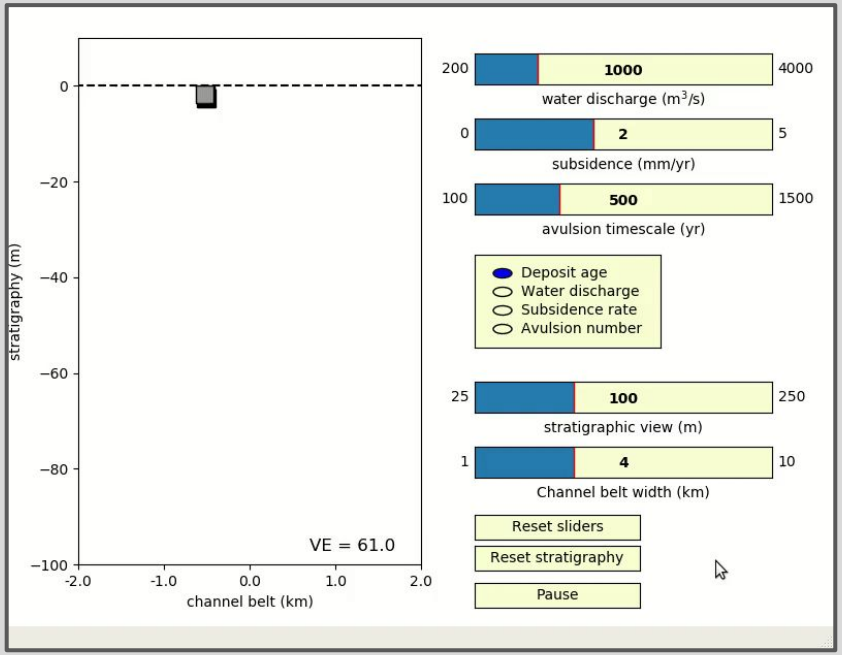
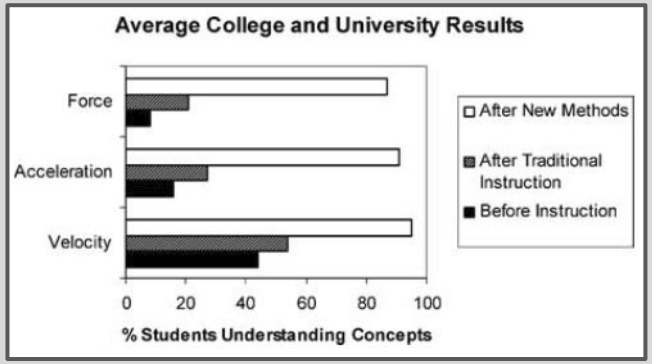
Developing and teaching interactive sedimentology and stratigraphy computer activities



Andrew J. Moodie
amoodie@rice.edu
May 21, 2019
CSDMS, Boulder, CO

“Active learning” improves student knowledge retention

“...there is broad but uneven support for the core elements of active, collaborative, cooperative, and problem-based learning.” [Prince, 2004]



traditionally “laboratory” activities

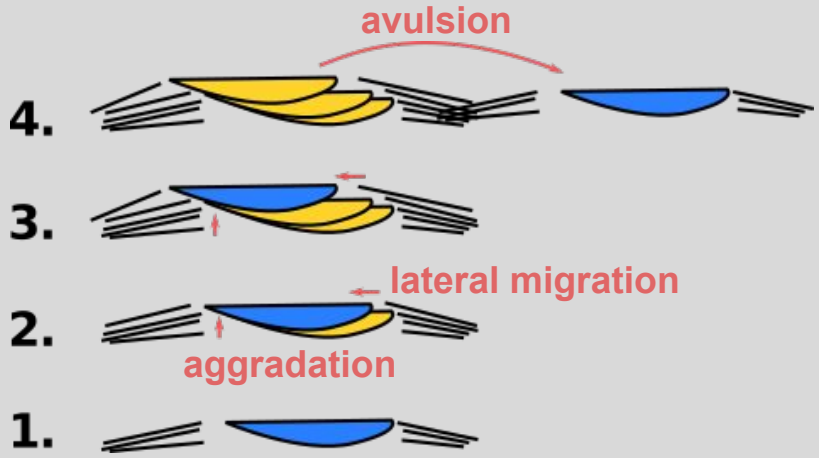
Active learning is any instructional method that **engages** students in the learning process, requiring students to do **meaningful activities** and think about what they are doing.

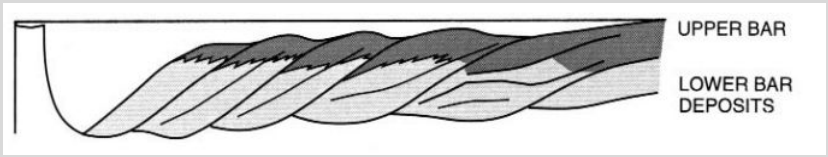
credit @RobertCMahon

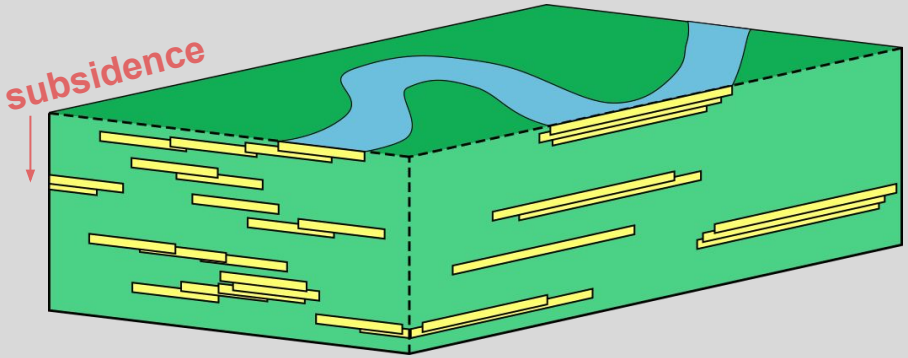
Bonwell, 1991; Laws et al., 1999; Prince, 2004

Active learning with rivers2stratigraphy

- What controls stacking patterns in alluvial stratigraphy?
  - channel geometry
  - lateral migration
  - avulsion timing
  - subsidence
- How do we interpret these controls in the rock record?

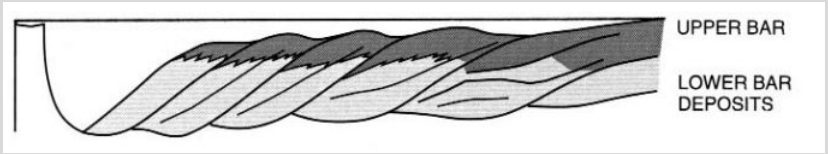


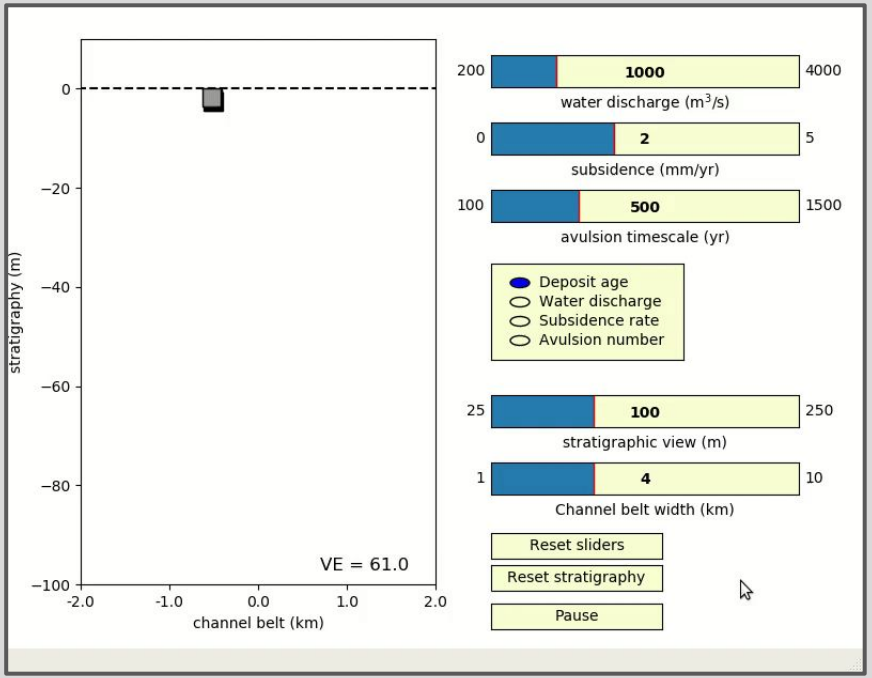




Active learning with rivers2stratigraphy

- Interact with a few “elements” which control the conversion of active fluvial body into basin-scale stratigraphy
- Change display to show different attributes of sand bodies
- Pause and examine the patterns you have created



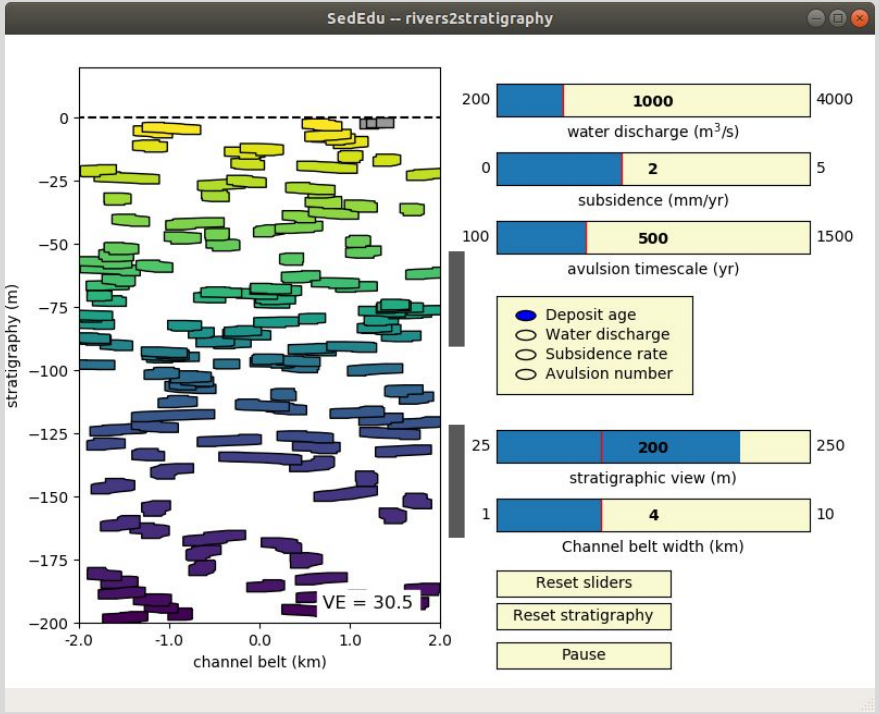


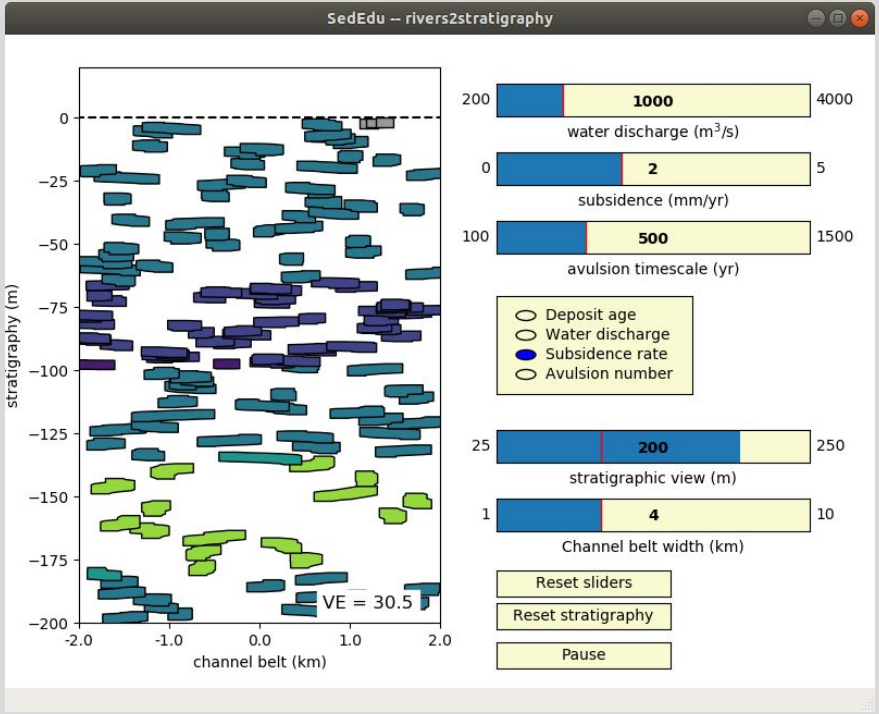
Bridge and Tye, 2000

Interactive computer-based activities in multiple settings

Lecture:

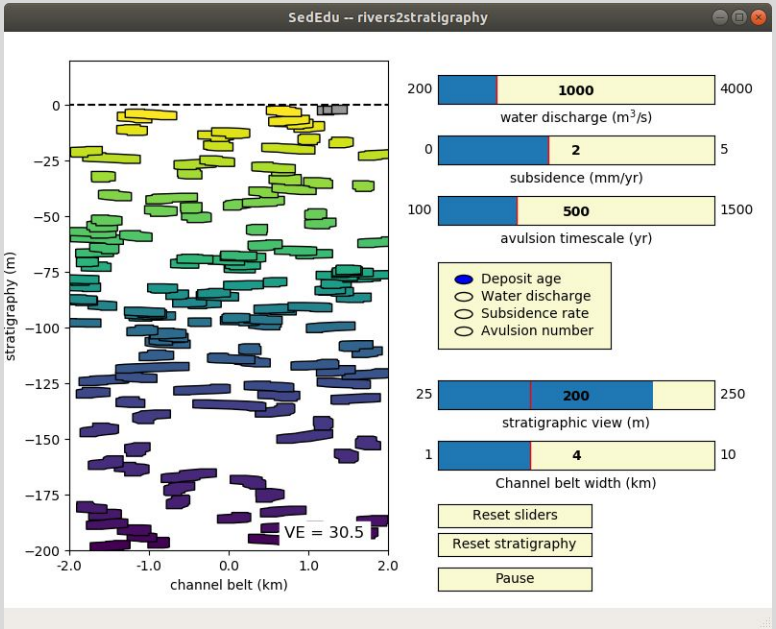
Research question: What were the controls on the observed alluvial architecture?

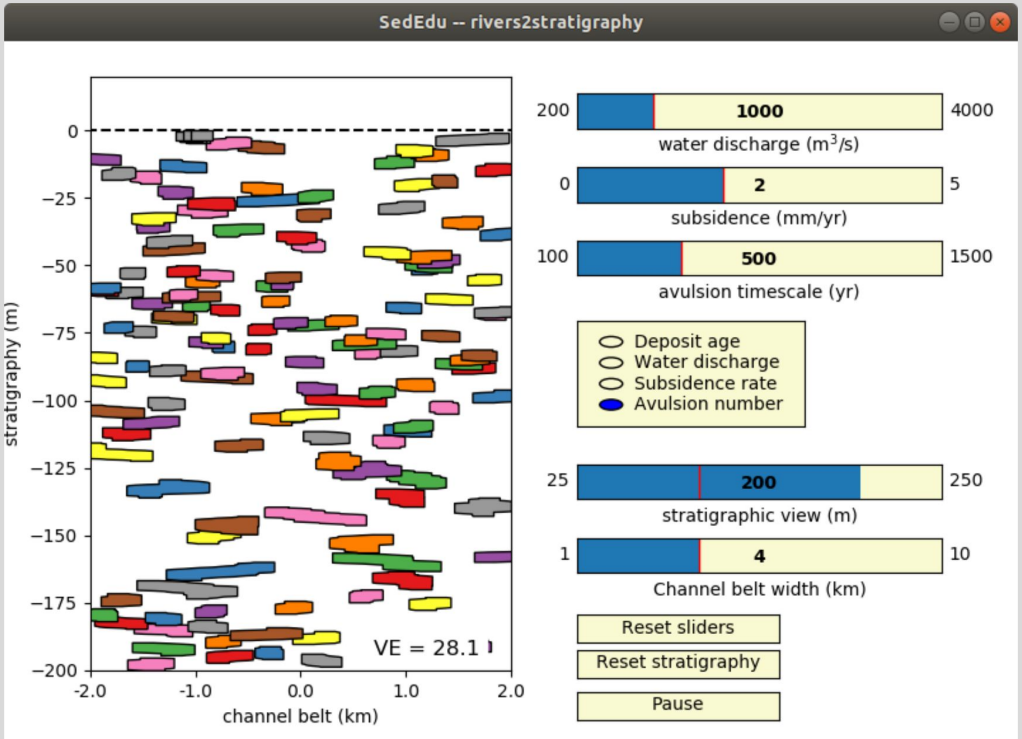




Interactive computer-based activities in multiple settings

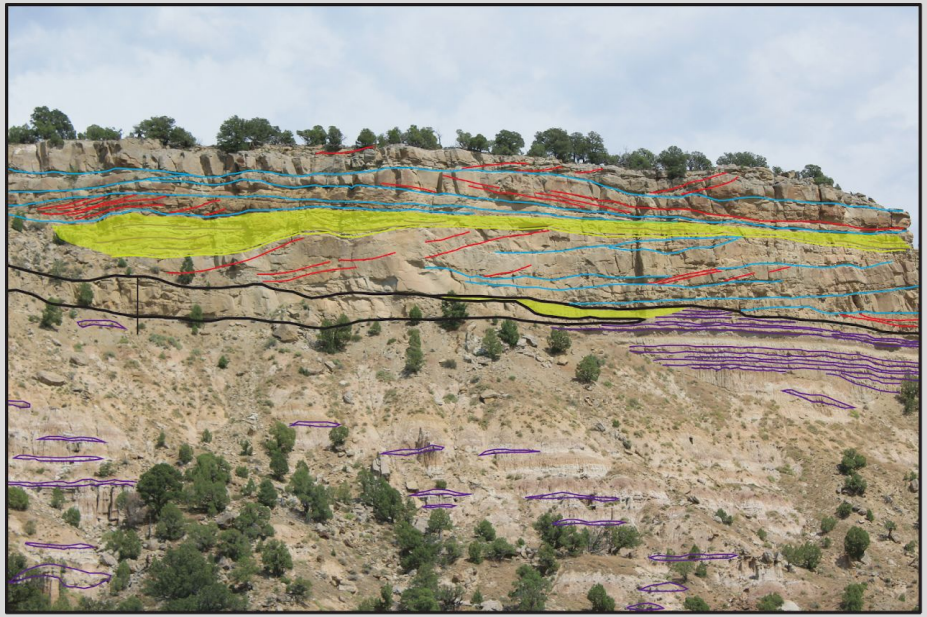
Lecture:

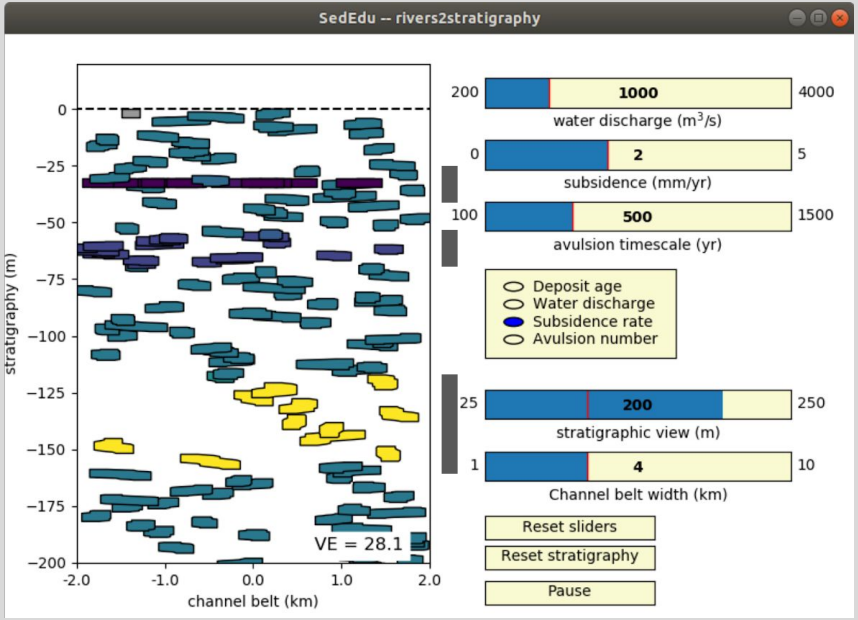




Interactive computer-based activities in multiple settings

Laboratory:





- interpret the outcrop
- what led to the changes in alluvial architecture?

- recreate the outcrop stacking pattern
- what change in variables is needed? Is this realistic?

courtesy Brady Foreman

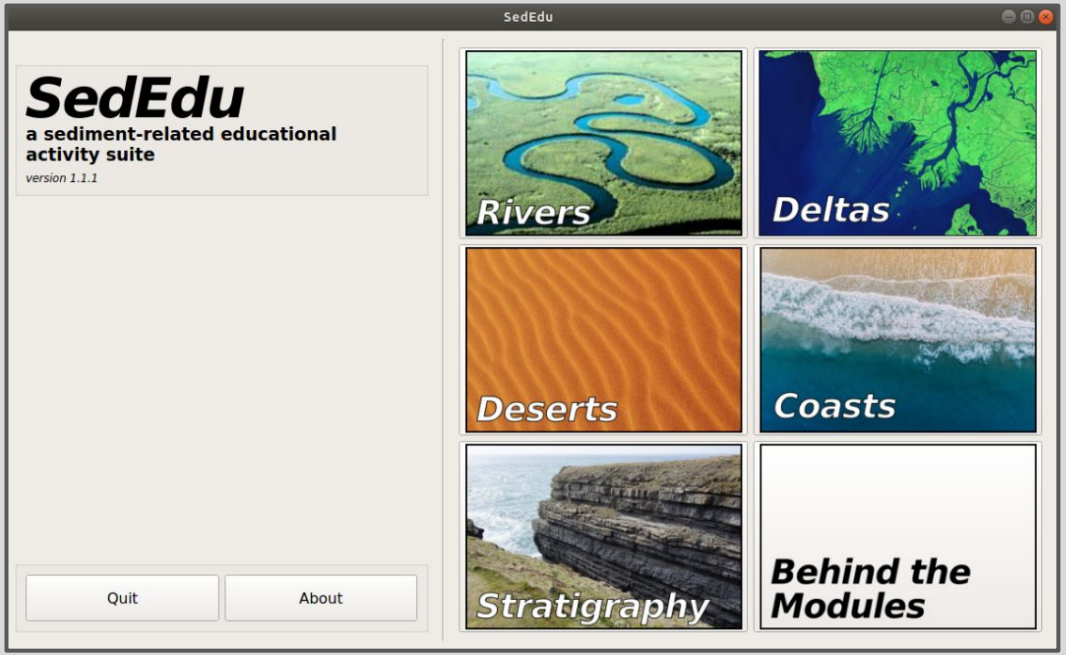
A consistent framework: *SedEdu*

SedEdu is a suite of educational activities related to geomorphology and sedimentology.

The suite is targeted at educators who want to bring engaging, interactive, and scientifically relevant activities into their classroom.

SedEdu is built entirely in Python and is free and open source software.

Modules included in ***SedEdu*** are built by researchers and are designed to showcase their research topics in a digestible manner.



Vision: students download once at the start of a course, and lecturers can use as they see fit

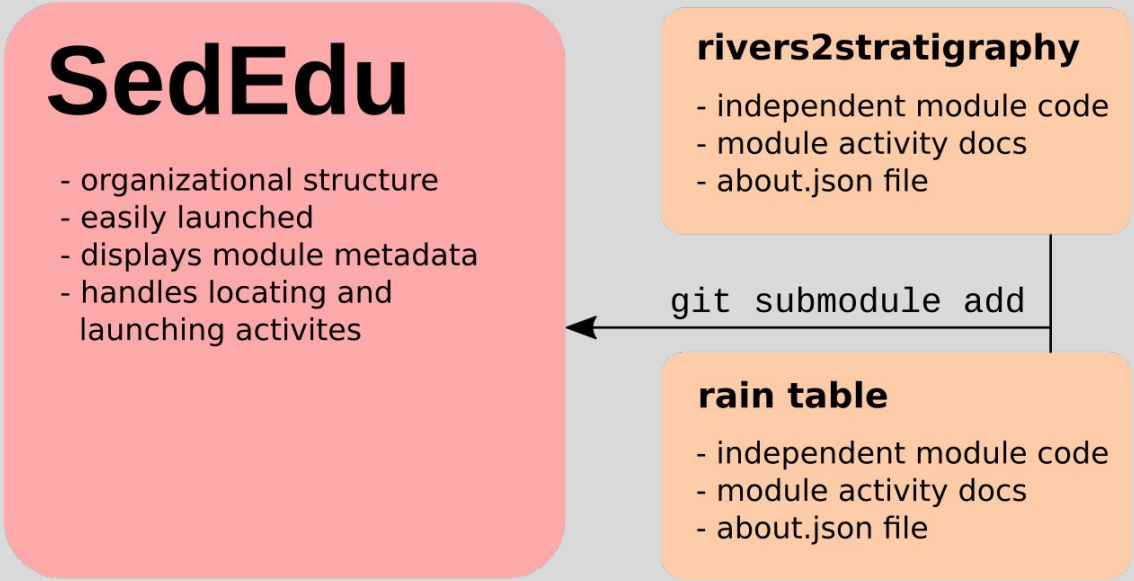
```
conda install sededu -c sededu
```

The structure of *SedEdu*

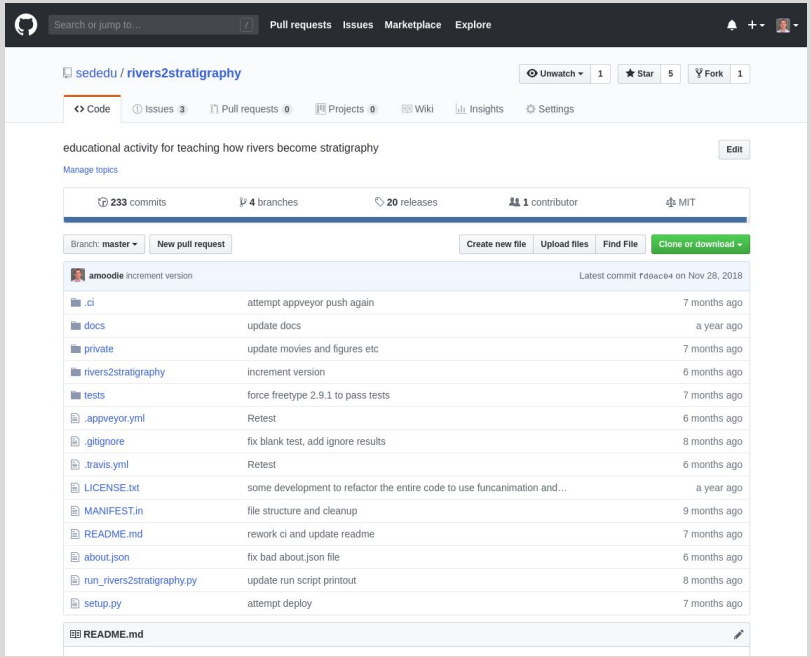
Each module is independently managed, in its own GitHub repository.

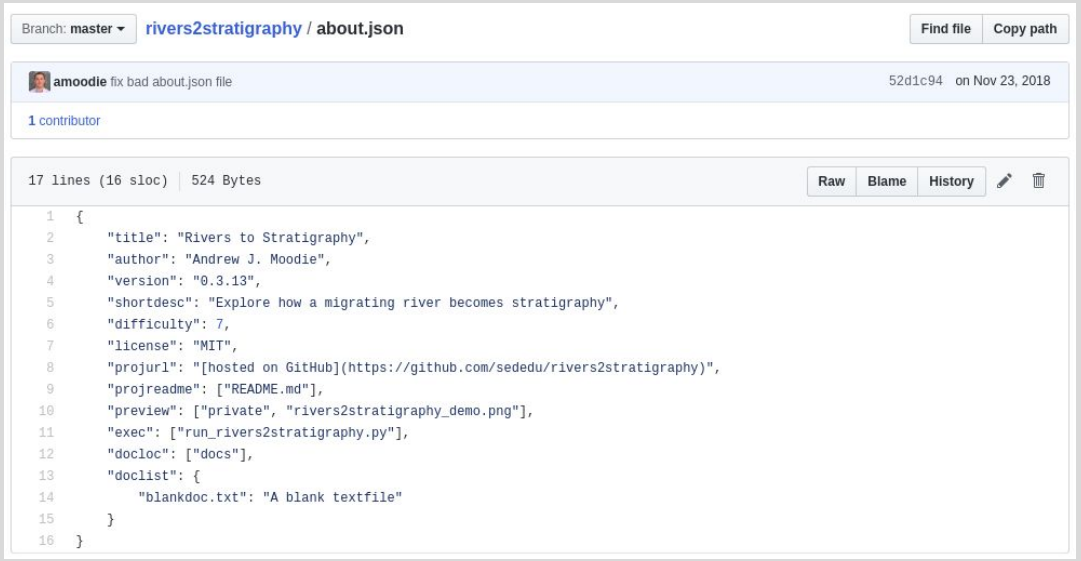
A single 1) instructions file, and 2) include command is needed to bring a module into ***SedEdu***.

Anyone can develop and integrate their work!



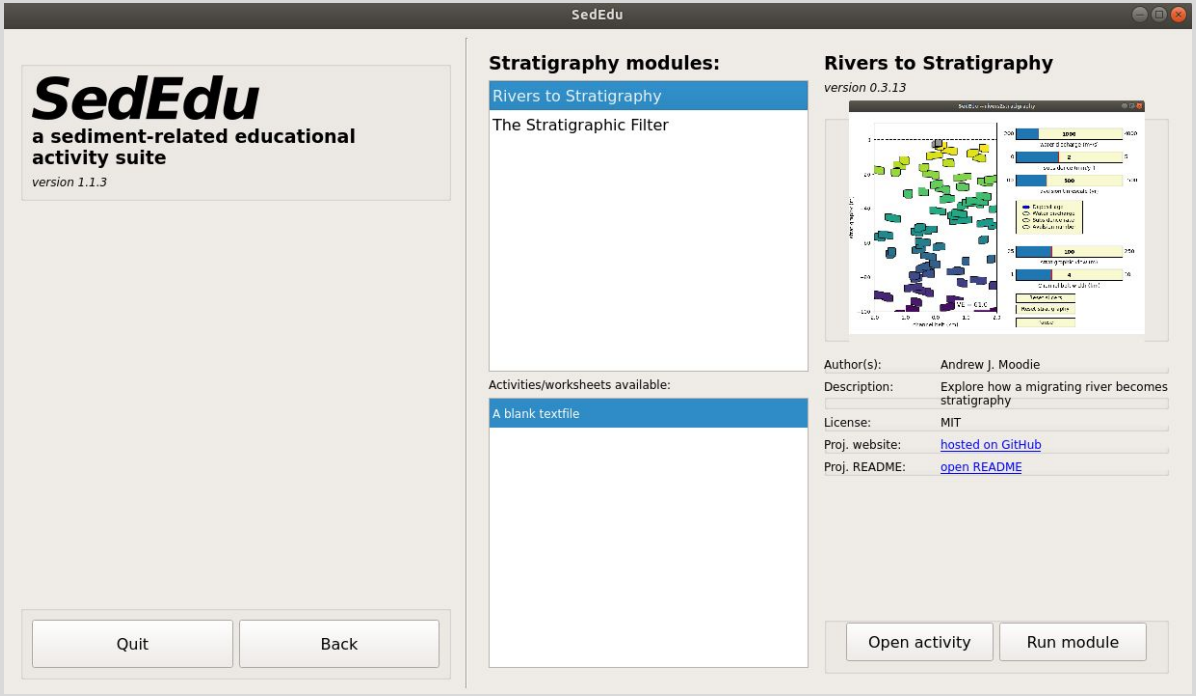
It is easy to incorporate your own module into SedEdu!






```
git submodule add -b master https://github.com/sededu/rivers2stratigraphy \
    sededu/modules/stratigraphy/rivers2stratigraphy
```

It is easy to incorporate your own module into SedEdu!



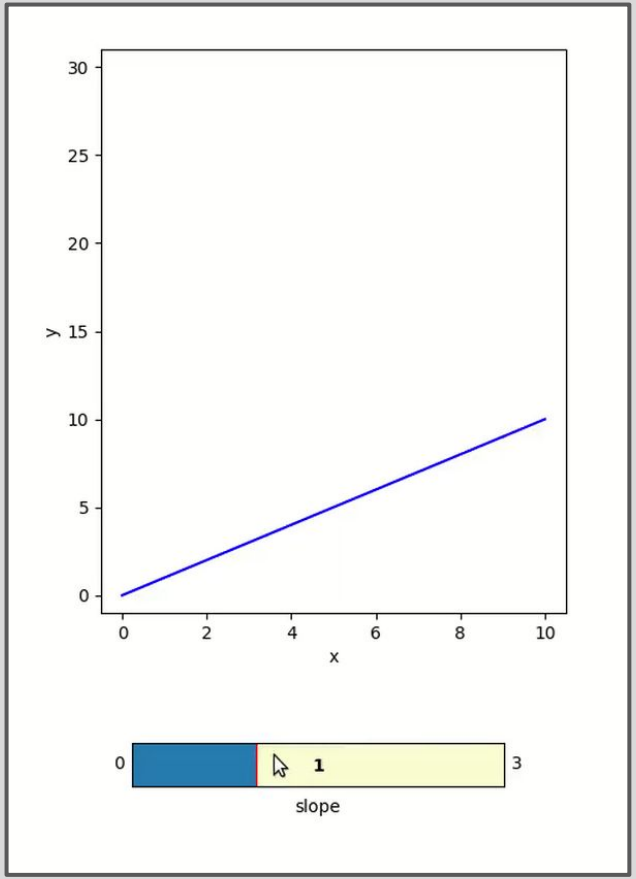

```
git submodule add -b master https://github.com/sededu/rivers2stratigraphy \
      sededu/modules/stratigraphy/rivers2stratigraphy
```

The basic structure of a module

At minimum:

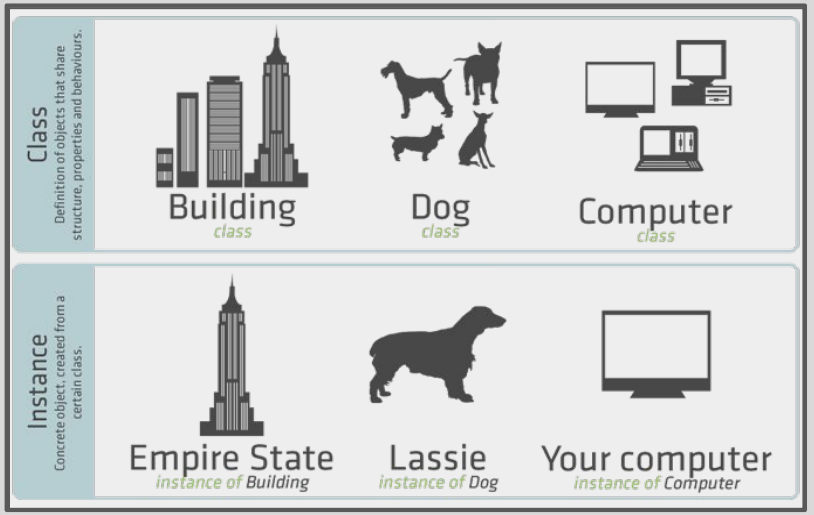
- a plot
- a widget
  - sliders
  - buttons
  - checkboxes
  - lassos

- We use “objects” to build up the module, and interact with the objects.



Object-oriented programming

- Object oriented programming (OOP) is based on the concept of “objects”, which may contain:
  - data, in the form of fields, often known as **attributes**
  - and code, in the form of procedures, often known as **methods**
- An object’s procedures can access and modify the data fields of the object (objects have a notion of “**self**”)
- Objects are **instances** of a **class**



Object-oriented programming

- Object oriented programming (OOP) is based on the concept of “objects”, which may contain:
  - data, in the form of fields, often known as **attributes**
  - and code, in the form of procedures, often known as **methods**

```
1  class Dog:
2      """
3      Dog(name, age) returns an instance of the Dog class.
4      """
5
6      species = 'canis' # class attribute
7      vertebrate = True # class attribute
8
9      def __init__(self, name, age):
10         # initializer method
11         self.name = name
12         self.age = age
13
14     def bark(self):
15         """
16         writes out woof to console
17         """
18         print("woof")
19
```

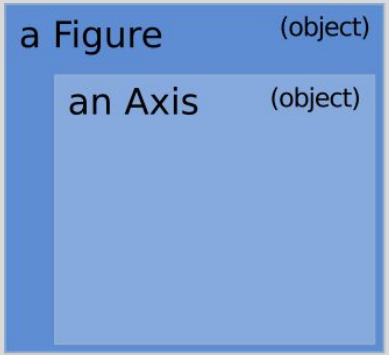
```
26 maggie = Dog(name="Maggie", age=6)
27 olive = Dog("Olive", 7)
```

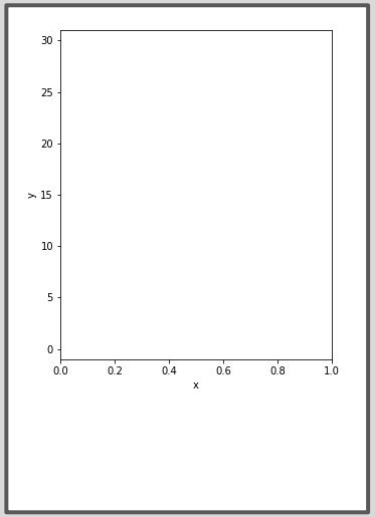
```
In [2]: print(olive)
<__main__.Dog object at 0x7fa354539c18>

In [3]: print(olive.name)
Olive

In [4]: print(olive.bark)
<bound method Dog.bark of <__main__.Dog object at 0x7fa354539c18>>

In [5]: olive.bark()
woof
```

The basic structure of a module

the foundation of the module is an **instance** of the Figure **class**

an Axis **instance** is needed to display plot elements.



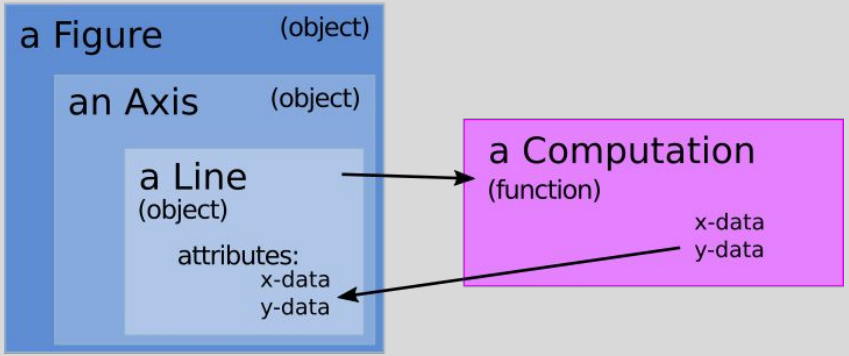


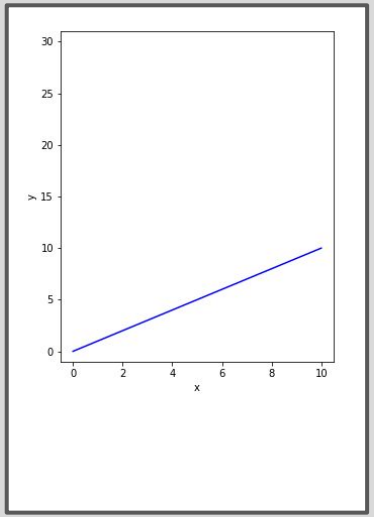
The basic structure of a module

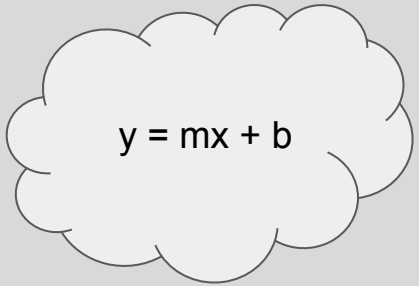
the foundation of the module is an **instance** of the Figure **class**

an Axis **instance** is needed to display plot elements.

a **function** does some computation and makes x- and y-data, which is used to make a Line **instance**.







The basic structure of a module

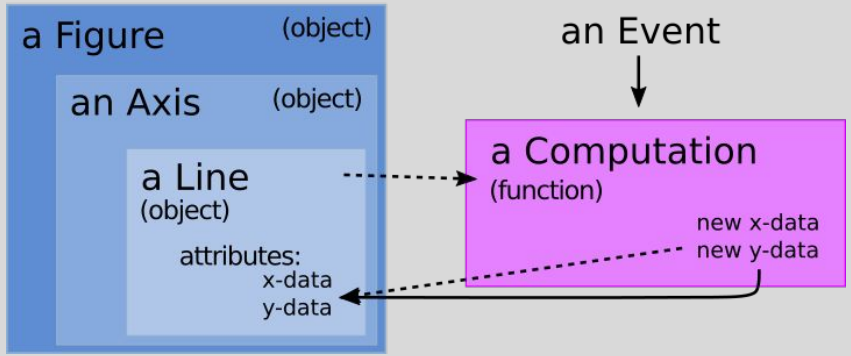
the foundation of the module is an **instance** of the Figure **class**

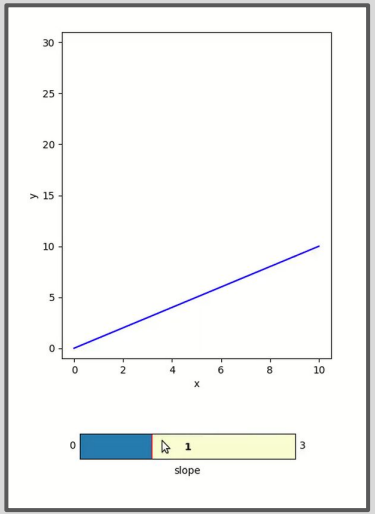
an Axis **instance** is needed to display plot elements.

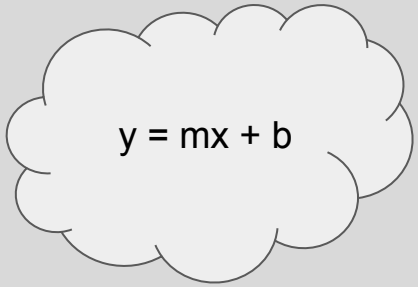
a **function** does some computation and makes x- and y-data, which is used to make a Line **instance**.

every **event**, we redo the computation and **update** the **instance** of Line.

update ≠ new instance

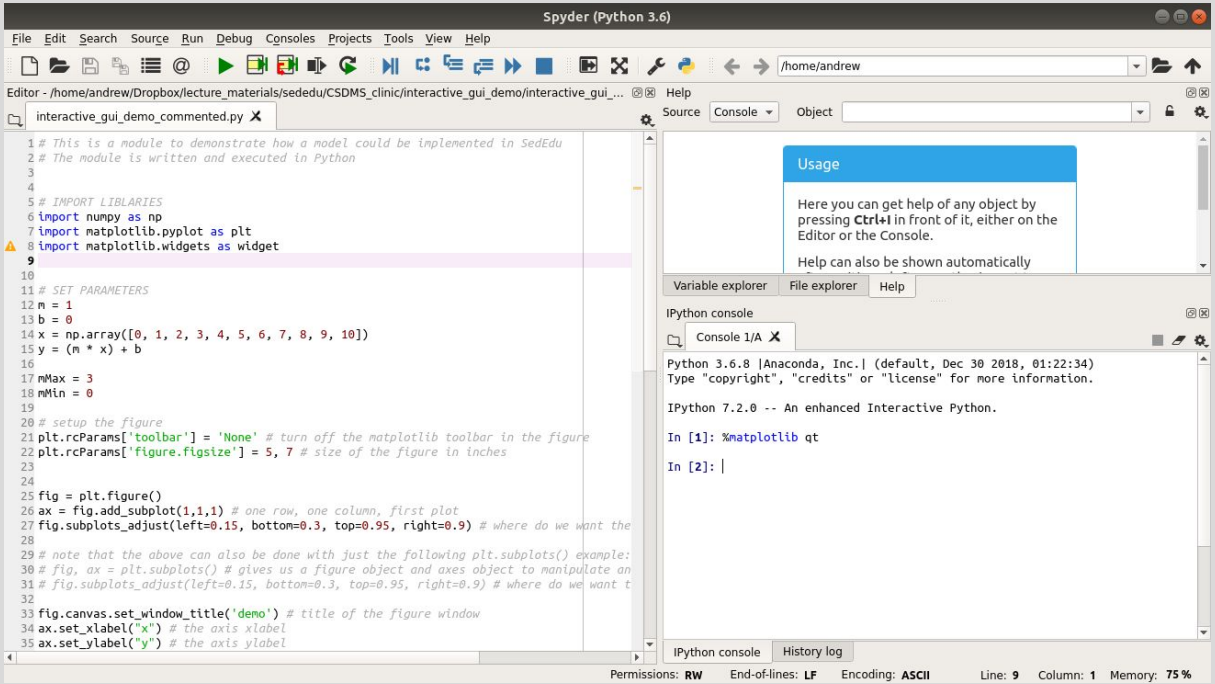






1. Download the latest docs version: `github.com/sededu/CSDMS_clinic`

2. Open the `interactive_gui_demo_commented.py` file in Spyder



*** don't forget to run `%matplotlib qt` in the console ***

Setting up parameters for the “model”

In Python, we import libraries (packages) first

`matplotlib` provides most of the objects we will instantiate to make modules

Python imports keep a clean namespace (e.g., `np.array()`)

Next, we set the parameters we need to make a line (slope, intercept, x-values, and y-values)

`mMax` and `mMin` are the limits to the slider, we’ll come back to this later...

```
1  # This is a module to demonstrate how a model could be implemented in SedEdu
2  # The module is written and executed in Python
3
4
5  # IMPORT LIBLARIES
6  import numpy as np
7  import matplotlib.pyplot as plt
8  import matplotlib.widgets as widget
9
10
11 # SET PARAMETERS
12 m = 1
13 b = 0
14 x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
15 y = (m * x) + b
16
17 mMax = 3
18 mMin = 0
19
```

Setting up the figure and axis

`plt` is the `matplotlib` library we imported

Use the `figure()` **method** to create an **instance** of a `Figure` (`fig`)

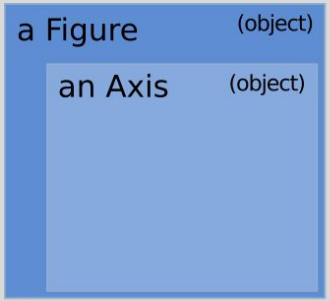
Attach an `Axes` **instance** to the `fig` object

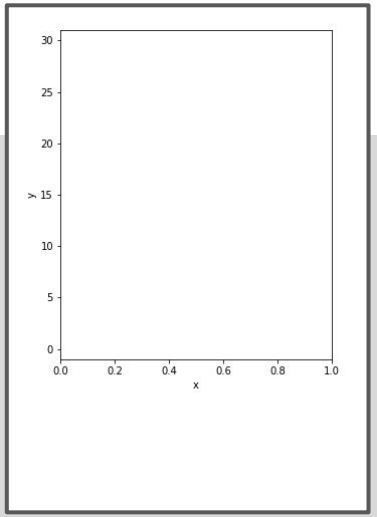
Adjust **attributes** of that `ax` object, using the provided **methods**

[`matplotlib` Axes documentation]

Now, run the code (green play button at top), and see your plot!

```
20 # setup the figure
21 plt.rcParams['toolbar'] = 'None' # turn off the matplotlib toolbar in the figur
22 plt.rcParams['figure.figsize'] = 5, 7 # size of the figure in inches
23
24
25 fig = plt.figure()
26 ax = fig.add_subplot(1,1,1) # one row, one column, first plot
27 fig.subplots_adjust(left=0.15, bottom=0.3, top=0.95, right=0.9) # where do we w
28
29 # note that the above can also be done with just the following plt.subplots() e
30 # fig, ax = plt.subplots() # gives us a figure object and axes object to manipu
31 # fig.subplots_adjust(left=0.15, bottom=0.3, top=0.95, right=0.9) # where do we
32
33 fig.canvas.set_window_title('demo') # title of the figure window
34 ax.set_xlabel("x") # the axis xlabel
35 ax.set_ylabel("y") # the axis ylabel
36 plt.ylim(-1, 31) # the axis y limits
37
```





Adding plot elements

Now we add a `Line2D` **instance** (`theline`) to the **`ax` instance** with the `plot` **method**

There are various “keyword arguments” you can pass during the instantiation of `theline`

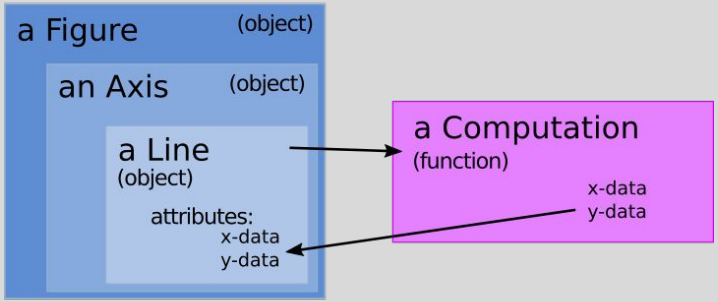
The `Slider` needs an `Axes` object to define its position (the vector: left, bottom, width, height)

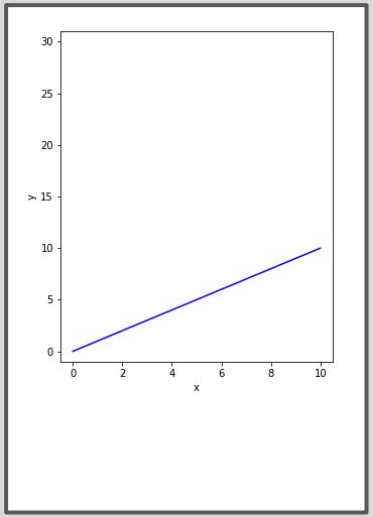
The arguments passed to `widget.Slider()` are:
the new `Axes` **instance**, the label, the minimum and maximum values, the initial value, the step interval, and the format specifier

```
39 # add plot elements
40 theline, = ax.plot(x, y, lw=1.5, color='blue')
41
42
43 # add slider
44 widget_color = 'lightgoldenrodyellow'
45
46 slide_ax = plt.axes([0.2, 0.1, 0.6, 0.05], facecolor=widget_color)
47 slide = widget.Slider(slide_ax, 'slope', mMin, mMax,
48                       valinit=m, valstep=0.1,
49                       valfmt='%g', transform=ax.transAxes)
50
```

`Line2D` documentation (`**kwargs`)

`widget.Slider()` documentation





Updating the plot on events

We define a **function** (outside the scope of any class) which:

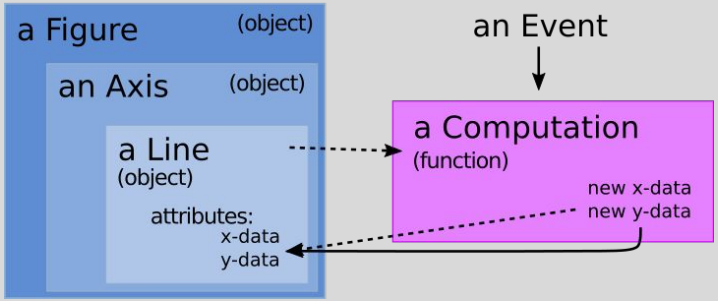
- reads the value **attribute** (`val`) from the slider `slide`
- recomputes the y-data using new slope (`them`)
- uses `theline`'s **method** `set_ydata()` to update the y-data

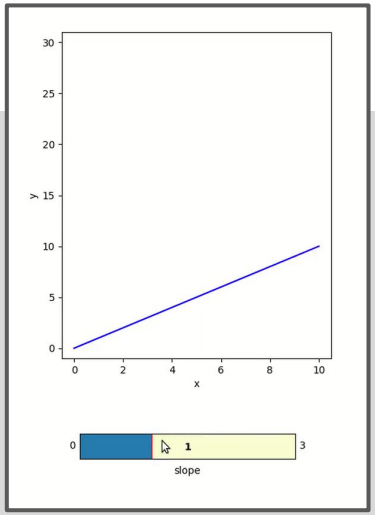
Finally, we connect `slide`'s **method** `on_changed()` to the `update` **function**

Now, run the code (green play button at top), and see your plot!

```
52 # DEFINE FUNCTIONS
53 def update(event):
54
55     # read values from the slider
56     them = slide.val
57
58     # compute new y values
59     they = (them * x) + b
60
61     # update the plot
62     theline.set_ydata(they)
63
64     # redraw the canvas
65     fig.canvas.draw_idle()
66
67
68 # connect widgets
69 slide.on_changed(update)
```

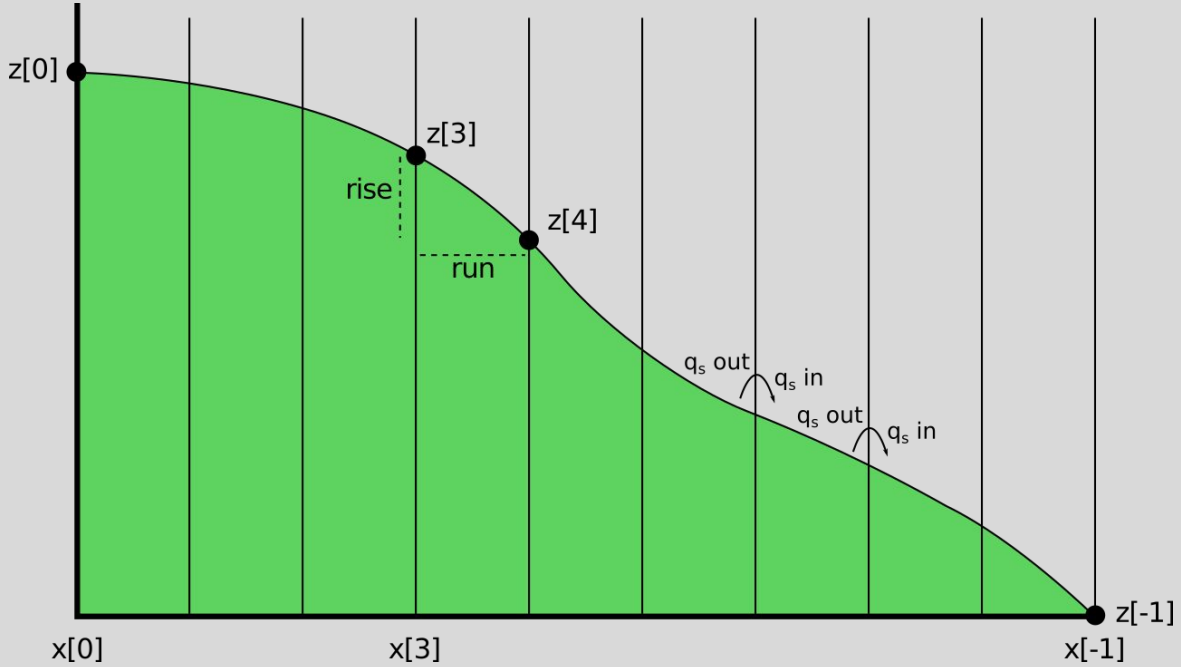
whitespace matters





Now, let's create a geomorphology activity!

We're going to make a simulation of a hillslope, which evolves by a diffusion-like process called "creep".



S = rise / run

$q_s = S \times D$

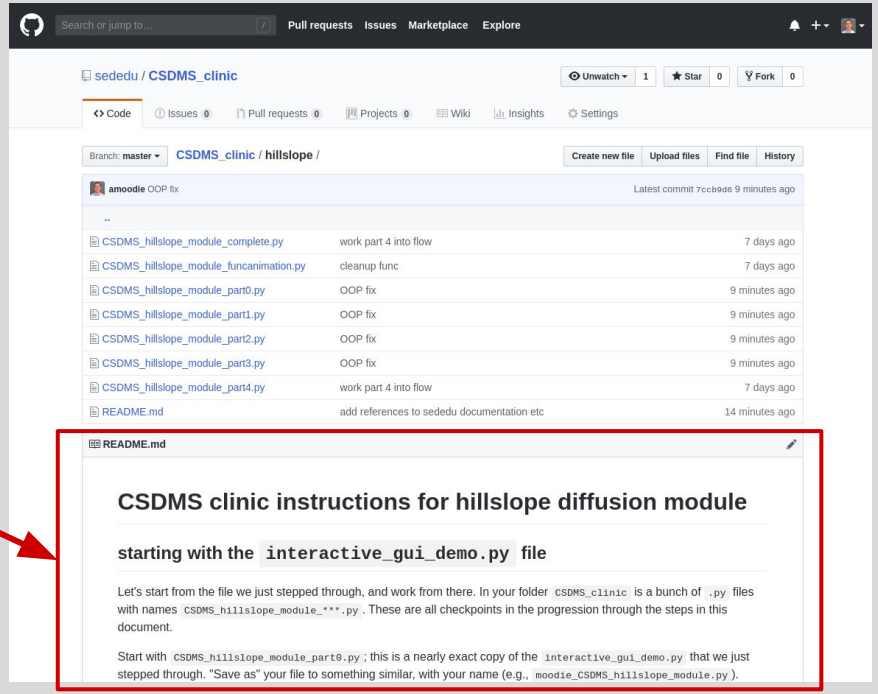
$dz / dt = -dq_s / dx$

Now, let's create a geomorphology activity!

Open the file:
`hillslope/CSDMS_hillslope_module_part0.py`

This is just a copy of the sloping line activity; we will build on this through the remainder of the clinic.

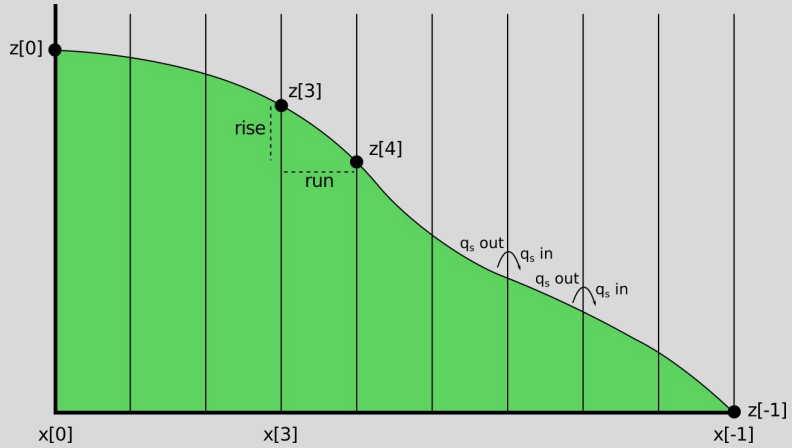
Instructions for working through the activity are on the README page for the `hillslope` folder on GitHub.



Some helpful reminders

Python indexing starts at `0`, and uses half-open “slicing”:

```
Python indexes and slices for a six-element list.
Indexes enumerate the elements, slices enumerate the spaces between the elements.

Index from rear:    -6  -5  -4  -3  -2  -1      a=[0,1,2,3,4,5]    a[1:]==[1,2,3,4,5]
Index from front:    0   1   2   3   4   5      len(a)==6          a[:5]==[0,1,2,3,4]
                  +---+---+---+---+---+---+     a[0]==0            a[:-2]==[0,1,2,3]
                  | a | b | c | d | e | f |     a[5]==5            a[1:2]==[1]
                  +---+---+---+---+---+---+     a[-1]==5           a[1:-1]==[1,2,3,4]
Slice from front: :   1   2   3   4   5   :     a[-2]==4
Slice from rear:  :  -5  -4  -3  -2  -1   :
                                                b=a[:]
                                                b==[0,1,2,3,4,5] (shallow copy of a)
```



Python whitespace matters:

```
counter = 1
while (counter <= 5):
    if counter < 2:
        print("Less than 2")
    elif counter > 4:
        print("Greater than 4")
    counter += 1
```

The arguments passed to `widget.Slider()` are:

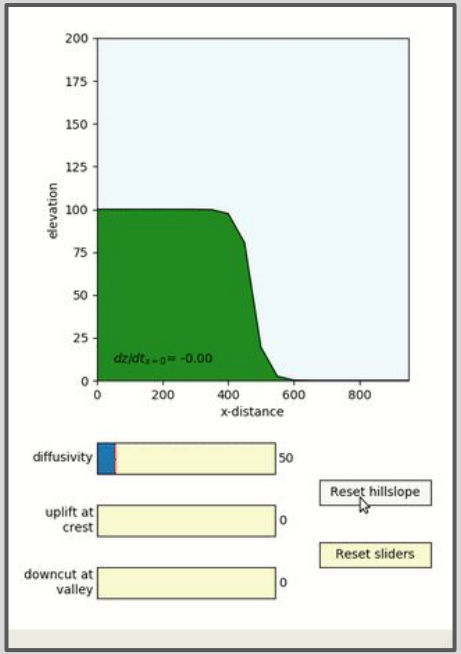
the new `Axes` **instance**, the label, the minimum and maximum values, the initial value, the step interval, and the format specifier

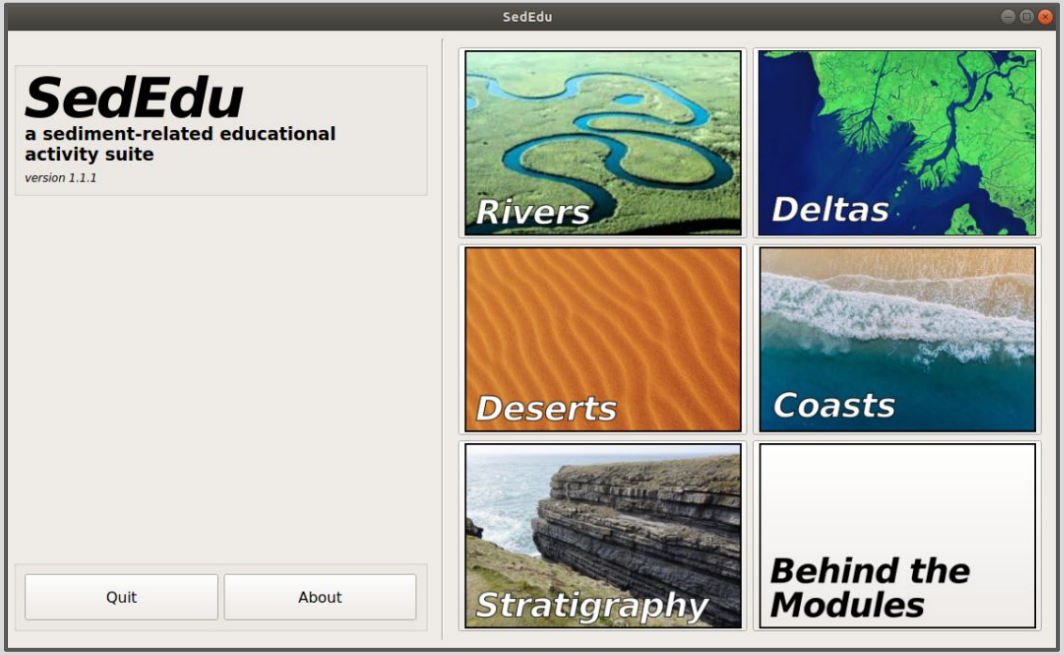
```
46 slide_ax = plt.axes([0.2, 0.1, 0.6, 0.05], facecolor=widget_color)
47 slide = widget.Slider(slide_ax, 'slope', mMin, mMax,
48                       valinit=m, valstep=0.1,
49                       valfmt='%g', transform=ax.transAxes)
50
```

Final remarks

These modules are an opportunity for **broader impacts** and bringing geoscience to many classrooms.

```
conda install sededu -c sededu
```





Final remarks

Thank you, sincerely, for participating in this clinic

I hope that you have learned:

- how to bring active learning into your classroom
- the components that make up an interactive module
- how to develop a module

Please email me if you have any further questions or feedback on this clinic (`amoodie@rice.edu`)

There are more resources available online at:

https://github.com/sededu/sededu

Let's connect!

I will be here through Thursday

or

Andrew J. Moodie
amoodie@rice.edu
andrewjmoodie.com
@MoodieStrat

References

Bonwell, C.C., and J. A. Eison, “Active Learning: Creating Excitement in the Classroom,” ASHEERIC Higher Education Report No.1, George Washington University, Washington, DC , 1991.

Laws, P., D. Sokoloff, and R. Thornton, “Promoting Active Learning Using the Results of Physics Education Research,” UniServeScience News, Vol. 13, July 1999.

Prince, M. (2004). Does Active Learning Work? A Review of the Research. Journal of Engineering Education, 93(3), 223-231. doi: 10.1002/j.2168-9830.2004.tb00809.x.