

ST. PAUL'S SR. SEC. SCHOOL.



SPECIMEN

ABU ROAD

20XX-XX
PROJECT FILE

Bank Management

Submitted to

Mr. Dharendra P.S. Chauhan
(Head of Computer Dept.)

Submitted by

Student Name
Class:-12th XX

.

SPECIMEN

© Can Stock Photo - csp18229338



CERTIFICATE

This Project is certify the project entitled Informatics Practiced done by “**Student Name** “ during the session 20XX-XX under my supervision for 12th. C.B.S.E. examination is satisfactory and approved for submission.

Mrs. Sapna Singh Chauhan
PRINCIPAL

Mr. Dharendra P.S. Chauhan
(Head of Computer Dept.)



ACKNOWLEDGMENT

I take this opportunity to express my profound indebtedness to my guide **Mr. Dhirendra P.S. Chauhan** whose keen interest, Valuable guidance and useful suggestion has helped me a lot throughout present studies and in making this project a success.

I am extremely grateful to our Honorable Principal **Mrs. Sapna Singh Chauhan**

Who provided us facilities in completion of this project “**Bank Management**”

Student Name

()
XIIth XXX



© Can Stock Photo - csp18229338

Python

Python was originally conceptualized by Guido van Rossum in the late 1980s as a member of the National Research Institute of Mathematics and Computer Science. Initially, it was designed as a response to the ABC programming language that was also foregrounded in the Netherlands. Among the main features of Python compared to the ABC language was that Python had exception handling and was targeted for the Amoeba operating system (go Python!).

Python History and Versions

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in the December 1989 by **Guido Van Rossum** at CWI in Netherland.
- In February 1991, van Rossum published the code (labeled version 0.9.0) to alt.sources.
- In 1994, Python 1.0 was released with new features like: lambda, map, filter, and reduce.
- Python 2.0 added new features like: list comprehensions, garbage collection system.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify fundamental flaw of the language.
- *ABC programming language* is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
- Python is influenced by following programming languages:
 - ABC language.
 - Modula-3



Python Variables

Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.

Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.

It is recommended to use lowercase letters for variable name. Rahul and rahul both are two different variables.

Identifier Naming

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

- The first character of the variable must be an alphabet or underscore (_).
- All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore or digit (0-9).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
- Identifier name must not be similar to any keyword defined in the language.
- Identifier names are case sensitive for example my name, and MyName is not the same.
- Examples of valid identifiers : a123, _n, n_9, etc.
- Examples of invalid identifiers: 1a, n%4, n 9, etc.

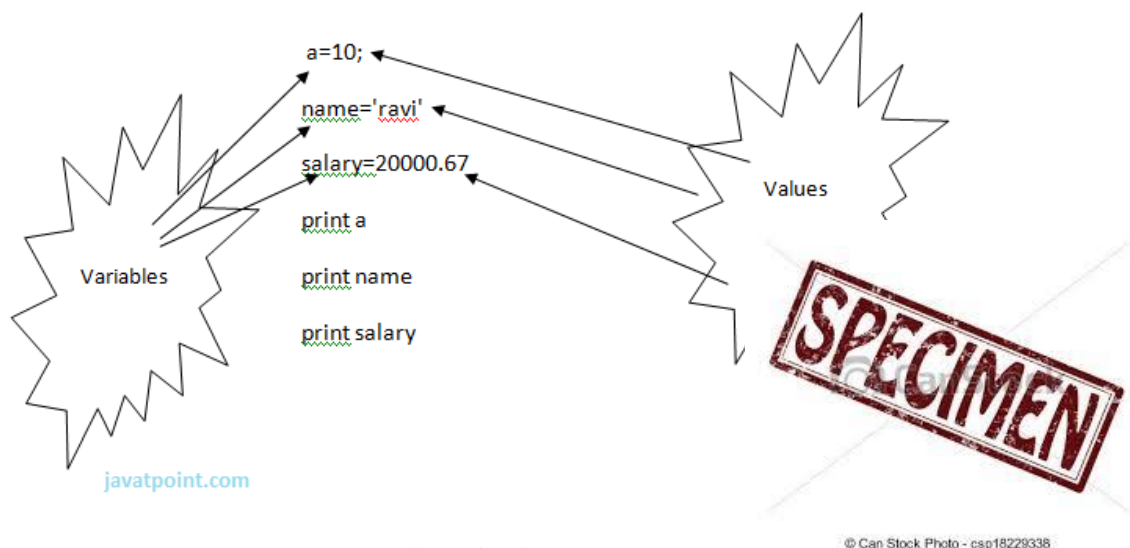
Declaring Variable and Assigning Values

Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

Eg:



Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignment.

Specimen Copy

We can apply multiple assignments in two ways either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Lets see given examples.

1. Assigning single value to multiple variables

Eg:

```
x=y=z=50
print x
print y
print z
```

Output:

```
>>>
50
50
50
```

2. Assigning multiple values to multiple variables:

Eg:

```
a,b,c=5,10,15
print a
print b
print c
```

Output:

```
1. >>>
2. 5
3. 10
4. 15
5. >>>
```

The values will be assigned in the order in which variables appears.

Basic Fundamentals:

This section contains the basic fundamentals of Python like :

i) Tokens and their types.

ii) Comments

a) Tokens:

- Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement.
- Token is the smallest unit inside the given program.

There are following tokens in Python:

- Keywords.
- Identifiers.
- Literals.
- Operators.



Python Data Types

Variables can hold values of different data types. Python is a dynamically typed language hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

Python enables us to check the type of the variable used in the program. Python provides us the **type()** function which returns the type of the variable passed.

Consider the following example to define the values of different data types and checking its type.

```
A=10
b="Hi Python"
c = 10.5
print(type(a));
print(type(b));
print(type(c));
```

Output:

```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types

A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary

In this section of the tutorial, we will give a brief introduction of the above data types. We will discuss each one of them in detail later in this tutorial.

Numbers

Number stores numeric values. Python creates Number objects when a number is assigned to a variable. For example;

```
a = 3 , b = 5 #a and b are number objects
```

Python supports 4 types of numeric data.

1. int (signed integers like 10, 2, 29, etc.)
2. long (long integers used for a higher range of values like 908090800L, -0x1929292L, etc.)
3. float (float is used to store floating point numbers like 1.9, 9.902, 15.2, etc.)
4. complex (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

Python allows us to use a lower-case L to be used with long integers. However, we must always use an upper-case L to avoid confusion.

A complex number contains an ordered pair, i.e., $x + iy$ where x and y denote the real and imaginary parts respectively).

String

The string can be defined as the sequence of characters represented in the quotation marks. In python, we can use single, double, or triple quotes to define a string.

Specimen Copy

String handling in python is a straightforward task since there are various inbuilt functions and operators provided.

In the case of string handling, the operator + is used to concatenate two strings as the operation "hello"+"python" returns "hello python".

The operator * is known as repetition operator as the operation "Python " *2 returns "Python Python ".

The following example illustrates the string handling in python.

```
str1 = 'hello javatpoint' #string str1
str2 = ' how are you' #string str2
print (str1[0:2]) #printing first two character using slice operator
print (str1[4]) #printing 4th character of the string
print (str1*2) #printing the string twice
print (str1 + str2) #printing the concatenation of str1 and str2
```

Output:

```
he
o
hello javatpointhello javatpoint
hello javatpoint how are you
```

List

Lists are similar to arrays in C. However; the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

Consider the following example.

```
l = [1, "hi", "python", 2]
print (l[3:]);
print (l[0:2]);
print (l);
print (l + l);
print (l * 3);
```

Output:

```
[2]
[1, 'hi']
[1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
```



Tuple

A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

Let's see a simple example of the tuple.

```
t = ("hi", "python", 2)
```

```
print (t[1:]);
```

```
print (t[0:1]);
```

```
print (t);
```

```
print (t + t);
```

```
print (t * 3);
```

```
print (type(t))
```

```
t[2] = "hi";
```

Output:

```
('python', 2)
('hi',)
('hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2)
('hi', 'python', 2, 'hi', 'python', 2, 'hi', 'python', 2)
<type 'tuple'>
```

Traceback (most recent call last):

File "main.py", line 8, in <module>

```
t[2] = "hi";
```

TypeError: 'tuple' object does not support item assignment



Dictionary

Dictionary is an ordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type whereas value is an arbitrary Python object.

The items in the dictionary are separated with the comma and enclosed in the curly braces {}.

Consider the following example.

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'};
print("1st name is "+d[1]);
print("2nd name is "+ d[4]);
print (d);
print (d.keys());
print (d.values());
```

Output:

```
1st name is Jimmy
2nd name is mike
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
[1, 2, 3, 4]
['Jimmy', 'Alex', 'john', 'mike']
```



Python If-else statements

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

In python, decision making is performed by the following statements.

Indentation in Python

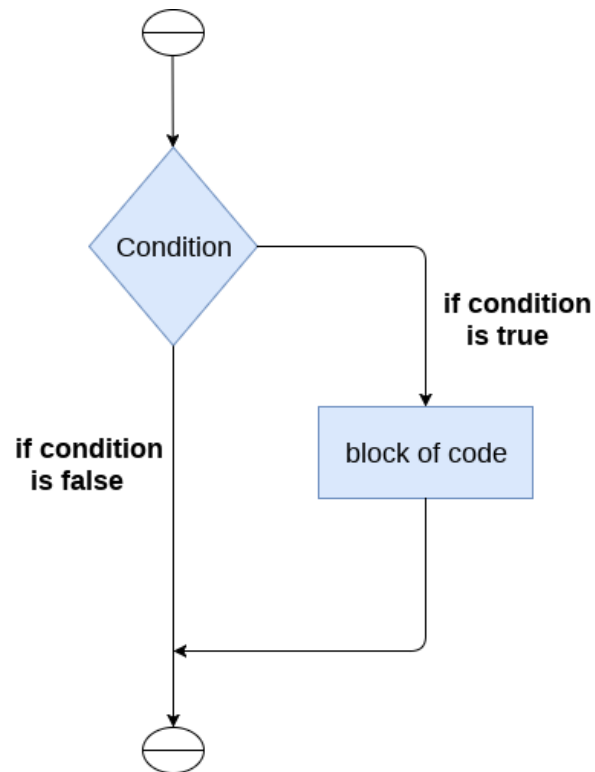
For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block. If two statements are at the same indentation level, then they are the part of the same block.

Generally, four spaces are given to indent the statements which are a typical amount of indentation in python.

Indentation is the most used part of the python language since it declares the block of code. All the statements of one block are intended at the same level indentation. We will see how the actual indentation takes place in decision making and other stuff in python.

The if statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.



The syntax of the if-statement is given below.

if expression:
statement

Example 1

```
num = int(input("enter the number?"))
```

```
if num%2 == 0:  
    print("Number is even")
```

Output:

```
enter the number?10  
Number is even
```



© Can Stock Photo - csp18229338

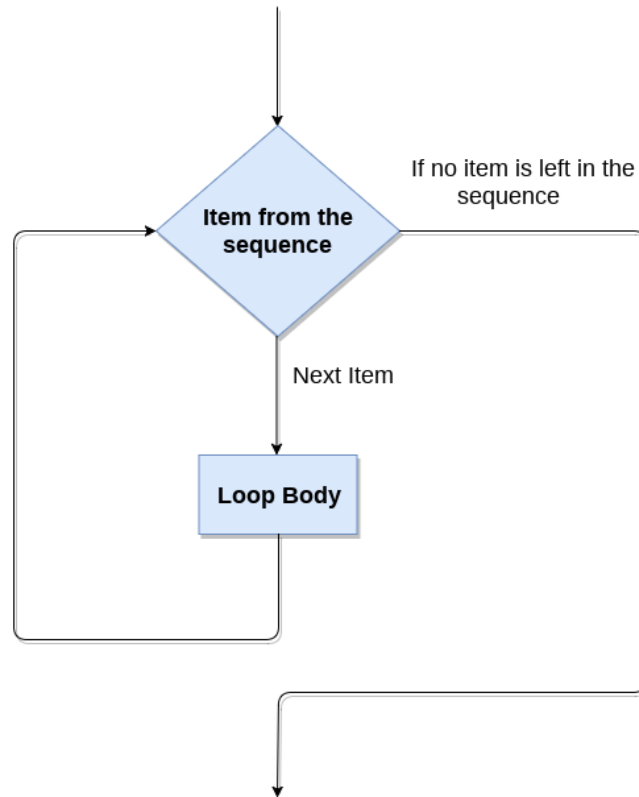
Python for loop

The **for loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

The syntax of for loop in python is given below.

for iterating_var **in** sequence:

statement(s)

*Example*

i=1

```
n=int(input("Enter the number up to which you want to print the natural numbers: "))
```

```
for i in range(0,10):
```

```
    print(i,end = ' ')
```

Python while loop

The while loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed as long as the given condition is true.

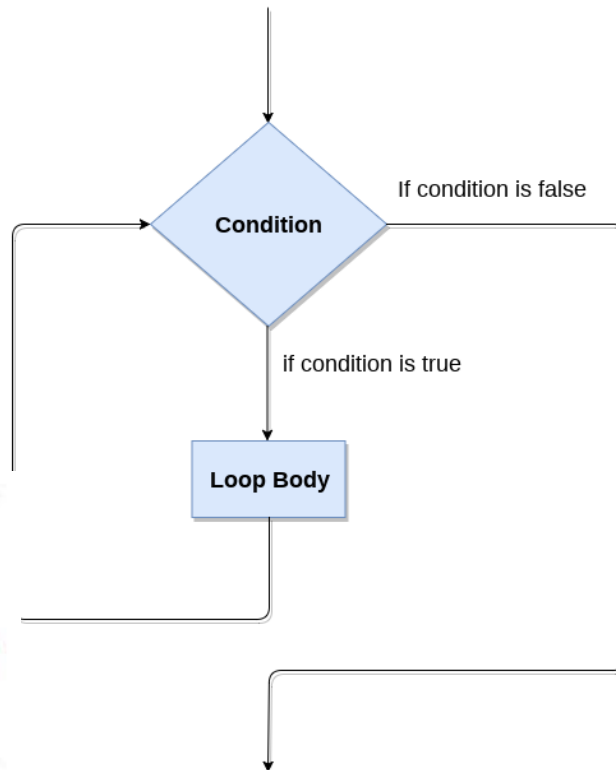
It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

The syntax is given below.

while expression:

statements

Here, the statements can be a single statement or the group of statements. The expression should be any valid python expression resulting into true or false. The true is any non-zero value.



SPECIMEN

© Can Stock Photo - csp18229338

Python break statement

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

The break is commonly used in the cases where we need to break the loop for a given condition.

The syntax of the break is given below.

1. #loop statements
2. **break;**

Python continue Statement

The continue statement in python is used to bring the program control to the beginning of the loop. The continue statement skips the remaining lines of code inside the loop and start with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

The syntax of Python continue statement is given below.

```
#loop statements
```

```
continue;
```

```
#the code to be skipped
```

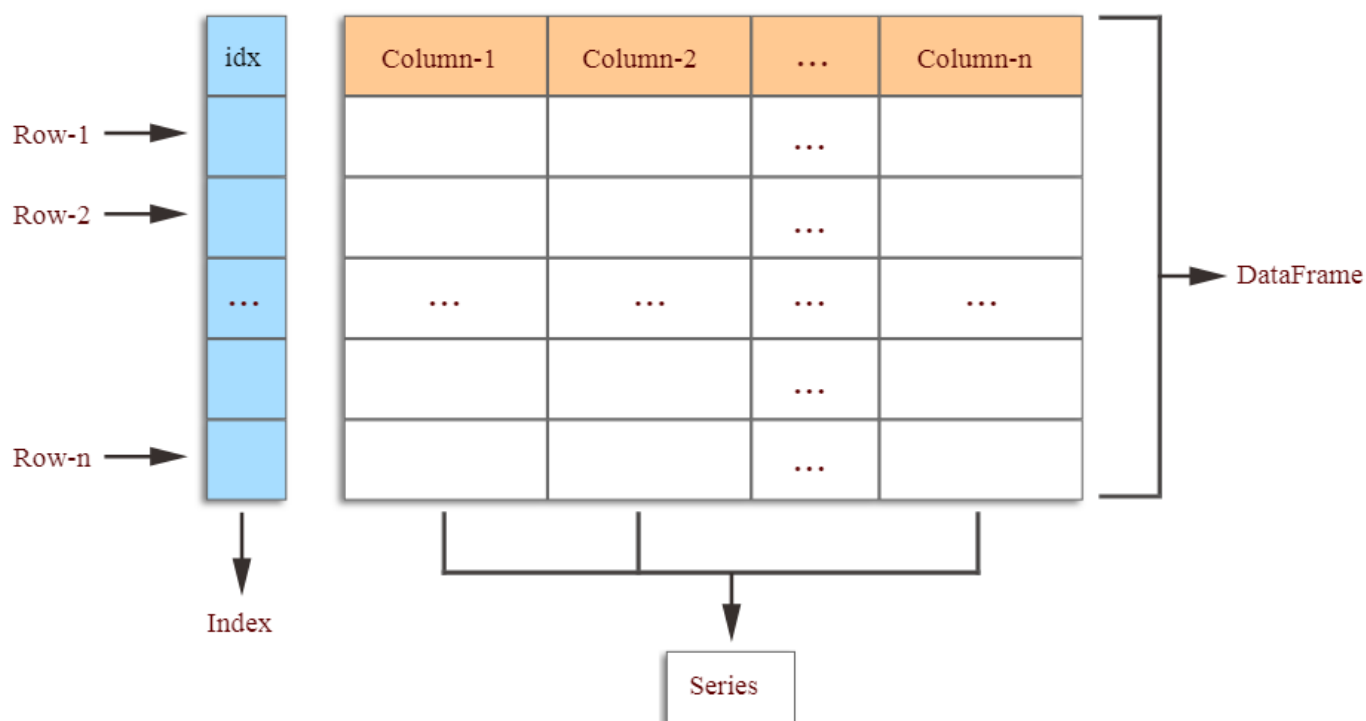


© Can Stock Photo - csp18229338

What is Pandas?

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with 'relation' or 'labeled' data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

Pandas Data structure



© w3resource.c

pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data with row and column labels
- Any other form of observational / statistical data sets.

Pandas has following data structures:

1. DataFrame
2. Series
3. ndarray

Pandas DataFrame

Pandas DataFrame is a 2-dimensional labeled data structure with columns of different types. It can be thought of as a matrix but having data with different types.

Characteristics:

1. Multiple rows and columns of data
2. Each row represents a sample of data
3. Each row represents a variable/dimension of dataset
4. Usually, all the data in a column is of the same type.

Pandas First Steps

Install and import

Pandas is an easy package to install. Open up your terminal command line (for PC users) and install it using either of the following commands:

```
pip install pandas
```

How to Imports Pandas

```
import pandas as pd
```

Create Dataframe:

```
import pandas as pd
```

```
df = pd.DataFrame({'X':[78,85,96,80,86],  
'Y':[84,94,89,83,86], 'Z':[86,97,96,72,83]});
```

```
print(df)
```

Sample Output:

	X	Y	Z
0	78	84	86
1	85	94	97
2	96	89	96
3	80	83	72
4	86	86	83



© Can Stock Photo - csp18229338

Create DataFrame:

```
import pandas as pd

s = pd.Series([2, 4, 6, 8, 10])

print(s)
```

Copy

Sample Output:

```
0    2
1    4
2    6
3    8
4   10
```

**Core components of pandas: Series and DataFrames**

The primary two components of pandas are the `Series` and `DataFrame`.

A `Series` is essentially a column, and a `DataFrame` is a multi-dimensional table made up of a collection of `Series`.

Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Creating DataFrames

Creating DataFrames right in Python is good to know and quite useful when testing new methods and functions you find in the pandas docs.

There are *many* ways to create a DataFrame from scratch, but a great option is to just use a simple `dict`.

```
data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2]
}

df = pd.DataFrame(data)

print(df)
```

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Set the index no

The **Index** of this DataFrame was given to us on creation as the n
our own when we initialize the DataFrame.

```
df = pd.DataFrame(data, index=['June', 'Robert', 'Lily', 'David'],
print(df)
```

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

Python Pandas - Series

Specimen Copy

Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.

pandas.Series

A pandas Series can be created using the following constructor –

```
pandas.Series( data, index, dtype, copy)
```

The parameters of the constructor are as follows –

Sr.No	Parameter & Description
1	data data takes various forms like ndarray, list, constants
2	index Index values must be unique and hashable, same length as data. Default np.arange(n) if no index is passed.
3	dtype dtype is for data type. If None, data type will be inferred
4	copy Copy data. Default False

A series can be created using various inputs like –

- Array
- Dict
- Scalar value or constant

Create an Empty Series

A basic series, which can be created is an Empty Series.

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
s = pd.Series()
print s
```

Its output is as follows –

SPECIMEN

© Can Stock Photo - csp18229338

```
Series([], dtype: float64)
```

Create a Series from ndarray

If data is an ndarray, then index passed must be of the same length. If no index is passed, then by default index will be **range(n)** where **n** is array length, i.e., [0,1,2,3.... **range(len(array))-1**].

Example 1

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print s
```

Its output is as follows –

```
0    a
1    b
2    c
3    d
dtype: object
```

We did not pass any index, so by default, it assigned the indexes ranging from 0 to **len(data)-1**, i.e., 0 to 3.

Example 2

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
print s
```

Its output is as follows –

```
100    a
101    b
102    c
103    d
dtype: object
```

We passed the index values here. Now we can see the customized indexed values in the output.

Create a Series from dict

A **dict** can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index. If **index** is passed, the values in data corresponding to the labels in the index will be pulled out.

Example 1

```
#import the pandas library and aliasing as pd
import pandas as pd
```

```
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print s
```

Its output is as follows –

```
a 0.0
b 1.0
c 2.0
dtype: float64
```

Observe – Dictionary keys are used to construct index.

Example 2

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data, index=['b', 'c', 'd', 'a'])
print s
```

Its output is as follows –

```
b 1.0
c 2.0
d NaN
a 0.0
dtype: float64
```

Observe – Index order is persisted and the missing element is filled with NaN (Not a Number).

Create a Series from Scalar

If data is a scalar value, an index must be provided. The value will be repeated to match the length of **index**

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
s = pd.Series(5, index=[0, 1, 2, 3])
print s
```

Its output is as follows –

```
0    5
1    5
2    5
3    5
dtype: int64
```

Accessing Data from Series with Position

Data in the series can be accessed similar to that in an **ndarray**.



© Can Stock Photo - csp18229338

Example 1

Retrieve the first element. As we already know, the counting starts from zero for the array, which means the first element is stored at zeroth position and so on.

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first element
print s[0]
```

Its **output** is as follows –

1

Example 2

Retrieve the first three elements in the Series. If a : is inserted in front of it, all items from that index onwards will be extracted. If two parameters (with : between them) is used, items between the two indexes (not including the stop index)

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first three element
print s[:3]
```

Its **output** is as follows –

```
a    1
b    2
c    3
dtype: int64
```

Example 3

Retrieve the last three elements.

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the last three element
print s[-3:]
```

Its **output** is as follows –

```
c    3
d    4
e    5
dtype: int64
```

Retrieve Data Using Label (Index)

A Series is like a fixed-size **dict** in that you can get and set values by index label.

Example 1

Retrieve a single element using index label value.

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve a single element
print s['a']
```

Its output is as follows –

1

Example 2

Retrieve multiple elements using a list of index label values.

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve multiple elements
print s[['a','c','d']]
```

Its output is as follows –

```
a    1
c    3
d    4
dtype: int64
```



© Can Stock Photo - csp18229338

Example 3

If a label is not contained, an exception is raised.

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve multiple elements
print s['f']
```

Its output is as follows –

```
...
KeyError: 'f'
```

Introduction to Data Structures

Pandas deals with the following three data structures –

- Series
- DataFrame
- Panel

These data structures are built on top of Numpy array, which means they are fast.

Dimension & Description

The best way to think of these data structures is that the higher dimensional data structure is a container of its lower dimensional data structure. For example, DataFrame is a container of Series, Panel is a container of DataFrame.

Data Structure	Dimensions	Description
Series	1	1D labeled homogeneous array, size immutable.
Data Frames	2	General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.
Panel	3	General 3D labeled, size-mutable array.

Building and handling two or more dimensional arrays is a tedious task, burden is placed on the user to consider the orientation of the data set when writing functions. But using Pandas data structures, the mental effort of the user is reduced.

For example, with tabular data (DataFrame) it is more semantically helpful to think of the **index** (the rows) and the **columns** rather than axis 0 and axis 1.

Mutability

All Pandas data structures are value mutable (can be changed) and except Series all are size mutable. Series is size immutable.

Note – DataFrame is widely used and one of the most important data structures. Panel is used much less.

Series

Series is a one-dimensional array like structure with homogeneous data. For example, the following series is a collection of integers 10, 23, 56, ...

10	23	56	17	52	61	73	90	26	72
----	----	----	----	----	----	----	----	----	----

Key Points

- Homogeneous data
- Size Immutable
- Values of Data Mutable



DataFrame

DataFrame is a two-dimensional array with heterogeneous data. For example,

Name	Age	Gender	Rating
Steve	32	Male	3.45
Lia	28	Female	4.6
Vin	45	Male	3.9
Katie	38	Female	2.78

The table represents the data of a sales team of an organization with their overall performance rating. The data is represented in rows and columns. Each column represents an attribute and each row represents a person.

Data Type of Columns

The data types of the four columns are as follows –

Column	Type
Name	String
Age	Integer
Gender	String
Rating	Float

Key Points

- Heterogeneous data
- Size Mutable

- Data Mutable

Panel

Panel is a three-dimensional data structure with heterogeneous data. It is hard to represent the panel in graphical representation. But a panel can be illustrated as a container of DataFrame.

Key Points

- Heterogeneous data
- Size Mutable
- Data Mutable



Python Pandas - Basic Functionality

By now, we learnt about the three Pandas DataStructures and how to create them. We will majorly focus on the DataFrame objects because of its importance in the real time data processing and also discuss a few other DataStructures.

Series Basic Functionality

Sr.No.	Attribute or Method & Description
1	axes Returns a list of the row axis labels
2	dtype Returns the dtype of the object.
3	empty Returns True if series is empty.

4	ndim Returns the number of dimensions of the underlying data, by definition 1.
5	size Returns the number of elements in the underlying data.
6	values Returns the Series as ndarray.
7	head() Returns the first n rows.
8	tail() Returns the last n rows.

Let us now create a Series and see all the above tabulated attributes operation.

Example

```
import pandas as pd
import numpy as np

#Create a series with 100 random numbers
s = pd.Series(np.random.randn(4))
print s
```

Its **output** is as follows –

```
0    0.967853
1   -0.148368
2   -1.395906
3   -1.758394
dtype: float64
```

axes

Returns the list of the labels of the series.

```
import pandas as pd
import numpy as np

#Create a series with 100 random numbers
s = pd.Series(np.random.randn(4))
print ("The axes are:")
print s.axes
```

Its output is as follows –

```
The axes are:
[RangeIndex(start=0, stop=4, step=1)]
```

The above result is a compact format of a list of values from 0 to 5, i.e., [0,1,2,3,4].

empty

Returns the Boolean value saying whether the Object is empty or not. True indicates that the object is empty.

```
import pandas as pd
import numpy as np

#Create a series with 100 random numbers
s = pd.Series(np.random.randn(4))
print ("Is the Object empty?")
print s.empty
```

Its output is as follows –

```
Is the Object empty?
False
```

ndim

Returns the number of dimensions of the object. By definition, a Series is a 1D data structure, so it returns

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print s

print ("The dimensions of the object:")
print s.ndim
```

Its output is as follows –

```
0    0.175898
1    0.166197
```



© Can Stock Photo - csp18229338

Specimen Copy

```
2 -0.609712
3 -1.377000
dtype: float64
```

The dimensions of the object:
1

size

Returns the size(length) of the series.

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(2))
print s
print ("The size of the object:")
print s.size
```

Its output is as follows –

```
0 3.078058
1 -1.207803
dtype: float64
```

The size of the object:
2

values

Returns the actual data in the series as an array.

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print s

print ("The actual data series is:")
print s.values
```

Its output is as follows –

```
0 1.787373
1 -0.605159
2 0.180477
3 -0.140922
dtype: float64
```

The actual data series is:
[1.78737302 -0.60515881 0.18047664 -0.1409218]



© Can Stock Photo - csp18229338

Head & Tail

To view a small sample of a Series or the DataFrame object, use the head() and the tail() methods.

head() returns the first **n** rows (observe the index values). The default number of elements to display is five, but you may pass a custom number.

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print ("The original series is:")
print s

print ("The first two rows of the data series:")
print s.head(2)
```

Its output is as follows –

The original series is:

```
0    0.720876
1   -0.765898
2    0.479221
3   -0.139547
dtype: float64
```

The first two rows of the data series:

```
0    0.720876
1   -0.765898
dtype: float64
```

tail() returns the last **n** rows (observe the index values). The default number of elements to display is five, but you may pass a custom number.

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print ("The original series is:")
print s

print ("The last two rows of the data series:")
print s.tail(2)
```

Its output is as follows –

The original series is:

```
0   -0.655091
1   -0.881407
2   -0.608592
3   -2.341413
dtype: float64
```

The last two rows of the data series:



© Can Stock Photo - csp18229338

```
2 -0.608592
3 -2.341413
dtype: float64
```

Python Pandas - Panel

A **panel** is a 3D container of data. The term **Panel data** is derived from econometrics and is partially responsible for the name pandas – **pan(el)-da(ta)-s**.

The names for the 3 axes are intended to give some semantic meaning to describing operations involving panel data. They are –

- **items** – axis 0, each item corresponds to a DataFrame contained inside.
- **major_axis** – axis 1, it is the index (rows) of each of the DataFrames.
- **minor_axis** – axis 2, it is the columns of each of the DataFrames.

pandas.Panel()

A Panel can be created using the following constructor –

```
pandas.Panel(data, items, major_axis, minor_axis, dtype, copy)
```

The parameters of the constructor are as follows –

Parameter	Description
data	Data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame
items	axis=0
major_axis	axis=1

SPECIMEN

© Can Stock Photo - csp18229338

minor_axis	axis=2
dtype	Data type of each column
copy	Copy data. Default, false

Create Panel

A Panel can be created using multiple ways like –

- From ndarrays
- From dict of DataFrames

From 3D ndarray

```
# creating an empty panel
import pandas as pd
import numpy as np

data = np.random.rand(2,4,5)
p = pd.Panel(data)
print p
```

Its output is as follows –

```
<class 'pandas.core.panel.Panel'>
```

```
Dimensions: 2 (items) x 4 (major_axis) x 5 (minor_axis)
Items axis: 0 to 1
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 4
```

Note – Observe the dimensions of the empty panel and the above panel, all the objects are different.

From dict of DataFrame Objects

```
#creating an empty panel
import pandas as pd
import numpy as np

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
        'Item2' : pd.DataFrame(np.random.randn(4, 2))}
```



```
p = pd.Panel(data)
print p
```

Its output is as follows –

```
Dimensions: 2 (items) x 4 (major_axis) x 3 (minor_axis)
Items axis: Item1 to Item2
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 2
```

Create an Empty Panel

An empty panel can be created using the Panel constructor as follows –

```
#creating an empty panel
import pandas as pd
p = pd.Panel()
print p
```

Its output is as follows –

```
<class 'pandas.core.panel.Panel'>
Dimensions: 0 (items) x 0 (major_axis) x 0 (minor_axis)
Items axis: None
Major_axis axis: None
Minor_axis axis: None
```

Python

Machine Learning



© Can Stock Photo - csp18229338

Machine Learning is making the computer learn from studying data and statistics. Machine Learning is a step into the direction of artificial intelligence (AI). Machine Learning is a program that analyses data and learns to predict the outcome.

Data Types

To analyze data, it is important to know what type of data we are dealing with. We can split the data types into three main categories:

- **Numerical**
- **Categorical**
- **Ordinal**

Numerical data are numbers, and can be split into two numerical categories:

- **Discrete Data**
 - numbers that are limited to integers. Example: The number of cars passing by.
- **Continuous Data**
 - numbers that are of infinite value. Example: The price of an item, or the size of an item

Categorical data are values that cannot be measured up against each other. Example: a color value, or any yes/no values.

Ordinal data are like categorical data, but can be measured up against each other. Example: school grades where A is better than B and so on.

Mean Median Mode

In Machine Learning (and in mathematics) there are often three values that interests us:

- **Mean** - The average value
- **Median** - The mid point value
- **Mode** - The most common value

Example: We have registered the speed of 13 cars:

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

What is the average, the middle, or the most common speed value?

Mean

The mean value is the average value.

To calculate the mean, find the sum of all values, and divide the sum by the number of values:

$$(99+86+87+88+111+86+103+87+94+78+77+85+86) / 13 = 89.77$$

```
import numpy

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

x = numpy.mean(speed)

print(x)
```



© Can Stock Photo - csp18229338

```
89.76923076923077
```

Median

The median value is the value in the middle, after you have sorted all the values:

77, 78, 85, 86, 86, 86, 87, 87, 88, 94, 99, 103, 111

It is important that the numbers are sorted before you can find the median.

```
import numpy

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

x = numpy.median(speed)

print(x)
```

```
87.0
```

If there are two numbers in the middle, divide the sum of those numbers by two.

77, 78, 85, 86, 86, 86, 87, 87, 94, 98, 99, 103

$$(86 + 87) / 2 = 86.5$$

Example

Using the NumPy module:

```
import numpy

speed = [99,86,87,88,86,103,87,94,78,77,85,86]

x = numpy.median(speed)

print(x)
```

```
86.5
```



© Can Stock Photo - csp18229338

Mode

The Mode value is the value that appears the most number of times:

99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86 = 86

The SciPy module has a method for this:

Example

Use the SciPy `mode()` method to find the number that appears the most:

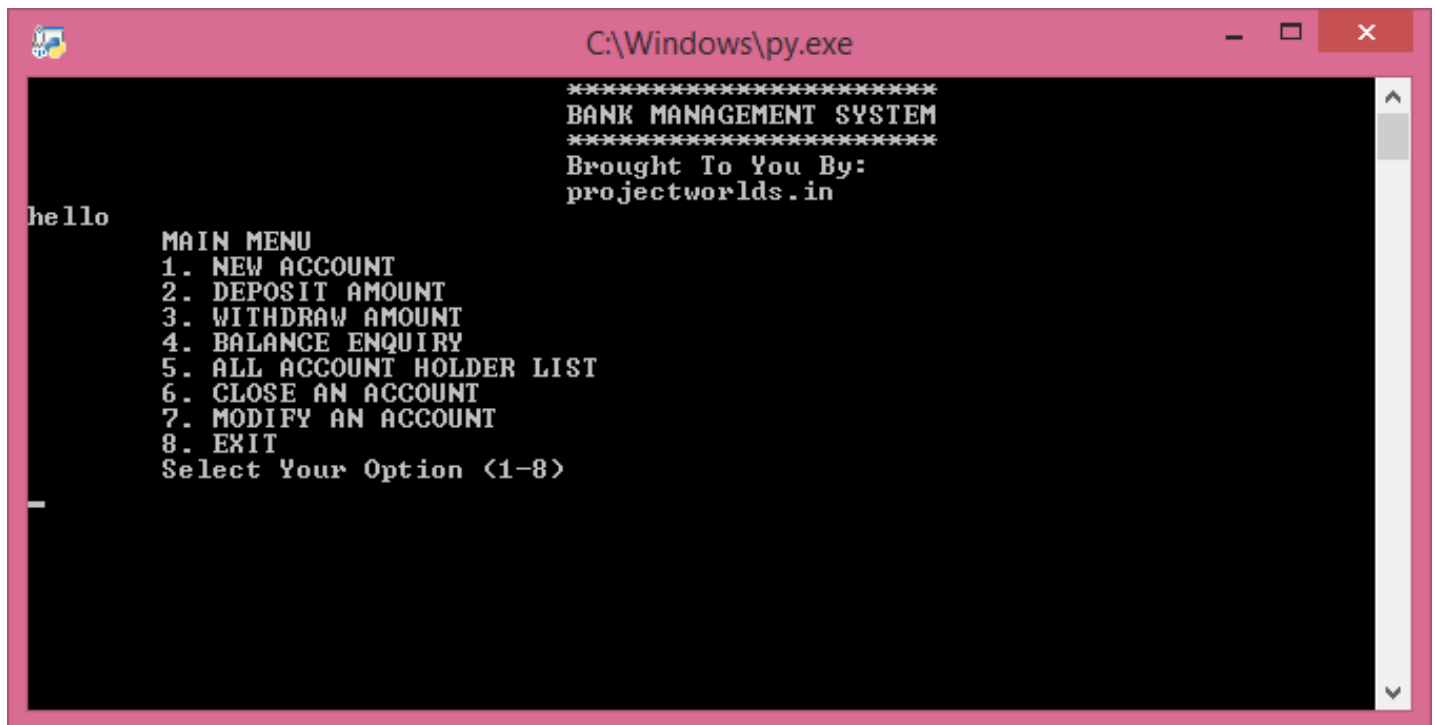
```
from scipy import stats  
  
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]  
  
x = stats.mode(speed)  
  
print(x)
```

```
ModeResult(mode=array([86]), count=array([3]))
```



© Can Stock Photo - csp18229338

Bank Management



```

C:\Windows\py.exe
*****
BANK MANAGEMENT SYSTEM
*****
Brought To You By:
projectworlds.in

hello
MAIN MENU
1. NEW ACCOUNT
2. DEPOSIT AMOUNT
3. WITHDRAW AMOUNT
4. BALANCE ENQUIRY
5. ALL ACCOUNT HOLDER LIST
6. CLOSE AN ACCOUNT
7. MODIFY AN ACCOUNT
8. EXIT
Select Your Option <1-8>

```

MODULES USED IN PROJECT

```
import pickle
```

```
import os
```



© Can Stock Photo - csp18229338

CLASS USED IN PROJECT

```
class account(object):

    def __init__(s):

        s.acno=0

        s.name=""

        s.deposit=0

        s.type=""

    def create_account(s): #function to get data from user

        name=raw_input("\n\nEnter the name of the account holder: ")

        s.name=name.capitalize()

        type=raw_input("\n\nEnter type of the account (C/S): ")

        s.type=type.upper()

        s.deposit=input("\n\nEnter initial amount\n(>=500 for Saving and >=1000 for Current): ")

    def show_account(s): #function to show data on screen

        print "\nAccount No. :", s.acno

        print "\nAccount holder name: ", s.name

        print "\nType of account", s.type

        print "\nBalance amount: ", s.deposit

    def modify(s):      #function to get new data from user

        print "\nAccount No. : ", s.acno

        s.name=raw_input("\n\nEnter the name of account holder: ")

        type=raw_input("\n\nEnter type of account (C/S): ")

        s.type=type.upper()

        s.deposit=input("\n\nEnter the amount: ")

    def dep(s,x):      #function to accept amount and add to balance

        s.deposit+=x

    def draw(s,x):      #function to accept amount and subtract from balance amount
```



© Can Stock Photo - csp18229338

Specimen Copy

s.deposit==x

def report(s): #function to show data in tabular format

print "%-10s"%s.acno,"%-20s"%s.name,"%-10s"%s.type,"%-6s"%s.deposit

def retacno(s): #function to return account number

return s.acno

def retdeposit(s): #function to return balance amount

return s.deposit

def rettype(s): #function to return type of account

return s.type



© Can Stock Photo - csp18229338

FUNCTION TO GENERATE ACCOUNT NUMBER

```
C:\Python27\python.exe
Enter Your Choice(1~8): 1

Enter the name of the account holder: Younus
Enter type of the account (C/S): s
Enter initial amount
(<=5000 for Saving and >=10000 for Current): 6000

Account Created Successfully

YOUR ACCOUNT NUMBER IS: 1001

=====
MAIN MENU
1. New Account
2. Deposit Amount
3. Withdraw Amount
4. Balance Enquiry
5. All Account Holder List
6. Close An Account
7. Modify An Account
8. Exit
```

def gen_acno():

try:

5/12/2020

Specimen Copy

Specimen Copy

```
inFile=open("account2.dat","rb")

outFile=open("text2.dat","wb")

n=inFile.read()

n=int(n)

while True:

    n+=1

    outFile.write(str(n))

inFile.close()

outFile.close()

os.remove("account2.dat")

os.rename("text2.dat","account2.dat")

yield n

    except IOError:

print "I/O error occured"
```



FUNCTION TO WRITE RECORD IN BINARY FILE

```
def write_account():  
    try:  
        ac=account()  
        outFile=open("account.dat","ab")  
        ch=gen_acno()  
        ac.acno=ch.next()  
        ac.create_account()  
        pickle.dump(ac,outFile)  
        outFile.close()  
        print "\n\n Account Created Successfully"  
        print "\n\n YOUR ACCOUNT NUMBER IS: ",ac.retacno()  
    except:  
        pass
```



© Can Stock Photo - csp18229338

FUNCTION TO DISPLAY ACCOUNT DETAILS GIVEN BY USER

```
def display_sp(n):  
    flag=0  
    try:  
        inFile=open("account.dat","rb")  
        print "\nBALANCE DETAILS\n"  
        while True:  
            ac=pickle.load(inFile)  
  
            if ac.retacno()==n:  
                ac.show_account()  
                flag=1  
  
        except EOFError:  
            inFile.close  
            if flag==0:  
                print "\n\nAccount number not exist"  
  
        except IOError:  
            print "File could not be open !! Press any Key..."
```



© Can Stock Photo - csp18229338

FUNCTION TO MODIFY RECORD OF FILE

```
def modify_account(n):  
    found=0  
  
    try:  
        inFile=open("account.dat","rb")  
        outFile=open("temp.dat","wb")  
  
        while True:  
            ac=pickle.load(inFile)  
  
            if ac.retacno()==n:  
                print 30*"-"  
                ac.show_account()  
                print 30*"-"  
                print "\n\nEnter The New Details of Account"  
                ac.modify()  
                pickle.dump(ac,outFile)  
                print "\n\n\tRecord Updated"  
                found=1  
  
            else:  
                pickle.dump(ac,outFile)  
  
        except EOFError:  
            inFile.close()  
            outFile.close()  
  
        if found==0:  
            print "\n\nRecord Not Found "  
  
    except IOError:  
        print "File could not be open !! Press any Key..."
```



```
os.remove("account.dat")
```

```
os.rename("temp.dat","account.dat")
```

```
*****
```

FUNCTION TO DELETE RECORD OF FILE

```
*****
```

```
def delete_account(n):
```

```
    found=0
```

```
    try:
```

```
        inFile=open("account.dat","rb")
```

```
        outFile=open("temp.dat","wb")
```

```
        while True:
```

```
            ac=pickle.load(inFile)
```

```
            if ac.retacno()==n:
```

```
                found=1
```

```
                print "\n\nRecord Deleted .."
```

```
            else:
```

```
                pickle.dump(ac,outFile)
```

```
    except EOFError:
```

```
        inFile.close()
```

```
        outFile.close()
```

```
        if found==0:
```

```
            print "\n\nRecord Not Found"
```

```
    except IOError:
```

```
        print "File could not be open !! Press any Key..."
```

```
os.remove("account.dat")
```



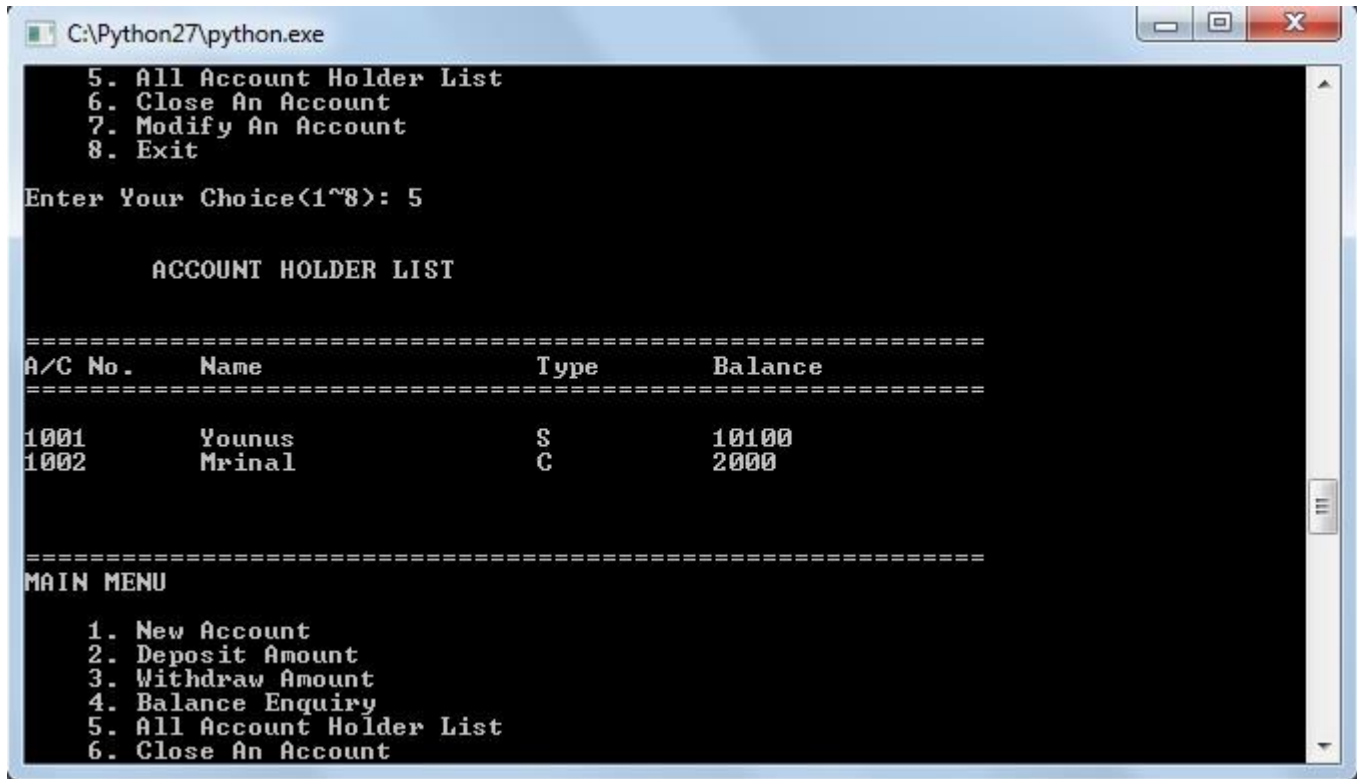
© Can Stock Photo - csp18229338

```
os.rename("temp.dat","account.dat")
```

```
*****
```

FUNCTION TO DISPLAY ALL ACCOUNT DETAILS

```
*****
```



```
C:\Python27\python.exe
5. All Account Holder List
6. Close An Account
7. Modify An Account
8. Exit
Enter Your Choice(1~8): 5

ACCOUNT HOLDER LIST

=====
A/C No.      Name                Type      Balance
=====
1001         Younus                S          10100
1002         Mrinal                C          2000
=====

MAIN MENU

1. New Account
2. Deposit Amount
3. Withdraw Amount
4. Balance Enquiry
5. All Account Holder List
6. Close An Account
```

```
def display_all():
```

```
    print "\n\n\tACCOUNT HOLDER LIST\n\n"
```

```
    print 60*"="
```

```
    print "%-10s"% "A/C No.", "%-20s"% "Name", "%-10s"% "Type", "%-6s"% "Balance"
```

```
    print 60*"=", "\n"
```

```
    try:
```

```
        inFile=open("account.dat","rb")
```

```
        while True:
```

```
            ac=pickle.load(inFile)
```

```
            ac.report()
```

```
            except EOFError:
```

```
        inFile.close()
```



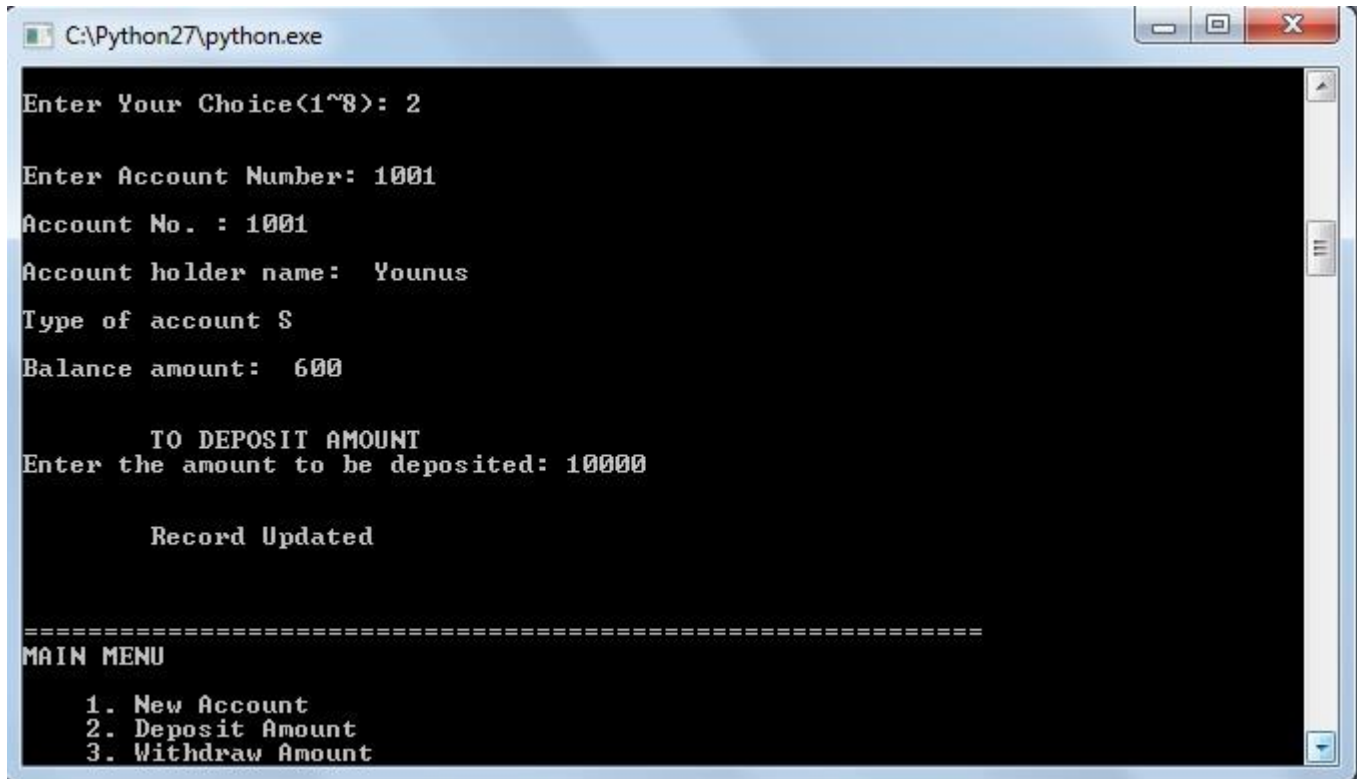
© Can Stock Photo - csp18229338

Specimen Copy

except IOError:

print "File could not be open !! Press any Key..."

FUNCTION TO DEPOSIT/WITHDRAW AMOUNT FOR GIVEN ACCOUNT



The screenshot shows a Windows command prompt window titled "C:\Python27\python.exe". The application is a banking system. It prompts the user to "Enter Your Choice(1~8): 2". Then it asks for "Enter Account Number: 1001", "Account No. : 1001", "Account holder name: Younus", "Type of account \$", and "Balance amount: 600". It then displays "TO DEPOSIT AMOUNT" and prompts "Enter the amount to be deposited: 10000". After processing, it shows "Record Updated". Finally, it displays a "MAIN MENU" with three options: "1. New Account", "2. Deposit Amount", and "3. Withdraw Amount".

```
def deposit_withdraw(n,option):
```

```
    found=0
```

```
    try:
```

```
        inFile=open("account.dat","rb")
```

```
        outFile=open("temp.dat","wb")
```

```
        while True:
```

```
            ac=pickle.load(inFile)
```

```
            if ac.retacno()==n:
```

```
                ac.show_account()
```

```
                if option==1:
```



© Can Stock Photo - csp18229338

Specimen Copy

```
print "\n\n\tTO DEPOSIT AMOUNT"

amt=input("Enter the amount to be deposited: ")

ac.dep(amt)

elif option==2:

    print "\n\n\tTO WITHDRAW AMOUNT"

    amt=input("Enter amount to be withdraw: ")

    bal=ac.retdeposit()-amt

    if((bal<500 and ac.rettype()=="S")or(bal<1000 and ac.rettype()=="C")):

        print "Insufficient balance"

    else:

        ac.draw(amt)

    pickle.dump(ac,outFile)

    found=1

    print "\n\n\tRecord Updated"

else:

    pickle.dump(ac,outFile)

except EOFError:

    inFile.close()

    outFile.close()

    if found==0:

        print "\n\nRecord Not Found"

except IOError:

    print "File could not be open !! Press any Key..."

os.remove("account.dat")

os.rename("temp.dat","account.dat")
```



INTRODUCTORY FUNCTION

```
def intro():
```

```
    print "\n\n\tBANK"
```

```
    print "\n\tMANAGEMENT"
```

```
    print "\n\nMADE BY : Enter your name"
```

```
    print "\nSCHOOL : Enter your school name"
```



© Can Stock Photo - csp18229338

THE MAIN FUNCTION OF PROGRAM

```
C:\Python27\python.exe

      BANK
      MANAGEMENT

MADE BY : Enter your name
SCHOOL : Enter your school name

=====
MAIN MENU
1. New Account
2. Deposit Amount
3. Withdraw Amount
4. Balance Enquiry
5. All Account Holder List
6. Close An Account
7. Modify An Account
8. Exit

Enter Your Choice(1~8): _
```



```
intro()
```

```
while True:
```

```
    print 3*"\\n",60*"="
```

```
    print """"MAIN MENU
```

```
1. New Account
```

```
2. Deposit Amount
```

```
3. Withdraw Amount
```

```
4. Balance Enquiry
```

```
5. All Account Holder List
```

```
6. Close An Account
```

```
7. Modify An Account
```

```
8. Exit
```

```
""""
```

```
try:
```

```
    ch=input("Enter Your Choice(1~8): ")
```

```
    if ch==1:
```

```
        write_account()
```

```
    elif ch==2:
```

```
        num=input("\\n\\nEnter Account Number: ")
```

```
        deposit_withdraw(num,1)
```

```
    elif ch==3:
```

```
        num=input("\\n\\nEnter Account Number: ")
```

```
        deposit_withdraw(num,2)
```



elif ch==4:

num=input("\n\nEnter Account Number: ")

display_sp(num)

elif ch==5:

display_all()

elif ch==6:

num=input("\n\nEnter Account Number: ")

delete_account(num)

elif ch==7:

num=input("\n\nEnter Account Number: ")

modify_account(num)

elif ch==8:

break

else:

print "Input correchr choice...(1-8)"

except NameError:

print "Input correct choice...(1-8)"

raw_input("\n\n\n\nTHANK YOU\n\nPress any key to exit...")



© Can Stock Photo - csp18229338

END OF PROJECT

```
C:\Python27\python.exe

=====
MAIN MENU
1. New Account
2. Deposit Amount
3. Withdraw Amount
4. Balance Enquiry
5. All Account Holder List
6. Close An Account
7. Modify An Account
8. Exit

Enter Your Choice(1~8): 8

THANK YOU
Press any key to exit..._
```

SPECIMEN

© Can Stock Photo - csp18229338



© Can Stock Photo - csp18229338

Bibliography

Book: - Informatics Practices

Writer by:

1. Sumita Arora 12th IP
2. Oxford 12th IP

THE END



© Can Stock Photo - csp18229338