**Introduction**

In India, the supply of LPG through pipelines is not possible due to shortage of LPG. As technology being improved many gas agencies or distributors have implemented IVRS these days although due to daily busy schedules, customer finds very difficult to book new cylinder. So, our proposal is to completely automate the process of refill booking without human intervention that accordingly will help consumer against foul play. Based on the usages of lpg cylinder in domentic and number of members stays in the house, data has been recorded based on the everytime the customer refills the lpg cylinder. Our aim is to predict the next refilling date of each family.

**PROBLEM STATEMENT**

* The existing system is basically a manual system, where if the user needs cylinders then he needs to contact the seller and book them.
* While cooking cylinder gets over.
* Many family use single cylinder so for them its very difficult when cylinder gets over.

**AIM AND OBJECTIVE OF PROJECT**

The main objective of the proposed work is to predict the next LPG refilling date based on the previous uses.

**Task carried out in the internship**

**Learning Python**

# Numbers and more in Python!

In this lecture, we will learn about numbers in Python and how to use them.

We'll learn about the following topics:

1.) Types of Numbers in Python

2.) Basic Arithmetic

3.) Differences between classic division and floor division

4.) Object Assignment in Python

## Types of numbers

Python has various "types" of numbers (numeric literals). We'll mainly focus on integers and floating point numbers.

Integers are just whole numbers, positive or negative. For example: 2 and -2 are examples of integers.

Floating point numbers in Python are notable because they have a decimal point in them, or use an exponential (e) to define the number. For example 2.0 and -2.1 are examples of floating point numbers. 4E2 (4 times 10 to the power of 2) is also an example of a floating point number in Python.

Throughout this course we will be mainly working with integers or simple float number types.

Here is a table of the two main types we will spend most of our time working with some examples:

|  |  |
| --- | --- |
| **Examples** | **Number "Type"** |
| 1,2,-5,1000 | Integers |
| 1.2,-0.5,2e2,3E2 | Floating-point numbers |

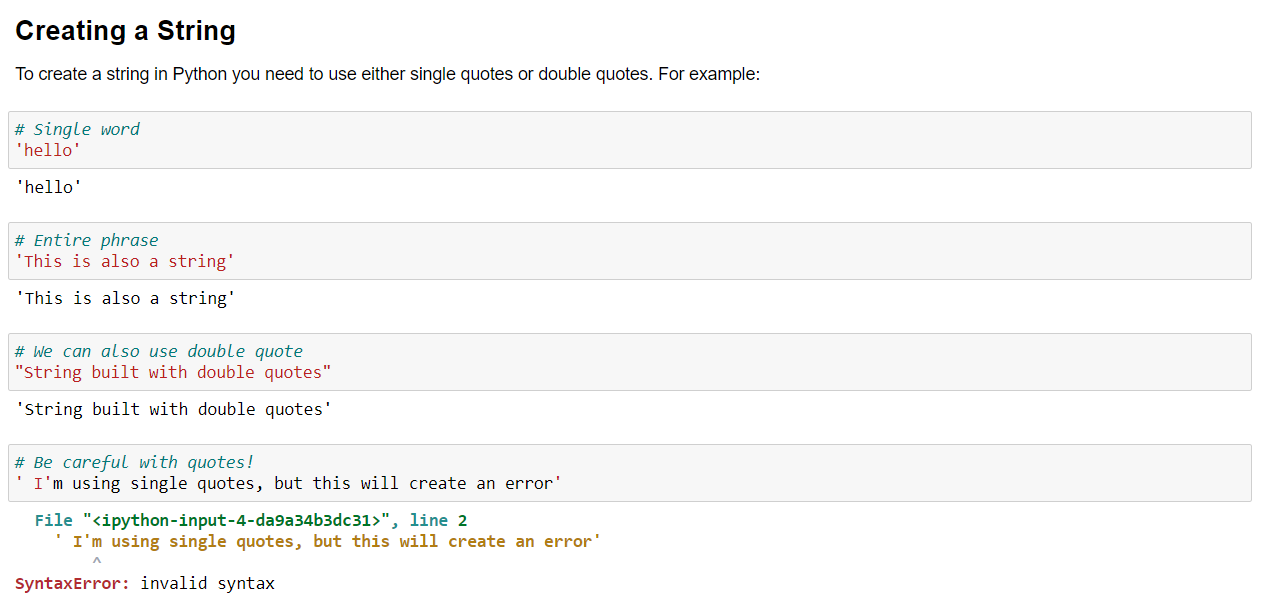
Now let's start with some basic arithmetic.

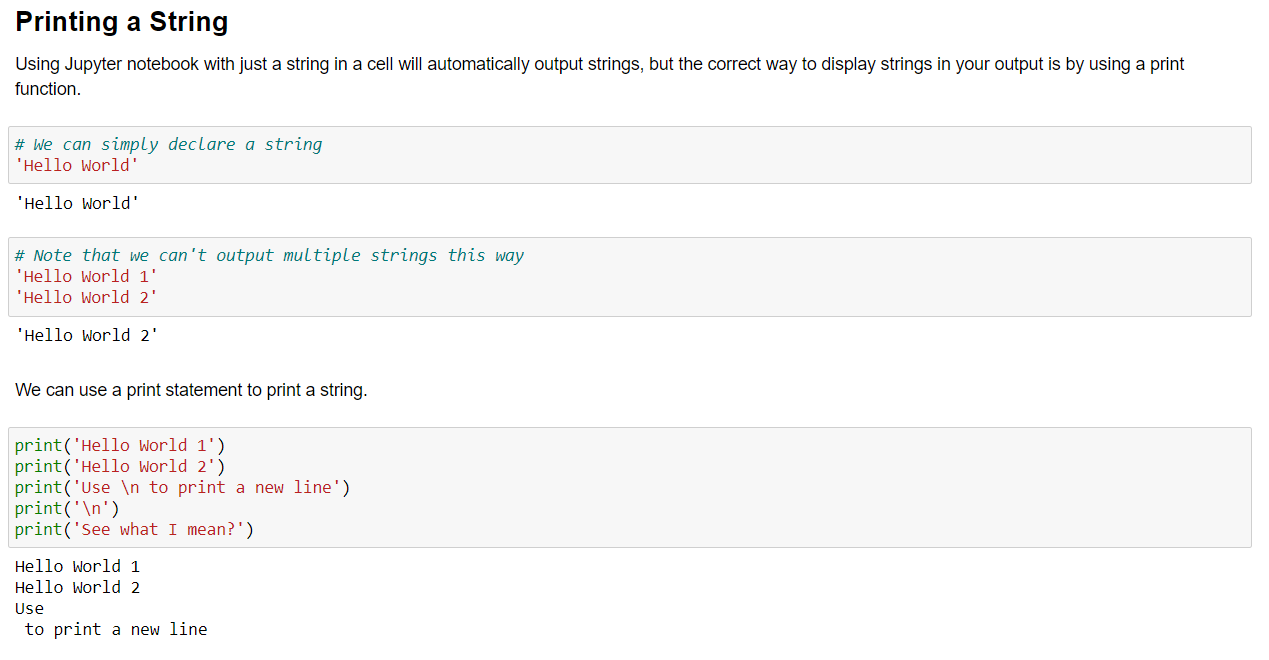
### Basic Arithmetic

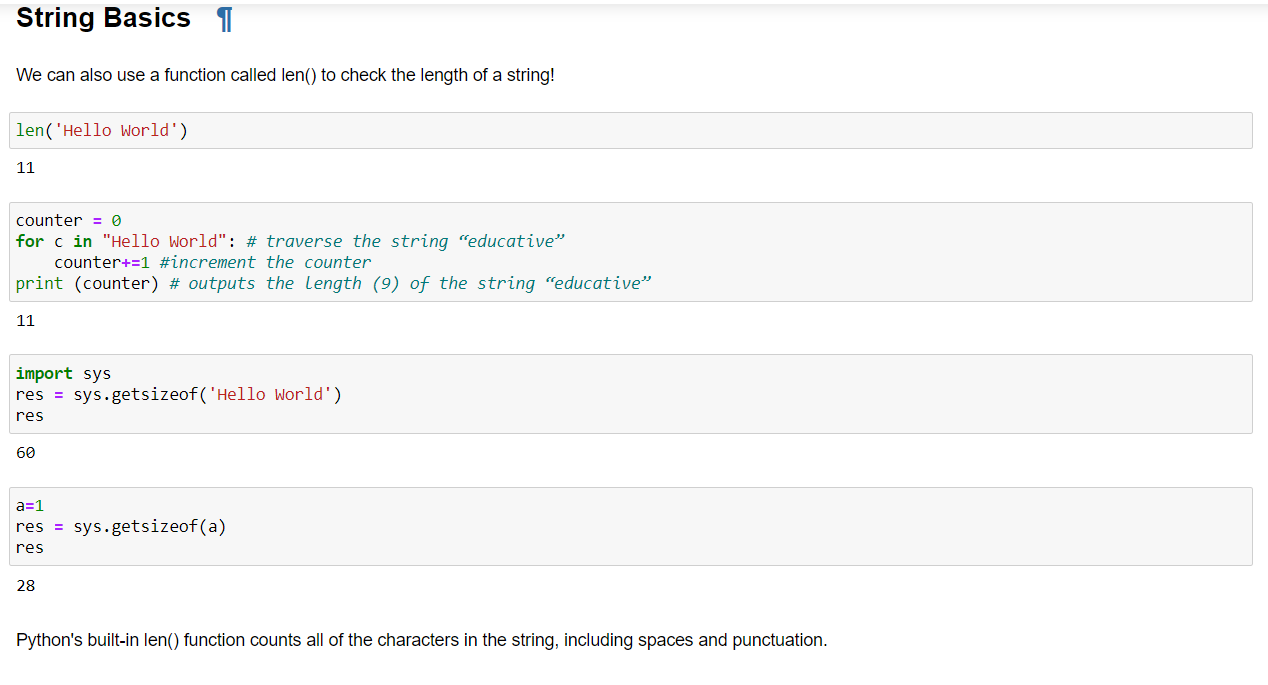


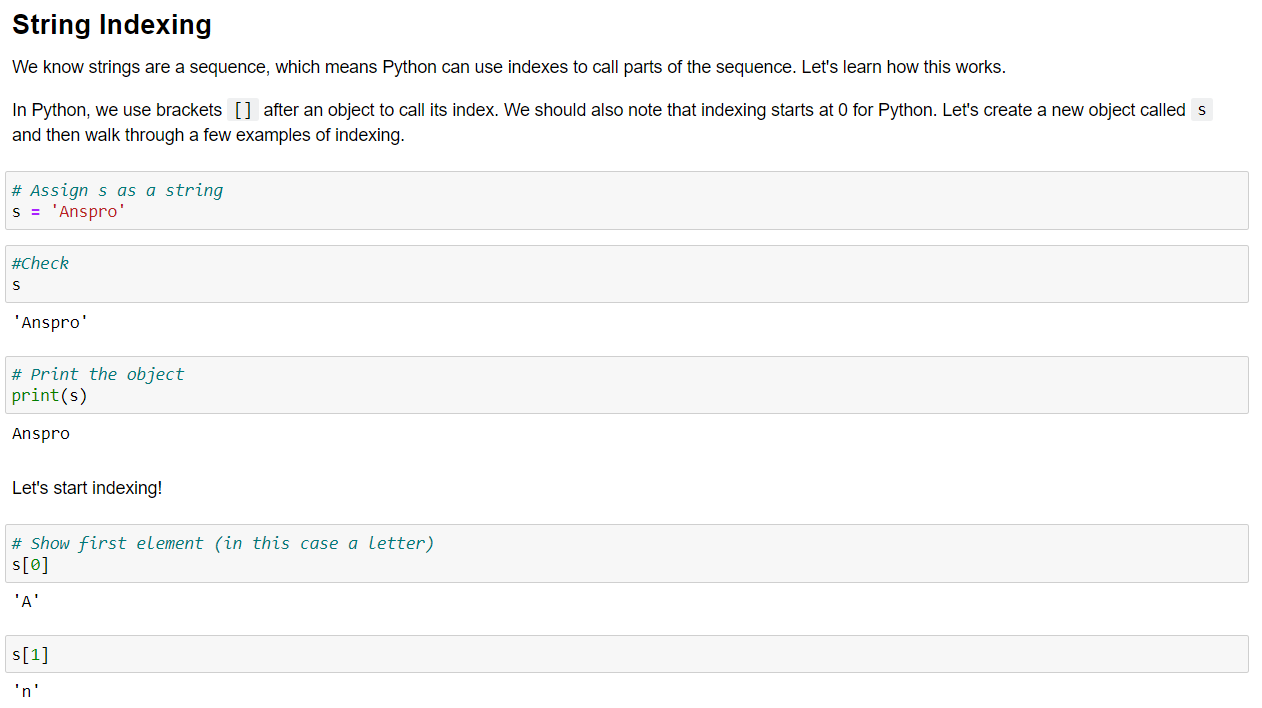
# Strings

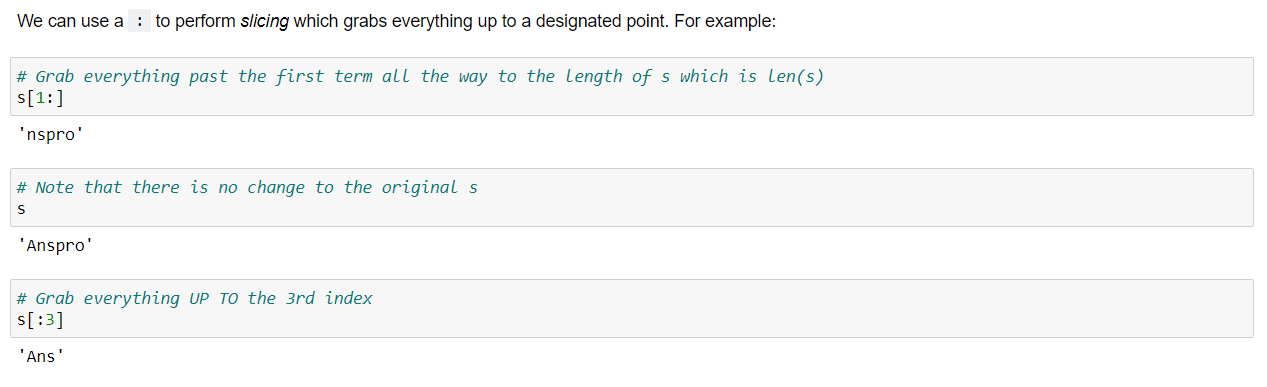
Strings are used in Python to record text information, such as names. Strings in Python are actually a sequence, which basically means Python keeps track of every element in the string as a sequence. For example, Python understands the string "hello' to be a sequence of letters in a specific order. This means we will be able to use indexing to grab particular letters (like the first letter, or the last letter).









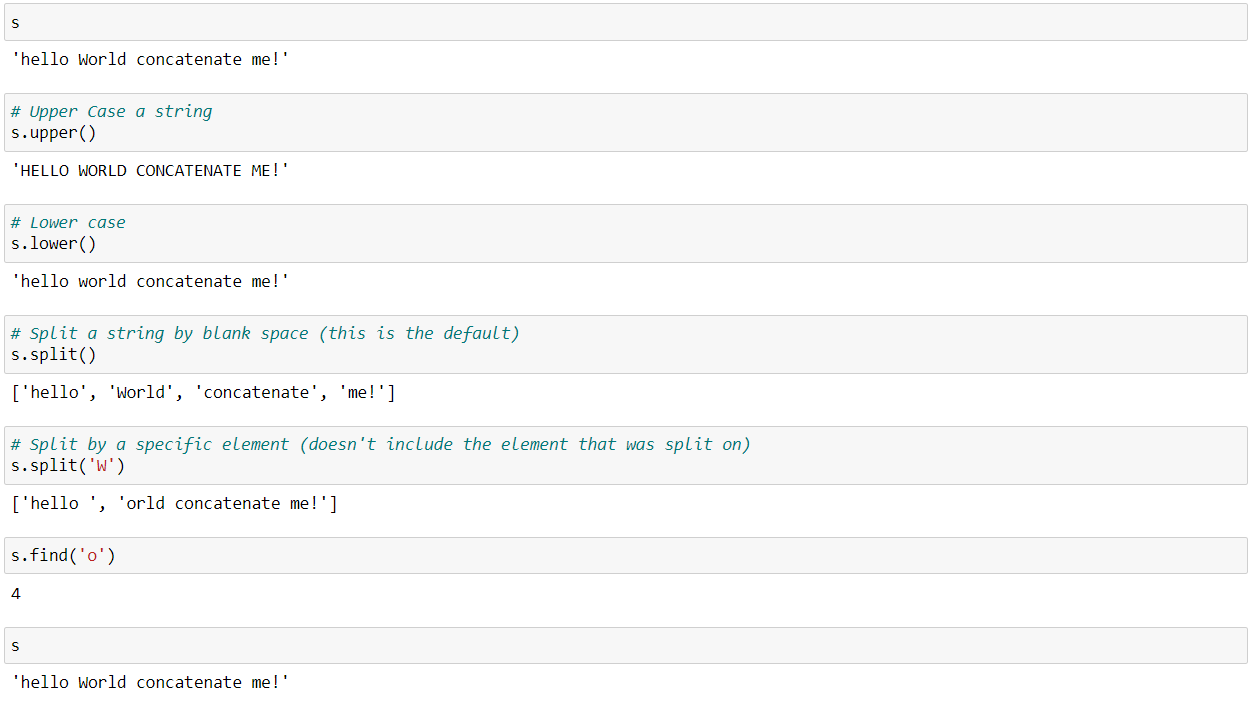


## Basic Built-in String methods

Objects in Python usually have built-in methods. These methods are functions inside the object (we will learn about these in much more depth later) that can perform actions or commands on the object itself.

We call methods with a period and then the method name. Methods are in the form:

object.method(parameters)

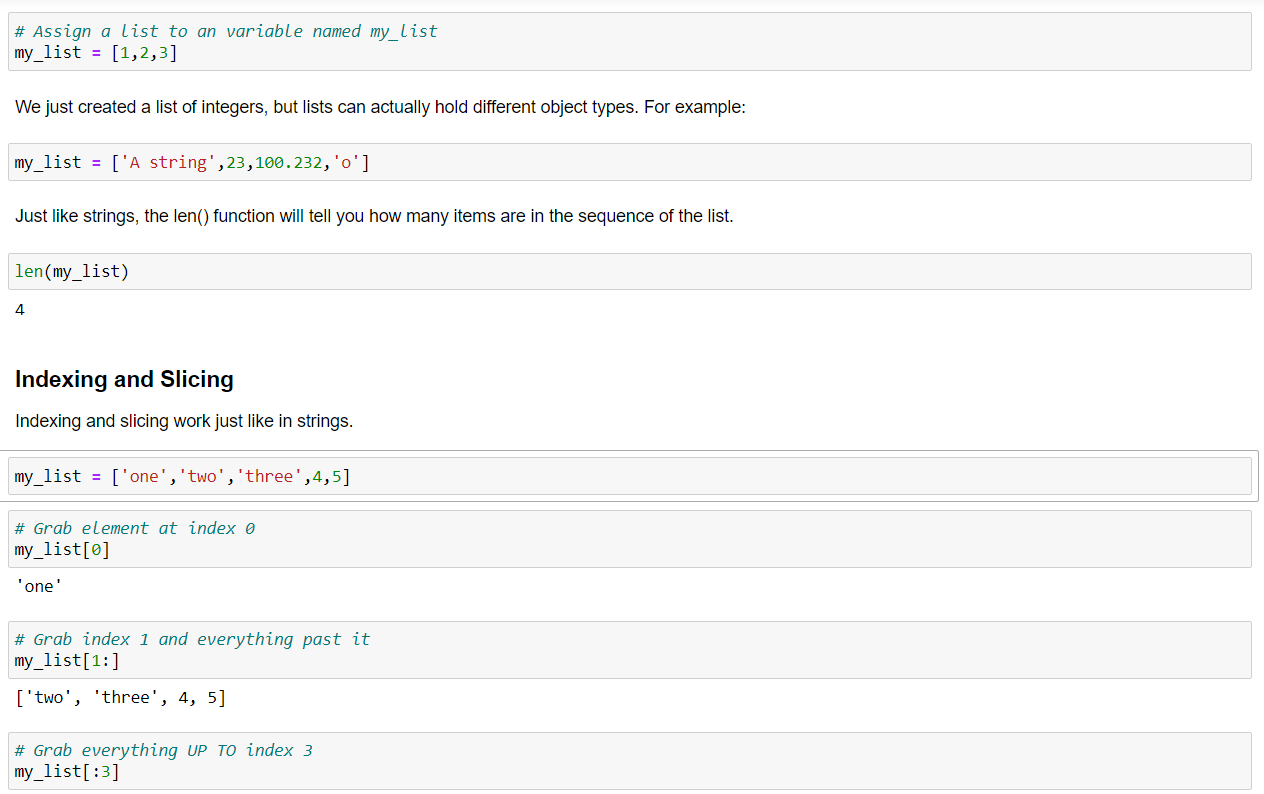






# Lists

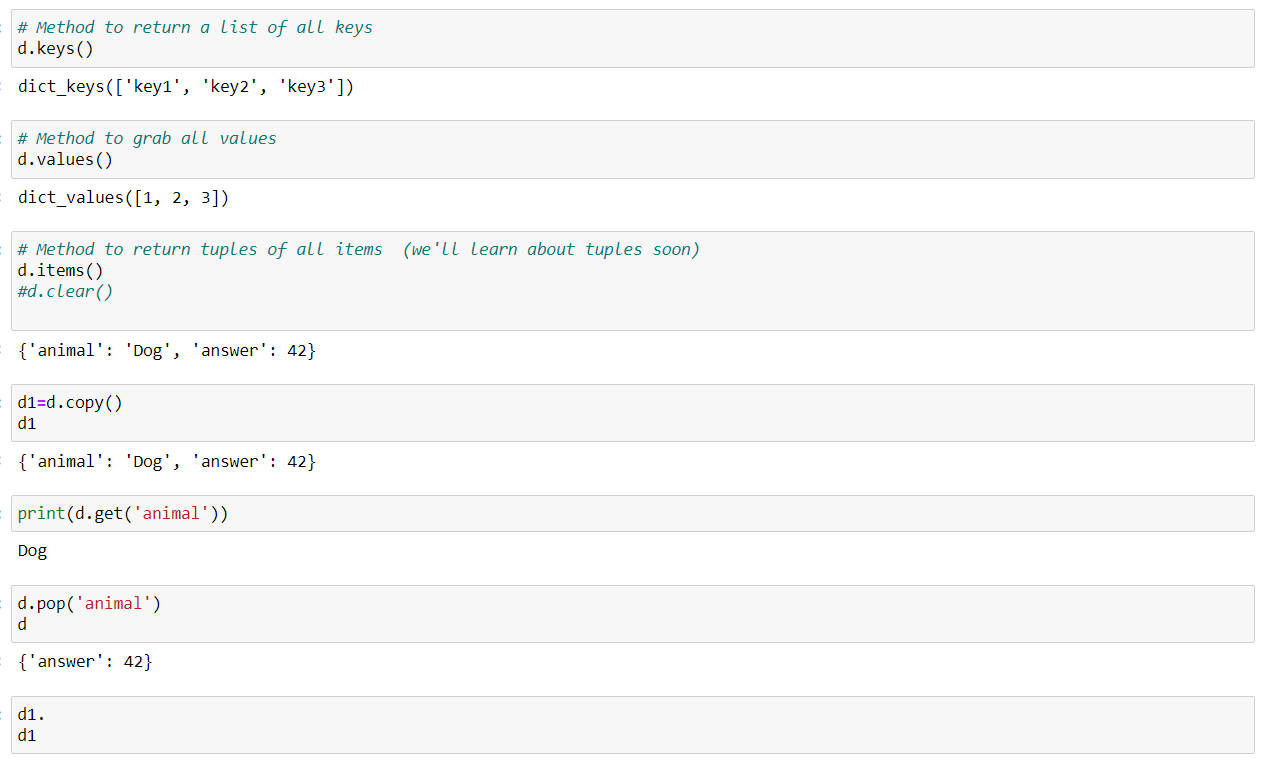
Earlier when discussing strings we introduced the concept of a sequence in Python. Lists can be thought of the most general version of a sequence in Python. Unlike strings, they are mutable, meaning the elements inside a list can be changed! A list is a collection which is ordered and changeable. In Python lists are written with square brackets.



## List Methods

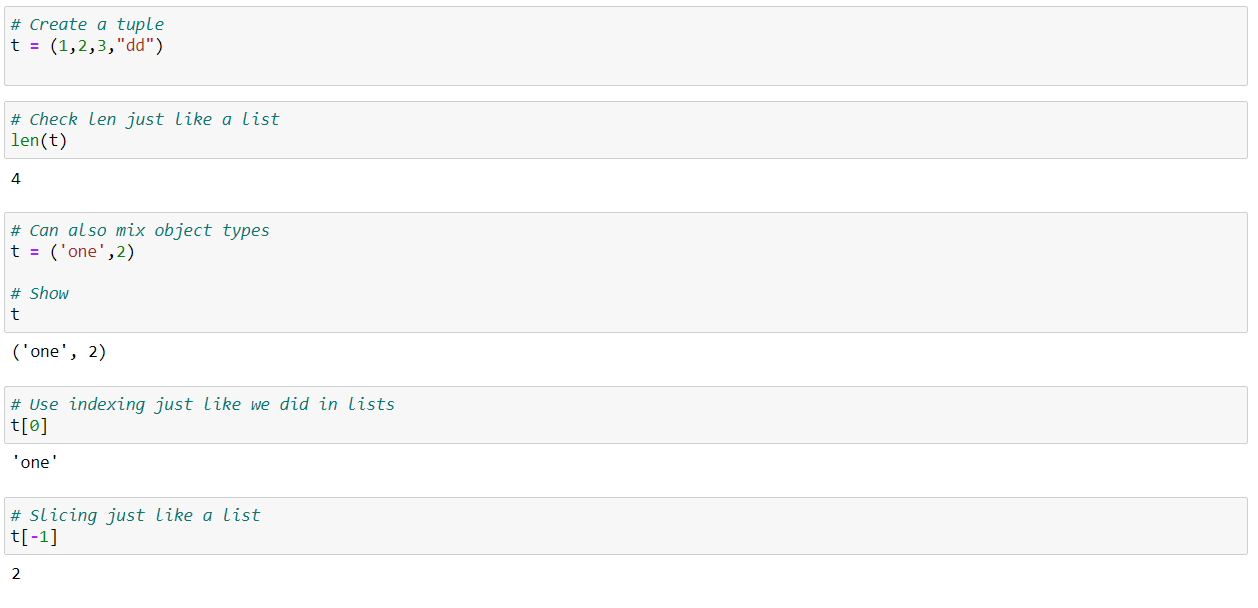
# Dictionaries

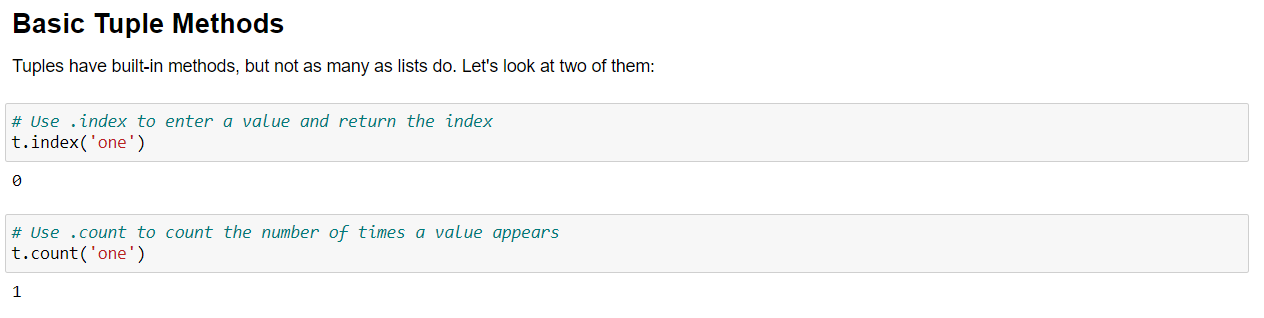
We've been learning about sequences in Python but now we're going to switch gears and learn about mappings in Python. If you're familiar with other languages you can think of these Dictionaries as hash tables.

# Tuples

In Python tuples are very similar to lists, however, unlike lists they are immutable meaning they can not be changed. You would use tuples to present things that shouldn't be changed, such as days of the week, or dates on a calendar.









**Comparison Operators**

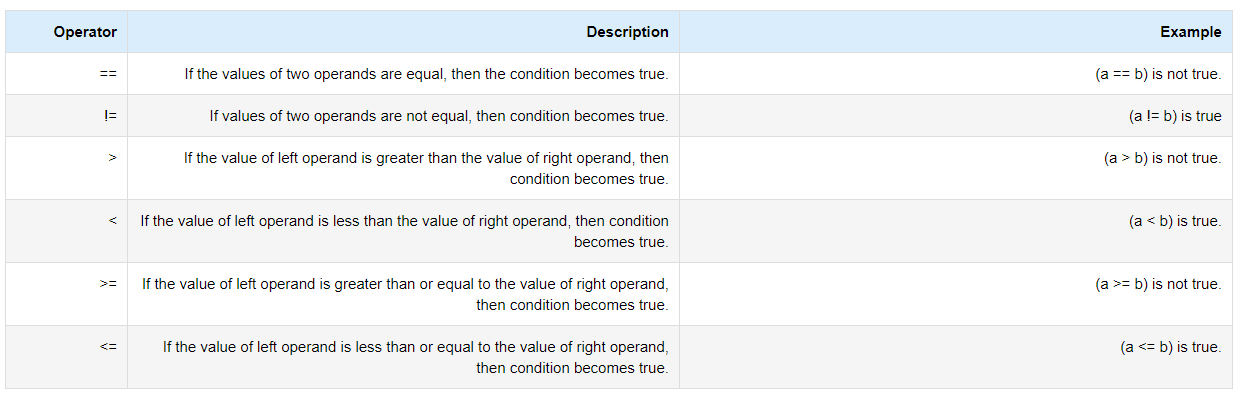
In this lecture we will be learning about Comparison Operators in Python. These operators will allow us to compare variables and output a Boolean value (True or False).

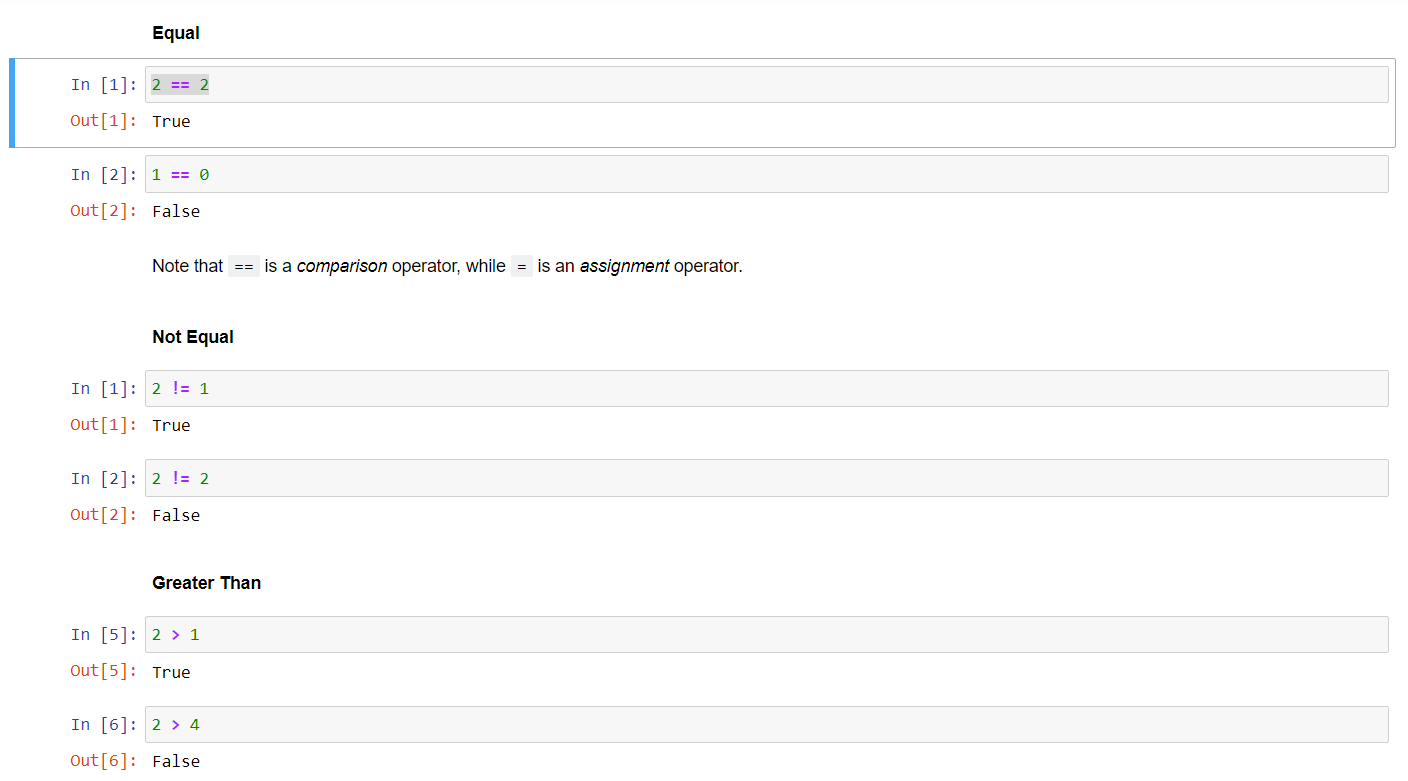
If you have any sort of background in Math, these operators should be very straight forward.

First we'll present a table of the comparison operators and then work through some examples:

**Table of Comparison Operators**

In the table below, a=3 and b=4.



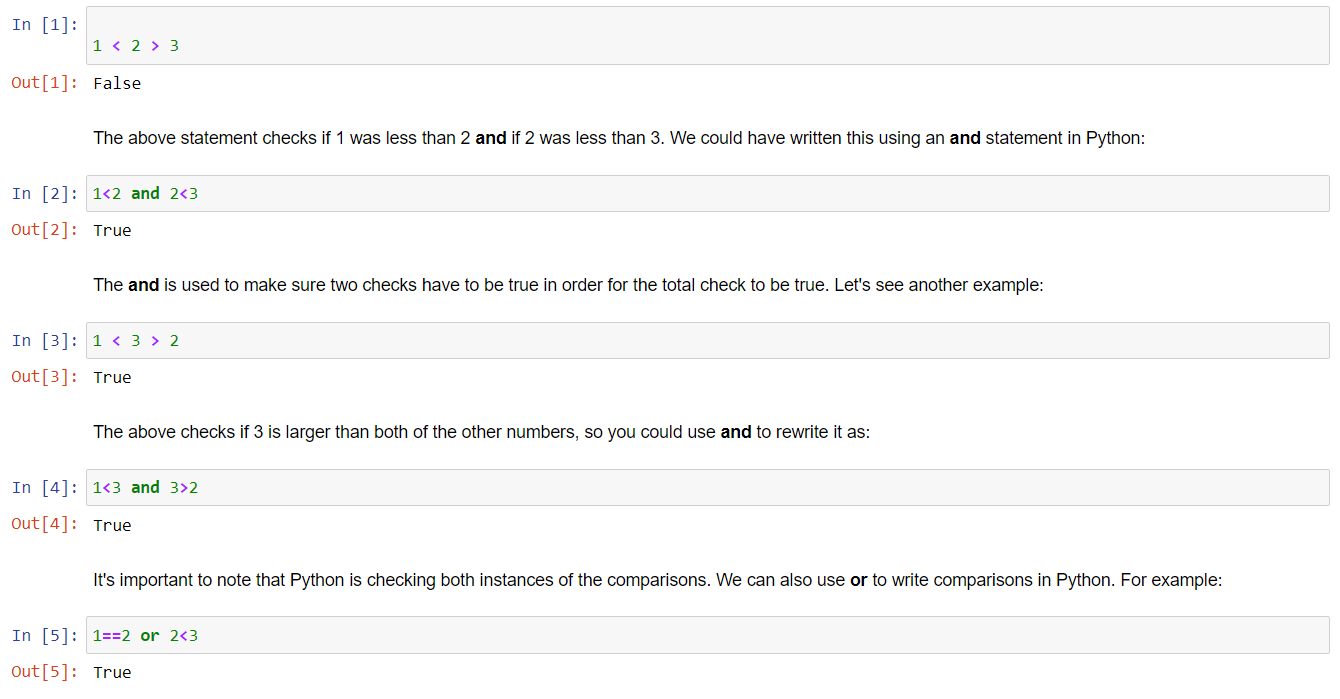


# Chained Comparison Operators

An interesting feature of Python is the ability to chain multiple comparisons to perform a more complex test. You can use these chained comparisons as shorthand for larger Boolean Expressions.

In this lecture we will learn how to chain comparison operators and we will also introduce two other important statements in Python: **and** and **or**.

Let's look at a few examples of using chains:



**if, elif, else Statements**

if Statements in Python allows us to tell the computer to perform alternative actions based on a certain set of results.

Verbally, we can imagine we are telling the computer:

"Hey if this case happens, perform some action"

We can then expand the idea further with elif and else statements, which allow us to tell the computer:

"Hey if this case happens, perform some action. Else, if another case happens, perform some other action. Else, if *none* of the above cases happened, perform this action."

Let's go ahead and look at the syntax format for if statements to get a better idea of this:

if case1:

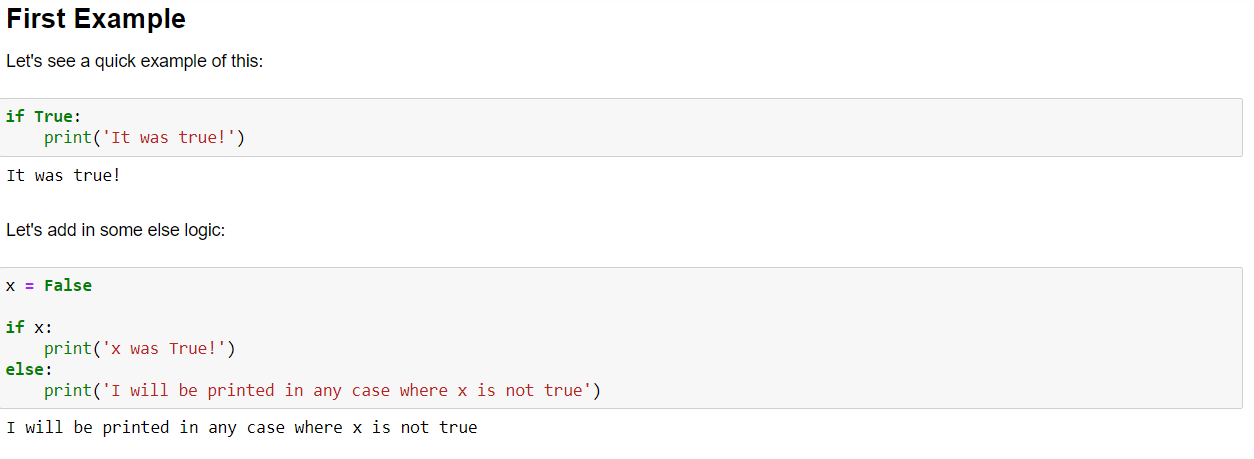
perform action1

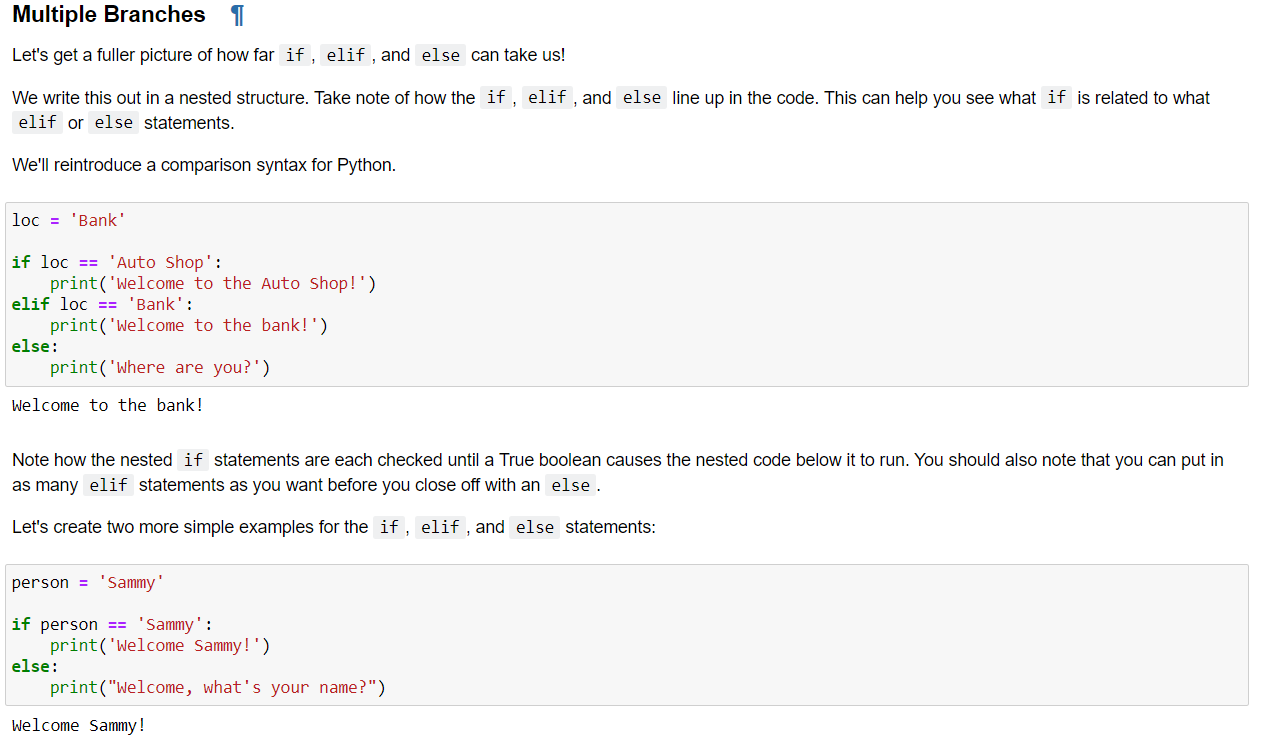
elif case2:

perform action2

else:

perform action3





# for Loops

A for loop acts as an iterator in Python; it goes through items that are in a sequence or any other iterable item. Objects that we've learned about that we can iterate over include strings, lists, tuples, and even built-in iterables for dictionaries, such as keys or values.

We've already seen the for statement a little bit in past lectures but now let's formalize our understanding.

Here's the general format for a for loop in Python:

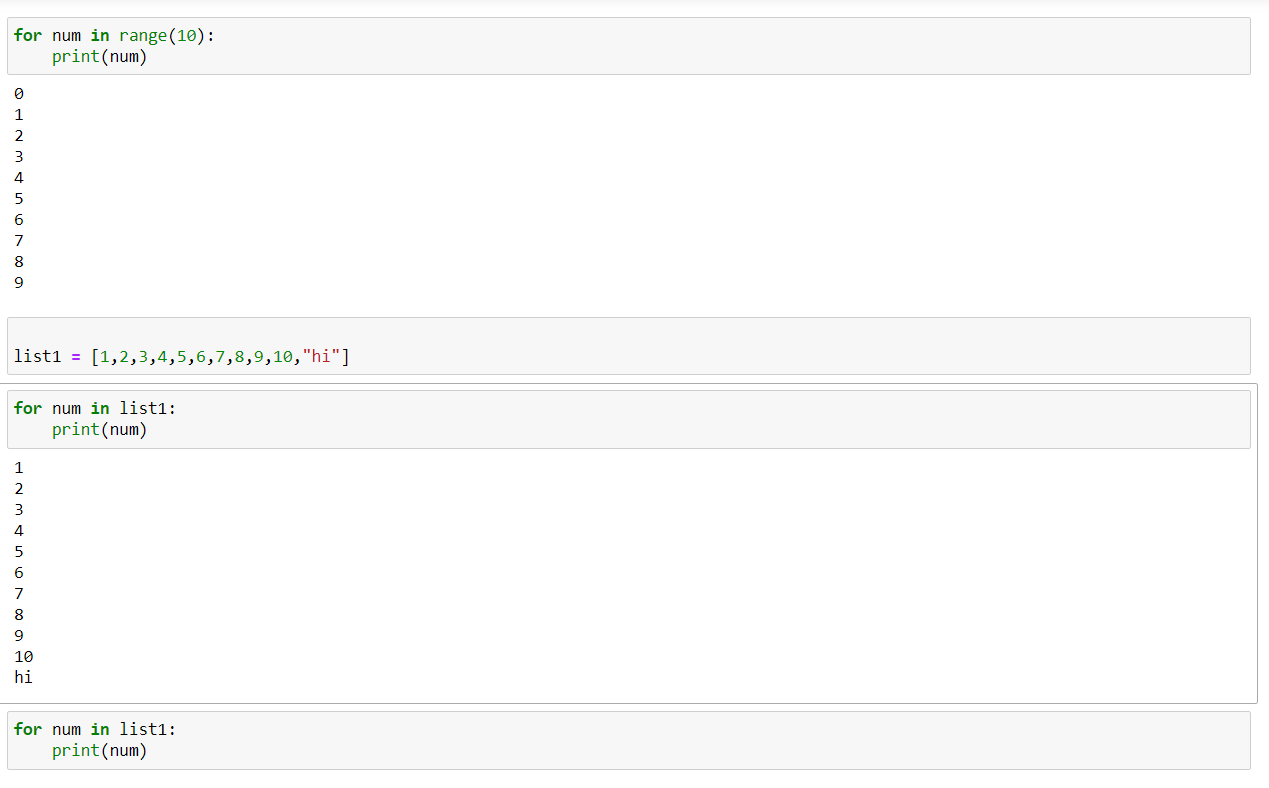
for item in object:

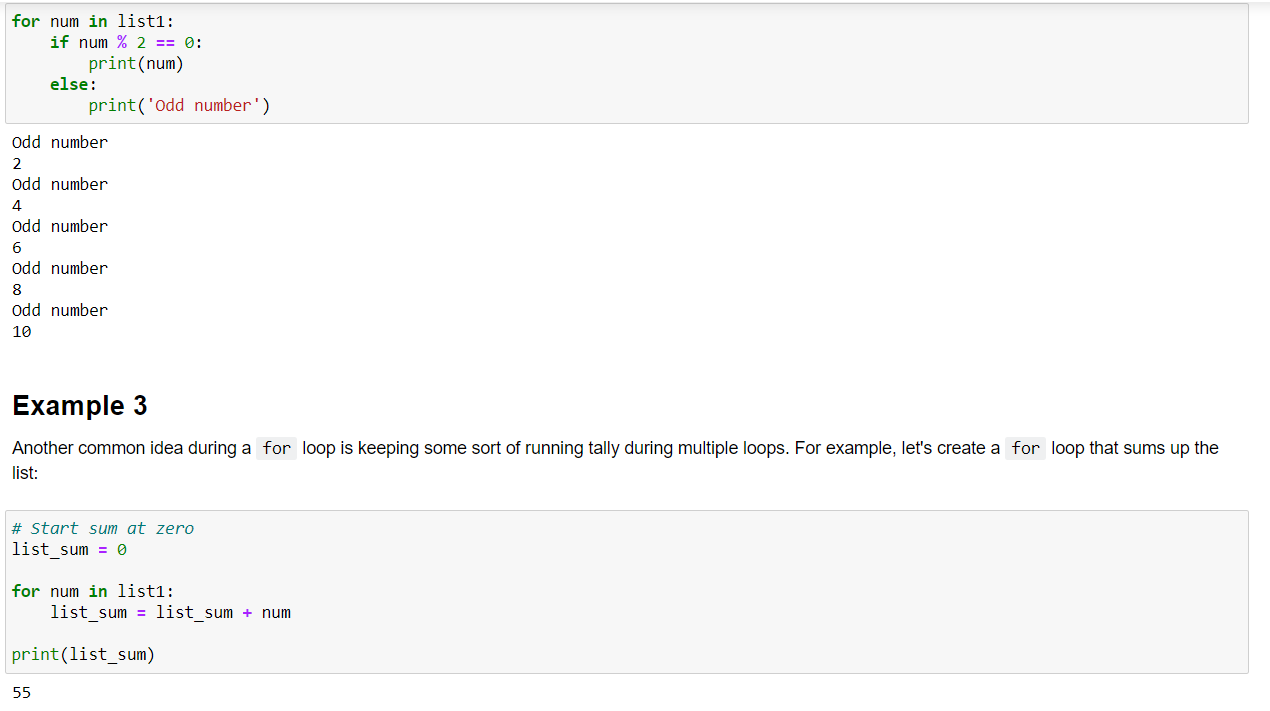
statements to do stuff

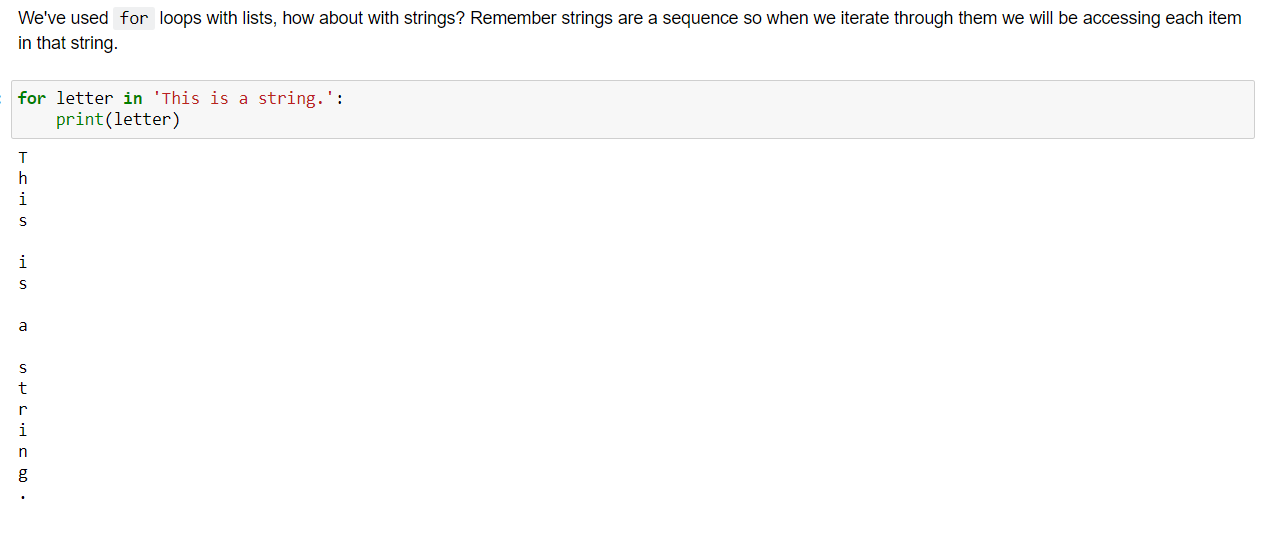
The variable name used for the item is completely up to the coder, so use your best judgment for choosing a name that makes sense and you will be able to understand when revisiting your code. This item name can then be referenced inside your loop, for example if you wanted to use if statements to perform checks.

Let's go ahead and work through several example of for loops using a variety of data object types. We'll start simple and build more complexity later on.

Iterating through a list







# while Loops

The while statement in Python is one of most general ways to perform iteration. A while statement will repeatedly execute a single statement or group of statements as long as the condition is true. The reason it is called a 'loop' is because the code statements are looped through over and over again until the condition is no longer met.

The general format of a while loop is:

while test:

code statements

else:

final code statements

Let’s look at a few simple while loops in action.

x = 0

while x < 10:

print('x is currently: ',x)

print(' x is still less than 10, adding 1 to x')

x+=1

x is currently: 0

x is still less than 10, adding 1 to x

x is currently: 1

x is still less than 10, adding 1 to x

x is currently: 2

x is still less than 10, adding 1 to x

x is currently: 3

x is still less than 10, adding 1 to x

x is currently: 4

x is still less than 10, adding 1 to x

x is currently: 5

x is still less than 10, adding 1 to x

x is currently: 6

x is still less than 10, adding 1 to x

x is currently: 7

x is still less than 10, adding 1 to x

x is currently: 8

x is still less than 10, adding 1 to x

x is currently: 9

x is still less than 10, adding 1 to x

x = 0

while x < 10:

print('x is currently: ',x)

print(' x is still less than 10, adding 1 to x')

x+=1

else:

print('All Done!')

x is currently: 0

x is still less than 10, adding 1 to x

x is currently: 1

x is still less than 10, adding 1 to x

x is currently: 2

x is still less than 10, adding 1 to x

x is currently: 3

x is still less than 10, adding 1 to x

x is currently: 4

x is still less than 10, adding 1 to x

x is currently: 5

x is still less than 10, adding 1 to x

x is currently: 6

x is still less than 10, adding 1 to x

x is currently: 7

x is still less than 10, adding 1 to x

x is currently: 8

x is still less than 10, adding 1 to x

x is currently: 9

x is still less than 10, adding 1 to x

All Done!

# break, continue, pass

We can use break, continue, and pass statements in our loops to add additional functionality for various cases. The three statements are defined by:

break: Breaks out of the current closest enclosing loop.

continue: Goes to the top of the closest enclosing loop.

pass: Does nothing at all.

Thinking about break and continue statements, the general format of the while loop looks like this:

while test:

code statement

if test:

break

if test:

continue

else:

break and continue statements can appear anywhere inside the loop’s body, but we will usually put them further nested in conjunction with an if statement to perform an action based on some condition.

Let's go ahead and look at some examples!

x = 0

while x < 10:

print('x is currently: ',x)

print(' x is still less than 10, adding 1 to x')

x+=1

if x==3:

print('x==3')

else:

print('continuing...')

continue

output

x is currently: 0

x is still less than 10, adding 1 to x

continuing...

x is currently: 1

x is still less than 10, adding 1 to x

continuing...

x is currently: 2

x is still less than 10, adding 1 to x

x==3

x is currently: 3

x is still less than 10, adding 1 to x

continuing...

x is currently: 4

x is still less than 10, adding 1 to x

continuing...

x is currently: 5

x is still less than 10, adding 1 to x

continuing...

x is currently: 6

x is still less than 10, adding 1 to x

continuing...

x is currently: 7

x is still less than 10, adding 1 to x

continuing...

x is currently: 8

x is still less than 10, adding 1 to x

continuing...

x is currently: 9

x is still less than 10, adding 1 to x

continuing...

x = 0

while x < 10:

print('x is currently: ',x)

print(' x is still less than 10, adding 1 to x')

x+=1

if x==3:

print('Breaking because x==3')

break

else:

print('continuing...')

continue

x is currently: 0

x is still less than 10, adding 1 to x

continuing...

x is currently: 1

x is still less than 10, adding 1 to x

continuing...

x is currently: 2

x is still less than 10, adding 1 to x

Breaking because x==3

# Functions

## Introduction to Functions

This lecture will consist of explaining what a function is in Python and how to create one. Functions will be one of our main building blocks when we construct larger and larger amounts of code to solve problems.

**So what is a function?**

Formally, a function is a useful device that groups together a set of statements so they can be run more than once. They can also let us specify parameters that can serve as inputs to the functions.

On a more fundamental level, functions allow us to not have to repeatedly write the same code again and again. If you remember back to the lessons on strings and lists, remember that we used a function len() to get the length of a string. Since checking the length of a sequence is a common task you would want to write a function that can do this repeatedly at command.

**def Statements**

Let's see how to build out a function's syntax in Python. It has the following form:

**def** name\_of\_function(arg1,arg2):

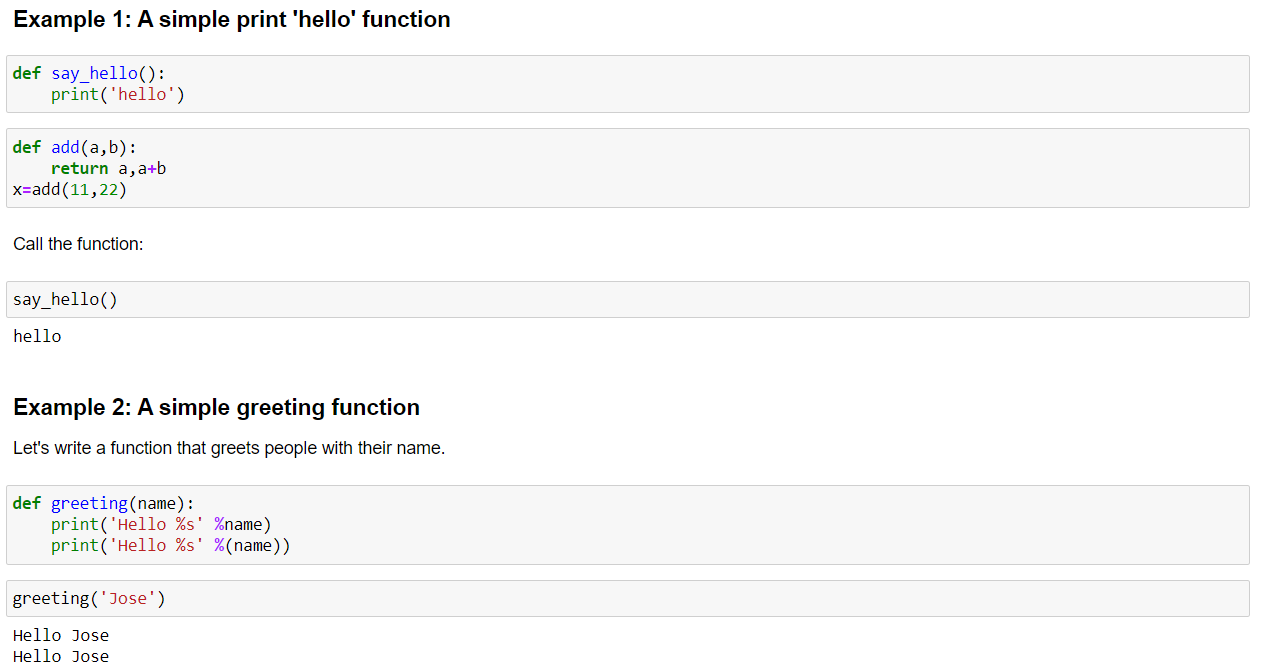
'''

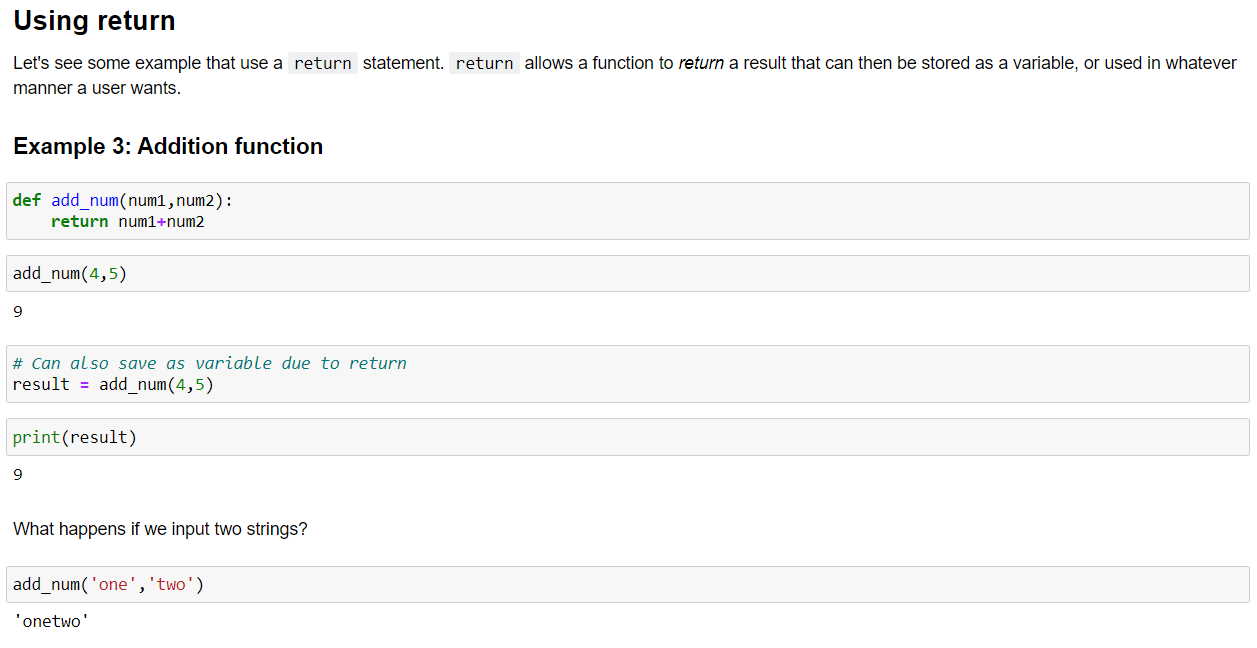
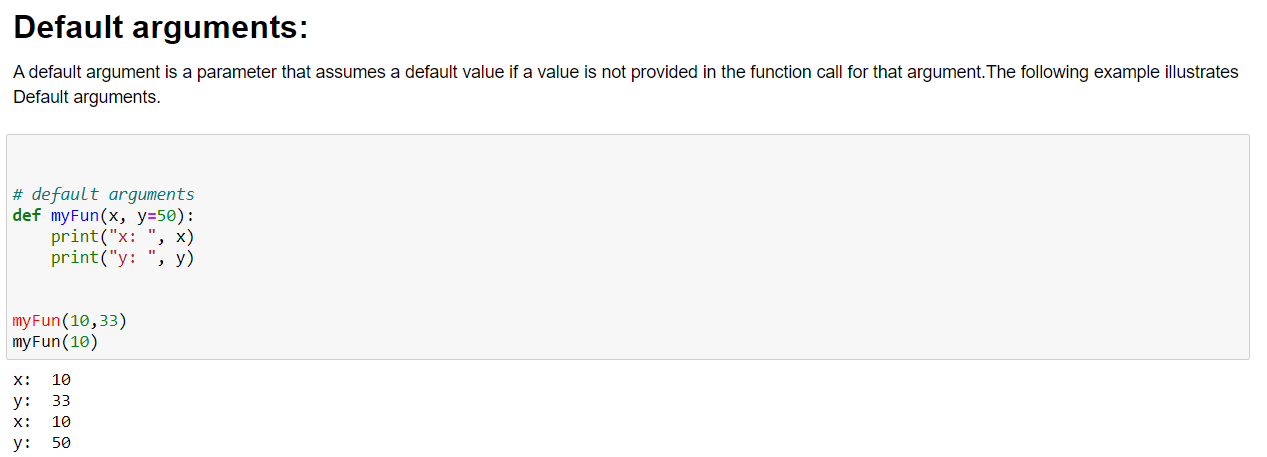
This is where the function's Document String (docstring) goes

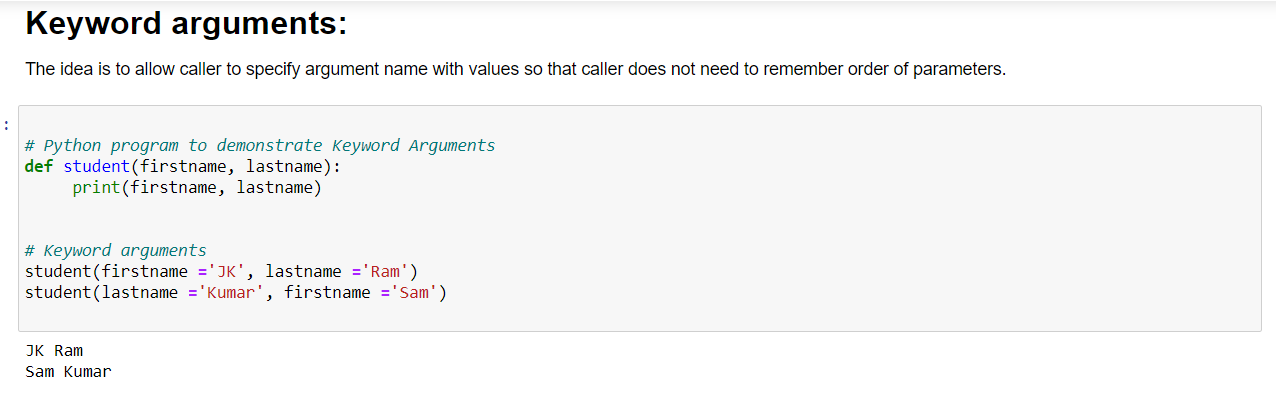
'''

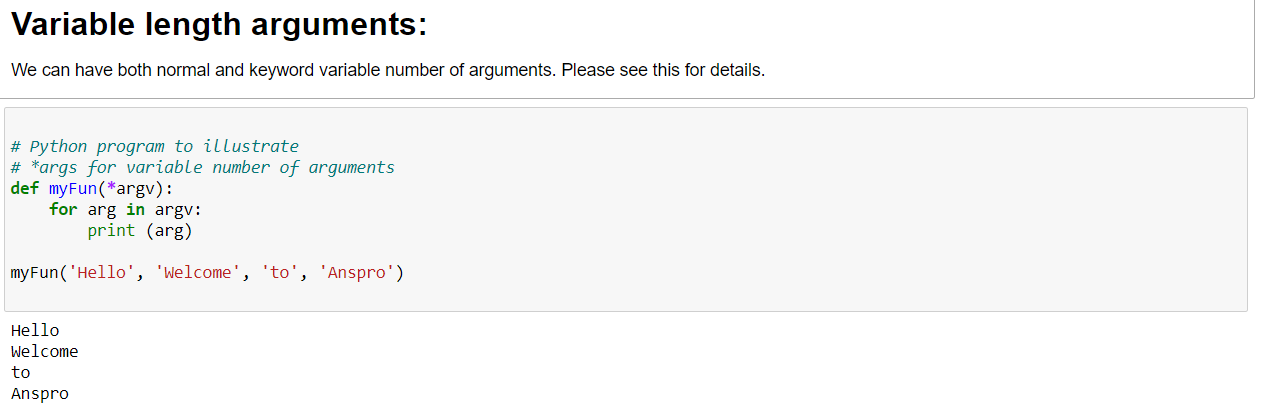
*# Do stuff here*

*# Return desired result*







**Anonymous functions:**

In Python, anonymous function means that a function is without a name. As we already know that def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions. Please see this for details.

*# Python code to illustrate cube of a number*

*# using labmda function*

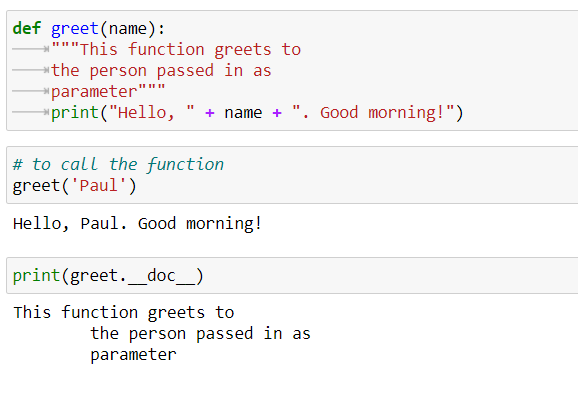
cube **=** **lambda** x,y: x**\***x**\***x

print(cube(3,7))

# Docstring

The first string after the function header is called the docstring and is short for documentation string. It is used to explain in brief, what a function does.

Although optional, documentation is a good programming practice. Unless you can remember what you had for dinner last week, always document your code.



Modules

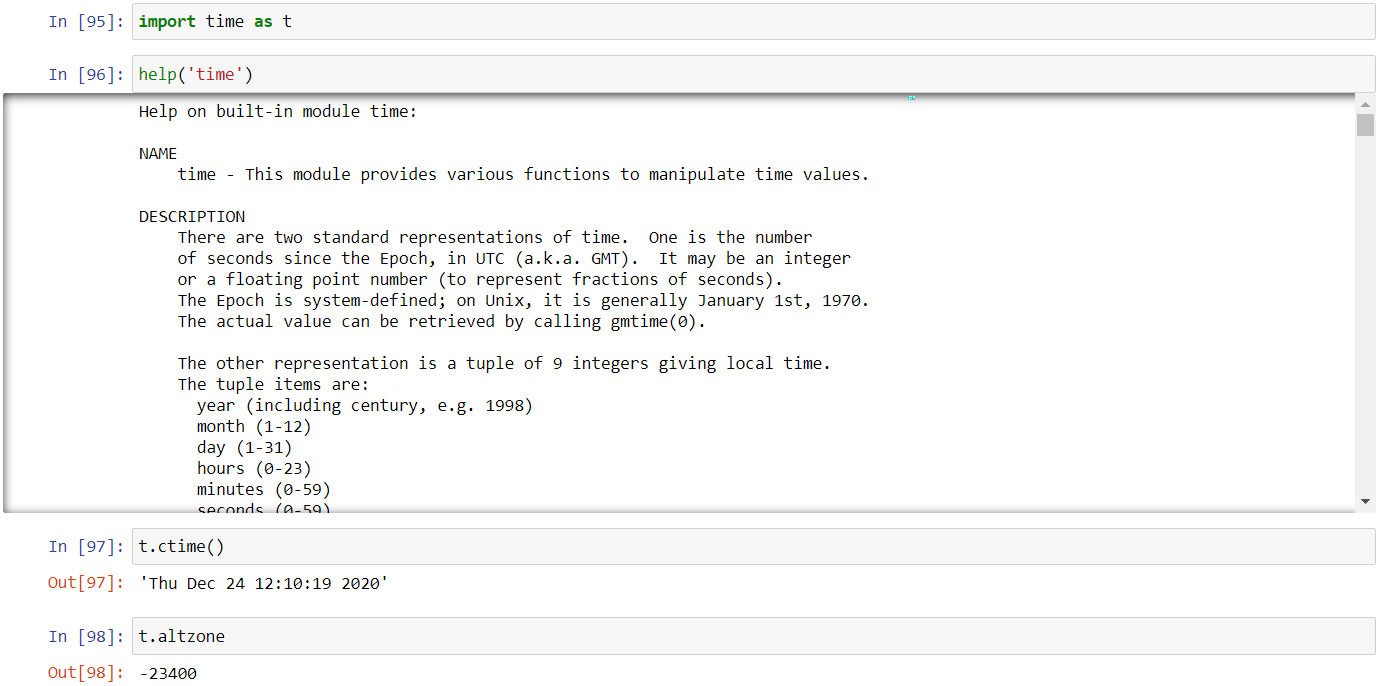
def calc\_shipping():

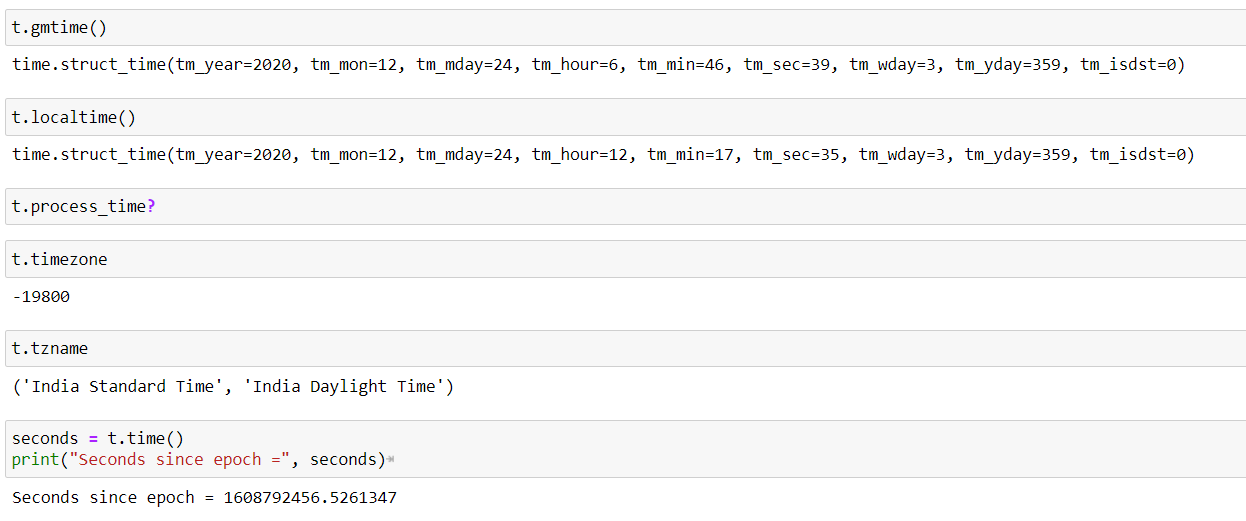
print("calc shipping..")

def calc\_tax():

print("calc tax")





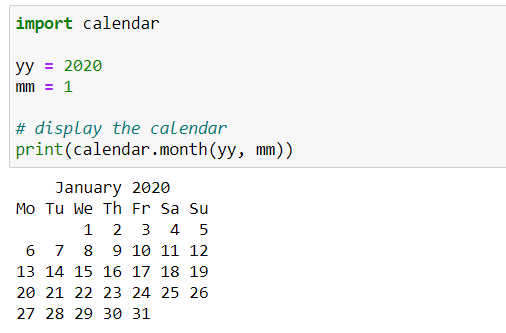


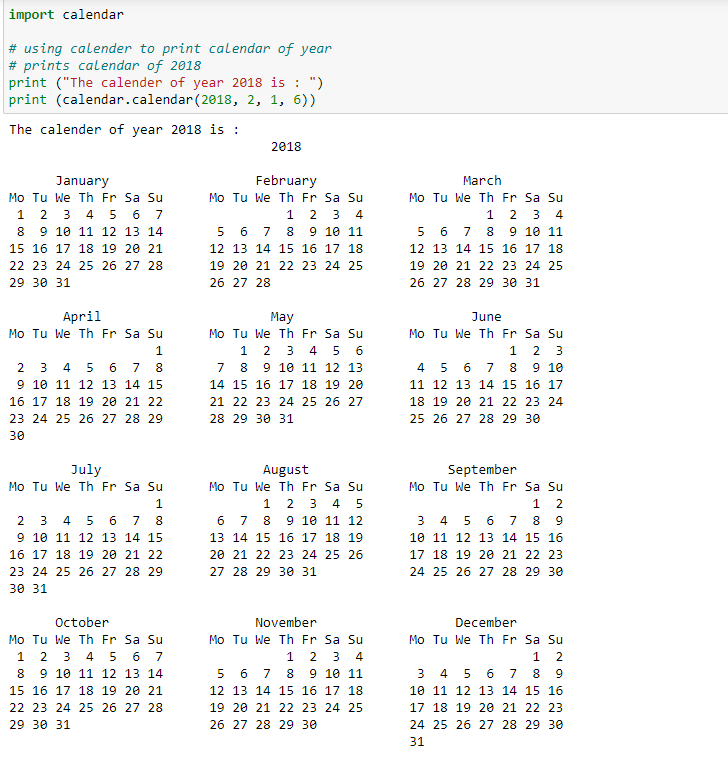


# Python | Calendar Module

Python defines an inbuilt module calendar which handles operations related to calendar.

Calendar module allows output calendars like the program and provides additional useful functions related to the calendar. Functions and classes defined in Calendar module use an idealized calendar, the current Gregorian calendar extended indefinitely in both directions. By default, these calendars have Monday as the first day of the week, and Sunday as the last (the European convention).





**SYSTEM REQUIREMENTS**

**Introduction**

Framework Requirement Specification (SRS) is a focal report, which outlines the foundation of the item headway handle. It records the necessities of a structure and in addition has a delineation of its noteworthy highlight. A SRS is basically an affiliation's seeing (in making) of a customer or potential client's edge work necessities and conditions at a particular point in time (for the most part) before any veritable design or change work. It's a two-way insurance approach that ensures that both the client and the affiliation understand exchange's necessities from that perspective at a given point in time.

The sythesis of programming need detail reduces headway effort, as careful review of the report can reveal oversights, mixed up presumptions, and inconsistencies in front of plan for the change cycle when these issues are less requesting to right. The SRS discusses the thing however not the wander that made it, thusly the SRS fills in as a start for later change of the finished thing.

The SRS may should be changed, be that as it may it gives a foundation to continued with creation appraisal. In direct words, programming need assurance is the starting phase of the item change activity. The SRS implies unraveling the musings in the brains of the clients – the data, into a formal chronicle – the yield of the essential stage. In this manner the yield of the stage is an arranged of formally decided necessities, which in a perfect world are done and relentless, while the information has none of these properties.

Hardware requirements

Processor : Intel i3

RAM : 4GB

Hard Disk : 160GB

Software requirements

* Operating System: Windows 10
* Language : Python
* Tool : Jupyter Notebook

**PYTHON**

Python is an object-oriented programming language created by Guido Rossum in 1989. It is ideally designed for rapid prototyping of complex applications. It has interfaces to many OS system calls and libraries and is extensible to C or C++. Many large companies use the Python programming language include NASA, Google, YouTube, BitTorrent, etc.

Python is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and other advanced fields of Computer Science. Python had deep focus on code readability & this class will teach you python from basics.

Here python language used for IR sensor and HD camera

## Characteristics of Python

* It provides rich data types and easier to read syntax than any other programming languages
* It is a platform independent scripted language with full access to operating system API's
* Compared to other programming languages, it allows more run-time flexibility
* It includes the basic text manipulation facilities of Perl and Awk
* A module in Python may have one or more classes and free functions
* Libraries in Pythons are cross-platform compatible with Linux, MacIntosh, and Windows

**Python 3.7 ide:**

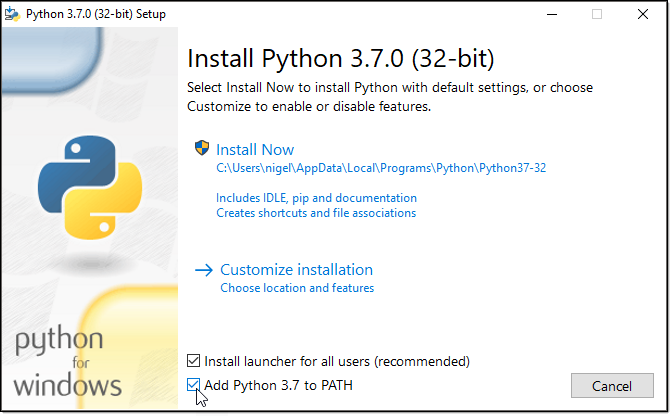
Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

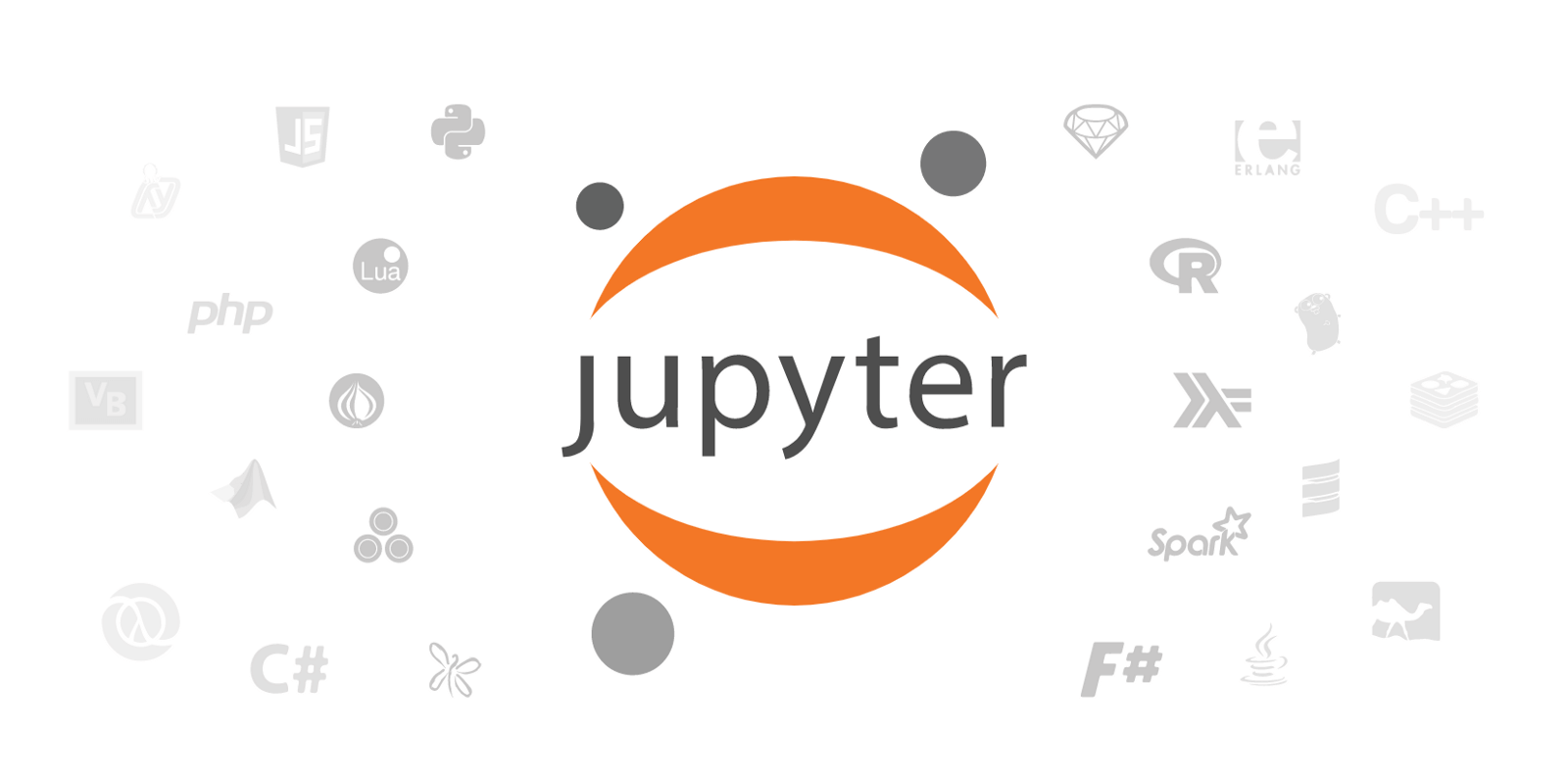
The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see [The Python Standard Library](about:blank#library-index). [The Python Language Reference](about:blank#reference-index) gives a more formal definition of the language. To write extensions in C or C++, read [Extending and Embedding the Python Interpreter](about:blank#extending-index) and [Python/C API Reference Manual](about:blank#c-api-index). There are also several books covering Python in depth.

****

**Jupyter Notebook**

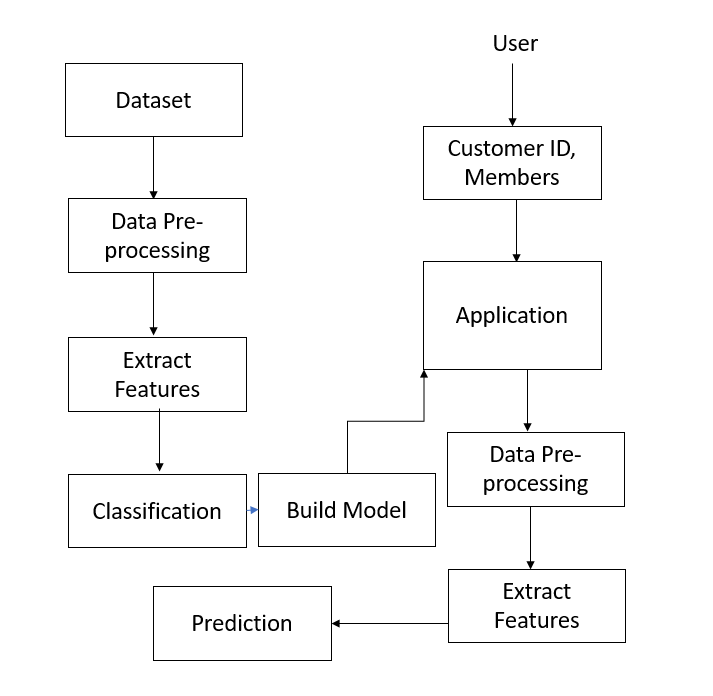


The Jupyter Notebook is an unbelievably incredible asset for intelligently creating & introducing information science ventures. This article will show how to set Jupyter Notebooks on your machine & how to use for data science projects. First, however: what is a "note pad"? A note pad coordinates code & its yield into a solitary archive that consolidates representations, story text, numerical conditions, & other media. This natural work process advances iterative & fast turn of events, settling on note pad an inexorably well-known decision at the core of contemporary data science, examination, & progressively science at large. As a component of the open source Project Jupyter, they ȧre totally free.

The Jupyter venture is the replacement to the prior IPython Notebook, which was first distributed as a model in 2010. Despite the fact that it is conceivable to utilize various programming dialects inside Jupyter Notebooks, this article concentrates on Python as it is the most well-known use case.

To capitalize on this instructional exercise, you should be acquainted with programming, explicitly Python & pȧndas explicitly. All things considered, in the event that you have involvement in another dialect, the Python in this article shouldn't be excessively obscure, will in any case assist you with getting Jupyter Notebooks set up locally. Jupyter Notebooks can likewise go about as an adaptable stage for getting to holds with pȧndas & even Python.

Architecture Diagram



**Implementation**

**Algorithm :**

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

# K-Nearest Neighbors

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

“Birds of a feather flock together.”



Image showing how similar data points typically exist close to each other

Notice in the image above that most of the time, similar data points are close to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

**Note:** An understanding of how we calculate the distance between points on a graph is necessary before moving on. If you are unfamiliar with or need a refresher on how this calculation is done, thoroughly read “Distance Between 2 Points” in its entirety, and come right back.

There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.

## The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbors

3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances

5. Pick the first K entries from the sorted collection

6. Get the labels of the selected K entries

7. If regression, return the mean of the K labels

8. If classification, return the mode of the K labels

## Choosing the right value for K

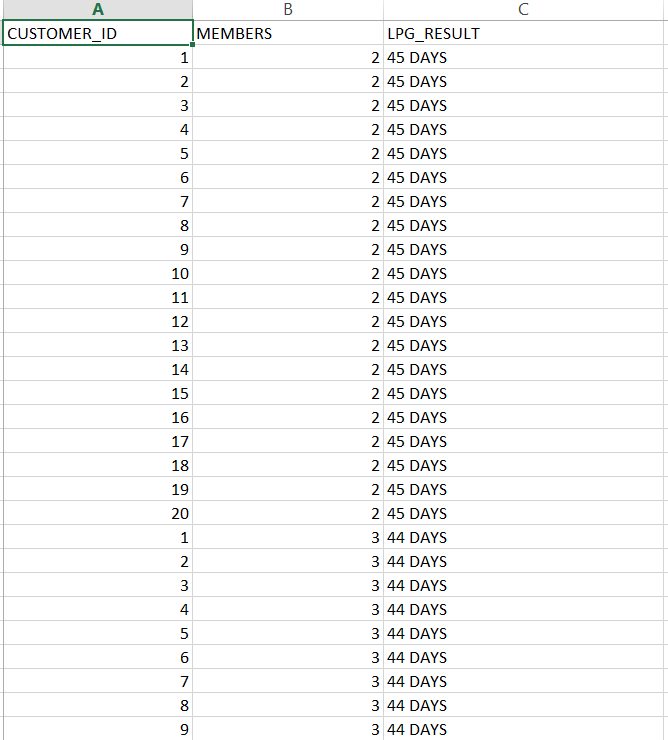
To select the K that’s right for your data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm’s ability to accurately make predictions when it’s given data it hasn’t seen before.

Here are some things to keep in mind:

1. As we decrease the value of K to 1, our predictions become less stable. Just think for a minute, imagine K=1 and we have a query point surrounded by several reds and one green (I’m thinking about the top left corner of the colored plot above), but the green is the single nearest neighbor. Reasonably, we would think the query point is most likely red, but because K=1, KNN incorrectly predicts that the query point is green.
2. Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point). Eventually, we begin to witness an increasing number of errors. It is at this point we know we have pushed the value of K too far.
3. In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.

## Advantages

1. The algorithm is simple and easy to implement.
2. There’s no need to build a model, tune several parameters, or make additional assumptions.
3. The algorithm is versatile. It can be used for classification, regression, and search (as we will see in the next section).



Modules Used

import os

import time

import joblib

import warnings

import numpy as np

import pandas as pd

from sklearn import metrics

from sklearn.metrics import accuracy\_score

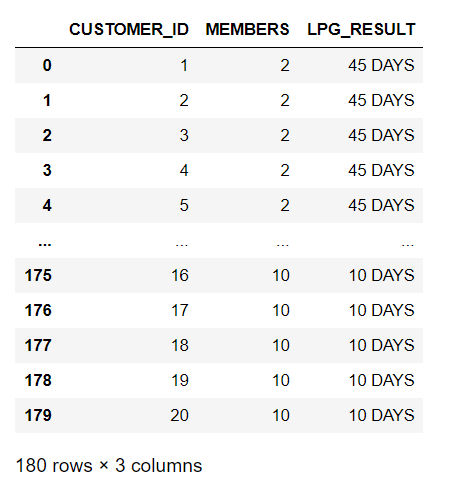
from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

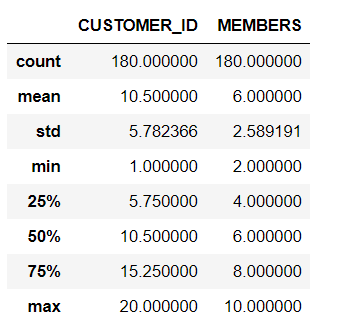
warnings.filterwarnings(action = 'ignore')

from sklearn.neighbors import KNeighborsClassifier

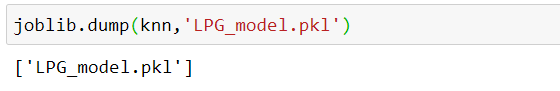
from sklearn.model\_selection import train\_test\_split



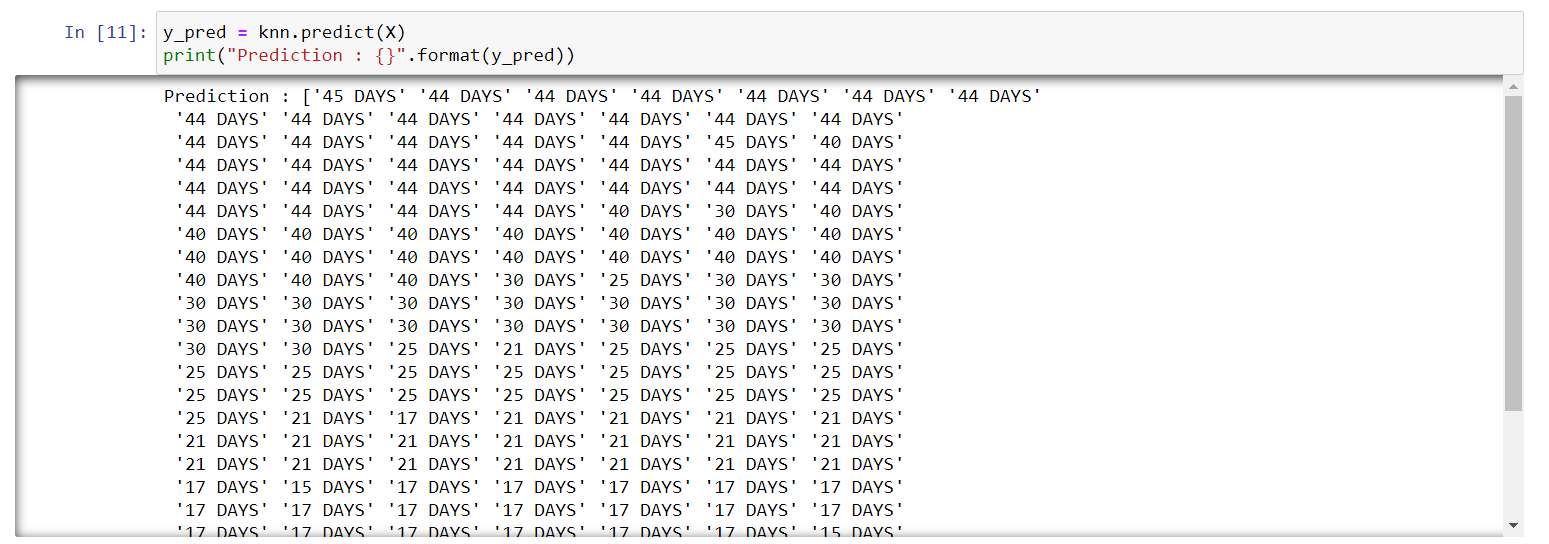
Dataframe describe



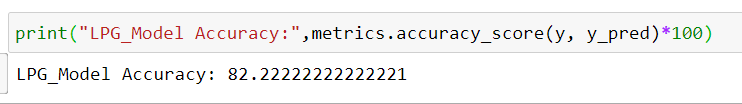
Model Generation



Prediction Test



**Accuracy**



Classicification Report

