Introduction to python

Dr Claire L Hobday

Following software carpentry exercises: Plotting and programming in python

https://swcarpentry.github.io/python-novice-gapminder/reference/

This morning's content:

- Running and quitting python
- Variables and assignment
- Data types and type conversion
- Built in functions
- Libraries

Running and Quitting

Overview

Teaching: 15 min
Exercises: 0 min

Questions

- How can I run Python programs?

Objectives

- Launch the JupyterLab server.
- Create a new Python script.
- Create a Jupyter notebook.
- Shutdown the JupyterLab server.
- Understand the difference between a Python script and a Jupyter notebook.
- Create Markdown cells in a notebook.
- Create and run Python cells in a notebook.

Getting jupyter to work

Everyone should have python3 and jupyter installed on their PC.

Jupyter is included as part of the the Anaconda Python distribution.

Starting JupyterLab

Mac OS X

To start the JupyterLab server you will need to access the command line through the Terminal. There are two ways to open Terminal on Mac.

1. In your Applications folder, open Utilities and double-click on Terminal
2. Press `Command` + `spacebar` to launch Spotlight. Type `Terminal` and then double-click the search result or hit `Enter`

After you have launched Terminal, type the command to launch the JupyterLab server.

```
Bash
$ jupyter lab
```

Windows Users

To start the JupyterLab server you will need to access the open Anaconda Prompt.

Press `Windows Logo Key` and search for `Anaconda Prompt`, click the result or press enter.

After you have launched the Anaconda Prompt, type the command:

```
Bash
$ jupyter lab
```

Jupyter provides an interactive way of writing and running python programs

You can:

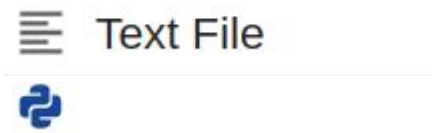
- test code as you go
- see what graphs will look like
- try out new things

before you make it one big long script to execute

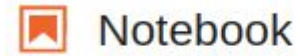
Using jupyter lab

1. Creating a text file

   Converting to .py file



2. Creating a jupyter notebook



3. Opening a python console

4. Arranging your workspace

   TRY:

Use the Jupyter Notebook for editing and running Python.

- While it's common to write Python scripts using a text editor, we are going to use the Jupyter Notebook for the remainder of this workshop.
- This has several advantages:
  - You can easily type, edit, and copy and paste blocks of code.
  - Tab complete allows you to easily access the names of things you are using and learn more about them.
  - It allows you to annotate your code with links, different sized text, bullets, etc. to make it more accessible to you and your collaborators.
  - It allows you to display figures next to the code that produces them to tell a complete story of the analysis.
- Each notebook contains one or more cells that contain code, text, or images.

📌 Code vs. Text

Jupyter mixes code and text in different types of blocks, called cells. We often use the term "code" to mean "the source code of software written in a language such as Python". A "code cell" in a Notebook is a cell that contains software; a "text cell" is one that contains ordinary prose written for human beings.

The Notebook has Command and Edit modes.

- If you press `Esc` and `Return` alternately, the outer border of your code cell will change from gray to blue.
- These are the **Command** (gray) and **Edit** (blue) modes of your notebook.
- Command mode allows you to edit notebook-level features, and Edit mode changes the content of cells.
- When in Command mode (esc/gray),
  - The `b` key will make a new cell below the currently selected cell.
  - The `a` key will make one above.
  - The `x` key will delete the current cell.
  - The `z` key will undo your last cell operation (which could be a deletion, creation, etc).
- All actions can be done using the menus, but there are lots of keyboard shortcuts to speed things up.

Command Vs. Edit

In the Jupyter notebook page are you currently in Command or Edit mode?
Switch between the modes. Use the shortcuts to generate a new cell. Use the shortcuts to delete a cell. Use the shortcuts to undo the last cell operation you performed.

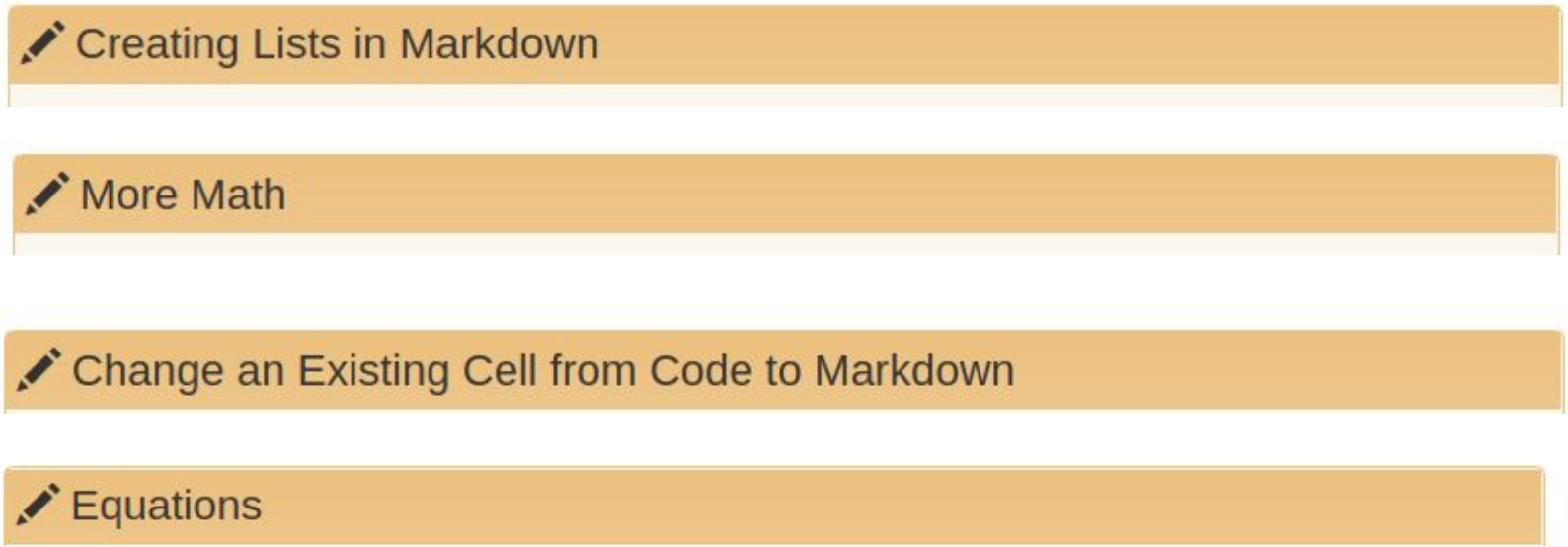
Use the keyboard and mouse to select and edit cells.

- Pressing the `Return` key turns the border blue and engages Edit mode, which allows you to type within the cell.
- Because we want to be able to write many lines of code in a single cell, pressing the `Return` key when in Edit mode (blue) moves the cursor to the next line in the cell just like in a text editor.
- We need some other way to tell the Notebook we want to run what's in the cell.
- Pressing `Shift`+`Return` together will execute the contents of the cell.
- Notice that the `Return` and `Shift` keys on the right of the keyboard are right next to each other.

The Notebook will turn Markdown into pretty-printed documentation.

- Notebooks can also render Markdown.
  - A simple plain-text format for writing lists, links, and other things that might go into a web page.
  - Equivalently, a subset of HTML that looks like what you'd send in an old-fashioned email.
- Turn the current cell into a Markdown cell by entering the Command mode (`Esc`/gray) and press the `M` key.
- `In [ ]:` will disappear to show it is no longer a code cell and you will be able to write in Markdown.
- Turn the current cell into a Code cell by entering the Command mode (`Esc`/gray) and press the `y` key.

Try some of the exercises in the carpentry page



Closing JupyterLab

- From the Menu Bar select the “File” menu and the choose “Quit” at the bottom of the dropdown menu. You will be prompted to confirm that you wish to shutdown the JupyterLab server (don’t forget to save your work!). Click “Confirm” to shutdown the JupyterLab server.
- To restart the JupyterLab server you will need to re-run the following command from a shell.

```
$ jupyter lab
```

Closing JupyerLab

Practice closing and restarting the JupyterLab server.

Key Points

- Python scripts are plain text files.
- Use the Jupyter Notebook for editing and running Python.
- The Notebook has Command and Edit modes.
- Use the keyboard and mouse to select and edit cells.
- The Notebook will turn Markdown into pretty-printed documentation.
- Markdown does most of what HTML does.

Variables and Assignment

Overview

Teaching: 10 min
Exercises: 10 min
Questions

- How can I store data in programs?

Objectives

- Write programs that assign scalar values to variables and perform calculations with those values.
- Correctly trace value changes in programs that use scalar assignment.

Use variables to store values.

- **Variables** are names for values.
- In Python the `=` symbol assigns the value on the right to the name on the left.
- The variable is created when a value is assigned to it.
- Here, Python assigns an age to a variable `age` and a name in quotes to a variable `first_name`.

Python

```
age = 42
first_name = 'Ahmed'
```

- Variable names
  - can **only** contain letters, digits, and underscore `_` (typically used to separate words in long variable names)
  - cannot start with a digit
- Variable names that start with underscores like `__alistairs_real_age` have a special meaning so we won't do that until we understand the convention.

Variables must be created before they are used.

- If a variable doesn't exist yet, or if the name has been mis-spelled, Python reports an error. (Unlike some languages, which "guess" a default value.)

Python

```
print(last_name)
```

Error

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-c1fbb4e96102> in <module>()
----> 1 print(last_name)

NameError: name 'last_name' is not defined
```

- The last line of an error message is usually the most informative.
- We will look at error messages in detail later.

📌 Variables Persist Between Cells

Be aware that it is the *order* of execution of cells that is important in a Jupyter notebook, not the order in which they appear. Python will remember *all* the code that was run previously, including any variables you have defined, irrespective of the order in the notebook. Therefore if you define variables lower down the notebook and then (re)run cells further up, those defined further down will still be present. As an example, create two cells with the following content, in this order:

Python

```
print(myval)
```

Python

```
myval = 1
```

If you execute this in order, the first cell will give an error. However, if you run the first cell *after* the second cell it will print out `1`. To prevent confusion, it can be helpful to use the `Kernel` -> `Restart & Run All` option which clears the interpreter and runs everything from a clean slate going top to bottom.

Variables can be used in calculations.

- We can use variables in calculations just as if they were values.
  - Remember, we assigned the value `42` to `age` a few lines ago.

Python

```
age = age + 3
print('Age in three years:', age)
```

Output

```
Age in three years: 45
```

Use an index to get a single character from a string.

- The characters (individual letters, numbers, and so on) in a string are ordered. For example, the string `'AB'` is not the same as `'BA'`. Because of this ordering, we can treat the string as a list of characters.
- Each position in the string (first, second, etc.) is given a number. This number is called an **index** or sometimes a subscript.
- Indices are numbered from 0.
- Use the position's index in square brackets to get the character at that position.

```
print(atom_name[0])
```

Python

```
atom_name = 'helium'
print(atom_name[0])
```

Output

```
h
```

Use a slice to get a substring.

- A part of a string is called a **substring**. A substring can be as short as a single character.
- An item in a list is called an element. Whenever we treat a string as if it were a list, the string's elements are its individual characters.
- A slice is a part of a string (or, more generally, any list-like thing).
- We take a slice by using `[start:stop]`, where `start` is replaced with the index of the first element we want and `stop` is replaced with the index of the element just after the last element we want.
- Mathematically, you might say that a slice selects `[start:stop)`.
- The difference between `stop` and `start` is the slice's length.
- Taking a slice does not change the contents of the original string. Instead, the slice is a copy of part of the original string.

Python

```
atom_name = 'sodium'
print(atom_name[0:3])
```

Output

```
sod
```

Use the built-in function `len` to find the length of a string.

Python

```
print(len('helium'))
```

Output

```
6
```

- Nested functions are evaluated from the inside out, like in mathematics.

Python is case-sensitive.

- Python thinks that upper- and lower-case letters are different, so `Name` and `name` are different variables.
- There are conventions for using upper-case letters at the start of variable names so we will use lower-case letters for now.

MOST IMPORTANT THING YOU WILL LEARN TODAY!

Use meaningful variable names.

- Python doesn't care what you call variables as long as they obey the rules (alphanumeric characters and the underscore).

Python

```
flabadab = 42
ewr_422_yY = 'Ahmed'
print(ewr_422_yY, 'is', flabadab, 'years old')
```

- Use meaningful variable names to help other people understand what the program does.
- The most important "other person" is your future self.

Try some of the exercises in the carpentry page

- Swapping Values
- Predicting Values
- Challenge
- Choosing a Name
- Slicing practice
- Slicing concepts

Key Points

- Use variables to store values.
- Use `print` to display values.
- Variables persist between cells.
- Variables must be created before they are used.
- Variables can be used in calculations.
- Use an index to get a single character from a string.
- Use a slice to get a substring.
- Use the built-in function `len` to find the length of a string.
- Python is case-sensitive.
- Use meaningful variable names.

Data Types and Type Conversion

Overview

Teaching: 10 min
Exercises: 10 min
Questions

- What kinds of data do programs store?
- How can I convert one type to another?

Objectives

- Explain key differences between integers and floating point numbers.
- Explain key differences between numbers and character strings.
- Use built-in functions to convert between integers, floating point numbers, and strings.

Every value has a type.

- Every value in a program has a specific type.
- Integer (`int`): represents positive or negative whole numbers like 3 or -512.
- Floating point number (`float`): represents real numbers like 3.14159 or -2.5.
- Character string (usually called “string”, `str`): text.
  - Written in either single quotes or double quotes (as long as they match).
  - The quote marks aren’t printed when the string is displayed.

Use the built-in function `type` to find the type of a value.

- Use the built-in function `type` to find out what type a value has.
- Works on variables as well.
  - But remember: the *value* has the type — the *variable* is just a label.

Types control what operations (or methods) can be performed on a given value.

- A value's type determines what the program can do to it.

You can use the "+" and "*" operators on strings.

- "Adding" character strings concatenates them.
- Multiplying a character string by an integer *N* creates a new string that consists of that character string repeated *N* times.
  - Since multiplication is repeated addition.

Strings have a length (but numbers don't).

- The built-in function `len` counts the number of characters in a string.
- But numbers don't have a length (not even zero).

Must convert numbers to strings or vice versa when operating on them.

- Cannot add numbers and strings.

Can mix integers and floats freely in operations.

- Integers and floating-point numbers can be mixed in arithmetic.
  - Python 3 automatically converts integers to floats as needed. (Integer division in Python 2 will return an integer, the *floor* of the division.)

Variables only change value when something is assigned to them.

- If we make one cell in a spreadsheet depend on another, and update the latter, the former updates automatically.
- This does **not** happen in programming languages.

Try some of the exercises in the carpentry page

- Fractions
- Automatic Type Conversion
- Choose a Type
- Division Types
- Arithmetic with Different Types
- Strings to Numbers
- Complex Numbers

Key Points

- Every value has a type.
- Use the built-in function `type` to find the type of a value.
- Types control what operations can be done on values.
- Strings can be added and multiplied.
- Strings have a length (but numbers don't).
- Must convert numbers to strings or vice versa when operating on them.
- Can mix integers and floats freely in operations.
- Variables only change value when something is assigned to them.

Built-in Functions and Help

Overview

Teaching: 15 min
Exercises: 10 min
Questions

- How can I use built-in functions?
- How can I find out what they do?
- What kind of errors can occur in programs?

Objectives

- Explain the purpose of functions.
- Correctly call built-in Python functions.
- Correctly nest calls to built-in functions.
- Use help to display documentation for built-in functions.
- Correctly describe situations in which SyntaxError and NameError occur.

Use comments to add documentation to programs.

Python

```
# This sentence isn't executed by Python.
adjustment = 0.5   # Neither is this - anything after '#' is ignored.
```

This helps you and other people work out what you have done.

In addition to calling variables sensible things, commenting your programs is one of the most important practises you can develop as a coder.

You will thank yourself later

A function may take zero or more arguments.

- We have seen some functions already — now let's take a closer look.
- An *argument* is a value passed into a function.
- `len` takes exactly one.
- `int`, `str`, and `float` create a new value from an existing one.
- `print` takes zero or more.
- `print` with no arguments prints a blank line.
  - Must always use parentheses, even if they're empty, so that Python knows a function is being called.

Python

```
print('before')
print()
print('after')
```

Output

```
before

after
```

Commonly-used built-in functions include `max`, `min`, and `round`.

- Use `max` to find the largest value of one or more values.
- Use `min` to find the smallest.
- Both work on character strings as well as numbers.
  - "Larger" and "smaller" use (0-9, A-Z, a-z) to compare letters.

Python

```
print(max(1, 2, 3))
print(min('a', 'A', '0'))
```

Output

```
3
0
```

Functions may only work for certain (combinations of) arguments.

- `max` and `min` must be given at least one argument.
  - "Largest of the empty set" is a meaningless question.
- And they must be given things that can meaningfully be compared.

Python

```
print(max(1, 'a'))
```

Error

```
TypeError                                 Traceback (most recent call last)
<ipython-input-52-3f049acf3762> in <module>
----> 1 print(max(1, 'a'))

TypeError: '>' not supported between instances of 'str' and 'int'
```

Functions may have default values for some arguments.

- `round` will round off a floating-point number.
- By default, rounds to zero decimal places.

Python

```
round(3.712)
```

Output

```
4
```

- We can specify the number of decimal places we want.

Python

```
round(3.712, 1)
```

Output

```
3.7
```

Use the built-in function `help` to get help for a function.

- Every built-in function has online documentation.

Python

```
help(round)
```

Output

```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.  Otherwise
    the return value has the same type as the number.  ndigits may be negative.
```

Python reports a syntax error when it can't understand the source of a program.

- Won't even try to run the program if it can't be parsed.

Python

```
# Forgot to close the quote marks around the string.
name = 'Feng
```

Error

```
  File "<ipython-input-56-f42768451d55>", line 2
    name = 'Feng
                ^
SyntaxError: EOL while scanning string literal
```

Python reports a runtime error when something goes wrong while a program is executing.

Python

```
age = 53
remaining = 100 - aege # mis-spelled 'age'
```

Error

```
NameError                                 Traceback (most recent call last)
<ipython-input-59-1214fb6c55fc> in <module>
      1 age = 53
----> 2 remaining = 100 - aege # mis-spelled 'age'

NameError: name 'aege' is not defined
```

- Fix syntax errors by reading the source and runtime errors by tracing execution.

The Jupyter Notebook has two ways to get help.

- Place the cursor anywhere in the function invocation (i.e., the function name or its parameters), hold down `shift`, and press `tab`.
- Or type a function name with a question mark after it.

This is super informative, telling you how a function can be used.

```
: max?
```

```
Docstring:
max(iterable, *[, default=obj, key=func]) -> value
max(arg1, arg2, *args, *[, key=func]) -> value

With a single iterable argument, return its biggest item. The
default keyword-only argument specifies an object to return if
the provided iterable is empty.
With two or more arguments, return the largest argument.
Type:      builtin_function_or_method
```

Every function returns something.

- Every function call produces some result.
- If the function doesn't have a useful result to return, it usually returns the special value `None`.

Python

```
result = print('example')
print('result of print is', result)
```

Output

```
example
result of print is None
```

Try some of the exercises in the carpentry page

- What Happens When
- Spot the Difference
- Why Not?
- Last Character of a String

Key Points

- Use comments to add documentation to programs.
- A function may take zero or more arguments.
- Commonly-used built-in functions include `max`, `min`, and `round`.
- Functions may only work for certain (combinations of) arguments.
- Functions may have default values for some arguments.
- Use the built-in function `help` to get help for a function.
- The Jupyter Notebook has two ways to get help.
- Every function returns something.
- Python reports a syntax error when it can't understand the source of a program.
- Python reports a runtime error when something goes wrong while a program is executing.
- Fix syntax errors by reading the source code, and runtime errors by tracing the program's execution.

Libraries

Overview

Teaching: 10 min
Exercises: 10 min
Questions

- How can I use software that other people have written?
- How can I find out what that software does?

Objectives

- Explain what software libraries are and why programmers create and use them.
- Write programs that import and use libraries from Python's standard library.
- Find and read documentation for standard libraries interactively (in the interpreter) and online.

Most of the power of a programming language is in its libraries.

- A *library* is a collection of files (called *modules*) that contains functions for use by other programs.
  - May also contain data values (e.g., numerical constants) and other things.
  - Library's contents are supposed to be related, but there's no way to enforce that.
- The Python standard library is an extensive suite of modules that comes with Python itself.
- Many additional libraries are available from PyPI (the Python Package Index).
- We will see later how to write new libraries.

📌 Libraries and modules

A library is a collection of modules, but the terms are often used interchangeably, especially since many libraries only consist of a single module, so don't worry if you mix them.

A program must import a library module before using it.

- Use `import` to load a library module into a program's memory.
- Then refer to things from the module as `module_name.thing_name`.
  - Python uses `.` to mean "part of".
- Using `math`, one of the modules in the standard library:

Python

```
import math

print('pi is', math.pi)
print('cos(pi) is', math.cos(math.pi))
```

Output

```
pi is 3.141592653589793
cos(pi) is -1.0
```

- Have to refer to each item with the module's name.
  - `math.cos(pi)` won't work: the reference to `pi` doesn't somehow "inherit" the function's reference to `math`.

Import specific items from a library module to shorten programs.

- Use `from ... import ...` to load only specific items from a library module.
- Then refer to them directly without library name as prefix.

Python

```
from math import cos, pi

print('cos(pi) is', cos(pi))
```

Output

```
cos(pi) is -1.0
```

Chemistry specific libraries?

-ase

Atomic simulation environment

-pymatgen

Python Materials Genomics

-biopython

-astropy

Let’s work through some examples of what these libraries could help you do!

There are lots of libraries, more than listed here, this is just to give you a flavour!

Try some of the exercises in the carpentry page

- Exploring the Math Module
- Locating the Right Module
- Jigsaw Puzzle (Parson's Problem) Programming Example
- When Is Help Available?
- Importing With Aliases
- Importing With Aliases
- Importing Specific Items
- Reading Error Messages

Libraries

Key Points

- Most of the power of a programming language is in its libraries.
- A program must import a library module in order to use it.
- Use `help` to learn about the contents of a library module.
- Import specific items from a library to shorten programs.
- Create an alias for a library when importing it to shorten programs.