Intermediate Python
(Part 2)

Workshop Lead: Benjamin Z. Rudski

Date: July 18, 2024

McGill initiative in Computational Medicine

Mission statement: deliver quality workshops designed to help biomedical researchers develop the skills they need to succeed.

Location: 740 Dr. Penfield Avenue, Montreal, Quebec

Scan the QR code to sign up for our **mailing list**

Contact: workshop-micm@mcgill.ca

Summer 2024 Workshop Series

Workshop	Date	Lead/Facilitator	Location	Registration
How to think in Code	July 3 10AM-1PM	Thomas Zheng	Education Room 133	Closed
Intro to UNIX and HPC	July 11 9AM-1pm	Georgi Mehri	Education Room 133	Closed
Intro to Git & GitHub	July 12 1PM-5PM	Adrien Osakwe	Education Room 133	Closed
Intro to Python (Part 1)	July 16 9AM-1PM	Benjamin Rudski	Education Room 133	Open
Intermediate Python (Part 2)	July 18 9AM-1PM	Benjamin Rudski	Education Room 133	Open
Fundamentals of Machine Learning	July 24 9AM-1PM	Tugce Gurbuz	Education Room 133	Open
Intro to Matlab	August 7 9AM-1PM	Meghana Munipalle	Education Room 133	TBA
Intro to R (Part 1)	August 12 9AM-1PM	TBA	Education Room 133	TBA
Intermediate R (Part 2)	August 14 1PM-5PM	Gerardo Martinez	Education Room 133	TBA
Intro to Bayesian Inference in R	August 16 1PM-5PM	Adrien Osakwe	Education Room 133	TBA
Proteogenomics	August 19 1PM-5PM	Thomas Zheng	Education Room 133	TBA

https://www.mcgill.ca/micm/training/workshops-series

About me

- BSc from McGill in Hon. CS/Bio, Minor Math
- Third-year PhD candidate in Quantitative Life Sciences (QLS)
- Research on trabecular bone structure in the Reznikov Lab, McGill Bioengineering
  - 3D image processing and analysis
  - Programming is an almost-daily task in my life!

https://github.com/bzrudski

Outline

0. **Module 0 – Welcome (10 minutes)**
    a. Introduction & overview
    b. Motivational problem context: Protein analysis.
1. **Module 1 – Introduction to Functions (45 minutes)**
    a. Function Overview
    b. Writing Custom Functions
    c. Documenting Functions
    d. Exercise
2. **Module 2 – Advanced Python Features (45 minutes)**
    a. One-liners: Some Python Syntactic Sugar
    b. A Brief Overview of Classes
    c. Enumerations
    d. Introduction to Data Classes
    e. Exercise

Outline

3. **Module 3 – Introduction to NumPy Arrays (1 hour)**
    a. Introduction to Arrays
    b. Basic Array Operations
    c. Advanced Array Operations
    d. Exercise
4. **Module 4 – Visualising Data with Matplotlib (50 minutes)**
    a. Creating Plots with Matplotlib
    b. Advanced Plotting Techniques
    c. Exploring the Matplotlib Documentation
    d. Exercise
5. **Module 5 – Wrap-up (10 minutes)**
    a. What to learn next? How?
    b. How to get help and how not to get help
    c. Other cool programming topics

Module 0
Welcome & Problem Context

Review from Tuesday

- Key concepts:
  - Variables
  - Control flow
  - Loops
  - Strings
  - Collections
- So… is that everything there is to know?

We've just scratched the surface…

Why not just stick to simple stuff?

- Real world problems get complicated…
- Good news! Other people have written helpful code!
- We started seeing modules. These are **important**!

Mini-Project: Analysing Proteins

- **Problem scenario:** You analyse protein sequences from different species in both health and disease.
- **How can we easily process these sequences?**
- **How can we analyse the different amino acid properties?**
- **How can we easily visualise the results?**

To the Notebook!

Module 2
Advanced Python Techniques

What is an Object?

Object

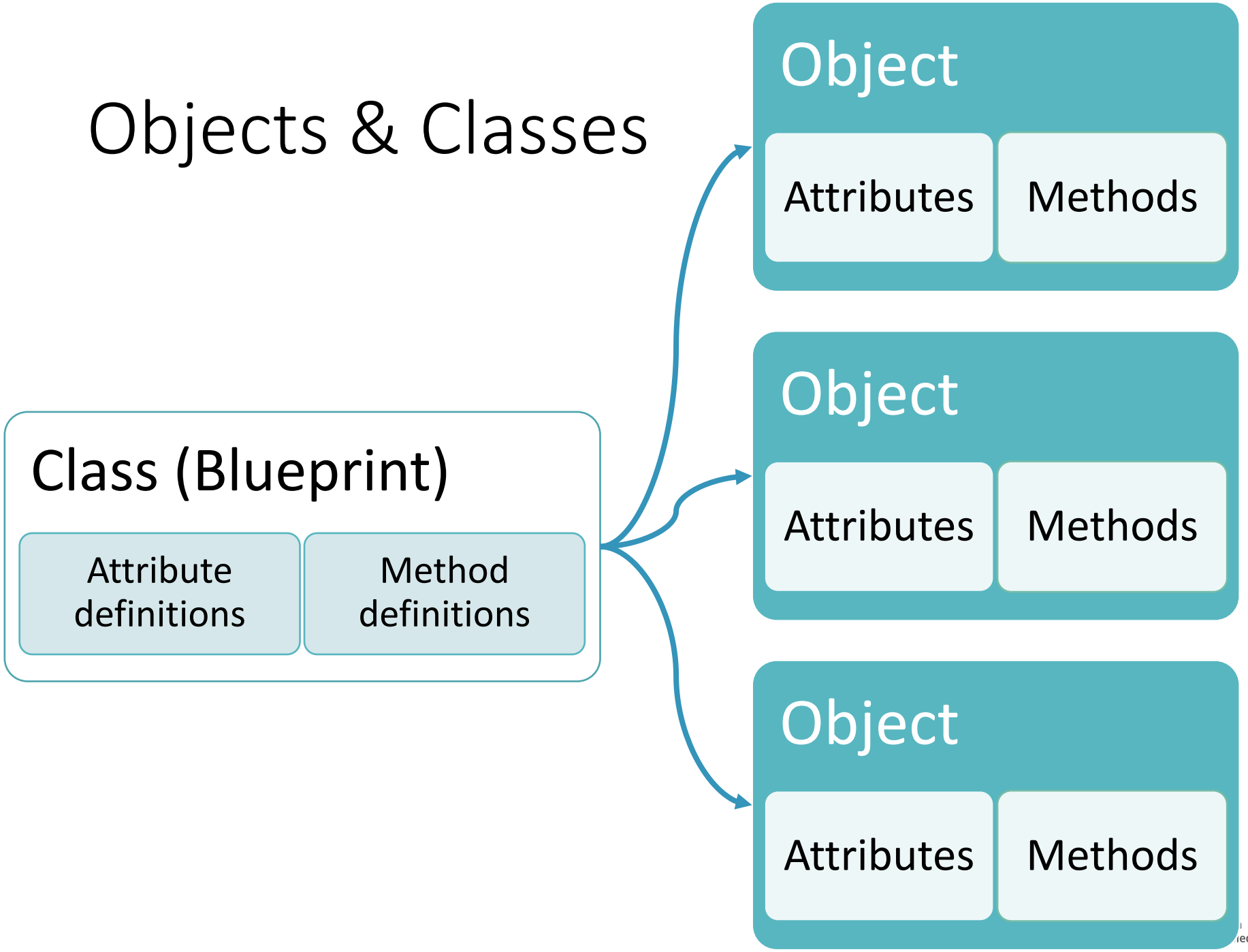
Attributes

- Variables
- Describe the object

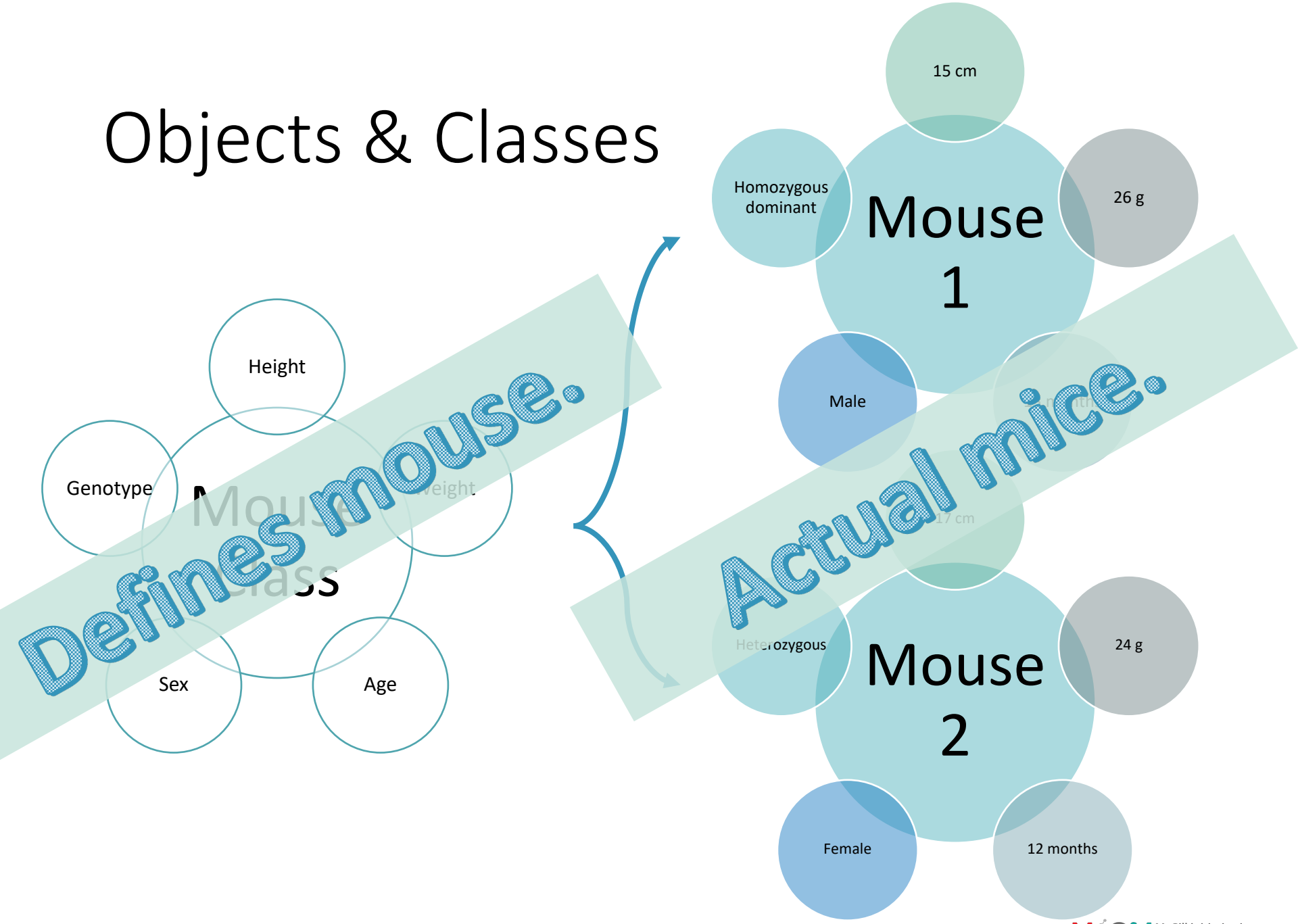
Methods

- Functions
- Compute values
- Alter the object

Objects & Classes



Objects & Classes



Module 3
Introduction to NumPy Arrays

my_array

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)

Shape: (5, 6)

`my_array[0]`?

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)

Shape: (5, 6)

my_array[0]

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)

Shape: (5, 6)

`my_array[:, 0]`?

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)

Shape: (5, 6)

```
my_array[:, 0]
```

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)

Shape: (5, 6)

`my_array[1:3, 2:4]`?

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)

Shape: (5, 6)

`my_array[1:3, 2:4]`

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)

Shape: (5, 6)

`my_array[0:5:2]`?

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)

Shape: (5, 6)

`my_array[0:5:2]`

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)

Shape: (5, 6)

Concluding Remarks

To summarize

- ✓ **Functions** allow us to package up behaviour to use over and over.
- ✓ **Data classes** and **enumerations** help us easily represent the real world in code.
- ✓ **NumPy arrays** allow easily storing many numbers and performing operations without having to loop.
- ✓ **Matplotlib** can be used to generate many different types of plots.

Now you are ready to:

- Define your own functions to simplify important tasks.
- Define new classes to help represent new types of data.
- Use NumPy arrays to perform large calculations.
- Use Matplotlib to easily visualise your results.

Acknowledgements

- Thank you to MiCM for giving me this opportunity and for helping me along the way.
- Thank you to the professors from the McGill School of Computer Science for helping me along my programming journey and for inspiring me to share my programming experience with others.
- Thank you to Professor Mathieu Blanchette, whose COMP 204 course helped introduce me to Python (back in Fall 2018).