ISA 419: Data-Driven Security

02: Python Programming Basics

Fadel M. Megahed, PhD

Endres Associate Professor
Farmer School of Business
Miami University

@FadelMegahed
fmegahed
fmegahed@miamioh.edu
? Automated Scheduler for Office Hours

Spring 2024

Quick Refresher of Last Class

- ✅ Define **information security (Infosec)**, its **main goals**, and how it fits within a firm's overall security protocols
- ✅ Describe the **three main steps** in information security: prevention, detection, and response.
- ✅ Explain why **prevention as a sole security measure is deemed to fail**.
- ✅ Describe **course structure**, **goals**, and **overview**.

Learning Objectives for Today's Class

- Use pseudocode to map out a problem.
- Python syntax, data types, and data structures
- Convert data types using type casting
- Manipulate lists and use methods on lists

Use Pseudocode to Map out a Cybersecurity Problem

Background: Why is BUS 104 is a Pre-Req?

From the MU Bulletin

> **BUS 104. Introduction to Computational Thinking for Business. (2)**
>
> As part of the Farmer School of Business first-year integrated core curriculum, this course introduces students to the fundamentals of computational thinking as an aid to data-driven business problem-solving. Topics include: computational thinking as **problem solving**, **representing data through abstractions**, and **thinking in terms of algorithms** (loops, conditions, reusable code, functions and events) to **automate finding solutions**. The course lays the foundations for students identifying, analyzing, and implementing solutions for data-driven business problems and the communication of results.

Source: Business Analysis (BUS) | Miami University Bulletin.

Background: (Distributed) Denial of Service Attacks

Credits: Jeff Crume. (2023) IBM Technology: Denial of Service Attacks Explained.

Simplifying our Task Ahead: SYN Flood Attacks

- We will focus on the simple **SYN flood attack**.
- Attackers send a succession of SYN requests to a target's system.

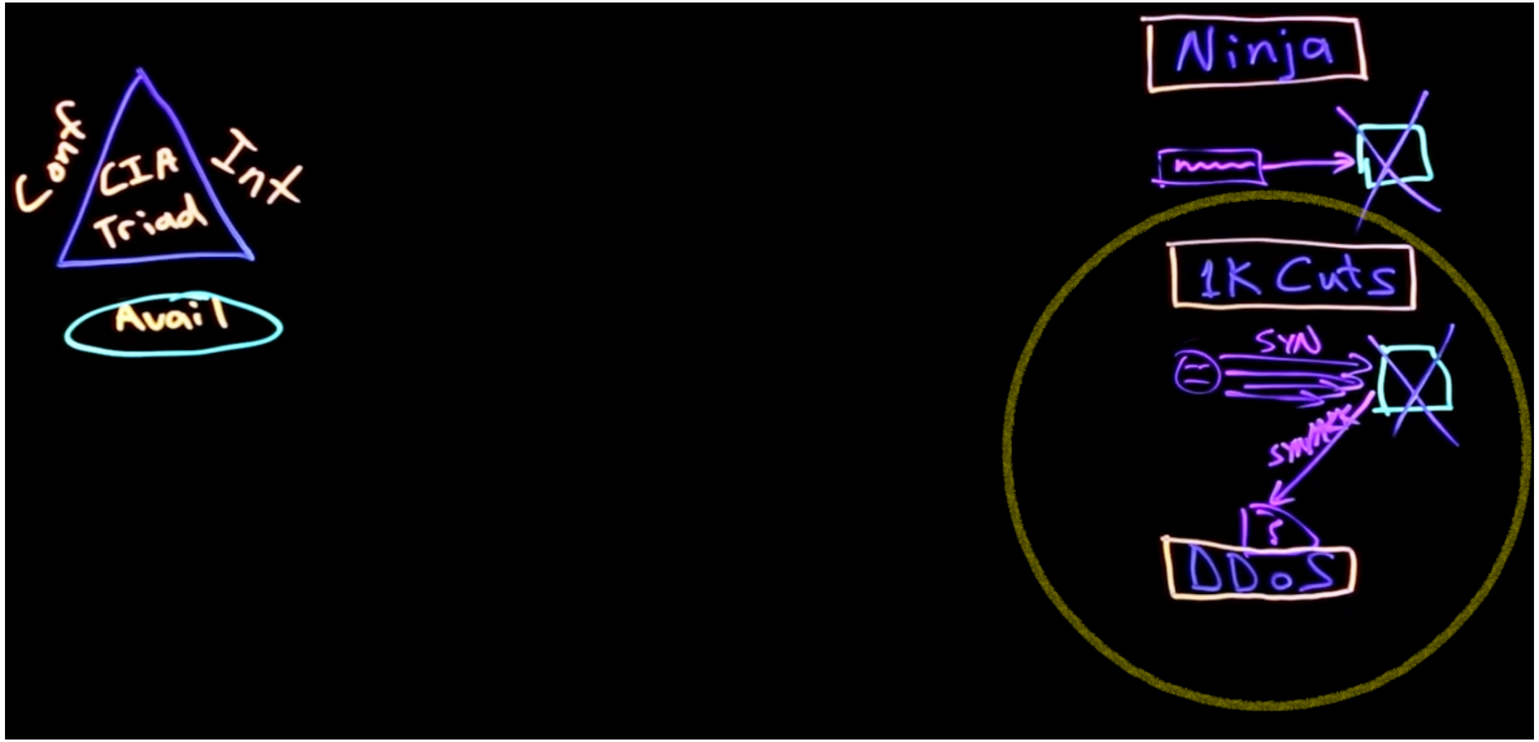


Image Source: Captured from the Jeff Crume. (2023) IBM Technology: Denial of Service Attacks Explained video (~ the frame corresponding to 3:47). Then, the image was adjusted by: (a) removing the presenter's picture, and (b) adding a circle around the SYN attack.

Class Activity: Detecting SYN Flood Attacks

10:00

Task Q1 Q2 Q3 Q4 Q5

- Toward creating an algorithm for monitoring network traffic and detecting SYN flood attacks, in pairs, please think and answer the following questions.
- Write down your answers in each tab, after each question.

Class Activity: Detecting SYN Flood Attacks

10:00

Task Q1 Q2 Q3 Q4 Q5

- What kind of **information** or **data points** do you think would be important to collect and analyze?
- **Data Points to Collect:**
  - Edit me.
  - Edit me.
  - Edit me.

Class Activity: Detecting SYN Flood Attacks

10:00

Task Q1 **Q2** Q3 Q4 Q5

- In a DDoS attack, what aspects of network traffic might change when there's an attack happening? How could you measure or quantify these changes?
- **Changes in Network Traffic during an Attack:**
  - Edit me.
  - Edit me.
  - Edit me.

Class Activity: Detecting SYN Flood Attacks

10:00

Task Q1 Q2 Q3 Q4 Q5

- Think about monitoring network traffic over a period of time. What time-related factors should you consider when designing a detection system?
- **Changes in Network Traffic during an Attack:** Edit me.

Class Activity: Detecting SYN Flood Attacks

10:00

Task Q1 Q2 Q3 **Q4** Q5

- If you were trying to identify multiple requests coming from the same IP address, how would you keep track of these requests? What kind of **data structure** could you use to achieve this?
- **Tracking Requests from the Same IP:** Edit me.

Class Activity: Detecting SYN Flood Attacks

10:00

Task Q1 Q2 Q3 Q4 Q5

- What could be some potential signs that a specific IP address is involved in a DDoS attack?
- How would you set a threshold to determine when an IP address should be flagged as a potential source of an attack?

- **Setting a Threshold:** Edit me.

Computational Thinking: From Problem to Code/Solution

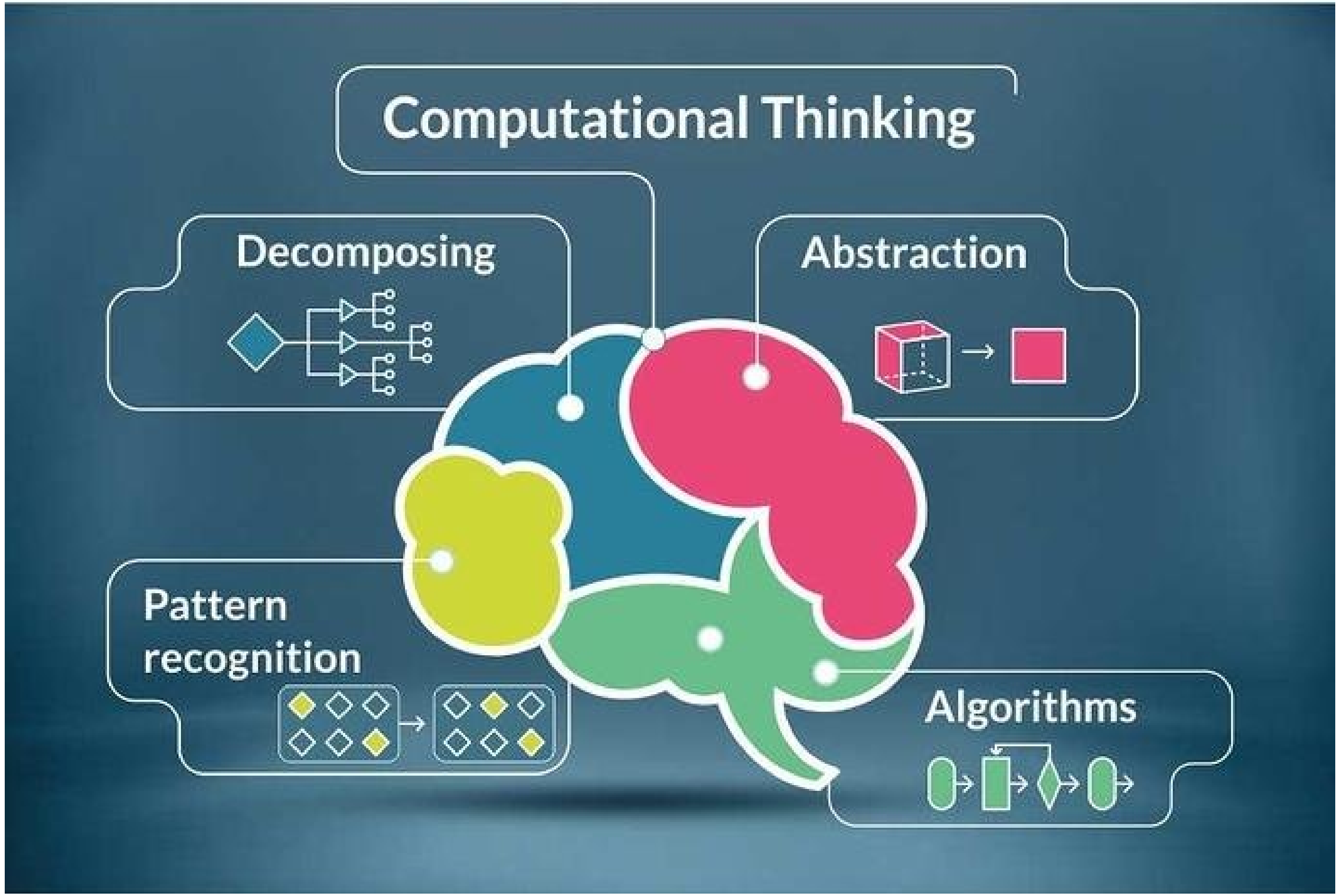
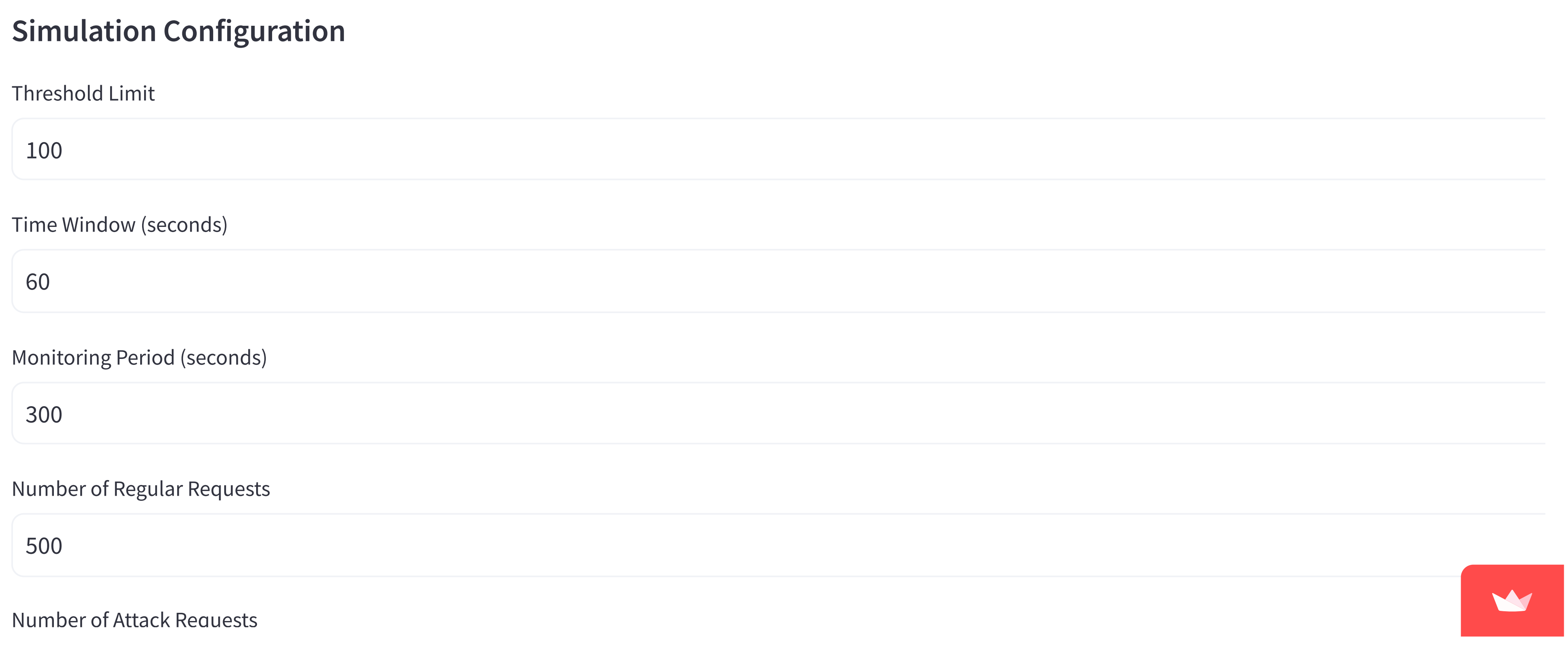


Image Source: Harimurti, R., & Asto, B. I. (2019, May). The concept of computational thinking toward information and communication technology learning. In IOP Conference Series: Materials Science and Engineering (Vol. 535, No. 1, p. 012004). IOP Publishing. Figure 02.

Pseudocode for Detecting SYN Flood Attacks

```
BEGIN DDoS_Detection
    SET threshold_limit to a significant number representing unusual traffic
    SET time_window to the period of monitoring traffic (e.g., per minute)
    SET ip_request_count to an empty data structure (e.g., dictionary)

    for each request in network_traffic
        GET ip_address from request
        GET request_time from request

        if ip_address is not in ip_request_count
            SET ip_request_count[ip_address] to 1
        else
            INCREMENT ip_request_count[ip_address] by 1

        if ip_request_count[ip_address] exceeds threshold_limit within time_window
            FLAG ip_address as potential DDoS source
            TAKE necessary action (e.g., block IP, alert admin)

        if time elapsed > time_window
            RESET ip_request_count for each ip_address

    END FOR loop
END DDoS_Detection
```

App: Detecting Simulated Attacks based on Pseudocode



Simulation Configuration

Threshold Limit

100

Time Window (seconds)

60

Monitoring Period (seconds)

300

Number of Regular Requests

500

Number of Attack Requests

Notes: Students are encouraged to "play" with the app's simulation configurations to better understand the pseudo_code. The app can be opened in your browser by navigating to: https://muddos.streamlit.app/.

Python Syntax, Data Types & Data Structures

Why are we Discussing Syntax?

> Good coding style is like **correct punctuation**: you can manage without it, **butitsuremakesthingseasiertoread**.
> -- The tidyverse style guide

> Code is **read much more often than it is written**. The guidelines provided here are intended to improve the readability of code ...
> -- PEP 8 - Style Guide for Python Code

Python Syntax: Key PEP 8 Guidelines

Indentation

- Use **4 spaces per indentation level**.
- Do **not mix tabs and spaces for indentation**.
- Example:

```python
def my_function():
    for i in range(10):
        print(i)
```

Naming Conventions

- **Functions and variables:** `lower_case_with_underscores`
- **Classes:** `CapWords` (or CamelCase)
- **Constants:** `UPPER_CASE_WITH_UNDERSCORES`
- Example:

```python
class MyClass:
    CONSTANT_VALUE = 42

    def my_method(self):
        my_variable = 10
```

Tip: Consistency with these guidelines improves code readability and maintainability. See PEP 8 - Style Guide for more details.

Python Syntax: Key PEP 8 Guidelines

Comments

- Write **clear comments (full sentences)** for tricky parts of the code.
- **Inline comments** should be used **sparingly**.
- Example:

```
# Calculate the square of x (yes)
square = x ** 2  # Square value (No)
```

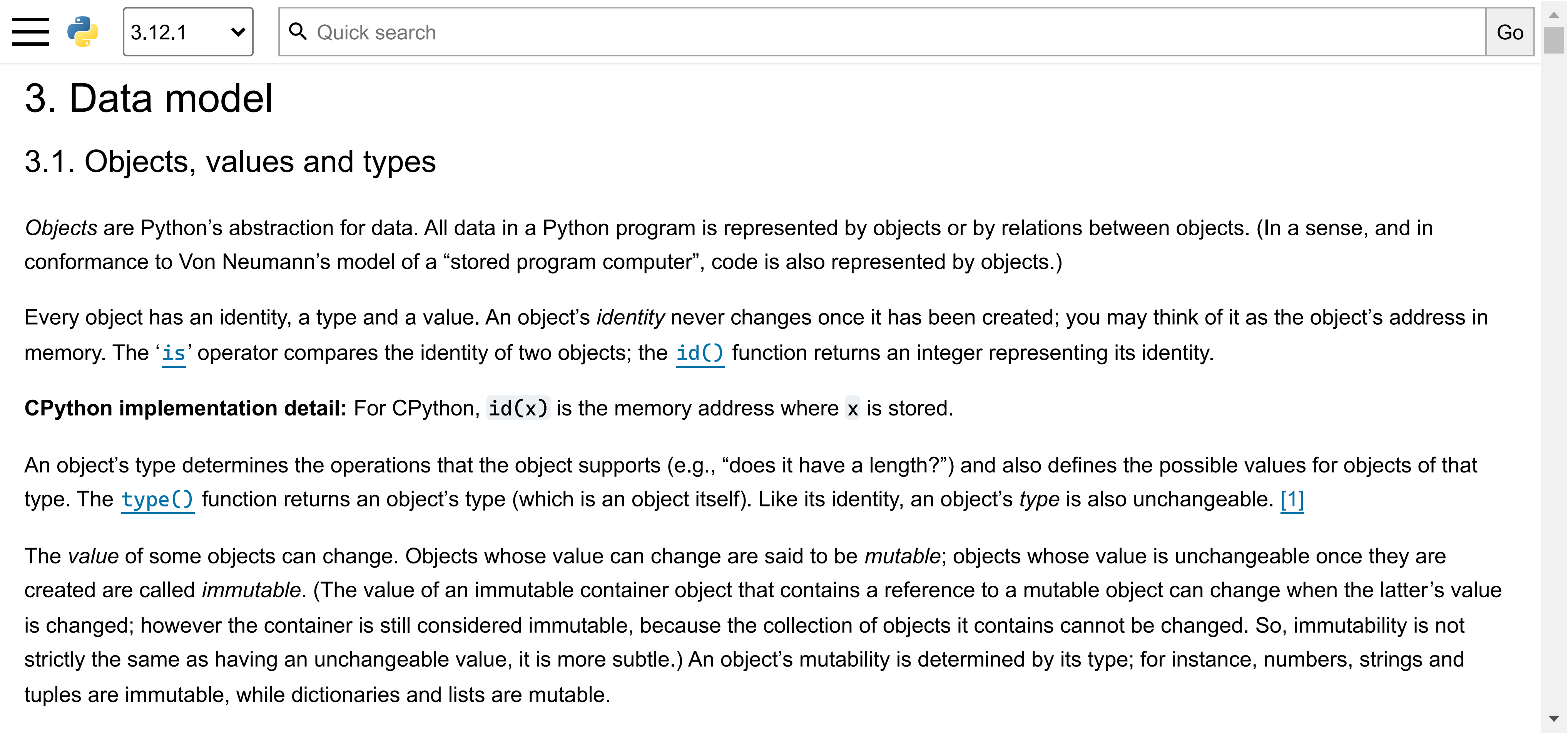
Importing Modules

- Imports should be on separate lines.
- **Order:** standard library (e.g., `os`) ⟶ related third-party (e.g., `pandas`) ⟶ local application/library specific imports (e.g. from other `.py` files).
- Example:

```
import os
import sys

import numpy as np
import pandas as pd

from my_local_module import my_function
```

Tip: Consistency with these guidelines improves code readability and maintainability. See PEP 8 - Style Guide for more details.

Python's Data Model: "Everything is an Object in Python"



3. Data model

3.1. Objects, values and types

Objects are Python's abstraction for data. All data in a Python program is represented by objects or by relations between objects. (In a sense, and in conformance to Von Neumann's model of a "stored program computer", code is also represented by objects.)

Every object has an identity, a type and a value. An object's *identity* never changes once it has been created; you may think of it as the object's address in memory. The '`is`' operator compares the identity of two objects; the `id()` function returns an integer representing its identity.

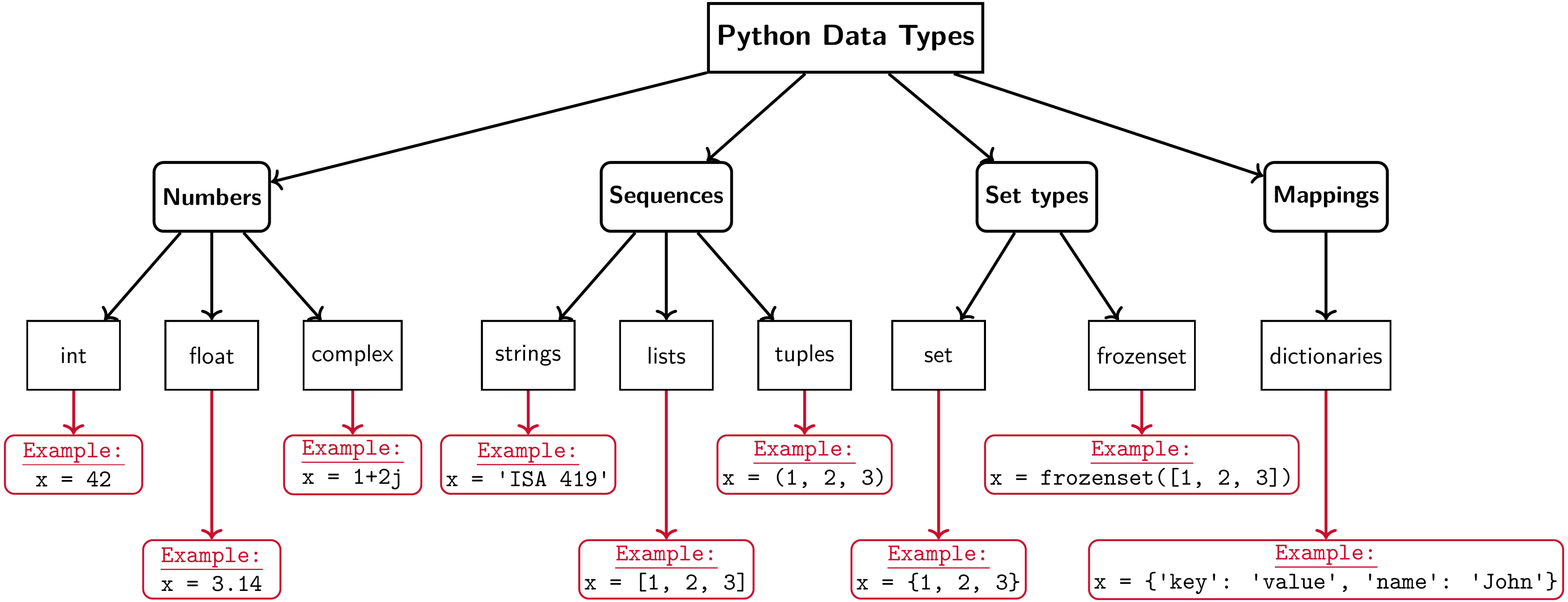
CPython implementation detail: For CPython, `id(x)` is the memory address where `x` is stored.

An object's type determines the operations that the object supports (e.g., "does it have a length?") and also defines the possible values for objects of that type. The `type()` function returns an object's type (which is an object itself). Like its identity, an object's *type* is also unchangeable. [1]

The *value* of some objects can change. Objects whose value can change are said to be *mutable*; objects whose value is unchangeable once they are created are called *immutable*. (The value of an immutable container object that contains a reference to a mutable object can change when the latter's value is changed; however the container is still considered immutable, because the collection of objects it contains cannot be changed. So, immutability is not strictly the same as having an unchangeable value, it is more subtle.) An object's mutability is determined by its type; for instance, numbers, strings and tuples are immutable, while dictionaries and lists are mutable.

Source: The embedded page is from 3. Data model - Python 3.12.1 documentation.

Data Types



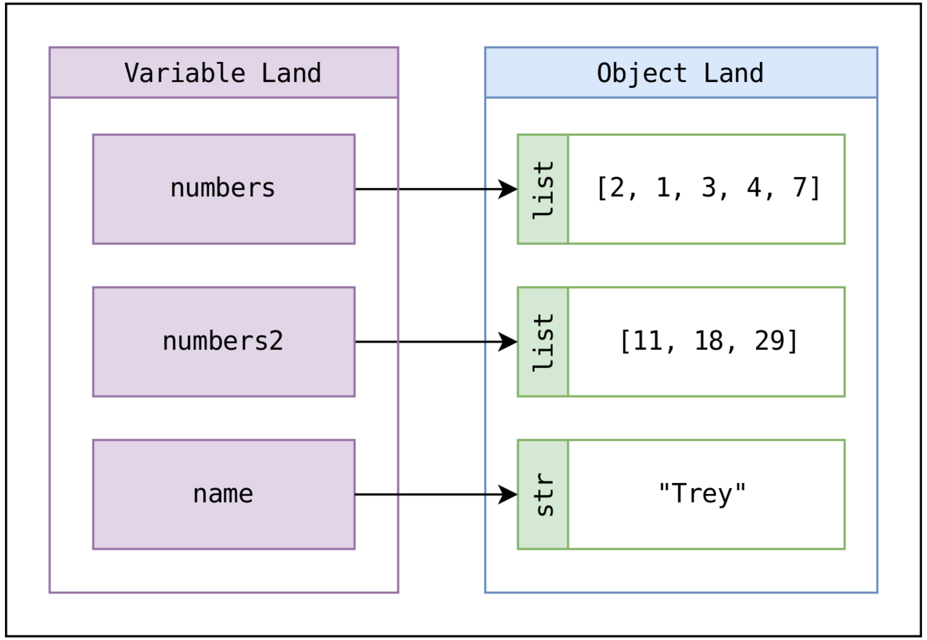
An overview of the most common data types in Python. Note that I excluded the `None` type as well as `modules` and `callables` from this flowchart (for the sake of conciseness).

Image Credits: Created by the author (Fadel M. Megahed) for our ISA 419 course, based on the information provided in the Python's Data model documentation.

Python Variables are Pointers, Not Buckets

- Variables in Python are not buckets containing things; they're **pointers** (they point to objects).
- A **pointer** just represents the **connection between a variable and an objects**.
- Imagine **variables** living in *variable land* and **objects** living in *object land*. A pointer is a little arrow that connects each variable to the object it **points to**.
- Assignment statements point a variable to an object.

```python
numbers = [2, 1, 3, 4, 7]
numbers2 = [11, 18, 29]
name = "Trey"
```

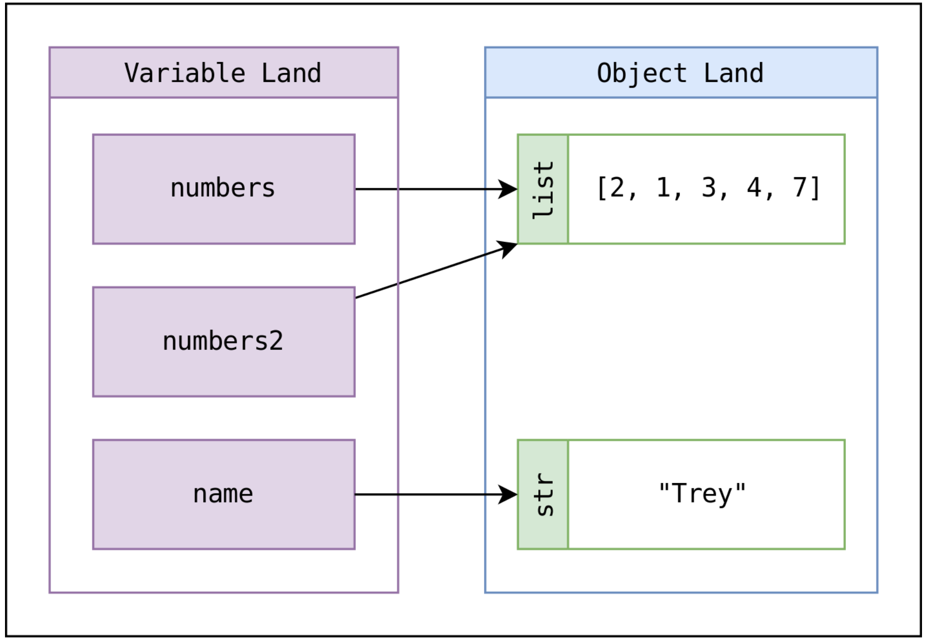


Source: The content and the figure in this slide are from Trey Hunter (2022). Variables and objects in Python

Assignment Points a Variable to an Object

- Variables in Python are not buckets containing things; they're **pointers** (they point to objects).
- A **pointer** just represents the **connection between a variable and an objects**.
- Imagine **variables** living in *variable land* and **objects** living in *object land*. A pointer is a little arrow that connects each variable to the object it **points to**.
- Assignment statements point a variable to an object.

```
numbers = [2, 1, 3, 4, 7]
numbers2 = numbers
name = "Trey"
```



Source: The content and the figure in this slide are from Trey Hunter (2022). Variables and objects in Python

Implications: Equality Compares Objects

Python's `==` operator checks that two objects **represent the same data** (a.k.a. **equality**):

The variables `my_numbers` and `your_numbers` point to **objects representing the same data**, but the objects they **point to are not the same**.

```
my_numbers = [2, 1, 3, 4, 7]
your_numbers = [2, 1, 3, 4, 7]

my_numbers == your_numbers
```

```
## True
```

```
my_numbers is your_numbers
```

```
## False
```

So changing one variable/object **does not** change the other:

```
my_numbers[0] = 7

my_numbers == your_numbers
```

```
## False
```

Changing the Object will Change Both Variables

If two variables **point to the same object:**

```
my_numbers_again = my_numbers
my_numbers is my_numbers_again
```

```
## True
```

Changing the object one variable points to **also changes** the object the other variable points to:

```
my_numbers_again.append(419)

print('my_numbers_again =', my_numbers_again, 'and my_numbers also =', my_numbers, '.')
```

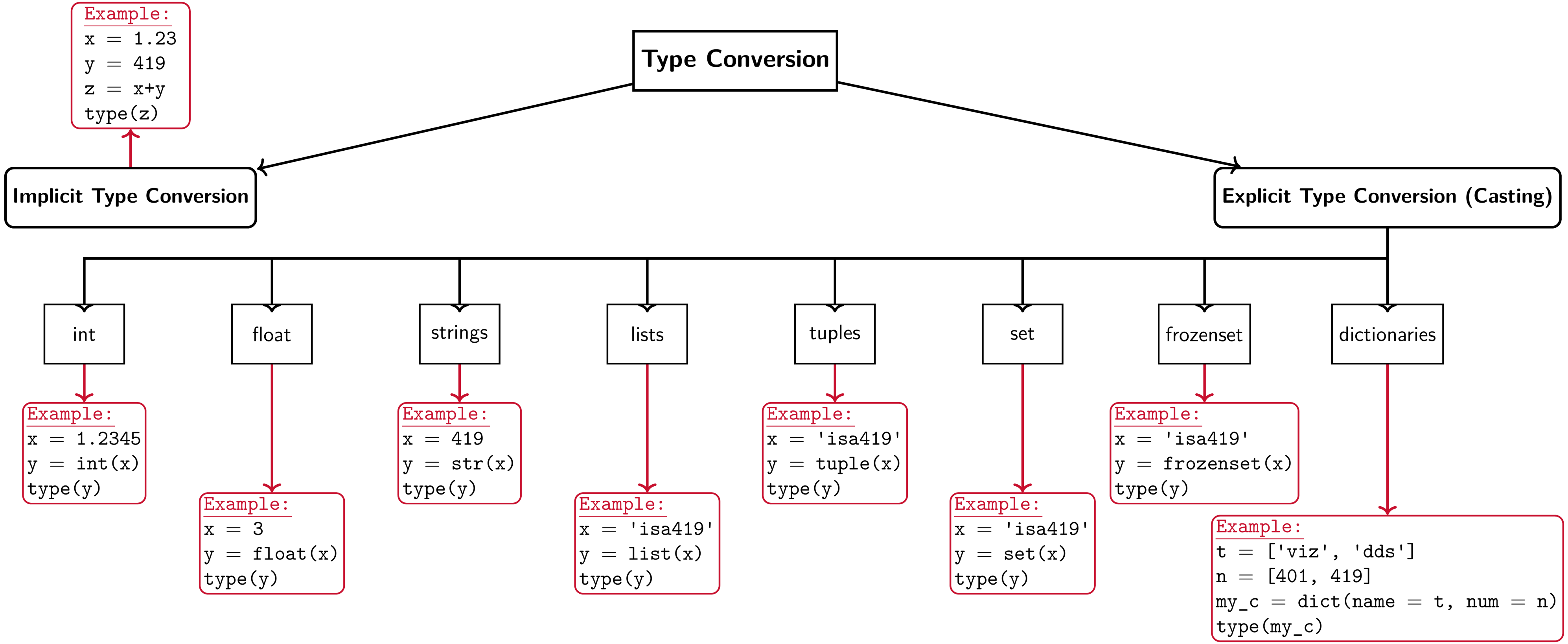
```
## my_numbers_again = [7, 1, 3, 4, 7, 419] and my_numbers also = [7, 1, 3, 4, 7, 419] .
```

On a positive note, this does not apply to strings and numbers (both are **immutable**).

Source: The content in this slide is adapted from Trey Hunter (2022). Variables and objects in Python

Convert Basic Data Types

Type Conversions in Python



Example:
```
x = 1.23
y = 419
z = x+y
type(z)
```

Example:
```
x = 1.2345
y = int(x)
type(y)
```

Example:
```
x = 3
y = float(x)
type(y)
```

Example:
```
x = 419
y = str(x)
type(y)
```

Example:
```
x = 'isa419'
y = list(x)
type(y)
```

Example:
```
x = 'isa419'
y = tuple(x)
type(y)
```

Example:
```
x = 'isa419'
y = set(x)
type(y)
```

Example:
```
x = 'isa419'
y = frozenset(x)
type(y)
```

Example:
```
t = ['viz', 'dds']
n = [401, 419]
my_c = dict(name = t, num = n)
type(my_c)
```

An overview of type conversions in Python.

Image Credits: Created by the author (Fadel M. Megahed) for our ISA 419 course.

Class Activity: Type Conversion

03:00

Task

Observations

- Based on the previous chart, use Google Colab (or your preferred Python environment) to run the following two code examples.
- Write down your observations in the next tab.

```
course_a = 'isa419'
course_list_a = list(course_a)
print(course_list_a)
print(type(course_list_a))
```

```
course_b = 419
course_list_b = list(course_b)
print(course_list_b)
print(type(course_list_b))
```

Class Activity: Type Conversion

03:00

Task Observations

- **My observations from these simple examples are:**
  - Edit me.
  - Edit me.
  - Edit me.

Manipulate Lists and Use Methods on Lists

Lists in Python

- A **list** is a **mutable** sequence of elements, i.e., it can be changed after it is created.
- Lists are created by placing elements inside square brackets `[]`, separated by commas.
- Lists can contain elements of different types, including other lists.
- Lists are **ordered** and **indexed** (i.e., you can access elements by their position).

Lists: Indexing and Slicing

- **Indexing:** Accessing individual elements in a list.
  - Starts from 0 (for the first element) and goes up to `n-1` (for the last element).
  - To reference the last element, you can use `-1`, and so on.
  - You can call a particular item out of a list by using its index. For example, `my_list[0]` will return the first item in the list.

- **Slicing:** Accessing a subset of elements in a list, using the `:` operator, where: `my_list[start:stop:step]`.
  - If `start` not provided, it defaults to 0
  - If `end` is not provided, it defaults to the length of the list.
  - If `step` is not provided, it defaults to 1.
    - `my_list[:3]` returns the first three elements.
    - `my_list[3:]` returns all elements from the fourth to the last. *`my_list[::2]` returns every other element, and
    - `my_list[::-1]` returns the list in reverse order.

Python List Functions

List Function	Description	Example

`len()`	Returns the length of the list.	`my_list = [1,2,3]`
		`length = len(my_list)`
		`# length is 3`

`max()`	Returns the maximum value in the list.	`my_list = [1,2,3]`
		`maximum = max(my_list)`
		`# maximum is 3`

`min()`	Returns the minimum value in the list.	`my_list = [1,2,3]`
		`minimum = min(my_list)`
		`# minimum is 1`

`sum()`	Returns the sum of all elements in the list.	`my_list = [1,2,3]`
		`total = sum(my_list)`
		`# total is 6`

`sort()`	Sorts the list in ascending order.	`my_list = [3,2,1]`
		`my_list.sort()`
		`# my_list is now [1, 2, 3]`

Python List Functions (Cont.)

List Function	Description	Example

`index(item)`	Returns the index of the first occurrence of `item`.	`my_list = [1,2,3]`
		`my_list.index(2)`
		`# index is 1`

`append(item)`	Adds `item` to the end of the list.	`my_list = [1,2,3]`
		`my_list.append(4)`
		`# my_list is now [1, 2, 3, 4]`

`pop(index)`	Removes and returns the element at `index`.	`my_list = [1,2,3]`
		`popped = my_list.pop(1)`
		`# popped is 2,`
		`# my_list is [1, 3]`

`remove(item)`	Removes the first occurrence of `item` from the list.	`my_list = [1,2,3]`
		`my_list.remove(2)`
		`# my_list is [1, 3, 2]`

Class Activity: List Manipulation

10:00

Task Solution

In pairs, build on the code below to:

```
# your path will be different
with open("../../data/simulated_logs.txt", "r") as file:
    lines = file.readlines()

# Strip newline characters from each line
log_entries = [line.strip() for line in lines]
type(log_entries)
```

- Show how to get the **total number of log entries**.
- Show how to slice the **first 5 log entries**.
- **Filter out all** log entries that contain the word **ERROR**.
- **Count** the number of **ERROR** entries.

Class Activity: List Manipulation

10:00

Task Solution

In Google Colab (or your favorite Python environment), write code to address the questions in the previous tab.

Recap

Summary of Main Points

By now, you should be able to do the following:

- Use pseudocode to map out a problem.
- Python Syntax, Data Types, and Data Structures
- Convert data types using type casting
- Manipulate lists and use methods on lists

📝 Review and Clarification 📝

- **Class Notes**: Take some time to revisit your class notes for key insights and concepts.
- **Zoom Recording**: The recording of today's class will be made available on Canvas approximately 3-4 hours after the end of class.
- **Questions**: Please don't hesitate to ask for clarification on any topics discussed in class. It's crucial not to let questions accumulate.