



# The Problem

**Computing and programming skills are necessary for all scientists.**

- large datasets
- 2D/3D visualisations
- numerical models
- complex statistics
- and more...

**Traditional geoscience curricula do not teach these skills.**

# Our Solution

**Programming & modelling using Python now central to 1st year core curriculum in University of Wollongong School of Earth & Environmental Sciences**

**Embedded in practicals for *Earth's Interconnected Spheres*:**

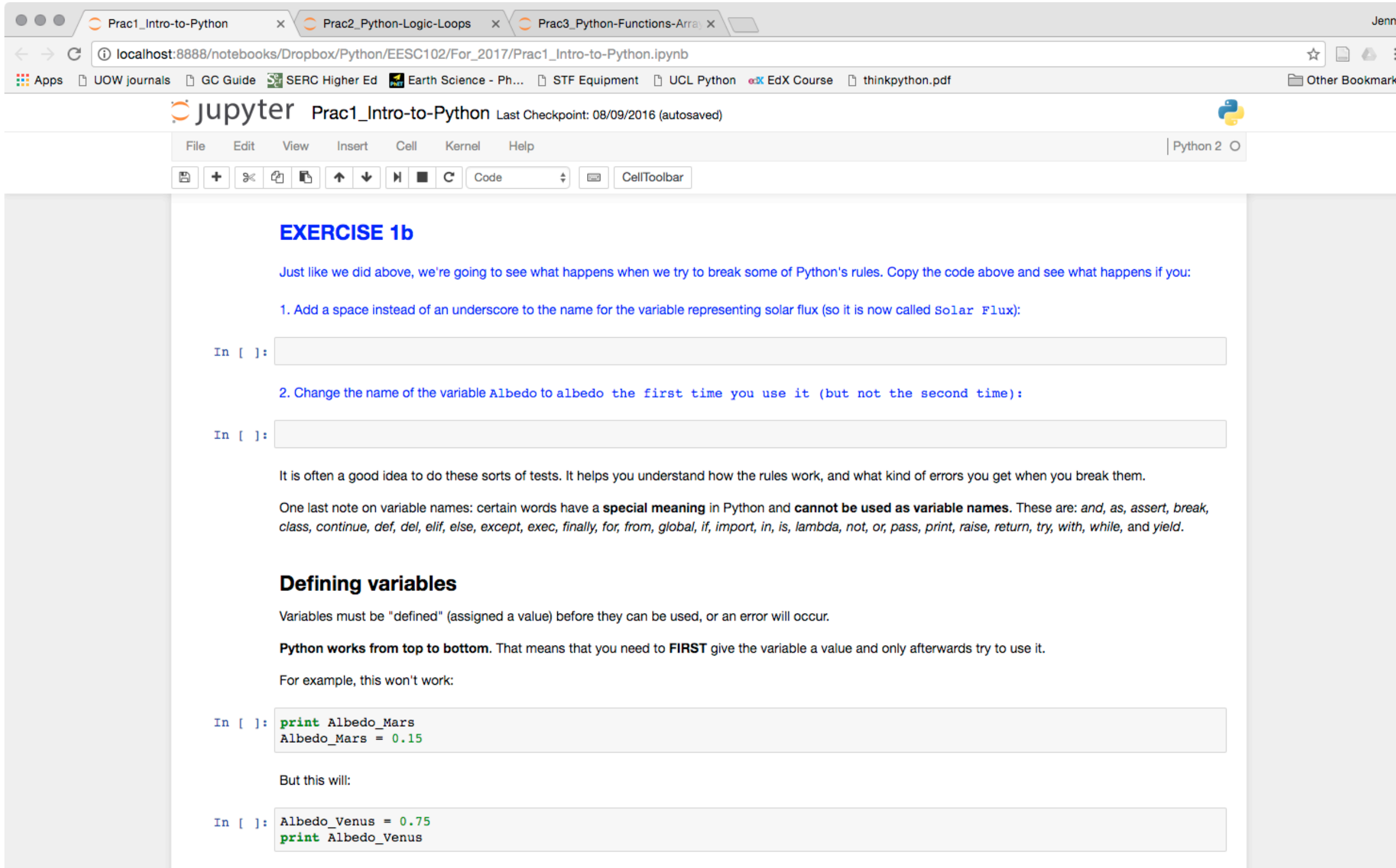
- **1 of 4 compulsory core subjects for all majors**
- **100-120 students**
- **no mathematics prerequisites and no assumed prior programming experience**

**Why Python?**

- **free, easy to install, and widely used**
- **easy to learn “natural” syntax (`print "Hello"`)**
- **simple but ultimately powerful, including geoscience integration (e.g. ArcGIS)**

# Implementation

## Exercises in *Jupyter Notebook*:



The screenshot shows a Jupyter Notebook interface in a web browser. The browser tabs include 'Prac1\_Intro-to-Python', 'Prac2\_Python-Logic-Loops', and 'Prac3\_Python-Functions-Arra...'. The address bar shows the URL 'localhost:8888/notebooks/Dropbox/Python/EESC102/For\_2017/Prac1\_Intro-to-Python.ipynb'. The notebook title is 'Prac1\_Intro-to-Python' and it shows the last checkpoint as '08/09/2016 (autosaved)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help) and a toolbar with icons for adding, deleting, and running cells. The main content area is titled 'EXERCISE 1b' and contains the following text:

Just like we did above, we're going to see what happens when we try to break some of Python's rules. Copy the code above and see what happens if you:

1. Add a space instead of an underscore to the name for the variable representing solar flux (so it is now called `Solar Flux`):

In [ ]:

2. Change the name of the variable `Albedo` to `albedo` the first time you use it (but not the second time):

In [ ]:

It is often a good idea to do these sorts of tests. It helps you understand how the rules work, and what kind of errors you get when you break them.

One last note on variable names: certain words have a **special meaning** in Python and **cannot be used as variable names**. These are: `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `exec`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `not`, `or`, `pass`, `print`, `raise`, `return`, `try`, `with`, `while`, and `yield`.

### Defining variables

Variables must be "defined" (assigned a value) before they can be used, or an error will occur.

**Python works from top to bottom.** That means that you need to **FIRST** give the variable a value and only afterwards try to use it.

For example, this won't work:

```
In [ ]: print Albedo_Mars
Albedo_Mars = 0.15
```

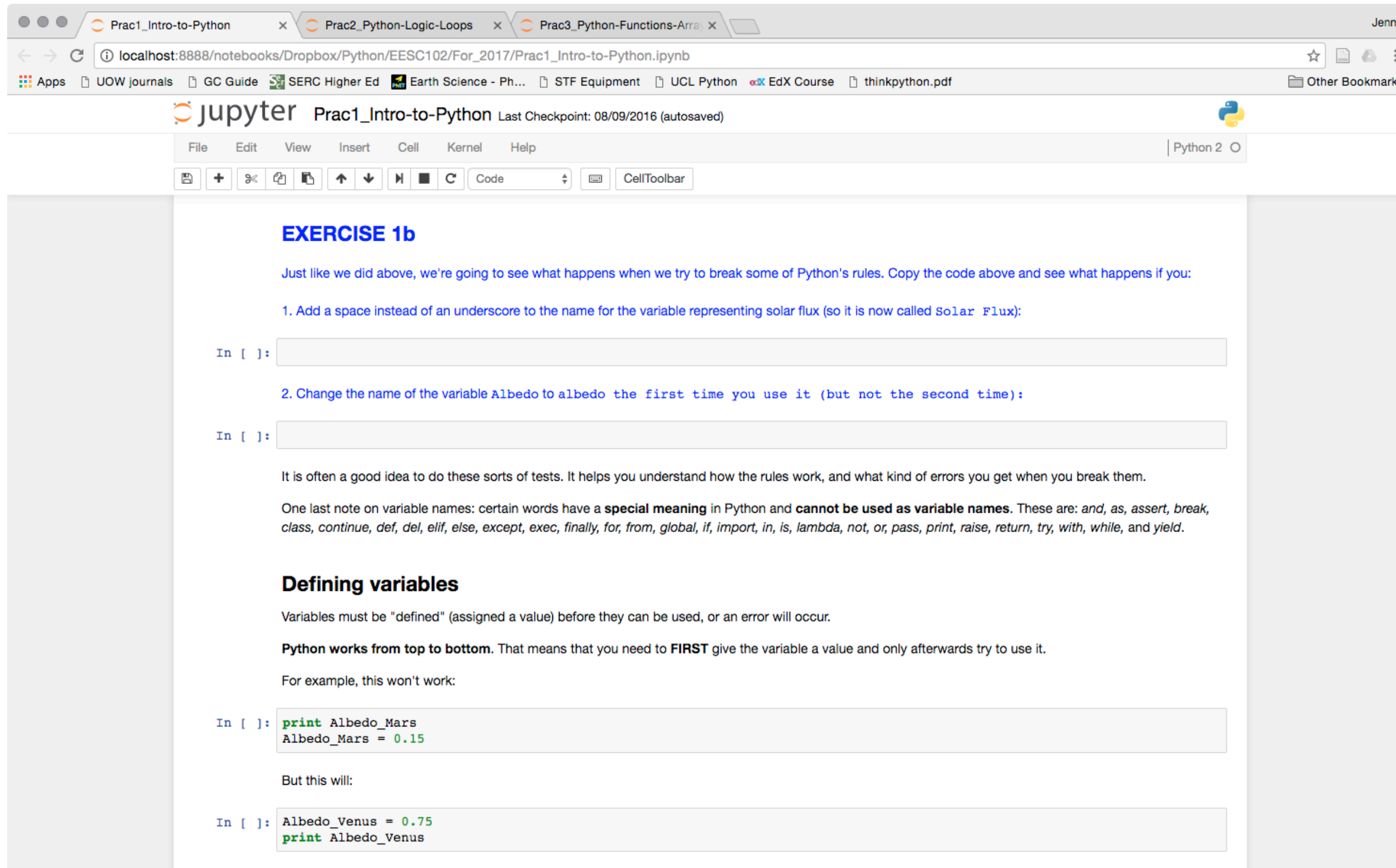
But this will:

```
In [ ]: Albedo_Venus = 0.75
print Albedo_Venus
```



# Implementation

## Exercises in *Jupyter Notebook*:



The screenshot shows a Jupyter Notebook interface with the following content:

**EXERCISE 1b**

Just like we did above, we're going to see what happens when we try to break some of Python's rules. Copy the code above and see what happens if you:

1. Add a space instead of an underscore to the name for the variable representing solar flux (so it is now called `Solar Flux`):

In [ ]:

2. Change the name of the variable `Albedo` to `albedo` the first time you use it (but not the second time):

In [ ]:

It is often a good idea to do these sorts of tests. It helps you understand how the rules work, and what kind of errors you get when you break them.

One last note on variable names: certain words have a **special meaning** in Python and **cannot be used as variable names**. These are: `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `exec`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `not`, `or`, `pass`, `print`, `raise`, `return`, `try`, `with`, `while`, and `yield`.

**Defining variables**

Variables must be "defined" (assigned a value) before they can be used, or an error will occur.

**Python works from top to bottom.** That means that you need to **FIRST** give the variable a value and only afterwards try to use it.

For example, this won't work:

```
In [ ]: print Albedo_Mars
Albedo_Mars = 0.15
```

But this will:

```
In [ ]: Albedo_Venus = 0.75
print Albedo_Venus
```

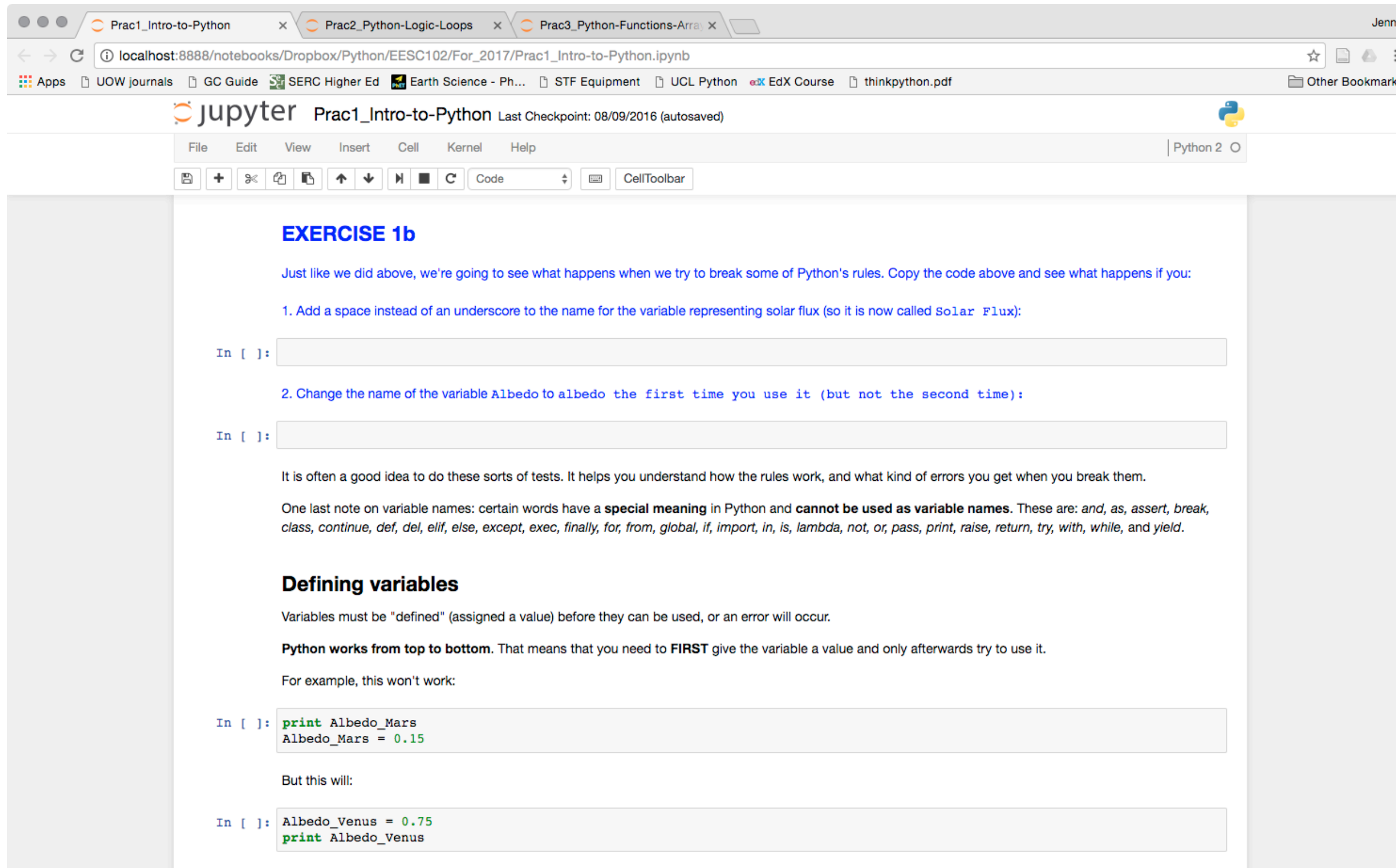
Weeks 1-4: computing fundamentals

5-8: simple box models

9-12: group projects

# Implementation

## Exercises in *Jupyter Notebook*:



The screenshot shows a Jupyter Notebook with the following content:

### EXERCISE 1b

Just like we did above, we're going to see what happens when we try to break some of Python's rules. Copy the code above and see what happens if you:

1. Add a space instead of an underscore to the name for the variable representing solar flux (so it is now called `Solar Flux`):

In [ ]:

2. Change the name of the variable `Albedo` to `albedo` the first time you use it (but not the second time):

In [ ]:

It is often a good idea to do these sorts of tests. It helps you understand how the rules work, and what kind of errors you get when you break them.

One last note on variable names: certain words have a **special meaning** in Python and **cannot be used as variable names**. These are: `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `exec`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `not`, `or`, `pass`, `print`, `raise`, `return`, `try`, `with`, `while`, and `yield`.

### Defining variables

Variables must be "defined" (assigned a value) before they can be used, or an error will occur.

**Python works from top to bottom.** That means that you need to **FIRST** give the variable a value and only afterwards try to use it.

For example, this won't work:

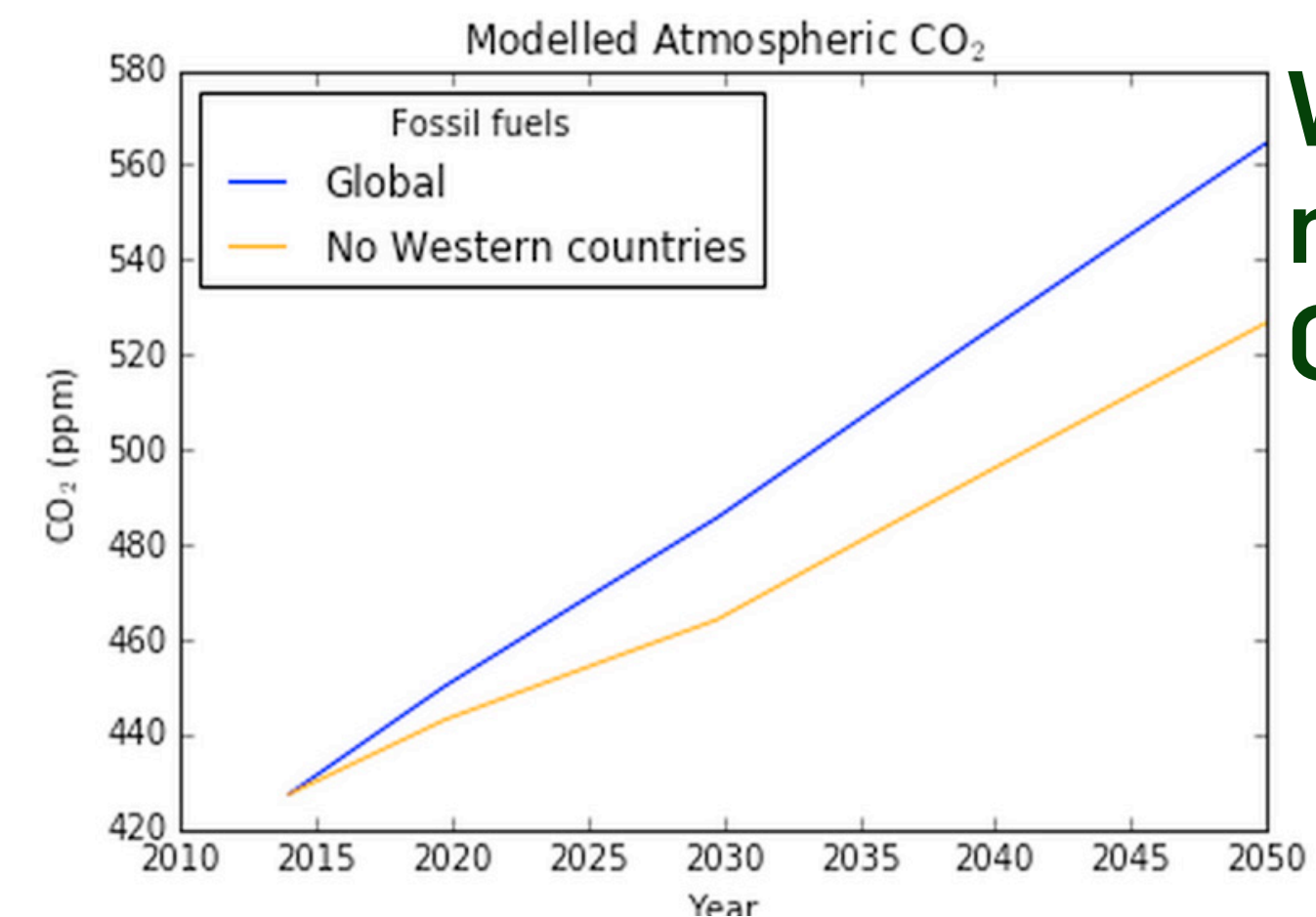
```
In [ ]: print Albedo_Mars
Albedo_Mars = 0.15
```

But this will:

```
In [ ]: Albedo_Venus = 0.75
print Albedo_Venus
```

Weeks 1-4: computing fundamentals  
5-8: simple box models  
9-12: group projects

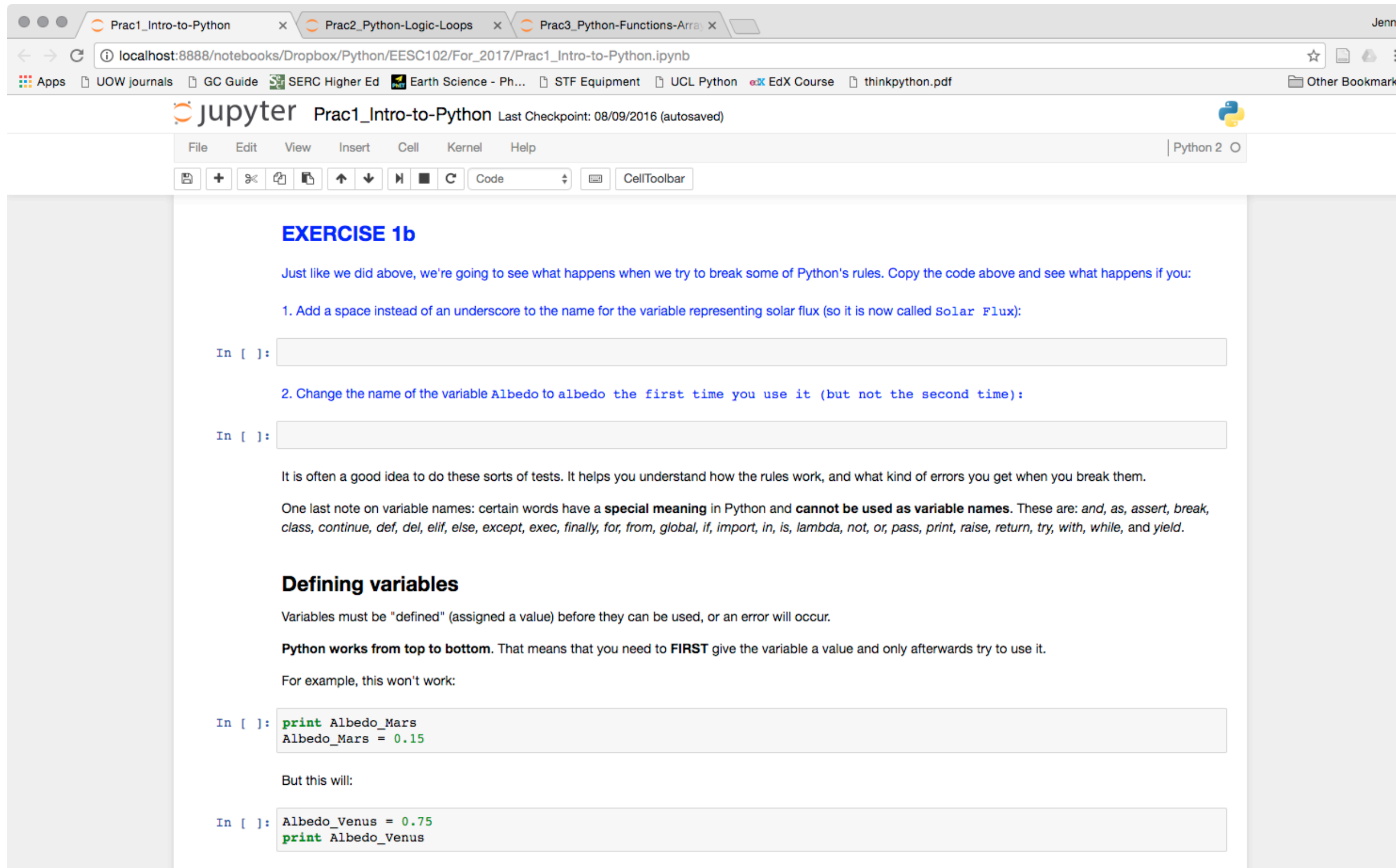
## Examples:



Which countries most affect global CO<sub>2</sub> concentration?

# Implementation

## Exercises in *Jupyter Notebook*:



The screenshot shows a Jupyter Notebook browser interface. The browser tabs include 'Prac1\_Intro-to-Python', 'Prac2\_Python-Logic-Loops', and 'Prac3\_Python-Functions-Arra...'. The address bar shows 'localhost:8888/notebooks/Dropbox/Python/EESC102/For\_2017/Prac1\_Intro-to-Python.ipynb'. The notebook title is 'Prac1\_Intro-to-Python' with a last checkpoint of '08/09/2016 (autosaved)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help) and a toolbar with icons for file operations and code execution. The main content area is titled 'EXERCISE 1b' and contains the following text:

Just like we did above, we're going to see what happens when we try to break some of Python's rules. Copy the code above and see what happens if you:

1. Add a space instead of an underscore to the name for the variable representing solar flux (so it is now called `Solar Flux`):

In [ ]:

2. Change the name of the variable `Albedo` to `albedo` the first time you use it (but not the second time):

In [ ]:

It is often a good idea to do these sorts of tests. It helps you understand how the rules work, and what kind of errors you get when you break them.

One last note on variable names: certain words have a **special meaning** in Python and **cannot be used as variable names**. These are: `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `exec`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `not`, `or`, `pass`, `print`, `raise`, `return`, `try`, `with`, `while`, and `yield`.

### Defining variables

Variables must be "defined" (assigned a value) before they can be used, or an error will occur.

**Python works from top to bottom.** That means that you need to **FIRST** give the variable a value and only afterwards try to use it.

For example, this won't work:

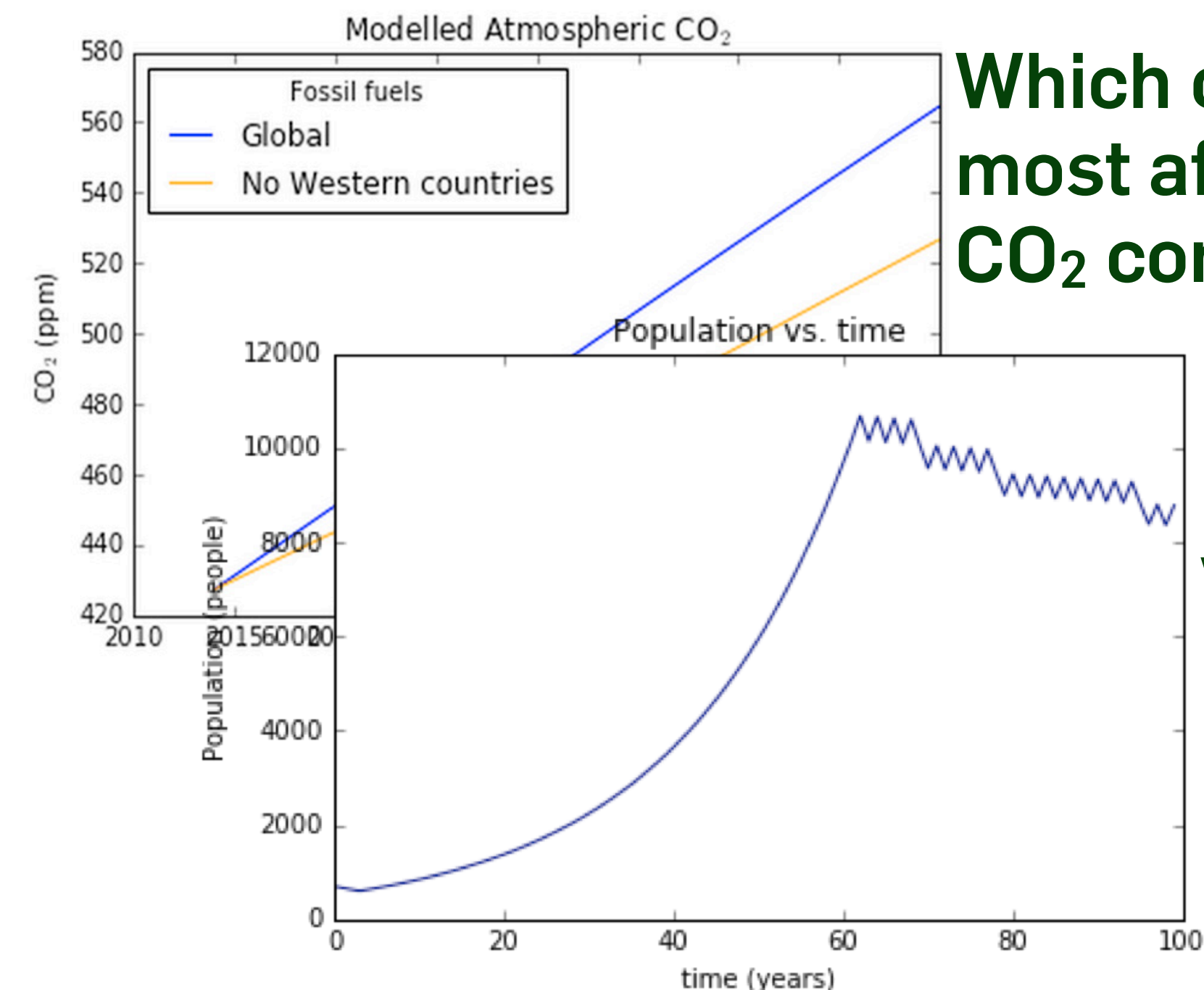
```
In [ ]: print Albedo_Mars
Albedo_Mars = 0.15
```

But this will:

```
In [ ]: Albedo_Venus = 0.75
print Albedo_Venus
```

- Weeks 1-4: computing fundamentals
- 5-8: simple box models
- 9-12: group projects

## Examples:



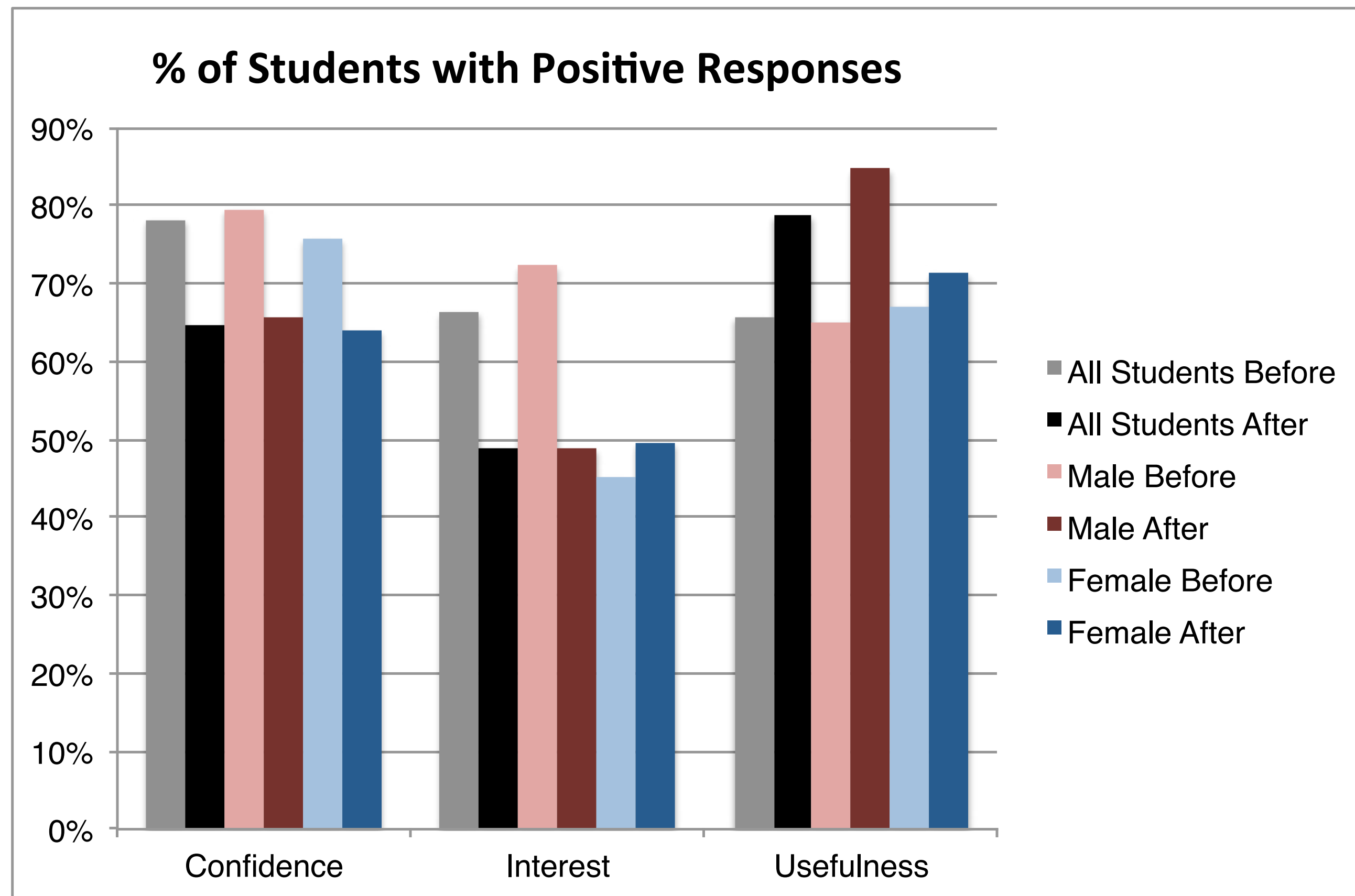
Which countries most affect global CO<sub>2</sub> concentration?

How might "resource wars" affect population?



# Impacts

Pre- and post-surveys of **CONFIDENCE, INTEREST, & USEFULNESS** showed mixed results:



**Confidence** ↓ for all students

**Interest** ↓ for men but ↑ for women

**Usefulness** ↑ for all students

***Positive comments increased with time***



# Lessons Learned

1. **Students DO develop proficiency but find process uncomfortable**
2. **Need incentives to come prepared (pre-lab readings & quizzes)**
3. **Need “stop-and-think” goalposts (modifications to existing exercises)**
4. **Need early & frequent positive feedback (alums as student mentors)**
5. **Need ongoing practice with new skills (continue in 2nd year curriculum)**

**Thanks to** C. Brewer, T. Codilean, L. Chisholm, Z. Jacobs, B. Bukosa, J. Greenslade, S. Inakollu, & EESC102 students!