Python (Semester 2 2024)

Course Logistics, Introduction to Python (I)

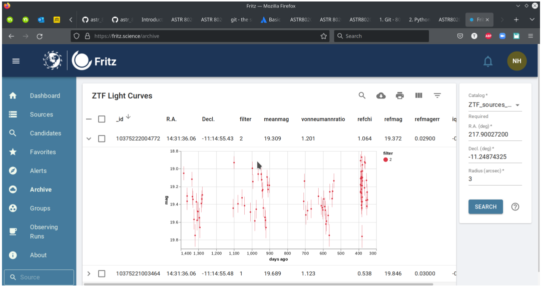
Nina Hernitschek
Centro de Astronomía CITEVA
Universidad de Antofagasta

August 19, 2024

what you will learn in this class

this course will prepare you for “doing science” with current and upcoming large astronomical surveys:

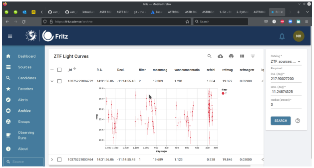
accessing astronomical data



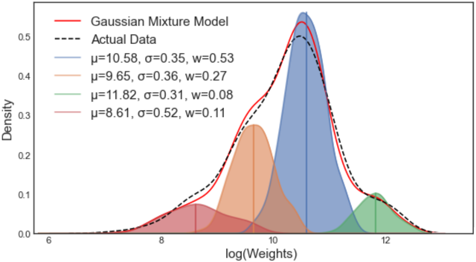
what you will learn in this class

this course will prepare you for “doing science” with current and upcoming large astronomical surveys:

accessing astronomical data



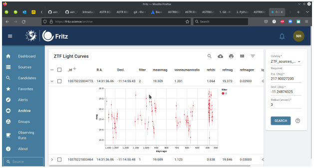
applying statistical methods



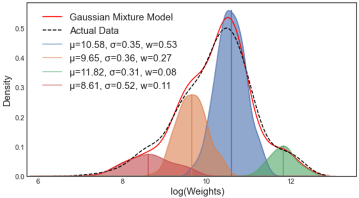
what you will learn in this class

this course will prepare you for “doing science” with current and upcoming large astronomical surveys:

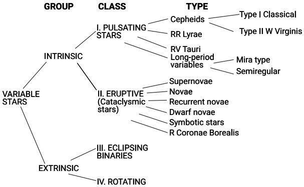
accessing astronomical data



applying statistical methods



carry out classification



Course Logistics

course content:

- lecture: Monday 10-11:30
- tutorial session: Wednesday 10-11

grading:

- participation: 20 % of total grade
- project presentations (project idea + project status + final): 40 % of total grade
- project report: 40 % of total grade

contact and course material:

- e-mail: `nina.hernitschek@uantof.cl`
- github: `https://github.com/ninahernitschek/python_2024_2`

Course Logistics

Your project will be graded based on:

- project presentations (project idea + project status + final): 40 % of total grade
- project report: 40 % of total grade

Course Logistics

Your project will be graded based on:

- project presentations (project idea + project status + final): 40 % of total grade
- project report: 40 % of total grade

What will be your project?

Course Logistics

Your project will be graded based on:

- project presentations (project idea + project status + final): 40 % of total grade
- project report: 40 % of total grade

What will be your project?

Chose your own! Select some astronomical data of your choice and **try to analyze them** with the methods shown in the lectures and Jupyter notebooks.

Your **three presentations** should show the progress.

Course Logistics

August 19 Lecture 1: Introduction to Python (I), **August 21** Tutorial 1

August 26 Lecture 2: Introduction to Python (II), **August 28** Tutorial 2

September 2 Q&A Session, **September 4** *Project Idea Presentation*

September 9 Lecture 3: Data Exploration, **September 11** Tutorial 3

September 23 Lecture 4: Data Visualization, **September 25** Tutorial 4

September 30 Lecture 5: Efficient Programming, **October 2** Tutorial 5

October 7 Lecture 6: Statistical Packages, **October 9** *Project Status Presentation*, Tutorial 6

October 14 Lecture 7: Astronomical Data Sources, **October 16** Tutorial 7

October 28 Lecture 8: Astronomical Packages, **October 30** Tutorial 8

November 4 Tutorial 9 (I), **November 6** Tutorial 9 (II)

November 11 *Final Presentation*

Rules for Coding, Presentations, Report

coding: If you have a question when something doesn't work, summarize what you tried - often this will even lead to the solution.

project presentation:

- LaTeX
- figures: all own figures should be in vectorized pdf format
- data and aim of project: data description (incl. citation)
- own work: properly cite what is not your own work; discuss how the previous work is similar to or different from your own work
- implementation: medium-level implementation description with libraries/ software frameworks (incl. citation), project milestones ⇒ more details than in a research paper
- related work: include both work aimed at similar problems and work that employs similar solutions to yours
- discussion: reflect your approach (strengths, weaknesses, limitations), lessions learned
- bibliography

Textbooks

In addition to using the exensive online documentation, I can recommmend the following textbooks:

Statistics, Data Mining, and Machine Learning in Astronomy: A Practical Python Guide for the Analysis of Survey Data; Ivezić, Ž., Connolly, A. J., VanderPlas, J. T., Gray, A.; Princeton Series in Modern Observational Astronomy.

Practical Statistics for Astronomers; Wall, J. V., and Jenkins, C. R.; Cambridge. Observing Handbooks for Research and Astronomers.

Python machine learning by example: The easiest to get into machine learning; Liu, Y.; Packt Publishing.

Why use Python?

The **Python ecosystem** provides a single environment that is sufficient for the vast majority of astronomical analysis. It does so on several levels:

Why use Python?

The **Python ecosystem** provides a single environment that is sufficient for the vast majority of astronomical analysis. It does so on several levels:

- open source, which means it is free (as opposed to proprietary languages (like IDL) which require you to buy a license)
- powerful language, many scientific libraries available.
- strong set of 3rd party analysis tools that are professionally and actively developed.
- robust methods for binding with C, C++ and FORTRAN libraries (speed, legacy)
- standard library support for web, GUI, databases, process management, etc.
- very active user and developer community

Why use Python?

The Python 3 documentation can be found here:
`https://docs.python.org/3/`

In addition, Python libraries typically are very well documented, such as `mathplotlib`'s documentation:
`https://matplotlib.org/`.

Python in Astronomy

Python has become the language of choice for astronomers and astrophysicists working with data analysis and visualization.

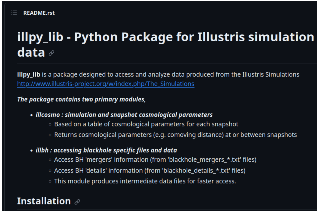
Python in Astronomy

Python has become the language of choice for astronomers and astrophysicists working with data analysis and visualization.

examples:

Cosmologists running Large-scale simulations (usually with C++ or Fortran for efficiency) use Python for **analyzing and visualizing** the results of their simulations.

Alert brokers allow **rapid access to astronomical observations** for the astronomical community world-wide.



README.rst

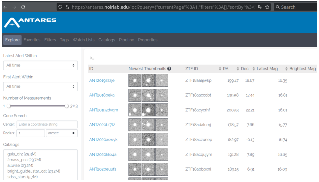
illpy_lib - Python Package for Illustris simulation data

illpy_lib is a package designed to access and analyze data produced from the Illustris Simulations http://www.illustris-project.org/w/index.php/The_Simulations

The package contains two primary modules,

- ***illcosmo : simulation and snapshot cosmological parameters***
  - Based on a table of cosmological parameters for each snapshot
  - Returns cosmological parameters (e.g. comoving distance) at or between snapshots
- ***illbh : accessing blackhole specific files and data***
  - Access BH 'mergers' information (from 'blackhole_mergers_*.txt' files)
  - Access BH 'details' information (from 'blackhole_details_*.txt' files)
  - This module produces intermediate data files for faster access.

Installation



The Python Language

Python is an **interpreted high-level programming language** for general-purpose programming.

The Python Language

Python is an **interpreted high-level programming language** for general-purpose programming.

Definitions:

An *interpreted* language is a language in which the implementations execute instructions directly without earlier compiling a program into machine language.

The Python Language

Python is an **interpreted high-level programming language** for general-purpose programming.

Definitions:

An *interpreted* language is a language in which the implementations execute instructions directly without earlier compiling a program into machine language.

High-level programming languages are designed to allow humans to write computer programs without having to have specific knowledge of the processor or hardware that the program will run on.

The Python Language

Python is an **interpreted high-level programming language** for general-purpose programming.

Definitions:

An *interpreted* language is a language in which the implementations execute instructions directly without earlier compiling a program into machine language.

High-level programming languages are designed to allow humans to write computer programs without having to have specific knowledge of the processor or hardware that the program will run on.
Examples of high-level programming languages in active use today include:

- Python
- Java
- C++
- C#
- Visual Basic

Installing Python

Nowadays many Linux and UNIX distributions, and even some Windows computers come with Python already installed.

If you do need to install Python: it is easy, download packages can be found on the official website:

`https://www.python.org/`

`https://www.python.org/downloads/`

Installing Python

The Anaconda Repository

It features over 8,000 open-source data science and machine learning packages, built and compiled for all major operating systems and architectures.

`https://www.anaconda.com/`

Conda

Conda is an open-source package and environment management system for Windows, macOS, and Linux. Conda quickly installs, runs, and updates packages and their dependencies. It also easily creates, saves, loads, and switches between environments on your local computer. Despite created for Python programs, it can package and distribute software for any language.

```
[bash-3.2$ conda install pandas
Collecting package metadata (current_repodata.json) : done
Solving environment: done

## Package Plan ##

    environment location: /opt/anaconda3

    added / updated specs:
        - pandas

The following packages will be downloaded:

package               |         build
-------------------------------------
pandas- 1.            | py39he9d5cce_1          9.4 MB
-------------------------------------------------------
                                Total:          9.4 MB

The following packages will be UPDATED:
```

Installing Python

Python can be used in a variety of ways: from the console (terminal), by loading program code previously written and saved in an editor, or as Jupyter notebooks.

We use Python 3.x. Python 2 is officially not supported as of January 1, 2020. In case you're still on Python 2.7, upgrade now as this course has a lot of code that won't run on Python 2.

Installing Python

Python can be used in a variety of ways: from the console (terminal), by loading program code previously written and saved in an editor, or as Jupyter notebooks.

We use Python 3.x. Python 2 is officially not supported as of January 1, 2020. In case you're still on Python 2.7, upgrade now as this course has a lot of code that won't run on Python 2.

After the installation process is done, go to your shell's command prompt via Terminal:

```
$ python --version
```

The response should look like:

```
Python 3.5.1 :: Anaconda 2.4.0 (x86_64)
```

Interacting with Python

Within Python, you have access to two different terminals, the Python terminal and the iPython terminal (interactive Python).

To begin an interactive iPython session, simply type

```
$ ipython
```

You should see your prompt change to

```
[IN]:
```

with a line number. You are now in iPython.

To instead run a Python script, simply type

```
$ python3 my_script.py
```

Interacting with Python

As it's a tradition to begin a programming course with a guide to showing the canonical phrase "Hello World" on the screen, we just do so.

All you have to do is type

```
print('hello world!')
```

in iPython and press `Enter`. The terminal will respond by showing your phrase in the output of the line below.

Interacting with Python


```
print('hello world!')
```

Definition

A print statement is a line of code which tells the interpreter (the program that turns your commands into computer bits and executes them) to output something to the screen.

In the above the phrase "hello world!" was placed inside quotation marks, and being surrounded by parenthesis.
Here, `print` is acting as a **function**, something that takes an argument and returns a result, similar to a mathematical function such as $sin(x)$.
The quotes, on the other hand, are described in the next section - they are the `string` data type, and can be either single or double quoted.

Data Types

Python, like most programming languages, divides up all the possible data into what are called **data types**.

Data Types

Python, like most programming languages, divides up all the possible data into what are called **data types**.

Definition

Data types are the fundamental building blocks of a code, a property that every object/element/variable in a written code will have, and which will determine the rules by which a programming language operates on them.

Data Types

Python, like most programming languages, divides up all the possible data into what are called **data types**.

Definition

Data types are the fundamental building blocks of a code, a property that every object/element/variable in a written code will have, and which will determine the rules by which a programming language operates on them.

Some of these different data types seem obvious: clearly a word like `"star"` is a fundamentally different data type than an list of numbers `[1,2,3,4,5]`.

Data Types

Python, like most programming languages, divides up all the possible data into what are called **data types**.

Definition

Data types are the fundamental building blocks of a code, a property that every object/element/variable in a written code will have, and which will determine the rules by which a programming language operates on them.

Some of these different data types seem obvious: clearly a word like `"star"` is a fundamentally different data type than an list of numbers `[1,2,3,4,5]`.

Other divisions seem more arbitrary at first glance: For example Python makes the distinction between integers (the counting numbers), and floats (numbers with decimals). It does so because of the way computer processors store information in bits, but it leads to the interesting (and important) characteristic that `42` and `42.` are different in Python, and take up different amounts of computer memory.

Data Types

Some **basic data types** are listed and defined below:

Data Types

Some **basic data types** are listed and defined below:

Integers: The counting numbers. Examples: `-1`, `0`, `1`, `2`, `3`, `4`, `5`, ...

Floats: Decimal numbers. Examples: `1.`, `2.345`, `6.3`, `999.99999`, ...

Strings: An iterable data type most commonly used to hold words/phrases or path locations. Denoted by single or double quotes. Examples: `"cat"` , `"/home/username/directory"`, `"1530"`, ...

Boolean: A data type with only two possible values: `True`, or `False`. They are used in conditional statements.

Lists: Stored lists of any combination of data types, denoted with brackets. Examples: `[1,2,'star','fish']` or `[1, 2, [3, 4, 5], 'star']` (notice that you can have lists within lists)

Numpy Arrays: Like lists, but can only contain one data type at a time, and have different operations. Defined in `numpy`, not native Python, but so ubiquitous we include them here.

Data Types

Python also allows for **composed data types** such as the ones below:

Tuples: Also like a list, but immutable (un-changable). These are defined with parentheses. Example: `tuple_1 = ('hi', 1, 4, 'bye')`

Dictionaries: A collection of pairs, where one is a *key* and the other is a *value*. One can access the value attached to a key by indexing the dictionary by key:
`dictionary_name['key']`

Basic Mathematical Operations

Within Python you can perform simple to very complex mathematical operations. Let's see how adding and subtracting works in iPython.

Basic Mathematical Operations

Within Python you can perform simple to very complex mathematical operations. Let's see how adding and subtracting works in iPython.

```
[IN]: 3 + 5
[OUT]: 8
[IN]: 9 - 3
[OUT]: 6
```

Basic Mathematical Operations

Within Python you can perform simple to very complex mathematical operations. Let's see how adding and subtracting works in iPython.

```
[IN]: 3 + 5
[OUT]: 8
[IN]: 9 - 3
[OUT]: 6
```

We can also test multiplication and division (denoted in Python with $*$ and /):

```
[IN]: 4 * 3
[OUT]: 12.0
[IN]: 1 / 2
[OUT]: 0.5
```

Basic Mathematical Operations

In Python 2.x the result of a division of integers was an integer. Now in Python 3.x the result is a float.

Basic Mathematical Operations

In Python 2.x the result of a division of integers was an integer. Now in Python 3.x the result is a float.
For your general knowledge, there is a function for converting integers to floats: `float(·)`, e.g. `float(2)`. This is called **hard casting**. Examples:

```
[IN]: x = float(x)
[IN]: x = int(x)
```

Basic Mathematical Operations

In Python 2.x the result of a division of integers was an integer. Now in Python 3.x the result is a float.
For your general knowledge, there is a function for converting integers to floats: `float(·)`, e.g. `float(2)`. This is called **hard casting**. Examples:

```
[IN]: x = float(x)
[IN]: x = int(x)
```

A much faster way to create floats when you are entering a number manually, which is simply to add a decimal (period) to any number, e.g. `12.`, `1./2`.

Basic Mathematical Operations

Another basic math operation in Python is exponentiation. Is denoted with a double asterisk (**). An example:

```
[IN]: 2**3
[OUT]: 8
```

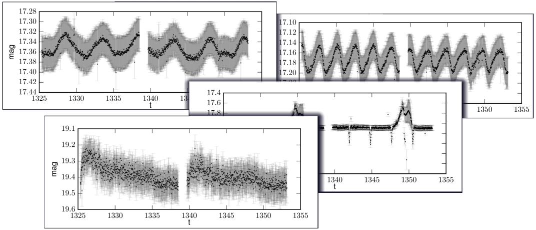
To perform operations like sin, cos, sqrt, etc., the use of some additional **packages** is required.

Storing and Manipulating Data in Python

Our primary goal for using Python as astronomers is as a tool with which to explore and manipulate data.

different types of data typically found in **astronomy**:

astronomical images, light curves, spectra, the output files of a supercomputer simulation, etc.

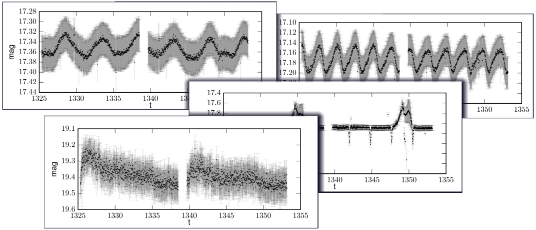


Storing and Manipulating Data in Python

Our primary goal for using Python as astronomers is as a tool with which to explore and manipulate data.

different types of data typically found in **astronomy**:

astronomical images, light curves, spectra, the output files of a supercomputer simulation, etc.



what all of these share in common is that they represent the plural inherent in the word "data": we might have a collection of 10,000, or even over a million stars or other objects

Variables

Definition

A variable is a user-defined, symbolic name which points to a spot in a computer's memory where a value has been stored. The variable's name can then be used to retrieve the value, and the value can be changed at will.

Variables

Definition

A variable is a user-defined, symbolic name which points to a spot in a computer's memory where a value has been stored. The variable's name can then be used to retrieve the value, and the value can be changed at will.

Declaring variables in Python is easy. For example:

```
[IN]: x = 5.0
[IN]: y = 'cat'
[IN]: header = 'Signal ' + 'Analysis'
```

Variables

Definition

A variable is a user-defined, symbolic name which points to a spot in a computer's memory where a value has been stored. The variable's name can then be used to retrieve the value, and the value can be changed at will.

Declaring variables in Python is easy. For example:

```
[IN]: x = 5.0
[IN]: y = 'cat'
[IN]: header = 'Signal ' + 'Analysis'
```

The latter is a way to concatenate a string.

Variables

Notice that Python doesn't output anything when you declare a variable as it did when you entered a math operation. But you can print them by typing:

```
[IN]: print(x)
[OUT]: 5.0
```

Variables

Variables in Python are mutable - they can be changed. An example:

Variables

Variables in Python are mutable - they can be changed. An example:

```
[IN]: x = 5
[IN]: x = 3
```

Variables

Variables in Python are mutable - they can be changed. An example:

```
[IN]: x = 5
[IN]: x = 3
```

If you output `x` you would find it is equal to 3. Another example:

```
[IN]: x = 4
[IN]: x = 2 * x + 7
```

Variables

Variables in Python are mutable - they can be changed. An example:

```
[IN]: x = 5
[IN]: x = 3
```

If you output `x` you would find it is equal to 3. Another example:

```
[IN]: x = 4
[IN]: x = 2 * x + 7
```

In this case, the new value for `x` at the end of the line would be 2 times the value of `x` going in, plus 7. More of this:

Variables

Variables in Python are mutable - they can be changed. An example:

```
[IN]: x = 5
[IN]: x = 3
```

If you output `x` you would find it is equal to 3. Another example:

```
[IN]: x = 4
[IN]: x = 2 * x + 7
```

In this case, the new value for `x` at the end of the line would be 2 times the value of `x` going in, plus 7. More of this:

```
[IN] : x = 4.
[IN] : y = 3.
[IN] : z = x * y
[IN] : x = z + 2
```

That is probably a bit confusing to follow, and **illustrates why typically we avoid redefining variables**, and instead come up with new variables to store the results of computations.

Arrays vs. Lists

We begin with lists, as these are the native data type within Python. Let's define a list:

Arrays vs. Lists

We begin with lists, as these are the native data type within Python. Let's define a list:

```
[IN]: my_list = [1,5,2,7,3,7,8]
```

Arrays vs. Lists

We begin with lists, as these are the native data type within Python. Let's define a list:

```
[IN]: my_list = [1,5,2,7,3,7,8]
```

Assume this list contains the rounded distances in parsecs of several nearby stars. We multiply by 2, as e.g. an equation has a factor of $2d$:

```
[IN]: my_list*2
[OUT]: [1,5,2,7,3,7,8,1,5,2,7,3,7,8]
```

Arrays vs. Lists

We begin with lists, as these are the native data type within Python. Let's define a list:

```
[IN]: my_list = [1,5,2,7,3,7,8]
```

Assume this list contains the rounded distances in parsecs of several nearby stars. We multiply by 2, as e.g. an equation has a factor of $2d$:

```
[IN]: my_list*2
[OUT]: [1,5,2,7,3,7,8,1,5,2,7,3,7,8]
```

This obviously didn't work as intended: a new list with the original list repeated N times was created.
This default behavior this actually makes sense: recall that lists can contain any data type, and indeed any combination of data types. If our list contained a mix including strings or any other non-numerical data type, this operation would fail if defined this way.

Arrays vs. Lists

The correct way to multiply the list elements by a factor N is the following: We have do utilize what is known as **iterating** to go through the list one by one and replace each value with N times itself. Example:

```
[IN]: [i*2 for i in my_list]
[OUT]: [2,10,4,14,6,14,16]
```

As it turns out, there is a shorter way to apply mathematical operations to every element of a collection. Numpy arrays allow to perform mathematical operations on entire arrays all at once. Example:

```
[IN]: my_array = np.array([1,2,3])
[IN]: my_array*2
[OUT]: array([2,4,6])
```

This is also faster than iteration as arrays are being treated computationally as matrices, which can be computed very efficiently.

Data Structures

From this we see: Particularly when dealing with large datasets, you almost certainly will be working with (numpy) arrays.

How to operate on those array containers? How to access data?

Data Structures

From this we see: Particularly when dealing with large datasets, you almost certainly will be working with (numpy) arrays.

How to operate on those array containers? How to access data?

The procedure of extracting subsets of a larger array/list is known as **slicing**, or **indexing**.
By convention, this index starts with 0, rather than one (this is true for most programming languages). Here is a sample list, with the indices for each entry listed below:

```
list_1 = [1, 2, 3, 4, 'star', 198]
index:    0  1  2  3  4        5
```

Data Structures

From this we see: Particularly when dealing with large datasets, you almost certainly will be working with (numpy) arrays.

How to operate on those array containers? How to access data?

The procedure of extracting subsets of a larger array/list is known as **slicing**, or **indexing**.
By convention, this index starts with 0, rather than one (this is true for most programming languages). Here is a sample list, with the indices for each entry listed below:

```
list_1 = [1, 2, 3, 4, 'star', 198]
index:    0  1  2  3  4       5
```

We now extract the first entry of the array:

```
[IN]: list_1 = [1, 2, 3, 4, 'star', 198]
[IN]: x = list_1[0]
[IN]: print x
[OUT]: 1
```

Array Manipulation

```
list_1 = [1, 2, 3, 4, 'star', 198]

index:     0  1  2  3  4       5
```

We can also extract the last entry (in a very convenient way):

```
[IN]: list_1 = [1, 2, 3, 4, 'star', 198]
[IN]: x = list_1[0]
[IN]: print x
[OUT]: 1
[IN]: y = list_1[-1]
[IN]: print y
[OUT]: 198
```

The same works for the second last entry, and so on.

Array Manipulation

There is also a way to slice through multiple indices at once. The format is as follows:

```
var = array[from:to]
```

Array Manipulation

There is also a way to slice through multiple indices at once. The format is as follows:

```
var = array[from:to]
```

An example: We have a string '123456789' and we want to extract the first four elements and convert them into an integer:

```
[IN]:x = '123456789'
[IN]: H = int(x[0:4])
[IN]: print(H)
[IN]: 1234
```

Array Manipulation

There is also a way to slice through multiple indices at once. The format is as follows:

```
var = array[from:to]
```

An example: We have a string '123456789' and we want to extract the first four elements and convert them into an integer:

```
[IN]:x = '123456789'
[IN]: H = int(x[0:4])
[IN]: print(H)
[IN]: 1234
```

Caution: Python slicing will not include the end index.

Array Manipulation

There is also a way to slice through multiple indices at once. The format is as follows:

```
var = array[from:to]
```

An example: We have a string '123456789' and we want to extract the first four elements and convert them into an integer:

```
[IN]:x = '123456789'
[IN]: H = int(x[0:4])
[IN]: print(H)
[IN]: 1234
```

Caution: Python slicing will not include the end index.
As a shortcut, if you are starting from the beginning, or slicing from some midpoint to the end, you can omit the 0 or the final index after, i.e.,

```
[IN]: print x[0:4]
```

is equivalent to

```
[IN]: print x[:4]
```

Array Manipulation

Often astronomical data (like images or tables) are stored in higher-dimensional arrays, essentially matrices described by 2 indices, a row and a column.

Array Manipulation

Often astronomical data (like images or tables) are stored in higher-dimensional arrays, essentially matrices described by 2 indices, a row and a column.

Example:

```
[IN]: print A
[OUT]: [[1 , 3, 4, 5, 6]
[ 4, 5, 9, 3, 7]
[ 9, 4, 6, 7, 1 ]]
```

We slice it with two indices, row, then column.

Array Manipulation

Often astronomical data (like images or tables) are stored in higher-dimensional arrays, essentially matrices described by 2 indices, a row and a column.

Example:

```
[IN]: print A
[OUT]: [[1 , 3, 4, 5, 6]
[ 4, 5, 9, 3, 7]
[ 9, 4, 6, 7, 1 ]]
```

We slice it with two indices, row, then column.
Caution: row then column translates into (y,x), which is the opposite of how we are usually taught to determine ordered pairs of coordinates.

Array Manipulation

Often astronomical data (like images or tables) are stored in higher-dimensional arrays, essentially matrices described by 2 indices, a row and a column.

Example:

```
[IN]: print A
[OUT]: [[1 , 3, 4, 5, 6]
[ 4, 5, 9, 3, 7]
[ 9, 4, 6, 7, 1 ]]
```

We slice it with two indices, row, then column.
Caution: row then column translates into (y,x), which is the opposite of how we are usually taught to determine ordered pairs of coordinates.

Example: From the above data structure, we want to get the 3 in the second row:

```
[IN]: A[1][3]
[OUT]: 3
```

Array Manipulation

Often astronomical data (like images or tables) are stored in higher-dimensional arrays, essentially matrices described by 2 indices, a row and a column.

Example:

```
[IN]: print A
[OUT]: [[1 , 3, 4, 5, 6]
[ 4, 5, 9, 3, 7]
[ 9, 4, 6, 7, 1 ]]
```

We slice it with two indices, row, then column.
Caution: row then column translates into (y,x), which is the opposite of how we are usually taught to determine ordered pairs of coordinates.

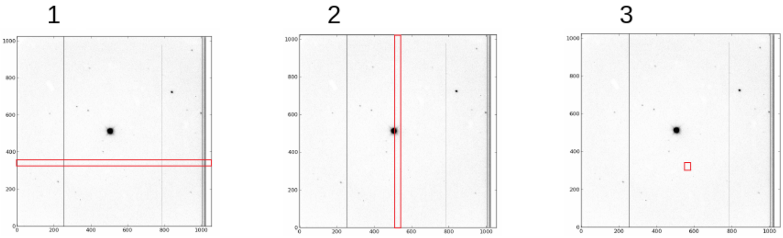
Example: From the above data structure, we want to get the 3 in the second row:

```
[IN]: A[1][3]
[OUT]: 3
```

Alternatively, you can use the comma syntax `A[1,3]` to equal effect.

Array Manipulation

Given a 2D array, we can also illustrate this in the following way:



1. `array[350:370,:]`
   this takes the full rows 350-370 in the image

2. `array[:,350:360]`
   this takes the full columns 350-360 in the image

3. `array[350:370, 350:360]`
   this takes the box of the region between rows 350-370 and columns 350-360 in image

Dictionaries

The final primary data container data type in Python are dictionaries.

Definition

A dictionary is a Python container, like a list or array, but which uses *keys* instead of indices to specify elements within the container. That is, the order of elements (values) in a dictionary is irrelevant, and values are retrieved by indexing for the appropriate key (which can be almost anything).

Dictionaries

Dictionaries in Python are created using curly brackets, inside which go key-value pairs (colon separated), which themselves are separated by commas. Example:

```
simple_dict = {'key1':value1, 'key2':value2}
```

Dictionaries

Dictionaries in Python are created using curly brackets, inside which go key-value pairs (colon separated), which themselves are separated by commas. Example:

```
simple_dict = {'key1':value1, 'key2':value2}
```

To pull a value the dictionary, the key is used. Example:

```
pulled_value = simple_dict['key1']
```

Dictionaries

Dictionaries in Python are created using curly brackets, inside which go key-value pairs (colon separated), which themselves are separated by commas. Example:

```
simple_dict = {'key1':value1, 'key2':value2}
```

To pull a value the dictionary, the key is used. Example:

```
pulled_value = simple_dict['key1']
```

We can also easily change values in a dictionary, or add new key-value pairs, using this index notation. Example:

```
simple_dict['key2'] = new_val
```

Motivation
Course Logistics
Why use Python?
Getting started
Data Types
Basic Math
Variables and Data Structures
Outlook

Dictionaries

Dictionaries in Python are created using curly brackets, inside which go key-value pairs (colon separated), which themselves are separated by commas. Example:

```
simple_dict = {'key1':value1, 'key2':value2}
```

To pull a value the dictionary, the key is used. Example:

```
pulled_value = simple_dict['key1']
```

We can also easily change values in a dictionary, or add new key-value pairs, using this index notation. Example:

```
simple_dict['key2'] = new_val
```

To add a new key-value pair:

```
simple_dict['new_key'] = new_value
```

These examples use string keys, but any other data type would be possible.

An Outlook: Control Flow Statements

We have seen how to write simple Python commands.

To put those commands together into Python scripts, we will see how to use **control flow statements**, and also how to access more complex algorithms from **libraries**.