Open a file and print the content:

```python
f = open("demofile.txt", "r")
print(f.read())
```

Definition and Usage

The `open()` function opens a file, and returns it as a file object.

Read more about file handling in our chapters about File Handling.

Syntax

```python
open(file, mode)
```

Parameter Values

Parameter	Description
file	The path and name of the file
mode	A string, define which mode you want to open the file in:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exist

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

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

```
In [5]: f = open("C:/NKNPYTHCODES-03SEP2021/iris.csv", "r")
        print(f.read())
```

```
Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
3,4.7,3.2,1.3,0.2,Iris-setosa
4,4.6,3.1,1.5,0.2,Iris-setosa
5,5.0,3.6,1.4,0.2,Iris-setosa
6,5.4,3.9,1.7,0.4,Iris-setosa
7,4.6,3.4,1.4,0.3,Iris-setosa
8,5.0,3.4,1.5,0.2,Iris-setosa
9,4.4,2.9,1.4,0.2,Iris-setosa
10,4.9,3.1,1.5,0.1,Iris-setosa
11,5.4,3.7,1.5,0.2,Iris-setosa
12,4.8,3.4,1.6,0.2,Iris-setosa
13,4.8,3.0,1.4,0.1,Iris-setosa
14,4.3,3.0,1.1,0.1,Iris-setosa
```

```
In [8]: f = open("C:/NKNPYTHCODES-03SEP2021/iris.csv", "r")

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

```
Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species

1,5.1,3.5,1.4,0.2,Iris-setosa

2,4.9,3.0,1.4,0.2,Iris-setosa

3,4.7,3.2,1.3,0.2,Iris-setosa

4,4.6,3.1,1.5,0.2,Iris-setosa

5,5.0,3.6,1.4,0.2,Iris-setosa

6,5.4,3.9,1.7,0.4,Iris-setosa

7,4.6,3.4,1.4,0.3,Iris-setosa

8,5.0,3.4,1.5,0.2,Iris-setosa

9,4.4,2.9,1.4,0.2,Iris-setosa
```

```
In [1]:  # load csv module
         import csv

         # open file for reading
         with open('C:/NKNPYTHCODES-03SEP2021/Iris.csv') as csvDataFile:

             # read file as csv file
             csvReader = csv.reader(csvDataFile)

             # for every row, print the row
             for row in csvReader:
                 print(row)
```

```
['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm', 'Species']
['1', '5.1', '3.5', '1.4', '0.2', 'Iris-setosa']
['2', '4.9', '3.0', '1.4', '0.2', 'Iris-setosa']
['3', '4.7', '3.2', '1.3', '0.2', 'Iris-setosa']
['4', '4.6', '3.1', '1.5', '0.2', 'Iris-setosa']
['5', '5.0', '3.6', '1.4', '0.2', 'Iris-setosa']
['6', '5.4', '3.9', '1.7', '0.4', 'Iris-setosa']
['7', '4.6', '3.4', '1.4', '0.3', 'Iris-setosa']
['8', '5.0', '3.4', '1.5', '0.2', 'Iris-setosa']
['9', '4.4', '2.9', '1.4', '0.2', 'Iris-setosa']
['10', '4.9', '3.1', '1.5', '0.1', 'Iris-setosa']
['11', '5.4', '3.7', '1.5', '0.2', 'Iris-setosa']
```

Read CSV

csv stands for "comma-separated values". they are a common file format for data exchange, storage, and editing. in fact, the .csv files you may open in a spreadsheet application (like excel) are just plain text files, with one very simple rule:

all of the fields in your records must be separated by commas.

For example, the following might be a small part of a sample spreadsheet in csv format:

```
"first_name","last_name","email","address","city","state","zip","phone"
"charlie","davidson","charlie@davidson.com","123 main street, akron, ohio","akron, ohio","z
"tanya","jones","tanya@jones.com", "734 main street", "ny", "new york", "nyc", "12354"
```

We import the **csv module**. This is a simple module to read/write csv files in python.

```
import csv
```

You can read every row in the file. Every row is returned as an array and can be accessed as such, to print the first cells we could simply write:

```
print(row[0])
```

For the second cell, you would use:

```
print(row[1])
```

It is better to have the data in arrays, because it's easier to understand than those indices like [0],[1],[2] etc.

You can do that by adding the cells to a list during loading. The example below demonstrates this:

```
# load module
import csv

# first cell data
dates = []

# second cell data
scores = []

# open file for reading
with open('file.csv') as csvDataFile:

    # open file as csv file
    csvReader = csv.reader(csvDataFile)

    # loop over rows
    for row in csvReader:

        # add cell [0] to list of dates
        dates.append(row[0])

        # add cell [1] to list of scores
        scores.append(row[1])

# output data
print(dates)
print(scores)
```

```python
import csv

def readMyFile(filename):
    dates = []
    scores = []

    with open(filename) as csvDataFile:
        csvReader = csv.reader(csvDataFile)
        for row in csvReader:
            dates.append(row[0])
            scores.append(row[1])

    return dates, scores


dates,scores = readMyFile('file.csv')

print(dates)
print(scores)
```

```python
# load csv module
import csv

# open file for reading
with open('C:/NKNPYTHCODES-03SEP2021/Iris.csv') as csvDataFile:

    # read file as csv file
    csvReader = csv.reader(csvDataFile)

    # for every row, print the row
    for row in csvReader:
        print(row)
```

```python
# load csv module
import csv

IDList = []

SepalLength = []
```

```python
# open file for reading
with open('C:/NKNPYTHCODES-03SEP2021/Iris.csv') as csvDataFile:

    # read file as csv file
    csvReader = csv.reader(csvDataFile)

    # for every row, print the row
    for row in csvReader:
        IDList.append(row[0])
        SepalLength.append(row[1])

    print('Values in IDLIST')
    for x in IDList:
        print(x, end = ' ')

    print('\n \n')
    print('Values in SepaLength')
    for x in SepaLength:
        print(x, end = ' ')
```

```
Values in IDLIST
Id 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 4
5 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150


Values in SepaLength
SepalLengthCm 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2
5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2
5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1
5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3
6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

```python
# open file for reading
with open('C:/NKNPYTHCODES-03SEP2021/Iris.csv') as csvDataFile:

    # read file as csv file
    csvReader = csv.reader(csvDataFile)

    # for every row, print the row
    for row in csvReader:
        IDList.append(row[0])
        SepalLength.append(row[1])

    print('Values in IDLIST')
    for x in IDList:
        print(x, end = '\t')

    print('\n \n')
    print('Values in SepaLength')
```

```python
    for x in SepalLength:
      print(x, end = '\t')
```

```python
import csv

def readMyFile(filename):
    IDList = []
    SepalLength = []

    with open(filename) as csvDataFile:
      csvReader = csv.reader(csvDataFile)
      for row in csvReader:
        IDList.append(row[0])
        SepalLength.append(row[1])

    return IDList, SepalLength

IDList, SepalLength = readMyFile('C:/NKNPYTHCODES-03SEP2021/Iris.csv')

print(IDList)
print('\n\n')
print(SepalLength)


ncol = len(next(reader)) # Read first line and count columns
f.seek(0)           # go back to beginning of file
for row in reader:
    pass #do stuff
```

C:/NKNPYTHCODES-03SEP2021/Iris.csv
```python
from csv import reader
# open file in read mode
with open('C:/NKNPYTHCODES-03SEP2021/Iris.csv', 'r') as read_obj:
    # pass the file object to reader() to get the reader object
    csv_reader = reader(read_obj)
    # Iterate over each row in the csv using reader object
    for row in csv_reader:
        # row variable is a list that represents a row in csv
        print(row)
```

#PRINTS THE NO OF COLUMNS OR ATTRIBUTES IN THE CSV FILE.
```python
print(len(row))
```

```python
from csv import reader
# skip first line i.e. read header first and then iterate over each row od csv as a list
with open('C:/NKNPYTHCODES-03SEP2021/Iris.csv', 'r') as read_obj:
    csv_reader = reader(read_obj)
    header = next(csv_reader)
    # Check file as empty
    if header != None:
        # Iterate over each row after the header in the csv
        for row in csv_reader:
            # row variable is a list that represents a row in csv
            print(row)


# This code reads only the column names from a given csv file.
from csv import DictReader
# open file in read mode
with open('C:/NKNPYTHCODES-03SEP2021/Iris.csv', 'r') as read_obj:
    # pass the file object to DictReader() to get the DictReader object
    csv_dict_reader = DictReader(read_obj)
    # get column names from a csv file
    column_names = csv_dict_reader.fieldnames
    print(column_names)


# This code reads only the column names from a given csv file and stores it in a List data structure.
from csv import DictReader
# open file in read mode

colnames = [] #A List structure to store the column names of a csv file.

with open('C:/NKNPYTHCODES-03SEP2021/Iris.csv', 'r') as read_obj:
    # pass the file object to DictReader() to get the DictReader object
    csv_dict_reader = DictReader(read_obj)
    # get column names from a csv file
    column_names = csv_dict_reader.fieldnames
    print(column_names)


#Print the first column name from column_names
#print(column_names[0])


for x in column_names:
```

```
print(x, sep = '')
```

```
In [6]: %run C:/NKNPYTHCODES-03SEP2021/script02.py

        Enter the number: 234
        Input Number by the user =  234
        Orignum =  0
        Input Number by the user =  0
        Reversed Number =  432
```

C:/NKNPYTHCODES-03SEP2021/script02.py

#Normal Logic to reverse a given number in python

orignum = int(input("Enter the number: "))

n1 = orignum

revnum = 0

sum = 0

print("Input Number by the user = ", orignum)

while(orignum>0):

   mod1 = orignum % 10

   revnum = (revnum * 10) + mod1
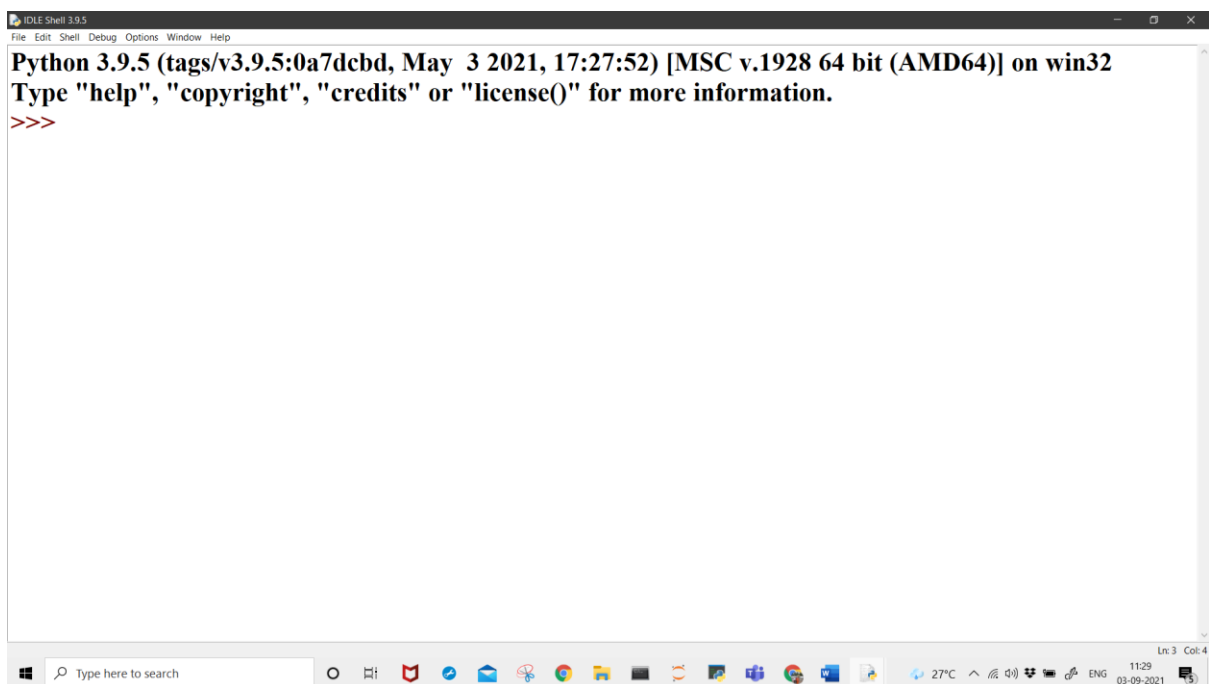
   orignum = (orignum // 10)

print("Orignum = ", orignum)

print("Input Number by the user = ", orignum)

print("Reversed Number = ", revnum)

****

RUNNING A .PY SCRIPT FILE USING IDLE IN PYTHON

```python
#Normal Logic to reverse a given number in python
orignum = int(input("Enter the number: "))
n1 = orignum
revnum = 0
sum = 0
print("Input Number by the user = ", orignum)
while(orignum>0):
    mod1 = orignum % 10
    revnum = (revnum * 10) + mod1
    orignum = (orignum // 10)

print("Orignum = ", orignum)
print("Input Number by the user = ", orignum)
print("Reversed Number = ", revnum)
```
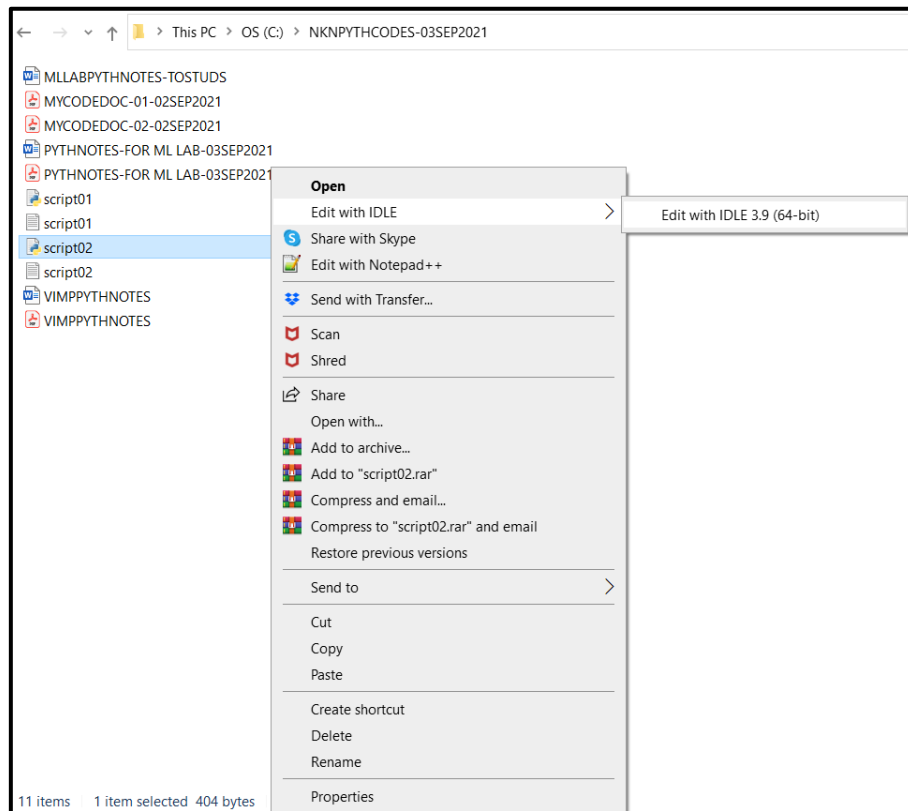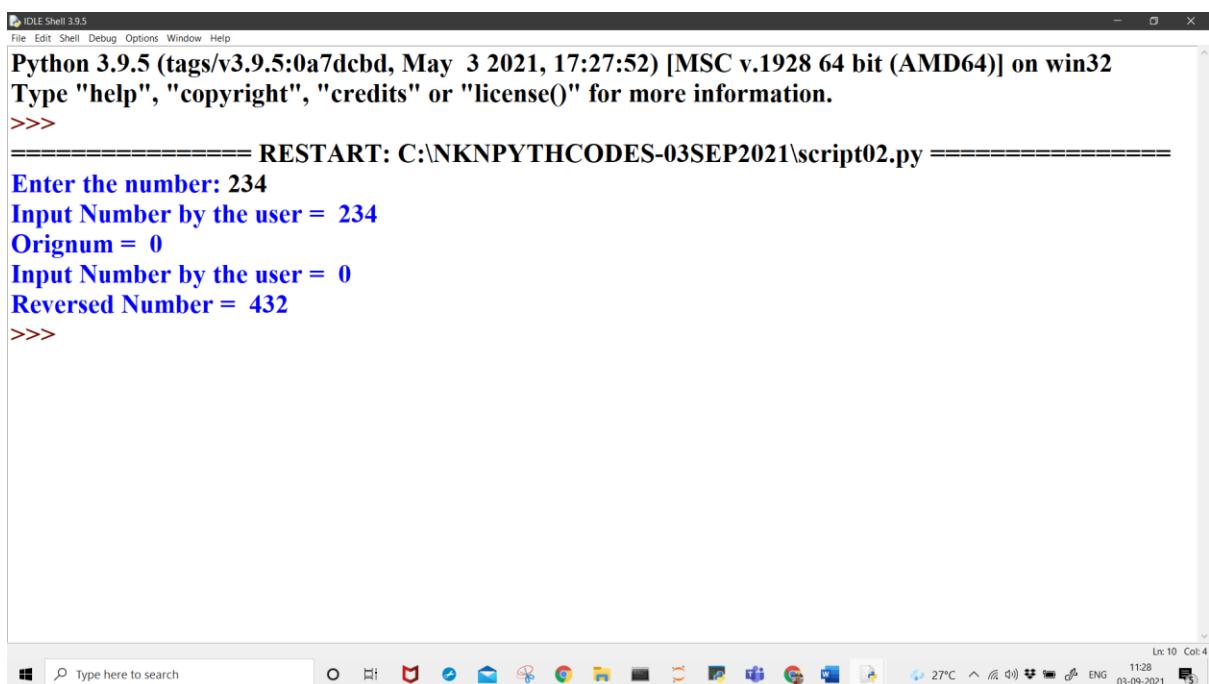
```
IDLE Shell 3.9.5                                                                    —  □  ×
File  Edit  Shell  Debug  Options  Window  Help
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\NKNPYTHCODES-03SEP2021\script02.py ================
Enter the number: 234
Input Number by the user =  234
Orignum =  0
Input Number by the user =  0
Reversed Number =  432
>>>



                                                                          Ln: 10  Col: 4
```

```python
a = "Hello"
print(a)



a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)



a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)



a = "Hello, World!"
print(a[1])



for x in "banana":
  print(x)



a = "Hello, World!"
print(len(a))



txt = "The best things in life are free!"
print("free" in txt)



txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.")



txt = "The best things in life are free!"
print("expensive" not in txt)



txt = "The best things in life are free!"
if "expensive" not in txt:
  print("No, 'expensive' is NOT present.")
```

```python
b = "Hello, World!"
print(b[2:5])


b = "Hello, World!"
print(b[:5])


b = "Hello, World!"
print(b[2:])
```

Negative Indexing

```python
b = "Hello, World!"
print(b[-5:-2])


a = "Hello, World!"
print(a.upper())


a = "Hello, World!"
print(a.lower())


a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"


a = "Hello, World!"
print(a.replace("H", "J"))


a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']


a = "Hello"
b = "World"
c = a + b
print(c)
```

```python
a = "Hello"
b = "World"
c = a + " " + b
print(c)


age = 36
txt = "My name is John, I am " + age
print(txt)


age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))


quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

Output:

```
I want 3 pieces of item 567 for 49.95 dollars.
```

```python
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

Output:

```
I want to pay 49.95 dollars for 3 pieces of item 567
```

Escape Characters

Other escape characters used in Python:

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string

index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value

strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Python Lists

```
mylist = ["apple", "banana", "cherry"]
```

List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

Example

Create a List:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

List items can be of any data type:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

```
list1 = ["abc", 34, True, 40, "male"]
```

type()

From Python's perspective, lists are defined as objects with the data type 'list':

```
<class 'list'>
```

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

The list() Constructor

It is also possible to use the `list()` constructor when creating a new list.

```python
# note the double round-brackets
thislist = list(("apple", "banana", "cherry"))
print(thislist)
```

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered* and changeable. No duplicate members.

*As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

List items are indexed and you can access them by referring to the index number:

Note: The first item has index 0.

Negative Indexing

Negative indexing means start from the end

`-1` refers to the last item, `-2` refers to the second last item etc.

```python
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

```python
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Note: The search will start at index 2 (included) and end at index 5 (not included).

```python
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

Specify negative indexes if you want to start the search from the end of the list:

This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):

```python
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

```python
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

```python
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

```python
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

```python
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```

```python
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

To insert a list item at a specified index, use the `insert()` method.

The `insert()` method inserts an item at the specified index:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

To append elements from *another list* to the current list, use the `extend()` method.

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

The `extend()` method does not have to append *lists*, you can add any iterable object (tuples, sets, dictionaries etc.).

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

The `pop()` method removes the specified index.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

If you do not specify the index, the `pop()` method removes the last item.

```python
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

The `del` keyword also removes the specified index:

```python
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

The `del` keyword can also delete the list completely.

```python
thislist = ["apple", "banana", "cherry"]
del thislist
```

The `clear()` method empties the list.

The list still remains, but it has no content.

```python
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

You can loop through the list items by using a `for` loop:

```python
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

You can also loop through the list items by referring to their index number.

Use the `range()` and `len()` functions to create a suitable iterable.

Print all items by referring to their index number:

```python
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

Using a While Loop

You can loop through the list items by using a `while` loop.

Use the `len()` function to determine the length of the list, then start at 0 and loop your way through the list items by refering to their indexes.

Remember to increase the index by 1 after each iteration.

```python
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

List Comprehension offers the shortest syntax for looping through lists:

```python
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a `for` statement with a conditional test inside:

```python
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

With list comprehension you can do all that with only one line of code:

```python
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

The Syntax

```
newlist = [expression for item in iterable if condition == True]
```

The return value is a new list, leaving the old list unchanged.

Condition

The *condition* is like a filter that only accepts the items that valuate to `True`.

Example

Only accept items that are not "apple":

```
newlist = [x for x in fruits if x != "apple"]
```

The condition `if x != "apple"` will return `True` for all elements other than "apple", making the new list contain all fruits except "apple".

The *condition* is optional and can be omitted:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits]

print(newlist)
```

Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

Arrays

Note: This page shows you how to use LISTS as ARRAYS, however, to work with arrays in Python you will have to import a library, like the NumPy library.

Arrays are used to store multiple values in one single variable:

```python
cars = ["Ford", "Volvo", "BMW"]
```

Access the Elements of an Array

You refer to an array element by referring to the *index number*.

Example

Get the value of the first array item:

```python
x = cars[0]
```

Example

Modify the value of the first array item:

```python
cars[0] = "Toyota"
```

```python
x = len(cars)
```

Looping Array Elements

You can use the `for in` loop to loop through all the elements of an array.

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

Adding Array Elements

You can use the `append()` method to add an element to an array.

Example

Add one more element to the `cars` array:

```
cars.append("Honda")
```

Removing Array Elements

You can use the `pop()` method to remove an element from the array.

```
cars.pop(1)
```

You can also use the `remove()` method to remove an element from the array.

Note: The list's `remove()` method only removes the first occurrence of the specified value.

```
cars.remove("Volvo")
```

User Input

Python allows for user input.

That means we are able to ask the user for input.

The method is a bit different in Python 3.6 than Python 2.7.

Python 3.6 uses the `input()` method.

Python 2.7 uses the `raw_input()` method.

```python
username = input("Enter username:")
print("Username is: " + username)
```

```python
username = raw_input("Enter username:")
print("Username is: " + username)
```

To make sure a string will display as expected, we can format the result with the `format()` method.

String format()

The `format()` method allows you to format selected parts of a string.

Sometimes there are parts of a text that you do not control, maybe they come from a database, or user input?

To control such values, add placeholders (curly brackets `{}`) in the text, and run the values through the `format()` method:

Add a placeholder where you want to display the price:

```python
price = 49
txt = "The price is {} dollars"
print(txt.format(price))
```

Format the price to be displayed as a number with two decimals:

```python
txt = "The price is {:.2f} dollars"
```

If you want to use more values, just add more values to the format() method:

```python
print(txt.format(price, itemno, count))
```

And add more placeholders:

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {} pieces of item number {} for {:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

You can use index numbers (a number inside the curly brackets {0}) to be sure the values are placed in the correct placeholders:

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {0} pieces of item number {1} for {2:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

Also, if you want to refer to the same value more than once, use the index number:

```
age = 36
name = "John"
txt = "His name is {1}. {1} is {0} years old."
print(txt.format(age, name))
```

You can also use named indexes by entering a name inside the curly brackets {carname} , but then you must use names when you pass the parameter values txt.format(carname = "Ford") :

```
myorder = "I have a {carname}, it is a {model}."
print(myorder.format(carname = "Ford", model = "Mustang"))
```

Array Methods

Python has a set of built-in methods that you can use on lists/arrays.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

#Normal Logic to reverse a given number in python

orignum = int(input("Enter the number: "))

n1 = orignum

revnum = 0

sum = 0

print("Input Number by the user = ", orignum)

while(orignum>0):

mod1 = orignum % 10

revnum = (revnum * 10) + mod1

orignum = (orignum // 10)

print("Orignum = ", orignum)

print("Input Number by the user = ", orignum)

print("Reversed Number = ", revnum)

5 NUMBER SUMMARY

```python
import numpy as np
from numpy import percentile
from numpy.random import rand


str1 = input('Enter elements of a list separated by space ')


print("\n")


str2 = str1. split()


# print list
print('list: ', str2)


for i in range(len(str2)):
    #convert each item into an int type
    str2[i] = int(str2[i])


arr1 = np.array(str2)


print("Arr1 elements: ", end = ' ')
print(arr1)


#Find the quartiles (25th, 50th, and 75th percentiles) of the array
quartiles = percentile(arr1, [25, 50, 75])


data_min, data_max = arr1.min(), arr1.max()


print('   5 number summary      ', end = ' ')


print('*********************************************')


print('Minimum = ',data_min)
print('Q1:%.3f ', quartiles[0])
print('Median: %.3f', quartiles[1])
print('Q1:%.3f ', quartiles[2])
print('Q1:%.3f ', data_max)
```

```python
print('***********************************************')
```