CHAPTER 7

# Lists and Tuples

# Topics

- Sequences
- Introduction to Lists
- List Slicing
- Finding Items in Lists with the `in` Operator
- List Methods and Useful Built-in Functions
- Copying Lists
- Processing Lists
- Two-Dimensional Lists
- Tuples
- Plotting List Data with the `matplotlib` Package

# Sequences

- **<u>Sequence</u>: an object that contains multiple items of data**
  - The items are stored in sequence one after another
- **Python provides different types of sequences, including lists and tuples**
  - The difference between lists and tuples
    - a list is mutable (can be changed)
    - a tuple is immutable (can not be changed)

# Introduction to Lists

- **List: an object that contains multiple data items**
  - Element: An item in a list
  - Format: *list* = [*item1*, *item2*, etc.]

**Figure 7-1** A list of integers


```
even_numbers = [2, 4, 6, 8, 10]
```

**Figure 7-2** A list of strings


```
names = ['Molly', 'Steven', 'Will', 'Alicia', 'Adriana']
```

# Introduction to Lists

- **<u>List</u>: an object that contains multiple data items**
  - Can hold items of different types

**Figure 7-3** A list holding different types


```
info = ['Alicia', 27, 1550.87]
```

# Introduction to Lists

- `print` function can be used to display an entire list

```
# Program printing entire list by using print function
def main():
    even_numbers = [2, 4, 6, 8, 10]
    names = ['Molly', 'Steven', 'Will', 'Alicia', 'Adriana']
    info = ['Alicia', 27, 1550.87]
    print(even_numbers)
    print(names)
    print(info)
# Calling Main Function
main()
```

**Program Output**

```
[2, 4, 6, 8, 10]
['Molly', 'Steven', 'Will', 'Alicia', 'Adriana']
['Alicia', 27, 1550.87]
```

# Introduction to Lists

- **Built-in `list()` function can convert certain types of objects to lists**
  - Example: Following statement converts the range function's iterable object to a list

```
numbers = list(range(5))
```

## Example Program:

```
# Program printing entire list by using print function
def main():
    numbers=list(range(11))
    even_numbers = list(range(0,11,2))
    names = ['Molly', 'Steven', 'Will', 'Alicia', 'Adriana']
    odd_numbers = list(range(1,11,2))
    print(numbers)
    print(odd_numbers)
    print(even_numbers)
# Calling Main Function
main()
```

**Program Output**

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 3, 5, 7, 9]
[0, 2, 4, 6, 8, 10]
```

# The Repetition Operator *

- **<u>Repetition operator</u>: * makes multiple copies of a list and joins them together**
  - The * symbol is a repetition operator when applied to a sequence and an integer
    - Sequence is left operand, number is right
  - General format: *list * n*

## Example Program:

```
# Program uses * repetition operator to dublicate a list
def main():
    x=[0]*5
    y=list(range(1,6,1))*2
    z=3*list(range(1,6,1))
    w=y*2
    print(x)
    print(y)
    print(z)
    print(w)
# Calling Main Function
main()
```

**Program Output**

```
[0, 0, 0, 0, 0]
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

# Iterating over a List using `for` loop

- **You can iterate over a list using a `for` loop**
  - Format: `for x in list:`
  - `x` variable will iterate through each elements in the list

## Example Program-1:

```
# Program uses for repetition to to print all elements in a list
def main():
    numbers = [45, 55, 75, 105]
    for x in numbers:
        print(x)

# Calling Main Function
main()
```

**Program Output**

```
45
55
75
105
```

## Example Program-2:

```
# Program uses for repetition to print all elements in a list
names=['Alice','John','Taylor','Robert']

for name in names:
    print(name)
```

**Program Output**

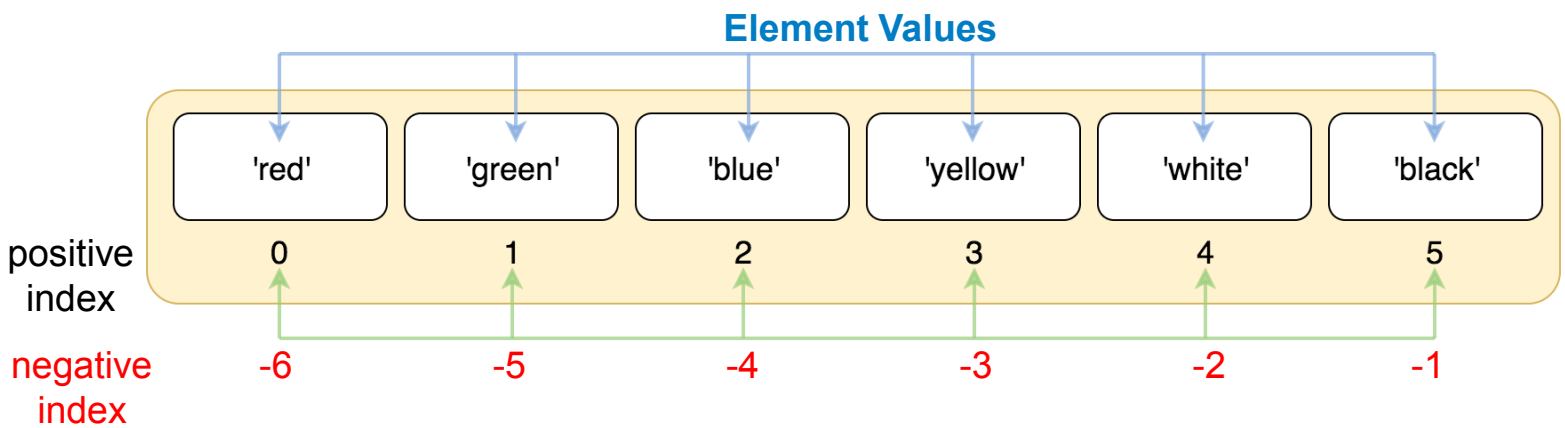
```
Alice
John
Taylor
Robert
```

# Indexing

- **<u>Index</u>: a number specifying the position of an element in a list**
  - Enables access to individual element in list
  - Index of first element in the list is 0, second element is 1, and n'th element is n-1
  - Negative indexes identify positions relative to the end of the list
    - The index -1 identifies the last element, -2 identifies the next to last element, etc.
  - Positive and negative indexes are also called as front and rear indexes in literature.

**<u>Example:</u> Let's consider the colors list with 6 elements.**

```
colors=['red','green','blue','yellow','white','black']
```



# Example: Using Indexes to Process a list

- **Let's consider that we have a list with 4 elements as:**

```
my_list = [10, 20, 30, 40]
```

- **Let's print the each elements by using indexes (inefficient!)**

```
print(my_list[0], my_list[1], my_list[2], my_list[3])
```

- **List items can be processed by looping through indexes.**

**Using `while` repetition:**

```
index = 0              #This is the index of first element
while index < 4:       #Remember last index is 3 < 4
      print(my_list[index])
      index += 1       #Going to the next element by +1
```

**Using `for` repetition:**

```
for index in range(4):
       print(my_list[index])
```

**In both, indexing start from zero going up to n-1 where n is the number of elements in the list.**

# Example: `IndexError` Exception

- An `IndexError` exception is raised if an invalid index is used

## Example Program:

```
# This program demonstrates IndexError Exception
def main():
    # Creating a list with 6 Elements - So last element index is 5
    colors=['red','green','blue','yellow','white','black']
    for index in range(7):
        print(index,':',colors[index])

# Calling Main Function
main()
```

**Wrong: index iterates through 0 – 6**

**Correct: it should iterates 0-5**

**Solution 6 must be in the range function.**

**Program Output**

```
0 : red
1 : green
2 : blue
3 : yellow
4 : white
5 : black
Traceback (most recent call last):
  File "C:/Python37/sil.py", line 10, in <module>
    main()
  File "C:/Python37/sil.py", line 7, in main
    print(index,':',colors[index])
IndexError: list index out of range
```

`IndexError` **Exception**

# The `len` function

- **`len` function: returns the length of a sequence such as a list**
  - Example: *`size = len(my_list)`*
  - Returns the number of elements in the list, so the index of last element is `len(list)-1`
  - Can be used to prevent an `IndexError` exception when iterating over a list with a loop

## Example Program:

```
# This program uses len to determine the number of elements in list
def main():
    # Creating a list with 6 Elements
    colors=['red','green','blue','yellow','white','black']
    for index in range(len(colors)):
        print(index,':',colors[index])

# Calling Main Function
main()
```

**Program Output**

```
0 : red
1 : green
2 : blue
3 : yellow
4 : white
5 : black
```

# Lists Are Mutable

- **Mutable sequence: the items in the sequence can be changed**
  - Lists are mutable, and so their elements can be changed
- **An expression such as**

`list[1] = new_value` **can be used to assign a new value to a list element of index 1.**

- **Must use a valid index to prevent an `IndexError` exception**

## Example Program:

```
# This program demonstrates changing the elements in a list
def main():
    numbers = [1, 2, 3, 4, 5] #Create a list with 5 elements.
    numbers[0] = 10
    numbers[1] += 2
    numbers[2] *= numbers[3]
    numbers[4] = numbers[0]+2*numbers[3]
    print(numbers)

# Calling Main Function
main()
```

**Program Output**

```
[10, 4, 12, 4, 18]
```

# Example: Sales Tracking by Using List

- A program keeps track of sales amounts for 5 days.

```
# The NUM_DAYS constant holds the number of
# days that we will gather sales data for.
NUM_DAYS = 5

def main():
    # Create a list to hold the sales
    # for each day.
    sales = [0] * NUM_DAYS

    # Create a variable to hold an index.
    index = 0

    print('Enter the sales for each day.')

    # Get the sales for each day.
    while index < NUM_DAYS:
        print('Day #', index + 1, ': ', sep='', end='')
        sales[index] = float(input())
        index += 1

    # Display the values entered.
    print('Here are the values you entered:')
    for value in sales:
        print(value)

# Call the main function.
main()
```

Creating a list of 5 elements initially all 0

Index iterates through 0 - 4

inputting sales[index]

printing sales values in `for` loop

**Program Output**

```
Enter the sales for each day.
Day #1: 1247.35
Day #2: 869.45
Day #3: 1250
Day #4: 1300.11
Day #5: 1121.09
Here are the values you entered:
1247.35
869.45
1250.0
1300.11
1121.09
```

# Concatenating Lists

- **<u>Concatenate</u>: join two things together**
- **The + operator can be used to concatenate two lists**
  - Cannot concatenate a list with another data type, such as a number
- **The += augmented assignment operator can also be used to concatenate lists**

## Example Program-1:

```
# Concatenating two integer lists
def main():
    list1 = [1, 2, 3, 4]
    list2 = [5, 6, 7, 8]
    list3 = list1 + list2
    print('list3:',list3)

# Call the main function.
main()
```

**Program Output**

```
list1: [1, 2, 3, 4]
list2: [5, 6, 7, 8]
list3: [1, 2, 3, 4, 5, 6, 7, 8]
```

## Example Program-2:

```
# This program demonstrates cocantenating string lists
def main():
    girl_names = ['Joanne', 'Karen', 'Lori']
    boy_names = ['Chris', 'Jerry', 'Will']
    all_names = girl_names + boy_names
    girl_names+=['Jessica','Susan']
    boy_names+=['Thomas']
    print('Girl_Names:',girl_names)
    print('Boy_Names:',boy_names)
    print('All_Names:',all_names)
#Calling Main Function
main()
```

**Program Output**

```
Girl_Names: ['Joanne', 'Karen', 'Lori', 'Jessica', 'Susan']
Boy_Names: ['Chris', 'Jerry', 'Will', 'Thomas']
All_Names: ['Joanne', 'Karen', 'Lori', 'Chris', 'Jerry', 'Will']
```

**7.1** What will the following code display?

```
numbers = [1, 2, 3, 4, 5]
numbers[2] = 99
print(numbers)
```

```
[1, 2, 99, 4, 5]
```

**7.2** What will the following code display?

```
numbers = list(range(3))
print(numbers)
```

```
[0, 1, 2]
```

**7.3** What will the following code display?

```
numbers = [10] * 5
print(numbers)
```

```
[10, 10, 10, 10, 10]
```

**7.4** What will the following code display?

```
numbers = list(range(1, 10, 2))
for n in numbers:
        print(n, end='')
```

```
13579
```

**7.5** What will the following code display?

```
numbers = [1, 2, 3, 4, 5]
print(numbers[-2])
```

```
4
```

**7.6** How do you find the number of elements in a list?

`len` function gives the size of the list – number of elements in the list.

**7.8** What will the following code display?

```
numbers1 = [1, 2, 3]
numbers2 = [10, 20, 30]
numbers2 += numbers1
print(numbers1)
print(numbers2)
```

```
[1, 2, 3]
[10, 20, 30, 1, 2, 3]
```

# List Slicing

- **<u>Slice</u>: a span of items that are taken from a sequence – slicing helps us to get certain part of the list.**
  - `list` function is used for slicing
  - Format: *`list[start : end]`*
  - Span is a list containing copies of elements from *`start`* up to, but not including, *`end`*
    - If *`start`* not specified, `0` is used for start index
    - If *`end`* not specified, `len(list)` is used for end index
  - Slicing expressions can include a step value and negative indexes relative to end of list

## Example:

indexes: 0 1 2 3 4 5 6

```
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
mid_days = days[2:5]
week_days = days[1:6]
print('Mid-Days:',mid_days)
print('Week-Days:',week_days)
```

**Slicing the list from 2 to 4 (not including 5)**

**Slicing the list from 1 to 5 (not including 6)**

**Program Output**

```
Mid-Days: ['Tuesday', 'Wednesday', 'Thursday']
Week-Days: ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

# Example: List Slicing by using `list()`

Below example shows the various usage of `list()` function for slicing.

```
    indexes:      0   1   2   3   4   5   6   7   8   9
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> print(numbers[2:7])
[3, 4, 5, 6, 7]
>>> print(numbers[:4])
[1, 2, 3, 4]
>>> print(numbers[5:])
[6, 7, 8, 9, 10]
>>> print(numbers[:])
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> print(numbers[2:8:2])
[3, 5, 7]
>>> print("Let's use some negative indexes")
Let's use some negative indexes
>>> print(numbers[-5:])
[6, 7, 8, 9, 10]
>>> print(numbers[:-3])
[1, 2, 3, 4, 5, 6, 7]
>>> print(numbers[-7:-2])
[4, 5, 6, 7, 8]
>>> print("Does it accept reverse order?")
Does it accept reverse order?
>>> print(numbers[7:2])
[]
>>> print(numbers[-2:-7])
[]
>>> print('*** CONCLUSION ***')
*** CONCLUSION ***
>>> print('SLICES IN REVERSE ORDER IS NOT POSSIBLE')
SLICES IN REVERSE ORDER IS NOT POSSIBLE
>>> print('Position of the start < end values!')
Position of the start < end values!
>>>
```

# Finding Items in Lists with the `in` Operator

- **You can use the `in` operator to determine whether an item is contained in a list.**
  - General format: *`item in list`*
  - Returns `True` if the item is in the list, or `False` if it is not in the list
- **Similarly you can combine `not` and `in` operators as `not in` to determine whether an item is not in a list.**

**Example:**

```
>>> numbers = [-7, 21, 34, 14, 2]
>>> x = 47
>>> y = -7
>>> search=2
>>> name = 'Ali'
>>> x in numbers
False
>>> y in numbers
True
>>> search in numbers
True
>>> name in numbers
False
>>> name not in numbers
True
```

**One may use the resulting `True` or `False` Boolean values in an if decision statement to evaluate if any value is in the list or not.**

# Example: Using `in` operator with a list

Below example shows the usage of `in` operator to determine whether an item is in a list or not.

```
# This program demonstrates the in operator
# used with a list.

def main():
    # Create a list of product numbers.
    prod_nums = ['V475', 'F987', 'Q143', 'R688']

    # Get a product number to search for.
    search = input('Enter a product number: ')

    # Determine whether the product number is in the list.
    if search in prod_nums:
        print(search, 'was found in the list.')
    else:
        print(search, 'was not found in the list.')

# Call the main function.
main()
```

**Could be also done by using `not in` operator.**

```
if search not in prod_nums:
        print(search, 'was not found in the list.')
else:
        print(search, 'was found in the list.')
```

**Example Program Runs:**

**Program Output** (with input shown in bold)

```
Enter a product number: Q143 Enter
Q143 was found in the list.
```

**Program Output** (with input shown in bold)

```
Enter a product number: B000 Enter
B000 was not found in the list.
```

**7.14** What will the following code display?

```
names = ['Jim', 'Jill', 'John', 'Jasmine']
if 'Jasmine' not in names:
        print('Cannot find Jasmine.')
else:
        print("Jasmine's family:",names)
```

```
Jasmine's family: ['Jim', 'Jill', 'John', 'Jasmine']
```

**7.X** What will the following code display?

```
names = ['Jim', 'Jill', 'John', 'Jasmine']
if 'Jasmine' in names:
        print("Jasmine's family:",names)
else:
         print('Cannot find Jasmine.')
```

**It displays the same output as in the 7.14.**

```
Jasmine's family: ['Jim', 'Jill', 'John', 'Jasmine']
```

# List Methods: `index(item)` List Method

- **`index(item)`: used to determine where an item is located in a list**
  - Returns the index of the first element in the list containing `item`
  - Raises `ValueError` exception if *`item`* not in the list
    - Must make sure that the item value is in the list.

**Example:** Using `try/except` statement for `index` method.

```
names = ['Jim', 'Jill', 'John', 'Jasmine']
name=input('Enter a name to search:')
try:
    index = names.index(name)
    print(name,'is found with index of',index)
except ValueError:
    print('Cannot find',name,'in the name list')
```

**Alternatively:** Using `in` operator to check if the item is in the list then it uses the `index` method to find index of the item.

```
names = ['Jim', 'Jill', 'John', 'Jasmine']
name=input('Enter a name to search:')
if name in names:
    index = names.index(name)
    print(name,'is found with index of',index)
else:
    print('Cannot find',name,'in the name list')
```

**Example Program Runs:**

```
Enter a name to search:John
John is found with index of 2
Enter a name to search:Thomas
Cannot find Thomas in the name list
```

**Both codes does the same task but in different ways.**

# Example: Changing an Element in a list

Following program demonstrates how to change a certain item in a list.

```
# This program demonstrates how to get the index of an item
# in a list and then replace that item with a new item.

def main():
    # Create a list with some items.
    foods = ['Pizza', 'Burgers', 'Kebap','Chips','Lahmacun']

    # Display the list.
    print('Here are the items in the foods list:')
    print(foods)
    # Get the item to change.
    item = input('Which item should I change? ')
    if item in foods:    # Check if in the list
        # If so then Get the item's index in the list.
        item_index = foods.index(item)
        # Get the value to replace it with from user
        new_item = input('Enter the new value: ')

        # Replace the old item with the new item.
        foods[item_index] = new_item
        # Display the list.
        print('Here is the revised list:')
        print(foods)
    else:  # If not found on list print message and finish
        print(item,'is not found in the list.')

# Call the main function.
main()
```

**Example Program Run:**

```
Here are the items in the foods list:
['Pizza', 'Burgers', 'Kebap', 'Chips', 'Lahmacun']
Which item should I change? Burgers
Enter the new value: Beyran
Here is the revised list:
['Pizza', 'Beyran', 'Kebap', 'Chips', 'Lahmacun']
```

# Review Questions

Page 421-422

## Multiple Choice

1. This term refers to an individual item in a list.
   a. element
   b. bin
   c. cubbyhole
   d. slot
4. This is the last index in a list.
   a. 1
   b. 99
   c. 0
   d. The size of the list minus one
5. This will happen if you try to use an index that is out of range for a list.
   a. A `ValueError` exception will occur.
   b. An `IndexError` exception will occur.
   c. The list will be erased and the program will continue to run.
   d. Nothing—the invalid index will be ignored.
7. When the `*` operator's left operand is a list and its right operand is an integer, the operator becomes this.
   a. The multiplication operator
   b. The repetition operator
   c. The initialization operator
   d. Nothing—the operator does not support those types of operands.
11. If you call the `index` method to locate an item in a list and the item is not found, this happens.
   a. A `ValueError` exception is raised.
   b. An `InvalidIndex` exception is raised.
   c. The method returns −1.
   d. Nothing happens. The program continues running at the next statement.

# Algorithm Workbench



1. Write a statement that uses the `list` and `range` functions to create a list of the numbers from 1 to 100.

```
numbers = list(range(1,101))
```

2. Assume `names` references a list. Write a `for` loop that displays each element of the list.

```
for item in names:
    print(item)
```

OR

```
for index in range(len(names)):
    print(names[index])
```

7. What will `list1` and `list2` contain after the following statements are executed?

```
list1 = [1, 2] * 2      → list1 becomes [1, 2, 1, 2]
list2 = [3]             → list2 becomes [3]
list2 += list1          → list2 becomes [3, 1, 2, 1, 2]
```

So after executing these statements at the end

```
list1 is [1, 2, 1, 2]
list2 is [3, 1, 2, 1, 2]
```

# Programming Exercises



## 2. Lottery Number Generator

Design a program that generates a seven-digit lottery number. The program should generate seven random numbers, each in the range of 0 through 9, and assign each number to a list element. (Random numbers were discussed in Chapter 5.) Then write another loop that displays the contents of the list.

```
import random

def main():
    # Initialize list of numbers.
    number_list = [0, 0, 0, 0, 0, 0, 0]

    # Assign random numbers to each elements in the list.
    for i in range(7):
        number_list[i] = random.randint(0, 9)

    # Display numbers in a single line going through each elements.
    for i in range(7):
        print (number_list[i], end=' ')

# Call the main function.
main()
```

**Program Output:**

```
3 5 3 4 0 8 8
```

# Programming Exercises



## 8. Name Search

- `GirlNames.txt` file contains a list of the 200 most popular names given to girls born in the United States from the year 2000 through 2009.
- `BoyNames.txt` This file contains a list of the 200 most popular names given to boys born in the United States from the year 2000 through 2009.

Write a program that reads the contents of the two files into two separate lists. The user should be able to enter a boy's name, a girl's name, or both, and the application will display messages indicating whether the names were among the most popular.

BoyNames.txt - Notepad

File Edit Format View Help

```
Brandon
Christian
Dylan
Samuel
Benjamin
Zachary
Nathan
Logan
Justin
Gabriel
Jose
Austin
Kevin
Elijah
Caleb
```

GirlNames.txt - Notepad

File Edit Format View Help

```
Emily
Madison
Emma
Olivia
Hannah
Abigail
Isabella
Samantha
Elizabeth
Ashley
Alexis
Sarah
Sophia
Alyssa
Grace
```

**Example Program Output:**

```
Enter a boy's name, or N for not to search:John
Enter a girl's name, or N for not to search:July
John is one of the most popular boy's names.
July is not one of the most popular girl's names.
```

# Programming Exercises



## 8. Name Search

```
def main():
    # Open the files for reading.
    boy_input = open('BoyNames.txt', 'r')
    girl_input = open('GirlNames.txt', 'r')
    # Read all the lines in the files into a lists.
    popular_boys = boy_input.readlines()
    popular_girls = girl_input.readlines()
    # Strip trailing '\n' from all elements of the lists.
    for i in range(len(popular_boys)):
        popular_boys[i] = popular_boys[i].rstrip('\n')
    for i in range(len(popular_girls)):
        popular_girls[i] = popular_girls[i].rstrip('\n')

    # Obtain user inputs.
    boy = input("Enter a boy's name, or N for not to search:")
    girl = input("Enter a girl's name, or N for not to search:")

    # Display result for boy's name entered by user
    if boy=='N':
        print("You chose not to enter a boy's name.")
    elif boy in popular_boys: # Searching by using in operator
        print(boy, "is one of the most popular boy's names.")
    else:
        print(boy, "is not one of the most popular boy's names.")

    # Display result for girl's name entered by user
    if girl=='N':
        print("You chose not to enter a girl's name.")
    elif girl in popular_girls: # Searching by using in operator
        print(girl, "is one of the most popular girl's names.")
    else:
        print(girl, "is not one of the most popular girl's names.")

# Call the main function.
main()
```

Opening files in 'r' mode

Reading all lines by using readlines method.
Transferring all names into lists.

Stripping '\n' characters

Getting user inputs

Checking user input for boys.
If it is not N then
It searches the name in the list.

Doing same thing for girls

# List Methods in Python



**Table 7-1** A few of the list methods

| Method | Description |
|---|---|
| `append(item)` | Adds *item* to the end of the list. |
| `index(item)` | Returns the index of the first element whose value is equal to item. A `ValueError` exception is raised if item is not found in the list. |
| `insert(index, item)` | Inserts *item* into the list at the specified *index*. When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one position toward the end of the list. No exceptions will occur if you specify an invalid index. If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list. |
| `sort()` | Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value). |
| `remove(item)` | Removes the first occurrence of *item* from the list. A `ValueError` exception is raised if item is not found in the list. |
| `reverse()` | Reverses the order of the items in the list. |

# List Methods in Python

- `append(item)`: used to add items to a list – *`item`* is appended to the end of the existing list
- `insert(index, item)`: used to insert *`item`* at position *`index`* in the list
- `sort()`: used to sort the elements of the list in ascending order
- `remove(item)`: removes the first occurrence of *`item`* in the list
- `reverse()`: reverses the order of the elements in the list

# Example: Using `append` method 

Below example shows the usage of `append` method to add item to a list.

```
# This program demonstrates how the append
# method can be used to add items to a list.

def main():
    # First, create an empty list.
    name_list = []

    # Create a variable to control the loop.
    again = 'Y'

    # Add some names to the list.
    while again.upper() == 'Y':
        # Get a name from the user.
        name = input('Enter a name: ')

        # Append the name to the list.
        name_list.append(name)

        # Add another one?
        print('Do you want to add another name?')
        again = input('y = yes, anything else = no: ')
        print()

    # Display the names that were entered.
    print('Here are the complete list.')

    print(name_list)

# Call the main function.
main()
```

Appending name item to the list

**Example Program Run:**

```
Enter a name: George
Do you want to add another name?
y = yes, anything else = no: y

Enter a name: Thomas
Do you want to add another name?
y = yes, anything else = no: y

Enter a name: Andrew
Do you want to add another name?
y = yes, anything else = no: y

Enter a name: Ahmet
Do you want to add another name?
y = yes, anything else = no: n

Here are the names you entered.
['George', 'Thomas', 'Andrew', 'Ahmet']
```

**Note:** `upper` string method is converting a string to uppercase.

# Example: Using `insert` method 

Below example shows the usage of `insert` method to insert an item to a specific place in a list.

```
# This program demonstrates the insert method.

def main():
    # Create a list with some names.
    names = ['James', 'Kathryn', 'Bill']

    # Display the list.
    print('The list before the insert:')
    print(names)

    # Insert a new name at element 0.
    names.insert(0, 'Joe')

    # Display the list again.
    print('The list after the insert:')
    print(names)

# Call the main function.
main()
```

Inserting 'Joe' to index 0 (as first item)

**Program Output**

```
The list before the insert:
['James', 'Kathryn', 'Bill']
The list after the insert:
['Joe', 'James', 'Kathryn', 'Bill']
```

# Notes on list `insert` method

- `insert(index, item)`: used to insert *`item`* at position *`index`* in the list

```
>>> n=[1,2,3,4,5]
>>> n
[1, 2, 3, 4, 5]
>>> n.insert(2,7)
>>> n
[1, 2, 7, 3, 4, 5]
```

- **What if the index is out of boundaries (size)?**

```
>>> n=[1,2,3,4,5]
>>> n.insert(2,7)
>>> n
[1, 2, 7, 3, 4, 5]
>>> n.insert(20,-1)
>>> n
[1, 2, 7, 3, 4, 5, -1]
```

- **If the index is out of boundaries then the item is inserted as the last item of the list. So no exception raised!**

# Example: Using `sort` method



Below examples show the usage of `sort` method to sort a list.

**Example 1:**

```
my_list = [9, 1, 0, 2, 14, -6, 7, 4, -1, 8]
print('Original order:', my_list)
my_list.sort()
print('Sorted order:', my_list)
```

**Example 1 Program Run:**

```
Original order: [9, 1, 0, 2, 14, -6, 7, 4, -1, 8]
Sorted order: [-6, -1, 0, 1, 2, 4, 7, 8, 9, 14]
```

**Example 2:**

```
my_list = ['beta', 'alpha', 'delta', 'gamma']
print('Original order:', my_list)
my_list.sort()
print('Sorted order:', my_list)
```

**Example 2 Program Run:**

```
Original order: ['beta', 'alpha', 'delta', 'gamma']
Sorted order: ['alpha', 'beta', 'delta', 'gamma']
```

**Example 3:**

```
my_list = [1.21, 3.24, 0.78, -2.74, 3.61, 1.79, 0.83, -0.11 ]
print('Original order:', my_list)
my_list.sort()
print('Sorted order:', my_list)
```

**Example 3 Program Run:**

```
Original order: [1.21, 3.24, 0.78, -2.74, 3.61, 1.79, 0.83, -0.11]
Sorted order: [-2.74, -0.11, 0.78, 0.83, 1.21, 1.79, 3.24, 3.61]
```

# Example: Using `remove` method 

Below example shows the usage of `remove` method to remove item from a list.

```
# This program demonstrates how to use the remove
# method to remove an item from a list.

def main():
    # Create a list with some items.
    food = ['Pizza', 'Burgers', 'Chips']

    # Display the list.
    print('Here are the items in the food list:')
    print(food)

    # Get the item to remove.
    item = input('Which item should I remove? ')

    try:
        # Remove the item.
        food.remove(item)

        # Display the list.
        print('Here is the revised list:')
        print(food)

    except ValueError:
        print('That item was not found in the list.')

# Call the main function.
main()
```

Removing the item to the list

**Example Program Runs:**

```
Here are the items in the food list:
['Pizza', 'Burgers', 'Chips']
Which item should I remove? Burgers
Here is the revised list:
['Pizza', 'Chips']
```

```
Here are the items in the food list:
['Pizza', 'Burgers', 'Chips']
Which item should I remove? Chip
That item was not found in the list.
```

# Notes on list `remove` method

- **`remove(item)`: removes the first occurrence of *`item`* in the list**

```
>>> n=[1,2,3,4,5]*2
>>> n
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
>>> n.remove(2)
>>> n
[1, 3, 4, 5, 1, 2, 3, 4, 5]
```

- **What if the item to be removed not found?**

```
>>> n
[1, 3, 4, 5, 1, 2, 3, 4, 5]
>>> n.remove(3)
>>> n
[1, 4, 5, 1, 2, 3, 4, 5]
>>> n.remove(7)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    n.remove(7)
ValueError: list.remove(x): x not in list
```

- **When item to be removed is not found in the list then remove method raises exception! We must check if it is exist before attempting to remove to avoid exception!**

# Example: Using `reverse` method  Page 383

Below examples show the usage of `reverse` method to reverse a list.

**Example 1:**

```
my_list = [14, -2, 3, 6, 7, 0, 12]
print('Original order:', my_list)
my_list.reverse()
print('Reversed:', my_list)
```

**Example 1 Program Run:**

```
Original order: [14, -2, 3, 6, 7, 0, 12]
Reversed: [12, 0, 7, 6, 3, -2, 14]
```

**Example 2:**

```
my_list = ['and', 'or', 'nor', 'not']
print('Original order:', my_list)
my_list.reverse()
print('Reversed:', my_list)
```

**Example 2 Program Run:**

```
Original order: ['and', 'or', 'nor', 'not']
Reversed: ['not', 'nor', 'or', 'and']
```

**Example 3:** `sort` and `reverse` methods can be used to sort a list in *descending order*.

```
my_list = [2,7,-1,3,17,45,-8,11,89,32]
print('Original order:', my_list)
my_list.sort()
print('Sorted:', my_list)
my_list.reverse()
print('Reversed:', my_list)
```

**Example 3 Program Run:**

```
Original order: [2, 7, -1, 3, 17, 45, -8, 11, 89, 32]
Sorted: [-8, -1, 2, 3, 7, 11, 17, 32, 45, 89]
Reversed: [89, 45, 32, 17, 11, 7, 3, 2, -1, -8]
```

# Built-in List Functions in Python

- **`del` statement: removes an element from a specific index in a list**
  - General format: `del list[i]`
- **`min` and `max` functions: built-in functions that returns the item that has the lowest or highest value in a sequence**
  - The sequence (list) is passed as an argument.
  - The function/s returns a value back to the caller.
  - General format:

`min(list)` -> returns the minimum value in the list.

`max(list)` -> returns the maximum value in the list.

# Example: Using `del` function p383

- **`del` statement**: removes an element from a specific index in a list - General format: `del list[i]`

**Example 1:**

```
my_list = [11,-4,8,7,2]
print('Before deletion:', my_list)
del my_list[1]
print('After deletion:', my_list)
```

**Example 1 Program Run:**

```
Before deletion: [11, -4, 8, 7, 2]
After deletion: [11, 8, 7, 2]
```

**Example 2:**

```
my_list = ['Ali','Tom','John','Cindy','Andrew']
print('Before deletion:', my_list)
del my_list[2]
print('After deletion:', my_list)
del my_list[-2]
print('After deletion:', my_list)
```

**Example 2 Program Run:**

```
Before deletion: ['Ali', 'Tom', 'John', 'Cindy', 'Andrew']
After deletion: ['Ali', 'Tom', 'Cindy', 'Andrew']
After deletion: ['Ali', 'Tom', 'Andrew']
```

**Example 3:**

```
my_list = [11,-4,8,7,2]
print('Before deletion:', my_list)
del my_list[8]
print('After deletion:', my_list)
```

**Example 3 Program Run:**

```
Before deletion: [11, -4, 8, 7, 2]
Traceback (most recent call last):
  File "C:/Users/Fantom_E7000/Documents/sil.py", line 4, in <module>
    del my_list[8]
IndexError: list assignment index out of range
```

# Example: `min` and `max` functions p383

- **`min` and `max` functions: built-in functions that returns the item that has the lowest or highest value in a sequence**
  - General format:

`min(list)` -> returns the minimum value in the list.

`max(list)` -> returns the maximum value in the list.

**Example:**

```
list_a = [11, -4, 8, 52, 7, 2, -10, 18, 21, 45, 34 ]
list_b = [3.22, -1.24, 2.11, 3.45, -4.52, 11.2, -1.47]
print('Minimum of List a:', min(list_a))
print('Maximum of List a:', max(list_a))
print('Minimum of List b:', min(list_b))
print('Maximum of List b:', max(list_b))
```

**Program Output:**

```
Minimum of List a: -10
Maximum of List a: 52
Minimum of List b: -4.52
Maximum of List b: 11.2
```

7.15 What is the difference between calling a list's `remove` method and using the `del` statement to remove an element?

`remove` method is used remove a specific item.

`del` statement is used to remove an item with a specific index number.

**7.16** How do you find the lowest and highest values in a list?

Built-in `min` and `max` functions can be used to find the lowest and highest values in a list, respectively.

**7.17** Assume the following statement appears in a program:

```
names = []
```

Which of the following statements would you use to add the string `'Wendy'` to the list at index `0`? Why would you select this statement instead of the other?

**a.** `names[0] = 'Wendy'`

**b.** `names.append('Wendy')`

When we declare/define an empty list, there is no indexing described for the list yet. So `names[0]` indexing can not be used. But `append` method can be used to add an item to the list whether if the list is empty or not. So we use b to add an item to empty list.

**7.18** Describe the following list methods:

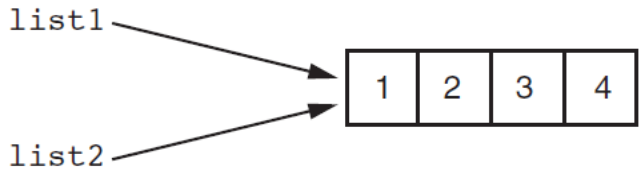
**a.** `index` **b.** `insert` **c.** `sort` **d.** `reverse`

# Copying Lists

- **In Python, assigning a list to another list simply makes both lists reference the same object in memory.**

## Example:

```
# Create a list.
list1 = [1, 2, 3, 4]
# Assign the list to the list2 variable.
list2 = list1
# Assigning new values to list items
list1[1]=10
list2[2]=-5
#Printing Lists
print('list1:',list1)
print('list2:',list2)
```




## Program Output:

```
list1: [1, 10, -5, 4]
list2: [1, 10, -5, 4]
```

# Copying Lists

- **To make a copy of a list you must copy each element of the list**
  - **<u>Two methods to do this:</u>**
    - Creating a new empty list and using a `for` loop to add a copy of each element from the original list to the new list

```
# Create a list with values.
list1 = [1, 2, 3, 4]
# Create an empty list.
list2 = []
# Copy the elements of list1 to list2.
for item in list1:
    list2.append(item)
```

    - Creating a new empty list and concatenating the old list to the new empty list

```
# Create a list with values.
list1 = [1, 2, 3, 4]
# Create a copy of list1.
list2 = [] + list1
```

  - **As a result, `list1` and `list2` will reference two separate but identical lists.**

# Processing Lists

- **List elements can be used in calculations**
- **To calculate total of numeric values in a list use loop with accumulator variable**
- **To average numeric values in a list:**
  - Calculate total of the values
  - Divide total of the values by `len(list)`
- **List can be passed as an argument to a function**

# Example: Processing Lists by using Index

- **Program keeps track of hourly pays for 6 employees**

```
# This program calculates the gross pay for
# each of Megan's baristas.
# NUM_EMPLOYEES is used as a constant for the
# size of the list.
NUM_EMPLOYEES = 6

def main():
    # Create a list to hold employee hours.
    hours = [0] * NUM_EMPLOYEES

    # Get each employee's hours worked.
    for index in range(NUM_EMPLOYEES):
        print('Enter the hours worked by employee ', \
              index + 1, ': ', sep='', end='')
        hours[index] = float(input())

    # Get the hourly pay rate.
    pay_rate = float(input('Enter the hourly pay rate: '))

    # Display each employee's gross pay.
    for index in range(NUM_EMPLOYEES):
        gross_pay = hours[index] * pay_rate
        print('Gross pay for employee ', index + 1, ': $', \
              format(gross_pay, ',.2f'), sep='')

# Call the main function.
main()
```

See Page 387

**Program Output** (with input shown in bold)

```
Enter the hours worked by employee 1: 10 Enter
Enter the hours worked by employee 2: 20 Enter
Enter the hours worked by employee 3: 15 Enter
Enter the hours worked by employee 4: 40 Enter
Enter the hours worked by employee 5: 20 Enter
Enter the hours worked by employee 6: 18 Enter
Enter the hourly pay rate: 12.75 Enter
Gross pay for employee 1: $127.50
Gross pay for employee 2: $255.00
Gross pay for employee 3: $191.25
Gross pay for employee 4: $510.00
Gross pay for employee 5: $255.00
Gross pay for employee 6: $229.50
```

# Example: Processing Lists by using Items

- **Program finds the total of values in a list.**

```
# This program calculates the total of the values
# in a list.

def main():
    # Create a list.
    numbers = [2, 4, 6, 8, 10]

    # Create a variable to use as an accumulator.
    total = 0

    # Calculate the total of the list elements.
    for value in numbers:
        total += value

    # Display the total of the list elements.
    print('The total of the elements is', total)

# Call the main function.
main()
```

See Page 388

**Program Output**

```
The total of the elements is 30
```

**Averaging can be done by dividing the total by the size of the list: `len(list)`**

**See Program 7-9 at Page 389**.

# Example: Passing a List to a Function

- **Program gets a function to find the total of values in a list.**

```
# This program uses a function to calculate the
# total of the values in a list.

def main():
    # Create a list.
    numbers = [2, 4, 6, 8, 10]

    # Display the total of the list elements.
    print('The total is', get_total(numbers))

# The get_total function accepts a list as an
# argument returns the total of the values in
# the list.
def get_total(value_list):
    total = 0
    for num in value_list:
        total += num

    return total #Returning Total

# Call the main function.
main()
```

See Page 390

**Program Output**

```
The total of the elements is 30
```

# Processing Lists (cont'd.)

- **List can be passed to a function**
- **A function can return a reference to a list**
  - Functions can return the whole list if needed

# Example: Returning a List from a Function

- **Program gets a function to return a list back to the caller.**

```
# This program uses a function to create a list.
# The function returns a reference to the list.
def main():
    # Get a list with values stored in it.
    numbers = get_values()

    # Display the values in the list.
    print('The numbers in the list are:')
    print(numbers)

# The get_values function gets a series of numbers
# from the user and stores them in a list. The
# function returns a reference to the list.
def get_values():

    values = [] # Create an empty list.

    # Create a variable to control the loop.
    again = 'Y'

    # Get values from the user and add them to the list.
    while again.upper() == 'Y':
        # Get a number and add it to the list.
        num = int(input('Enter a number: '))
        values.append(num)

        # Want to do this again?
        print('Do you want to add another number?')
        again = input('y = yes, anything else = no: ')

        return values # Return the list to the caller.

# Call the main function.
main()
```

See Page 391

**Program Output** (with input shown in bold)

```
Enter a number: 1 Enter
Do you want to add another number?
y = yes, anything else = no: y Enter

Enter a number: 2 Enter
Do you want to add another number?
y = yes, anything else = no: y Enter

Enter a number: 3 Enter
Do you want to add another number?
y = yes, anything else = no: y Enter
Enter a number: 4 Enter
Do you want to add another number?
y = yes, anything else = no: y Enter

Enter a number: 5 Enter
Do you want to add another number?
y = yes, anything else = no: n Enter
The numbers in the list are:
[1, 2, 3, 4, 5]
```

# Processing Lists (cont'd.)
# Working with Lists and Files

- **To save the contents of a list to a file:**
  - Use the file object's `writelines` method
    - Does not automatically write `\n` at then end of each item
    - So this is not very useful because `\n` is very important
  - Use a `for` loop to write each element and `\n`
- **To read data from a file use the file object's `readlines` method**
  - Use `readlines` to read the complete content of a file into a list.

# Example: Writing a List to a File p395-396

- **Use the file object's `writelines` method**
  - Does not automatically write `\n` at then end of each item

```
# This program uses the writelines method to save
# a list of strings to a file.
def main():
    # Create a list of strings.
    cities = ['New York', 'Boston', 'Atlanta', 'Dallas']
    # Open a file for writing.
    outfile = open('cities.txt', 'w')
    # Write the list to the file.
    outfile.writelines(cities)
    # Close the file.
    outfile.close()
# Call the main function.
main()
```

**After Program Run**

cities - Notepad

File Edit Format View Help

New YorkBostonAtlantaDallas

- **Use a `for` loop to write each element and `\n`**
  - Instead of using `outfile.writelines(cities)`
  - `Use write method to write each element individually.`

```
# Write the list to the file.
for item in cities:
    outfile.write(item + '\n')
```

**After Program Run**

cities - Notepad

File Edit Format View Help

New York
Boston
Atlanta
Dallas

# Example: Reading a File into a List p396

- To read data from a file use the file object's `readlines` method - complete content is transferred to a list.

```
# This program reads a file's contents into a list.

def main():
    # Open a file for reading.
    infile = open('cities.txt', 'r')

    # Read the contents of the file into a list.
    cities = infile.readlines()

    # Close the file.
    infile.close()

    # Strip the \n from each element.
    index = 0
    while index < len(cities):
        cities[index] = cities[index].rstrip('\n')
        index += 1

    # Print the contents of the list.
    print(cities)

# Call the main function.
main()
```

**Program Output**

```
['New York', 'Boston', 'Atlanta', 'Dallas']
```

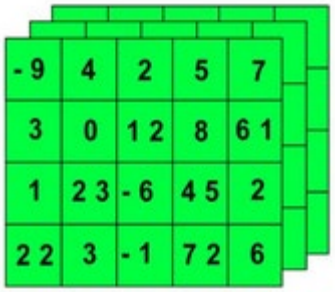
# Two-Dimensional Lists page 398

- **Two-dimensional list: a list that contains other lists as its elements**
  - Also known as nested list
  - Common to think of two-dimensional lists as having rows and columns
  - Useful for working with multiple sets of data
- **To process data in a two-dimensional list need to use two indexes**
- **Typically use nested loops to process**

**1D List like a vector in Math**


**2D List like a Matrix in Math**


**3D List like a Cube in Math**



# Two-Dimensional Lists (cont'd.)

```
>>> students = [['Joe', 'Kim'], ['Sam', 'Sue'], ['Kelly', 'Chris']]
```

**This creates a two dimensional list, size of 3 by 2. (3 Row – 2 Column)**

**Figure 7-5** A two-dimensional list

| | Column 0 | Column 1 |
|---|---|---|
| Row 0 | 'Joe' | 'Kim' |
| Row 1 | 'Sam' | 'Sue' |
| Row 2 | 'Kelly' | 'Chris' |

**Such lists can be processed by row or by individual elements.**

```
>>> print(students[0])
['Joe', 'Kim']
>>>
>>> print(students[2])
['Kelly', 'Chris']
>>>
>>> print(students[1][0])
Sam
>>>
>>> print(students[2][1])
Chris
>>>
>>> print(students)
[['Joe', 'Kim'], ['Sam', 'Sue'], ['Kelly', 'Chris']]
```

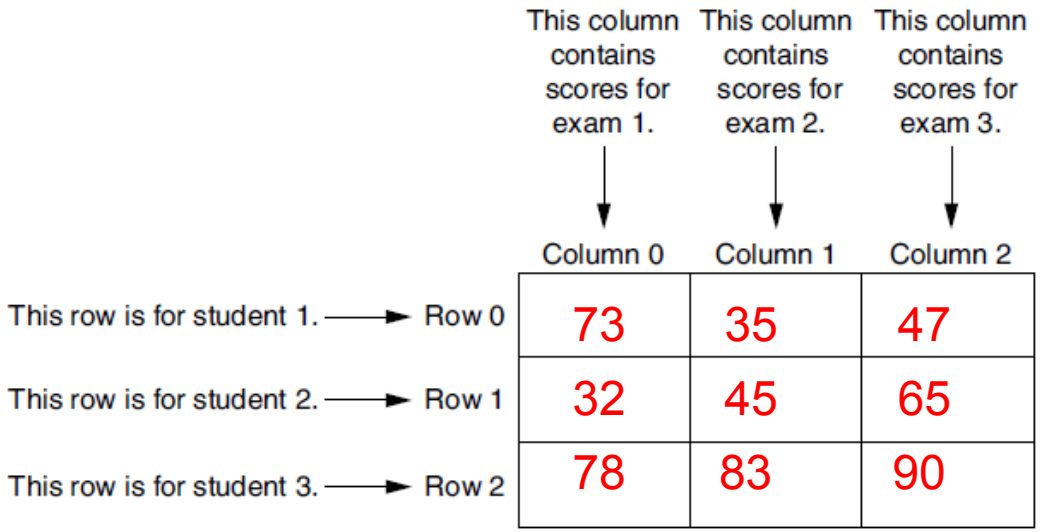
# Example: Exam Scores 2D List



- Imagine, you are dealing with three exam scores for three students.
- We can handle it by using 3-row by 3-column list as below

```
scores = [ [73,35,47], [32,45,65], [78,83,90] ]
```

- Graphically depicted in the figure below




- Subscripts/indexing for each element is shown in the following figure.

| | Column 0 | Column 1 | Column 2 |
|---|---|---|---|
| Row 0 | scores[0][0] | scores[0][1] | scores[0][2] |
| Row 1 | scores[1][0] | scores[1][1] | scores[1][2] |
| Row 2 | scores[2][0] | scores[2][1] | scores[2][2] |

# Example: Exam Scores 2D List



- **Imagine, you are dealing with three exam scores for three students.**
- **You can even add the name and last names of the students to the list.**
- **This can be handled by using 3-row by 5-column list as below.**

```
scores = [ ['Ali','Kurt', 73, 35, 47],
           ['Tom','Cury', 32, 45, 65],
           ['Joe','Doe',  78, 83, 90] ]
```

- **Graphically depicted in the figure below.**

| **Indexing** | Column 0 | Column 1 | Column 2 | Column 3 | Column 4 |
|---|---|---|---|---|---|
| Row 0 | **'Ali'** | **'Kurt'** | **73** | **35** | **47** |
| Row 1 | **'Tom'** | **'Cury'** | **32** | **45** | **65** |
| Row 2 | **'Joe'** | **'Doe'** | **78** | **83** | **90** |

```
>>> print(scores[0])
['Ali', 'Kurt', 73, 35, 47]
>>>
>>> print(scores[0])
['Ali', 'Kurt', 73, 35, 47]
>>> print(scores[2])
['Joe', 'Doe', 78, 83, 90]
>>> print(scores[1][2])
32
>>> for row in scores:
        print(row)
['Ali', 'Kurt', 73, 35, 47]
['Tom', 'Cury', 32, 45, 65]
['Joe', 'Doe', 78, 83, 90]
```

```
>>> for row in scores:
        for col in row:
            print(col)

Ali
Kurt
73
35
47
Tom
Cury
32
45
65
Joe
Doe
78
83
90
```

# Example: Exam Scores 2D List



```
scores = [ ['Ali','Kurt', 73, 35, 47],
           ['Tom','Cury', 32, 45, 65],
           ['Joe','Doe',  78, 83, 90] ]
```

- **We can process 2D list nested repetition structure.**
- **Let's use a indexing in our processing.**

```
scores = [ ['Ali','Kurt', 73, 35, 47],
           ['Tom','Cury', 32, 45, 65],
           ['Joe','Doe',  78, 83, 90] ]

# Printing the Header First
print('First\tLast\tEx-1\tEx-2\tEx-3')

# Let's print this in tabular form.
for r in range(3):
    for c in range(5):
        print(scores[r][c],end='\t')
    print()  # Going to new line
```

**Program Output**

```
First   Last    Ex-1    Ex-2    Ex-3
Ali     Kurt    73      35      47
Tom     Cury    32      45      65
Joe     Doe     78      83      90
```

# Example: Exam Scores 2D List



- **Write a program that displays the person's info and scores who got the lowest mark from the Ex-1.**

**Remember Our List is the following**

| First | Last | Ex-1 | Ex-2 | Ex-3 |
|---|---|---|---|---|
| Ali | Kurt | 73 | 35 | 47 |
| Tom | Cury | 32 | 45 | 65 |
| Joe | Doe | 78 | 83 | 90 |

```
scores = [ ['Ali','Kurt', 73, 35, 47],
           ['Tom','Cury', 32, 45, 65],
           ['Joe','Doe',  78, 83, 90] ]

# Extract the Exam 1 results (3rd Row in the list).
exam_1=[]  # Creating empty list
for row in scores:
     exam_1.append(row[2])  # Appending index 2 value

# Or Alternatively can be done as one line
#exam_1 = [ row[2] for row in scores ]

# Finding the minimum
minimum =  min(exam_1)
# Finding the index of minimum
min_index = exam_1.index(minimum)

# Printing the info
print ('Name:', scores[min_index][0])
print ('Lastname:', scores[min_index][1])
```

**Program Output**

```
Name: Tom
Lastname: Cury
```

# Example: 2D List – Assigning Number p401

- **Program that creates a two-dimensional list (3 by 4 ) and assigns random numbers to each of its elements.**

```
# This program assigns random numbers to
# a two-dimensional list.
import random

# Constants for rows and columns
ROWS = 3
COLS = 4

def main():
    # Create a two-dimensional list.
    values = [[0, 0, 0, 0],
              [0, 0, 0, 0],
              [0, 0, 0, 0]]

    # Fill the list with random numbers.
    for r in range(ROWS):
        for c in range(COLS):
            values[r][c] = random.randint(1, 100)

    # Display the random numbers.
    print(values)

# Call the main function.
main()
```

**Program Output**

```
[[100, 91, 85, 92], [82, 33, 18, 78], [6, 92, 55, 63]]
```

**7.19** Look at the following interactive session, in which a two-dimensional list is created. How many rows and how many columns are in the list?

```
numbers = [[1, 2], [10, 20], [100, 200], [1000, 2000]]
```

**It is 4 rows and 2 column list.**

**7.20** Write a statement that creates a two-dimensional list with three rows and four columns. Each element should be assigned the value 0.

```
my_list=[ [0,0,0,0], [0,0,0,0], [0,0,0,0] ]
```

**7.21** Write a set of nested loops that display the contents of the `numbers` list shown in Checkpoint question **7.19**.

```
for row in numbers:
    for col in row:
        print(col)
```

# Tuples

- **<u>Tuple</u>: an immutable sequence**
  - Very similar to a list
  - Once it is created it cannot be changed
  - Format: `tuple_name = (item1, item2)`

```
>>> my_tuple = (1, 2, 3, 4, 5)
>>> print(my_tuple)
(1, 2, 3, 4, 5)
```

```
>>> names = ('Holly', 'Warren', 'Ashley')
>>> print(names)
('Holly', 'Warren', 'Ashley')
```

# Tuples (cont'd.)

- **Tuples support operations as lists**
  - Subscript indexing for retrieving elements
  - Methods such as `index`
  - Built in functions such as `len, min, max`
  - Slicing expressions
  - The `in`, `+`, and `*` operators

```
>>> names = ('Holly', 'Warren', 'Ashley')
>>>
>>> for n in names:
        print(n)


Holly
Warren
Ashley
```

```
>>> for i in range(len(names)):
        print(names[i])


Holly
Warren
Ashley
```

# Tuples (cont'd.)

- **Remember Tuples are immutable.**
- **We can not modify the tuples.**
- **Tuples do not support the methods:**
  - `append`
  - `remove`
  - `insert`
  - `reverse`
  - `sort`

# Tuples vs. Lists

- **Advantages for using tuples over lists:**
  - Processing tuples is faster than processing lists.
  - If the data no need to change then Tuples are better option than using lists.
  - Tuples are safe – data is not allowed to change.
  - Some operations in Python require use of tuples.
- **`list()` function: converts tuple to list**
- **`tuple()` function: converts list to tuple**

# Converting Between Lists and Tuples

- **built-in `list()` and `tuple()` functions can be used to do conversion between tuple and list.**
- `list()` function: converts tuple to list

```
>>> number_tuple = (1, 2, 3)
>>> number_list = list(number_tuple)
>>> print(number_tuple)
(1, 2, 3)
>>> print(number_list)
[1, 2, 3]
```

- `tuple()` function: converts list to tuple

```
>>> str_list = ['one', 'two', 'three']
>>> str_tuple = tuple(str_list)
>>> print(str_list)
['one', 'two', 'three']
>>> print(str_tuple)
('one', 'two', 'three')
```

**7.22** What is the primary difference between a list and a tuple?

Lists are mutable. But tuples are immutable.

**7.23** Give two reasons why tuples exist.

Processing tuples are faster.

Tuples are safe.

Some operations in Python requires usage of tuples.

**7.24** Assume `my_list` references a list. Write a statement that converts it to a tuple.

```
my_tuple = tuple( my_list)
```

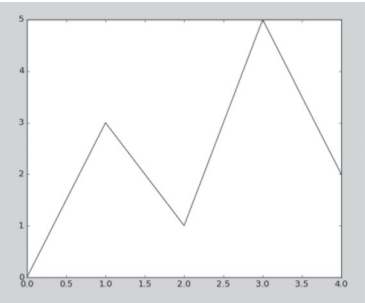
**7.25** Assume `my_tuple` references a tuple. Write a statement that converts it to a list.

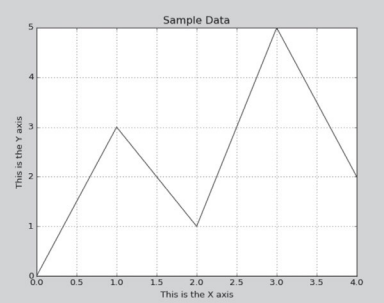
```
my_list = list( my_tuple )
```

# Plotting Data with `matplotlib`

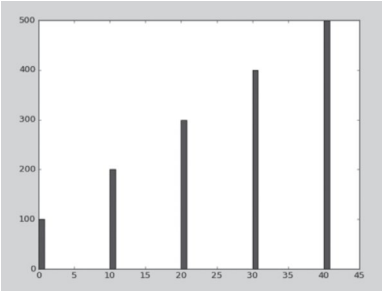
- The `matplotlib` package is a library for creating two-dimensional charts and graphs.

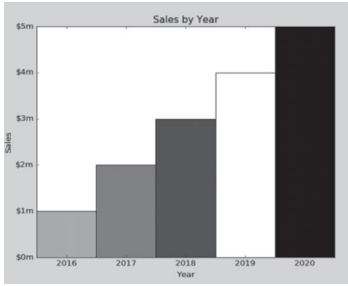
## LINE GRAPH



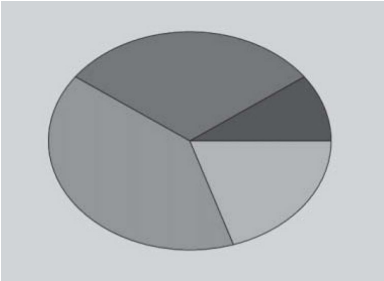


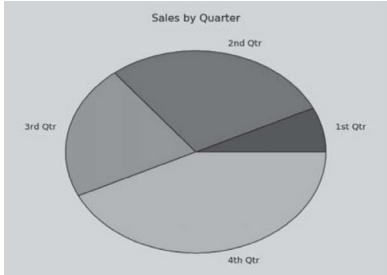

## BAR CHART






## PIE CHART






- **It is <u>not part of the standard Python library</u>, so you will have to install it separately, after you have installed Python on your system.**

# Installing `matplotlib` Library

- **To install `matplotlib` on a Windows system, open a Command Prompt window and enter this command:**

```
pip install matplotlib
```

```
C:\Python37\Scripts>pip install matplotlib
Collecting matplotlib
  Downloading https://files.pythonhosted.org/packages/3f/16/4500e22ea8d11f4946bc
902695d0113f82a0aaca45f352478f157ca6623d/matplotlib-3.0.2-cp37-cp37m-win32.whl (
8.7MB)
    100% |[redacted]| 8.7MB 1.3MB/s
Collecting python-dateutil>=2.1 (from matplotlib)
  Downloading https://files.pythonhosted.org/packages/41/17/c62faccbfbd163c7f57f
3844689e3a78bae1f403648a6afb1d0866d87fbb/python_dateutil-2.8.0-py2.py3-none-any.
whl (226kB)
    100% |[redacted]| 235kB 3.3MB/s
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 (from matplotlib)
  Downloading https://files.pythonhosted.org/packages/de/0a/001be530836743d8be6c
2d85069f46fecf84ac6c18c7f5fb8125ee11d854/pyparsing-2.3.1-py2.py3-none-any.whl (6
1kB)
```

- **To install `matplotlib` on a Mac or Linux system, open a Terminal window and enter this command:**

```
sudo pip3 install matplotlib
```

- **See Appendix F in your textbook for more information about packages and the `pip` utility.**

**No need to install library every time on your PC.**

# Importing `matplotlib` Library

- **To verify the package was installed, start IDLE and enter this command:**

```
>>> import matplotlib
```

```
>>> import matplotlib
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import matplotlib
ModuleNotFoundError: No module named 'matplotlib'
```

- **If you don't see any error messages, you can assume the package was properly installed.**

```
>>> import matplotlib
>>>
```

- **You can start using the library methods and functions.**

# `pyplot` Module under `matplotlib`

- **The `matplotlib` package contains a module named `pyplot` that you will need to import.**

- **Use the following `import` statement to import the module and create an <u>alias</u> named `plt`:**

```
import matplotlib.pyplot as plt
```

- **Instead of typing `mathplotlib.pyplot`, we can type `plt`**

***For more information about the `import` statement, see Appendix E in your textbook.***

# Plotting a Line Graph with the `plot` Function

- **Use the `plot` function to create a line graph that connects a series of points with straight lines.**
- **The line graph has a horizontal *X* axis, and a vertical *Y* axis.**
- **Each point in the graph is located at a ( *X* , *Y* ) coordinate.**

**Example: We want to play (0, 0) , (1, 3), (2, 1),(3, 5) and (4, 2)**

- **Create two lists to hold the x- and y- coordinates**

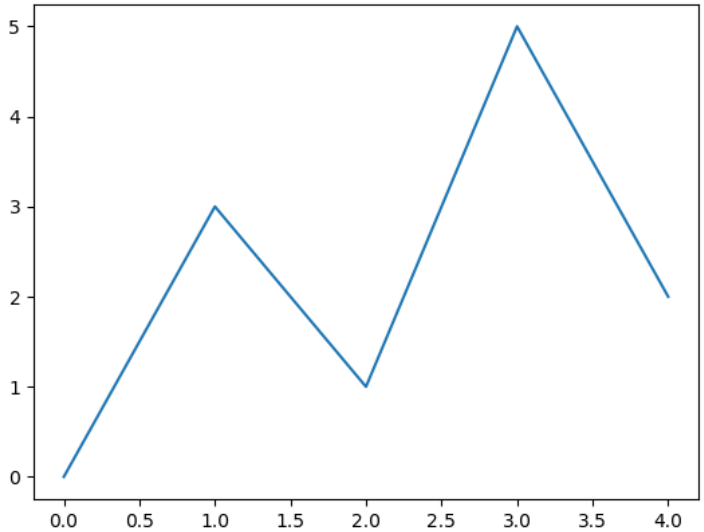
```
x_coords = [0, 1, 2, 3, 4]
y_coords = [0, 3, 1, 5, 2]
```

- Use `plot` function to plot the data

```
plt.plot(x_coords, y_coords)
```

- **Use `show` function to display graph.**

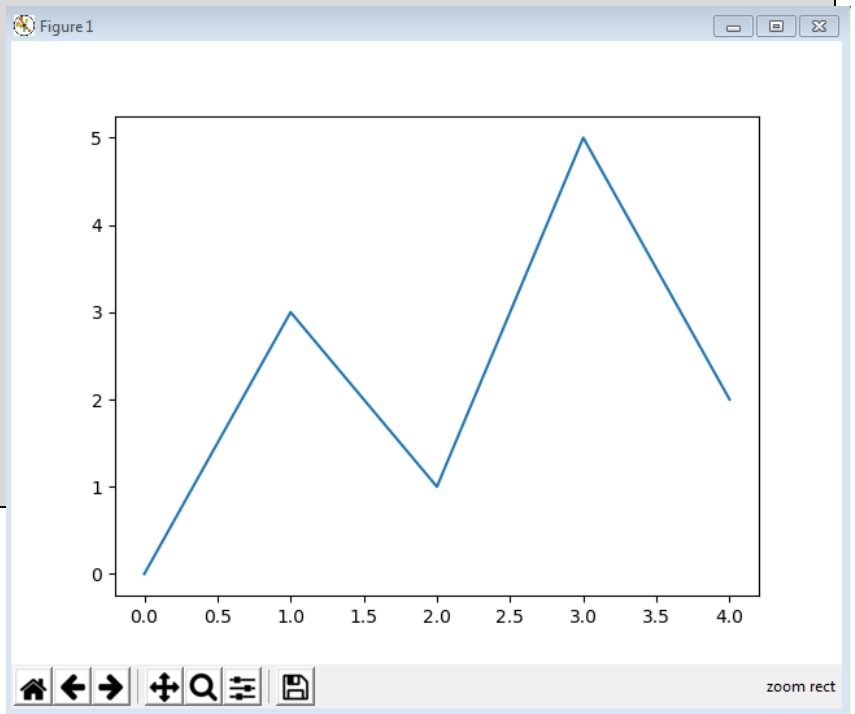
```
plt.show()
```



# Plotting a Line Graph with the `plot` Function

**Program 7-19** Page 404

```
# This program displays a simple line graph.
import matplotlib.pyplot as plt

def main():
    # Create lists with the X and Y coordinates of each data point.
    x_coords = [0, 1, 2, 3, 4]
    y_coords = [0, 3, 1, 5, 2]

    # Build the line graph.
    plt.plot(x_coords, y_coords)

    # Display the line graph.
    plt.show()

# Call the main function.
main()
```



# Line Graph: `xlim` and `ylim` functions

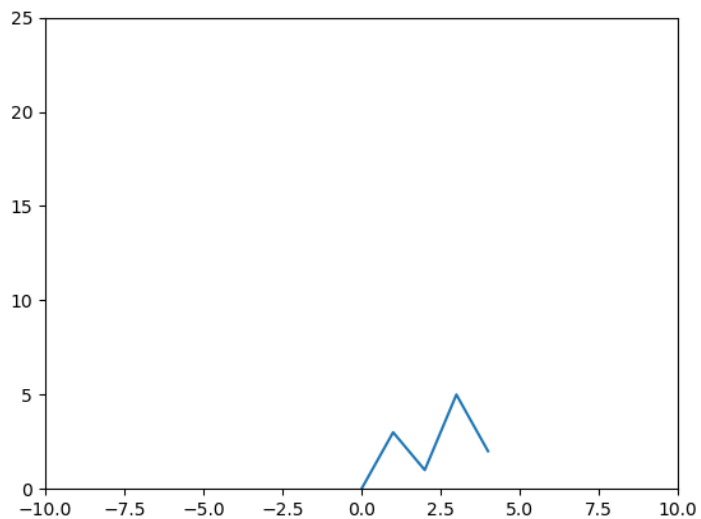
- **You can change the lower and upper limits of the *X* and *Y* axes by calling the `xlim` and `ylim` functions.**

**Example:**

```
plt.xlim(xmin=-10, xmax=10)
plt.ylim(ymin=0, ymax=50)
```

- **This code does the following:**
  - Causes the *X* axis to begin at -10 and end at 10
  - Causes the *Y* axis to begin at 0 and end at 50

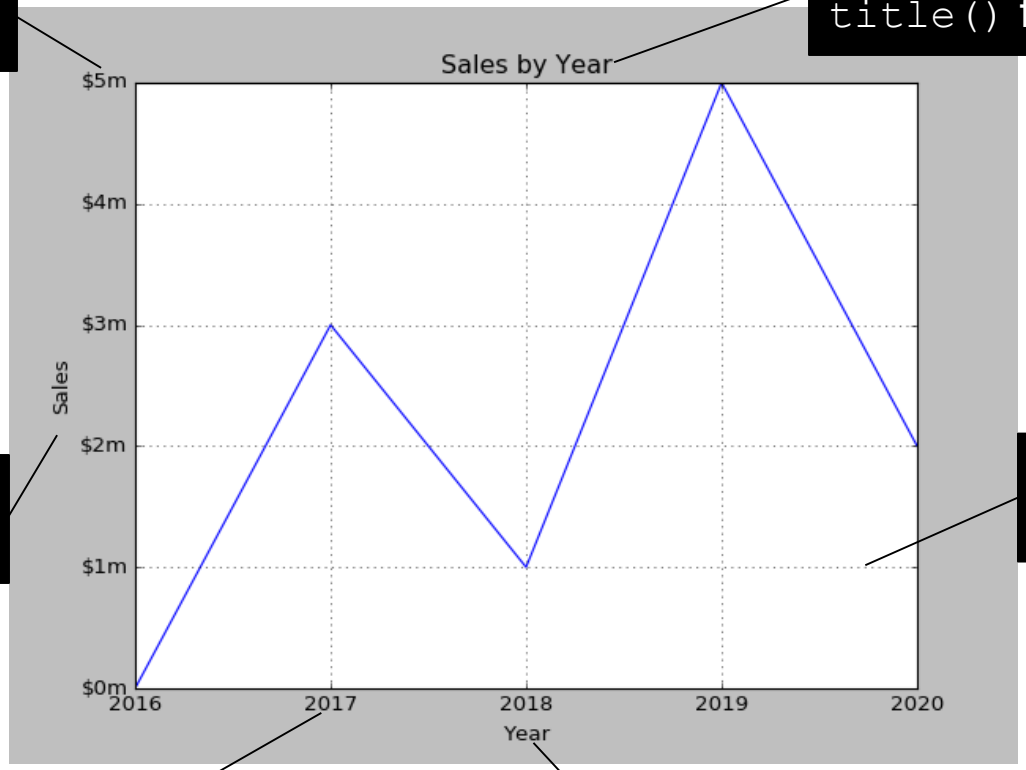
```
# This program displays a simple line graph.
import matplotlib.pyplot as plt

def main():
    # Create lists with the X and Y coordinates
    x_coords = [0, 1, 2, 3, 4]
    y_coords = [0, 3, 1, 5, 2]

    # Build the line graph.
    plt.plot(x_coords, y_coords)


    #Setting Max and Min Values
    plt.xlim(xmin=-10, xmax=10)
    plt.ylim(ymin=0, ymax=25)

    # Display the line graph.
    plt.show()


# Call the main function.
main()
```



# Further Functions to Modify a Graph

Change y-ticks by the `yticks()` function.

Display title by the `title()` function.




Display y-label by the `ylabel()` function.

Display gridlines `grid()` function.

Change xticks by the `xticks()` function.

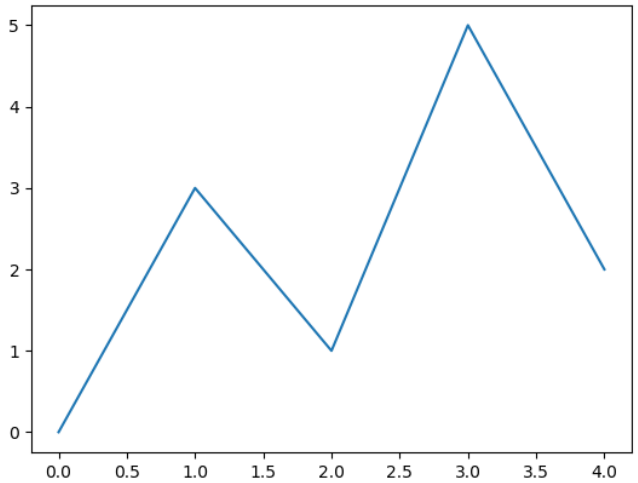
Display x-label by the `xlabel()` function.

# Line Graph: `xticks` and `yticks` functions

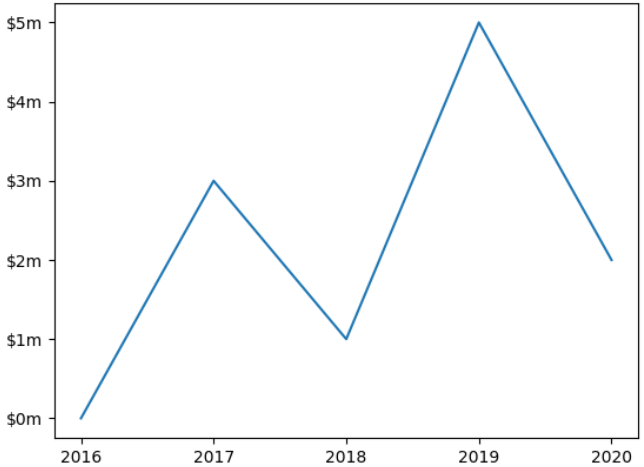
- **You can customize each tick mark's label with the `xticks` and `yticks` functions.**
- **These functions each take two lists as arguments.**
  - The first argument is a list of tick mark locations
  - The second argument is a list of labels to display at the specified locations.

```
#Changing x- and y- ticks
plt.xticks([0, 1, 2, 3, 4],
       ['2016', '2017', '2018', '2019', '2020'])
plt.yticks([0, 1, 2, 3, 4, 5],
       ['$0m', '$1m', '$2m', '$3m', '$4m', '$5m'])
```

**Before Changing Ticks**




**After Changing Ticks**

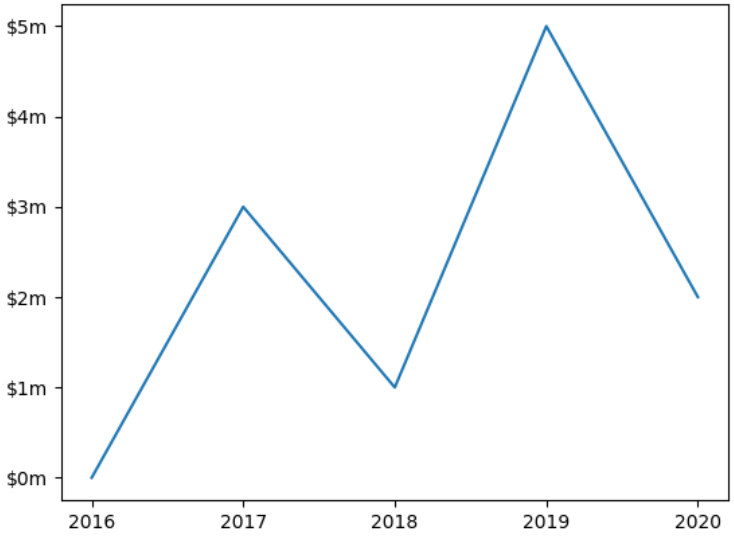


# Line Graph: `title` function

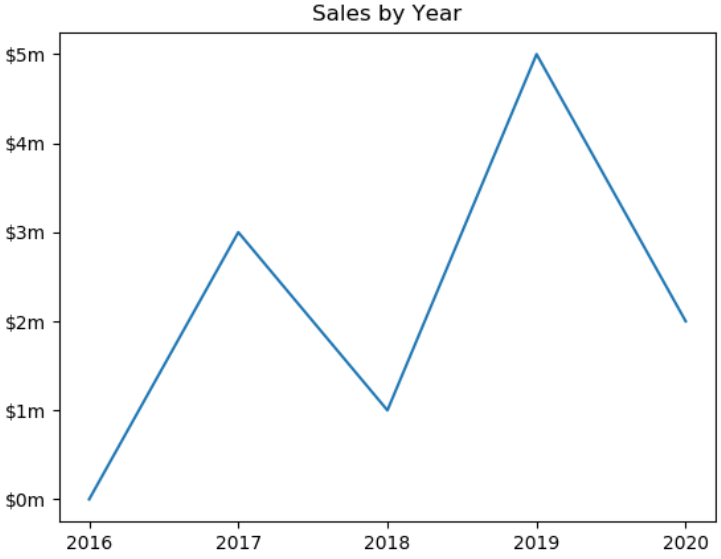
- **You set the title of the plot by using `title` function.**

```
# Add a title.
plt.title('Sales by Year')
```

**Before**




**After**

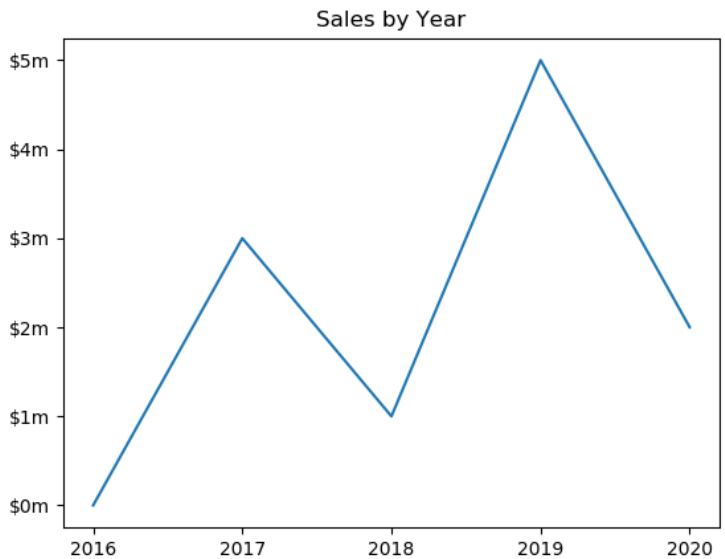


# Line Graph: `x-` and `y-label` functions

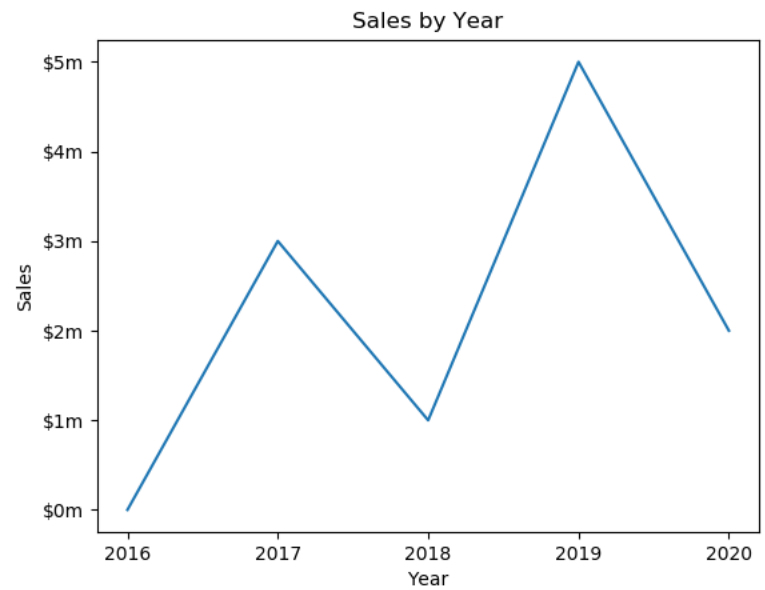
- **Use the x- and y-label functions to display x- and y-labels.**

```
# Add labels to the axes.
plt.xlabel('Year')
plt.ylabel('Sales')
```

**Before**




**After**

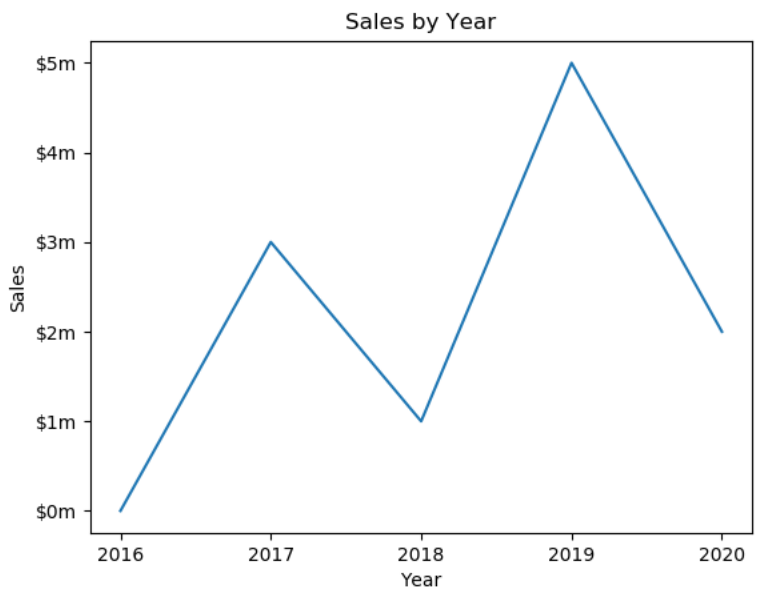


# Line Graph: `grid` function

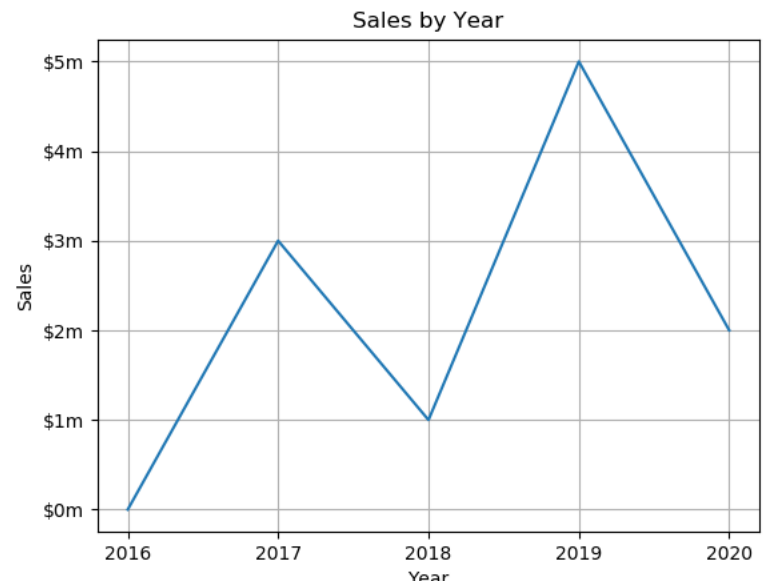
- **You can display gridlines on the the plot by using `grid` function.**

```
# Add a grid.
plt.grid(True)
```

**Before**




**After**

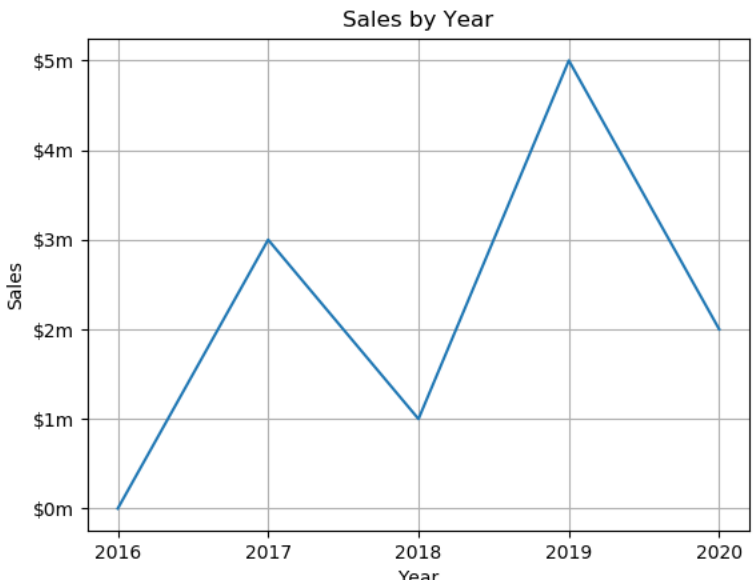


# Line Graph: Displaying Markers at Data Points

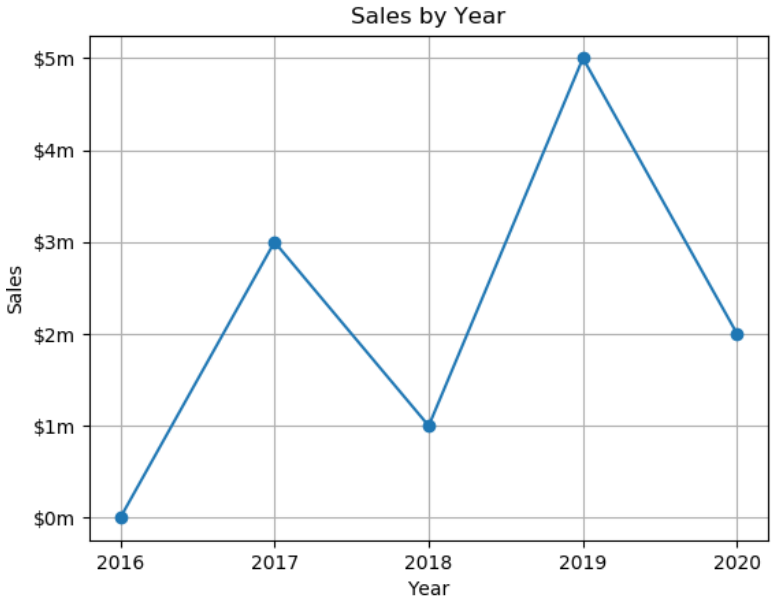
- **You can display a round dot as a marker at each data point in your line graph by using the keyword argument `marker='o'` with the plot function.**

```
# Build the line graph.
plt.plot(x_coords, y_coords, marker='o')
```

**Before**




**After**



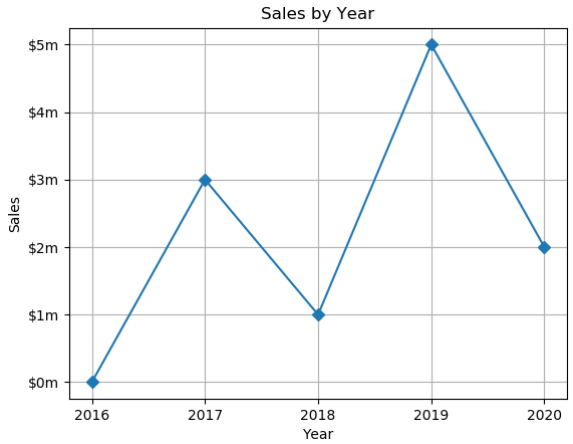
# Line Graph: Other Marker Symbols p413

- **In addition to round dots, you can display other types of marker symbols.**

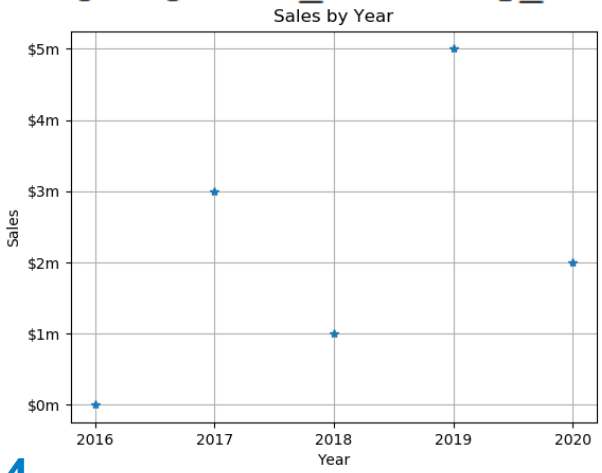
**Table 7-2** Some of the marker symbols

| marker= Argument | Result |
|---|---|
| `marker='o'` | Displays round dots as markers |
| `marker='s'` | Displays squares as markers |
| `marker='*'` | Displays small stars as markers |
| `marker='D'` | Displays small diamonds as markers |
| `marker='^'` | Displays upward triangles as markers |
| `marker='v'` | Displays downward triangles as markers |
| `marker='>'` | Displays right-pointing triangles as markers |
| `marker='<'` | Displays left-pointing triangles as markers |

```
# Build the line graph.
plt.plot(x_coords, y_coords, marker='D')
```




```
# Build the line graph.
plt.plot(x_coords, y_coords,'*')
```

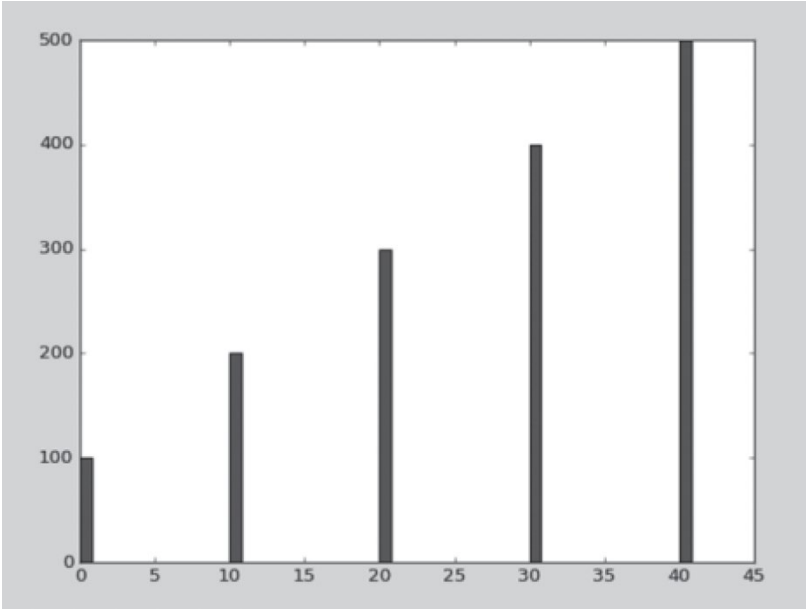



If the marker character is passed to plot function as a positional argument (instead of passing it as a keyword argument), the plot function will draw markers at the data points, but it will not connect them with lines.

See Program 7-24 at Page 414

# Plotting a Bar Chart – `bar` Function

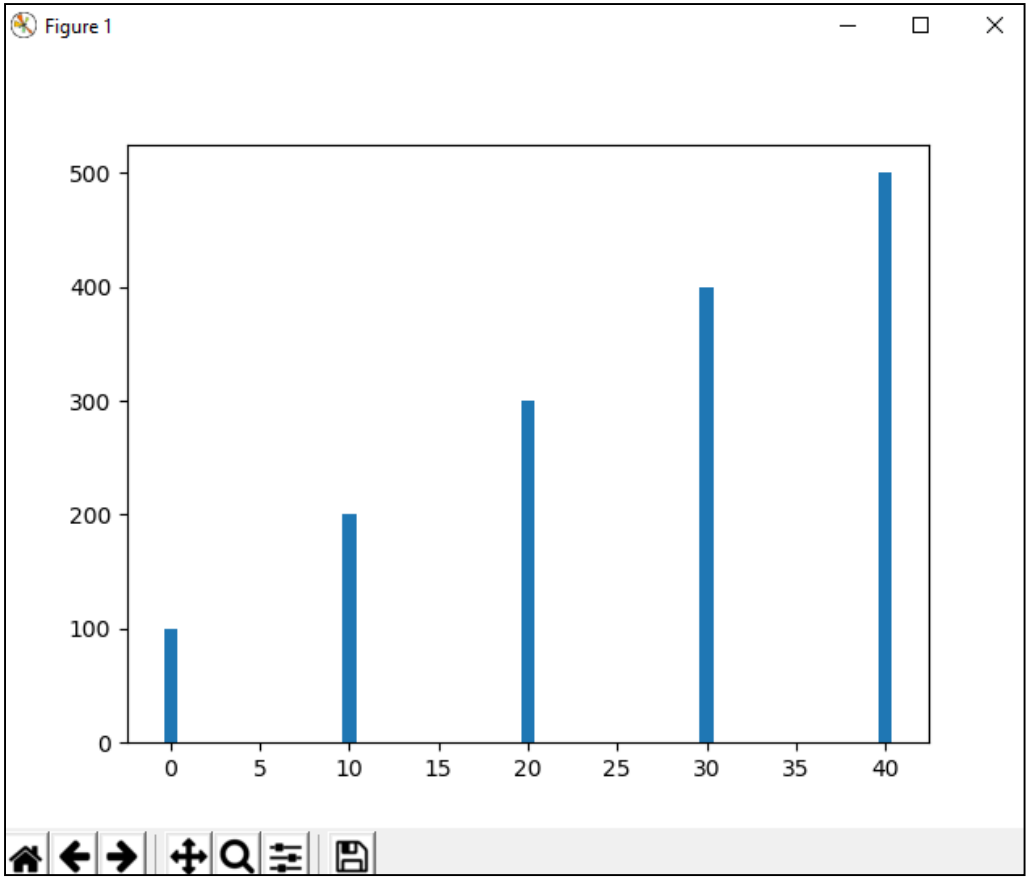
- **the `bar` function in the `matplotlib.pyplot` module can be used to create a bar chart.**




- **The function needs two lists:**
  - the *X* coordinates of each bar's left edge
  - the heights of each bar, along the *Y* axis.

# Plotting a Bar Chart by using `bar` function

```
import matplotlib.pyplot as plt
left_edges = [0, 10, 20, 30, 40]
heights = [100, 200, 300, 400, 500]
plt.bar(left_edges, heights)
plt.show()
```

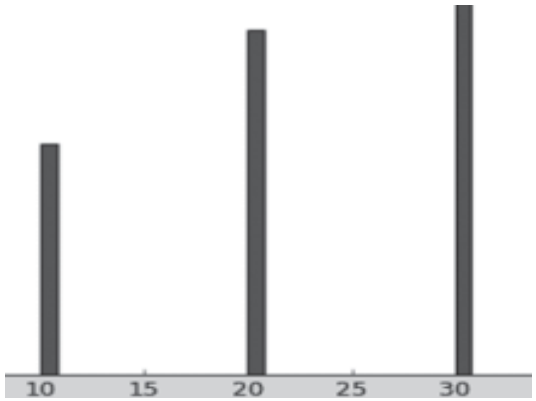



**Note:** **x-values represents the position of the bars in the chart.**

**In Python 3, these values are centered.**

**Book is written according to Python 2 where left edges of the bars are these values.**

**Python 2**

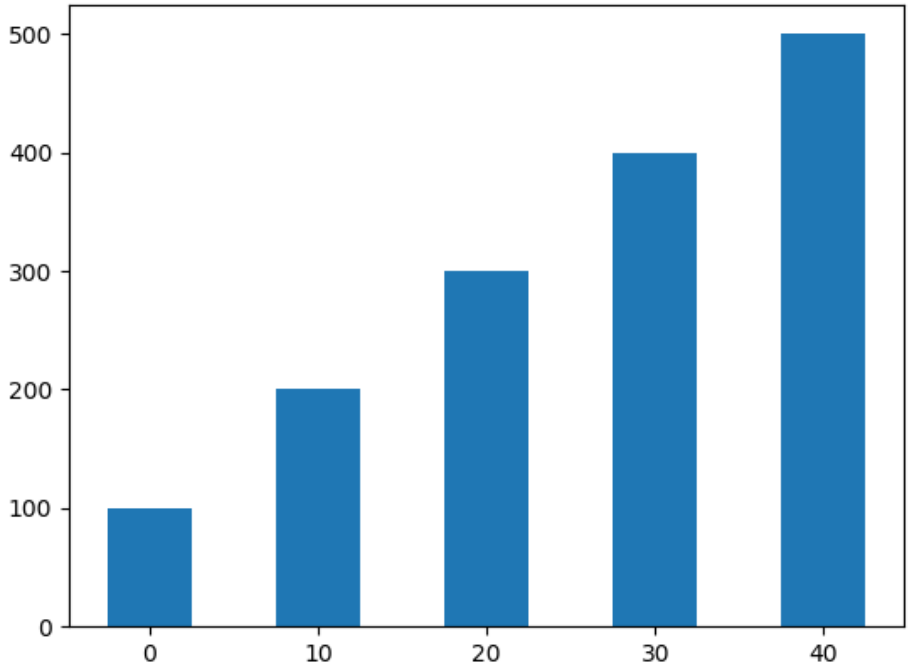


# Plotting a Bar Chart: Bar Width Argument

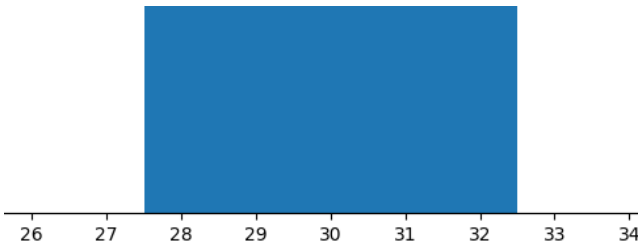
- **The default width of each bar in a bar graph is 0.8 along the *X* axis.**


- **You can change the bar width by passing a third argument to the** `bar` **function.**

```
left_edges = [0, 10, 20, 30, 40]
heights = [100, 200, 300, 400, 500]
bar_width = 5
plt.bar(left_edges, heights, bar_width)
plt.show()
```




**Bar Width is now 5**

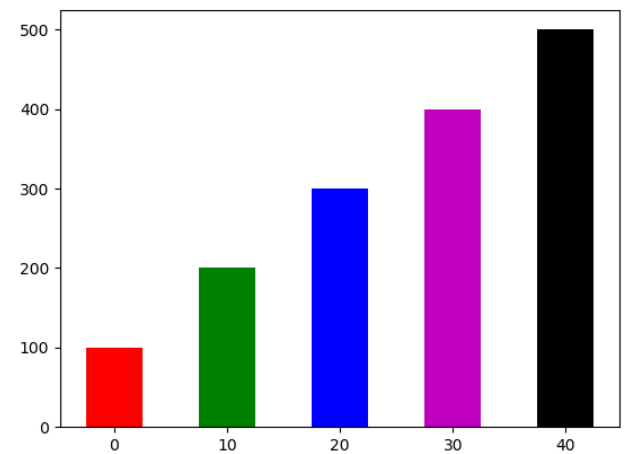


# Plotting a Bar Chart: Color of Bars

- **The `bar` function has a `color` parameter that you can use to change the colors of the bars.**
- **The argument that you pass into this parameter is a <u>tuple</u> containing a series of color codes.**

| Color Code | Corresponding Color |
|---|---|
| 'b' | Blue |
| 'g' | Green |
| 'r' | Red |
| 'c' | Cyan |
| 'm' | Magenta |
| 'y' | Yellow |
| 'k' | Black |
| 'w' | White |

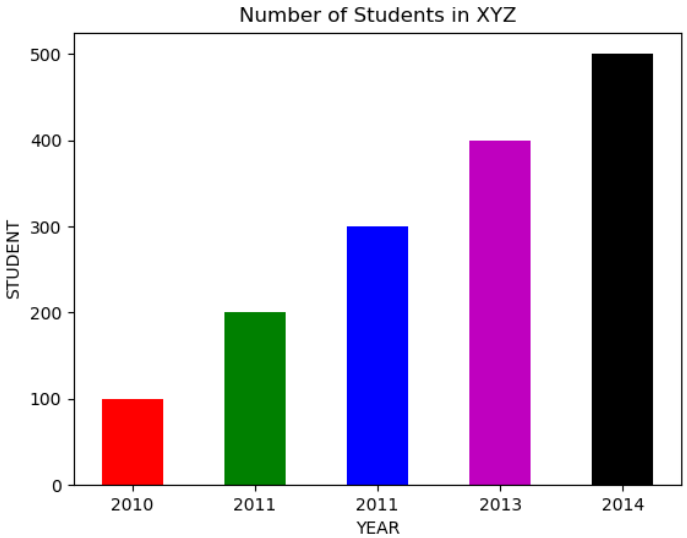
```
x = [0, 10, 20, 30, 40]
y = [100, 200, 300, 400, 500]
bw=5
# Defining Bar Colors as a tuple - bc
bc=('r', 'g', 'b', 'm', 'k')

plt.bar(x, y, bw, color=bc )
plt.show()
```



# Plotting a Bar Chart – Other Functions

- **title, `xlabel`, `ylabel` , `xticks and yticks` functions can be also used for Bar Charts.**

```
x = [0, 10, 20, 30, 40]
y = [100, 200, 300, 400, 500]
#  Defining Bar Colors
bc=('r', 'g', 'b', 'm', 'k')
# Defining a Value for Bar Width
bw=5

#Displaying title for Bar Chart
plt.title('Number of Students in XYZ')
#Changing x- and y labels
plt.xlabel('YEAR')
plt.ylabel('STUDENT')
#Changing x- and y- ticks
plt.xticks([0, 10, 20, 30, 40],
       ['2010', '2011', '2011', '2013', '2014'])


# Build the bar chart.
plt.bar(x, y, bw, color=bc)

# Display the bar chart
plt.show()
```

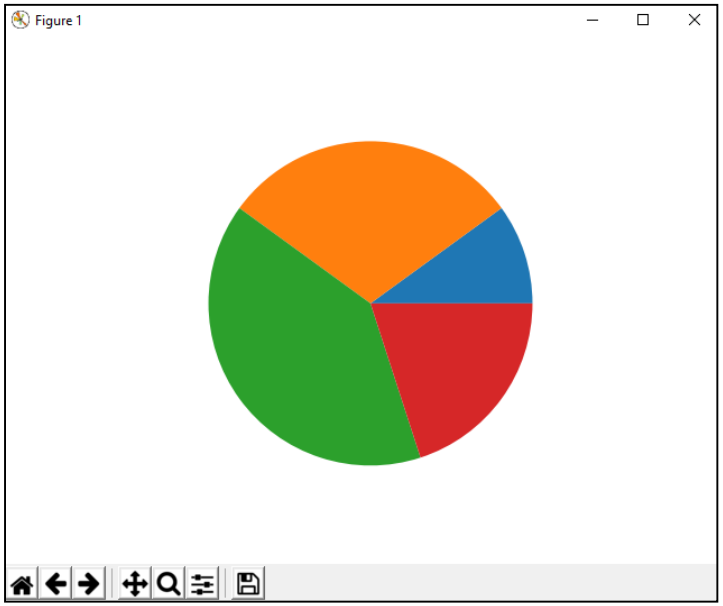


# Plotting a Pie Chart

- **You can use the `pie` function available in the `matplotlib.pyplot` module to create a pie chart.**
- **When you call the `pie` function, you pass a list of values as an argument.**
  - The sum of the values will be used as the value of the whole.
  - Each element in the list will become a slice in the pie chart.
  - The size of a slice represents that element's value as a percentage of the whole.

**Example:**

```
values = [20, 60, 80, 40]
plt.pie(values)
plt.show()
```

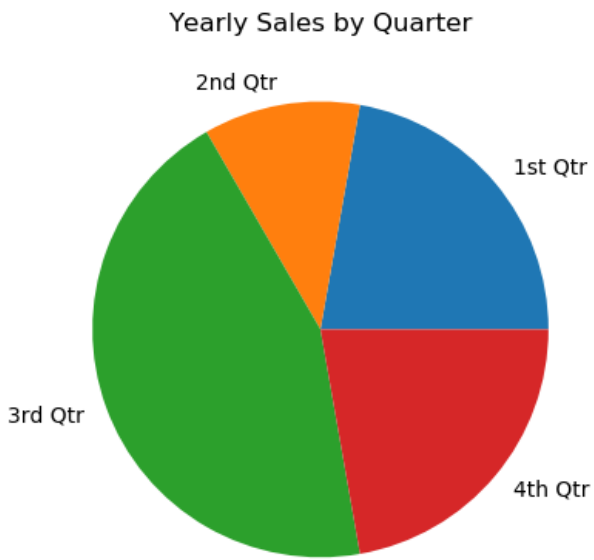


# Plotting a Pie Chart – Slice Labels

- **The `pie` function has a `labels` parameter that you can use to display labels for the slices in the pie chart.**
- **The argument that you pass into this parameter is a list containing the desired labels, as strings.**

## Example:

```
sales = [200, 100, 400, 200]
slice_labels = ['1st Qtr', '2nd Qtr', '3rd Qtr', '4th Qtr']
plt.pie(sales, labels=slice_labels)
plt.title('Yearly Sales by Quarter')
plt.show()
```

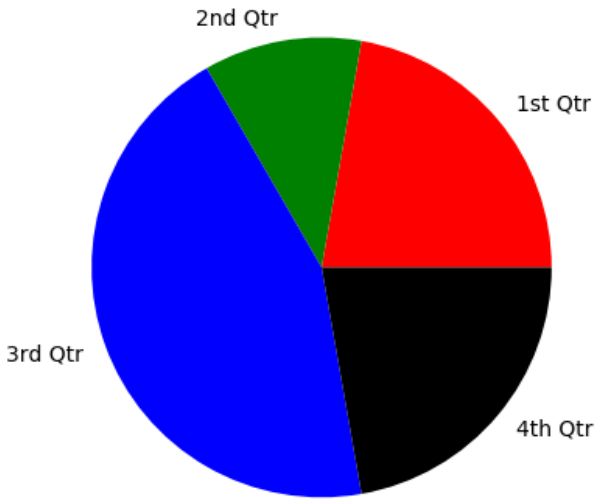


# Plotting a Pie Chart – Slice Colors

- **The `pie` function automatically changes the color of the slices, in the following order:**
  - blue, green, red, cyan, magenta, yellow, black, and white.

- **You can specify a different set of colors, however, by passing a tuple of color codes as an argument to the `pie` function's `colors` parameter:**

```
slice_labels = ['1st Qtr', '2nd Qtr', '3rd Qtr', '4th Qtr']
slice_colors=('r', 'g', 'b', 'k')
plt.pie(sales, labels=slice_labels, colors=slice_colors)
```



**7.26** To create a graph with the plot function, what two arguments you must pass? **X values and Y values as list**

**7.27** What sort of graph does the plot function produce?

**Line graphs**

**7.28** What functions do you use to add labels to the *X* and *Y* axes in a graph? **`xlabel and ylabel`**

**7.29** How do you change the lower and upper limits of the *X* and *Y* axes in a graph? **By using `xlim and ylim functions.`**

**7.30** How do you customize the tick marks along the *X* and *Y* axes in a graph? **By using `xticks and yticks functions.`**

**7.31** To create a bar chart with the bar function, what two arguments you must pass? **X values and Y values (heights of the bars).**

**7.32** Assume the following statement calls the bar function to construct a bar chart with four bars. What color will the bars be?

plt.bar(x_values, y_values, color=('r', 'b', 'r', 'b')) **–red-blue-red-blue**

**7.33** To create a pie chart with the pie function, what argument you must pass? **values to be plotted must be passed as a list.**

# Summary

- **This chapter covered:**
  - Lists, including:
    - Repetition and concatenation operators
    - Indexing
    - Techniques for processing lists
    - Slicing and copying lists
    - List methods and built-in functions for lists
    - Two-dimensional lists
  - Tuples, including:
    - Immutability
    - Difference from and advantages over lists
  - Plotting charts and graphs with the **`matplotlib`** Package