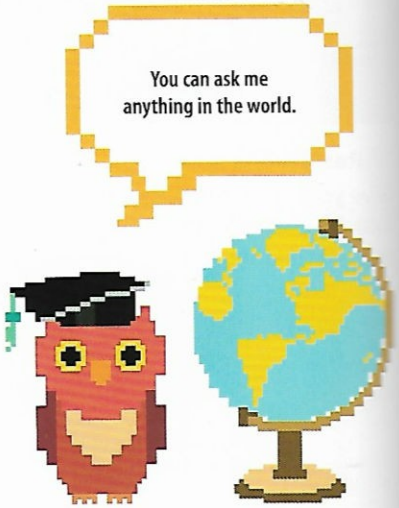
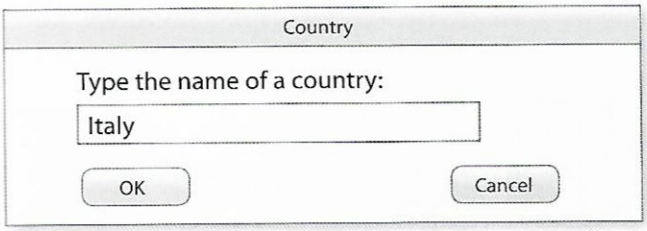
Ask the Expert

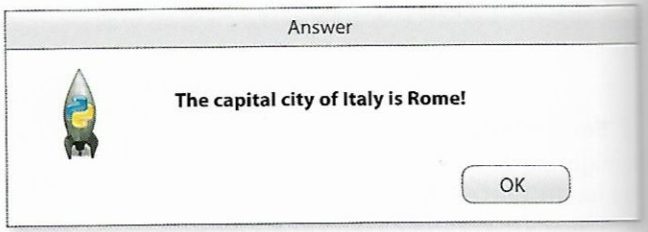
Can you name all the capital cities in the world? Or the players in your favourite sports team? Everyone's an expert on something. In this project, you'll code a program that can not only answer your questions, but also learn new things and become an expert.

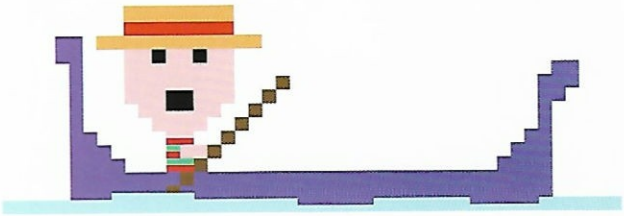


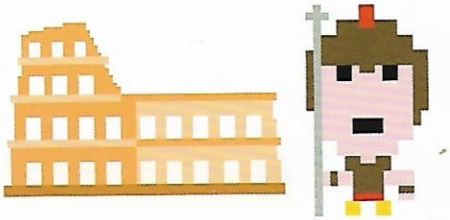
What happens

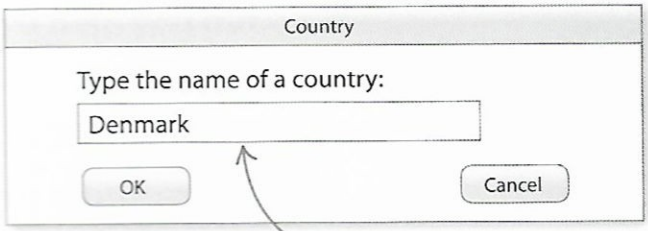
An input box asks you to enter the name of a country. When you type in your answer, the program tells you what the capital city is. If the program doesn't know, it asks you to teach it the correct answer. The more people use the program, the smarter it gets!



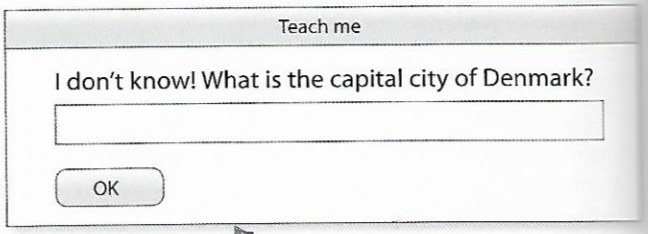








Enter name of a country



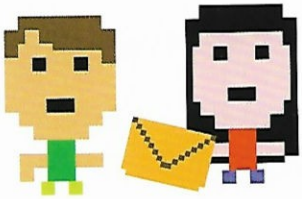
The program will ask you if it doesn't know the answer.

How it works

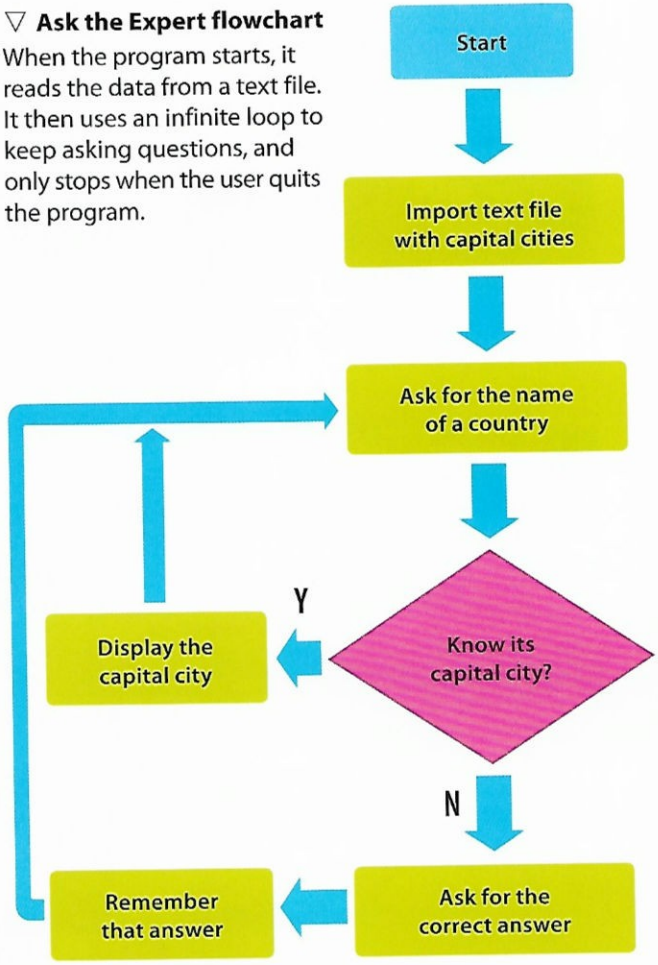
The program gets the information about capital cities from a text file. You'll use the `Tkinter` module to create the popup boxes that let the program and user communicate. When a new capital city is entered by a user, the information is added into the text file.

△ **Dictionaries**
You'll store the names of countries and their capitals in a dictionary. Dictionaries work a bit like lists, but each item in a dictionary has two parts, called a key and a value. It's usually quicker to look things up in a dictionary than it is to find something in a long list.

▷ **Communication**
The program uses two new `Tkinter` widgets. The first, `simpledialog()`, creates a popup box that asks the user to input the name of a country. The second, `messagebox()`, displays the capital city.



▽ **Ask the Expert flowchart**
When the program starts, it reads the data from a text file. It then uses an infinite loop to keep asking questions, and only stops when the user quits the program.



LINGO

Expert systems

An expert system is a computer program that is a specialist on a particular topic. Just like a human expert, it knows the answers to many questions, and it can also make decisions and give advice. It can do this because a programmer has coded it with all the data it needs and rules about how to use the data.

△ **Auto wizards**
Motor companies create expert systems that are full of information about how their cars function. If your car breaks down, a mechanic can use these systems to solve the problem. It's like having a million expert mechanics look at the problem rather than just one!

First steps

Follow these steps to build your own expert system using Python. You'll need to write a text file of country capitals, open a **`Tkinter`** window, and create a dictionary to store all the knowledge.

1 Prepare the text file

First make a text file to hold a list of capital cities of the world. Create a new file in IDLE and type in the following facts.

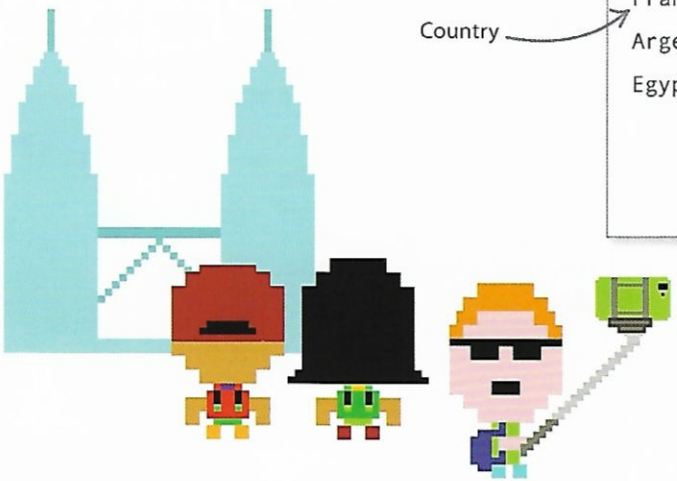
Untitled.txt

```
India/New Delhi
China/Beijing
France/Paris
Argentina/Buenos Aires
Egypt/Cairo
```

Country

Capital city

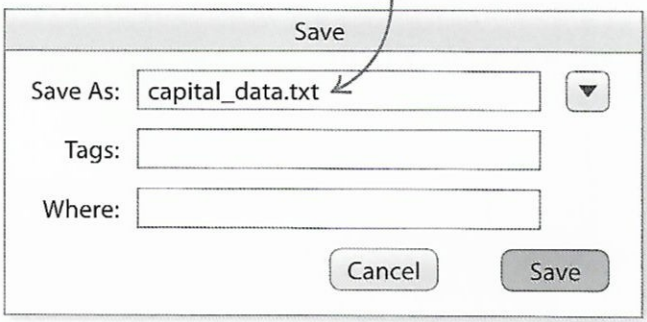
The forward slash (/) character is used to split the country and the city.



2 Save the text file

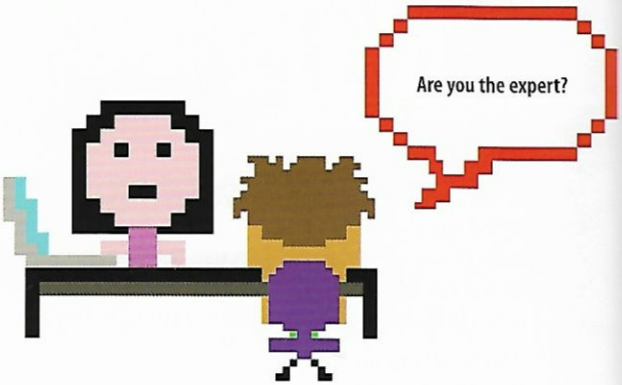
Save the file as "capital_data.txt". The program will get its specialist knowledge from this file.

Type "txt" at the end of the filename, instead of "py".



3 Create the Python file

To write the program, create a new file and save it as "ask_expert.py". Make sure you save it in the same folder as your text file.



4 Import Tkinter tools

To make this program you'll need some widgets from the **Tkinter** module. Type this line at the top of your program.

Load these two widgets from the **Tkinter** module.

```
from tkinter import Tk, simpledialog, messagebox
```

5 Start Tkinter

Next add the following code to display the title of the project in the shell. **Tkinter** automatically creates an empty window. You don't need it for this project, so hide it with a clever line of code.

```
print('Ask the Expert - Capital Cities of the World')
root = Tk()
root.withdraw()
```

Hide the **Tkinter** window.

Create an empty **Tkinter** window.

6 Test the code

Try running your code. You should see the name of the project displayed in the shell.

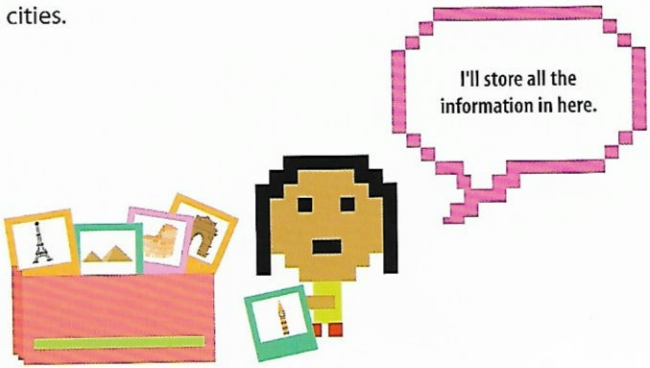
7 Set up a dictionary

Now type this line of code after the code you wrote for Step 5. The new code sets up the dictionary that will store the names of the countries and their capital cities.

This creates an empty dictionary called **the_world**.

```
the_world = {}
```

Use curly brackets.



EXPERT TIPS

Using a dictionary

A dictionary is another way you can store information in Python. It is similar to a list, but each item has two parts: a key and a value. You can test it out by typing this into the shell window.

```
favorite_foods = {'Simon': 'pizza', 'Jill': 'pancakes', 'Roger': 'custard'}
```

This is the key.

This is the value.

A colon is used immediately after the key.

Each item in the dictionary is separated by a comma.

Dictionaries use curly brackets.

▽ **1.** To show the contents of a dictionary, you have to print it. Try printing `favorite_foods`.

```
print(favorite_foods)
```

Type this in the shell and hit enter/return.

▽ **2.** Now add a new item to the dictionary: Julie and her favorite food. She likes cookies.

```
favorite_foods['Julie'] = 'cookies'
```

Key

Value

▽ **3.** Jill has changed her mind—her favorite food is now tacos. You can update this information in the dictionary.

```
favorite_foods['Jill'] = 'tacos'
```

Updated value

▽ **4.** Finally, you can look up Roger's favorite food in the dictionary by simply using his name as the key.

```
print(favorite_foods['Roger'])
```

Use the key to look up the value.

It's function time!

The next stage of the project involves creating the functions that you'll need to use in your program.

8 **File input**

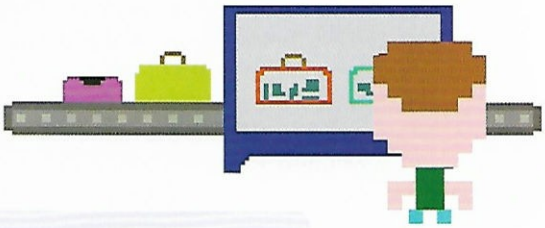
You need a function to read in all the information stored in your text file. It will be similar to the one you used in Countdown Calendar to read in data from your events file. Add this code after the `Tkinter` import line.

```
from tkinter import Tk, simpledialog, messagebox

def read_from_file():
    with open('capital_data.txt') as file:
```

This line opens the text file.

9 Line by line

Now use a **for** loop to go through the file line by line. Just as in Countdown Calendar, you must remove the invisible newline character. Then you need to store the values of country and city in two variables. Using the split command, the code will return the two values. You can store these values in two variables using one line of code.



```
def read_from_file():
    with open('capital_data.txt') as file:
        for line in file:
            line = line.rstrip('\n')
            country, city = line.split('/')
```

This removes the newline character.

This stores the word before "/" in the variable **country**.

This stores the word after "/" in the variable **city**.

The "/" character splits the line.

10 Add data to the dictionary

At this stage, the variables **country** and **city** hold the information you need to add into the dictionary. For the first line in your text file, **country** would hold "India" and **city** would hold "New Delhi". This next line of code adds them into the dictionary.

```
def read_from_file():
    with open('capital_data.txt') as file:
        for line in file:
            line = line.rstrip('\n')
            country, city = line.split('/')
            the_world[country] = city
```

This is the value.

This is the key.

11 File output

When the user types in a capital city the program doesn't know about, you want the program to insert this new information into the text file. This is called file output. It works in a similar way to file input, but instead of reading the file, you write into it. Type this new function after the code you typed in Step 10.

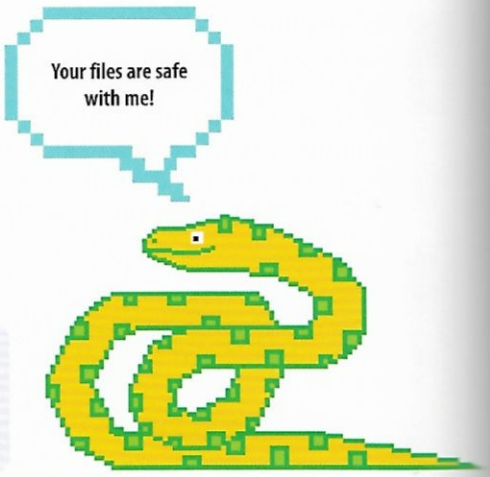
```
def write_to_file(country_name, city_name):
    with open('capital_data.txt', 'a') as file:
```

This function will add new country and capital city names to the text file.

The **a** means "append", or add, new information to the end of the file.

12 **Write to the file**

Now add a line of code to write the new information into the file. First the code will add a newline character, which tells the program to start a new line in the text file. Then it writes the name of the country followed by a forward slash (/) and the name of the capital city, such as Egypt/Cairo. Python automatically closes the text file once the information has been written into it.




```
def write_to_file(country_name, city_name):
    with open('capital_data.txt', 'a') as file:
        file.write('\n' + country_name + '/' + city_name)
```

Code the main program

You've written all the functions you need, so it's time to start coding the main program.

13 **Read the text file**

The first thing you want the program to do is to read the information from the text file. Add this line after the code you wrote in Step 7.

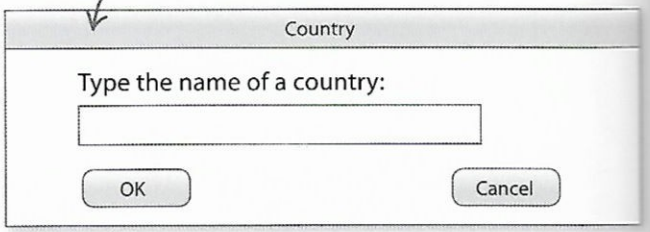
Run the `read_from_file` function.

```
read_from_file()
```

14 **Start the infinite loop**

Next add the code below to create an infinite loop. Inside the loop is a function from the `Tkinter` module: `simpledialog.askstring()`. This function creates a box on the screen that displays information and gives a space for the user to type an answer. Test the code again. A box will appear asking you for the name of a country. It may be hidden behind the other windows.

This is the box created by `simpledialog`.



This appears in the box to tell the user what to do.

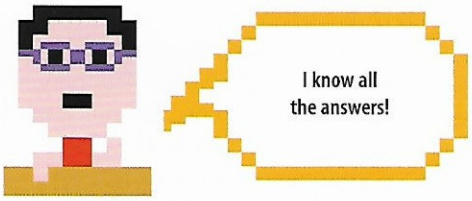
```
read_from_file()

while True:
    query_country = simpledialog.askstring('Country', 'Type the name of a country:')
```

The answer the user types is stored in this variable.

This is the title of the box.

15 Know the answer?

Now add an **if** statement to see if the program knows the answer. This will check whether the country and its capital city are stored in the dictionary.

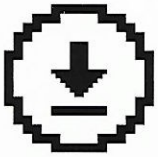



```
while True:
    query_country = simpledialog.askstring('Country', 'Type the name of a country:')

    if query_country in the_world:
```

Will return True if the country input by the user is stored in **the_world**.

16 Display the correct answer

If the country is in **the_world**, you want the program to look up the correct answer and display it on the screen. To do this, use the **messagebox.showinfo()** function from the **Tkinter** module. This displays the message in a box with an OK button. Type this inside the **if** statement.



Don't forget to save your work.

Using **query_country** as the key, this line looks up the answer from the dictionary.

```
if query_country in the_world:
    result = the_world[query_country]
    messagebox.showinfo('Answer',
                        'The capital city of ' + query_country + ' is ' + result + '!')
```

This is the title of the box.

This variable stores the answer (the value from the dictionary).

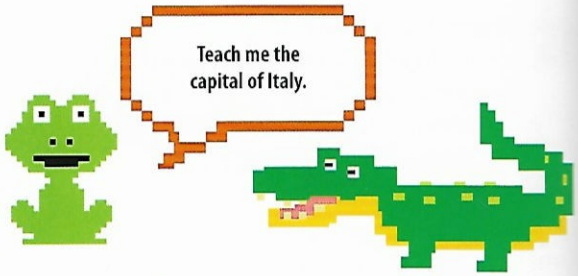
This message will be displayed inside the box.

17 Test it out

If your code has a bug, now would be a good time to catch it. When it asks you to name a country, type "France". Does it give you the correct answer? If it doesn't, look back over your code carefully and see if you can find out where it's gone wrong. What would happen if you typed in a country that wasn't in the text file? Try it out to see how the program responds.

18 **Teach it**

Finally, add a few more lines after the `if` statement. If the country isn't in the dictionary, the program asks the user to enter the name of its capital city. This capital city is added to the dictionary, so that the program remembers it for next time. Then the `write_to_file()` function adds the city to the text file.




```
    if query_country in the_world:
        result = the_world[query_country]
        messagebox.showinfo('Answer',
                            'The capital city of ' + query_country + ' is ' + result + '!')
    else:
        new_city = simpledialog.askstring('Teach me',
                                          'I don\'t know!' +
                                          'What is the capital city of' + query_country + '?')
        the_world[query_country] = new_city
        write_to_file(query_country, new_city)

root.mainloop()
```

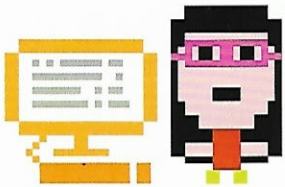
Ask the user to type in the capital city and store it in `new_city`.

This adds `new_city` to the dictionary, using `query_country` as the key.

Write the new capital city into the text file, so that it gets added to the program's knowledge.

19 **Run it**

That's it. You've created a digital expert! Now run the code and start quizzing it!



Hacks and tweaks

Take your program to the next level and make it even smarter by trying out these suggestions.

▷ Around the world

Turn your program into a geographical genius by creating a text file that contains every country in the world and its capital city. Remember to put each entry on a new line in this format: country name/capital city.

▽ Capitalize

If the user forgets to use a capital letter to name the country, the program won't find the capital city. How can you solve this problem using code? Here's one way to do it.

```
query_country = simpledialog.askstring('Country', 'Type the name of a country:')
query_country = query_country.capitalize()
```

This function turns the first letter in a string into a capital letter.

sports_teams.txt

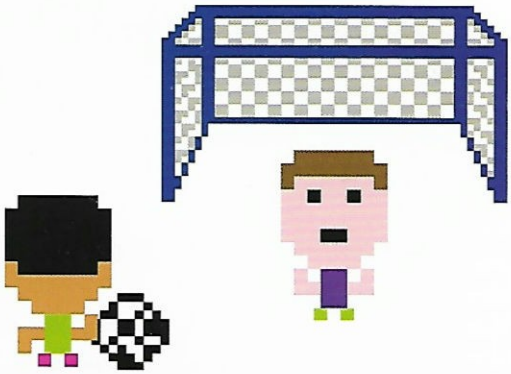
```
Castle United/Bobby Welsh
Dragon Rangers/Alex Andrews
Purple Giants/Sam Sloan
```

Coach's name

Team name

◁ Different data

At the moment, the program only knows about capital cities of the world. You can change that by editing the text file so that it stores facts about a subject on which you're an expert. For example, you could teach it the names of famous sports teams and their coaches.



▷ Fact check

Your program currently adds new answers straight into the text file, but it can't check if the answers are correct. Tweak the code so that new answers are saved in a separate text file. Then you can check them later before adding them to the main text file. Here's how you can change the code.

```
def write_to_file(country_name, city_name):
    with open('new_data.txt', 'a') as file:
        file.write('\n' + country_name + '/' + city_name)
```

This stores the new answers in a different text file, called **new_data**.