BIOS-584 Python Programming (Non-Bios Student)

Week 07

Instructor: Tianwen Ma, Ph.D.

Department of Biostatistics and Bioinformatics,
Rollins School of Public Health,
Emory University

Announcement

- HW7 and final project deliverable 1 both due on **Oct 24**.
- Since HW7, the prompt will NOT be specific to each step.
- You design your own function to achieve the task.
  - The solution is not unique.
- Feel free to ask questions or stop by OH. I am sure you will!

Announcement

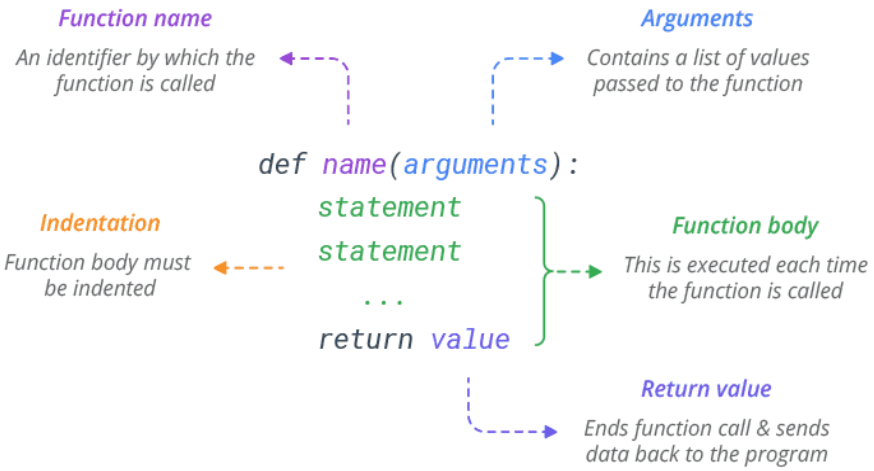
- You will start to write codes in .py file (after HW7)
  - Formatting will matter more.
  - You will try debugging using PyCharm.
- Do not feel frustrated for programming
  - Ask for help from Madeline and me
  - Believe in yourself

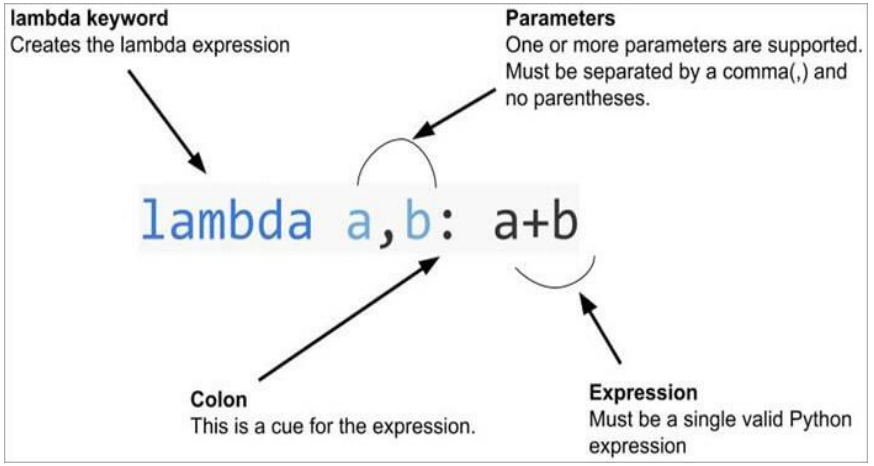
Announcement

- I will host Georgia Statistics Day and be gone on Oct 31
- No class that day
- Quiz 3
- Finish 10-11 in class
  - Early or late quiz is NOT allowed
- Details will be available soon

Recall from previous class

- How to write customized functions using def and return
- Parameters, arguments, and return values
- Lambda function





Lecture Overview

- Array continued
  - Broadcasting
  - Subsetting, Slicing, and Indexing
  - Array manipulation
  - Additional visualization technique
- week-07-numpy-array-continued-with-solution.ipynb

Rationale behind Transition

- Before we move on to variable scoping in Python functions, it is necessary to introduce advanced features on NumPy arrays.
- This lecture requires some spatial logic/reasoning and may be challenging.
- Feel free to stop and ask questions.

Array Mathematics

- In week 3, we covered basic mathematical operations on array.
  - + or np.add()
  - - or np.subtract()
  - * or np.multiply()
  - / or np.divide()
  - % or np.remainder()
  - np.exp(), np.log(), np.sqrt() …
- Previously, we argued that the dimension of both arrays must be the same.
  - However, that is a sufficient condition.

Broadcasting in NumPy array

- Recall we talked about operations between a single array and a scalar
- Suppose a is a 1-dim vector with 3 elements
- We can add or multiply a scalar to an array
  - $a + 3 = \begin{pmatrix} a_1 + 3 \\ a_2 + 3 \\ a_3 + 3 \end{pmatrix}$
- 3 here is essentially broadcast from a scalar to a 1-dim vector of the same length as a to make two arrays "compatible" for element-wise addition.

Broadcasting in NumPy array

- It is a mechanism to allow NumPy to work with arrays of different shapes when performing arithmetic operations.
- If you have a “large” array and a “small” array, small array will be used multiple times to perform an operation on the large array.
- You need to follow certain rules.

Broadcasting in NumPy array

- The dimensions of arrays are “compatible”

Broadcasting in NumPy array

- Two dimensions are compatible when one of them is 1.
- If the dimensions are NOT compatible, you will receive a ValueError.

Broadcasting in NumPy array

- The arrays can only be broadcast together if they are compatible in all dimensions.

Broadcasting in NumPy array

- Despite the seemingly different dimensions between x and y, the two can be added together.
- That is because they can be compatible in all dimensions.
  - Array x has dimensions 3 x 4
  - Array y has dimensions 5 x 1 x 4
  - The middle dimension in y (1) and the first dimension in x (3) -> y behaves as if it were copied along that dimension
  - The dimension of the resulting array is the maximum size along each dimension of x and y, i.e., (5, 3, 4)

(1x) 3 x 4
5 x 1 x 4
↓
5 x 3 x 4

Summary Functions

- We also covered summary functions including
  - np.sum(), np.mean(), np.min(), np.max(), np.std()
  - np.corrcoef() (Pearson product-moment correlation)
- Suppose we have an array defined as arr
  - np.sum(arr)
  - arr.sum()

Logical Operations

- We covered and/&, or/| to perform logical operations
- NumPy also have equivalent functions to achieve those goals:
  - np.array_equal()
  - np.logical_and()
  - np.logical_or()
  - np.logical_not()

Subset, Slice, and Index Arrays

- We covered subsetting and indexing lists.
- You use square brackets [] as the index operator
- Generally, you pass integers to [], but you can also put a colon : or a combination of colon with integers to designate elements/rows/columns you want to select
- Remember that Python's index starts from zero!

Subsetting with Examples

- See examples in the Jupyter Notebook

Slicing with Examples

- In slicing, you consider not just particular values of your arrays but consider levels of rows and columns.
  - You basically work with “regions” of data instead of pure “locations”.
- See examples in the Jupyter Notebook
  - Pay attention to the definitions of : and …

Indexing

- Boolean indexing: Instead of selecting elements, rows, or columns based on index number, you select those values from your array that fulfill a certain condition.
- You can utilize logical operators such as | and &

Useful functions

- np.lookfor(): Perform a keyword search on docstrings
- np.info(): Quick explanations and code examples of functions, classes, or modules.
- ?fun_name: A universal help function
- Search the function name + API online

Array Manipulation

- When the dimensions of arrays are NOT compatible, array manipulation is required to perform operations.
  - Transpose
  - Resize and reshape
  - Append
  - Insert and delete array elements
  - Join and split arrays

Transpose

- Transpose refers to permute the dimensions of the original array.
- np.transpose() or .T
- Try transposing a 1-dim array

Transpose

- Transpose refers to permute the dimensions of the original array.
- np.transpose() or .T
- Try transposing a 1-dim array
  - There is no effect when you transpose a 1-dim array.

Resize

- One way to make arrays compatible is to modify the dimension of the array.
- You can use np.resize() function
- Two usages:
  - arr_new = np.resize(arr_old, new_shape) -> additional portion filled with copies of the original array
  - arr_old.resize(new_shape) -> additional portion filled with zeros (no need to assign it to a new name)
- new_shape can be either a list [] or a tuple ()
- See an interesting example

Two usages of np.resize()

- arr_new = np.resize(arr_old, new_shape)
  - Additional portion filled with copies of the original array
- arr_old.resize(new_shape)
  - Additional portion filled with zeros
  - No need to assign it to a new variable name!
- new_shape can be either a list [] or a tuple ()
- See an interesting example on Jupyter Notebook

Reshape

- You give a new shape to an array without changing its data
- The total size of the new array is UNCHANGED.
  - For example, if array x is a 3 x 4 matrix, after you reshape it to 1-dim vector, it has a length of 12.
- x.size tells you the size of the array.
  - Or np.size(x)
- arr_new = np.reshape(arr_old, new_shape)
- arr_new = arr_old.reshape(new_shape)

Append

- Similar to a list, you can append an array to the original one.
- The new array is “glued” to the end of that original array.
- arr_new = np.append(arr_old, small_arr_new)
- Note that the axis option makes a difference!
  - From now on, you start to learn performing a task along a particular axis (or axes).
- See the variation of the example in Jupyter Notebook.

Insert and delete array elements

- While append only add something to the end, insert can add the new array to any other location. Sort of index location
- np.insert(arr, obj, values, axis=None)
  - A copy of the new array with values inserted is returned. Assignment to a new variable is required.
- np.delete(arr, obj, axis=None)
  - A copy of arr with corresponding elements specified by obj removed. Assignment to a new variable is required.

Join arrays

- You can “merge” or join arrays.
- This is super helpful in practice
- A couple of functions listed in the following slide

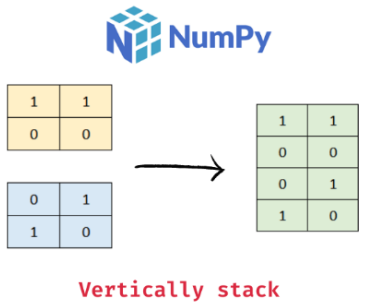
Join a seq of arrays along an existing axis

- np.concatenate([a1, a2, ...], axis=0, ...)
- np.concat([a1, a2, ...], axis=0, ...)
- The input arrays must have the same shape, except in the dimension corresponding to axis.
  - Preferably saved them as a list.

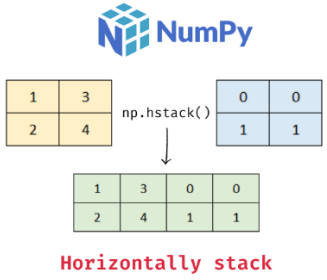
Join a seq of arrays along a new axis

- np.stack([a1, a2, ...], axis=0, ...)
- Each array input must have the same shape
  - Preferably saved them as a list.
- The axis parameter specifies an index of the new axis in the dimensions of the resulting array.
- axis=0 is the first dimension of the new array
- axis=-1 is the last dimension of the new array

Join arrays



- Stack arrays in sequence vertically (row wise)
  - np.vstack([a1, a2, ...])
- Stack arrays in sequences horizontally (row wise)
  - np.hstack([a1, a2, ...])
- The above two functions make most sense for arrays with up to 3 dimensions.
  - No need to specify the axis



Split arrays

- Similarly, you can split an existing arrays to multiple sub-arrays
- Sort of reverse operations w.r.t. stack/concatenate
- Three useful functions in the following slides

Split an array into sub-arrays

- np.split(arr, indices_or_sections, axis=0)

Split an array into sub-arrays vertically (row-wise)

- np.vsplit(arr, indices_or_sections)

Split an array into sub-arrays horizontally (column-wise)

- np.hsplit(arr, indices_or_sections)

Visualize NumPy Arrays

- Np.hist() or np.histogram()
- Scatterplot
- Covered in previous lectures

Visualize NumPy Arrays

- Use np.meshgrid() to visualize an array with 2-d arrays of x and y coordinate values
  - You create a rectangular grid out of x and y values
  - np.meshgrid() takes two 1-dim arrays and produces 2d-array corresponding to all pairs of (x, y) in the two arrays
  - Use these matrices to make plots (usually with a z coordinate value)
- This can be used to visualize a covariance matrix, a contour plot, etc.
- You will plot many covariance matrices in HW7.

Visualize NumPy Arrays

```
# Create an array
points = np.arange(-5, 5, 0.01)

# Make a meshgrid
xs, ys = np.meshgrid(points, points)
z = np.sqrt(xs ** 2 + ys ** 2)

# Display the image on the axes
plt.imshow(z, cmap=plt.cm.gray)

# Draw a color bar
plt.colorbar()

# Show the plot
plt.show()
```