Welcome to
CHE 384T: Computational Methods in Materials Science

Introduction to Computational Materials Science

Programming Day 1

The University of Texas at Austin
McKetta Department
of Chemical Engineering
Cockrell School of Engineering

Announcements

Python Pre-test

Due 08/30 11:59pm

Bring a computer to Friday Lecture

No class 09/02

Programming Days (approximately every other Friday):

L3	F Aug 30	Installation/set up; Jupyter, Modules and packages, environments What is object oriented programming? Why Python? global v local variables, manipulating lists and arrays, operators, (formatting strings), sets, tuples, lists, dictionaries, dataframes
L5	F Sep 6	conditions, loops, functions, classes and objects opening a github account, testbeds, measuring speed and optimizing code, C libraries, documentation/sphinx, PEP8
L12	F Sep 20	Visualization with python
L21	F Oct 11	ASE calculators
L24	F Oct 18	Python extras: list comprehension, exception handling decorators, lambda functions, regular expressions Peer sharing of Python tricks
6	F Nov 1	DFT tutorial: convergence, scf, relaxation, band structure advanced: phonon calculation, magnetic materials, surface properties

Approximate Schedule and Reading list for CHE384T

L1	Intro to the Course	Ch. 1, Appendix A
L2, L5	Random Walk Diffusion	Ch. 2, Appendix B7, C5, I2-I3
L7, L8	Intro to crystal structure, defect in materials	Appendix B1-B5
L10	Simulating finite systems	Ch. 3
L11, L13 L14	Interatomic potentials	Ch. 5
L16-L22	Molecular dynamics	Ch. 6, Appendix I4 Appendix G
L23, L25	Monte Carlo	Ch. 7, Appendix C4, D1-D4
L25-L32	Electronic structure and DFT	Ch. 4, Appendix F, Supplemental reading
L34	Materials informatics	
L35	Kinetic Monte Carlo	Ch. 9
L37	Monte Carlo as mesoscale Cellular automata	Ch. 11
L38	Quantum computing	

Programming Day Agenda

Why Python?
- General reasons
- Reasons specific to materials science

Setting up your environment
- Using the terminal/command shell
- Install Python 3.x, Anaconda, Jupyter Lab
- Environments
- Picking an IDE or text editor

Python primer
- Basic intro to coding in Python
  - variable types, manipulating lists and arrays, functions
- Modules and packages

Programming Language Paradigms

Imperative Programming: use statements to change program's state;
includes explicit initialization of variables, loops, conditionals
e.g., C, C++, Java, Python

Declarative Programming: instructions are *implied* by output declared
desired computation through composing functions
(as in the case for functional programming);
no or little emphasis on actual implementation
e.g., SQL, HTML

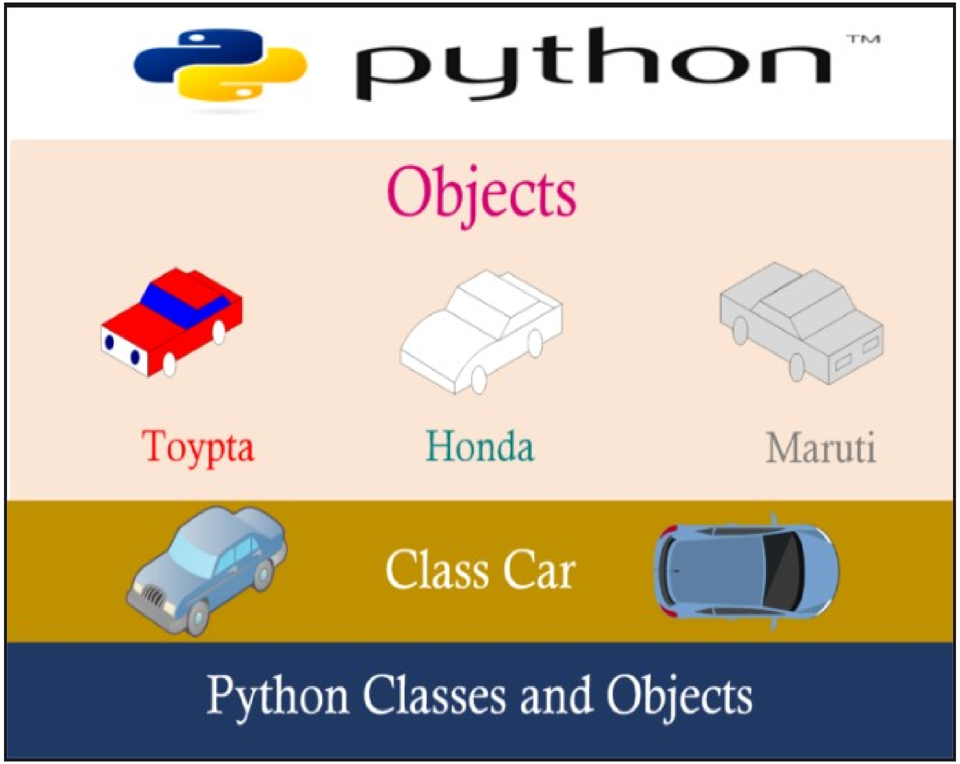
- Programming is self-contained and stateless
- Not suitable for any complex algorithm development

Object-oriented Programming (OOP): data and code stored as objects
e.g., C, C++, Java, Python

- Great for when you have a fixed set of operations on things;
  as your code evolves, you primarily add new classes that implement existing methods
- Not great if you have new operations you need to add

Programming Language Paradigms

Object-oriented Programming (OOP): data and code stored as objects
e.g., C, C++, Java, Python



"Everything is an object in Python"

Why Python?

Readability: Python syntax is one of the closest to human readable language; great for prototyping code

Portability: cross-platform compatibility on any operating system

Extendable: Python code can be rewritten/connected to other languages

Community: the Python community is large and well-documented; there are entire annual conferences and workshops

Modular: lots of packages and libraries for add-on capabilities; Relative ease to create and release your own packages

Open source: Innovation is collaborative; no subscriptions or accounts
A brief history of open source software

Readab

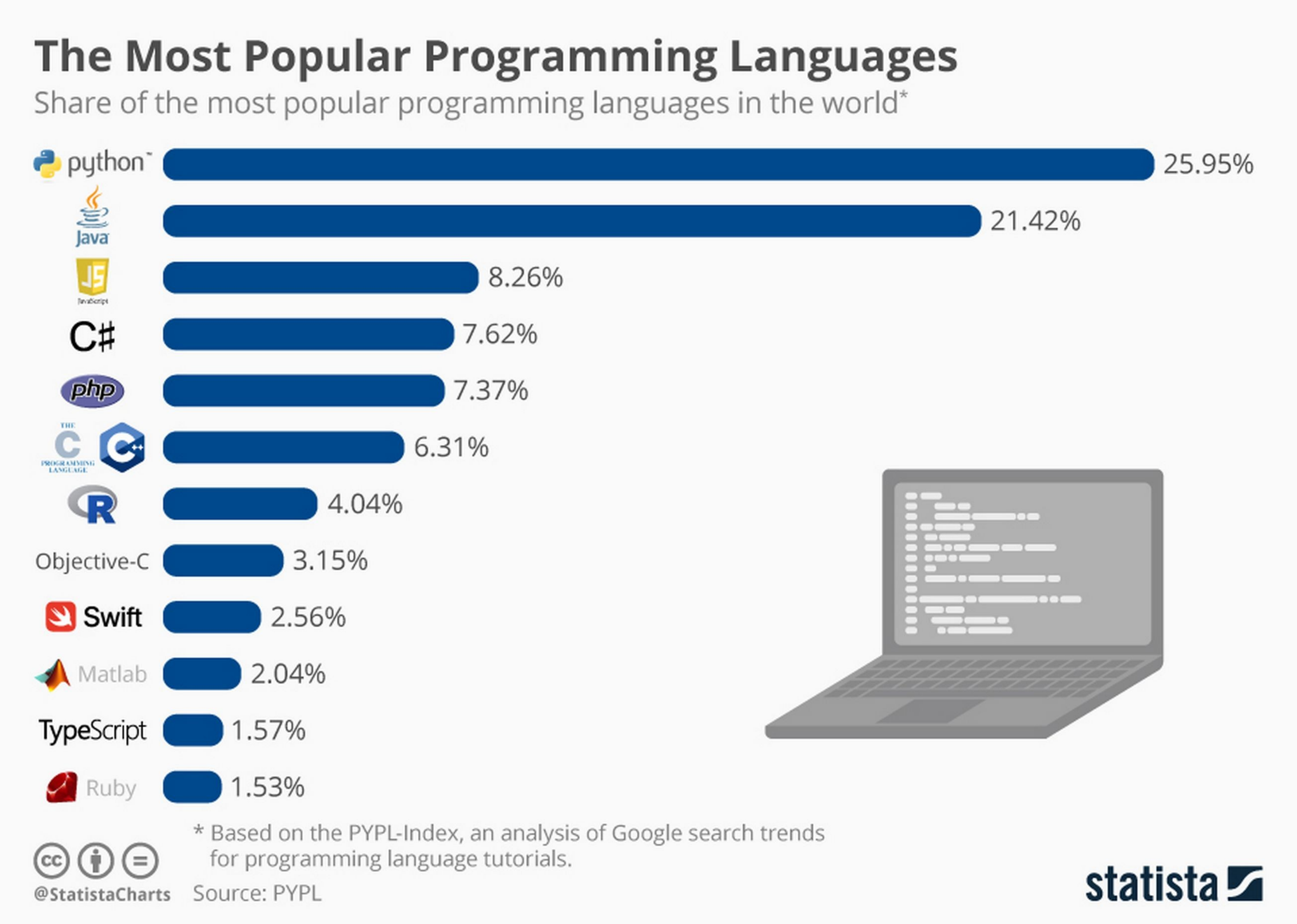
Portabi

Extend

Commu

Modula

Open s



Setting up your environment

Choose: an operating system (Windows, Linux, MacOS)
Use: the terminal/command shell
Install: Python 3.x, Anaconda, Jupyter Lab
Environments in conda: version control your packages
Select: an IDE or text editor

https://www.python.org/downloads/

Anaconda (package management/deployment):
https://docs.anaconda.com/anaconda/install/

Jupyter (web-based interactive development environment):
https://jupyter.org/install

Python Primer(s): manageme your environment

Some modules and packages
(`pip install` or `conda`):

- **numpy**: lean array computing
- **scipy**: extends numpy for array computing
- **matplotlib.pyplot**: basic visualization
- **seaborn**: prettified matplotlib
- **plotly**: interactive plots, great for data science
- **icecream**: debugging beyond `print()`

Related to materials science:

- **ase**: interface with materials science simulations,
  lots of open-source codes
- **pymatgen**: interface with materials science simulations,
  mostly VASP
- **pyscf:** quantum chemistry code

Related to data science:

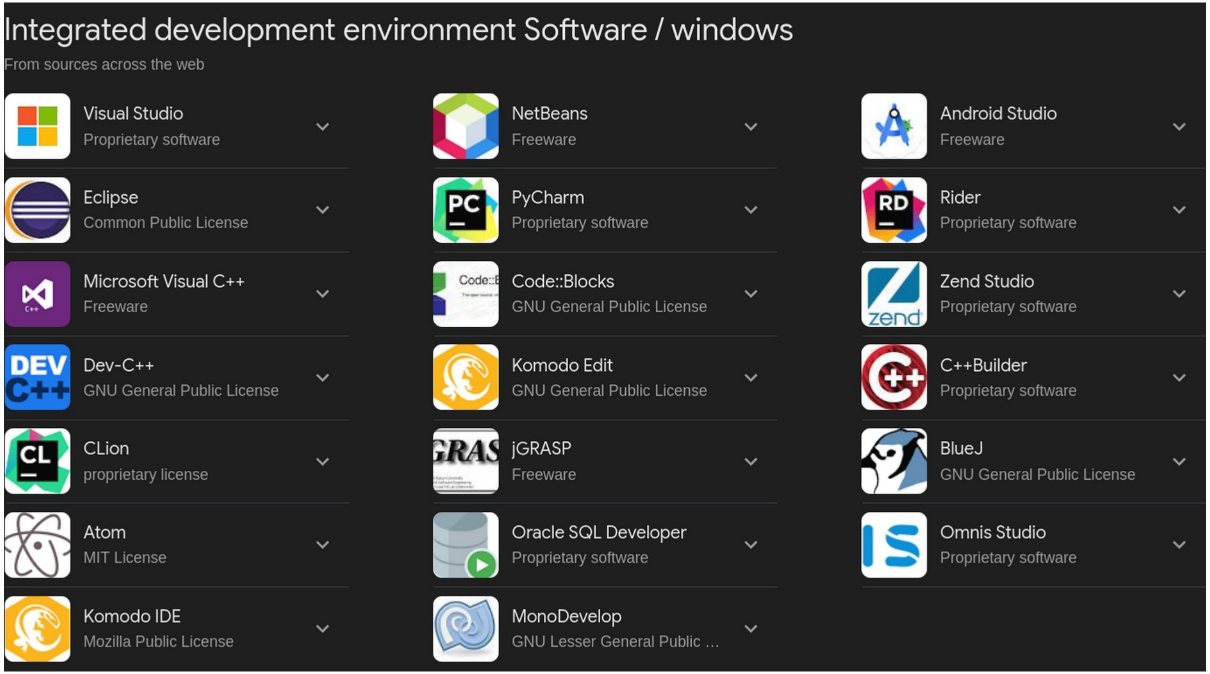
- **Pandas**: dataframes
- **Sklearn**: some basic ML
- **PyTorch**: for deep learning,
  good for prototyping
- **Tensorflow**: for deep learning

`pip`: python package management

`conda`: general system package management

Setting up your environment

Choose: an operating system (Windows, Linux, MacOS)
Use: the terminal/command shell
Install: Python 3.x, Anaconda, Jupyter Lab
Environments: version control your packages
Select: an IDE or text editor



In-terminal text editors:

- **nano**: light-weight
- **vim**: versatile
- **emacs**: highly customizable

Python Primer(s)

Basic coding in Python

Free, no account needed:

- Tutorials from the official documentation: https://docs.python.org/3/tutorial/index.html
- LearnPython.org: https://www.learnpython.org/
- Runestone- Python for Everybody: https://runestone.academy/ns/books/published/py4e-int/index.html
- Automate the Boring Stuff: https://automatetheboringstuff.com/
- FreeCodeCamp.org (4 hr Youtube video): https://www.youtube.com/watch?v=rfscVS0vtbw
- Google for Education: https://developers.google.com/edu/python

Free, account needed:

- Udemy: https://www.udemy.com/course/pythonforbeginnersintro/
- Microsoft: https://learn.microsoft.com/en-us/training/modules/intro-to-python/

Jupyter Lab & Jupyter Notebooks

Tutorial on Jupyter notebooks:
https://utexas.instructure.com/files/79016777/download?download_frd=1

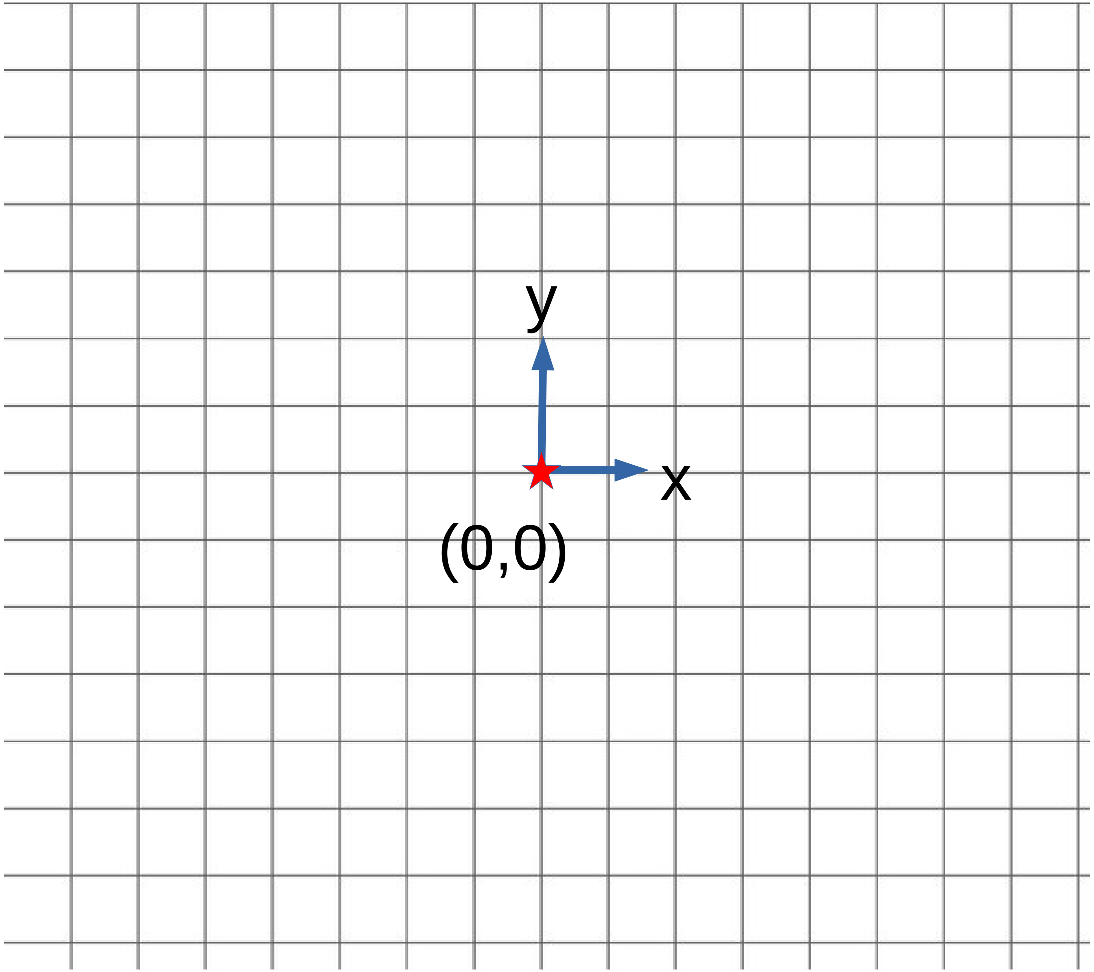
Jupyter notebook: exercises-random-walk-EXERCISES.ipynb

PDF: exercises-random-walk-EXERCISES.pdf

Example solutions report:

https://utexas.instructure.com/files/79270937/download?download_frd=1

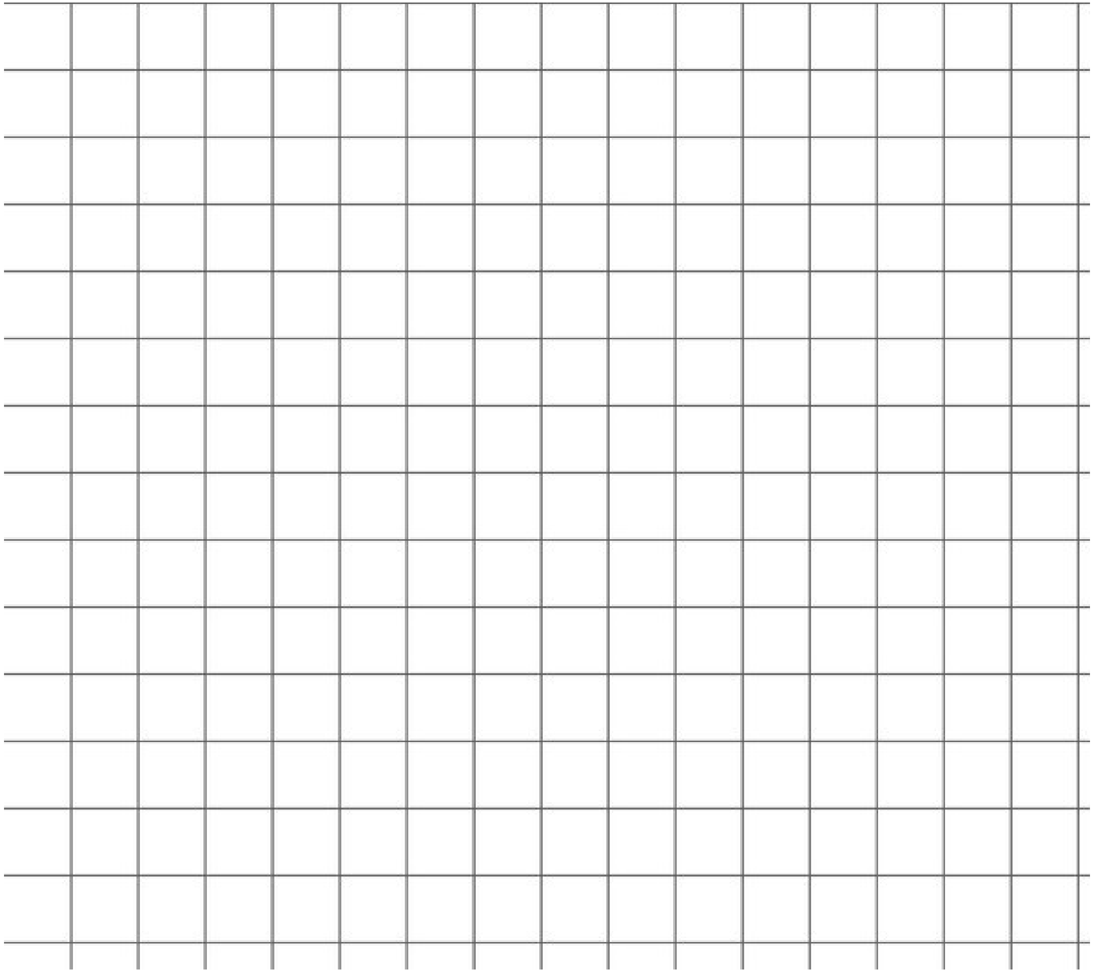
Random walk diffusion: an atomic model for diffusion



Rules for the random walker:

- divide time into *nt* discrete steps spaced by Δ*t* time, where *nt* is an integer and Δ*t* is a number
- can only move 1 space at each time step
- equal and random probability of moving up, down, left, right

Random walk diffusion: an atomic model for diffusion



Random walk diffusion: a small simulation

https://rwd2d-mercury.runmercury.com/

An implementation of the 2D Random Walk Diffusion

Objective	Concept	Python Representation
User chooses *nt* = number of time steps	Variable, integer	`nt`

An implementation of the 2D Random Walk Diffusion

Objective	Concept	Python Representation
User chooses *nt* = number of time steps	Variable, integer	`nt`

Keep track of position of random walker at each time step.
Let's assume it starts at the origin.
	Array (list of items)
	e.g., [3, 4.5, 8, -1]
	
	2D → *x* and *y* coordinate for each position
		Use the library numpy, shorthand is np:
		
		`x = np.zeros(nt+1)`
		`y = np.zeros(nt+1)`

An implementation of the 2D Random Walk Diffusion

Objective	Concept	Python Representation
User chooses *nt* = number of time steps	Variable, integer	`nt`

Keep track of position of random walker at each time step. Let's assume it starts at the origin.	Array (list of items) e.g., [3, 4.5, 8, -1]
	2D → *x* and *y* coordinate for each position
		Use the library numpy, shorthand is np:
		`x = np.zeros(nt+1)`
		`y = np.zeros(nt+1)`

Specify how the position changes at each time step.		`delx = np.array([?,?,?,?])` `dely = np.array([?,?,?,?])`

An implementation of the 2D Random Walk Diffusion

Objective	Concept	Python Representation
User chooses *nt* = number of time steps	Variable, integer	`nt`

Keep track of position of random walker at each time step.
Let's assume it starts at the origin.
	Array (list of items)
	e.g., [3, 4.5, 8, -1]
	
	2D → *x* and *y* coordinate for each position
		Use the library numpy, shorthand is np:
		
		`x = np.zeros(nt+1)`
		`y = np.zeros(nt+1)`

Specify how the position changes at each time step.		`delx = np.array([?,?,?,?])` `dely = np.array([?,?,?,?])`

Save each new position of the diffusion path	Index the array
	i.e., access a specific element
	"zero index"
		`x = [1,2,3]`
		`x[0] = 1`
		`x[1] = 2`

An implementation of the 2D Random Walk Diffusion

Objective	Concept	Python Representation

Repeat for *nt* times	for loop
	range function
		Input:
		`for i in range(3):`
		`    print(i)`
		Output:
		`0`
		`1`
		`2`

Encode the random number to a change in position of the random walker	Generate a (pseudo)-random number	`np.floor(4* np.random.rand(nt))` Generate random number b/t 0 and 1 Random number b/t 0 and 4 Random integer: 0, 1, 2, 3