Warmup (L1)

- Log in to Jupyter: https://jupyter.math.uwaterloo.ca/
- Create a new Notebook for “Python 3 (ipykernel)”.
- Write this into Jupyter and press the run button:

```
!wget https://student.cs.uwaterloo.ca/~cs114/src/Assignment-00.ipynb
```

- Open “Assignment-00.ipynb” in the left panel
- Do what the first box says.

CS114

Module 1: Introduction

Administrata

CS114 M1

Administrata

- The course web site is the central “hub” for course information:
  - https://student.cs.uwaterloo.ca/~cs114/
- The course outline and contact info for course staff is there
  - Instructor: Gregor Richards
  - Coordinator: Scott Freeman King
  - ISAs and IAs: cs114@uwaterloo.ca

Tools and software

- In this course you will be learning to use one popular programming language: Python
- We will mostly be running Python in a web environment called “Jupyter”, by creating so-called “Jupyter Notebooks”.
- Jupyter is a popular environment to use Python on the web, but you will also learn to run it on your own computers.
- All software in this course is free

Resources

- Slides: Available on course web site (*after* each lesson day)
- Jupyter notebook: Available from Math department, https://jupyter.math.uwaterloo.ca/
- Assignments: Linked from course web site
- Textbook: Optional, linked via course web page
- Alternate textbook: Also optional, available via library access online
- Later in the term you will need a computer to run Python locally

Tutorials

- There are *mandatory* tutorial problems in this course
- Tutorials without tutorial problems are hands-on experience
- No tutorials this week, they start next week (Friday the 16th)
- *TUTORIALS WILL BE HELD IN COMPUTER LABS:* MC2062 and MC2063 (use either)
  - This is not the classroom shown in Quest, because Quest doesn't let us assign computer labs

Office hours

- Instructor and ISAs have office hours
- Some are in the Physics Tutorial Center
  - Other students in the PTC can help, but only our ISAs know this course. Make sure you check who you're talking to!
- The schedule will be posted on the course web page
- Also start next week

Web resources

- Piazza: Course discussion should go here. Make sure you're signed up. Also a sort of "always-there" office hours.
- Web site: https://student.cs.uwaterloo.ca/~cs114/ . This is the hub for all online resources (everything else is linked from here).
  - Some parts of the course web site may only appear later this week or next week.

Course components

- Assignments (see outline)
- Tutorial problems (go to Tutorials!)
- Exams
  - You must pass the exam weighted average to pass the course!
- In-lecture quizzes/participation
- Marking scheme described in outline (soon)
- Review academic policies in outline as well

Academic integrity

- Cheating on assignments is a disservice to yourself: you won't be prepared for exams, and you must pass the exams to pass the course
- The elephant in the room: AI
  - AI is stupid and bad at everything. It feels like the opposite when you ask it about something you're not an expert at. It will feel much smarter than you at Python until you learn how mediocre its solutions are.
  - Learning to use AI properly is valuable, but *you* must be the expert. Use of AI is disallowed on most assignments.

When is it cheating?

- You are allowed and encouraged to talk to other students about *concepts*
- You cross the line when you talk about *code* or *solutions*
- When in doubt, refrain or ask a member of course staff
  - Feel free to ask on Piazza in a private post

Assignments

- Programming is 10% writing code, 90% figuring out why your code doesn't work
  - (That's just as true of somebody with decades of experience as somebody with days of experience!)
- Start assignments early! Don't judge time by how long you spend programming. Debugging is most of the time!

Tutorial problems

- Tutorial problems are programming problems done in tutorial
- The ISA will do their best to provide (tutorial) support, but you have to write it yourself, with limited time
- Problems are designed to be solvable in an hour, but the same logic applies: expect to spend most of the hour debugging!

Submissions

- A uwaterloo CS system called Markus.
  - https://markus.student.cs.uwaterloo.ca/
- Assignments and tutorial problems are graded in three stages:
  - Correctness (your code does the right thing)
    - Stage 1: automatic in Markus when you submit
    - Stage 2: by TAs after the deadline
  - Stage 3: Style (your code is understandable)

A subset of Python

- Python is way bigger than we're going to learn in this course
- You are not required to stick to the subset we'll see in this course
- Unfortunately, code that uses advanced features looks like AI ☹
- So, if you do, tell us (in the code) where you learned the features! Cite your sources!

Computation

CS114 M1

What is computation?

- I’m from the last generation that was told this by our teachers:

  “You won’t always have a calculator with you!”

- This advice means well but totally misses the point: knowing how to multiply in your head doesn’t help you know when multiplication is needed
- *Applying* math is a creative endeavor

Calculation

- Calculation is the mechanics of math
  - (Addition, subtraction, multiplication, division, etc.)
- Example: two-body problem

Two-body problem

Given two astronomical bodies' positions and velocities, we can predict their locations and velocities with a simple mathematical expression: just plug in the time

Computation

- Calculation with *repetition* and *decision making*
- Must take a step, then use its results to decide what to do next
- Example: *n*-body problem

n-body problem

With the addition of even a third astronomical body, the problem becomes *chaotic*. Every movement affects the expression. Can no longer predict indefinitely in one calculation.

n-body problem

Instead, we must calculate one small step (a second, a day, a week…), then adjust based on the new positions. Repeat, over and over, until we reach the time we want.

Computation

- Computation is more powerful than calculation
- A computer is to computation as a calculator is to calculation: it is just a dumb computing machine
- Correctly *applying* computation is a creative endeavor

A creative endeavor…

The bad news:

I can’t teach you to creatively apply computation to a problem. No one can.

All I can do is teach you the tools, and give you problems to (hopefully) build your intuition.

Computation and Python

- The only language computers directly understand is electrical impulses
- That's… not very human
- Software exists to translate abstract expressions of computation into those electrical impulses
- *Python* is one such piece of software, and the name for the language to express computation

Why Python

- There are thousands (probably hundreds of thousands) of programming languages
- Why this one?
- Reasons for language choice are *extrinsic*
- Scientists use Python
- It's not that Python is magically good at science, but scientists use Python, and momentum begets momentum

```
def advance(dt, n, bodies, pairs):
    for i in range(n):
        for ([x1, y1, z1], v1, m1,
             [x2, y2, z2], v2, m2) in pairs:
            dx = x1 - x2
            dy = y1 - y2
            dz = z1 - z2
            dist = sqrt(dx * dx + dy * dy + dz * dz)
            mag = dt / (dist*dist*dist)
            b1m = m1 * mag
            b2m = m2 * mag
            v1[0] -= dx * b2m
            v1[1] -= dy * b2m
            v1[2] -= dz * b2m
            v2[2] += dz * b1m
            v2[1] += dy * b1m
            v2[0] += dx * b1m
        for (r, [vx, vy, vz], m) in bodies:
            r[0] += dt * vx
            r[1] += dt * vy
            r[2] += dt * vz
```

```
def advance(dt, n, bodies, pairs):
    for i in range(n):
        for ([x1, y1, z1], v1, m1,
             [x2, y2, z2], v2, m2) in pairs:
            dx = x1 - x2
            dy = y1 - y2
            dz = z1 - z2
            dist = sqrt(dx * dx + dy * dy + dz * dz)
            mag = dt / (dist*dist*dist)
            b1m = m1 * mag
            b2m = m2 * mag
            v1[0] -= dx * b2m
            v1[1] -= dy * b2m
            v1[2] -= dz * b2m
            v2[2] += dz * b1m
            v2[1] += dy * b1m
            v2[0] += dx * b1m
        for (r, [vx, vy, vz], m) in bodies:
            r[0] += dt * vx
            r[1] += dt * vy
            r[2] += dt * vz
```

Calculation

Repetition

```
def advance(dt, n, bodies, pairs):
    for i in range(n):
        for ([x1, y1, z1], v1, m1,
             [x2, y2, z2], v2, m2) in pairs:
            dx = x1 - x2
            dy = y1 - y2
            dz = z1 - z2
            dist = sqrt(dx * dx + dy * dy + dz * dz)
            mag = dt / (dist*dist*dist)
            b1m = m1 * mag
            b2m = m2 * mag
            v1[0] -= dx * b2m
            v1[1] -= dy * b2m
            v1[2] -= dz * b2m
            v2[2] += dz * b1m
            v2[1] += dy * b1m
            v2[0] += dx * b1m
        for (r, [vx, vy, vz], m) in bodies:
            r[0] += dt * vx
            r[1] += dt * vy
            r[2] += dt * vz
```

But I thought computers did *x*

- Everything else is just set dressing on computation
- When a computer is displaying this slide, the shape of each letter is computed through a piece of software called a *font rasterizer*
  - (etc., etc., etc.)

Be the emperor of your computer

CS114 M1

Jupyter

- Let’s open a Jupyter notebook and see some Python
- https://jupyter.math.uwaterloo.ca/
  - Other Jupyter servers are available, and you can even run your own, but we’ll be using Math’s here
  - If you ever need an alternate Jupyter server, Google Colab is free

Jupyter as a calculator

- Let's calculate $\frac{4(7+2)^2}{6+5}$
- Note that the result is not an integer, and it's shown rounded (not as an exact fraction)
- Be careful of this rounding; it's real!
- Python has BEDMAS/PEMDAS

Jupyter as a computer

- Python is an *imperative programming language*
- What this means is that you're giving the computer a sequence of *commands* (synonym: imperatives) to run in order
- You're in command. *It only does what you ask.* Imperative comes from the same root as emperor (in Latin, imperator)

Syntax of a program

- Syntax: The grammar of a (programming) language
- Our commands are *statements*
- Mathematical calculations are *expressions*
- Statements contain expressions

Jupyter as a computer

- Calculations are usually incomplete commands
  - For convenience, Jupyter will show the result of the last calculation, but not others
- If you want it to output the result, command it to do so!
  - … by asking it to "`print`", for silly historical reasons

`print(expression)` or
`print(expression, expression…)`

New syntax!

- What's going on with "`print(...)`"?
- `print` is a *procedure*, and this is a *procedure call*
  - (Actually, we far more often use the word "function")
- A procedure is a way of boxing up and parameterizing computational steps

Procedures vs. functions

Brief pedantry aside:

Technically, *functions* are mathematically pure (they aren't composed of imperative steps) and *procedures* are not. In practice, programmers usually don't make this distinction, so we'll call `print` a function.

Strings

- You may want to print text, not just numbers
- In Python, a piece of text is called a *string*
- Just surround it in quotes:

```
print("Hello, world!")
```

(Single quotes also work, and are more common than double quotes. I use double quotes because I'm old.)

The power of change

- Make your steps do something by storing values in *variables*
- Store to a variable with =

```
x = 42
```

- Retrieve the value by using the name

```
x / 12
```

The power of change

- Variables are *variable*. You can not only set them, but *change* them.

```
x = 7
print("x is", x)
x = x * 6 # Huh???
print("x is", x)
```

- Mathematicians hate it!
  (This isn't what "variable" or "=" mean in math)

One step at a time

- Remember, we're running commands *in order*
- If you change a variable after it's been used, that use is in the past

```
x = 7
y = x * 2
x = x * 3
print("x is", x)
print("y is", y)
```

Variable names

- Variable name must
  - start with a letter or underscore, and
  - contain only letters, underscores, and numerical digits. In particular, no spaces.
- Nothing stops you from overwriting `print` or other functions (other than common sense), so try not to step on your own toes!

Variable names

- Multiple words in a variable name: `snake_case` and `camelCase` are both common.
- Style conventions:
  - Use either `snake_case` or `camelCase`, but be consistent
  - Use descriptive names (not `x`) wherever possible

Jupyter is weird

- Enter this into a cell and run it:

```
z = 42
```

- Then clear the cell, and replace it with this:

```
z / 7
```

- You'll notice that it shows a result. It remembered `z`.

Jupyter is weird

- This “hidden state” is there so that cells can work together (set `z` in one cell, use it in another)
- But, can allow cells to work together through time (set `z` in one cell, delete that assignment, then use it anyway)
- To make sure you’re not relying on this, occasionally use the “run all” button
  - This resets all the variables (called “restarting the kernel”)

Comments

- In an earlier example, I snuck in the question “Huh?”:
  - `x = x * 6 # Huh???`
- This is called a *comment*. You can make your code easier to read by describing what’s happening using comments.
- Starts with a **#**, goes to the end of the line
- Can be a line of its own

More power

- Right now we have a five-function calculator
- The functions in a scientific calculator are available in a *module* called `math`
- *Modules* are bundles of functions, values, variables, or anything else, grouped together for organization

Pythogoras

- Let’s calculate the length of the hypotenuse of a right triangle given the length of its two sides

Pythogoras

- Let's calculate the length of the hypotenuse of a right triangle given the length of its two sides

```
import math
math.sqrt(a**2 + b**2)
```

Smarter imports

- You can also import a specific name

```
from math import sqrt, log
sqrt(a**2 + b**2)
```

- There's no consistent style for whether to import specific names or whole modules. Use whatever is readable. You *do not* have to be consistent if importing multiple modules.

Too many things!

- The `math` module provides a *lot* of functions… and there are a *lot* of modules!
- Find what's in it with the *tab* key
- The `help` function gives you help on anything you've imported

```
help(sqrt)
```

- You can also use the `dir` function (instead of tab) to find what's in a module

```
dir(math)
```