Lists and Tuples

陳建良

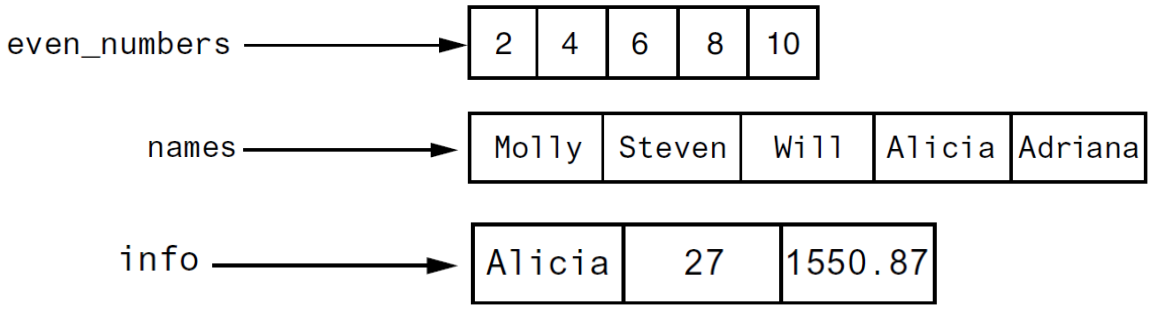
Sequences

- A sequence is an object that holds multiple items of data, stored one after the other.
- Python provides various ways to perform operations on the items that are stored in a sequence.
- There are several different types of sequence objects in Python.
- Two of the fundamental sequence types: lists and tuples.
- Both lists and tuples are sequences that can hold various types of data.
- The difference between lists and tuples is simple:
  - a list is mutable, which means that a program can change its contents,
  - but a tuple is immutable, which means that once it is created, its contents cannot be changed.

Introduction to Lists

- A *list* is an object that contains multiple data items. Each item that is stored in a list is called an *element*.

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

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

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



- Python also has a built-in list() function that can convert certain types of objects to lists.

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

- The range function is called with 5 passed as an argument. The function returns an iterable containing the values 0, 1, 2, 3, 4.
- The iterable is passed as an argument to the list() function. The list() function returns the list [0, 1, 2, 3, 4].
- The list [0, 1, 2, 3, 4] is assigned to the numbers variable.
- Here is another example:

```
numbers = list(range(1, 10, 2))
```

The Repetition Operator

- The repetition operator makes multiple copies of a list and joins them all together. Here is the general format:

list * *n*

```
1  >>> numbers = [0] * 5 Enter
2  >>> print(numbers) Enter
3  [0, 0, 0, 0, 0]
4  >>>
```

```
1  >>> numbers = [1, 2, 3] * 3 Enter
2  >>> print(numbers) Enter
3  [1, 2, 3, 1, 2, 3, 1, 2, 3]
4  >>>
```

Iterating over a List with the for Loop

```
numbers = [99, 100, 101, 102]
for n in numbers:
    print(n)
```

Indexing

- Index: 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.

- An IndexError exception will be raised if you use an invalid index with a list. For example,

```
#This code will cause an IndexError exception.
my_list = [10, 20, 30, 40]
index = 0
while index < 5:
    print(my_list[index])
    index += 1
```

The len Function

- Python has a built-in function named len that returns the length of a sequence, such as a list.

```
my_list = [10, 20, 30, 40]
size = len(my_list)
```

- The len function can be used to prevent an IndexError exception when iterating over a list with a loop. Here is an example:

```
my_list = [10, 20, 30, 40]
index = 0
while index < len(my_list):
    print(my_list[index])
    index += 1
```

Lists Are Mutable

- Lists in Python are *mutable*, which means their elements can be changed.
- Consequently, an expression in the form *list*[*index*] can appear on the left side of an assignment operator.

```
1  numbers = [1, 2, 3, 4, 5]
2  print(numbers)
3  numbers[0] = 99
4  print(numbers)
```

```
1   # The NUM_DAYS constant holds the number of
2   # days that we will gather sales data for.
3   NUM_DAYS = 5
4
5   def main():
6       # Create a list to hold the sales
7       # for each day.
8       sales = [0] * NUM_DAYS
9
10      # Create a variable to hold an index.
11      index = 0
12
13      print('Enter the sales for each day.')
14
15      # Get the sales for each day.
16      while index < NUM_DAYS:
17          print('Day #', index + 1, ': ', sep='', end='')
18          sales[index] = float(input())
19          index += 1
20
21      # Display the values entered.
22      print('Here are the values you entered:')
23      for value in sales:
24          print(value)
25
26  # Call the main function.
27  main()
```

Program Output (with input shown in bold)

```
Enter the sales for each day.
Day #1: 1000 Enter
Day #2: 2000 Enter
Day #3: 3000 Enter
Day #4: 4000 Enter
Day #5: 5000 Enter
Here are the values you entered:
1000.0
2000.0
3000.0
4000.0
5000.0
```

Concatenating Lists

- To concatenate means to join two things together. You can use the + operator to concatenate two lists.

  list1 = [1, 2, 3, 4]

  list2 = [5, 6, 7, 8]

  list3 = list1 + list2

```
>>> girl_names = ['Joanne', 'Karen', 'Lori'] Enter
>>> girl_names += ['Jenny', 'Kelly'] Enter
>>> print(girl_names) Enter
['Joanne', 'Karen', 'Lori', 'Jenny', 'Kelly']
>>>
```

List Slicing

- In Python, you can write expressions that select subsections of a sequence, known as slices.
- A *slice* is a span of items that are taken from a sequence. To get a slice of a list, you write an expression in the following general format:

list_name[*start* : *end*]

- days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
- mid_days = days[2:5]

```
1  >>> numbers = [1, 2, 3, 4, 5] Enter
2  >>> print(numbers) Enter
3  [1, 2, 3, 4, 5]
4  >>> print(numbers[1:3]) Enter
5  [2, 3]
6  >>>
```

```
1  >>> numbers = [1, 2, 3, 4, 5] Enter
2  >>> print(numbers) Enter
3  [1, 2, 3, 4, 5]
4  >>> print(numbers[:3]) Enter
5  [1, 2, 3]
6  >>>
```

- Because the starting index was omitted, the slice contains the elements from index 0 up to 3.

```
1  >>> numbers = [1, 2, 3, 4, 5] Enter
2  >>> print(numbers) Enter
3  [1, 2, 3, 4, 5]
4  >>> print(numbers[2:]) Enter
5  [3, 4, 5]
6  >>>
```

- Because the ending index was omitted, the slice contains the elements from index 2 through the end of the list.

```
1  >>> numbers = [1, 2, 3, 4, 5] Enter
2  >>> print(numbers) Enter
3  [1, 2, 3, 4, 5]
4  >>> print(numbers[:]) Enter
5  [1, 2, 3, 4, 5]
6  >>>
```

- If you leave out both the *start* and *end* index in a slicing expression, you get a copy of the entire list.

```
1  >>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] Enter
2  >>> print(numbers) Enter
3  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
4  >>> print(numbers[1:8:2]) Enter
5  [2, 4, 6, 8]
6  >>>
```

```
1  >>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] Enter
2  >>> print(numbers) Enter
3  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
4  >>> print(numbers[-5:]) Enter
5  [6, 7, 8, 9, 10]
6  >>>
```

Finding Items in Lists with the in Operator

- You can search for an item in a list using the in operator.
- Here is the general format of an expression written with the in operator to search for an item in a list:

  item in *list*

- The expression returns true if *item* is found in the *list*, or false otherwise.

```
1  # This program demonstrates the in operator
2  # used with a list.
3
4  def main():
5      # Create a list of product numbers.
6      prod_nums = ['V475', 'F987', 'Q143', 'R688']
7
8      # Get a product number to search for.
9      search = input('Enter a product number: ')
10
11     # Determine whether the product number is in the list.
12     if search in prod_nums:
13         print(search, 'was found in the list.')
14     else:
15         print(search, 'was not found in the list.')
16
17 # Call the main function.
18 main()
```

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.
```

List Methods and Useful Built-in Functions

- Lists have numerous methods that allow you to add elements, remove elements, change the ordering of elements, and so forth.

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.

Program Output (with input shown in bold)

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

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

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

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

Here are the names you entered.
Kathryn
Chris
Kenny
Renee
```

```
# 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 == '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 names you entered.')

    for name in name_list:
        print(name)

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

Program Output (with input shown in bold)

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

```
# 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.
    food = ['Pizza', 'Burgers', 'Chips']

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

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

    try:
        # Get the item's index in the list.
        item_index = food.index(item)

        # Get the value to replace it with.
        new_item = input('Enter the new value: ')

        # Replace the old item with the new item.
        food[item_index] = new_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()
```

```
1   # This program demonstrates the insert method.
2
3   def main():
4       # Create a list with some names.
5       names = ['James', 'Kathryn', 'Bill']
6
7       # Display the list.
8       print('The list before the insert:')
9       print(names)
10
11      # Insert a new name at element 0.
12      names.insert(0, 'Joe')
13
14      # Display the list again.
15      print('The list after the insert:')
16      print(names)
17
18  # Call the main function.
19  main()
```

Program Output

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

```
1   # This program demonstrates how to use the remove
2   # method to remove an item from a list.
3
4   def main():
5       # Create a list with some items.
6       food = ['Pizza', 'Burgers', 'Chips']
7
8       # Display the list.
9       print('Here are the items in the food list:')
10      print(food)
11
12      # Get the item to change.
13      item = input('Which item should I remove? ')
14
15      try:
16          # Remove the item.
17          food.remove(item)
18
19          # Display the list.
20          print('Here is the revised list:')
21          print(food)
22
23      except ValueError:
24         print('That item was not found in the list.')
25
26  # Call the main function.
27  main()
```

Program Output (with input shown in bold)

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

The del Statement

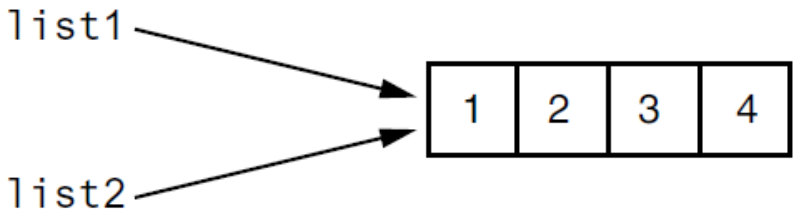
```
my_list = [1, 2, 3, 4, 5]
print('Before deletion:', my_list)
del my_list[2]
print('After deletion:', my_list)
```

The min and max Functions

```
my_list = [5, 4, 3, 2, 50, 40, 30]
print('The lowest value is', min(my_list))
print('The highest value is', max(my_list))
```

Copying Lists

```
# Create a list.
list1 = [1, 2, 3, 4]
# Assign the list to the list2 variable.
list2 = list1
```



```
1   >>> list1 = [1, 2, 3, 4] Enter
2   >>> list2 = list1 Enter
3   >>> print(list1) Enter
4   [1, 2, 3, 4]
5   >>> print(list2) Enter
6   [1, 2, 3, 4]
7   >>> list1[0] = 99 Enter
8   >>> print(list1) Enter
9   [99, 2, 3, 4]
10  >>> print(list2) Enter
11  [99, 2, 3, 4]
12  >>>
```

A simpler way

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

```
# 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)
```

Processing Lists

- Megan owns a small neighborhood coffee shop, and she has six employees who work as baristas (coffee bartenders). All of the employees have the same hourly pay rate. Megan has asked you to design a program that will allow her to enter the number of hours worked by each employee, then display the amounts of all the employees' gross pay. You determine the program should perform the following steps:
  1. For each employee: get the number of hours worked and store it in a list element.
  2. For each list element: use the value stored in the element to calculate an employee's gross pay. Display the amount of the gross pay.

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
```

```
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()
```

Totaling the Values in a List

```
1   # This program calculates the total of the values
2   # in a list.
3
4   def main():
5       # Create a list.
6       numbers = [2, 4, 6, 8, 10]
7
8       # Create a variable to use as an accumulator.
9       total = 0
10
11      # Calculate the total of the list elements.
12      for value in numbers:
13          total += value
14
15      # Display the total of the list elements.
16      print('The total of the elements is', total)
17
18  # Call the main function.
19  main()
```

Program Output

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

Averaging the Values in a List

```
1   # This program calculates the average of the values
2   # in a list.
3
4   def main():
5       # Create a list.
6       scores = [2.5, 7.3, 6.5, 4.0, 5.2]
7
8       # Create a variable to use as an accumulator.
9       total = 0.0
10
11      # Calculate the total of the list elements.
12      for value in scores:
13          total += value
14
15      # Calculate the average of the elements.
16      average = total / len(scores)
17
18      # Display the total of the list elements.
19      print('The average of the elements is', average)
20
21  # Call the main function.
22  main()
```

Program Output

```
The average of the elements is 5.3
```

Passing a List as an Argument to a Function

```
1   # This program uses a function to calculate the
2   # total of the values in a list.
3
4   def main():
5       # Create a list.
6       numbers = [2, 4, 6, 8, 10]
7
8       # Display the total of the list elements.
9       print('The total is', get_total(numbers))
10
11  # The get_total function accepts a list as an
12  # argument returns the total of the values in
13  # the list.
14  def get_total(value_list):
15      # Create a variable to use as an accumulator.
16      total = 0
17
18      # Calculate the total of the list elements.
19      for num in value_list:
20          total += num
21
22      # Return the total.
23      return total
24
25  # Call the main function.
26  main()
```

Program Output

```
The total is 30
```

Returning a List from a Function

```
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():
    # Create an empty list.
    values = []

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

    # Get values from the user and add them to
    # the list.
    while again == '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: ')
        print()

    # Return the list.
    return values

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

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]
```

- Dr. LaClaire gives a series of exams during the semester in her chemistry class. At the end of the semester, she drops each student's lowest test score before averaging the scores. She has asked you to design a program that will read a student's test scores as input and calculate the average with the lowest score dropped. Here is the algorithm that you developed:

1. *Get the student's test scores.*
2. *Calculate the total of the scores.*
3. *Find the lowest score.*
4. *Subtract the lowest score from the total. This gives the adjusted total.*
5. *Divide the adjusted total by 1 less than the number of test scores. This is the average.*
6. *Display the average.*

Program Output (with input shown in bold)

```
Enter a test score: 92 Enter
Do you want to add another score?
Y = yes, anything else = no: y Enter

Enter a test score: 67 Enter
Do you want to add another score?
Y = yes, anything else = no: y Enter

Enter a test score: 75 Enter
Do you want to add another score?
Y = yes, anything else = no: y Enter

Enter a test score: 88 Enter
Do you want to add another score?
Y = yes, anything else = no: n Enter

The average, with the lowest score dropped is: 85.0
```

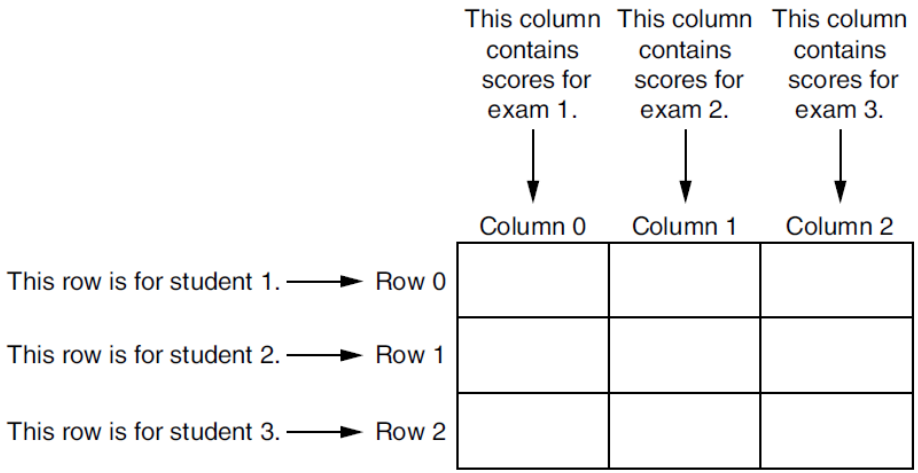
Two-Dimensional Lists

- A two-dimensional list is a list that has other lists as its elements.

```
1  >>> students = [['Joe', 'Kim'], ['Sam', 'Sue'], ['Kelly', 'Chris']] Enter
2  >>> print(students) Enter
3  [['Joe', 'Kim'], ['Sam', 'Sue'], ['Kelly', 'Chris']]
4  >>> print(students[0]) Enter
5  ['Joe', 'Kim']
6  >>> print(students[1]) Enter
7  ['Sam', 'Sue']
8  >>> print(students[2]) Enter
9  ['Kelly', 'Chris']
10 >>>
```

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

```
scores = [[0, 0, 0],
          [0, 0, 0],
          [0, 0, 0]]
```



	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]`

```
1  # This program assigns random numbers to
2  # a two-dimensional list.
3  import random
4
5  # Constants for rows and columns
6  ROWS = 3
7  COLS = 4
8
9  def main():
10     # Create a two-dimensional list.
11     values = [[0, 0, 0, 0],
12               [0, 0, 0, 0],
13               [0, 0, 0, 0]]
14
15     # Fill the list with random numbers.
16     for r in range(ROWS):
17         for c in range(COLS):
18             values[r][c] = random.randint(1, 100)
19
20     # Display the random numbers.
21     print(values)
22
23 # Call the main function.
24 main()
```

Program Output

```
[[4, 17, 34, 24], [46, 21, 54, 10], [54, 92, 20, 100]]
```

Tuples

- A *tuple* is a sequence, very much like a list.
- The primary difference between tuples and lists is that tuples are **immutable**.
- That means once a tuple is created, it cannot be changed.

```
>>> my_tuple = (1, 2, 3, 4, 5) Enter
>>> print(my_tuple) Enter
(1, 2, 3, 4, 5)
>>>
>>> names = ('Holly', 'Warren', 'Ashley') Enter
>>> for n in names: Enter
        print(n) Enter Enter
Holly
Warren
Ashley
>>>
```

```
>>> names = ('Holly', 'Warren', 'Ashley') Enter
>>> for i in range(len(names)): Enter
        print(names[i]) Enter Enter

Holly
Warren
Ashley
>>>
```

- Tuples support all the same operations as lists, except those that change the contents of the list. Tuples support the following:
  1. Subscript indexing (for retrieving element values only)
  2. Methods such as index
  3. Built-in functions such as len, min, and max
  4. Slicing expressions
  5. The in operator
  6. The + and * operators
- Tuples do not support methods such as **append**, **remove**, **insert**, **reverse**, and **sort**.

Converting Between Lists and Tuples

```
>>> number_tuple = (1, 2, 3) Enter
>>> number_list = list(number_tuple) Enter
>>> print(number_list) Enter
[1, 2, 3]
>>> str_list = ['one', 'two', 'three'] Enter
>>> str_tuple = tuple(str_list) Enter
>>> print(str_tuple) Enter
('one', 'two', 'three')
>>>
```

Plotting List Data With the matplotlib Package

- The matplotlib package is a library for creating two-dimensional charts and graphs.
- It is not part of the standard Python library, so you will have to install it.
- To install matplotlib, You can enter the following command:

  pip install matplotlib

- Starting IDLE and entering the command

  >>> import matplotlib

Importing the pyplot Module

- The matplotlib package contains a module named pyplot that you will need to import in order to create all of the graphs.

```
import matplotlib.pyplot

matplotlib.pyplot.plot(arguments...)
```

- Or

```
import matplotlib.pyplot as plt

plt.plot(arguments...)
```

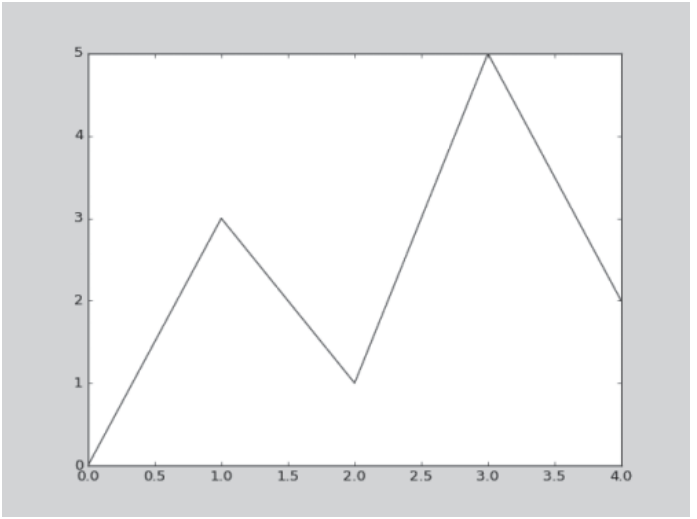
Plotting a Line Graph

- For example, suppose we have five data points, located at the following coordinates:

  (0, 0) 、(1, 3) 、(2, 1) 、(3, 5) 、(4, 2)

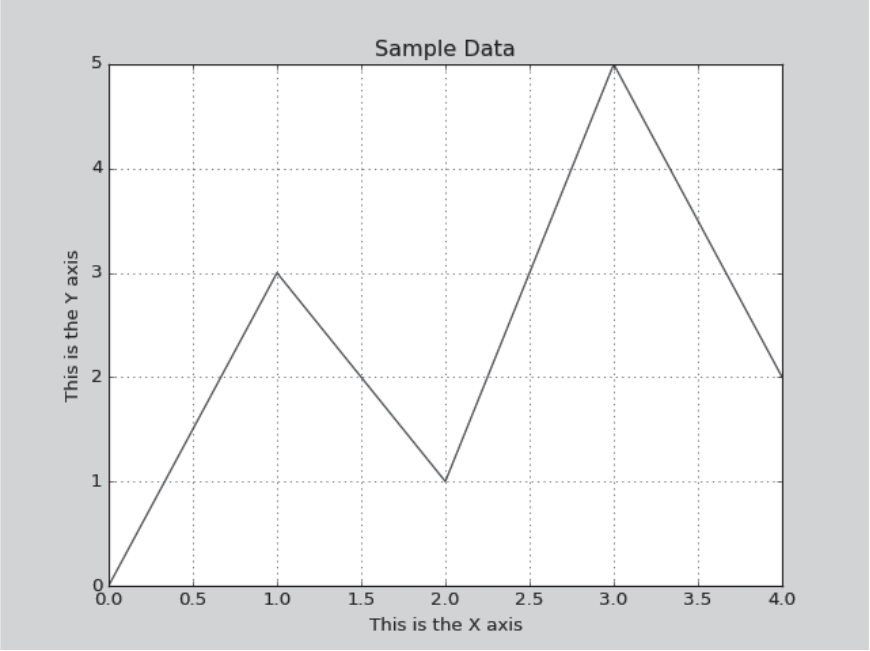
- code

```
x_coords = [0, 1, 2, 3, 4]
y_coords = [0, 3, 1, 5, 2]
plt.plot(x_coords, y_coords)
plt.show()
```



Adding a Title, Axis Labels, and a Grid

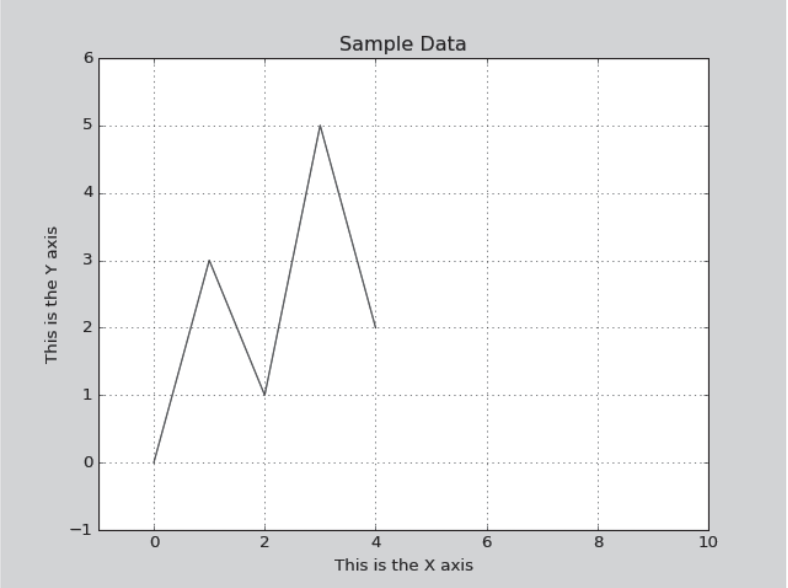
```
# Add a title.
plt.title('Sample Data')
# Add labels to the axes.
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
# Add a grid.
plt.grid(True)
```



Customizing the *X* and *Y* Axes

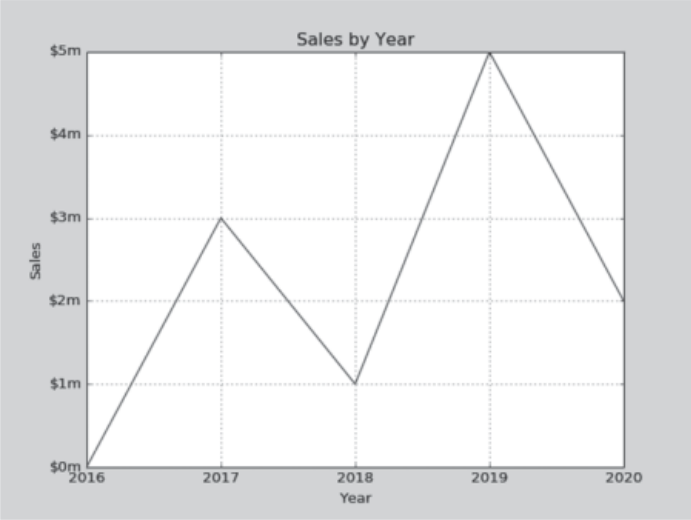
- You can change the lower and upper limits of the *X* and *Y* axes by calling the **xlim** and **ylim** functions.

```
# Set the axis limits.

plt.xlim(xmin=-1, xmax=10)

plt.ylim(ymin=-1, ymax=6)
```



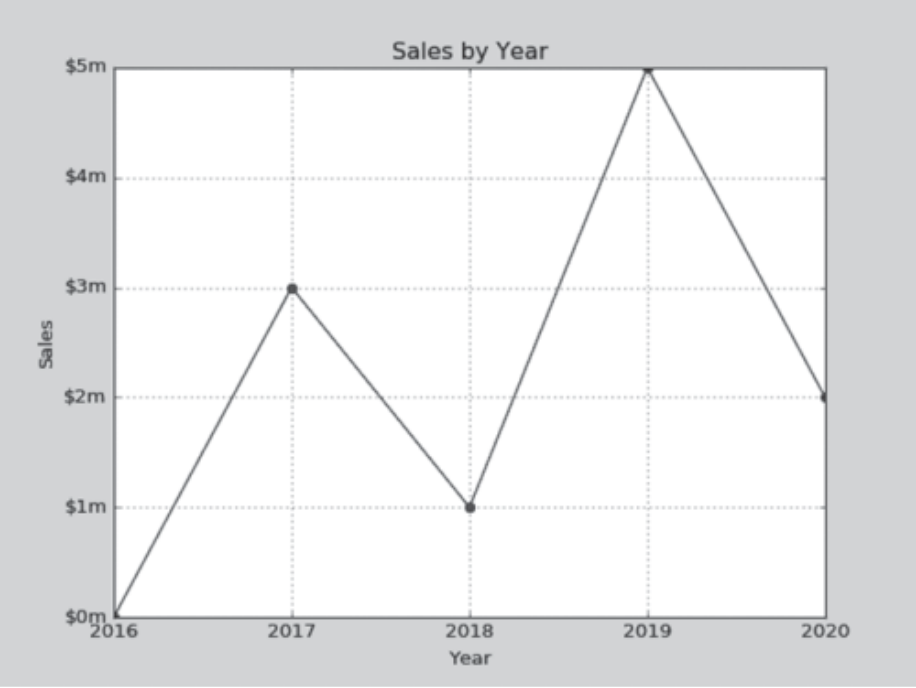
- You can customize each tick mark's label with the **xticks** and **yticks** functions.

```
# Customize the tick marks.
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'])
```



Displaying Markers at the Data Points

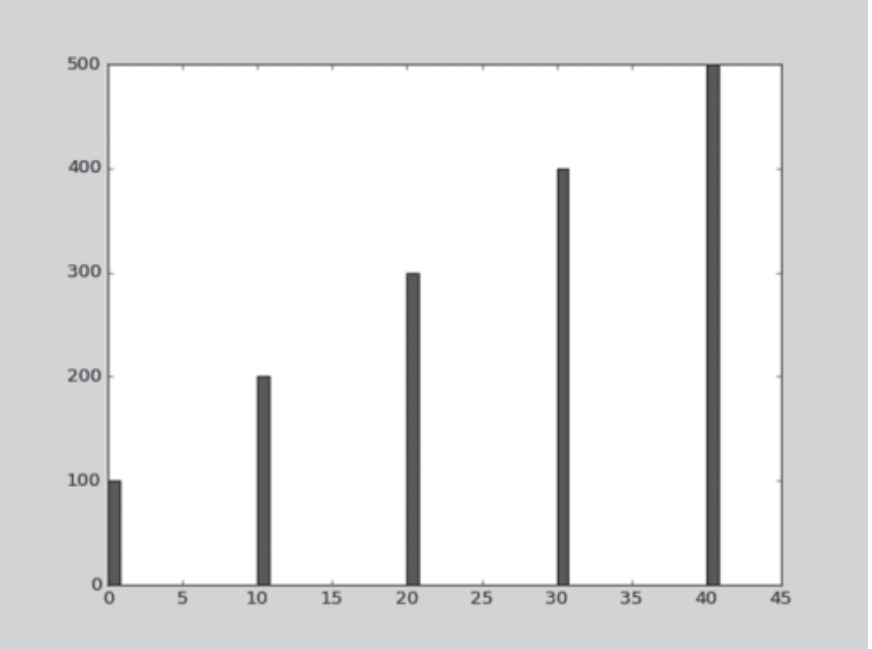
```
# Build the line graph.

plt.plot(x_coords, y_coords, marker='o')
```



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

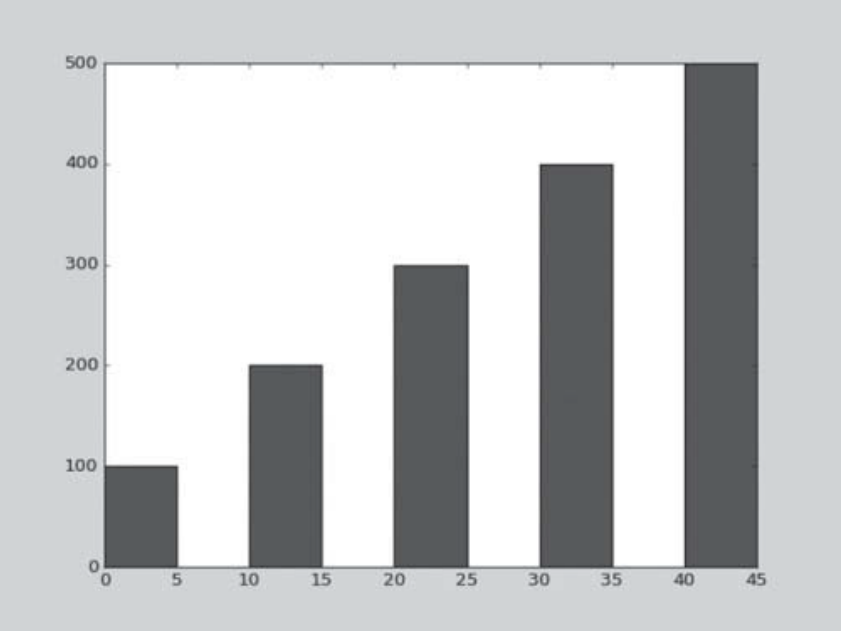
Plotting a Bar Chart

```
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()
```



Customizing the Bar Width

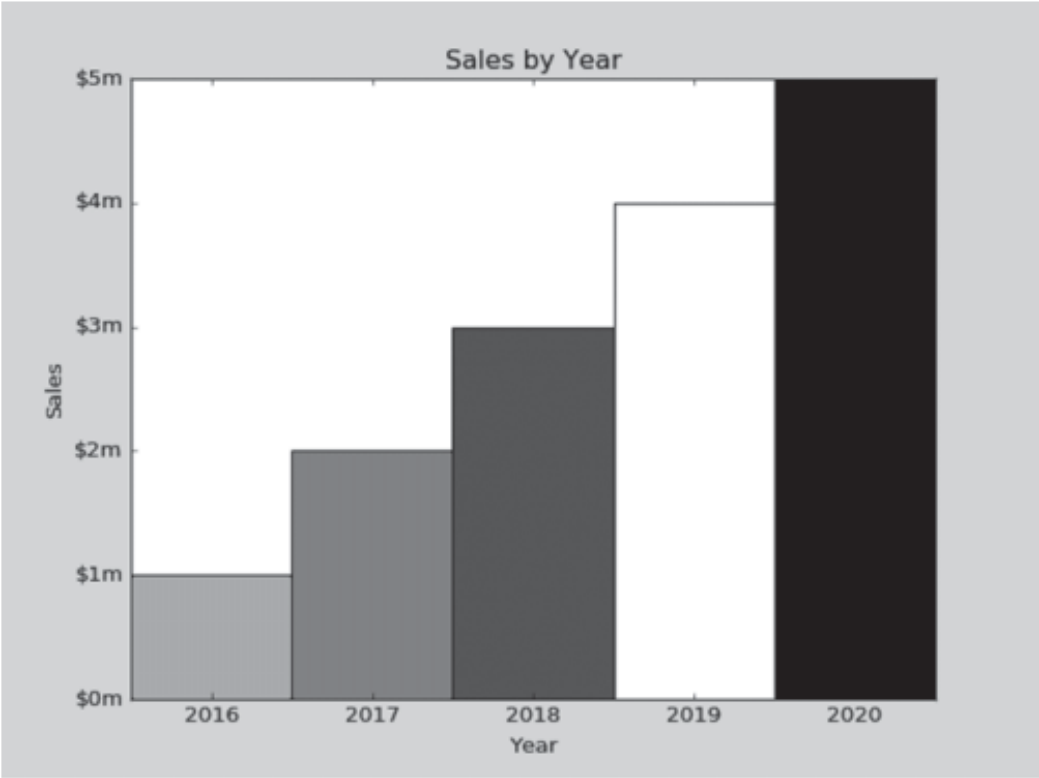
- bar_width = 5
- plt.bar(left_edges, heights, bar_width)



Changing the Colors of the Bars

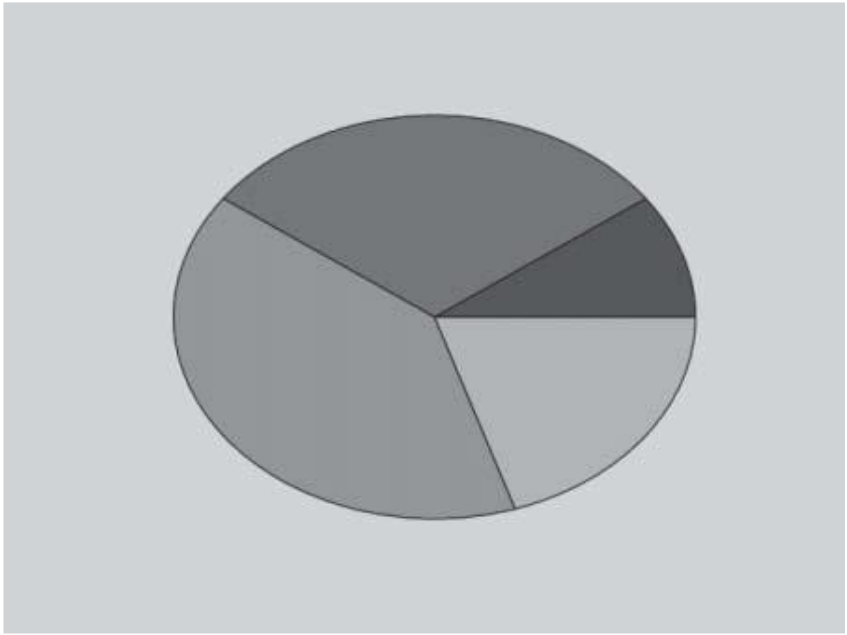
Color Code	Corresponding Color
`'b'`	Blue
`'g'`	Green
`'r'`	Red
`'c'`	Cyan
`'m'`	Magenta
`'y'`	Yellow
`'k'`	Black
`'w'`	White

- plt.bar(left_edges, heights, bar_width, color=('r', 'g', 'b', 'y', 'k'))



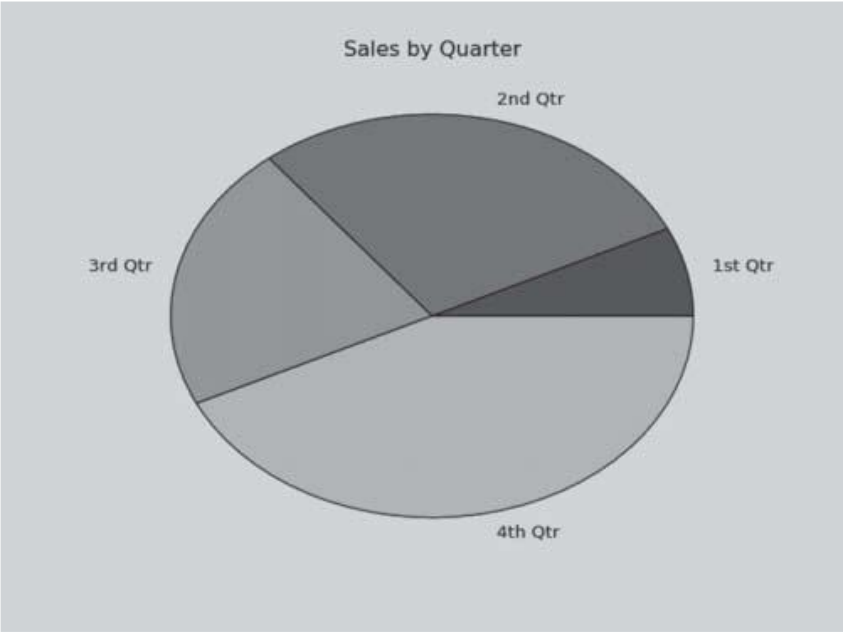
Plotting a Pie Chart

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



Displaying Slice Labels and a Chart Title

```
import matplotlib.pyplot as plt
sales = [100, 400, 300, 600]
slice_labels = ['1st Qtr', '2nd Qtr', '3rd Qtr', '4th Qtr']
plt.pie(sales, labels=slice_labels)
plt.title('Sales by Quarter')
plt.show()
```



Changing the Colors of the Slices

- plt.pie(values, colors=('r', 'g', 'b', 'w', 'k'))