How to figure things out in Python

1) You can get documentation for nearly any function and package.

```
>>> help(print)  # to get documentation about the print function
```

Anything after "#" on a line is ignored by Python, it's a **comment** for your understanding.

Someone told me/I read online that comments are unnecessary. **Don't listen to them.**

2) If you have the name of some data structure, you can ask what type it is.

```
>>> type(__name__)  # get the type, a.k.a. the class, of __name__
<class 'str'>
```

3) Given some data structure, look inside it.

```
>>> dir(__name__)  # look inside the __name__ structure
```

4) There's google.

The Python Ecosystem

- Although we will focus on the core Python language, the true strength of Python is the large body of available additional modules.
- These modules provide all kinds of functionality.
- There are many modules in the standard library that comes with Python ("batteries included").
  E.g. modules for GUIs, databases, random numbers, regular expressions, testing, ...
- There are even more third-party modules available.
- The official repository for third-party modules is the Python Package Index (http://pypi.Python.org/pypi) with over 100,000 packages.
- Most Python distributions come with the "`pip`" command, with which you can install packages from pypi.
  (For Anaconda, you'd use the `conda` command instead).

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Different interfaces to Python

There are a number of ways to use Python:

1. Standard, non-interactive mode of Python
   This requires an existing Python script. Simply open a terminal and type `python <SCRIPTNAME>`, and the code gets executed.
2. Standard, interactive mode of Python
   Simply open a terminal and type `python`, and you get a prompt like >>>.
   You can type commands at the prompt, they get executed, then you get another prompt.
3. IPython interactive mode
   Requires IPython installation. Then type "`ipython`", and you get a prompt that looks like "`In [1]:`".
   Features tab completion, command history, and special commands.
4. Jupyter notebooks
   Input and output cells in your browser, with the Python back-end running possibly remotely. Harder to convert to scripts, and a bit more up-front work to get going.
   https://jupyter.scinet.utoronto.ca

Interpretation

What happens when we type "`print(9999+11111)`" on the Python prompt?

- First, note that Python was waiting for input, and allows you to edit that input. It doesn't 'do' anything until you hit enter.

  (in case of IPython, you can scroll through history and use tab completion, which are not 'doing nothing', but are still not doing Python)

- Once you hit enter, Python will check syntax, identifying functions, keywords, arguments, special characters, ...

- If it makes sense syntactically, it will then execute that command, i.e. translate it into (nested) function calls that at the lowest level are in machine code that the CPU understands.

- Python does this one line at a time, which puts it in the category of **interpreted languages**.

Example: 9999+11111

```
>>> print(9999+11111)
```

First action by Python: Syntax checking

- `print` is a name.
- It should be a function, because it is followed by parentheses.
- The argument of the function is 9999+11111
- This is two 'literals' (numbers), separated by the plus sign, which is valid.

Second action by Python: Execution

- Store the integer 9999.
- Store the integer 11111.
- Call the + operator, with those integers as arguments.
- This "returns" a new integer.
- The print function is called with that new, temporary integer as an argument.
- Temporary integers are discarded.

What are modules and packages?

- Modules are python files.
- Modules are meant to be imported into other python files or scripts.
- Convenient way to store functions that you might use in multiple projects.
- One or more modules can form a package.

Example

module file:

```
# file: yearprop.py
"""Deal with properties of calendar years."""
def is_leap_year(year):
  """Determines if a give year is a leap year.
  Argument year is the year to investigate.
  Returns True is year is a leap year, else
  False.
  """
  ...
```

usage in another script:

```
# file: yearquery.py
"""Ask for years and say if they're leap years"
import yearprop

# use the function yearprop.is_leap_year
year = int(input("Give a year"))
if yearprop.is_leap_year(year):
  print(year,"is a leap year")
else:
  print(year,"is not a leap year")
```

Modules - Details

- We do not have to specify `.py` in the `import` statement.
- The file name without `.py` is the name of the module.
- The module file has to be in the same directory as the script, unless you install it (*later*)
- When putting functions in a module and importing that module, it gets put in the **namespace** of the modules. The name of the namespace is the name of the module. (in the example, the namespace was `yearprop`)

You can change the namespace that the modules functions end up in

- Changing the name of the module: `import yearprop as yp`
- Importing specific functions: `from yearprop import is_leap_year`
- Importing everything: `from yearprop *`