# Python Programming Basics with Examples

Python is a popular and a powerful scripting language that can do everything, you can perform [**web scraping**](https://likegeeks.com/python-web-scraping/), networking tools, scientific tools, Raspberry PI programming, Web development, video games, and much more. With Python programming, you can do even system programming regardless the platform you are using.

Machine learning work can be done using some Python libraries like [**TensorFlow**](https://likegeeks.com/tensorflow-examples/) which is used in many artificial intelligence projects.

You can perform [**natural language processing (NLP)**](https://likegeeks.com/nlp-tutorial-using-python-nltk/) using Python libraries like NLTK.

We will discuss the basic Python programming in this post, and on the future posts will build tools and see Python programming in action.

If you find the content of this exercise is a little tricky, tell me so I start from zero level with Python programming, since I assume this for beginners with a very little understanding of what Python is.

## Manipulating Strings

Strings in Python are immutable, so they cannot be changed. Any change to a string’s contents requires making a new copy.

Dealing with string is very simple in Python.

### String Concatenation

|  |
| --- |
| str = "welcome " + "to Python"    print (str) |

This is string concatenation, we can also multiply a string like this:

### String Multiplication

|  |
| --- |
| str = "LikeGeeks" \* 2    print (str) |

### Concatenate with Non-string

You can concatenate with non-string by just converting the value to a string using the str() function like this:

|  |  |
| --- | --- |
| 1  2  3 | str = "This is test number " + str(15)    print (str) |

### Search for a Substring

You can search for a substring using find method like this:

|  |  |
| --- | --- |
| 1  2  3 | str = "welcome to likegeeks website"    print(str.find("likegeeks")) |

The find method prints the position of the first occurrence of the likegeeks string if it’s found.

If nothing found, it will return -1 as a result.

The find function starts from the first character, however, you can start from the nth character like this:

|  |  |
| --- | --- |
| 1  2  3 | str = "welcome to likegeeks website"    print(str.find("likegeeks",12)) |

Since we start from the 12th character, there is no word called likegeeks from that position so it will return -1.

### Get Substrings

So we got the index of the string we’re searching for, now we want to print that matched.

You can print string by index like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | str = "first second third"    print(str[:2])    print(str[2:])    print(str[3:5])    print(str[-1]) |

The first print line prints from the first character till the second character, while the second print line prints from the second character till the end. Notice the position of the colon. The count starts from zero.

If you use a negative number, the counting will start backward instead, like the fourth print line which prints the last character in the string.

### Replace strings

You can replace a string using the replace method like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | str = "This website is about programming"    str2 = str.replace("This", "That")    print(str2) |

If you have many occurrences and you want to replace the first occurrence only, you can specify that:

|  |  |
| --- | --- |
| 1  2  3  4  5 | str = "This website is about programming I like this website"    str2 = str.replace("website", "page",1)    print(str2) |

The first word only got replaced.

### Strip Strings

You can trim white spaces from a string using the strip method like this:

|  |  |
| --- | --- |
| 1  2  3 | str = "   This website is about programming    "    print(str.strip()) |

You can strip from the right only or left only using rstrip() or lstrip() methods respectively.

### Change Character Case

You can change the case of the characters if you want to compare them or something.

|  |  |
| --- | --- |
| 1  2  3  4  5 | str="Welcome to likegeeks"    print(str.upper())    print(str.lower()) |

### Convert Strings to Numbers

We have the str() function which casts the value to a string, but this is not the only cast function in Python programming.

There are int() , float() , long() and other cast functions that you can use.

The int() function cast the input to integer, while float() function cast the input to float.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | str="10"    str2="20"    print(str+str2)    print(int(str)+int(str2)) |

The first print line just concatenates the two numbers without summation, while the second print line adds the two values and print the total.

### Count Strings

You can use the min(), max() and len() functions to calculate the minimum character or maximum character value or the total length of characters.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | str="welcome to likegeeks website"    print(min(str))    print(max(str))    print(len(str)) |

### Iterate over Strings

You can iterate over the string and manipulate every character individually like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | str="welcome to likegeeks website"    for i in range(len(str)):        print(str[i]) |

The len() function counts the length of objects.

### Encode Strings

If you are using Python 3, all strings are stored as Unicode strings by default, but if you are using Python 2 you may need to encode your strings like this:

|  |  |
| --- | --- |
| 1  2  3 | str="welcome to likegeeks website"    str.encode('utf-8') |

## Manipulating numbers

Numbers in Python programming are defined like this:

a=15

You can define integers, floats the same way.

If you have float number you can round it, using int() function like this:

|  |  |
| --- | --- |
| 1  2  3 | a=15.5    print(int(a)) |

### Round Numbers

You can use the round() function to round numbers like this:

|  |  |
| --- | --- |
| 1  2  3 | a=15.5652645    print(round(a,2)) |

Just specify how many numbers need to be rounded to the round() function.

### User-defined Precision Numbers

You may need to work with floating numbers that are of arbitrary precision.

You can use the decimal module that handles numbers of user-defined precision.

You can import the module like this:

|  |  |
| --- | --- |
| 1  2  3 | from decimal import \*    a=Decimal(5.5) |

### Generate Random Numbers

The random module in Python gives functions to generate random numbers.

|  |  |
| --- | --- |
| 1  2  3 | import random    print(random.random()) |

The generated number is between 0.0 and 1.0.

You can generate a random number from your choices like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | import random    numbers=[1,2,3,4,5,6,7]    print(random.choices(numbers)) |

## Manipulating Dates and Times

Python programming language provides a module called datetime that helps in handling dates, times.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import datetime    cur\_date = datetime.datetime.now()    print(cur\_date)    print(cur\_date.year)    print(cur\_date.day)    print(cur\_date.weekday())    print(cur\_date.month)    print(cur\_date.time()) |

You can extract the value you need from the date, like the above examples.

You can get the differences between two times or two dates like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import datetime    time1 = datetime.datetime.now()    time2 = datetime.datetime.now()    timediff = time2 - time1    print(timediff.microseconds()) |

The timediff variable is an object of type timedelta, however, you can create this kind of object yourself like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | time1 = datetime.datetime.now()    time2 = datetime.timedelta(days=3)    time3=time1+time2    print(time3.date()) |

### Format Date and time

The strftime() method takes a format specification and it formats the date or time based on that format.

And the following table specifies some of the format options that you can use:

%a          Prints locale abbreviated name of the weekday.

%A         Prints the weekday.

%w         Prints the day of the week as a number.

%d          Prints the day of the month as a zero-padded number.

%b          Prints the month as locale abbreviated name.

%B          Prints the month as locale full name.

%m        Prints the month as a zero-padded number.

%y          Prints the year as a zero-padded two-digit number.

%Y         Prints the year as a zero-padded four-digit number.

%H         Prints the hour (24-hour clock) as a zero-padded number.

%I           Prints the hour (12-hour clock) as a zero-padded number.

%p          Prints AM or PM.

%M        Prints the minute as a zero-padded number.

%S          Prints the second as a zero-padded number.

|  |  |
| --- | --- |
| 1  2  3  4  5 | import datetime    date1 = datetime.datetime.now()    print(date1.strftime('%d. %B %Y %I:%M%p')) |

### Create Date from String

You can use the strptime() function to create a date from a string like this:

date1=datetime.datetime.strptime(“2015-11-21”, “%Y-%m-%d”)

Also, it can be created like this:

date1= datetime.datetime(year=2015, month=11, day=21)

## Dealing with File System

Dealing with files is very easy in Python programming, believe it or not, this is the easiest language you can use to deal with files. You can say that Python is the easiest language in doing many things.

### Copy Files

The shutil module contains a function for copying files.

|  |  |
| --- | --- |
| 1  2  3 | import shutil    copied\_path = shutil.copy('my\_file.txt', 'copied\_file.txt') |

If my\_file.txt is a [**symlink**](https://likegeeks.com/basic-linux-commands-part2/#Symbolic-Links), the above code will create copied\_file.txt as a separate file.

You can create a copy of a symlink instead like this:

copied\_path = shutil.copy('my\_file.txt', 'copied\_file.txt', follow\_symlinks=False)

### Move Files

You can move files from one location to another like this:

|  |  |
| --- | --- |
| 1  2  3 | import shutil    shutil.move('file1.txt', 'file3.txt') |

You can rename a file using the rename function from os module like this:

|  |  |
| --- | --- |
| 1  2  3 | import os    os.rename('file1.txt', 'file3.txt') |

### Read and Write Text Files

You can use the open function to open files, and then use the read or write methods to read from them and write to them.

|  |  |
| --- | --- |
| 1  2  3  4  5 | fd = open('file1.txt')    content = fd.read()    print(content) |

First, we open the file for reading using the open function, then we start reading the file content using read function, finally, we put the grabbed content into the variable content.

You can specify how many bytes you want to read for the read() function:

fd.read(20)

If the file is not too big, you can read the entire contents into a list, then iterate over that list to print the output.

|  |  |
| --- | --- |
| 1  2  3 | content = fd.readlines()    print(content[0]) |

You can write to a file by specifying the mode to open function like this. You have two modes of writing, the write mode and append mode.

This is the write mode where you will overwrite the old file content.

|  |  |
| --- | --- |
| 1  2  3 | fd = open('file1.txt','w')    content = fd.write('YOUR CONTENT GOES HERE') |

And this is the append mode:

|  |  |
| --- | --- |
| 1  2  3 | fd = open('file1.txt','a')    content = fd.write('YOUR CONTENT GOES HERE') |

### Creating Directories

You can create a new directory using mkdir function from os module like this:

|  |  |
| --- | --- |
| 1  2  3 | import os    os.mkdir('./NewFolder) |

This code will throw an error if the directory exists. Don’t worry, we will talk about exception handling in future posts so you can avoid such errors.

### Get Access & Modification & Creation Time

You can use getmtime(), getatime() and getctime() to get modification time, access time and creation time respectively.

The returned time is formatted as a Unix timestamp, we can convert it to human readable format like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import os    import datetime    tim=os.path.getctime('./file1.txt')    print(datetime.datetime.fromtimestamp(tim)) |

### Iterating Over Files

You can use listdir() function from os module to get the files:

|  |  |
| --- | --- |
| 1  2  3  4  5 | import os    files= os.listdir('.')    print(files) |

Also, you can use the glob module to do the same thing:

|  |  |
| --- | --- |
| 1  2  3  4  5 | import glob    files=glob.glob('\*')    print(files) |

You can write any extension for file globbing, like \*.doc to get all word documents only.

### Serializing Python Objects

This process is used to serialize Python object to a byte stream for later reuse.

You can do that using pickle module:

|  |  |
| --- | --- |
| 1  2  3  4  5 | import pickle    fd = open('myfile.pk ', 'wb')    pickle.dump(mydata,fd) |

You can deserialize this data using load() function like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | import pickle    fd = open('myfile.pk ', 'rb')    mydata = pickle.load(fd) |

### Compressing Files

The Python standard library enables you to work with different types of compressed files like tar, zip, gzip, bzip2.

To work with a zip file you can use the zipfile module:

|  |  |
| --- | --- |
| 1  2  3  4  5 | import zipfile    my\_zip = zipfile.ZipFile('zipped\_file.zip', mode='r')    print(file.namelist()) |

You can create a zip file from your files like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import zipfile    file=zipfile.ZipFile('files.zip','w')    file.write('file1.txt')    file.close() |

You can extract the zip file using extractall() method like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import zipfile    file=zipfile.ZipFile('files.zip','r')    file.extractall()    file.close() |

Also, you can append files to existing zip file by using append mode like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import zipfile    file=zipfile.ZipFile('files.zip','a')    file.write('file2.txt')    file.close() |

The same coding above is done when dealing with gz or bz files. You need to use gzip module or bz2 module.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import gzip    import bz2    gz\_file=gzip.GzipFile('files.gz','r')    bz\_file=bz2.BZ2File('fiels.bz2','r') |

Then you can read and write in the same way.

You can deal with rar files using **unrar** package. First, install the package

pip install unrar

Then you can use it the same way.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import unrar.rarfile    m=unrar.rarfile.RarFile('file.rar')    m.namelist()    m.extractall() |

### Parse CSV Files

A very useful package called **pandas**. This package can parse CSV and excel files, and extract data from it easily.

First, install the package

pip install pandas

Then you can use it in your modules like this:

|  |  |
| --- | --- |
| 1  2  3 | import pandas    data=pandas.read\_csv('file.csv) |

Pandas treats the first column as a label for each row by default.You can pass the  index\_col parameter to specify the column index if it’s not the first column.

If there are no row labels in your document, you should use the parameter index\_col=False.

To write to CSV file, you can use to\_csv() method.

data.to\_csv('file.csv)

### Parse Excel Files

You can use read\_excel() method from pandas module to parse excel files.

data = pd.read\_excel('file.xls', sheetname='Sheet1')

If you have multiple sheets, you can load it like this:

data = pd.ExcelFile('file.xls')

You can write to excel files like this:

data.to\_excel('file.xls', sheet='Sheet1')

## Networking and Connectivity

The Python language has a socket class that provides a way of accessing the network at a low level, it has to support many networking protocols.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import socket    host = '192.168.1.5'    port = 5050    m\_sock = socket.create\_connection ((host, port)) |

This code we establish a connection to a host on IP 192.168.1.5 on port 5050.

Once a socket has been opened, you can send and receive data.

m\_sock.sendall(b'Hello World')

Notice that I used the b character before the string because data needs to be a byte string.

If have a bigger message you should iterate over your message like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | msg = b'Longer Message Goes Here'    mesglen = len(msg)    total = 0    while total < msglen:        sent = m\_sock.send(msg[total:])        total = total + sent |

For receiving data, you need to tell the methods how many bytes to read in at a time.

data\_in = m\_sock.recv(2000)

This works because you know for sure that the message being sent is less than 2000 bytes long.

If the message is big, you must loop over and over until you collect all of the separate chunks.

|  |  |
| --- | --- |
| 1  2  3 | buffer = bytearray(b' ' \* 2000)    m\_sock.recv\_into(buffer) |

Here we define an empty buffer, then we start to write the message into the buffer.

### Reading an E-Mail from POP mail server

We already discussed [**Linux mail server**](https://likegeeks.com/linux-mail-server/) and everything about it, now, how we can access it using the Python programming?

The poplib module enables you to communicate with a POP server.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import getpass,poplib    pop\_serv = poplib.POP3('192.168.1.5')    pop\_serv.user("myuser")    pop\_serv.pass\_(getpass.getpass()) |

The getpass module asks the end user for passwords securely.

If you need a secure connection, use the POP3\_SSL class instead.

To get the message list and the message count, you can do it like this:

|  |  |
| --- | --- |
| 1  2  3 | msg\_list = pop\_serv.list()              # to list the messages    msg\_count = pop\_serv.msg\_count()        # to get message count |

Don’t forget to close any open connections after you finish working with the POP server.

pop\_serv.quit()

### Reading an E-Mail from IMAP mail server

You can work with an [**IMAP email server**](https://likegeeks.com/linux-mail-server/#POP3-and-IMAP-Protocol-Basics) using the imaplib module.

|  |  |
| --- | --- |
| 1  2  3  4  5 | import imaplib, getpass    my\_imap = imaplib.IMAP4('imap.server.com')    my\_imap.login("myuser", getpass.getpass()) |

If you are using SSL on your IMAP server, you should use the IMAP4\_SSL class instead.

To get a list of e-mails, you need to actually do a search like this:

data = my\_imap.search(None, 'ALL')

Then you can iterate over the returned e-mail indices in the data variable and fetch the message

msg = my\_imap.fetch(email\_id, '(RFC822)')

Finally, don’t forget to close the connection:

|  |  |
| --- | --- |
| 1  2  3 | my\_imap.close()    my\_imap.logout() |

### Sending an E-Mail

E-mails are sent using the SMTP protocol. The smtplib in Python is used to handle this.

|  |  |
| --- | --- |
| 1  2  3  4  5 | import smtplib, getpass    my\_smtp = smtplib.SMTP(smtp.server.com')    my\_smtp.login("myuser", getpass.getpass()) |

If you are using SSL on your [**SMTP server**](https://likegeeks.com/linux-mail-server/#Linux-SMTP-Server), you should use the SMTP\_SSL class instead.

Once the connection is opened, you can send the message like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | from\_addr = 'me@example.com'    to\_addr = 'you@example.com'    msg = 'From: me@example.com\r\nTo: you@example.com\r\n\r\nHello, this is a test message'    my\_smtp.sendmail(from\_addr, to\_addr, msg) |

### Web Crawling

To talk to a web server, you need to use urllib.request submodule.

|  |  |
| --- | --- |
| 1  2  3  4  5 | import urllib.request    my\_web = urllib.request.urlopen('https://www.google.com')    print(my\_web.read()) |

### Post to a Web Page

If you need to submit a web form, you know that you should send POST request to the web page and that’s what we will do.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import urllib.request    my\_data = b'Your Data Goes Here'    my\_req = urllib.request.Request('http://localhost', data=my\_data,method='POST')    my\_frm = urllib.request.urlopen(my\_req)    print(my\_frm.status) |

Note that we can use mechanize or urllib2, there are many ways to achieve that.

### Create a Mini Server

The socket class supports listening for incoming connections.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | import socket    host = ''    port = 3535    my\_server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)    my\_server.bind((host, port))    my\_server.listen(1) |

Now you can accept incoming connections like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | addr = my\_server.accept()    print('Connected ... ', addr)    data = conn.recv(1024) |

Also, don’t forget to close the connection after you’ve finished

conn.close()

## Threading in Python Programming

Running multiple processes in parallel, which is called threading is very useful, especially when you need to run a process in a different thread and keep your current thread unattached to avoid freezing.

The Python language includes a module called threading that contains a Thread class.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import threading    def print\_message():        print('The message got printed from a different thread')    my\_thread = threading.Thread(target=print\_message)    my\_thread.start() |

If the function takes a long time to finish, you can check to see whether it is still running or not by using the is\_alive() method.

Sometimes your threads need to access global resources safely. You can do that using locks.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import threading    num = 1    my\_lock = threading.Lock()    def my\_func():        global num, my\_lock        my\_lock.acquire()        sum = num + 1        print(sum)        my\_lock.release()    my\_thread = threading.Thread(target=my\_func)    my\_thread.start() |

## Using Raspberry PI

With raspberry PI you can create your own technology. It’s a single board computer with cheap price.

You can use the Python module **RPi.GPIO** to work with the Raspberry PI.

First, install the package in your Raspberry PI like this:

$ sudo apt-get install python-dev python-rpi.gpio

Now you can use it in your scripts. You can write output on the Raspberry Pi’s GPIO bus:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import RPi.GPIO as GPIO    GPIO.setmode(GPIO.BOARD)    GPIO.setup(1, GPIO.OUT, initial=GPIO.LOW)    GPIO.output(1, GPIO.HIGH) |

### Reading from the Raspberry Pi’s GPIO

You can use the RPi.GPIO Python module to read data in from the GPIO like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | import RPi.GPIO    RPi.GPIO.setup(1, GPIO.IN)    if RPi.GPIO.input(1):        print('Input was HIGH')    else:        print('Input was LOW') |