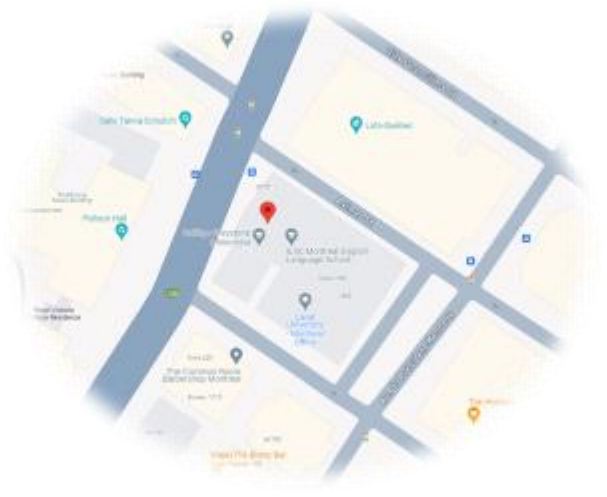
McGill initiative in Computational Medicine

Intermediate Python Part 2

Workshop Lead:
Month Date, Year

McGill | Quantitative Life Sciences | Sciences quantitatives du vivant

QLS-MiCM mission statement: deliver quality workshops designed to help biomedical researchers develop the skills they need to succeed.



Location: 550 Sherbrooke Street, Montreal, Quebec

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

Contact: workshop-micm@mcgill.ca

McGill initiative in Computational Medicine

Workshop Series

Workshop	Date	Location	Registration
How to think in Code	September 18 10AM-12PM	Education Room 133	Closed
Intro to Git & GitHub	September 25 8AM-12PM	Education Room 133	Closed
Intro to Unix	September 27 1PM-3PM	Education Room 133	Closed
Intro to Python (Part 1)	October 28 9AM-11AM	Education Room 133	Open
Intermediate Python (Part 2)	November 1 10AM30-12AM30	McIntyre Room 519	Open
Exploring Matlab	November 4 10AM-12PM	Education Room 133	Open
Intro to R (Part 1)	November 13 1PM-5PM	Education Room 133	Open
Statistics in R (Part 2)	November 18 1PM-5PM	McIntyre Room 519	Open
Data Visualization	November 21 2PM-6PM	Education Room 133	TBA
Intro to scRNA-seq	November 25 10AM-12PM	Education Room 133	TBA
Advanced scRNA-seq	December 2 10AM-12PM	Education Room 133	TBA

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

McGill | Quantitative Life Sciences | Sciences quantitatives du vivant

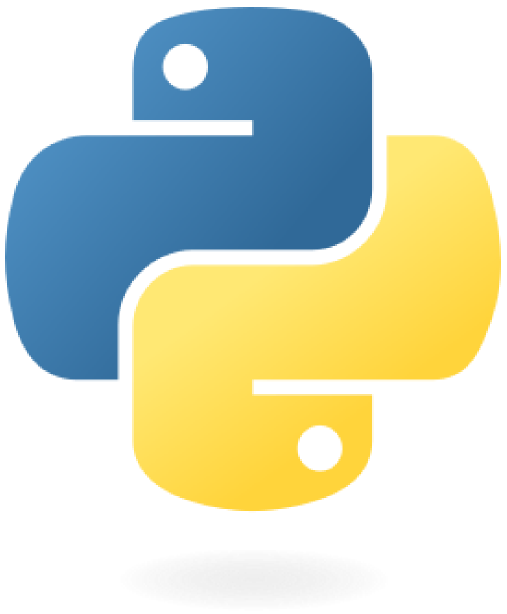
Outline

1. **Module 1 – Getting Up to Speed (10 minutes)**
   a. Quick Review
2. **Module 2 – Introduction to Functions (45 minutes)**
   a. Function Overview
   b. Writing Custom Functions
   c. Documenting Functions
   d. Exercise
3. **Module 3 – Modules and Packages (45 minutes)**
   a. Using Modules
   b. Package and Environment Management
   c. Exercise: Using `textwrap` to nicely print DNA sequences.
4. **Module 4 – Where to go from here (10 minutes)**
   a. What to learn next? How?
   b. How to get help and how not to get help
   c. Glimpse of other cool programming topics

Module 1
Getting Up to Speed

What is Python?

Welcome to the Python Programming Language!

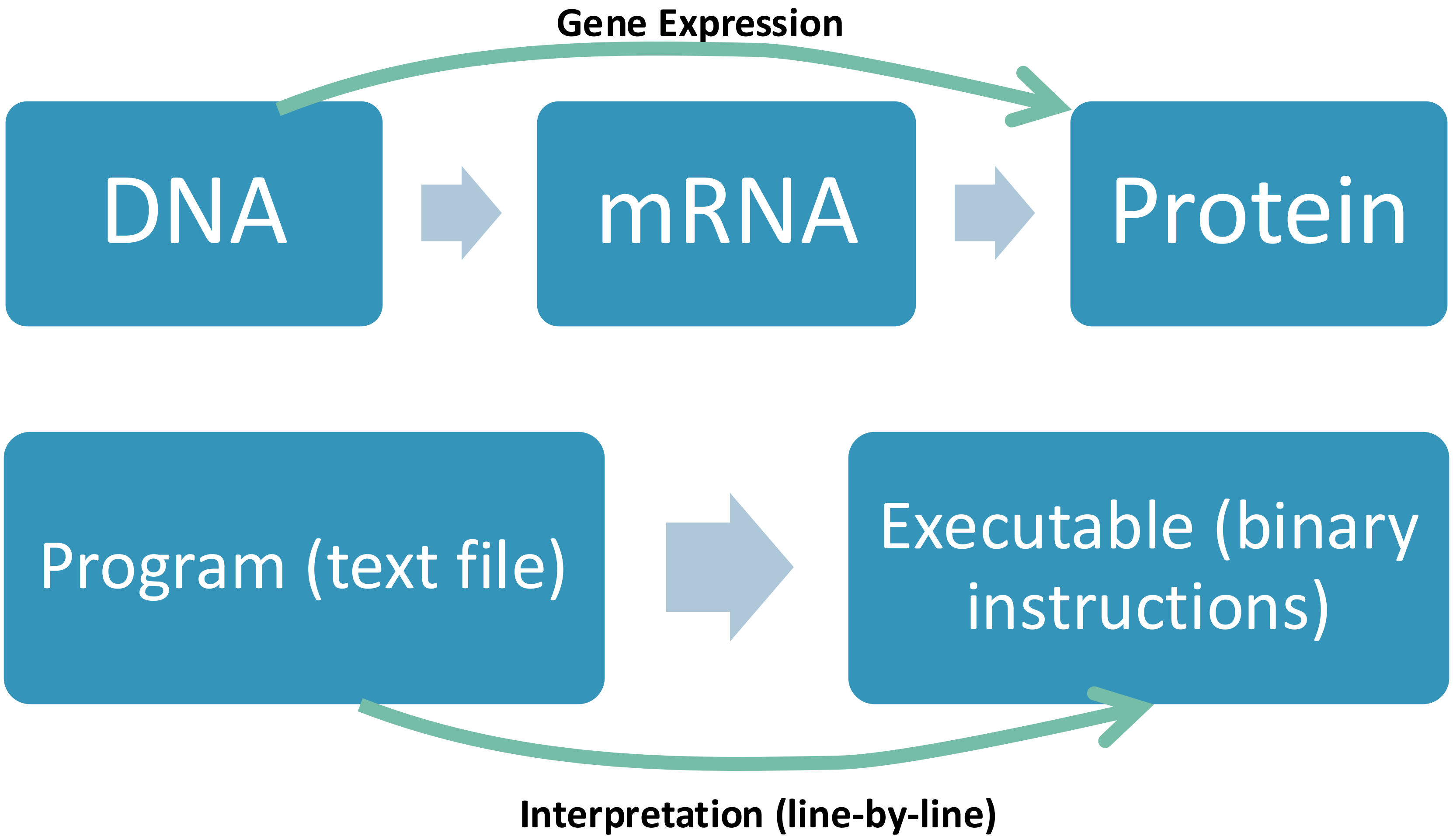


- For more history: https://en.wikipedia.org/wiki/History_of_Python
- Introduced in 1991 by Guido van Rossum
- Features:
  - Free and Open Source
  - Interpreted
  - Object-Oriented
- https://python.org

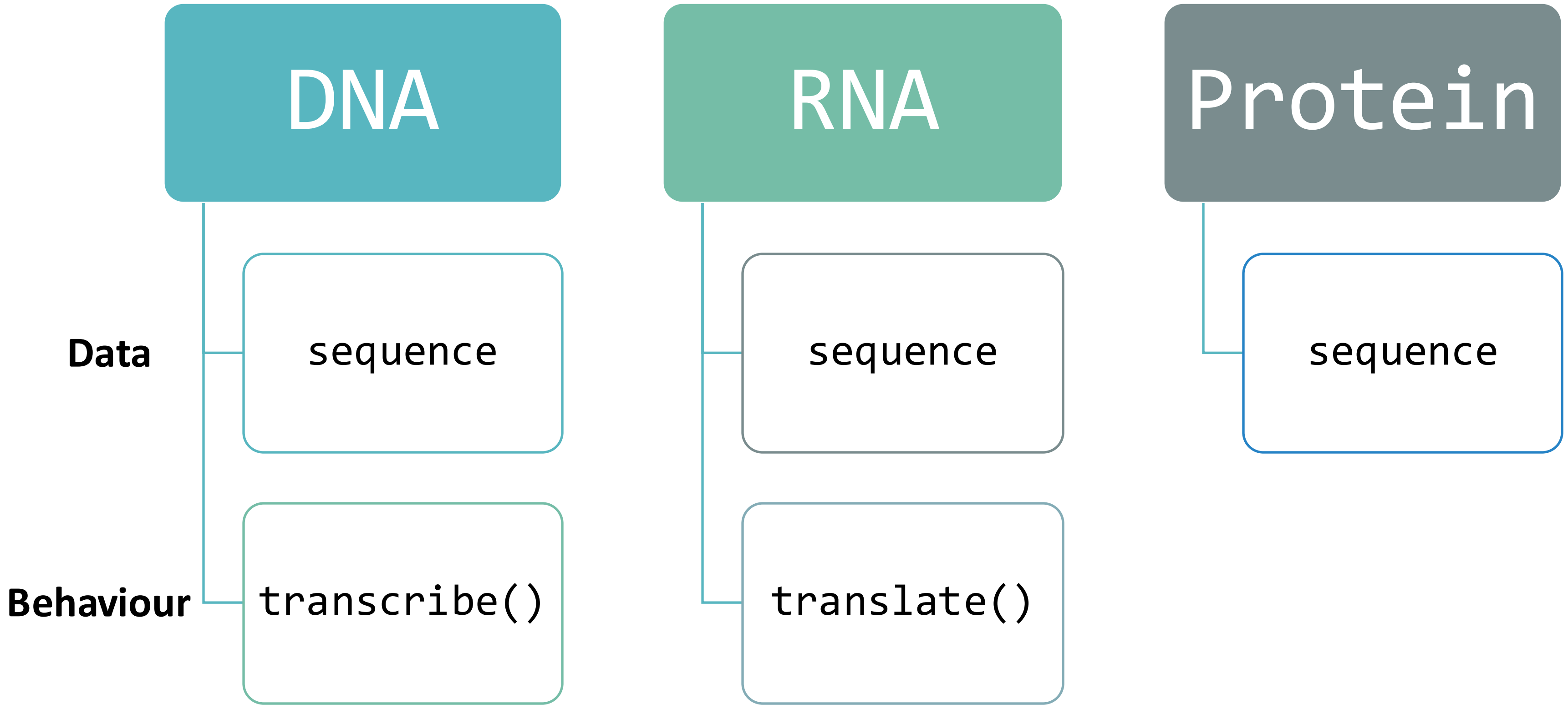
Free and Open Source

- Everyone is free to: download, use, modify and redistribute Python.
- Python is developed, in part, by **the community of users**.
- https://docs.python.org/3/license.html

Interpreted


Gene Expression
DNA
mRNA
Protein
Program (text file)
Executable (binary instructions)
Interpretation (line-by-line)

Object-Oriented



Key Ideas and Syntax

Variables & Data Types

```
# Primitive Data Types
my_str = "Hello!"
my_int = 5
my_float=4.5
my_bool
          (404, "Not Found")
my_list = ["A", "G", "C", "U", "T"]
my_dict = {
    "purines": ["A", "G"],
    "pyrimidines": ["C", "T", "U"],
}
```

Variable name must contain only **letters, numbers and underscores**. Can’t start with a number!

Control Flow

if block is **required**!

elif blocks are optional; no fixed number.

Maximium one else block.

```
if some_boolean:
    run_some_code()
    ...
elif some_other_boolean:
    run_other_code()
    ...
elif yet_another_boolean:
    run_other_other_code()
    ...
else:
    run_code_if_no_match()
    ...


run_code_regardless()
```

Python relies on **indents** to define blocks.

If you stop indenting, the code runs regardless.

Loops

(Almost) never use a raw Boolean.

Block is defined by indents.

```
# while loop
while some_boolean:
     run_some_code()
     ...
     update_boolean()

run_other_code()
```

Update the Boolean during the loop.

```
# for loop
for variable in some_iterable:
     run_some_code()

run_other_code()
```

`variable` gets updated automatically by taking the next value.

Block is defined by indents.

Interactive Workshop!

- Let’s go to a Jupyter Notebook:

To the repository!

Module 3

Modules and Packages

Environments

Project 1	Project 2	Project 3
Python 3.9	Python 3.11	Python 3.8
NumPy 1.26	NumPy 2.0	scikit-image 0.21
SciPy 1.12	scikit-learn 1.2	Matplotlib 3.3
Matplotlib 3.6		PyQt5 5.15.5

To summarize

- ✓ **Functions** define chunks of reusable behaviour that we can easily call again. Functions should be **documented** using **docstrings**.
- ✓ Python uses a system of **modules** to allow easy reuse of code.
- ✓ **Modules** can be **built-in, user-defined** or included in **packages**.
- ✓ There are various tools to **install** packages, including **`pip`** and **conda**.
- ✓ Projects can use **environments** to manage dependencies.

Now you are ready to:

- Define and call new functions.
- Import and use code from built-in Python modules.
- Install new packages to access even more tools.

Acknowledgements

- Thank you to QLS-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).
- Thank you to the Python community!