An introduction to electronics

Mar 04, 2020

CONTENTS

CHAPTER

ONE

CONTENTS:

1.1 Session 00

1.2 Session 01

1.2.1 Introduction

Welcome to *Learn to code for data analysis*! This course has been written to show how a little bit of code can go a long way in making sense of the increasingly vast amounts of open data available online.

If youre used to the manual approach of copying and pasting formulas in spreadsheets, youll see that writing a few lines of code to manipulate data can be quicker, less error-prone, and more powerful.

Please note: in the following video, where reference is made to a study week, this corresponds to Weeks 1 and 2 of this course.

RUTH ALEXANDER: **;

Hello. Im Ruth Alexander a journalist with the BBC. I work on the Radio 4 programmes Money Box and More or Less. Ill be introducing each week of this course which will show you how you can write code to work with data. Writing computer programmes is a way of learning what sorts of data computers can represent, how they can make decisions about data and what they can do as a result. Youll be using a notebook style of programming that helps you keep track of each data manipulation step as it happens. Each week of this course is organised around a small data analysis project. Youll get a notebook with exercises to work through and test your coding.

The week will end with a quiz to help you consolidate your learning. Programming requires practise so do have a go at all the exercises and quiz questions and get involved in the discussions. Well also suggest ways for you to extend each weeks project if you have the time. As youll see if we put the data into the right sort of format a single line of code can do a lot of work with it. The few and short lines of code youll be writing over the coming weeks employ exactly the same principles used in software developed for any purpose be it games or online shopping.

This week youll learn some of those general programming principles, how to apply them to data analysis and how to use and share notebooks. Enjoy.

You will write code in a programming language called Python, named after the comedy group Monty Python. Python is very popular for teaching programming, but is also widely used in professional software development.

The notebooks Ruth mentions in the video use an environment (like a program) called Jupyter, which allows you to write notebooks that include Python code. Jupyter is used by many scientists and data analysts as part of their workflow. Although you will use Python and Jupyter at a basic level, you will be learning tools used by the pros.

Python, Jupyter and other software you will need to take part in the course are included in the Anaconda distribution (like a package). Youll have an opportunity to download the software in the next step.

For now, think about your goals are for taking this course. Do you have any previous coding or data analysis experience, e.g.ăwith spreadsheets? Perhaps you want to find more efficient ways of working with data, or perhaps youre intrigued by the idea of what code is and how it can possibly help you work with data.

The Open University would really appreciate a few minutes of your time to tell us about yourself and your expectations for the course before you begin, in our optional start-of-course survey. Participation will be completely confidential and we will not pass on your details to others.

We hope you enjoy the course!

- Michel Wermelinger
- Rob Griffiths
- Tony Hirst

1.2.2 1 Install the software

To code in the course notebooks that Ruth mentioned in the video youll need to install some software.

Were going use a program called Jupyter that opens in your web browser and allows you to write notebooks that include Python code. Jupyter and other software you will need to take part in the course are freely available and you have two options.

Online CoCalc service

The advantages of using CoCalc are that you dont have to install any software and you can work on the course exercises from anywhere there is an internet connection. The disadvantages are that you will need a good internet connection, running the code in your notebook may take time if there are many simultaneous users on CoCalc and you may occasionally lose the latest changes you make in your notebook, because the service will periodically reset.

However, since notebooks are regularly auto-saved, the risk of losing work should be rather small. CoCalc offers a paid plan that has better performance and stability than the free plan. The Open University and the authors have no commercial affiliation with CoCalc.

Install Anaconda package

The other option is to install on your laptop or desktop the free Anaconda package, which includes all necessary software for this course. If you plan to work on this course from multiple computers, you will need to install Anaconda on each one. You can use cloud storage, like Dropbox, to keep your notebooks in sync across machines. Anaconda doesnt have the limitations of CoCalc, so we recommend you use Anaconda if you are going to work on this course always from the same computer.

You should now read the instructions for installing Anaconda or creating a CoCalc projectfor this course. Dont forget to test everything is working, as explained in the instructions.

The installation of Anaconda is different for Windows, Macs and Linux. Please follow the appropriate instructions.

We advise you to accept the pre-filled defaults suggested during the installation process.

Notebooks

Each week you will use two notebooks (and any necessary data files): an exercise notebook and a project notebook. The notebooks are this courses programming environment, where you will do your own coding.

The exercise notebook contains all the code shown throughout the week, so that you can try it out for yourself, any time you wish. The exercise notebook also contains all the weeks exercises. You will be able to solve several exercises just by slightly modifying our code.

The project notebook contains the weeks written-up data analysis project, including all necessary code. If you have the extra time, youre encouraged to modify the project notebooks to write up your own data analyses.

You should now download from here the notebooks and data files needed for the whole course.

Youll open the notebooks using Jupyter, which is part of Anaconda and CoCalc. You will learn how to use notebooks later this week, after youve seen what Python code looks like.

Note: please ensure that you abide by any terms and conditions associated with these pieces of software.

1.1 Start with a question

Data analysis often starts with a question or with some data.

Figure 1

A question leads to data that can answer it, and looking at the available data helps to make a question precise or may trigger new questions, which, in turn, may require further data. Data analysis is thus often an iterative process: the questions determine which data to obtain, and the data influences which questions to ask and what the scope of the analysis is. How this weeks project came about is an example of such an iterative process.

I (Michel) was watching a news programme mentioning the fight against tuberculosis (TB) as part of the United Nations Millenium Development Goals. Wishing to know how serious TB is, I browsed the World Health Organization (WHO) website and found a dataset with the number of TB cases and deaths per country per year, from 2007 to 2013. This in turn raised the question of whether a high (or low) number could be mainly due to the country having a large (or small) population. Some more browsing revealed the WHO also has population data from 1990 to 2013.

That was enough data for the fuzzy question: how serious is TB? It was time to make it precise. I chose to measure the effect of TB in terms of deaths, which led to the following questions:

- What is the total, smallest, largest, and average number of deaths due to TB?
- What is the death rate (number of deaths divided by population) of each country?
- Which countries have the smallest and largest number of deaths?
- Which countries have the smallest and largest death rate?

Answering these questions for the whole world and for seven years (2007–2013) would be a bit too much for this initial project. A subset was needed. I decided to take only the latest data for 2013 and, being Portuguese, to focus on the Portuguese-speaking countries. One of them, Brazil, is part of the BRICS group of major emerging economies, so for more diversity the other four countries would be included too: Russia, India, China and South Africa. The project was finally defined! Ive added links to the data below if youd like to take a look!

Activity 1 What would you ask?

Before you embark on coding the analysis to get answers, what other questions could be asked of the datasets described?

What countries would you be interested in? What groups of countries might be interesting to analyse?

Note down some of your questions so that you can come back to them later.

WHO POPULATION - DATA BY COUNTRY (LATEST YEAR)

WHO TB MORTALITY AND PREVALENCE - DATA BY COUNTRY (2007 - PRESENT)

Next, Ill explain how I started to organise the information.

1.2 Variables and assignments

With the choice of data and questions confirmed, the coding can begin.

Figure 2

To introduce the basics of coding, I will show you a very simple approach, only suitable for the smallest of datasets. Please bear with me. In the second part of the week I will show you the proper approach. Read through this step and the next – **youre not expected to write code just yet**. In Exercise 1, a bit further on in this week, youll be asked to start writing code.

Ok, lets start. I want the computer to calculate the total number of deaths in 2013. For the computer to do that, it must first be told what is the number of deaths in each country in that year. Ill start with my home country.

``In []:``

```
[ ]:
deathsInPortugal = 100
```

The In[] line is Jupyters way of saying that what follows is code I typed in. And there it is: the first line of code! It is a command to the computer that could be translated to English as: find in the attic an empty box, put the number 100 in the box, and write deathsInPortugal on the box. (Arent you glad Python is more succinct than English?) In coding jargon, the attic is the computers memory, boxes are called **variables** (Ill explain why shortly), whats written on a box is the variables **name** , and storing a value in a variable is called an **assignment**.

By naming the boxes, I can later ask the computer to show the value in box `thingamajig` or take the values in boxes `stuff` and `moreStuff` and add them together.

To see whats inside a box, I can just write the name of the box on a line of its own. Jupyter will write the variables value on the screen, preceded by Out[], to clearly mark the output generated by the code. When you start to use the Jupyter notebooks, you will see numbers inside the square brackets, i.e.ăIn[1], In[2], etc., to indicate in which order the various pieces of code are being executed. Here we have omitted the numbers to avoid confusion between what you see here and what you see in your notebook.

``In []:``

[ ]:
```
deathsInPortugal
```

``Out[]:``

[ ]:
```
100
```

Each assignment is written on a line of its own. The computer executes the assignments line by line, from top to bottom. Thus, the program would continue as follows:

``In []:``

[ ]:
```
deathsInPortugal = 100
deathsInAngola = 200
deathsInBrazil = 300
```

I dont think I need to continue, you get the gist.

By the way, all numbers so far are fictitious. If I use real data, taken from the World Health Organization website, youll see a difference.

``In []:``

[ ]:
```
deathsInPortugal = 140
deathsInAngola = 6900
deathsInBrazil = 4400
deathsInPortugal
```

``Out[]:``

[ ]:
```
140
```

Notice what happened. When a value is assigned to an already existing variable, the value stored in that variable is unceremoniously chucked away and replaced by the new value. In the example, the second group of assignments replaced the values assigned by the first group and thus the current value of `deathsInPortugal` is 140 and no longer 100. Thats why the storage boxes are called variables: their content can vary over time.

To sum up, a **variable** is a named storage for values and an **assignment** takes the value on the right hand side of the equal sign (=) and stores it in the variable on the left-hand side.

In the next section, you will find out the importance of naming in Python.

1.3 The art of naming

Python is relatively flexible about what you name your variables but rather picky about the format of names.

Figure 3

I could have chosen `deaths_in_Brazil_in_2013`, `deathsBrazil`,`DeathsBrazil`, `dB` or even `stuff` for my variables. If a box in your attic were labeled `dB` or `stuff` though, would you know what it contains a year later? So, although you can, its better not to use cryptic, general, or very long names.

You cant use spaces to separate words in a name, you cant start a name with a digit and names are case-sensitive, i.e.ă`deathsBrazil` and `DeathsBrazil` are not the same variable. Making one of those mistakes will result in a **syntax error** (when the computer doesnt understand the line of code) or a **name error** (when the computer doesnt know of any variable with that name).

Lets see some examples. (Remember that youre not expected to write any code for this step.) The first example has spaces between the words, the second example has a digit at the start of the name, and the third example changes the case of a letter, resulting in an unknown name. The kind of error is always at the end of the error message.

```
In []:
```

```
[ ]:
deaths In Portugal = 140
File "<ipython-input-7-ded1a063fe45>", line 1
deaths In Portugal = 140
^
SyntaxError: invalid syntax
```

```
In []:
```

```
[ ]:
2013deathsInPortugal = 140
File "<ipython-input-8-af085101fcfc>", line 1
2013deathsInPortugal = 140
```

(continues on next page)

(continued from previous page)

```
^
SyntaxError: invalid syntax
```

In []:

[ ]:

```
deathsinPortugal
---------------------------------------

NameError Traceback (most recent call last)

<ipython-input-9-7d3c81b4fb34> in <module ()
----> 1 deathsinPortugal
NameError: name deathsinPortugal is not defined
```

Note that Jupyter doesnt write any `Out[]` because the code is wrong and thus doesnt generate any output.

In this course, to make names shorter to help fit lines of code on small screens, well use capitalisation instead of underscores to separate the different words of a name, as shown in the code so far. Such practice is called **camel case** independently of the name having **``oneHump``** (dromedary case just doesnt sound good, does it?) or **``moreThanTwoHumps``**. The convention in Python is to start variable names with lower case and well stick to it.

In the next section, download the notebook for this week and work through the first exercise – your first line of code!

1.4 Downloading the notebook and trying the first exercise

So far, Ive done the coding and youve read along. Booooring. Its time to use the Jupyter notebooks and work on the first exercise in the course.

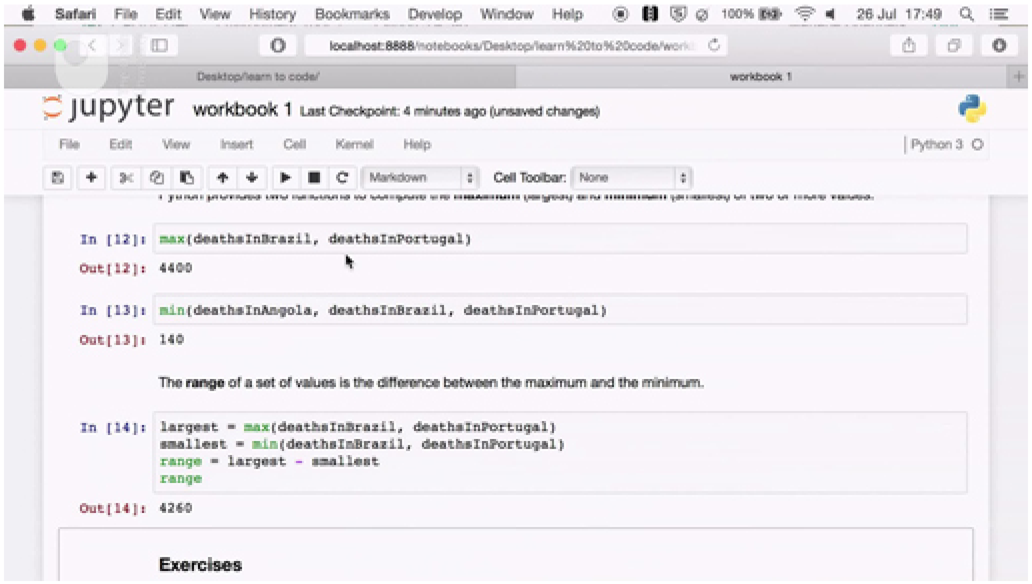
Exercise 1 Variables and assignments

If you havent yet installed the software package or created an account on CoCalc, do it now using these instructions!

Open the Exercise notebook 1, and put it in the folder you created. (Youll open the data later and learn how to use it in the notebook.)

Once you have installed the file, watch the video to learn how to work with Jupyter notebooks and complete Exercise 1. Pause the video frequently to repeat the demonstrated steps in your notebook. Throughout the week youll be directed back to the notebook to complete the other exercises.

NARRATOR: * In this screencast Im going to introduce you to Jupyter Notebooks. First youll need to start the Anaconda launcher. This screencast was done on a Mac. However, the same process applies to Windows. If youre a Linux user, youll need to use the command line as described in the installation instructions. To follow along on your computer, make sure you have created a folder for this course, and that it contains the exercise notebook for this week. This screencast uses earlier versions of the exercise notebook, and of the Anaconda software than you downloaded, so dont worry that things look slightly different. *;* Do not click on Update buttons in the Anaconda Launcher, because you should use the version you installed to avoid compatibility problems with the notebooks of this course. Once the Anaconda launcher has booted up, launch the ipython notebook. Whenever you see a circle, the mouse has been clicked. After a couple of screens you should see Jupyter running in a browser, and the contents of your home folder. Navigate to the folder you created, and open the relevant notebook. The first thing to appreciate is that Jupyter notebook consists of a sequence of individual cells. You can see the individual cells as I click on the left of each one. Each cell can contain text or code. *;* Before starting any exercises, you should execute all the code already in the notebook. Ill explain why in a moment. Go to the Cell menu and select Run all. As the notebook executes all code, it may automatically scroll to a different part of the notebook. Just scroll back to the start. Go to the first exercise. It asks you to add assignments for more countries into the preceding code cell. To select a cell, click to the left of it. A grey border shows the currently selected cell. To edit a cell, click inside it. The border becomes green to show the cell is in editing mode. *;* Once inside the cell press Enter a couple of times to put the cursor on a new line, then start typing an appropriate variable name, say deaths in Russia. Once youve start typing, names that have been used within the notebook can be accessed via the Tab key. So once youve typed de press the Tab key to get some auto-complete suggestions. Use the arrow keys to scroll through the options and press Enter to select the appropriate option. In this case Ill accept the first suggestion in the list and edit it to complete the assignment. Next start a new line and just enter the new variable name youve just added. Remember to use auto-complete to avoid spelling mistakes. *;* Now we can run the code. To execute only the current code cell, click on the Play button. The results appear below the code cell in a line titled Out. If you wish to split a cell, for example to separate the supplied code from the code you are adding, then put the cursor where you want to split the cell, go to the Edit menu and select Split Cell. Its easy to move cells around - for example we can cut a selected celland then paste it below another cell that you select. To save a snapshot of the notebook, called a checkpoint, click the Save and Checkpoint button. *;* If things go horribly wrong, you can revert the notebook to the last checkpoint by using the Revert to Checkpoint option in the File menu. To finish your session, go to the File menu and select Close and Halt. It is very important to note that opening a notebook does not execute any code cells. Any code output was saved from the previous session. So if I reopen the workbook and execute the code in the second cell alone, Ill get an error, because in this session (that is since the notebook was opened again) the first cell hasnt been executed and therefore the computer doesnt recognise the variable name deaths in Portugal. And thats why you should run all code after opening a notebook! *;* Its easy to add your own notes to the notebook. For example if I select the first celland then click on the plus button, I can insert a new cell below the current one. By default, a new cell is a code cell. Select Markdown to change it to a text cell. To edit a text cell, double-click inside the cell. Text is written in Markdown, a very simple formatting system. Here are some examples of what Markdown can do - pay attention to how prefixing or surrounding words with simple characters is all the information needed to format the text. *;* Also note the way a word or phrase to be hyperlinked is surrounded with square brackets and immediately followed by the URL, in round brackets. Once the text is written, click the Play button to see the formatted text in the cell. The Help menu contains links to information about Jupyter notebooks and Markdown formatting. As you get used to Jupyter, take a look at the keyboard shortcuts, as they will help you to work more efficiently. *;


Safari File Edit View History Bookmarks Develop Window Help
26 Jul 17:49
localhost:8888/notebooks/Desktop/learn%20to%20code/work
Desktop/learn to code/
workbook 1
jupyter
workbook 1 Last Checkpoint: 4 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Help
Python 3
Markdown
Cell Toolbar: None

```
In [12]: max(deathsInBrazil, deathsInPortugal)

Out[12]: 4400

In [13]: min(deathsInAngola, deathsInBrazil, deathsInPortugal)

Out[13]: 140
```

The **range** of a set of values is the difference between the maximum and the minimum.

```
In [14]: largest = max(deathsInBrazil, deathsInPortugal)
         smallest = min(deathsInBrazil, deathsInPortugal)
         range = largest - smallest
         range

Out[14]: 4260
```

Exercises

If you havent yet installed Jupyter and Anaconda, do it now using these instructions.

1.5 Expressions

Ive told the computer the deaths in Angola, Brazil and Portugal. I can now ask it to add them together to obtain the total deaths.

In []:

[ ]:

```
deathsInAngola + deathsInBrazil + deathsInPortugal
```

Out[]:

[ ]:

```
11440
```

A fragment of code that has a value is called an **expression**. Calculating the value of an expression is called **evaluating** the expression. If the expression is on a line of its own, the Jupyter notebook displays its value, as above.

The value of an expression can of course be assigned to a variable.

In []:

[ ]:

```
totalDeaths = deathsInAngola + deathsInBrazil + deathsInPortugal
```

Note that no value is displayed because the whole line of code is not an expression, its a **statement** , a command to the computer. In this case the statement is an assignment. You will see another kind of statement later this week.

To see the value, you learned that you must write the variables name.

In []:

[ ]:

```
totalDeaths
```

Out[]:

[ ]:

```
11440
```

This is really just a special case of the general rule that writing an expression on its own shows its value. A variable (which stores a value) is just an example of an expression (which is anything that has a value).

I can now write an expression to compute the average number of deaths: its the total divided by the number of countries.

In []:

[ ]:

```
totalDeaths / 3
```

Out[]:

[ ]:

```
3813.3333333333335
```

Python has of course all four arithmetic **operators** : addition (+), division (/), subtraction (-) and multiplication (*). Ill use the last two later in the week. Python follows the conventional operator precedence: multiplication and division before addition and subtraction, unless parentheses are used to change the order. For example, (3+4)/2 is 3.5 but 3+4/2 is 5.

Figure 4

Now practice writing expressions and complete Exercise 2 in the notebook.

Exercise 2 Expressions

Go back to the Exercise notebook 1 you used in Exercise 1. In Exercise 2 youll see an example of operator precedence and practise writing expressions.

If youre using Anaconda, remember that to open the notebook youll need to navigate to it using Jupyter. Whether youre using Anaconda or CoCalc, once the notebook is open, run all the code before doing the exercise.

Writing code for the first time can be difficult but stick with it.

In the next section, you will find out about functions.

1.6 Functions

After the total and the average, next on my to-do list is to calculate the largest number of deaths.

Figure 5

This will be the **maximum**. It takes another single line of code to calculate it.

In []:

[ ]:

```
max(deathsInAngola, deathsInBrazil, deathsInPortugal)
```

Out[]:

[ ]:

```
6900
```

In this expression, ``**max()**`` is a function – the parenthesis are a reminder that the name ``**max**`` doesnt refer to a variable. A **function** is a piece of code that calculates (**returns**) a value, given zero or more values (the functions **arguments**). In this case, ``**max()**`` has three arguments and returns the greatest of them. Actually, ``**max()**`` can calculate the maximum of two, three, four or more values.

In []:

[ ]:

```
max(deathsInBrazil, deathsInPortugal)
```

Out[]:

[ ]:

```
4400
```

The expressions above are function **calls**. Im calling the ``**max()**`` function with three or two arguments, and the value of the expression is the value returned by the function. A function is called by writing its name, followed by the arguments, within parentheses and separated by commas. Function names follow the same rules as variable names.

As you might expect, Python also has a function to calculate the smallest (minimum) of two or more values.

In []:

[ ]:
```
min(deathsInAngola, deathsInBrazil, deathsInPortugal)
```

Out[]:

[ ]:
```
140
```

The value returned by a function call can be assigned to a variable. Here is an example, which calculates the **range** of deaths. The range of a set of values is the difference between the largest and the smallest of them.

In []:

[ ]:
```
largest = max(deathsInAngola, deathsInBrazil, deathsInPortugal)


smallest = min(deathsInAngola, deathsInBrazil, deathsInPortugal)

range = largest - smallest
range
```

Out[]:

[ ]:
```
6760
```

Exercise 3 Functions

Identify different types of error (some of you may have experienced those already) in Exercise 3. Youll need to use the Week 1 notebook to answer question three.

__ 1. If the function name is misspelled as Min, what kind of error is it? __

A syntax error

Writing `Min(, )` instead of `min(, )` is not a syntax error, because both have the form of a function call. Names can use uppercase letters.

Take a look at The art of naming.

A name error

The computer will understand than `Min(, )` is a function call but doesnt know of any function with that name. Remember that names are case-sensitive.

Take a look at The art of naming.

__ 2. If a parenthesis or comma is forgotten, what kind of error is it? __

A name error

A parenthesis or comma is unrelated to how names are written.

Take a look at The art of naming.

A syntax error

A function call requires two parentheses around the arguments, and one comma between successive arguments. Forgetting any of them therefore deviates from the syntax of the Python language.

Take a look at Functions.

__ 3. Use Exercise 3 in the Week 1 exercise notebook to answer this question. __

__ What is the range of deaths among the BRICS countries (Brazil, Russia, India, China, South Africa)? __

4400

This is the minimum value (for Brazil), not the range.

Take a look at Functions.

65480

This is the average number of deaths, not the range.

Take a look at Expressions.

240000

This is the maximum value (for India), not the range.

Take a look at Functions.

327400

This is the total number of deaths not the range.

Take a look at Functions.

235600

The range is the maximum value (240 thousand for India) minus the minimum value (4400 for Brazil).

1.7 Comments

Last on my to-do list is the death rate, which is the number of deaths divided by the population.

Figure 6

A quick glance at the WHO website tells me Portugals population in 2013.

In []:

[ ]:
```
populationOfPortugal = 10608
```

Wait a minute! This cant be right. I know Portugal isnt a large country, but ten and a half thousand people is ridiculous. I look more carefully at the WHO website. Oh, the value is given in thousands of people; its 10 million and 608 thousand people. I could change the assignment to

In []:

[ ]:
```
populationOfPortugal = 10608000
```

but that could give the impression that the population had been counted exactly, whereas its more likely the number is an estimate based on a previous census. It also makes it easier to check my code against the WHO data if I use the exact same numbers.

I will therefore keep the original assignment but make a note of the unit, using a **comment** , a piece of text that documents what the code does.

In []:

[ ]:
```
# population unit: thousands of inhabitants
populationOfPortugal = 10608
# deaths unit: inhabitants
deathsInPortugal = 140
```

A comment starts with a hash sign (#) and goes until the end of the line. Computers ignore all comments, they just execute the code. Comments are your insurance policy: they help you understand your own code if you come back to it after a long break.

I can now compute the death rate, making sure I first convert the population into number of inhabitants, the same unit as deaths.

In []:

```
[ ]:
deathsInPortugal / (populationOfPortugal * 1000)
```

Out[]:

```
[ ]:
1.3197586726998491e-05
```

The death rate (roughly 140 people in 10 million) is a very small number, not very practical to display and reason about. Looking again at the WHO website, I note that other indicators, like TB prevalence, are given per 100 thousand inhabitants. I will do the same for the death rate. Since the population is already in thousands, dividing the deaths by the population gives me the number of deaths per thousand people. Thus, the number of deaths per 100 thousand people must be 100 times higher than that.

In []:

```
[ ]:
# death rate: deaths per 100 thousand inhabitants
deathsInPortugal * 100 / populationOfPortugal
```

Out[]:

```
[ ]:
1.3197586726998491
```

This finishes the basics of coding needed for this week. It took less than 30 lines of code

Test this out for yourself in Exercise 4 of the Week 1 exercise notebook.

Exercise 4 Comments

Complete the short exercise on the death rate in Exercise 4 in the Week 1 Exercise notebook.

Remember that once the notebook is open, run all the code, before doing the exercise.

1.8 Values have units

Before I move on, let me explain the importance of using comments to record units of measurement.

Figure 7

Values are not just numbers, they have units: degrees Celsius, number of inhabitants, thousands of gallons, etc. Always make a note of the units the value refers to, using comments. This makes it easier to check whether the expressions are right. Disregarding the units will lead to wrong calculations and results.

Activity 2

Have you come across horror stories that have happened due to mistakes in the unit of measurement?

Think through what happened and consider what you have learned from them.

1.2.3 2 This weeks quiz

Check what youve learned this week by taking the end-of-week quiz.

Week 1 practice quiz.

Open the quiz in a new window or tab then come back here when youve finished.

1.2.4 3 Summary

The first week of this course covered:

- installing software in course notebooks
- starting data analysis with a question
- the basics of coding
- naming formats
- recording units of measurement.

Next week, youll be introduced to pandas. Youll use Jupyter notebooks to write and execute simple programs with Python and the pandas module.

1.3 Session 02

1.3.1 1 Enter the pandas

As you probably realised, this way of coding is not practical for large scale data analysis.

Figure 1

Three lines of code were required for each country, to store the number of deaths, store the population, and calculate the death rate. With roughly 200 countries in the world, my trivial analysis would require 400 variables and typing almost 600 lines of code! Lifes too short to be spent that way.

Instead of using a separate variable for each datum, it is better to organise data as a table of rows and columns.

Table 1

Country	Deaths	Population
Angola	6900	21472
Brazil	4400	200362

Portugal

140

10608

In that way, instead of 400 variables, I only need one that stores the whole table. Instead of writing a mile long expression that adds 200 variables to obtain the total deaths, Ill write a short expression that calculates the total of the Deaths column, no matter how many countries (rows) there are.

To organise data into tables and do calculations on such tables, you and I will use the pandas module, which is included in Anaconda and CoCalc. A **module** is a package of various pieces of code that can be used individually. The pandas module provides very extensive and advanced data analysis capabilities to compliment Python. This course only scratches the surface of pandas.

I have to tell the computer that Im going to use a module.

In []:

[ ]:
```
from pandas import *
```

That line of code is an **import** statement: from the pandas module, import everything. In plain English: load into memory all pieces of code that are in the pandas module, so that I can use any of them. In the above statement, the asterisk isnt the multiplication operator but instead means everything.

Each weekly project in this course will start with this import statement, because all projects need the pandas module.

The words ``**from**`` and ``**import**`` are **reserved words** : they cant be used as variable, function or module names. Otherwise you will get a syntax error.

In []:

[ ]:
```
from = 100
File "<ipython-input-23-6958f0ebc10d>", line 1
from = 100
^
SyntaxError: invalid syntax
```

Jupyter notebooks show reserved words in boldface font to make them easier to spot. If you see a boldface name in an assignment (as you will for the code above), you must choose a different name.

Exercise 5 pandas

Question

Use Exercise 5 the Exercise notebook 1 to help you answer these questions about errors you might come across.

__ 1. What kind of error will you get if you misspell pandas as Pandas? __

A syntax error

Remember that after the reserved word from comes a module name.

Take a look at The art of naming .

A name error, reported as an import error

The computer is expecting a name but there is no module with the name Pandas in the Anaconda distribution. Remember that names are case-sensitive.

Question

__ 2. What kind of error will you get if you misspell import as impart? __

A name error

A name error only occurs when a name is undefined, but import is not a name, its a reserved word.

A syntax error

The computer is expecting a reserved word and anything else will raise a syntax error.

Question

__ 3. What kind of error will you get if you forget the asterisk? __

A name error

An asterisk is not a name so the reported error cant be this one.

A syntax error

The statement cannot end with the reserved word import; the computer is expecting an indication of what to import.

1.1 This weeks data

For the next part of the course youll need to download a file of data.

Figure 2

I have created a table with all the data necessary for the project and saved it in an Excel file. Excel is a popular application to create, edit and analyse tabular data. You wont need Excel to complete this course, but many datasets are provided as Excel files.

Open the data file WHO POP TB some.xls . The file is encoded using UTF-8, a character encoding that allows for accented letters. Do **not** open or edit the file, as you may change how it is encoded, which will lead to errors later on. If you do want to look at its contents, make a copy of the file and look at the copy.

Put the data file in the same folder (or CoCalc project) where you saved your exercise notebook. Done? Great, lets proceed to loading the data – youll learn how to do this in the next section.

1.2 Loading the data

Many applications can read files in Excel format, and pandas can too. Asking the computer to read the data looks like this:

In []:

[ ]:
```
data = read_excel('WHO POP TB some.xls')
data
```

Out[]:

Country

Population (1000s)

TB deaths

0

Angola

21472

6900

1

Brazil

200362

4400

2

China

1393337

41000

3

Equatorial Guinea

757

67

4

Guinea-Bissau

1704

1200

5

India

1252140

240000

6

Mozambique

25834

18000

7

Portugal

10608

140

8

Russian Federation

142834

17000

9

Sao Tome and Principe

193

18

10

South Africa

52776

25000

11

Timor-Leste

1133

990

The variable name data is not descriptive, but as there is only one dataset in our analysis, there is no possible confusion with other data, and short names help to keep the lines of code short.

The function `read_excel()` takes a file name as an argument and returns the table contained in the file. In pandas, tables are called **dataframes** . To load the `data`, I simply call the function and store the returned dataframe in a variable.

A file name must be given as a **string** , a piece of text surrounded by quotes. The quote marks tell Python that this isnt a variable, function or module name. Also, the quote marks state that this is a single name, even if it contains spaces, punctuation and other characters besides letters.

Misspelling the file name, or not having the file in the same folder as the notebook containing the code, results in a **file not found** error. In the example below there is an error in the file name.

In []:

[ ]:

```
data = read_excel('WHO POP TB same.xls')
data


---------------------------------------------


FileNotFoundError Traceback (most recent call last)

<ipython-input-25-c017b2500afa> in <module>()
----> 1 data = read_excel(WHO POP TB same.xls)
2 data



/Users/mw4687/anaconda/lib/python3.4/site-packages/pandas/io/excel.py in read_
↪excel(io, sheetname, **kwds)

130 engine = kwds.pop(engine, None)
131

--> 132 return ExcelFile(io, engine=engine).parse(sheetname=sheetname, **kwds)

133
134



/Users/mw4687/anaconda/lib/python3.4/site-packages/pandas/io/excel.py in __init__
↪(self, io, **kwds)


167 self.book = xlrd.open_workbook(file_contents=data)

168 else:

--> 169 self.book = xlrd.open_workbook(io)


170 elif engine == xlrd and isinstance(io, xlrd.Book):

171 self.book = io



/Users/mw4687/anaconda/lib/python3.4/site-packages/xlrd/__init__.py in open_
↪workbook(filename, logfile,
 verbosity, use_mmap, file_contents, encoding_override, formatting_info, on_demand,␣
↪ragged_rows)

392 peek = file_contents[:peeksz]
393 else:
--> 394 f = open(filename, "rb")
395 peek = f.read(peeksz)
396 f.close()
```

(continues on next page)

(continued from previous page)

```
FileNotFoundError: [Errno 2] No such file or directory: WHO POP TB same.xls
```

Jupyter notebooks show strings in red. If you see red characters until the end of the line, you have forgotten to type the second quote that marks the end of the string.

In the next section, find out how to select a column.

1.3 Selecting a column

Now you have the data, let the analysis begin!

Figure 3

Lets tackle the first part of the first question: What are the total, smallest, largest and average number of deaths due to TB? Obtaining the total number will be done in two steps: first select the column with the TB deaths, then sum the values in that column.

Selecting a single column of a dataframe is done with an expression in the format: ``**dataFrame[column name].**``

In []:

data['TB deaths']

Out[]:

[ ]:

```
0 6900
1 4400
2 41000
3 67
```

(continues on next page)

(continued from previous page)

```
4 1200
5 240000
6 18000
7 140
8 17000
9 18
10 25000
11 990
Name: TB deaths, dtype: int64
```

Strings are verbatim text, which means that the column name must be written exactly as given in the dataframe, which you saw after loading the data. The slightest deviation leads to a **key error** , which can be seen as a kind of name error. You can try out in the Week 2 exercise notebook what happens when misspelling the column name. The error message is horribly long. In such cases, just skip to the last line of the error message to see the type of error.

Put this learning into practice in Exercise 6.

Exercise 6 selecting a column

Question

In your Exercise notebook 1, select the population column and store it in a variable, so that you can use it in later exercises.

Remember that to open the notebook youll need to launch Anaconda and then navigate to the notebook using Jupyter. Once its open, run all the code.

Next, youll learn about making calculations on a column.

1.4 Calculations on a column

Having selected the column with the number of deaths per country, Ill add them with the appropriately named sum() method to obtain the overall total deaths.

A **method** is a function that can only be called in a certain context. In this course, the context will mostly be a dataframe or a column. A **method call** looks like a function call, but adds the context in which to call the method: `context.methodName(argument1, argument2, ...)` . In other words, a dataframe method can only be called on dataframes, a column method only on columns. Because methods are functions, a method call returns a value and is therefore an expression.

If all that sounded too abstract, heres how to call the `sum()` method on the TB deaths column. Note that `sum()` doesnt need any arguments because all the values are in the column.

In []:

[ ]:
```
tbColumn = data['TB deaths']
tbColumn.sum()
```

Out[]:

```
354715
```

The estimated total number of deaths due to TB in 2013 in the BRICS and Portuguese-speaking countries was over 350 thousand. An impressive number, for the wrong reasons.

Calculating the minimum and maximum number of deaths is done in a similar way.

```
In []:
```

```
tbColumn.min()
```

```
Out[]:
```

```
18
```

```
In []:
```

```
tbColumn.max()
```

```
Out[]:
```

```
240000
```

Like `sum()` , the column methods `min()` and `max()` dont need arguments, whereas the Python functions `min()` and `max()` did need them, because there was no context (column) providing the values.

The average number is computed as before, dividing the total by the number of countries.

```
In []:
```

```
tbColumn.sum() / 12
```

```
Out[]:
```

```
29559.583333333332
```

This kind of average is called the **mean** and theres a method for that.

```
In []:
```

```
tbColumn.mean()
```

```
Out[]:
```

```
29559.583333333332
```

Another kind of average measure is the **median** , which is the number in the middle, i.e.ăhalf of the values are above the median and half below it.

```
In []:
```

```
tbColumn.median()
```

```
Out[]:
```

```
5650.0
```

The mean is five times higher than the median. While half the countries had less than 5650 deaths in 2013, some countries had far more, which pushes the mean up.

The median is probably closer to the intuition you have of what average should mean (pun intended). News reports dont always make clear what average measure is being used, and using the mean may distort reality. For example, the mean household income in a country will be influenced by very poor and very rich households, whereas the median income doesnt take into account how poor or rich the extremes are: it will always be half the households below and half above the median.

Put this learning into practice in Exercise 7.

Exercise 7 calculations on a column

Question

Practise the use of column methods by applying them to the population column you obtained in Exercise 6 in the Exercise notebook 1. Remember to run all code before doing the exercise.

1.5 Sorting on a column

One of the research questions was: which countries have the smallest and largest number of deaths?

Being a small table, it is not too difficult to scan the TB deaths column and find those countries. However, such a process is prone to errors and impractical for large tables. Its much better to sort the table by that column, and then look up the countries in the first and last rows.

As youve guessed by now, sorting a table is another single line of code.

```
In []:
```

```
data.sort_values('TB deaths')
```

```
Out[]:
```

Country

Population (1000s)

TB deaths

9

Sao Tome and Principe

193

18

3

Equatorial Guinea

757

67

7

Portugal

10608

140

11

Timor-Leste

1133

990

4

Guinea-Bissau

1704

1200

1

Brazil

200362

4400

0

Angola

21472

6900

8

Russian Federation

142834

17000

6

Mozambique

25834

18000

10

South Africa

52776

25000

2

China

1393337

41000

5

India

1252140

240000

The dataframe method `sort_values()` takes as argument a column name and returns a new dataframe where the rows are in ascending order of the values in that column. Note that sorting doesnt modify the original dataframe.

```
In []:
```

```
data # rows still in original order
```

```
Out[]:
```

Country

Population (1000s)

TB deaths

0

Angola

21472

6900

1

Brazil

200362

4400

2

China

1393337

41000

3

Equatorial Guinea

757

67

4

Guinea-Bissau

1704

1200

5

India

1252140

240000

6

Mozambique

25834

18000

7

Portugal

10608

140

8

Russian Federation

142834

17000

9

Sao Tome and Principe

193

18

10

South Africa

52776

25000

11

Timor-Leste

1133

990

Its also possible to sort on a column that has text instead of numbers; the rows will be sorted in alphabetical order.

```
In []:
```

```
data.sort_values('Country')
```

```
Out[]:
```

Country

Population (1000s)

TB deaths

0

Angola

21472

6900

1

Brazil

200362

4400

2

China

1393337

41000

3

Equatorial Guinea

757

67

4

Guinea-Bissau

1704

1200

5

India

1252140

240000

6

Mozambique

25834

18000

7

Portugal

10608

140

8

Russian Federation

142834

17000

9

Sao Tome and Principe

193

18

10

South Africa

52776

25000

11

Timor-Leste

1133

990

Exercise 8 sorting on a column

Question

Use the Exercise notebook 1 to sort the table by population so that you can quickly see which are the least and the most populous countries. Remember to run all code before doing the exercise.

In the next section youll learn about calculations over columns.

1.6 Calculations over columns

The last remaining task is to calculate the death rate of each country.

You may recall that with the simple approach Id have to write:

[ ]:

```
rateAngola = deathsInAngola * 100 / populationOfAngola


rateBrazil = deathsInBrazil * 100 / populationOfBrazil
```

and so on, and so on. If youve used spreadsheets, its the same process: create the formula for the first row and then copy it down for all the rows. This is laborious and error-prone, e.g.ăif rows are added later on. Given that data is organised by columns, wouldnt it be nice to simply write the following?

`rateColumn = deathsColumn * 100 / populationColumn`

Say no more: your wish is pandass command.

In []:

[ ]:

```
deathsColumn = data['TB deaths']
populationColumn = data['Population (1000s)']
rateColumn = deathsColumn * 100 / populationColumn
rateColumn
```

Out[]:

```
0 32.134873

1 2.196025

2 2.942576

3 8.850727

4 70.422535

5 19.167186

6 69.675621

7 1.319759

8 11.901928

9 9.326425

10 47.370017
```

11 87.378641

dtype: float64

Tadaaa! With pandas, the arithmetic operators become much smarter. When adding, subtracting, multiplying or dividing columns, the computer understands that the operation is to be done row by row and creates a new column.

All well and nice, but how to put that new column into the dataframe, in order to have everything in a single table? In an assignment `variable = expression`, if the variable hasnt been mentioned before, the computer creates the variable and stores in it the expressions value. Likewise, if I assign to a column that doesnt exist in the dataframe, the computer will create it.

`In []:`

[ ]:

```
data['TB deaths (per 100,000)'] = rateColumn
data
```

`Out[]:`

Country

Population (1000s)

TB deaths

TB deaths (per 100,000)

0

Angola

21472

6900

32.134873

1

Brazil

200362

4400

2.196025

2

China

1393337

41000

2.942576

3

Equatorial Guinea

757

67

8.850727

4

Guinea-Bissau

1704

1200

70.422535

5

India

1252140

240000

19.167186

6

Mozambique

25834

18000

69.675621

7

Portugal

10608

140

1.319759

8

Russian Federation

142834

17000

11.901928

9

Sao Tome and Principe

193

18

9.326425

10

South Africa

52776

25000

47.370017

11

Timor-Leste

1133

990

87.378641

Thats it! Ive written all the code needed to answer the questions I had. Next Ill write up the analysis into a succinct and stand-alone notebook that can be shared with friends, family and colleagues or the whole world. Youll find that in the next section.

1.3.2 2 Writing up the analysis

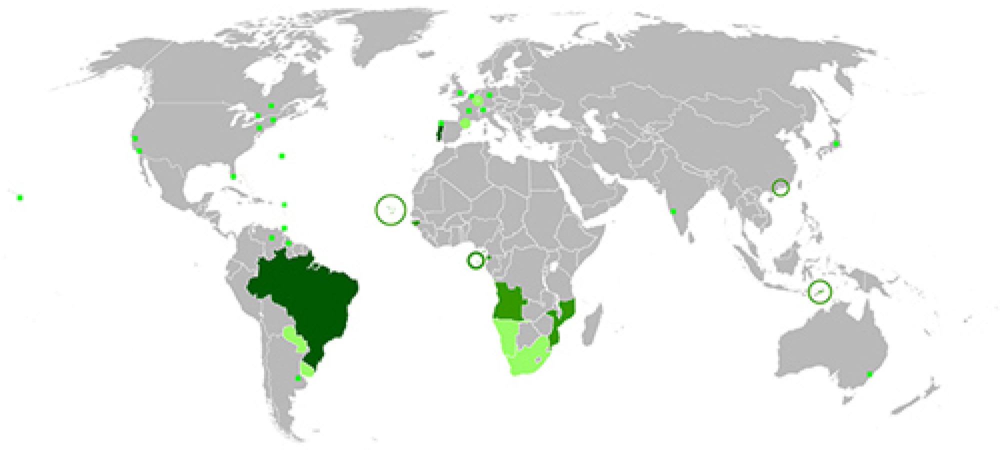


Figure 4 A map identifying Portuguese speaking countries

Once youve done your analysis, you may want to record it or share it with others. The best way is to write up what youve discovered.

There is no right or wrong way to write up data analysis but the important thing is to present the answers to the questions you had. To keep things simple, I suggest the following structure:

1. A descriptive title
2. An introduction setting the context and stating what you want to find out with the data.
3. A section detailing the source(s) of the data, with the code to load it into the notebook.
4. One or more sections showing the processes (calculating statistics, sorting the data, etc.) necessary to address the questions.
5. A conclusion summarising your findings, with qualitative analysis of the quantitative results and critical reflection on any shortcomings in the data or analysis process.

You dont need to explain your code, but its helpful to write the text in such a way that even readers who know nothing about Python or pandas can follow your analysis.

You can see how Ive written up the analysis by opening this weeks project notebook, which you can open in project_1: Deaths by tuberculosis .

In the next section, amend this project to produce your own version.

2.1 Practice project

Figure 5

Heres a quick project for you, which is about looking at TB deaths in *all* countries.

Activity 1 The project

Question

1. Open the data file WHO POP TB all.xls . Do **not** open or edit this file, to avoid changing its encoding. If you want to see the contents of the file, make a copy and look at the copy.
2. Open the project notebook.
3. If youre using CoCalc do the following two steps: Click on the File menu and select Download as and then IPython notebook (.ipynb).On your computer, rename the downloaded file so that it includes your name, e.g.ăTB deaths all world – Michel Wermelinger.ipynb. Then upload the renamed notebook to CoCalc and open it.
4. If youre using Anaconda do the following two steps: Click on the File menu and select Make a copy.Click on the title of the new notebook (project 1-Copy1) to rename it. Make sure to include your name in the file name, e.g.ăTB deaths all world – Michel Wermelinger.
5. In the new notebook, add your name to mine and update the date.
6. Edit the first code cell: change the file name to WHO POP TB all.xls, in order to load the data for all countries in the world.
7. Run all cells in the notebook. This might take a little while.
8. Add one line of code at the end to sort the table by the death rate, so that its easy to see the least and most affected countries.

9. Go through the notebook and change any text (in particular the conclusions) to reflect the new results.
10. Save and then close and halt the notebook.

If you happen to know how to use a spreadsheet application, then you can do a personal project: open the Excel file, remove all countries you are not interested in, and then do the analysis only for the remaining subset.

You might like to share your experience of working on this project with friends, family or colleagues.

2.2 Sharing your project notebook

Figure 6

Sharing work is a great way to solve problems and learn from others.

You are encouraged to share the analysis notebook that you created in the previous section. There are a few different ways you can do this. I will only mention two, sharing and publishing, depending on whether you want people to be able to change your notebook or only read it.

If you dont mind people editing and extending your notebook, like you have done with mine, then youll need to give them the notebook file (e.g.ăTB deaths all world – Michel Wermelinger.ipynb) and all necessary data files (just the WHO POP TB all.xls in this case). There are many ways you can share files with other people. One of the simplest is to create a zip archive, upload it to a cloud service like Dropbox or Google Drive, and publicise the download link. You could also share the link on your social media or via email.

If the intended recipients dont have the necessary software (Python, pandas and Jupyter) or you dont want anybody to change your notebook, you can still publish the analysis in read-only mode, i.e.ăpeople can read the text and code, see the resulting tables and numbers, but cant modify anything.

To do this, open your project notebook, run all the cells, double-check that there are no error messages and that all values and tables are shown as you want them to be, and save the notebook (without closing it).

If you use Anaconda, export the notebook by clicking Download as in the File menu and selecting the option you prefer. I prefer HTML because it looks much nicer. You can then share the single PDF or HTML file as before, by email, via Dropbox or Google Drive, on your blog and via a link.

If you use CoCalc, just click on the Publish button on the right side above your notebook, and you will get after a little while the link that you can share with others. Anyone can then read your notebook, even if they dont have a CoCalc account. For example, look at my Project 1 (its best to right-click and open this link in a new tab).

Now choose the sharing or publishing method, and get sharing!

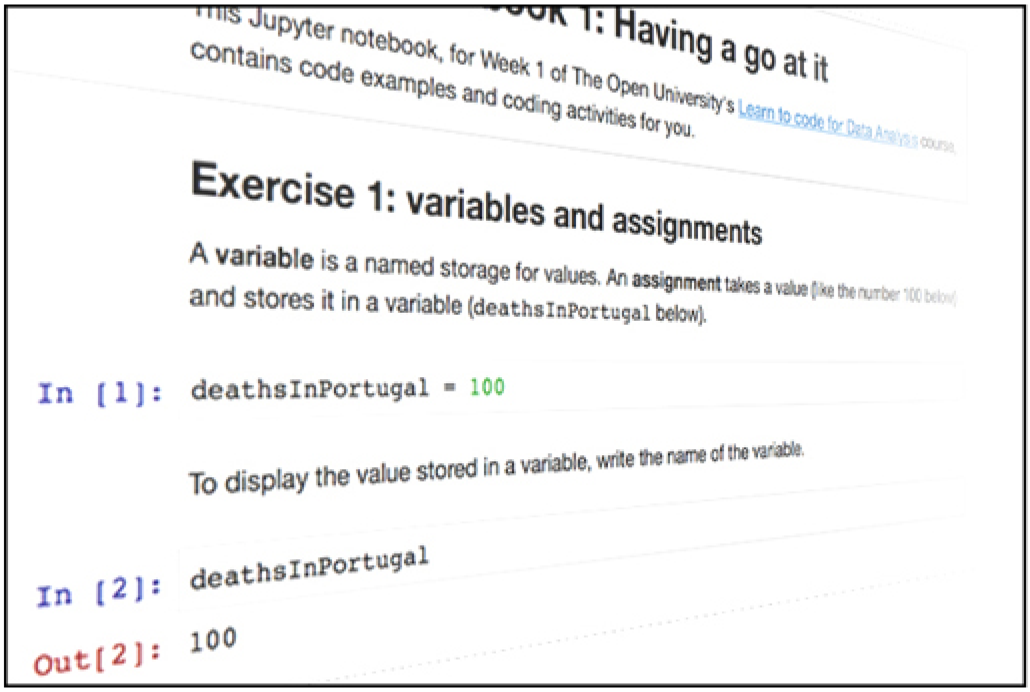
1.3.3 3 This weeks quiz

Check what youve learned this week by taking the end-of-week quiz.

Week 2 practice quiz

Open the quiz in a new window or tab then come back here when youve finished.

1.3.4 4 Summary



ook 1: Having a go at it

This Jupyter notebook, for Week 1 of The Open University's Learn to code for Data Analysis course, contains code examples and coding activities for you.

Exercise 1: variables and assignments

A **variable** is a named storage for values. An **assignment** takes a value (like the number 100 below) and stores it in a variable (`deathsInPortugal` below).

```
In [1]:  deathsInPortugal = 100
```

To display the value stored in a variable, write the name of the variable.

```
In [2]:  deathsInPortugal
Out[2]:  100
```

Figure 7

This week you used Jupyter notebooks to write and execute simple programs with Python and the pandas module. Youve learned how to:

- load a table from an Excel file
- select a column, and compute some simple statistics (like the total, minimum and median) about it.
- create a new column with values calculated from other columns
- sort a table by one of its columns.

Next week you will learn further ways to manipulate dataframes, in particular to clean data. You will also produce your first data chart, showing variations of values over time.

Futher reading

- WHO population – data by country (2013) **`link <>`__**
- WHO mortality and prevalence – data by country (2007 – present) **`link <>`__**

4.1 Week 1 and 2 glossary

Here are alphabetical lists, for quick look up, of what this week introduced.

Programming and data analysis concepts

An **assignment** is a statement of the form `variable = expression`. It evaluates the expression and stores its value in the variable. The variable is created if it doesnt exist. Each assignment is written on its own line.

CamelCase is a naming style in which names made of various words have each word capitalized, except possibly the first.

A **comment** is a note about the code. It starts with the hash sign (#) and goes until the end of the line.

A **dataframe** is the pandas representation of a table.

An **expression** is a fragment of code that can be **evaluated** , i.e.ăthat has a value, like a variable name.

A **file not found** error occurs if the computer cant find the given file, e.g.ăbecause the name is misspelled or because its in another folder.

A **function** takes zero or more **arguments** (values) and **returns** (produces) a value.

A **function call** is an expression of the form `functionName(argument1, argument2, )`.

An **import statement** of the form `from module import *` loads all the code from the given module.

The **maximum** and **minimum** of a set of values is the largest and smallest value, respectively.

The **mean** of a set of numbers is the sum of those numbers divided by how many there are.

The **median** of a set of numbers is the number in the middle, i.e.ăhalf of the numbers are below the median and half are above.

A **method** is a function that can only be called in a certain context, like a dataframe or a column.

A **method call** is an expression of the form `context.methodName(argument1, argument2, ...)`.

A **module** is a package of various pieces of code that can be used individually.

A **name** is a case-sensitive sequence of letters, digits and underscores. Names cannot start with a digit. Function, variable and module names usually start with lowercase.

A **name error** occurs if the computer doesnt recognize a name, e.g.ăif it was misspelled.

An **operator** is a symbol that represents some operation on one or two expressions, e.g.ăthe four basic arithmetic operators.

The **range** of a set of values is the difference between the maximum and the minimum.

A **reserved** word cannot be used as a name. Jupyter shows reserved words in green boldface.

A **statement** is a command for the computer to do something, e.g.ăto assign a value or to import some code.

A **string** is a verbatim piece of text, surrounded by quotes. Jupyter shows strings in red.

A **syntax error** occurs if the computer cant understand the code because it is not in the expected form, e.g.ăif a reserved word is used instead of a name or some punctuation is missing.

A **variable** is a named storage for values.

Reserved words

- `from`
- `import`

Functions and methods

`max(value1, value2, )` returns the maximum of the given values.

`column.max()` returns the maximum value in the column.

`min(value1, value2, )` returns the minimum of the given values.

`column.min()` returns the minimum value in the column.

`column.mean()` returns the mean of the values in the column.

`column.median()` returns the median of the values in the column.

`column.sum()` returns the total of the values in the column.

`dataFrame.sort_values(columnName)` takes a string with a columns name and returns a new dataframe, in which rows are sorted in ascending order according to the values in the given column.

`read_excel(fileName)` takes a string with an Excel file name, reads the file, and returns a dataframe representing the table in the file.

1.4 Session 03

1.5 Session 04

1.6 Session 05

1.7 Session 06

1.8 Session 07

1.9 Session 08