In [1]:

```
# In addition to passing default integer indexes, you can also
# pass named or labeled indexes to the loc function.
# Let's create a dataframe with named indexes. Run the
# following script to do so:

import pandas as pd
scores = [
{'Subject':'Mathematics', 'Score':85, 'Grade': 'B', 'Remarks': 'Good',
},
{'Subject':'History', 'Score':98, 'Grade': 'A','Remarks':
'Excellent'},
{'Subject':'English', 'Score':76, 'Grade': 'C','Remarks': 'Fair'},
{'Subject':'Science', 'Score':72, 'Grade': 'C','Remarks': 'Fair'},
{'Subject':'Arts', 'Score':95, 'Grade': 'A','Remarks': 'Excellent'},
]
my_df = pd.DataFrame(scores, index = ["Student1", "Student2", "Student3",
"Student4", "Student5"])
my_df

# From the output below, you can see that the my_dfdataframe
# now contains named indexes, e.g., Student1, Student2, etc.
```

Out[1]:

	Subject	Score	Grade	Remarks
Student1	Mathematics	85	B	Good
Student2	History	98	A	Excellent
Student3	English	76	C	Fair
Student4	Science	72	C	Fair
Student5	Arts	95	A	Excellent

In [2]:

```
# Let's now filter a record using Student1 as the index value in
# the loc function.

my_df.loc["Student1"]
```

Out[2]:

```
Subject    Mathematics
Score               85
Grade                B
Remarks           Good
Name: Student1, dtype: object
```

In [3]:

```
# As shown below, you can specify multiple named indexes in a
# list to the loc method. The script below filters records with
# indexes Student1 and Student2.

index_list = ["Student1", "Student2"]
my_df.loc[index_list]
```

Out[3]:

	Subject	Score	Grade	Remarks
Student1	Mathematics	85	B	Good
Student2	History	98	A	Excellent

In [4]:

```
# You can also find the value in a particular column while
# filtering records using a named index.
# The script below returns the value in the Grade column for the
# record with the named index Student1.

my_df.loc["Student1", "Grade"]
```

Out[4]:

```
'B'
```

In [5]:

```
# As you did with the default integer index, you can specify a
# range of records using the named indexes within the loc
# function.
# The following function returns values in the Grade column for
# the indexes from Student1 to Student2.

my_df.loc["Student1":"Student2", "Grade"]
```

Out[5]:

```
Student1    B
Student2    A
Name: Grade, dtype: object
```

In [6]:

```
# Let's see another example.
# The following function returns values in the Grade column for
# the indexes from Student1 to Student4.

my_df.loc["Student1":"Student4", "Grade"]
```

Out[6]:

```
Student1    B
Student2    A
Student3    C
Student4    C
Name: Grade, dtype: object
```

In [7]:

```
# You can also specify a list of Boolean values that correspond
# to the indexes to select using the loc method.
# For instance, the following script returns only the fourth
# record since all the values in the list passed to the loc function
# are false, except the one at the fourth index.

my_df.loc[[False, False, False, True, False]]
```

Out[7]:

	Subject	Score	Grade	Remarks
Student4	Science	72	C	Fair

In [8]:

```
# You can also pass dataframe conditions inside the loc method.
# A condition returns a boolean value which can be used to
# index the loc function, as you have already seen in the
# previous scripts.
```

In [11]:

```
# Before you see how loc function uses conditions, let's see the
# outcome of a basic condition in a Pandas dataframe. The
# script below returns index names along with True or False
# values depending on whether the Score column contains a
# value greater than 80 or not.

my_df["Score"]>80

# You can see Boolean values in the output. You can see that
# indexes Student1, Student2, and Student5 contain True.
```

Out[11]:

```
Student1     True
Student2     True
Student3    False
Student4    False
Student5     True
Name: Score, dtype: bool
```

In [12]:

```
# Now, let's pass the condition "my_df["Score"]>80" to the loc
# function.

my_df.loc[my_df["Score"]>80]

# In the output, you can see records with the indexes Student1,
# Student2, and Student5.
```

Out[12]:

	Subject	Score	Grade	Remarks
Student1	Mathematics	85	B	Good
Student2	History	98	A	Excellent
Student5	Arts	95	A	Excellent

In [13]:

```
# You can pass multiple conditions to the loc function. For
# instance, the script below returns those rows where the Score
# column contains a value greater than 80, and the Remarks
# column contains the string Excellent.

my_df.loc[(my_df["Score"]>80) & (my_df["Remarks"] == "Excellent")]
```

Out[13]:

	Subject	Score	Grade	Remarks
Student2	History	98	A	Excellent
Student5	Arts	95	A	Excellent

In [14]:

```
# Finally, you can also specify column names to fetch values
# from, along with a condition.
# For example, the script below returns values from the Score
# and Grade columns, where the Score column contains a value
# greater than 80.

my_df.loc[my_df["Score"]>80, ["Score","Grade"]]
```

Out[14]:

	Score	Grade
Student1	85	B
Student2	98	A
Student5	95	A

In [15]:

```
# Finally, you can set values for all the columns in a row using
# the loc function. For instance, the following script sets values
# for all the columns for the record at index Student4 as 90.

my_df.loc["Student4"] = 90
my_df
```

Out[15]:

	Subject	Score	Grade	Remarks
Student1	Mathematics	85	B	Good
Student2	History	98	A	Excellent
Student3	English	76	C	Fair
Student4	90	90	90	90
Student5	Arts	95	A	Excellent