

# AMS 595/691: Fundamentals of Computing: Part II

## Lecture 2: Built-in Data Structures; Exceptions

Xiangmin Jiao

Stony Brook University

# Built-in Data Structures

- Python has some built-in data structures (containers), such as lists, tuples, sets, dictionaries, etc.
- Lists in Python (enclosed by '[' and ']') are similar to cell arrays (enclosed by '{' and '}') in MATLAB
  - ▶ Implemented as a variable-length array, instead of linked list
  - ▶ Can store multiple different datatypes in a single list; e.g.,  
`a=[1,2,' abc ']`
  - ▶ Can be accessed by indexing and slicing
  - ▶ Can be modified by methods `append`, `insert`, `sort`, etc.
- Dictionaries are similar to list but with names (keys)
- Tuples (comma separated expressions) are similar to lists, but are immutable
- Sets (enclosed by '{' and '}') are similar to lists, but contain unordered unique elements; `frozenset` is its immutable counterpart
- Array is provided by other modules (later)
- Demo: [Jupyter notebook on advanced data types](#)

# Mutable vs. Immutable Objects

- Immutable objects
  - ▶ Basic data types: int, float, complex, string
  - ▶ tuple, frozenset
- Mutable objects
  - ▶ list, dict, set
  - ▶ numpy.array and most user-defined classes
- Assignment behaves differently for mutable and immutable objects
  - ▶ For immutable basic types (numbers and strings), '=' copies value; for mutable objects, '=' creates an "alias"; [see visualization here](#)
  - ▶ '=' not allowed for entities/data members of immutable objects
  - ▶ For mutable objects
    - ★ `trg=src` makes `trg` a *reference* to `src`, so changing entities or data members in `src` would change those in `trg` and vice versa
    - ★ `trg=copy.copy(src)` performs *shallow copy* (so does constructor (e.g. `trg=list(src)`) or slicing (e.g., `trg=src[:]`)), so mutable entities or data members are still references
    - ★ `trg=copy.deepcopy(src)` performs *deep copy*, so mutable entities or data members are copied recursively

# Exceptions

- Unlike *syntax errors*, which are grammar errors during parsing time, runtime errors are handled using *exceptions*
- Exceptions are handled using try-except-else-finally blocks
- There are many [built-in exceptions](#), such as division by zero, non-existing key in dictionary, open non-existing file, etc.
- It is a good practice to catch errors and report useful error messages
- Use the `raise` statement to raise or re-raise exceptions
- Demo: [Jupyter notebook on exceptions](#)