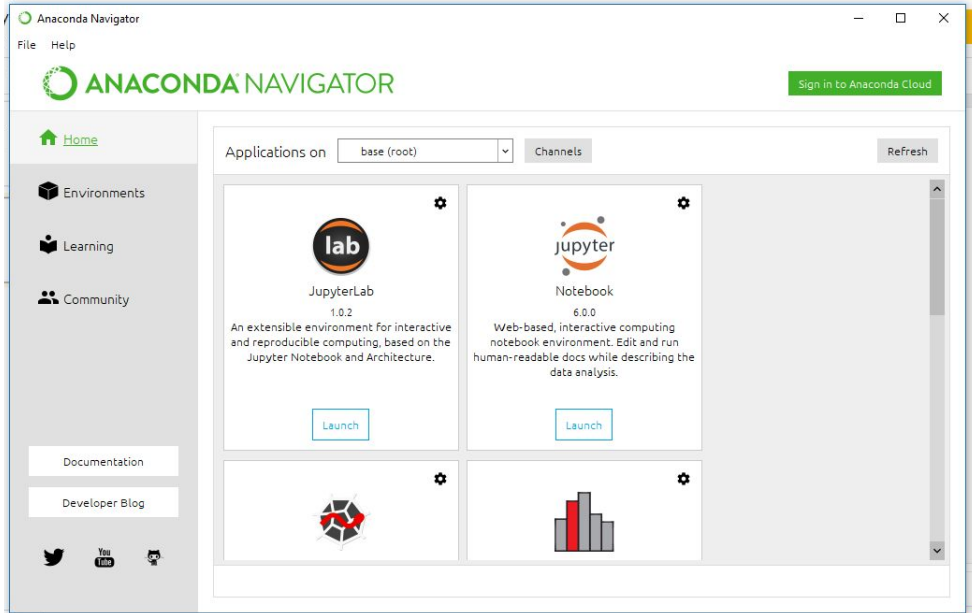
# Week 3: Arrays, Lists & Dictionaries

A lot of data, gotta find somewhere to put it

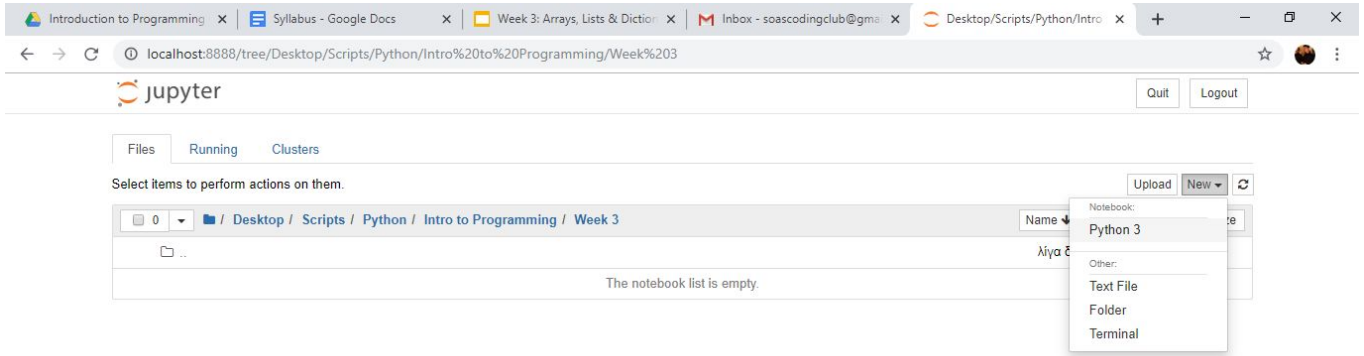
# Anaconda 3.7 & Jupyter Notebook

From now on we will be using Jupyter Notebook to work on our code.
Make sure to have downloaded Anaconda 3.7.



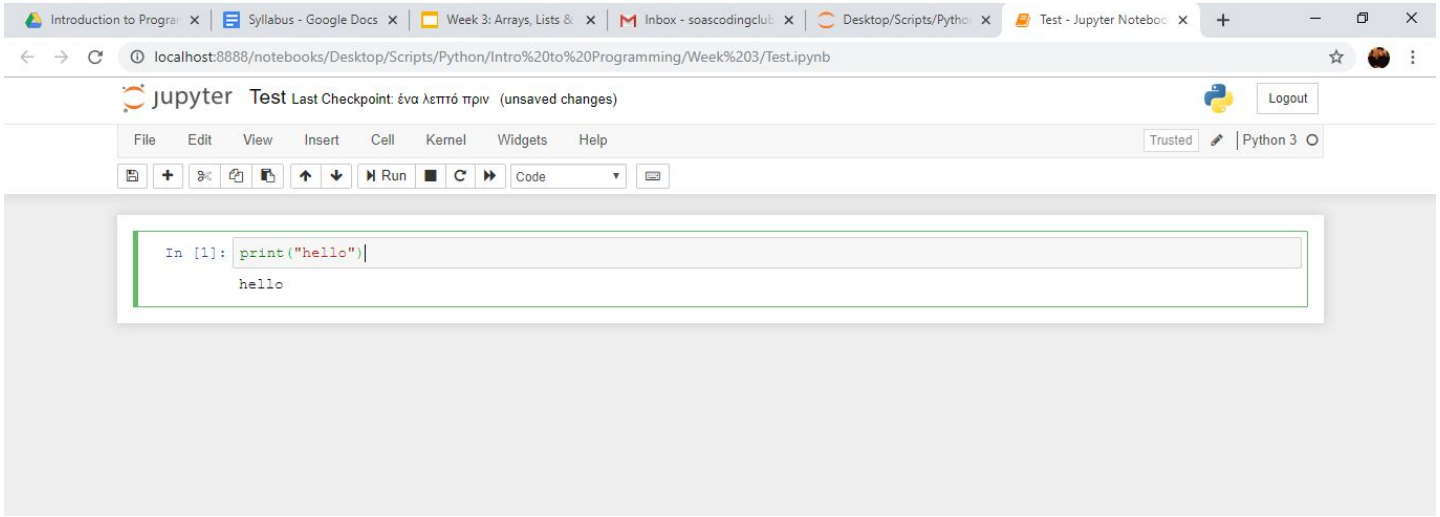
# Launch Jupyter & Set up a folder

Press launch and the navigate to where you want to store your Python scripts.



# Jupyter Notebook

The following is what you see when you run Jupyter. To run the code in each block click on the block and press **Ctrl + Enter.**



# Arranging Data. Lists

Sometimes we want to store data that is linked in a certain way. Say names of staff members of a company or, a small quantitative dataset. To store these structures we have **Lists**.

A list can be **any** of the basic data types. We can have an Integer list or a String list, etc.

list = [entry1, entry2, entry3, … ]

# Two ways of declaring lists

We might have a pre-filled list or one we want to populate during program runtime. As such we have two ways of creating arrays.

**Populated:** list_name = [entry1, entry2, entry3, … ]

**Unpopulated but defined:** list_name = []

# Data lacks uniformity

In Python lists do not have a predefined type and each cell can have a different data type. This helps avoid crashes but can make programs very easy to break. Using data type checks after user input is a good way to avoid insecure arrays.

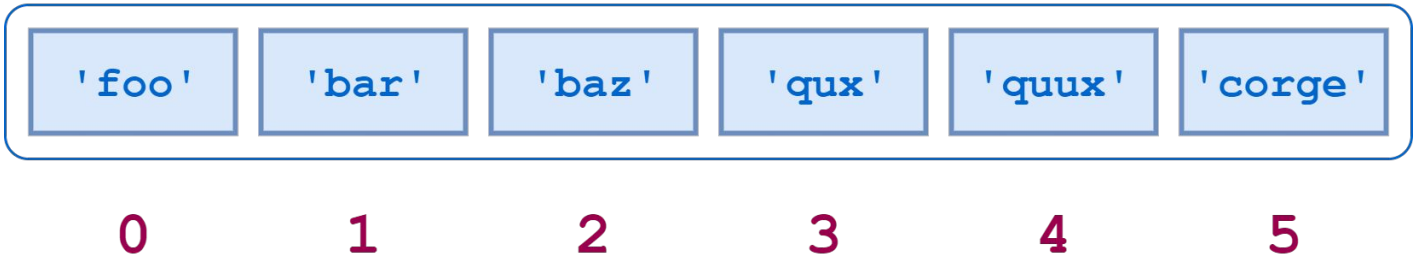
How to do this:

**type**(variable_to_check) **is** type_we_want

The above has a boolean output and can be used to make such checks.

# Accessing Lists (Recap)

Each list stores **TWO** instances of data for each entry. One is the **value** and the other is the **index**. Each list has indexes that start from **0** and run until the end of the list.




**list_name [index]** gives us the item in that index

# Exercise 1: Linear Search with Type Catch

Create a list with data of your choice and allow user input to input something that should be found in the array. Check to see if the input type matches the type of elements in the array and only if that is true run the program.

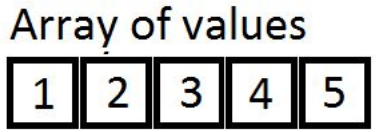
**Note that:** The list should be of a singular data type.

! There is a logic error in this exercise but try to solve it anyway !

**Time Allowed:** 10 Minutes.

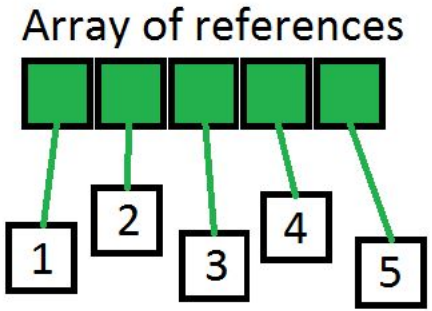
# One Type: Arrays

Lists allow us to group different data types under a single structure. Arrays allow us to group data of the same type under a single structure.




Fast random access,
least memorry consumption,
can contain only items of same type (size).

Represented by 'array' class in Python.

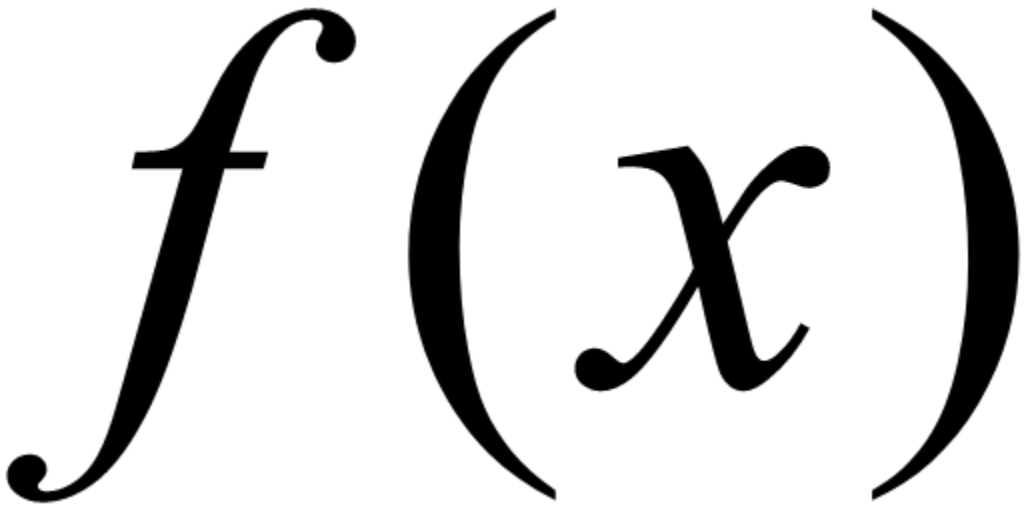



Fast random access,
needs one extra pointer for every item,
can contain items of different sizes.

Represented by 'list' class in Python.

# Detour: Functions

As in math, functions in Comp. Sci take an argument (x) and give us a value (y).



# Basic List-Array Functions

- len( ) → This gives us the length of the array. Starts at 1.
- append( ) → This adds the element which is specified in the brackets at the end of the array.
- remove( ) → Removes the first element with the specified value
- pop( ) → Removes the item in the specified position
- Reverse( ) → Reverses the array

# Let’s Explore

| Name | Return Type | Active (Modifies) |
|---|---|---|
| len | int | False |
| append | null | True |
| remove | null | True |
| pop | Array type | True |
| reverse | null | True |

For the following Exercise(s) look up how to use these functions

# Exercise 2: How many items?

Create an array of elements (String type) and allow a user to add more through an infinite loop. Make sure that the input type is Str and after an item is added output the total number of elements in the array.

**Allowed Time:** 7.5 Minutes

# Exercise 3: Kill the worst performers

Create an array of 10 Float values that represent running performance for a 100m race. Find those with the 3 worst times and “kill” them by removing their times. Extra points for good User Interface.

Use *sort( )* to sort the array before reversing

Don’t do it with a loop. Do it with sequential one line commands.

**Allowed Time:** 5 Minutes

# Tuples

Tuples are a type of list which is ordered and unchangeable. They work in similar fashion to arrays and lists but are defined differently.

Tuple_time = (“Word 1”, “Word 2”, “Word 3”)

Tuples can be changed if they are converted into a list. These will not be covered more and if you are interested in using them for some application read the Documentation.

# Btec Spreadsheets: Linking Lists

In contrast to C++ or Java, Python is heavily reliant on CSV as a form of creating multidimensional arrays. Thus, we will be looking at a simple way of creating a two dimensional array using two independent lists.

| Brad | Tom | Jane | Kyle | Anna |
|---|---|---|---|---|
| Goldman Sachs | HSBC | JP Morgan | Deutsche Bank | BOA |

# Exercise 4: Banker Catalogue (Contacts)

Given a triple linked list (Name, Company, Salary) create a program that adds new contacts with such details to the lists. Use appropriate user I/O.

Lists to be used:

Name = [Brad, Tom, Jane, Kyle, Anna]
Company = [Goldman Sachs, HSBC, JP Morgan, Deutsche Bank, BOA]
Salary = [95, 87, 125, 65, 250]

**Allowed Time:** 7.5 Minutes

# Dictionaries

A Dictionary is similar to a list in the way data is stored with the exception of indexing. Unlike lists and arrays whose indexes are numeric and relative to the items position in the structure, dictionaries use keys.

Dictionaries are therefore, associative arrays that associate the data of the Key with the data of the Entry.

You can think of them as a two column excel spreadsheet

# Classes, Java and .JSON

In many real life applications of object oriented programming, we tend to need to create classes (or objects) that contain empty but defined variables that each of these objects have.

```
//Route class that holds data for each

public class Route {
    //instance variables
    private int length;
    private double base_elevation;
    private double top_elevation;
    private int aspect;
    private String route_name;
    private double elevation_change;
    private double slope;
    private int add_index;
```

# Dictionaries as reference tables

Dictionaries can be used as simple reference tables. Since we need the Key to access each element, we can thus create simple lookup tables.

Example:

```
MLB_teams = {
    'Colorado' : 'Rockies'
    'Boston' : 'Red Sox'
    .
    .
    }
```

Accessing Dictionaries:
MLB_teams['Colorado'] → Rockies

# Exercise 5: Phone Book

Using a dictionary, create a phonebook with a name and a phone number. Using an array, have all the names you have in phone book and print them for the user. Then ask the user for input where the input should be put through the dictionary to print the phone number of the specified name.

You can use the expressions: *value **in** dictionary*
To check if the value exists and if not have an appropriate output.

**Allowed Time:** 10 Minutes.

# Arrays of Dictionaries. Arranging Objects

Since we can define a singular object using a dictionary, what if we could create an array of dictionaries with the same keys?

By doing this we can create the equivalent of a Java Object array in a data science rather than a OOP perspective.

A programmer would create objects, we are using dictionaries to create something akin to a spreadsheet.

# Problem: Contact Book

By creating an unbounded array which we sequentially fill with dictionaries using temporary variables, we can create a contact book array.

We will be using a basic String array to hold the Keys but also allow the user to add new contacts.

Code is in the Drive