

**Python Object Oriented Programming**

Assignment 4 - Project

# Python Object Oriented Programming

### Software Development

|  |  |
| --- | --- |
| image001.png | Python Object Oriented Programming |

### Pledge of Honour

You are required to include the following as documentation at the beginning of every assessment:

Student name: first\_name Last\_name

Student ID: 123456789

Pledge of Honour: I pledge by honour that this program is solely my own work.

### Plagiarism

Any course work presented for assessment must be your own work. Copying or paraphrasing someone else’s work be it published, unpublished or off the internet, without clearly acknowledging it constitutes plagiarism and is considered to be academic misconduct. You are required to sign an assignment declaration stating it is your own work. You may receive a zero for part or all of the assessment submitted in first instance. Repeated incidents of plagiarism or cheating could result in you being removed from the course or the programme.

### Marking Criteria of Assignment 4 - Project

This is how the questions are marked in Assignment 4 - Project:

To achieve the full marks on each question, your program must be complete and work correctly. This means:

* 1. The program must do what the question says.
  2. The program must make the same output as the Example Output.

Your program solutions need to have these documentation comments at the beginning:

* 1. Student name and ID
  2. Pledge of Honour declaration

Your program should use the Programming Best Practices at the end of this document.

### Assignment 4 - Project

**Learning outcomes:** The course work questions cover Learning Outcomes 1 – 3

**Aims:** The course work questions aim to practice these topics:

* Basic input and output operations
* Coding standards
* Debugging and testing by using features of Integrated Development Environment (IDE)
* The use of methods (both static and instance) and parameters in solving problems
* Design and implement classes and objects
* public and private class fields
* constructors, public, and private methods
* Modular programming
* Problem solving using classes and objects
* Derived classes and protected variables and methods: Examples and problem solving
* Polymorphism and its advantages in software development: Examples and problem solving
* Using object oriented programming to develop applications
* Object oriented programming and code reuse
* List of objects
* Object persistence with file I/O
* Graphical User Interfaces (GUI) and User Interactions

**Weighting:** 20%

**Marking allocation:** see page XX

**Due date:** see timetable on Moodle

**Marking process:**

Upload your solutions (as a zip file) to Moodle.

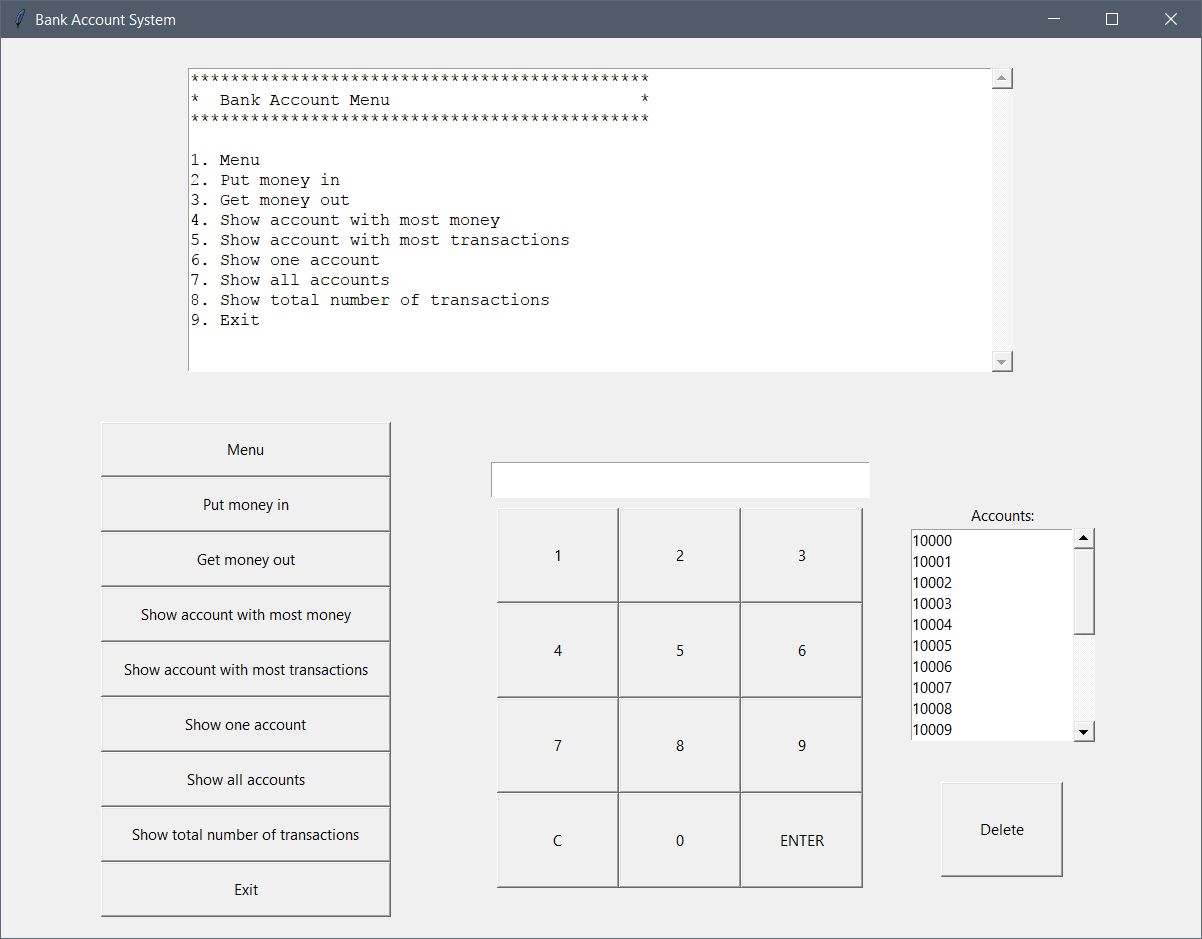
**There are 2 parts of this project. You will design and write a Python program, and you will make a presentation to the class at the end of the assignment.**

**This assignment will be done in groups of 4-5 students. Each student must work on the assignment. Each student will be asked questions in the presentation to see if they understand the code, and this will be used to grade each students’ understanding and contribution to the project.**

**There is a video on Moodle that shows how the program should work.**

Assignment 4 - Project

In this project we will make a **Bank Account System**. We will use *Tkinter* to make a GUI (Graphical User Interface) so that users can see the details of bank accounts.



**Main Text Box showing account details**

**Task Buttons to show different details in the text box**

**Account Text Box to enter account number**

**Number Buttons to type the account number**

**List Box to show all account numbers**

**Button   
to delete account**

Main Window

* The GUI has one **window** that shows details about the accounts and it has buttons to do tasks on the accounts.
* The main window has the title “Bank Account System”.
* You will need **Frames** to make the layout of the buttons and boxes.

Main Text Box showing account details

* The main **Text Box** can show different text when the user clicks buttons.
* It has a **scroll bar** on the right side.
* It will show the Menu text when the program starts.

**Task Buttons to show different details in the text box**

There are 9 **Task Buttons**:

1. **Menu**

This button will show the Menu in the **Main Text Box**.

1. **Put money in**
   * This button lets the user put money into 1 account.
   * It will first make the **Main Text Box** empty.
   * It will make the **Account Text Box** empty.
   * It will write “*Put money in …*” in the **Main Text Box**.
   * It will write “*Enter the account ID:* ” in the **Main Text Box**.
   * It will wait for the user to use the **Number Buttons** to enter the account number.
   * After the user has entered the account number and clicked the *ENTER* button, it will make the **Account Text Box** empty.
   * It will put the account number that the user typed into the **Main Text Box**.
   * It will then run a method in the **Bank** class to check if the account number exists.
   * If the account number does not exist, it will write “*Account number XXXXX cannot be found.*” and it will stop.
   * If the account number exists, it will write “*Enter amount:* ” in the **Main Text Box**.
   * It will wait for the user to use the **Number Buttons** to enter the amount of money.
   * After the user has entered the amount of money, it will make the **Account Text Box** empty.
   * It will write the amount of money that the user typed into the **Main Text Box**.
   * It will then run a method in the **Bank** class to **add** the money into the account.
   * If everything is OK, it will write “*Success*” in the **Main Text Box**.
   * If there is a problem, it will write “*Failed*” in the **Main Text Box**.
2. **Get money out**
   * This button lets the user take money out of 1 account.
   * It will first make the **Main Text Box** empty.
   * It will make the **Account Text Box** empty.
   * It will write “*Get money out …*” in the **Main Text Box**.
   * It will write “*Enter the account ID:* ” in the **Main Text Box**.
   * It will wait for the user to use the **Number Buttons** to enter the account number.
   * After the user has entered the account number and clicked the *ENTER* button, it will make the **Account Text Box** empty.
   * It will put the account number that the user typed into the **Main Text Box**.
   * It will then run a method in the **Bank** class to check if the account number exists.
   * If the account number does not exist, it will write “*Account number XXXXX cannot be found.*” and it will stop.
   * If the account number exists, it will write “*Enter amount:* ” in the **Main Text Box**.
   * It will wait for the user to use the **Number Buttons** to enter the amount of money.
   * After the user has entered the amount of money, it will make the **Account Text Box** empty.
   * It will write the amount of money that the user typed into the **Main Text Box**.
   * It will then run a method in the **Bank** class to **subtract** the money from the account.
   * If everything is OK, it will write “*Success*” in the **Main Text Box**.
   * If there is a problem, it will write “*Failed*” in the **Main Text Box**.
3. **Show account with most money**
   * This button will show the details of the account with the most money.
   * It will first make the **Main Text Box** empty.
   * It will make the **Account Text Box** empty.
   * It will write “*----------Bank account with the most money----------*” in the **Main Text Box**.
   * It will then run a method in the **Bank** class to find the details of the bank account with the most money.
   * It will write the details of the account with the most money in the **Main Text Box**. For example:

Name: Cruise Tom

Birthday: 1962/7/3

Account ID: 10004

Account Type: Premium

Money: 340000000.00

1. **Show account with most transactions (交易)**
   * This button will show the details of the account with the most number of transactions.
   * Each time money is put into an account or taken out of an account, the number of transactions for *that* account is increased by 1.
   * It will first make the **Main Text Box** empty.
   * It will make the **Account Text Box** empty.
   * It will write “*----------Bank account with the most transactions----------*” in the **Main Text Box**.
   * It will then run a method in the **Bank** class to find the details of the bank account with the most number of transactions.
   * It will write the details of the account with the most number of transactions in the **Main Text Box**. For example:

Name: Key John

Birthday: 1961/8/8

Account ID: 10002

Account Type: Gold

Money: 56666600.00

1. **Show one account**
   * This button will show the details of the account which the user chooses.
   * It will first make the **Main Text Box** empty.
   * It will make the **Account Text Box** empty.
   * It will write “*Show one account …*” in the **Main Text Box**.
   * It will write “*Enter the account ID:* ” in the **Main Text Box**.
   * It will wait for the user to use the **Number Buttons** to enter the account number.
   * After the user has entered the account number and clicked the *ENTER* button, it will make the **Account Text Box** empty.
   * It will put the account number that the user typed into the **Main Text Box**.
   * It will then run a method in the **Bank** class to check if the account number exists.
   * If the account number does not exist, it will write “*Account number XXXXX cannot be found.*” and it will stop.
   * If the account number exists, it will write the details of the account that the user chose in the **Main Text Box**. For example:

Name: Key John

Birthday: 1961/8/8

Account ID: 10002

Account Type: Gold

Money: 56666600.00

1. **Show all accounts**
   * This button will show the details of all of the accounts in the program.
   * It will first make the **Main Text Box** empty.
   * It will make the **Account Text Box** empty.
   * It will write “*Showing all accounts …*” in the **Main Text Box**.
   * It will then run a method in the **Bank** class to get the details for all of the accounts.
   * It will write the details of all of the accounts in the **Main Text Box**. For example:

Name: Key John

Birthday: 1961/8/8

Account ID: 10002

Account Type: Gold

Money: 56666600.00

---------

Name: Roberts Julia

Birthday: 1967/10/28

Account ID: 10003

Account Type: Premium

Money: 454545.00

---------

Name: Cruise Tom

...

1. **Show total number of transactions (交易)**
   * This button will show the total number of transactions for all accounts in the program.
   * It will first make the **Main Text Box** empty.
   * It will make the **Account Text Box** empty.
   * It will write “*Total transactions:* ” in the **Main Text Box**.
   * It will then run a **recursive** function in the **Bank** class to get the total number of transactions by looking at each account and adding the number of transactions for each account.
   * It will write the total number of transactions for all accounts in the **Main Text Box**. For example:

Total transactions:

3

1. **Exit**
   * This button will first save all of the account details into the same text file that was opened at the beginning of the program.
   * Then the main window will close, and the program will finish.

**Number Buttons to type the account number**

There are 12 buttons:

* Buttons **0** to **9** will put the numbers 0 to 9 into the **Account Text Box**.
* Button **C** will delete the last number from the **Account Text Box**.
* Button **ENTER** will be used when the user has finished writing the account number or finished writing the amount of money (described above).

**List Box to show all account numbers**

* The **List Box** will show all of the account numbers in the program.
* It has a **scroll bar** on the right side.
* All of the account numbers will show inside the **List Box** when the program starts.
* If an account is deleted, then the **List Box** will be updated so that the deleted account is not shown.

**Button to delete account**

* The **Delete Button** will delete the account that is selected inside the **List Box**.
* The account will be deleted from the program.
* After the **Delete Button** is clicked, the **List Box** will be updated so that the deleted account is not shown.
* It will make the **Main Text Box** empty.
* It will make the **Account Text Box** empty.
* It will write “*Account XXXXX deleted* ” in the **Main Text Box**.

Files and classes

You will need 3 files to make this program:

* **Text file** for all of the bank account data.
* A Python file called ***Bank.py*** which has all of the code and classes that change the data in the bank accounts.
* A Python file called ***GUI.py*** which has all of the code to make the windows, boxes and buttons using ***Tkinter***.

Text file

You will need this data to put into the program. This data has all of the details for the bank accounts in the program. Save it in a text file in the same folder as your Python files.

Adams,Brain,1960,11,5,10000,1003000.5,1

DrinkingWater,John,1951,11,30,10001,4000.34,0

Key,John,1961,8,8,10002,56666600.0,2

Roberts,Julia,1967,10,28,10003,454545.0,1

Cruise,Tom,1962,7,3,10004,340000000.0,1

Tom,Johnny,1954,9,25,10005,230.0,0

Haldeman,Joe,1916,6,9,10006,23055.5,0

King,Charles,1955,11,19,10007,2400.5,0

Thomson,James,1980,4,4,10008,330.5,0

Diaz,Cameron,1972,8,30,10009,23000000.6,1

Obama,Barak,1961,8,4,10010,2000000.4,2

Hoffman,Dustin,1937,8,8,10011,23000000.0,1

Williams,Robin,1951,7,21,10012,22000000.5,1

Crowe,Russell,1964,4,7,10013,44000000.3,1

Minogue,Kylie,1968,5,28,10014,10000000.3,1

Keating,Paul,1944,1,18,10015,4000000.8,2

Gaga,Lady,1986,3,28,10016,50000000.0,1

Clark,Helen,1950,2,26,10017,400000.0,2

Suzenski,Mikhail,1951,3,2,10018,0.0,0

Yeltsin,Boris,1931,2,1,10019,10000000.5,2

Bank.py file

* You will need to make a Python file called ***Bank.py***
* This file will have all of the classes that do tasks for the bank accounts.
* Inside the *Bank.py* file, you will need 3 classes:

1. **Class Person**
2. **Class Bank\_Account**
3. **Class Bank\_Account\_Tasks**

* **Class Person:**
  + There are 5 object variables:

1. first\_name
2. last\_name
3. birth\_year
4. birth\_month
5. birth\_day
   * The \_\_init\_\_ method will set these object variables.
   * There is also a method called show\_details() that **returns** a **string** with the full name and birthday of the **Person** object.

* **Class Bank\_Account:**
  + This class **inherits** class **Person**.
  + There is 1 class variable called transactions
  + There are 3 object variables:

1. account\_id
2. money
3. account\_type
   * The \_\_init\_\_ method will set these object variables.
   * There is also an **override** method with the name show\_details() that **calls** the show\_details() **super class** and also **returns** a string with the account ID, the money and the account type.
     + If the **account type** is 0, this means the **account type** is “Regular”
     + If the **account type** is 1, this means the **account type** is “Premium”
     + If the **account type** is 2, this means the **account type** is “Gold”
   * There is a method called money\_in() that has a parameter money
     + It will add money to the bank account’s money variable.
     + It will return **True**
   * This is a method called money\_out() that has a parameter money
     + If there is enough money in the bank account, then money will be taken away from the bank account’s money variable, and the method will return **True**.
     + If there is NOT enough money in the bank account, then the method will return **False**.

* **Class Bank\_Account\_Tasks:**
  + This class hasa **class variable** which is a **list** called accounts\_list
  + The accounts\_list will have all of the bank account objects inside it.
  + There is also a **class variable** called file\_name that has the name of the text file with the bank account data.
  + Method read\_data()
    - This method will open the file and read the data.
    - A FOR loop will read each line
    - Inside the loop, the line is split into variables
    - The variables are used to make a Bank\_Account object
    - The Bank\_Account object is put into the accounts\_list
  + Method write\_data()
    - This method will save the data from the accounts\_list back into the text file.
    - A FOR loop will take each object, make a string with each of the variables separated by commas, and put it into a list.
    - After the list is made, a FOR loop will write each string from the list into the text file.
  + Method check\_account()
    - This method has one parameter string\_account\_id
    - It will search through the accounts\_list looking for the account ID that is the same as string\_account\_id
    - If the bank account with the same ID is not found, the method will return 0
  + Method put\_money\_in()
    - This method has 2 parameters string\_account\_id and amount
    - It will search through the accounts\_list looking for the account ID that is the same as string\_account\_id
    - When the bank account with the same ID is found, the money\_in() method is **called** in the bank\_account class. Give the money\_in() method the amount variable as a parameter.
    - If the money\_in() method in the bank\_account class returns **True**, increase the transactions variable by 1 and make the put\_money\_in() method return “*Success”*
    - If the money\_in() method in the bank\_account class returns **False**, make the put\_money\_in() method return “*Failed”*
  + Method get\_money\_out()
    - This method has 2 parameters string\_account\_id and amount
    - It will search through the accounts\_list looking for the account ID that is the same as string\_account\_id
    - When the bank account with the same ID is found, the money\_out() method is **called** in the bank\_account class. Give the money\_out() method the amount variable as a parameter.
    - If the money\_out() method in the bank\_account class returns **True**, increase transactions variable by 1 and make the get\_money\_out() method return “*Success”*
    - If the money\_out() method in the bank\_account class returns **False**, make the get\_money\_out() method return “*Failed”*
  + Method show\_biggest\_account()
    - This method will return a string with the account details of the account that has the biggest amount of money in it.
    - It will search through the accounts\_list looking for the account that has the biggest amount of money.
    - When the bank account with the biggest amount of money is found, the show\_details() method is **called** in the bank\_account class. This will return a string of the account details.
    - The show\_biggest\_account() method will return this string.
  + Method most\_transactions()
    - This method will return a string with the account details of the account that has the biggest amount of transactions (交易).
    - It will search through the accounts\_list looking for the account that has the biggest amount of transactions.
    - When the bank account with the biggest amount of transactions is found, the show\_details() method is **called** in the bank\_account class. This will return a string of the account details.
    - The most\_transactions() method will return this string.
    - If there are 0 transactions for all accounts, the most\_transactions() method will return the string of details for the ***first*** bank account.
    - If there are 2 bank accounts with the same number of transactions, the most\_transactions() method will return the string of details for the ***first*** bank account with the most transactions.
  + Method get\_number\_of\_transactions()
    - This method will return the total number of transactions for all of the accounts in the program.
    - It will call the add\_transactions() method with the accounts\_list as a parameter.
    - The get\_number\_of\_transactions() method will return the number returned from the add\_transactions() method.
  + Method add\_transactions()
    - This is a **recursive function** with 1 parameter called a\_list
    - It will use **recursion** to add the number of transactions for all of the accounts in the program.
    - You **must** use **recursion** here.
  + Method show\_account()
    - This method has 1 parameter string\_account\_id
    - It will search through the accounts\_list looking for the account ID that is the same as string\_account\_id
    - When the bank account with the same ID is found, the show\_details() method is **called** in the bank\_account class. This will return a string of the account details.
    - The show\_account() method will return this string.
  + Method delete\_account()
    - This method has 1 parameter string\_account\_id
    - It will search through the accounts\_list looking for the account ID that is the same as string\_account\_id
    - When the bank account with the same ID is found, this bank account will be **deleted** from accounts\_list.

GUI.py file

* You will need to make a Python file called ***GUI.py***
* This file will need to import ***Tkinter*** and ***Bank\_Account\_Tasks*** from ***Bank***
* At the top of the file, you will need to set up *Tkinter*
* You will also need to make an object of the Bank\_Account\_Tasks() called bank
* You need to call the read\_data() method of the bank object.
* You need to make many methods that connect to the buttons on the window.
* For example, if the user clicks the *Menu* button, it will call a method in *GUI.py*:
  + Methodshow\_menu()
    - This will print the menu in the **Main Text Box**.
    - It will also put the account numbers into the **List Box**
* You need to write the code to make the **Main Text Box**, **Task Buttons**, **Account Text Box**, **Number Buttons**, **List Box**, and the **Button to delete account**
* You need to write the methods that will run when the user clicks these buttons.
* You are encouraged to add any variables or methods into any of the classes to make the program more efficient. For example, making the **Account Text Box** empty is needed many times. If this code is put into a method and called when it is needed, this will decrease the amount of code.
* This assignment is an example with simple operations. It does not need to be like a real system.

Marking Schedule

Python Program: **10 marks**

Presentation: **10 marks**

**Python Program:**

|  |  |
| --- | --- |
| Bank.py |  |
| Person class |  |
| Bank\_Account class |  |
| Bank\_Account\_Tasks class: |  |
| Initialisation |  |
| read\_data() |  |
| write\_data() |  |
| check\_account() |  |
| put\_money\_in() |  |
| get\_money\_out() |  |
| show\_biggest\_account() |  |
| most\_transactions() |  |
| get\_number\_of\_transactions() |  |
| add\_transactions() |  |
| show\_account() |  |
| delete\_account() |  |
|  |  |
| GUI.py |  |
| Setup |  |
| Show menu |  |
| Clear screen |  |
| Clear entry |  |
| Money In |  |
| Money Out |  |
| Most Money |  |
| Most Transactions |  |
| Show One Account |  |
| Show All Accounts |  |
| Sum Transactions |  |
| Delete Selected Account |  |
| Insert Text from Number Buttons |  |
| Delete One Number |  |
| Close Window |  |
| Save File |  |
| Populate List Box |  |
| Make Frames |  |
| Text Boxes |  |
| Buttons |  |
| List Box |  |
|  |  |
| Program Header - group details and pledge of honour |  |
| Coding Standards - correct use, formatting, comments, logical |  |
| TOTAL | **10** |

**Presentation:**

* The presentation will be done by all students in the group.
* Each student in the group must speak about the work they did in the project.
* Each student must show their understanding of the project.
* Pictures, videos, PPTs and demonstrations can be used to help explain the program.
* The total time for the presentation should be 10-20 minutes.

|  |  |
| --- | --- |
|  |  |
| Voice and audience interaction | 2.5 |
| Knowledge of Material | 2.5 |
| Answers to questions | 2.5 |
| Use of media to demonstrate understanding | 2.5 |
| TOTAL | **10** |

### Help

Programming Best Practices

Code readability is one of the first things we learn as programmers. A program is only written once but will be looked at many times by you or other people later. It is important to make your code readable and understandable. Here are some best practices when writing readable code.

1. **Indentation** is the whitespace characters you put at the beginning of a line of code. Indentation is important in Python. Here is an example:

Good indentation:

x = 5  
  
**if** x == 5:  
 print(**"x is equal to 5"**)

NOT good indentation **(error)**:

x = 5  
  
**if** x == 5:  
print(**"x is equal to 5"**)

**Code spacing**. This is another making code able to be read easily. Here are some examples of code spacing.

name = **"John Smith"** *# Good*name=**"John Smith"** *# NOT good*print(**"Hello world"**) *# Good*print (**"Hello world"**) *# NOT good*a = b + c *# Good*a=b+c; *# NOT good*a = b - c *# Good*a = b-c *# NOT good*a = b \* 2 + c / 3 *# Good*a = b\*2 + c/3 *# NOT good*

**Comments**. Try not to write comments that do not need to be written. Remember, you do not have to write a comment for every line of code. Here are 2 examples, where the comments in the first is good, and the number of comments in the second example is too large.

**def** GoodComments():  
  
 *# Calculate and print area of a square* l = int(input(**"Enter length of side: "**))  
  
 **if** l <= 0:  
 print(**"Please enter a side greater than 0"**)  
 **else**:  
 print(**"Area is {0}"**.format(l \* l))  
  
  
  
  
**def** TooManyComments():  
  
 *# Get length of side of the square from user* l = int(input(**"Enter length of side: "**))  
  
 *# If the length is less than or equal to 0* **if** l <= 0:  
 *# Show error message* print(**"Please enter a side greater than 0"**)  
 **else**: *# If the length is greater than 0  
 # Calculate and print the area of the square* print(**"Area is {0}"**.format(l \* l))

**Naming variables and methods**. The names of variables and methods are important programming. The names help you and others understand what the code does. For example, the variable name max\_score is a much better name than xyz. The second aspect of naming scheme is that you should have word boundaries in the name.

You should also keep the same style of naming. There are two options, **camelCase** and **underscores**. The **underscores** option is recommended for the Python language. For example:

my\_dog, max\_temperature