us to determine the resulting change in the rocket's velocity and acceleration.

we cannot use Newton's second law $\sum \vec{F} = m \vec{a}$ directly because m changes with time. Rocket propulsion is an important example of this situation.

In this problem, we will analyse the motion of a rocket, which changes its velocity (and hence its momentum) by ejecting burned fuel gases, thus causing it to accelerate in the opposite direction of the velocity of the ejected fuel, as shown in Fig(23).

At some moment in time, the rocket has a velocity v and mass m ; this mass is a combination of the mass of the empty rocket and the mass of the remaining unburned fuel it contains. The rocket accelerates and is propelled forward by burning the fuel it carries and ejecting the burned exhaust gases. The forward force on the rocket is the reaction to the backward force on the ejected material. If the burn rate of the fuel is constant, and the velocity at which the exhaust is ejected is also constant, what is the change in velocity of the rocket as a result of burning all of its fuel? If we assume that the rocket is in outer space, far from any planet where there is no gravitational force, no air resistance, or any other external forces acting on this system; as a result, momentum is conserved for this system. Thus, we can apply the conservation of momentum to answer the question.

At the start when $t = 0$, the total instantaneous rocket mass is m (i.e., m is the mass of the rocket body plus the mass of the fuel at that point in time), we define the rocket's instantaneous velocity to be v (in the $+x$ direction); this velocity is measured relative to an inertial reference system (the Earth, for example). Thus, the initial momentum of the system is:

$$P_i = mv \quad (3)$$

The rocket's engines start to burn fuel at a constant rate and eject the exhaust gases in the $-x$ direction. During an infinitesimal time interval dt , the engines eject a (positive) infinitesimal mass of gas dm_g at velocity u ; note that although the rocket velocity v is measured with respect to Earth, the exhaust gas velocity is measured with respect to the (moving) rocket. Measured with respect to the Earth, therefore, the exhaust gas has a velocity $(v - u)$. As a consequence of the ejection of the fuel gas, the rocket's mass decreases by dm_g , and its velocity increases by dv . Therefore, including both the change for the rocket and the change for the exhaust gas, the final momentum of the system is:

$$P_f = P_{\text{Rocket}} + P_{\text{Fuel}} = (m - dm_g)(v + dv) + dm_g(v - u) \quad (4)$$

Applying conservation of momentum, Eq(3) must equal to Eq(4), hence we obtain:

$$\begin{aligned} P_i &= P_f \\ mv &= (m - dm_g)(v + dv) + dm_g(v - u) \\ mv &= mv + mdv - dm_gv - dm_gdv + dm_gv - dm_gu \\ mdv &= dm_gdv + dm_gu \end{aligned}$$

Now, dm_g and dv are each very small; thus, their product dm_gdv is very, very small, much smaller than the other two terms in this expression. We neglect this term, therefore, and obtain:

$$mdv = dm_gu$$

Our next step is to remember that, since dm_g represents an increase in the mass of ejected gases, it must also represent a decrease in the mass of the rocket. This can be written as $dm_g = -dm$. Replacing this gives us:

$$mdv = -dmu \longrightarrow dv = -u \frac{dm}{m}$$

Integrating from the initial mass m_i to the final mass m of the rocket gives us the result we are after:

$$\begin{aligned} \int_{v_i}^v dv &= -u \int_{m_i}^m \frac{dm}{m} \\ v - v_i &= -u \ln \left(\frac{m}{m_i} \right) \end{aligned}$$

Or simply we can write:

$$v = v_i + u \ln \left(\frac{m_i}{m} \right) \quad (5)$$

This result is called the rocket equation. It was originally derived by the Soviet physicist Konstantin Tsiolkovsky in 1897. It gives us the change of velocity that the rocket obtains from burning a mass of fuel that decreases the total rocket mass from m_i down to m . As expected, the relationship between v and the change of mass of the rocket is nonlinear.

Let's consider a rocket that ejects burned fuel at a constant rate at a relative speed of $u = 2400\text{m/s}$. The final mass of the rocket after the ejection is $m = \frac{m_i}{4}$ of its initial mass m_i . If this rocket starts from rest, then the rocket's velocity after the ejection time can be calculated using Eq(5) as follows:

$$\begin{aligned} v &= v_i + u \ln \left(\frac{m_i}{m} \right) \\ &= 0 + 2400 \ln \left(\frac{m_i}{\frac{m_i}{4}} \right) \\ &= 2400 \ln(4) \\ &= 3327\text{m/sec} \end{aligned}$$

```
#import numpy
import numpy as np

#Define the ejected fuel speed
u = 2400 #m/sec

#Define the ratio of the intital mass to the rocket's mass after the fuel is ejected, i.e.,m0/m
MassRatio = 4;

#Define the initia rocket's speed
vi = 0;

#Calculate the rocket's speed after the fuel ejection
v = vi + u * np.log(MassRatio) #m/sec

display(v)
```

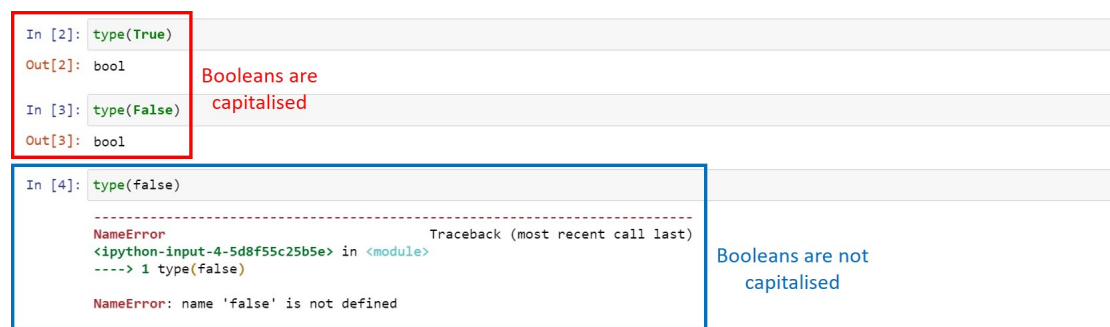
3327.1064666877373

Figure 24: Calculating the rocket's speed after fuel is being ejected.

5 Expressions and Booleans

Operators in Python are used to perform operations on variables and values. In this section, we will cover three main families of operators used in Python: comparison, logical, and identity operators, listed in Table(28).

Expression is a combination of objects and operators that computes a value. Many expressions involve what is known as the boolean data type. Objects of the boolean type have only two values: **True** or **False**. In Python, you need to capitalize these words for Python to understand them as a boolean type. Note from Fig(25) that not capitalising “False” results in an error as Python doesn’t know what this object is.



```
In [2]: type(True)
Out[2]: bool

In [3]: type(False)
Out[3]: bool

In [4]: type(false)
-----
NameError                                Traceback (most recent call last)
<ipython-input-4-5d8f55c25b5e> in <module>
----> 1 type(false)

NameError: name 'false' is not defined
```

Booleans are capitalised

Booleans are not capitalised

Figure 25: Booleans in Python.

The simplest case where a boolean data type occurs is when two objects are compared. We often need to compare objects in our programs. There are a total of six different comparison operations in Python, listed in Table(1). Although these are commonly used for numeric types, we can actually apply them to other types as well. The result of a comparison operation is always a boolean type, either **True** or **False**. An example of using these operators is shown in Fig(26).

Operations involving logic, so-called logical (or boolean) operations, take in one or more boolean objects and then they return one boolean object back to you. There are only three logical operators: “or”, “and”, and “not”.

Let’s start with “or”. “or” between **x** and **y** is going to be **True** if either **x** is **True** or **y** is **True**, or both are **True**. So for example, if we say **True or False**, then Python returns **True**. **True or True** would also be **True**. So the only time “or” would be **False**, if both the first and second object surrounding “or” are **False**.

On the other hand, “and” is only **True** if both objects are **True**. So, if we type **True and True**, the answer is going to be **True**. However, if we turned the second **True** to **False**, “and” is going to be **False**. So, in order for “and” to be **True**, both of the objects need to be **True**.

```
In [5]: x = 10
        y = 12

        # Output: x > y is False
        print('x > y is',x>y)

        # Output: x < y is True
        print('x < y is',x<y)

        # Output: x == y is False
        print('x == y is',x==y)

        # Output: x != y is True
        print('x != y is',x!=y)

        # Output: x >= y is False
        print('x >= y is',x>=y)

        # Output: x <= y is True
        print('x <= y is',x<=y)
```

Figure 26: Example of Comparison Operators in Python.

Finally, we have the “not” operator, which simply negates the value of the object. So if we say `not True`, Python gives us `False`. And if we say `not False`, Python returns to us `True`. Fig(27) shows an example illustrating this.

```
In [6]: x = True
        y = False

        # Output: x and y is False
        print('x and y is',x and y)

        # Output: x or y is True
        print('x or y is',x or y)

        # Output: not x is False
        print('not x is',not x)
```

Figure 27: Example of Logical Operators in Python.

The third family of operators that we need to cover is identity operators. Identity operators in Python are “`is`” and “`is not`”. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal do not imply that they are identical. Notice that this is different than asking if the contents of two objects are the same.

The example in Fig(28) shows that `x1` and `y1` are integers of the same values, so they are equal as well as identical. The same is the case with the strings `x2` and `y2`. But `x3` and `y3` are lists. They are equal but not identical. It is because the interpreter locates them separately in memory although they are equal. So, although these two lists have the same content, they are two individual and distinct objects. That’s why this comparison returns `False`.

```

In [7]: x1 = 5
        y1 = 5
        x2 = 'Hello'
        y2 = 'Hello'
        x3 = [1,2,3]
        y3 = [1,2,3]

        # Output: False
        print(x1 is not y1)

        # Output: True
        print(x2 is y2)

        # Output: False
        print(x3 is y3)

```

Figure 28: Example of Identity Operators in Python.

Operator	Name	Example
Comparison Operators		
==	Equal	$x == y$
!=	Not equal	$x \neq y$
>	Greater than	$x > y$
<	Less than	$x < y$
>=	Greater than or equal to	$x \geq y$
<=	Less than or equal to	$x \leq y$
Logical Operators		
and	Returns True if both statements are true	$x < 5$ and $x < 10$
or	Returns True if one of the statements is true	$x < 5$ and $x < 4$
not	Reverse the result, returns False if the result is true	not ($x < 5$ and $x < 10$)
Identity Operators		
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Table 1: Python Operators